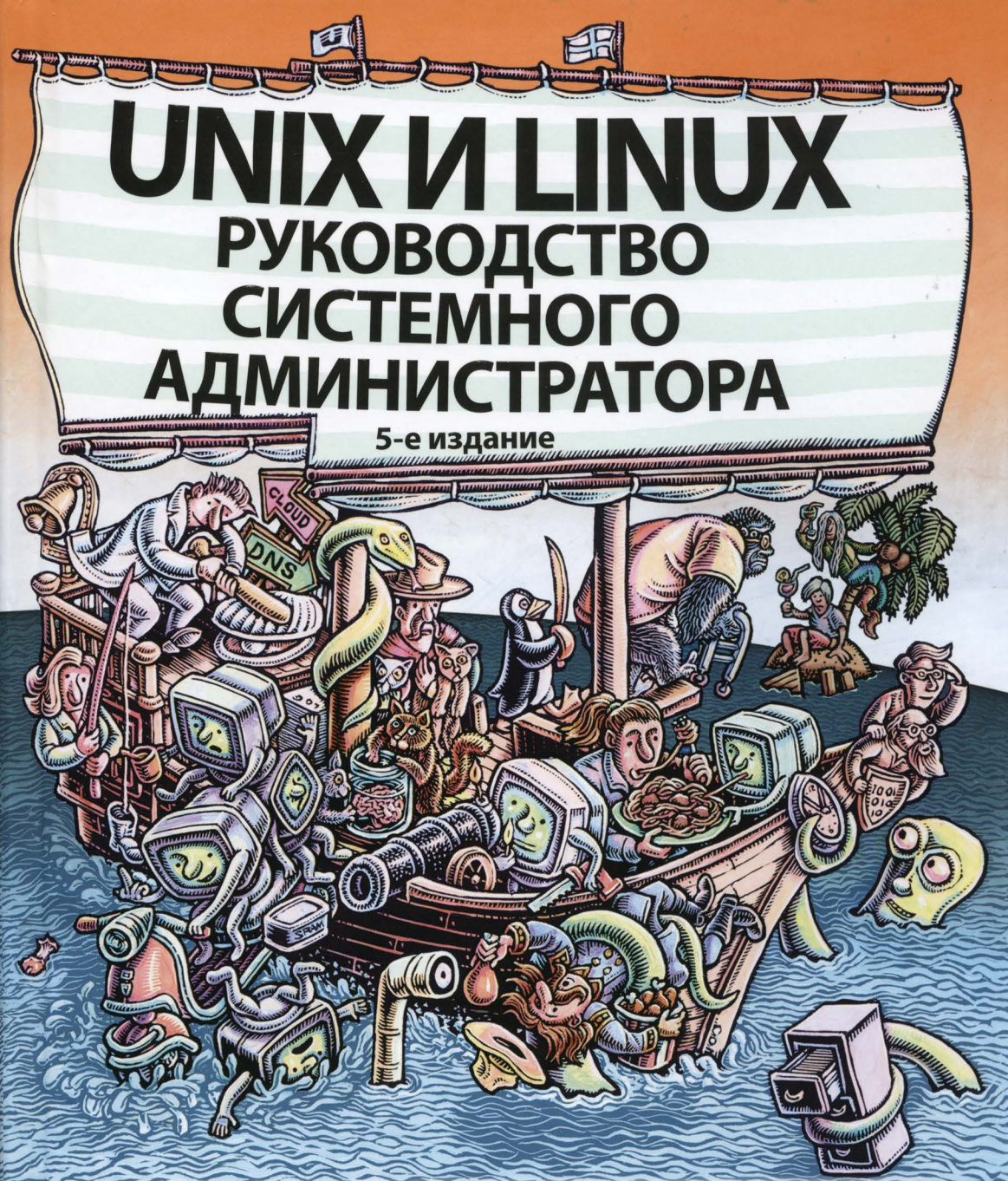
UNIX И LINUX
РУКОВОДСТВО СИСТЕМНОГО АДМИНИСТРАТОРА

5-е издание



ЭВИ НЕМЕТ, ГАРТ СНАЙДЕР, ТРЕНТ ХЕЙН, БЭН УЭЙЛИ, ДЭН МАКИН

при участии Джеймса Гарнетта, Фабрицио Бранка и Адриана Муата

UNIX И LINUX
РУКОВОДСТВО СИСТЕМНОГО АДМИНИСТРАТОРА

5-е издание

UNIX® AND LINUX® SYSTEM ADMINISTRATION HANDBOOK

FIFTH EDITION

Evi Nemeth
Garth Snyder
Trent R. Hein
Ben Whaley
Dan Mackin

with James Garnett, Fabrizio Branca, and Adrian Mouat



Boston • Columbus • Indianapolis • New York • San Francisco • Amsterdam • Cape Town
Dubai • London • Madrid • Milan • Munich • Paris • Montreal • Toronto • Delhi • Mexico City
São Paulo • Sydney • Hong Kong • Seoul • Singapore • Taipei • Tokyo

UNIX И LINUX
РУКОВОДСТВО СИСТЕМНОГО АДМИНИСТРАТОРА

5-е издание

Эви Немет,
Гарт Снайдер,
Трент Хейн,
Бэн Уэйли,
Дэн Макин

при участии Джеймса Гарнетта, Фабрицио Бранка и Адриана Муата

Москва • Санкт-Петербург
2020

ББК 32.973.26-018.2.75
Н50
УДК 681.3.07

ООО "Диалектика"

Зав. редакцией *С.Н. Тригуб*

Перевод с английского и редакция докт. физ.-мат. наук *Д.А. Клюшина*

По общим вопросам обращайтесь в издательство "Диалектика" по адресу:
info@dialektika.com, http://www.dialektika.com

Немет, Эви, **Снайдер**, Гарт, **Хейн**, Трент, **Уэйли**, Бен, **Макин**, Дэн.

Н50 Unix и Linux: руководство системного администратора, 5-е изд. : Пер. с англ. — СПб. : ООО "Диалектика", 2020. — 1168 с. : ил. — Парал. тит. англ.

ISBN 978-5-907144-10-1 (рус.)

ББК 32.973.26-018.2.75

Все названия программных продуктов являются зарегистрированными торговыми марками соответствующих фирм.

Никакая часть настоящего издания ни в каких целях не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами, будь то электронные или механические, включая фотокопирование и запись на магнитный носитель, если на это нет письменного разрешения издательства Addison-Wesley Publishing Company, Inc.

Copyright © 2020 by Dialektika Computer Publishing Ltd.

Authorized Russian translation of the English edition of *UNIX and Linux System Administration Handbook*, 5th Edition (ISBN 978-0-13-427755-4) © 2018 Pearson Education Inc.

This translation is published and sold by permission of Pearson Education Inc., which owns or controls all rights to publish and sell the same.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the Publisher.

Научно-популярное издание

Эви Немет, Гарт Снайдер, Трент Хейн, Бен Уэйли, Дэн Макин

Unix и Linux: руководство системного администратора
5-е издание

Подписано в печать 24.03.2020. Формат 70x100/16
Гарнитура Times
Усл. печ. л. 94,17. Уч.-изд. л. 81,2
Тираж 500 экз. Заказ № 2141

Отпечатано в АО "Первая Образцовая типография"
Филиал "Чеховский Печатный Двор"
142300, Московская область, г. Чехов, ул. Полиграфистов, д. 1
Сайт: www.chpd.ru, E-mail: sales@chpd.ru, тел. 8 (499) 270-73-59

ООО "Диалектика", 195027, Санкт-Петербург, Магнитогорская ул., д. 30, лит. А, пом. 848

ISBN 978-5-907144-10-1 (рус.)
ISBN 978-0-13-427755-4 (англ.)

© ООО "Диалектика", 2020
© Pearson Education, Inc., 2018

Оглавление

Предисловие 36
Введение 38

Часть I. Основы администрирования 41
Глава 1. С чего начать 43
Глава 2. Загрузка и системные демоны 69
Глава 3. Управление доступом и привилегии суперпользователя 103
Глава 4. Управление процессами 127
Глава 5. Файловая система 155
Глава 6. Инсталляция и управление программным обеспечением 187
Глава 7. Сценарии и командная оболочка 215
Глава 8. Управление учетными записями пользователей 273
Глава 9. Облачные вычисления 299
Глава 10. Журналирование 321
Глава 11. Драйверы и ядро 351
Глава 12. Печать 383

Часть II. Работа в сетях 395
Глава 13. Сети TCP/IP 397
Глава 14. Сетевые аппаратные средства 477
Глава 15. IP-маршрутизация 499
Глава 16. DNS: система доменных имен 517
Глава 17. Система единого входа 595
Глава 18. Электронная почта 613
Глава 19. Веб-хостинг 689

Часть III. Хранение данных 729
Глава 20. Дисковая память 731
Глава 21. Сетевая файловая система NFS 805
Глава 22. Файловая система SMB 833

Часть IV. Эксплуатация 845
Глава 23. Управление конфигурацией 847
Глава 24. Виртуализация 909
Глава 25. Контейнеры 923
Глава 26. Непрерывная интеграция и доставка 955
Глава 27. Безопасность 985
Глава 28. Мониторинг 1043
Глава 29. Анализ производительности 1071
Глава 30. Центры обработки данных 1093
Глава 31. Методология, политика и стратегии 1107
Краткая история системного администрирования 1139
Предметный указатель 1149

Содержание

О соавторах 33

Об авторах 34

Памяти Эви 35

Предисловие 36

Организация книги 36
Авторы 37
Контактная информация 37

Введение 38

Благодарности 39
От издательства 40

Часть I. Основы администрирования 41

Глава 1. С чего начать 43

1.1. Основные обязанности системного администратора 44
- Управление доступом 44
- Добавление оборудования 44
- Автоматизация задач 44
- Управление резервными копиями 44
- Установка и обновление программного обеспечения 45
- Мониторинг 45
- Исправление проблем 45
- Ведение локальной документации 45
- Бдительный мониторинг безопасности 46
- Настройка производительности 46
- Разработка правил 46
- Работа с поставщиками 46
- Тушение пожаров 46

1.2. Предварительный опыт 47
1.3. Дистрибутивы Linux 48
1.4. Примеры систем, используемых в этой книге 49
- Примеры дистрибутивов Linux 50
- Пример дистрибутива UNIX 51

1.5. Обозначения и типографские соглашения 52
1.6. Единицы измерения 53
1.7. `Man`-страницы и другая онлайн-документация 54
- Организация `man`-страниц 54
- Команда `man`: чтение страниц интерактивного руководства 55
- Хранение страниц интерактивного руководства 55

1.8. Другая официальная документация 56
- Руководства по конкретным системам 56
- Документация по конкретным пакетам 56

Книги 57
Документы RFC 57
1.9. Другие источники информации 57
Сохранение актуальности 58
Практические руководства и справочные сайты 58
Конференции 59
1.10. Способы поиска и установки программного обеспечения 60
Как определить, установлено ли программное обеспечение 60
Добавление нового программного обеспечения 61
Создание программного обеспечения из исходного кода 63
Установка с помощью веб-сценария 64
1.11. Где разместить программное обеспечение 65
1.12. Специализация и смежные дисциплины 66
Методология DevOps 66
Инженеры по надежности сайтов 66
Инженеры по безопасности 66
Сетевые администраторы 66
Администраторы баз данных 67
Инженеры центра сетевых операций 67
Технические специалисты центров обработки данных 67
Архитекторы 67
1.13. Литература 67
Системное администрирование и методология DevOps 68
Важные инструменты 68

Глава 2. Загрузка и системные демоны 69

2.1. Обзор процесса загрузки 69
2.2. Системные прошивки 70
BIOS или UEFI 71
Устаревший интерфейс BIOS 72
UEFI 72
2.3. Загрузчики 74
2.4. GRUB: универсальный загрузчик 74
Конфигурация GRUB 74
Командная строка GRUB 76
Параметры ядра Linux 76
2.5. Процесс загрузки FreeBSD 77
Вариант BIOS: `boot0` 77
Вариант UEFI 78
Конфигурация загрузчика 78
Команды загрузчика `loader` 79
2.6. Демоны управления системой 79
Обязанности демона `init` 80
Реализации демона `init` 80
Традиционный стиль `init` 81
Менеджер `systemd` против остального мира 82
Аргументы против `init` 82

2.7. Менеджер systemd в деталях 83
Модули и модульные файлы 83
Команда systemctl: управление менеджером systemd 84
Состояние модуля 85
Цели 87
Зависимости между модулями 88
Порядок выполнения 90
Более сложный пример файла 90
Локальные службы и настройки 91
Предостережения об управлении службами и запуском 92
Журнал systemd 94
2.8. Сценарии инициализации и запуска системы FreeBSD 95
2.9. Процедуры перезагрузки и выключения 96
Выключение физических систем 97
Выключение облачных систем 97
2.10. Что делать, если система не грузится? 97
Однопользовательский режим 98
Однопользовательский режим в системе FreeBSD 99
Однопользовательский режим с загрузчиком GRUB 100
Восстановление облачных систем 100

Глава 3. Управление доступом и привилегии суперпользователя 103

3.1. Стандартное управление доступом в UNIX 104
Контроль доступа к файловой системе 104
Владение процессом 105
Учетная запись суперпользователя root 106
Установка флагов setuid и setgid 106
3.2. Управление учетной записью root 107
Вход в учетную запись root 107
Команда su: замена идентификатора пользователя 108
Программа sudo: ограниченный вариант команды su 108
Отключение учетной записи root 115
Системные учетные записи, отличные от root 116
3.3. Расширения стандартной модели контроля доступа 117
Недостатки стандартной модели 118
PAM: подключаемые модули аутентификации 118
Kerberos: сетевая криптографическая аутентификация 119
Списки управления доступом к файловой системе 119
Возможности Linux 120
Пространства имен Linux 120
3.4. Современный контроль доступа 121
Отдельные экосистемы 121
Обязательный контроль доступа 122
Контроль доступа на основе ролей 123
SELinux: улучшенная безопасность Linux 123
AppArmor 125
3.5. Литература 126

Глава 4. Управление процессами 127
4.1. Компоненты процесса 127
Идентификатор процесса PID 128
Идентификатор родительского процесса PPID 128
Идентификатор пользователя UID и текущий идентификатор пользователя EUID 129
Идентификатор группы (GID) и текущий идентификатор группы (EGID) 129
Фактор уступчивости 130
Управляющий терминал 130
4.2. Жизненный цикл процесса 130
Сигналы 131
Команда `kill`: отправка сигналов 133
Состояния процессов и потоков 134
4.3. Команда `ps`: текущий контроль процессов 135
4.4. Интерактивный мониторинг процессов с помощью команды `top` 137
4.5. Команды `nice` и `renice`: изменение приоритета выполнения 139
4.6. Файловая система `/proc` 140
4.7. Команды `strace` и `truss`: отслеживание сигналов и системных вызовов 141
4.8. Процессы, вышедшие из-под контроля 143
4.9. Периодические процессы 145
Демон `cron`: команды расписания 145
Системные таймеры 150
Общее использование запланированных задач 153

Глава 5. Файловая система 155
5.1. Имена путей 157
5.2. Монтирование и демонтирование файловой системы 157
5.3. Структура файлового дерева 160
5.4. Типы файлов 162
Обычные файлы 164
Каталоги 164
Жесткая ссылка 164
Файлы символьных и блочных устройств 165
Локальные сокеты 166
Именованные каналы 166
Символические ссылки 166
5.5. Атрибуты файлов 167
Биты режима 167
Биты `setuid` и `setgid` 168
Дополнительный бит 169
Команда `ls`: просмотр атрибутов файла 169
Команда `chmod`: изменение прав доступа 170
Команды `chown` и `chgrp`: смена владельца и группы 172
Команда `umask`: задание стандартных прав доступа 173
Дополнительные флаги в системе Linux 173

5.6. Списки управления доступом 175
Предупреждение 175
Типы ACL 176
Реализация списков ACL 176
Поддержка ACL в системе Linux 177
Поддержка ACL в системе FreeBSD 177
Обзор POSIX ACL 178
Списки NFSv4 ACL 181

Глава 6. Инсталляция и управление программным обеспечением 187

6.1. Инсталляция операционных систем 188
Загрузка по сети на персональном компьютере 188
Настройка PXE 189
Использование Kickstart — автоматизированного инсталлятора Red Hat и CentOS 190
Автоматизированная инсталляция систем Debian и Ubuntu 193
6.2. Управление пакетами 196
6.3. Системы управления пакетами для Linux 198
Команда rpm: управление пакетами RPM 198
Команда `dpkg`: управление пакетами `.deb` 199
6.4. Использование высокоуровневых систем управления пакетами в системе Linux 200
Хранилища пакетов 201
APT: усовершенствованное средство управления пакетами 203
Настройка конфигурации хранилища 204
Пример файла `/etc/apt/sources.list` 205
Создание локального зеркала хранилища 206
Автоматизация работы системы APT 206
Система `yum`: управление выпусками для RPM 207
6.5. Управление программным обеспечением в системе FreeBSD 208
Базовая система 209
Менеджер пакетов `pkg` в системе FreeBSD 209
Коллекция портов 210
6.6. Локализация и настройка конфигурации программного обеспечения 211
Организация локализации 212
Структурные изменения 212
Ограничение количества выпусков 213
Тестирование 213
6.7. Литература 214

Глава 7. Сценарии и командная оболочка 215

7.1. Основы сценариев 216
Создание микросценариев 216
Хорошо изучите несколько инструментов 217
Автоматизируйте все, что возможно 217
Избегайте преждевременной оптимизации 218

Выберите правильный язык сценариев 218
Следуйте рекомендациям 220
7.2. Основы работы с оболочками 222
Редактирование команд 223
Каналы и перенаправление потоков 223
Использование переменных и кавычек 225
Переменные окружения 226
Команды фильтрации 227
7.3. Написание сценариев для оболочки sh 230
Выполнение 231
От команд к сценариям 232
Ввод и вывод данных 234
Пробелы в именах файлов 235
Функции и аргументы командной строки 235
Поток управления 237
Циклы 239
Арифметика 241
7.4. Регулярные выражения 241
Процесс сопоставления 242
Литеральные символы 242
Специальные символы 242
Примеры использования регулярных выражений 244
Захваты 245
Жадность, леность и катастрофический поиск с возвратом 246
7.5. Программирование на языке Python 247
Страсти по Python 3 247
Python 2 или Python 3? 248
Краткое введение в язык Python 249
Объекты, строки, числа, списки, словари, кортежи и файлы 250
Пример проверки ввода 252
Циклы 253
7.6. Программирование на языке Ruby 254
Инсталляция 255
Краткое введение в язык Ruby 255
Блоки 256
Символы и хеши опций 258
Регулярные выражения в языке Ruby 259
Язык Ruby как фильтр 260
7.7. Управление библиотекой и средой для Python и Ruby 260
Поиск и установка пакетов 261
Создание воспроизводимых сред 261
Несколько сред 262
7.8. Контроль версий с помощью системы Git 265
Простой пример Git 267
Ловушки Git 269
Коллективное кодирование с помощью системы Git 269

7.9. Литература 271
Оболочки и сценарии оболочки 271
Регулярные выражения 271
Python 272
Ruby 272

Глава 8. Управление учетными записями пользователей 273

8.1. Основы управления учетными записями 274
8.2. Файл /etc/passwd 274
Регистрационное имя 275
Зашифрованные пароли 276
Идентификатор пользователя 278
Идентификатор группы по умолчанию 278
Поле GECOS 279
Домашний каталог 279
Регистрационная оболочка 280
8.3. Файлы /etc/shadow 280
8.4. Файлы /etc/master.passwd и /etc/login.conf в системе FreeBSD 282
Файл /etc/master.passwd 282
Файл /etc/login.conf 283
8.5. Файл /etc/group 284
8.6. Подключение пользователей вручную: основные действия 285
Редактирование файлов passwd и group 286
Задание пароля 287
Создание домашнего каталога пользователя и инсталляция конфигурационных файлов 287
Установка прав доступа и владения 289
Конфигурирование ролей и административных привилегий 289
Заключительные действия 290
8.7. Добавление пользователей с помощью сценариев: useradd, adduser и newusers 290
Команда useradd в системе Linux 291
Команда adduser в системах Debian и Ubuntu 292
Команда adduser в системе FreeBSD 292
Команда newusers в системе Linux: добавление пользователей пакетом 293
8.8. Безопасное удаление учетных записей пользователей и файлов 294
8.9. Блокирование регистрационных имен пользователей 295
8.10. Уменьшение риска с помощью модулей PAM 296
8.11. Централизация управления учетными записями 296
Протокол LDAP и служба Active Directory 296
Системы “единого входа” 297
Системы управления учетными данными 297

Глава 9. Облачные вычисления 299

9.1. Облако в контексте 300
9.2. Выбор облачной платформы 301
Публичные, частные и гибридные облака 302

Amazon Web Services 303
Google Cloud Platform 303
DigitalOcean 304
9.3. Основы работы с облачными службами 304
Доступ к облаку 306
Регионы и зоны доступности 306
Виртуальные частные серверы 308
Сети 308
Хранилище 309
Идентификация и авторизация 310
Автоматизация 310
9.4. Облака: быстрый запуск VPS на платформе 311
Веб-службы Amazon 311
Интерфейс `aws`: управление подсистемами AWS 312
Google Cloud Platform 315
DigitalOcean 317
9.5. Контроль затрат 318
9.6. Литература 320

Глава 10. Журналирование 321

10.1. Местоположение файлов регистрации 323
Специальные журнальные файлы 325
Как просмотреть записи в журнале `systemd` 325
10.2. Журнал `systemd` 326
Настройка журнала `systemd` 327
Добавление дополнительных параметров фильтрации для журнала 328
Совместное использование с системой Syslog 328
10.3. Система Syslog 329
Чтение сообщений системы Syslog 330
Архитектура системы Rsyslog 331
Версии Rsyslog 331
Конфигурация Rsyslog 332
Примеры конфигурационных файлов 340
Отладка системы Syslog 343
10.4. Журнальная регистрация на уровне ядра и на этапе начальной загрузки 344
10.5. Управление журнальными файлами и их ротация 345
Утилита `logrotate`: кросс-платформенное управление журналами 345
Утилита `newsyslog`: управление журналами в системе FreeBSD 346
10.6. Управление журналами в крупном масштабе 347
Стек ELK 347
Graylog 348
Журналирование как услуга 348
10.7. Принципы обработки журнальных файлов 349

Глава 11. Драйверы и ядро 351

11.1. Ядра и системное администрирование 352
11.2. Нумерация версий ядра 353

Версии ядер для системы Linux 353
Версии ядер FreeBSD 353
11.3. Устройства и их драйверы 354
Файлы и номера устройств 354
Проблемы управления файлами устройств 356
Создание файлов устройств 356
Управление современными файловыми системами 356
Управление устройствами в Linux 357
Создание правил и постоянных имен 359
Управление устройствами в системе FreeBSD 362
11.4. Конфигурирование ядра Linux 364
Конфигурирование параметров ядра linux 364
Сборка ядра 366
Добавление драйвера устройства в Linux 368
11.5. Конфигурация ядра системы FreeBSD 368
Настройка параметров ядра FreeBSD 368
Сборка ядра FreeBSD 369
11.6. Загружаемые модули ядра 370
Загружаемые модули ядра в Linux 371
Загружаемые модули ядра в системе FreeBSD 372
11.7. Загрузка 373
Загрузочные сообщения системы Linux 373
Загрузочные сообщения системы FreeBSD 377
11.8. Загрузка альтернативных ядер в облаке 378
11.9. Ошибки ядра 379
Ошибки ядра Linux 380
Паника ядра в системе FreeBSD 382
11.10. Литература 382

Глава 12. Печать 383

12.1. Система печати CUPS 384
Интерфейсы для системы печати 384
Очередь на печать 385
Множество принтеров 385
Экземпляры принтеров 386
Сетевая печать 386
Фильтры 387
12.2. Управление сервером CUPS 388
Настройка сетевого сервера печати 388
Автоматическое конфигурирование принтера 389
Конфигурирование сетевых принтеров 389
Примеры конфигурирования принтеров 390
Отключение принтера 390
Другие связанные с конфигурированием задачи 391
12.3. Советы по выявлению проблем 392
Повторный запуск демона печати 392
Регистрационные журналы 392

Проблемы с прямой печатью 393
Проблемы с печатью в сети 393
12.4. Литература 394

Часть II. Работа в сетях 395

Глава 13. Сети TCP/IP 397

13.1. Система TCP/IP и Интернет 397
Кто управляет Интернетом 398
Сетевые стандарты и документация 399
13.2. Основы работы в сети 400
Версии IPv4 и IPv6 401
Пакеты и их инкапсуляция 403
Стандарты формирования фреймов Ethernet 404
13.3. Адресация пакетов 405
Аппаратная адресация (MAC) 405
IP-адресация 406
"Адресация" имен машин 407
Порты 407
Типы адресов 408
13.4. IP-адреса 409
Классы адресов в протоколе IPv4 409
Подсети IPv4 410
Трюки и инструменты для арифметических вычислений, связанных с подсетями 411
CIDR: протокол бесклассовой междоменной маршрутизации 412
Выделение адресов 413
Частные адреса и система NAT 413
Адресация в стандарте IPv6 415
13.5. Маршрутизация 419
Таблицы маршрутизации 419
Директивы переадресации протокола ICMP 421
13.6. ARP: протокол преобразования адресов в IPv4 и IPv6 422
13.7. DHCP: протокол динамического конфигурирования хостов 423
Программное обеспечение DHCP 423
Схема работы DHCP 424
Программное обеспечение DHCP, созданное организацией ISC 425
13.8. Вопросы безопасности 426
Перенаправление IP-пакетов 426
Директивы переадресации протокола ICMP 426
Маршрутизация по адресу отправителя 427
Широковещательные пакеты эхо-запросов и другие виды направленных широковещательных сообщений 427
Подмена IP-адресов 427
Встроенные брандмауэры 428
Виртуальные частные сети 429
13.9. Основы конфигурирования сети 430
Присвоение сетевых имен и IP-адресов 430

Настройка сетевых интерфейсов и протокола IP 432
Настройка маршрутизации 433
Конфигурирование DNS 435
Сетевое конфигурирование в различных системах 435
13.10. Сетевое конфигурирование в системе Linux 436
Программа NetworkManager 436
Команда `ip`: ручное конфигурирование сети 437
Сетевое конфигурирование в системе Ubuntu 438
Сетевое конфигурирование в системе Red Hat и CentOS 438
Настройка сетевого оборудования в системе Linux 440
Опции протокола Linux TCP/IP 441
Переменные ядра, связанные с безопасностью 443
13.11. Сеть FreeBSD 444
Команда `ifconfig`: настройка сетевых интерфейсов 444
Конфигурация сетевого оборудования в системе FreeBSD 445
Конфигурирование сети во время загрузки системы FreeBSD 445
Конфигурирование протокола TCP/IP в системе FreeBSD 445
13.12. Сетевые проблемы 446
Команда `ping`: проверьте, работает ли хост 447
Команда `traceroute`: трассировка IP-пакетов 449
Пакетные анализаторы трафика 452
Утилита `tcpdump`: пакетный анализатор трафика из командной строки 453
13.13. Мониторинг сети 455
Программа SmokePing: постепенный сбор статистики об эхо-запросах 455
Программа iPerf: отслеживание производительности сети 456
Программа Cacti: сбор и отображение данных 457
13.14. Брандмауэры и система NAT 458
Утилита `iptables` в системе Linux: правила, цепочки и таблицы 458
IPFilter для UNIX-систем 463
13.15. Облачные сети 465
Виртуальное частное облако AWS (VPC) 465
Сеть на платформе Google Cloud Platform 472
Сеть DigitalOcean 473
13.16. Литература 474
История 474
Классика 474
Протоколы 475

Глава 14. Сетевые аппаратные средства 477

14.1. Технология Ethernet: сетевая панацея 478
Как работает Ethernet 479
Топология Ethernet 479
Неэкранированная витая пара 480
Оптическое волокно 482
Соединение и расширение сетей Ethernet 483
14.2. Беспроводные сети: локальная сеть для кочевников 487
Стандарты беспроводных сетей 487
Доступ клиентов к беспроводной сети 488

Беспроводные коммутаторы и точки беспроводного доступа 488
Безопасность беспроводных сетей 490
14.3. SDN: программно-коммутируемые сети 491
14.4. Тестирование и отладка сетей 491
14.5. Прокладка кабелей 492
Неэкранированная витая пара 492
Офисные точки подключения 492
Стандарты кабельных систем 493
14.6. Проектирование сетей 494
Структура сети и архитектура здания 494
Расширение сетей 494
Перегрузка 495
Обслуживание и документирование 495
14.7. Управление сетью 495
14.8. Рекомендуемые поставщики 496
Кабели и разъемные соединения 496
Тестовые приборы 497
Маршрутизаторы/коммутаторы 497
14.9. Литература 497

Глава 15. IP-маршрутизация 499

15.1. Подробнее о маршрутизации пакетов 500
15.2. Демоны и протоколы маршрутизации 503
Дистанционно-векторные протоколы 503
Топологические протоколы 504
Метрика стоимости 505
Внутренние и внешние протоколы 505
15.3. Основные протоколы маршрутизации 506
Протоколы RIP и RIPng 506
Протокол OSPF 507
Протокол EIGRP 508
BGP: протокол граничного шлюза 508
15.4. Многоадресатная координация протокола маршрутизации 508
15.5. Выбор критериев стратегии маршрутизации 509
15.6. Демоны маршрутизации 510
Демон `routed`: устаревшая реализация в протоколе RIP 511
Пакет Quagga: основной демон маршрутизации 511
Маршрутизатор XORP 512
15.7. Маршрутизаторы Cisco 512
15.8. Литература 515

Глава 16. DNS: система доменных имен 517

16.1. Архитектура DNS 518
Запросы и ответы 518
Поставщики услуг DNS 519
16.2. DNS для поиска 519
`resolv.conf`: конфигурация клиентского модуля распознавания 519
`nsswitch.conf`: кого я запрашиваю по имени? 520

16.3. Пространство имен DNS 521
Регистрация доменного имени 522
Создание собственных поддоменов 522
16.4. Как работает система DNS 522
Серверы имен 522
Авторитетные и кеширующие серверы 523
Рекурсивные и нерекурсивные серверы 524
Записи о ресурсах 524
Делегирование 525
Кеширование и эффективность 526
Неоднозначные ответы и балансировка загрузки DNS 527
Отладка с помощью инструментов запросов 527
16.5. База данных DNS 530
Команды синтаксического анализатора в файлах зон 530
Записи о ресурсах 531
Запись SOA 534
Записи NS 536
Записи A 537
Записи AAAA 537
Записи PTR 538
Записи MX 539
Записи CNAME 540
Записи SRV 541
Записи TXT 542
Записи SPF, DKIM и DMARC 542
Записи о ресурсах DNSSEC 542
16.6. Программное обеспечение BIND 543
Компоненты системы BIND 543
Файлы конфигурации 543
Инструкция include 545
Инструкция options 545
Инструкция acl 551
Инструкция key (TSIG) 552
Инструкция server 552
Инструкция masters 553
Инструкция logging 553
Инструкция statistics-channels 553
Инструкция zone 554
Инструкция controls для команды rndc 557
16.7. Расщепление DNS и инструкция view 558
16.8. Примеры конфигурации системы BIND 560
Зона локального хоста 560
Небольшая компания, предоставляющая консалтинговые услуги в области безопасности 561
16.9. Обновление файла зоны 564
Передача зоны 565
Динамические обновления в системе BIND 565

16.10. Вопросы безопасности DNS 568
Еще раз о списках управления доступом на сервере BIND 568
Открытые распознаватели 569
Работа в виртуальном окружении `chroot` 570
Безопасные межсерверные взаимодействия посредством технологий TSIG и TKEY 570
Настройка технологии TSIG для сервера BIND 571
Технология DNSSEC 573
Правила протокола DNSSEC 574
Записи о ресурсах DNSSEC 574
Настройка протокола DNSSEC 575
Генерирование пар ключей 576
Подписание зоны 578
Цепочка доверия в протоколе DNSSEC 580
Смена ключей DNSSEC 580
Инструменты DNSSEC 581
Отладка протокола DNSSEC 583
16.11. Отладка сервера BIND 584
Журнальная регистрация на сервере BIND 584
Некорректное делегирование 591
16.12. Литература 592
Книги и другая документация 593
Ресурсы в Интернете 593
Документы RFC 593

Глава 17. Система единого входа 595

17.1. Основные элементы системы единого входа 596
17.2. LDAP: "облегченные" службы каталогов 597
Особенности LDAP 597
Структура данных LDAP 598
OpenLDAP: традиционный LDAP-сервер с открытым исходным кодом 599
389 Directory Server: альтернативный LDAP-сервер с открытым исходным кодом 600
Создание LDAP-запросов 601
Преобразования файлов паролей и групп LDAP 602
17.3. Использование служб каталогов для входа в систему 603
Система Kerberos 603
Демон `sssd`: служба системной безопасности 606
`nsswitch.conf`: переключатель службы имен 607
Модули PAM: украшение или чудо аутентификации? 607
17.4. Альтернативные подходы 610
NIS: сетевая информационная служба 611
Утилита `rsync`: более безопасная рассылка файлов 611
17.5. Литература 611

Глава 18. Электронная почта 613

18.1. Архитектура почтовой системы 613
Пользовательские агенты 614

Агенты передачи 615
Транспортные агенты 615
Локальные агенты доставки 616
Хранилища сообщений 616
Агенты доступа 616
18.2. Структура почтового сообщения 617
18.3. Протокол SMTP 619
Вы прислали мне привет (EHLO) 620
Коды ошибок протокола SMTP 621
Аутентификация SMTP 621
18.4. Спам и вредоносные программы 622
Подделки 623
Технология SPF и спецификации Sender ID 623
Системы DKIM 624
18.5. Конфиденциальность и шифрование сообщений 624
18.6. Почтовые псевдонимы 625
Загрузка псевдонимов из файла 627
Направление почты в файл 628
Направление почты в программу 628
Хешированная база данных псевдонимов 628
18.7. Конфигурация электронной почты 629
18.8. Почтовый агент sendmail 630
Файл переключения 631
Запуск программы sendmail 631
Почтовые очереди 633
Препроцессор m4 634
Фрагменты конфигурации программы sendmail 635
Конфигурационный файл, построенный на основе эталонного файла с расширением .mc 636
Примитивы конфигурации программы sendmail 637
Таблицы и базы данных 637
Обобщенные макросы и функциональные возможности 638
Конфигурация клиентов 643
Параметры конфигурации препроцессора m4 644
Средства программы sendmail для борьбы со спамом 646
Ретрансляция 646
Безопасность и программа sendmail 649
Владельцы файлов 650
Права доступа 651
Безопасная пересылка почты в файлы и программы 651
Опции безопасности 652
Выполнение программы sendmail в виртуальном каталоге (для настоящих параноиков) 653
Отражение атак типа “отказ от обслуживания” 654
TLS: безопасносный протокол транспортного уровня 654
Тестирование и отладка программы sendmail 655
Журнальная регистрация 656

18.9. Почтовый агент Exim 657
Инсталляция почтового сервера Exim 658
Загрузка почтового сервера Exim 659
Утилиты почтового сервера Exim 660
Язык конфигурации программы Exim 661
Файл конфигурации программы Exim 661
Глобальные параметры 662
Сканирование содержимого на этапе применения списков управления доступом 667
Аутентификаторы 667
Маршрутизаторы 668
Транспортные механизмы 672
Конфигурация `retry` 672
Конфигурация перезаписи 673
Функция локального сканирования 673
Регистрация 673
Отладка 674
18.10. Почтовый агент Postfix 675
Архитектура системы Postfix 675
Безопасность 677
Команды и документация системы Postfix 677
Конфигурация системы Postfix 678
Виртуальные домены 682
Управление доступом 683
Отладка 686
18.11. Литература 687
Литература по программе `sendmail` 688
Литература о системе Exim 688
Литература о системе Postfix 688
Документы RFC 688

Глава 19. Веб-хостинг 689

19.1. HTTP: протокол передачи гипертекста 689
Унифицированные указатели ресурсов (URL) 690
Структура транзакции протокола HTTP 691
Утилита `curl`: инструмент командной строки для работы с HTTP 694
Повторное использование TCP-соединений 695
HTTP на основе протокола TLS 696
Виртуальные хосты 696
19.2. Основы программного обеспечения для веба 697
Веб-серверы и прокси-сервер протокола HTTP 698
Балансировщики нагрузки 699
Кеши 701
Сети доставки контента 704
Языки веба 705
Интерфейсы прикладного программирования (API) 707
19.3. Облачный веб-хостинг 708
Сборка или покупка 709

Платформа как услуга 709
Статический хостинг содержимого 710
Бессерверные веб-приложения 710
19.4. Веб-сервер Apache `httpd` 711
Использование веб-сервера `httpd` 712
Конфигурация логистики веб-сервера `httpd` 712
Настройка виртуального хоста 714
Базовая аутентификация протокола HTTP 715
Ведение журнала 717
19.5. Веб-сервер NGINX 718
Установка и запуск NGINX 719
Настройка веб-сервера NGINX 719
Настройка TLS для NGINX 722
Балансировка нагрузки с помощью NGINX 723
19.6. Программное обеспечение HAProxy 724
Проверки работоспособности 725
Статистика сервера 726
Липкие сессии 726
Прекращение использования TLS 727
19.7. Литература 728

Часть III. Хранение данных 729

Глава 20. Дисковая память 731

20.1. Добавление диска 732
Рецепт для Linux 733
Рецепт для FreeBSD 734
20.2. Аппаратное обеспечение для хранения данных 735
Жесткие диски 736
Твердотельные диски 739
Гибридные диски 742
Расширенный формат и блоки по 4 КиБ 743
20.3. Интерфейсы устройств для хранения данных 744
Интерфейс SATA 744
Интерфейс PCI Express 744
Интерфейс SAS 745
Интерфейс USB 746
20.4. Подключение и низкоуровневое управление накопителями 747
Проверка инсталляции на уровне аппаратного обеспечения 747
Файлы дисковых устройств 748
Непостоянные имена устройств 749
Форматирование дисков и управление сбойными секторами 749
Безопасное стирание дисков ATA 750
Команды `hdparm` и `camcontrol`: параметры диска и интерфейса (Linux) 752
Мониторинг жесткого диска с помощью стандарта SMART 752
20.5. Программное обеспечение накопителей 753
Отображение устройств в системе Linux 755

20.6. Разбиение диска 756
Традиционное разбиение 758
Разбиение диска по схеме MBR 759
Схема GPT: таблица разделов GUID 759
Разбиение дисков в системе Linux 760
Разбиение дисков в системе FreeBSD 760
20.7. Управление логическими томами 761
Управление логическими томами в системе Linux 761
Управление логическими томами в FreeBSD 766
20.8. RAID: избыточные массивы недорогих дисков 767
Программная и аппаратная реализации системы RAID 767
Уровни системы RAID 767
Восстановление диска после сбоя 770
Недостатки конфигурации RAID 5 771
Команда `mdadm`: программное обеспечение RAID в системе Linux 772
20.9. Файловые системы 776
20.10. Традиционные файловые системы: UFS, ext4 и XFS 777
Терминология файловых систем 778
Полиморфизм файловых систем 779
Форматирование файловых систем 779
Команда `fsck`: проверка и исправление файловых систем 779
Монтирование файловой системы 781
Настройка автоматического монтирования 781
Монтирование USB-накопителя 784
Включение подкачки 784
20.11. Файловые системы следующего поколения: ZFS и Btrfs 785
Копирование при записи 785
Обнаружение ошибок 786
Производительность 786
20.12. Файловая система ZFS: все проблемы решены 787
ZFS в системе Linux 788
Архитектура ZFS 788
Пример: добавление диска 789
Файловые системы и свойства 789
Наследование свойств 791
Один пользователь — одна файловая система 792
Мгновенные копии и клоны 792
Неразмеченные логические тома 794
Управление пулом памяти 794
20.13. Файловая системы Btrfs:
облегченная версия ZFS для Linux 796
Btrfs или ZFS 797
Настройка и преобразование хранилища 797
Тома и подтома 800
Снимки тома 800
Поверхностные копии 801
20.14. Стратегия резервного копирования данных 802
20.15. Литература 803

Глава 21. Сетевая файловая система NFS 805
21.1. Введение в протокол NFS 805
Конкуренция 806
Проблемы, связанные с состоянием 806
Проблемы производительности 807
Безопасность 807
21.2. Основные идеи, лежащие в основе протокола NFS 808
Версии и история протокола 808
Удаленный вызов процедур 809
Транспортные протоколы 810
Состояние 810
Экспорт файловой системы 810
Блокировка файлов 811
Вопросы безопасности 812
Идентифицирующее отображение в версии 4 813
Учетные записи `root` и `nobody` 814
Производительность версии 4 815
21.3. Серверная часть протокола NFS 815
Файл `exports` в системе Linux 816
Файл `exports` в системе FreeBSD 819
Демон `nfsd`: обслуживание файлов 820
21.4. Клиентская часть протокола NFS 822
Монтирование файловых систем NFS на этапе начальной загрузки 824
Ограничения экспорта привилегированными портами 824
21.5. Идентифицирующее отображение в протоколе NFS 4 825
21.6. Команда `nfsstat`: отображение статистики NFS 825
21.7. Специализированные файловые серверы NFS 826
21.8. Автоматическое монтирование 827
Таблицы косвенных назначений 828
Таблицы прямых назначений 829
Главные таблицы 829
Исполняемые таблицы 830
Видимость программы `automount` 830
Реплицированные файловые системы и программа `automount` 831
Автоматическое монтирование (V3; все, кроме Linux) 831
Специфика системы Linux 832
21.9. Литература 832

Глава 22. Файловая система SMB 833
22.1. Samba: сервер SMB для UNIX 834
22.2. Инсталляция и конфигурации пакета Samba 835
Совместное использование файлов с локальной аутентификацией 836
Совместное использование файлов с помощью учетных записей, прошедших аутентификацию Active Directory 836
Настройка общих ресурсов 837
22.3. Монтирование общих SMB-ресурсов 839
22.4. Просмотр файлов на общих SMB-ресурсах 840

22.5. Обеспечение безопасности Samba-сервера 840
22.6. Отладка Samba-сервера 841
Запрос состояния Samba-сервера с помощью команды `smbstatus` 841
Настройка журнала Samba-сервера 842
Управление наборами символов 843
22.7. Литература 843

Часть IV. Эксплуатация 845

Глава 23. Управление конфигурацией 847

23.1. Краткое введение в управление конфигурацией 848
23.2. Опасности управления конфигурацией 848
23.3. Элементы управления конфигурацией 849
Операции и параметры 849
Переменные 851
Факты 851
Обработчики изменений 852
Привязки 852
Пакеты и репозитории пакетов 853
Среды 853
Учет и регистрация клиентов 854
23.4. Сравнение популярных систем CM 855
Терминология 856
Бизнес-модели 856
Архитектурные параметры 856
Параметры языка 858
Варианты управления зависимостями 859
Общие комментарии по поводу систему Chef 861
Общие комментарии по поводу системы Puppet 862
Общие комментарии по поводу систем Ansible и Salt 863
Ода YAML 863
23.5. Введение в систему Ansible 865
Пример использования системы Ansible 866
Настройка клиента 868
Группы клиентов 870
Присваивание переменных 870
Динамические и вычисляемые группы клиентов 871
Списки задач 872
Параметры состояния 874
Итерация 874
Взаимодействие с Jinja 875
Визуализация шаблона 875
Привязки: сценарии и файлы сценариев 876
Роли 878
Рекомендации по структурированию базы конфигурации 879
Параметры доступа в системе Ansible 880
23.6. Введение в систему Salt 882
Настройка миньонов 884

Привязка значения переменной к миньону 886
Сопоставление миньонов 887
Состояния системы Salt 888
Система Salt и препроцессор Jinja 889
Идентификаторы состояний и зависимости 891
Функции состояния и выполнения 892
Параметры и имена 893
Привязка состояний к миньонам 896
Состояния высокого уровня 896
Формулы Salt 897
Среды 898
Документация 902
23.7. Сравнение систем Ansible и Salt 903
Гибкость развертывания и масштабируемость 903
Встроенные модули и расширяемость 903
Безопасность 904
Разное 904
23.8. Рекомендации 905
23.9. Литература 908

Глава 24. Виртуализация 909

24.1. Виртуальный жаргон 910
Гипервизоры 910
Динамическая миграция 913
Образы виртуальных машин 913
Контейнеризация 913
24.2. Виртуализация с помощью системы Linux 915
Платформа Xen 915
Инсталляция гостевой операционной системы на платформе Xen 916
Платформа KVM 917
Инсталляция гостевой операционной системы на платформе KVM и ее использование 918
24.3. Система FreeBSD bhyve 919
24.4. Компания VMWare 919
24.5. Гипервизор VirtualBox 919
24.6. Программа Packer 920
24.7. Программа Vagrant 922
24.8. Литература 922

Глава 25. Контейнеры 923

25.1. Основные концепции 924
Поддержка ядра 924
Образы 925
Сеть 926
25.2. Докер: механизм с открытым исходным кодом 926
Базовая архитектура 927
Инсталляция 928
Настройка клиента 929

Методики работы с контейнерами 929
Тома 933
Контейнеры данных 934
Сети Docker 934
Драйверы хранилищ 937
Изменение параметров настройки демона `dockerd` 938
Сборка образа 939
Реестры 942
25.3. Контейнеры на практике 944
Ведение журнала 945
Советы по безопасности 946
Отладка и устранение неполадок 948
25.4. Создание и управление контейнерными кластерами 949
Краткий обзор программного обеспечения для управления контейнерами 951
Kubernetes 951
Mesos и Marathon 952
Менеджер Docker Swarm 953
Контейнерная служба AWS EC2 953
25.5. Литература 954

Глава 26. Непрерывная интеграция и доставка 955

26.1. Основные концепции 957
Принципы и практика 957
Флаги функций 961
26.2. Конвейеры 961
Процесс сборки 962
Тестирование 963
Развертывание 965
Методы развертывания с нулевым временем простоя 966
26.3. Jenkins: сервер автоматизации с открытым исходным кодом 967
Основные концепции сервера Jenkins 967
Распределенные сборки 969
Конвейер как код 969
26.4. Подход CI/CD на практике 970
Тривиальное веб-приложение UlsahGo 971
Модульное тестирование UlsahGo 972
Знакомство с конвейером Jenkins Pipeline 973
Создание образа DigitalOcean 975
Обеспечение единой системы тестирования 977
Тестирование дроплета 980
Развертывание приложения UlsahGo на паре дроплетов и балансировщике нагрузки 980
Выводы, сделанные из демонстрационного конвейера 981
26.5. Контейнеры и упрощение среды CI/CD 982
Контейнеры как среда сборки 983
Контейнерные образы как артефакты сборки 983
26.6. Литература 984

Глава 27. Безопасность 985
27.1. Элементы безопасности 986
27.2. Слабые места в системе защиты 987
Социальная инженерия 987
Уязвимости в программах 988
Распределенные атаки типа “отказ в обслуживании” (DDoS) 989
Инсайдерская информация 989
Ошибки конфигурации сети, системы или приложения 990
27.3. Основные вопросы безопасности 990
Обновления программного обеспечения 991
Ненужные службы 992
Удаленная регистрация событий 992
Резервные копии 993
Вирусы и черви 993
Руткиты 994
Фильтрация пакетов 994
Пароли и многофакторная аутентификация 995
Бдительность 995
Тестирование приложений на проникновение 995
27.4. Пароли и учетные записи пользователей 996
Изменение пароля 997
Хранилища и депоненты паролей 997
Устаревание паролей 998
Групповые и совместно используемые учетные записи 999
Пользовательские оболочки 999
Привилегированные учетные записи 999
27.5. Инструментальные средства защиты 1000
Команда `nmap`: сканирование сетевых портов 1000
Nessus: сетевой сканер нового поколения 1002
Metasploit: программа для выявления попыток проникновения 1002
Lynis: встроенный аудит безопасности 1003
John the Ripper: средство для выявления слабых паролей 1003
Bro: программная система для распознавания вторжения в сеть 1004
Snort: популярная программная система для распознавания проникновения в сеть 1005
OSSEC: система для распознавания вторжения в сеть на уровне хоста 1005
Fail2Ban: система отражения атаки методом перебора 1008
27.6. Основы криптографии 1008
Криптография с симметричными ключами 1009
Криптография с открытым ключом 1009
Инфраструктура с открытым ключом 1010
Протокол защиты транспортного уровня TLS 1012
Криптографические хеш-функции 1012
Генерация случайных чисел 1014
Выбор криптографического программного обеспечения 1015
Команда `openssl` 1016
Отладка TLS-сеанса с сервером 1017

PGP: довольно хорошая конфиденциальность 1017
Kerberos: унифицированный подход к сетевой безопасности 1018
27.7. Система SSH 1019
Основы OpenSSH 1019
Клиент `ssh` 1021
Аутентификация с помощью открытого ключа 1022
Демон `ssh-agent` 1024
Псевдонимы хостов в файле `~/.ssh/config` 1025
Мультиплексирование соединения 1026
Проброс портов 1026
Демон `sshd`: сервер OpenSSH 1027
Проверка ключа хоста с помощью записи SSHFP 1029
Передача файлов 1030
Альтернативы для безопасного входа в систему 1030
27.8. Брандмауэры 1031
Брандмауэры, фильтрующие пакеты 1031
Принципы фильтрации служб 1031
Брандмауэры, осуществляющие инспекцию пакетов с отслеживанием состояния соединений 1032
Насколько безопасны брандмауэры 1032
27.9. Виртуальные частные сети (VPN) 1033
Туннели IPsec 1033
Так ли уж нужны виртуальные частные сети 1034
27.10. Сертификаты и стандарты 1034
Сертификаты 1034
Стандарты безопасности 1035
27.11. Источники информации по вопросам обеспечения безопасности 1038
Сервер SecurityFocus.com и списки рассылки BugTraq и OSS 1038
Блог Брюса Шнайера 1038
Отчет компании Verizon Data Breach Investigations 1038
Институт SANS 1038
Информационные ресурсы отдельных дистрибутивов 1039
Другие списки рассылки и веб-сайты 1039
27.12. Что нужно делать в случае атаки на сервер 1040
27.13. Литература 1041

Глава 28. Мониторинг 1043

28.1. Обзор мониторинга 1044
Инструментарий 1044
Типы данных 1045
Ввод и обработка 1045
Уведомления 1046
Контрольные панели и пользовательские интерфейсы 1047
28.2. Культура мониторинга 1047
28.3. Платформы мониторинга 1048
Платформы реального времени с открытым исходным кодом 1049
Платформы временных рядов с открытым исходным кодом 1050
Платформы визуализации данных с открытым исходным кодом 1052

Коммерческие платформы мониторинга 1052
Размещенные платформы мониторинга 1053
28.4. Сбор данных 1054
StatsD: протокол передачи общих данных 1054
Сбор данных из вывода команды 1056
28.5. Мониторинг сетей 1057
28.6. Мониторинг систем 1058
Команды для мониторинга систем 1059
Сборщик обобщенных системных данных `collectd` 1059
Утилиты `sysdig` и `dtrace`: трассировки выполнения 1060
28.7. Мониторинг приложений 1061
Мониторинг системного журнала 1061
Supervisor + Munin: простой вариант для некоторых предметных областей 1062
Коммерческие средства мониторинга приложений 1062
28.8. Мониторинг безопасности 1063
Проверка целостности системы 1063
Контроль обнаружения вторжений 1065
28.9. SNMP: простой протокол сетевого управления 1065
Организация протокола SNMP 1066
Операции протокола SNMP 1067
Net-SNMP: средства для серверов 1067
28.10. Советы и рекомендации по мониторингу 1069
28.11. Литература 1070

Глава 29. Анализ производительности 1071

29.1. Принципы настройки производительности 1072
29.2. Способы повышения производительности 1073
29.3. Факторы, влияющие на производительность 1075
29.4. Захваченные циклы центрального процессора 1076
29.5. Как анализировать проблемы производительности 1077
29.6. Проверка производительности системы 1078
Инвентаризуйте свое оборудование 1078
Сбор данных о производительности 1080
Анализ использования центрального процессора 1080
Управление памятью в системе 1082
Анализ использования памяти 1084
Анализ операций обмена с диском 1085
Утилита `fio`: анализ производительности дисковой подсистемы 1086
Команда `sar`: сбор статистических данных и генерирование отчетов по ним 1087
Выбор планировщика ввода-вывода в системах Linux 1088
Программа `perf`: универсальный профилировщик системы Linux 1089
29.7. Помогите! Мой сервер тормозит! 1090
29.8. Литература 1092

Глава 30. Центры обработки данных 1093
30.1. Стойки 1094
30.2. Электропитание 1094
Требования к электроснабжению стоек 1096
Измерение 1098
Стоимость 1098
Удаленное управление 1098
30.3. Охлаждение и окружающая среда 1098
Оценка нагрузки на систему охлаждения 1099
Теплые и холодные отсеки 1101
Влажность 1102
Мониторинг окружающей среды 1102
30.4. Уровни надежности центров обработки данных 1103
30.5. Безопасность центров обработки данных 1103
Местонахождение 1104
Периметр 1104
Доступ к объекту 1104
Доступ к стойке 1105
30.6. Инструменты 1105
30.7. Литература 1106

Глава 31. Методология, политика и стратегии 1107
31.1. Великая единая теория: DevOps 1108
DevOps — это CLAMS 1109
Системное администрирование в мире DevOps 1112
31.2. Системы управления билетами и задачами 1113
Общие функции билетных систем 1113
Владелец билета 1114
Восприятие пользователями билетных систем 1115
Типовые билетные системы 1116
Диспетчеризация билетов 1116
31.3. Поддержка локальной документации 1117
Инфраструктура как код 1118
Стандарты документации 1118
31.4. Разделение окружающей среды 1119
31.5. Восстановление после аварий 1120
Оценка рисков 1120
Планирование мероприятий по восстановлению 1121
Подбор персонала на случай аварии 1123
Проблемы с безопасностью 1123
31.6. Инструкции и процедуры 1124
Различие между инструкциями и процедурами 1125
Лучшие практики применения инструкций 1126
Процедуры 1126
31.7. Соглашения о качестве оказываемых услуг 1127
Спектр услуг и их описание 1127

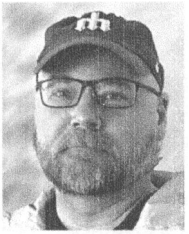
Стратегии управления очередями 1128
Показатели соответствия 1129
31.8. Соответствие законам и стандартам 1129
31.9. Правовые вопросы 1133
Конфиденциальность 1133
Реализация политики безопасности 1134
Контроль — это ответственность 1135
Лицензии на программное обеспечение 1135
31.10. Организации, конференции и другие ресурсы 1136
31.11. Литература 1137

Краткая история системного администрирования 1139

Рассвет компьютеризации: системные операторы (1952–1960) 1139
От узкой специализации к работе в режиме разделения времени (1961–1969) 1140
Рождение UNIX (1969–1973) 1140
UNIX становится знаменитой (1974–1990) 1142
Эра системных администраторов 1143
Документация по системному администрированию и обучение 1145
UNIX при смерти. Рождение Linux (1991–1995) 1145
Мир Windows (1996–1999) 1146
Расцвет UNIX и Linux (2000–2009) 1147
Системы UNIX и Linux в гипермасштабируемом облаке (2010– настоящее время) 1147
Завтрашний день UNIX и Linux 1148
Литература 1148

Предметный указатель 1149

О соавторах



Джеймс Гарнетт имеет степень доктора компьютерных наук в Университете Колорадо и является старшим инженером-программистом в компании Secure64 Software, Inc., где он разрабатывает технологии перехода с системы DDoS на ядра Linux. Если он не погружен в код ядра, значит, он находится где-то в глубине Каскадных гор в штате Вашингтон.



Фабрицио Бранка (`@fbrnc`) является ведущим разработчиком систем в компании AOE. Он, его жена и их двое детей только что вернулись в Германию после четырех лет жизни в Сан-Франциско. Фабрицио внес свой вклад в несколько проектов с открытым исходным кодом. Он фокусируется на архитектуре, инфраструктуре и высокопроизводительных приложениях. Кроме того, он разрабатывает процессы непрерывной разработки, тестирования и развертывания крупных проектов.



Адриан Муат (`@adrianmouat`) работает с контейнерами с самых первых дней появления технологии Docker. Он опубликовал книгу *Using Docker* (`amzn.to/2sVAIZt`) в издательстве О'Рейли. В настоящее время он является главным научным сотрудником общеевропейской компании Container Solutions, специализирующейся на консалтинге и разработке продуктов для микрослужб и контейнеров.

С общими комментариями и сообщениями об ошибках, пожалуйста, обращайтесь по адресу `ulsah@book.admin.com`. *Мы сожалеем, что не можем ответить на технические вопросы*

Об авторах



Эви Немет уволилась с факультета вычислительной техники Университета штата Колорадо в 2001 г. Многие годы она исследовала просторы Тихого океана на своей 40-футовой яхте "Wonderland" ("Страна чудес") до ее трагического исчезновения в 2013 г. Четвертое издание этой книги — это последнее издание, в котором она принимала активное участие, но мы изо всех сил старались сохранить ее текст там, где это было возможно.



Гарт Снайдер (`@GartSnyder`) работал в компаниях NeXT и Sun и получил степень бакалавра электротехники в колледже Суортмор, штат Пенсильвания, а также степени магистра медицины и делового администрирования в Университете Рочестера.



Трент Р. Хейн(`@trenthein`) — большой энтузиаст кибербезопасности и автоматизации. Помимо технологий, он любит езду на велосипеде, горные лыжи, ловлю рыбу нахлыстом, туристические походы, песни в стиле кантри, собак и оксфордскую запятую[1]. Трент получил степень бакалавра вычислительной техники в Университете штата Колорадо.



Бэн Уэйли — основатель компании WhaleTech, занимающейся независимым консалтингом. Он отмечен призом компании Amazon как один из выдающихся энтузиастов Amazon Web Services (AWS Community Heroes). Он получил степень бакалавра компьютерных наук в университете штата Колорадо в Боулдере.



Дэн Макин (`@dan_mackin`) получил ученую степень бакалавра электрической и компьютерной инженерии в университете штата Колорадо в Боулдере. Он применяет систему Linux и другие технологии с открытым исходным кодом не только для выполнения своих профессиональных обязанностей, но и в качестве хобби — для автоматизации сбора метеорологических данных. Дэн любит горные лыжи, парусный спорт, внутренний туризм, а также проводить время со своей женой и собакой.

---

[1] Запятая, которая ставится перед союзом and в конце перечисления. — *Примеч. ред.*

Памяти Эви

В каждой области знаний есть авторитет, который ее определяет и олицетворяет. В системном администрировании таким человеком является Эви Немет.

Это пятое издание книги, в которой Эви является главным автором. Хотя Эви не смогла физически присоединиться к нам, ее образ всегда был рядом, кроме того, ей принадлежат многие главы и примеры. Мы прилагали огромные усилия для сохранения необычного стиля Эви, для которого характерна объективность, техническая глубина и внимание к деталям.

Профессиональный математик и криптограф, Эви еще недавно работала профессором информатики в Университете Колорадо в Боулдере. То, как возникло системное администрирование, и какое участие Эви в нем приняла, подробно описано в последней главе этой книги "Краткая история системного администрирования".

На протяжении всей своей карьеры Эви с нетерпением ждала выхода на пенсию, чтобы отправиться в кругосветное плавание. В 2001 г. она наконец осуществила свою мечту — купила парусник Wonderland и отправилась в путешествие. Многие годы Эви приводила нас в восторг рассказами об удивительных островах, замечательных людях и парусных приключениях. Работая над двумя изданиями этой книги, Эви бросала якорь как можно ближе к берегу, чтобы загружать свои наброски через сети Wi-Fi.

Никогда не отказываясь от рискованных приключений, в июне 2013 г. Эви стала членом экипажа знаменитого парусника *Nina* и отправилась в плавание по Тасманскому морю. Вскоре после этого шхуна *Nina* исчезла в сильном шторме, и с тех пор мы ничего не слышали от Эви. Она жила своей мечтой.

Эви научила нас гораздо большему, чем системное администрирование. Даже когда ей исполнилось 70 лет, она оставалась лучшей: лучше всех организовывала сети, настраивала серверы, отлаживала ядра, колола дрова, жарила цыплят, пекла пироги и пила вино. Вместе с Эви все было достижимым.

Здесь невозможно кратко сформулировать всю мудрость Эви, но некоторые принципы глубоко внедрились в наши головы.

- Будьте консервативными по отношению к тому, что вы отправляете, и либеральными — к тому, что вы получаете.[1]
- Будьте либеральными к тому, кого вы нанимаете, но своевременно их увольняйте.
- Избегайте двусмысленности.
- Бакалавры — секретное сверхмощное оружие.
- Красных чернил не бывает слишком много.
- Вы не поймете, что вы делаете, пока не сделаете.
- Время для суши есть всегда.
- Не бойтесь попробовать что-то дважды.
- Всегда используйте программу `sudo`.

Мы уверены, что некоторые читатели спросят нас, что на самом деле означают некоторые из приведенных выше афоризмов. Мы, по примеру Эви, оставим это в качестве упражнения. Вы можете даже услышать, как она говорит: "Попробуйте сами. Посмотрите, как это работает".

Счастливого плавания, Эви. Мы скучаем по тебе.

---

[1]Этот принцип также известен как *Закон Постела*, названный в честь Джонатана Постела (Jon Postel), который был редактором серии документов RFC с 1969 г. до своей смерти в 1998 г.

Предисловие

Современные технологи — мастера в поиске ответов в Google. Если другой системный администратор уже столкнулся с проблемой (и, возможно, решил ее), вы найдете описание решения в Интернете. Мы приветствуем и поощряем этот открытый обмен идеями и решениями.

Если в Интернете есть так много информации, зачем нужно новое издание данной книги? Мы считаем, что эта книга способствует профессиональному росту системного администратора.

- Мы предлагаем принципы, руководство и контекст для надлежащего применения технологии. Важно рассмотреть любое пространство проблем с разных точек зрения. Необходимо владеть основами смежных дисциплин, таких как безопасность, согласованность, методология DevOps, облачные вычисления и жизненные циклы разработки программного обеспечения.
- Мы принимаем практический подход. Наша цель — обобщить коллективную точку зрения на системное администрирование и рекомендовать подходы, которые выдержали испытание временем. Эта книга содержит описания многочисленных случаев из практики и множество прагматических советов.
- Это книга не о том, как запустить операционную систему UNIX или Linux у себя дома, в гараже или на смартфоне. Вместо этого мы описываем управление производственными средами, такими как предприятия, правительственные учреждения и университеты. В этих средах есть требования, которые выходят далеко за пределы возможностей типичного любителя.
- Мы учим вас, как быть профессионалом. Эффективное системное администрирование требует как технических, так и программистских навыков. А также чувства юмора.

Организация книги

Книга разделена на четыре большие части: основы администрирования, сеть, хранение и эксплуатация.

Раздел "Основы администрирования" содержит широкий обзор операционных систем UNIX и Linux, с точки зрения системного администратора. Главы в этом разделе охватывают большинство фактов и методов, необходимых для запуска автономной системы.

Раздел "Сеть" описывает протоколы, используемые в системах UNIX, и методы, применяемые для настройки, расширения и поддержки сетей и серверов, работающих в Интернете. Здесь также рассматривается сетевое программное обеспечение высокого уровня. Среди предлагаемых тем — система доменных имен, электронная почта, единый вход и веб-хостинг.

В разделе "Хранение" рассматриваются проблемы хранения и управления данными. В этом разделе также рассматриваются подсистемы, которые допускают совместное использование файлов в сети, такие как сетевая файловая система и протокол SMB, совместимый с Windows.

Раздел "Эксплуатация" посвящен ключевым темам, с которыми сталкивается системный администратор на ежедневной основе при управлении производственными

средами. Эти темы включают в себя мониторинг, безопасность, производительность, взаимодействие с разработчиками и политику управления группой системного администрирования.

Авторы

Мы рады приветствовать Джеймса Гарнетта, Фабрицио Бранку и Адриана Муата в качестве соавторов этого издания. Глубокие знания этих ученых в разных областях значительно обогатили содержание книги.

Контактная информация

Пожалуйста, отправляйте предложения, комментарии и сообщения об ошибках на адрес `ulsah@book.admin.com`. Мы отвечаем на письма, но, пожалуйста, будьте терпеливы; иногда может пройти несколько дней до того, как один из нас сможет ответить. Из-за объема электронной почты, которая приходит на этот адрес, мы сожалеем, что не можем ответить на технические вопросы.

Чтобы просмотреть текущий список ошибок и другую актуальную информацию, посетите наш веб-сайт `admin.com`.

Надеемся, вам понравится эта книга, и желаем удачи в увлекательном системном администрировании!

Гарт Снайдер
Трент Р. Хейн
Бен Уэйли
Дэн Макин
Июль 2017 г.

Введение

В 1942 г. Уинстон Черчилль описал одну из первых битв Второй мировой войны: "Это еще не конец, это даже не начало конца, но это, возможно, конец начала". Я вспомнил эти слова, когда мне было предложено написать предисловие к пятому изданию "UNIX и Linux: руководство системного администратора". Исчезновение в море Эви Немет было большой печалью для сообщества UNIX, но я рад видеть ее наследие в виде этой книги и ее многочисленных достижений в области системного администрирования.

В основе Интернета изначально лежала система UNIX. В отличие от сложных коммерческих операционных систем своего времени система UNIX была минималистичной, ориентированной на инструменты, переносимой и широко используемой людьми, которые хотели делиться своей работой с другими. То, что мы сегодня называем программным обеспечением с открытым исходным кодом, в первые годы работы UNIX и Интернета было уже широко распространенным, но не имело названия. Технические и академические сообщества открыто публиковали свои результаты, потому что выгоды, очевидно, перевешивали издержки.

Подробные истории UNIX, Linux и Интернета были подробно описаны в других книгах. Я касаюсь этих очень тонких тем только для того, чтобы напомнить всем нам, что современный мир во многом обязан открытому исходному программному обеспечению и Интернету, а его первоисточником является UNIX.

Поскольку ранние UNIX- и интернет-компании стремились нанимать самых ярких людей и реализовывать самые инновационные функциональные возможности, переносимость программного обеспечения часто приносилась в жертву. В конце концов, системные администраторы вынуждены были учить немногое о многом, потому что никакие две операционные системы в стиле UNIX (тогда и сейчас) не были абсолютно одинаковыми. Являясь действующим системным администратором UNIX в середине 1980-х и позже, я должен был знать не только сценарии оболочки и конфигурацию Sendmail, но и драйверы устройств ядра. Также важно было знать, как исправить файловую систему с помощью восьмеричного отладчика. Веселые были времена!

Именно в это время появилось первое и все последующие издания данной книги. В разное время мы называли авторов "Эви и экипаж" или "Эви и ее дети". Поскольку я работал над программой Cron и сервером BIND, каждый раз, когда выходило очередное издание этой книги, Эви проводила одну-две недели со мной (с моей семьей и у меня на работе), чтобы убедиться, что книга содержит все, что нужно, в ней нет ошибок, а о каждой из программ сказано что-то уникальное и полезное. Честно говоря, общение с Эви было изнурительным, особенно когда ее что-то очень интересовало или приближался контрольный срок, или, как в моем случае, происходило и то, и другое. Тем не менее, как уже было сказано, я ужасно скучаю по Эви и ценю каждое воспоминание и каждую ее фотографию.

За десятилетия многое изменилось. Удивительно наблюдать, как эта книга развивается вместе с самой системой UNIX. Из каждого нового издания постепенно исчезают устаревшие и неактуальные технологии, чтобы освободить место темам, которые стали важными для администраторов UNIX, по крайней мере по мнению авторов.

Трудно поверить, что мы потратили десятки киловатт энергии на компьютеры размером с грузовик, чьи возможности теперь затмеваются смартфоном Android. В равной степени трудно поверить, что мы использовали сотни или тысячи серверных и настольных компьютеров с уже устаревшими технологиями, такими как `rdist`. В те годы из-

дания этой книги помогали таким людям, как я (и Эви), справляться с разнообразными, а иногда и особенными компьютерами, которые были *реальными*, а не виртуальными, и каждый из которых нужно было *поддерживать*, а не инсталлировать заново (или, как на платформе Docker, собирать заново) каждый раз, когда что-то требовало исправления или обновления.

Мы адаптируемся или сходим со сцены. "Дети Эви", которые продолжают наследие Эви, адаптировались и вернулись в этом пятом издании, чтобы рассказать вам, что необходимо знать о том, как работают современные компьютеры под управлением систем UNIX и Linux и как заставить их работать так, как вы хотите. Потеря Эви знаменует собой конец эры, но также поднимает вопрос о том, сколько аспектов системного администрирования ушло в историю вместе с ней. Я знаю десятки умных и успешных технологов, которые никогда не будут заделывать кабели в задней части стойки оборудования, не услышат тон модема и не увидят кабель RS-232. Это издание предназначено для тех, чьи системы живут в облаке или в виртуализированных центрах обработки данных; тех, чья административная работа в значительной степени принимает форму исходного кода автоматизации и конфигурации; тех, кто тесно сотрудничает с разработчиками, сетевыми инженерами, сотрудниками службы контроля и всеми другими рабочими пчелами, которые населяют современный улей.

Вы держите в руках новейшее, лучшее издание книги, чье рождение и эволюция точно отслеживали рождение и эволюцию UNIX и интернет-сообщества. Эви очень гордилась бы своими детьми, как из-за этой книги, так и из-за того, кем они оказались. Я ими горжусь.

Пол Викси (Paul Vixie)
Ла Хонда, Калифорния
Июнь 2017 г.

Благодарности

Многие люди внесли свой вклад в этот проект — от технических обзоров и конструктивных предложений до общей моральной поддержки. Следующие лица заслуживают особой благодарности.

Джейсон Каролан (Jason Carolan)	Нед Макклейн (Ned McClain)	Дэйв Рот (Dave Roth)
Рэнди Элсе (Randy Else)	Бет Мак-Элрой (Beth McElroy)	Питер Санкаускас (Peter Sankauskas)
Стив Геде (Steve Gaede)	Пол Нельсон (Paul Nelson)	Дипак Сингх (Deepak Singh)
Асиф Хан (Asif Khan)	Тим О'Рейли (Tim O'Reilly)	Пол Викси (Paul Vixie)
Сэм Лезерс (Sam Leathers)	Мадхури Пери (Madhuri Peri)	

Наш редактор в издательстве Pearson, Марк Тауб (Mark Taub), заслуживает огромной благодарности за его мудрость, терпеливую поддержку и покровительственное отношение на протяжении работы на книгой. Можно с уверенностью сказать, что это издание не получилось бы реализовать без него.

Мэри Лу Нор (Mary Lou Nohr) уже более 20 лет является нашим безжалостным редактором рукописей. Когда мы начали работу над этим изданием, Мэри Лу уже ушла на заслуженную пенсию. После многочисленных и продолжительных просьб она согласилась присоединиться к нам на бис. (И Мэри Лу Нор, и Эви Немет изображены на обложке. Вы можете найти их?)

У нас была фантастическая команда технических рецензентов. Три преданных друга оценили всю книгу: Джонатан Корбет (Jonathan Corbet), Пэт Парсегян (Pat Parseghian) и Дженнин Таунсенд (Jennine Townsend). Мы высоко ценим их упорство и тактичность.

Удивительные картинки и обложка этого издания были задуманы и исполнены Лизой Хейни (Lisa Haney). Ее портфолио находится в режиме онлайн на сайте `lisahaney.com`.

И последнее, но не менее важное: спасибо Ласло Немет (Laszlo Nemeth) за его готовность поддержать продолжение этой серии.

От издательства

Вы, читатель этой книги, и есть главный ее критик и комментатор. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересно услышать и любые другие замечания, которые вам хотелось бы высказать в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать нам электронное письмо, либо просто посетить наш веб-сайт и оставить свои замечания там. Одним словом, любым удобным для вас способом дайте нам знать, нравится или нет вам эта книга, а также выскажите свое мнение о том, как сделать наши книги более интересными для вас.

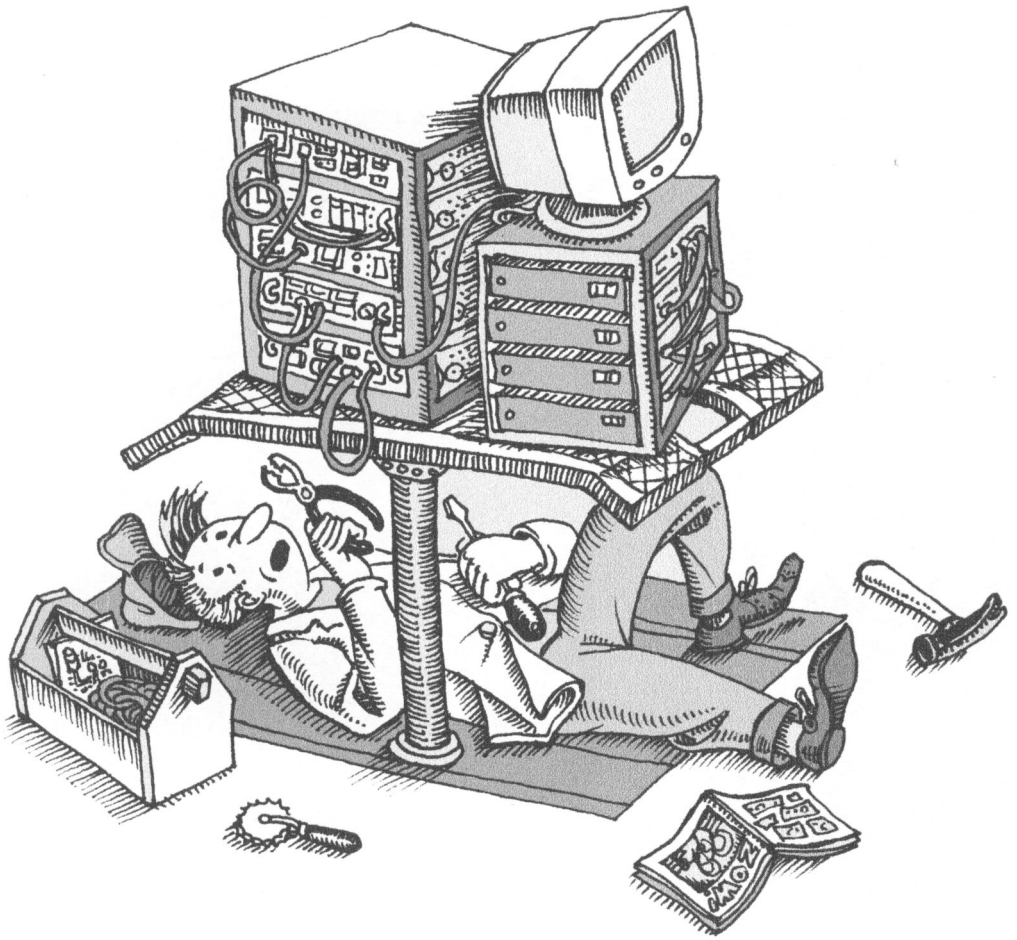
Посылая письмо или сообщение, не забудьте указать название книги и ее авторов, а также ваш обратный адрес. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию последующих книг.

Наши электронные адреса:

E-mail: `info@dialektika.com`
WWW: `http://www.dialektika.com`

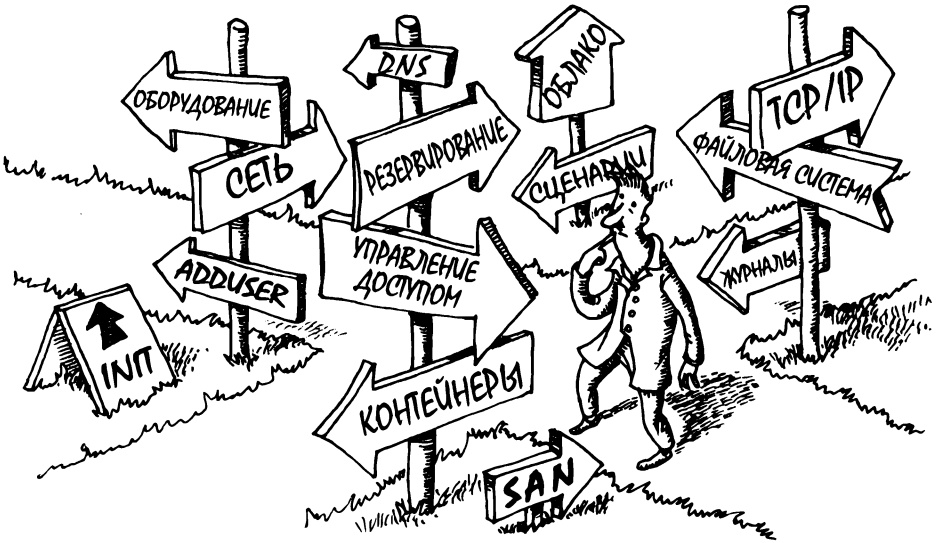
ЧАСТЬ I

Основы администрирования



Глава 1

С чего начать



Мы написали эту книгу, чтобы занять конкретную нишу в обширной экосистеме man-страниц, блогов, журналов, книг и других справочных материалов, которые должны удовлетворять потребности системных администраторов UNIX и Linux.

Во-первых, это общее руководство. В нем рассматриваются основные системы управления, выделяются разные части каждой из них и объясняется, как они работают вместе. Во многих случаях, когда вы должны выбирать между различными реализациями определенной концепции, мы описываем преимущества и недостатки самых популярных возможностей.

Во-вторых, это краткий справочник, в котором собрано то, что необходимо знать, чтобы выполнять типичные задачи на множестве распространенных систем UNIX и Linux. Например, команда **ps**, показывающая состояние выполняемых процессов, поддерживает более 80 параметров командной строки в системах Linux. Однако всего несколько комбинаций этих параметров удовлетворяют большинство потребностей системного администратора; мы приводим их в разделе 4.3.

Наконец, в этой книге основное внимание уделяется управлению корпоративными серверами и сетями, т.е. серьезному профессиональному системному администрированию. Легко настроить одну систему; сложнее сохранить распределенную облачную платформу, работающую без сбоев, несмотря на опасность распространения вирусов, нарушение связности сетей и целенаправленные атаки. Мы описываем методы и эмпирические правила, помогающие восстанавливать системы после сбоев, и предлагаем решения, которые масштабируются по росту размеров, сложности и неоднородности вашей империи.

Мы не претендуем на полную объективность, но считаем, что выразили свои предпочтения достаточно ясно. Одной из интересных особенностей системного администриро-

вания является то, что разумные люди могут иметь совершенно разные представления о наиболее подходящем решении. Мы предлагаем наши субъективные мнения как информацию к размышлению. Решайте сами, до какой степени вы с ними согласны и насколько они соответствуют вашей среде.

1.1. Основные обязанности системного администратора

В приведенных ниже разделах описаны основные задачи, которые должны выполнять администраторы. Эти обязанности не всегда возлагаются на одного человека, во многих организациях работа распределяется между несколькими членами команды. Однако по крайней мере один человек должен разбираться во всех компонентах и обеспечивать правильное решение каждой задачи.

Управление доступом

Системный администратор создает учетные записи для новых пользователей, удаляет учетные записи неактивных пользователей и решает все связанные с учетными записями проблемы, которые могут возникнуть (например, забытые пароли и потерянные пары ключей). Процесс добавления и удаления учетных записей обычно автоматизируется с помощью системы управления конфигурацией или централизованной службы каталогов.

🕮 Информация о работе с учетными записями пользователей приведена в главах 8, 17 и 23.

Добавление оборудования

Администраторы, работающие с физическим оборудованием (в отличие от облачных систем и систем, размещенных на виртуальном сервере), должны устанавливать и настраивать его так, чтобы операционная система могла его распознать. Функции поддержки оборудования могут варьироваться от простой задачи добавления сетевой интерфейсной карты до настройки специализированного внешнего массива хранения.

Автоматизация задач

Использование инструментов для автоматизации повторяющихся и трудоемких задач повышает эффективность работы, снижает вероятность ошибок и повышает вашу способность быстро реагировать на изменяющиеся требования. Администраторы стремятся сократить количество ручного труда, необходимого для бесперебойной работы систем. Знакомство с языками сценариев и инструментами автоматизации — важная часть работы.

🕮 Информацию о сценариях и автоматизации см. в главе 7.

Управление резервными копиями

Резервное копирование данных и их восстановление при необходимости являются важными административными задачами. Хотя резервное копирование — долгое и скучное занятие, частота бедствий в реальном мире просто слишком велика, чтобы можно было игнорировать эту работу.

🕮 Советы по резервному копированию см. в разделе 20.13.

Существуют операционные системы и отдельные пакеты программного обеспечения, которые зарекомендовали себя как хорошие инструменты и методы для облегчения

резервного копирования. Резервные копии должны выполняться по регулярному расписанию, а их восстановление должно периодически проверяться, чтобы гарантировать правильную работу.

Установка и обновление программного обеспечения

Программное обеспечение необходимо выбирать, инсталлировать и настраивать, причем, как правило, в разных операционных системах. По мере обновления заплаток и обновлений безопасности их необходимо тестировать, анализировать и внедрять в локальную среду, не ставя под угрозу стабильность производственных систем.

🕮 Информацию об управлении программным обеспечением см. в главе 6.

Термин *поставка программного обеспечения* относится к процессу выпуска обновленных версий, главным образом, собственного программного обеспечения. *Непрерывная поставка* переносит этот процесс на следующий уровень, предусматривая автоматическую регулярную поставку программного обеспечения пользователям по мере его разработки. Администраторы помогают внедрять надежные процессы поставки, отвечающие требованиям предприятия.

🕮 Информацию о развертывании и непрерывной поставке программного обеспечения см. в главе 26.

Мониторинг

Работа над решением проблемы обычно требует меньше времени, чем документирование и создание отчетов, и пользователи, работающие в организации, зачастую следуют по пути наименьшего сопротивления. Внешние пользователи чаще открыто публикуют свои жалобы, чем просят о помощи. Администраторы могут предотвратить жалобы, обнаруживая и устраняя проблемы до их открытого проявления. К задачам мониторинга относится обеспечение того, чтобы веб-службы быстро и корректно реагировали на события, собирали и анализировали файлы журналов и отслеживали доступность ресурсов сервера, таких как дисковое пространство. Все это открывает отличные возможности для автоматизации, а множество систем мониторинга (как коммерческих, так и с открытым исходным кодом) могут помочь системным администраторам решить эти задачи.

🕮 Информацию о мониторинге см. в главе 28.

Исправление проблем

Сбои сетевых системы происходят неожиданно, а иногда приводят к тяжелым последствиям. В обязанности администратора входят функции механика, который диагностирует проблемы и при необходимости прибегает к помощи экспертов. Поиск источника проблемы часто сложнее, чем ее решение.

🕮 Информацию о сетевых проблемах см. в разделе 13.12.

Ведение локальной документации

Администраторы выбирают поставщиков, пишут сценарии, разворачивают программное обеспечение и принимают множество других решений, которые могут быть не сразу очевидными или интуитивно понятными другим. Полная и точная документация является благом для членов команды, которые в противном случае вынуждены были

бы ночами выполнять обратное проектирование системы, чтобы устранить проблему. Подробная схема сети полезнее, чем многословный текст в описании проекта.

🕮 Информацию о локальной документации см. в разделе 31.3.

Бдительный мониторинг безопасности

Администраторы — первая линия защиты сетевых систем. Администратор должен выполнять правила безопасности и вырабатывать процедуры для предотвращения нарушений. В зависимости от контекста эти функции могут включать как несколько простых проверок несанкционированного доступа, так и сложную сеть ловушек и программ аудита. Системные администраторы осторожны по своей природе и часто являются специалистами по безопасности в технических организациях.

🕮 Информацию о системах безопасности см. в главе 27.

Настройка производительности

UNIX и Linux — это операционные системы общего назначения, которые хорошо подходят практически для любой мыслимой вычислительной задачи. Администраторы могут адаптировать системы для достижения оптимальной производительности в соответствии с потребностями пользователей, доступной инфраструктурой и услугами, предоставляемыми системами. Если сервер работает плохо, задача администратора — проанализировать его работу и определить области, которые нуждаются в улучшении.

🕮 Информацию о вопросах производительности см. в главе 29.

Разработка правил

По нормативно-правовым причинам в большинстве организаций необходимо установить правила, регулирующие допустимое использование компьютерных систем, управление данными и их хранение, конфиденциальность и безопасность сетей и систем, а также другие вопросы, представляющие интерес. Системные администраторы часто помогают организациям вырабатывать разумные правила, которые отвечают букве и духу закона и одновременно способствуют успеху и производительности.

🕮 Информацию о разработке локальных правил см. в разделе 1.8.

Работа с поставщиками

Для предоставления разнообразных вспомогательных услуг и продуктов, связанных с их вычислительной инфраструктурой, большинство организаций прибегает к посторонней помощи. Эту помощь могут оказывать разработчики программного обеспечения, поставщики облачной инфраструктуры, продавцы программных продуктов как услуг (service-as-a-service — SaaS), сотрудники службы поддержки, консультанты, подрядчики, эксперты по безопасности и поставщики платформ или инфраструктур. Администраторам можно поручить выбирать поставщиков, помогать в переговорах по контрактам и внедрять решения после завершения работы над документами.

Тушение пожаров

Хотя помощь другим людям с их разнообразными проблемами редко включается в описание функциональных обязанностей системного администратора, эти задачи занимают значительную часть большинства рабочих дней администраторов. На системных

администраторов обрушивается град жалоб, начиная с "Он еще вчера работал, а сегодня нет! Что вы изменили?" и заканчивая "Я пролил кофе на клавиатуру! Можно ли вылить на нее воду, чтобы смыть кофе?"

В большинстве случаев ваши ответы на эти вопросы гораздо больше влияют на то, насколько ценным администратором вас будут считать, чем любые технические навыки, которыми вы могли бы обладать. Вы можете либо выть от несправедливости, либо радоваться тому, что за одну хорошо выполненную просьбу о помощи вам будет начислено больше баллов, чем за пять часов ночной работы. Выбор за вами!

1.2. Предварительный опыт

В этой книге мы предполагаем, что у вас есть определенный опыт работы с системами Linux или UNIX. В частности, у вас должно быть общее представление о том, как система выглядит и воспринимается с точки зрения пользователя, поскольку мы не рассматриваем этот материал. Повысить уровень вашей квалификации помогут несколько хороших книг; см. раздел "Рекомендуемое чтение" в конце главы.

Мы любим хорошо продуманные графические интерфейсы. К сожалению, инструменты графического пользовательского интерфейса для системного администрирования в системах UNIX и Linux остаются рудиментарными по сравнению с богатством базового программного обеспечения. В реальном мире администраторам должно быть удобно использовать командную строку.

Для редактирования текста мы настоятельно рекомендуем изучить редактор **vi** (теперь он чаще всего рассматривается в расширенной форме, **vim**), который является стандартным для всех систем. Он простой, мощный и эффективный. Освоение редактора **vim** — это, пожалуй, лучший способ повышения производительности, доступный для администраторов. Используйте команду **vimtutor** для отличного интерактивного введения в эту тему.

Существует альтернатива — редактор **nano** с графическим пользовательским интерфейсом, представляющий собой простой и компактный "редактор для начинающих" с экранными подсказками. Не используйте его открыто; профессиональным администраторам зачастую очень не нравится, когда их коллеги используют этот редактор.

Хотя администраторы обычно не считаются разработчиками программного обеспечения, отраслевые тенденции размывают границы между этими функциями. Эффективные администраторы, как правило, знают много языков программирования и не прочь освоить новый язык, если возникнет такая необходимость.

🕮 Введение в сценарии см. в главе 7.

Для создания новых сценариев мы рекомендуем языки Bash (или **bash**, или **sh**), Ruby или Python. Bash — это командная оболочка, принятая по умолчанию для большинства систем UNIX и Linux. Bash примитивен как язык программирования, но является хорошим средством интеграции инструментов системного администратора. Python — это умный язык с хорошо понятным синтаксисом, широким сообществом разработчиков и библиотеками, которые облегчают решение многих задач. Разработчики, использующие язык Ruby, описывают его как "приятный" и "красивый". Языки Ruby и Python во многом похожи, и мы обнаружили, что они обладают одинаковыми функциональными возможностями для управления системами. Выбор между ними в основном зависит от личных предпочтений.

Мы также предполагаем, что вы умеете работать с инструментом **expect**, который представляет собой не язык программирования, а интерфейс для запуска интерактивных

программ. Это эффективная технология интеграции, которая может заменить некоторые сложные сценарии и проста в освоении.

Самые важные факты, которые необходимо знать о сценариях для работы с языками Bash, Python и Ruby, обобщаются в главе 7. В ней также рассматриваются регулярные выражения (шаблоны соответствия текста) и некоторые идиомы оболочки, полезные для системных администраторов.

1.3. Дистрибутивы Linux

В дистрибутив Linux входит ядро операционной системы и пакеты, содержащие все команды, которые можно выполнять в системе. Все дистрибутивы имеют одну и ту же линейку ядер, но формат, тип и количество пакетов немного отличаются. Дистрибутивы также различаются по направленности, уровню поддержки и популярности. По-прежнему существуют сотни независимых дистрибутивов Linux, но мы считаем, что в ближайшие годы в производственных средах будут преобладать дистрибутивы Debian и Red Hat.

По большому счету, различия между дистрибутивами Linux не являются очень уж большими. Довольно странно, что существует так много разных дистрибутивов, каждый из которых утверждает, что он отличается "легкой инсталляцией" и "крупной библиотекой программного обеспечения". Трудно избежать вывода о том, что людям просто нравится создавать новые дистрибутивы Linux.

Большинство основных дистрибутивов имеют относительно безболезненную процедуру инсталляции, среду рабочего стола и некоторую форму управления пакетами. Вы можете легко их испытать, запустив экземпляр на облаке или на локальной виртуальной машине.

По большей части незащищенность операционных систем общего назначения является следствием их сложности. Практически все ведущие дистрибутивы забиты множеством неиспользуемых пакетов программного обеспечения; в результате часто возникают уязвимости в области безопасности и сложности в управлении. В ответ появилось относительно новое поколение минималистских дистрибутивов. Лидером среди них является система CoreOS, которая предпочитает запускать все программное обеспечение в контейнерах. Alpine Linux — это легкий дистрибутив, который используется в качестве основы для многих публичных образов Docker. Учитывая эту редукционистскую тенденцию, мы ожидаем, что в ближайшие годы доля дистрибутивов Linux будет сокращаться.

Выбрав дистрибутив, вы делаете инвестиции в какой-то конкретный способ работы. Вместо того чтобы учитывать только функции инсталлированного программного обеспечения, разумно посмотреть, как ваша организация и конкретный поставщик будут работать друг с другом. Перечислим некоторые важные вопросы.

- Будет ли существовать этот дистрибутив через пять лет?
- Будет ли дистрибутив распространяться поверх последних исправлений уязвимости?
- Имеет ли этот дистрибутив активное сообщество и достаточную документацию?
- Если у меня возникнут проблемы, будет ли продавец говорить со мной и сколько это будет стоить?

Некоторые из самых популярных распространенных дистрибутивов перечислены в табл. 1.1.

Наиболее жизнеспособные дистрибутивы не обязательно являются корпоративными. Например, мы ожидаем, что дистрибутив Debian Linux (ладно-ладно, Debian GNU/Linux!) останется жизнеспособным в течение длительного времени, несмотря на то, что Debian не является компанией, ничего не продает и не предлагает поддержки на уров-

не предприятия. Проект Debian извлекает выгоду из существования преданной группы участников и огромной популярности дистрибутива Ubuntu, основанного на нем.

Таблица 1.1. Самые распространенные общедоступные дистрибутивы Linux

Дистрибутив	Веб-сайт	Комментарии
Arch	`archlinux.org`	Для тех, кто не боится командной строки
CentOS	`centos.org`	Бесплатный аналог Red Hat Enterprise
CoreOS	`coreos.com`	Все в контейнерах
Debian	`debian.org`	Бесплатный, как и большинство дистрибутивов GNUish
Fedora	`fedoraproject.org`	Испытательный стенд для Red Hat Linux
Kali	`kali.org`	Для специалистов по поиску уязвимых мест
Linux Mint	`linuxmint.com`	Ubuntu для настольных компьютеров
openSUSE	`opensuse.org`	Бесплатный аналог SUSE Linux Enterprise
openWRT	`openwrt.org`	Linux для маршрутизаторов и встроенных устройств
Oracle Linux	`oracle.com`	Версия RHEL, поддерживаемая Oracle
RancherOS	`rancher.com`	20МиБ, все в контейнерах
Red Hat Enterprise	`redhat.com`	Надежный, медленно меняющийся, коммерческий
Slackware	`slackware.com`	Древний, долгоживущий дистрибутив
SUSE Linux Enterprise	`suse.com`	Очень популярный в Европе, многоязычный
Ubuntu	`ubuntu.com`	Улучшенная версия Debian

Полный список дистрибутивов, включая множество неанглоязычных, можно найти на сайтах `lwn.net/Distributions` или `distrowatch.com`.

🕮 Дополнительную информацию о платформе Docker и контейнерах см. в главе 25.

1.4. Примеры систем, используемых в этой книге

В качестве основных примеров для этой книги мы выбрали три популярных дистрибутива Linux и один вариант UNIX: Debian GNU/Linux, Ubuntu Linux, Red Hat Enterprise Linux (и его клон CentOS) и FreeBSD. Эти системы являются репрезентативными для общего рынка с учетом значительной части инсталляций, используемых сегодня на крупных объектах.

Информация в этой книге обычно относится ко всем рассматриваемым системам, если не указано иное. Детали, характерные для отдельной системы, отмечены логотипом.


Debian GNU/Linux 9.0 “Stretch”


Ubuntu® 17.04 “Zesty Zapus”

RHEL 
Red Hat® Enterprise Linux® 7.1 и CentOS® 7.1


FreeBSD® 11.0

Большинство этих торговых марок принадлежат производителям, выпускающим соответствующее программное обеспечение, и используются с любезного разрешения их владельцев. Тем не менее поставщики не рецензировали и не утверждали содержание этой книги.

Мы неоднократно пытались, но не смогли получить разрешение от Red Hat на использование их логотипа с красной шляпой, поэтому были вынуждены использовать акроним RHEL.

Более подробная информация о каждой из иллюстративных систем приводится в нижеследующих разделах.

Примеры дистрибутивов Linux



Информация, характерная для Linux, а не для какого-либо конкретного дистрибутива, отмечена логотипом с пингвином Тукс, показанным слева.



Debian (произносится как "деб-ян" в честь покойного основателя Яна Мердока (Ian Merdock) и его жены Дебры (Debra)) является одним из самых старых и наиболее популярных дистрибутивов. Это некоммерческий проект с более чем тысячами участников по всему миру. Проект Debian поддерживает идеологическую приверженность развитию сообщества и открытому доступу, поэтому для него никогда не возникает вопросов о том, какие части дистрибутива являются свободными или распространяемыми.

Существуют три выпуска дистрибутива Debian, которые поддерживаются одновременно: стабильный, нацеленный на промышленные серверы; нестабильный, содержащий пакеты, которые могут иметь ошибки и уязвимости с точки зрения безопасности; и тестовый, который находится где-то посредине.



Дистрибутив Ubuntu основан на проекте Debian и поддерживает его приверженность бесплатному программному обеспечению с открытым исходным кодом. За системой Ubuntu стоит бизнес — компания Canonical Ltd., основанная предпринимателем Марком Шаттлвортом (Mark Shuttleworth).

Компания Canonical предлагает множество выпусков Ubuntu, предназначенных для облаков, настольных компьютеров и серверов без программного обеспечения. Существуют даже выпуски, предназначенные для телефонов и планшетов. Номера версий Ubuntu означают год и месяц, например версия 16.10 появилась в октябре 2016 года. Каждая версия также имеет аллитерационное кодовое имя, такое как Vivid Vervet или Wily Werewolf.

Ежегодно выпускаются две версии Ubuntu: в апреле и октябре. Апрельские выпуски в четные годы представляют собой долгосрочные версии поддержки (long-term support — LTS), которые обещают пять лет обновлений технического обслуживания. Это выпуски, рекомендованные для использования в производстве.

RHEL Система Red Hat была доминирующей силой в мире Linux более двух десятилетий, и ее дистрибутивы широко используются в Северной Америке и за ее пределами. По количеству инсталляций Red Hat, Inc. является самой успешной в мире компанией с открытым исходным кодом.

Система Red Hat Enterprise Linux, часто сокращаемая до RHEL, ориентирована на производственные среды крупных предприятий, которые требуют поддержки и консалтинговых услуг для бесперебойной работы своих систем. Как ни парадоксально, система RHEL является открытым исходным кодом, но требует лицензии. Если вы не хотите платить за лицензию, вы не имеете права инсталлировать Red Hat.

Компания Red Hat также спонсирует систему Fedora, коллективно разработанный дистрибутив, служащий инкубатором для ультрасовременного программного обеспечения, которое не считается достаточно стабильным для RHEL. Fedora используется в ка-

честве исходного тестового стенда для программного обеспечения и конфигураций, которые позже находят свой путь к RHEL.



Дистрибутив CentOS практически идентичен Red Hat Enterprise Linux, но он бесплатный. CentOS Project (`centos.org`) принадлежит Red Hat и выполняется ее ведущими разработчиками. Однако они работают отдельно от команды Red Hat Enterprise Linux. На дистрибутив CentOS не распространяется торговая марка Red Hat и в нем нет нескольких патентованных инструментов, но в других отношениях они эквивалентны.

CentOS — отличный выбор для организаций, которые хотят развернуть производственный дистрибутив, не выплачивая десятину Red Hat. Возможен и гибридный подход: основные серверы могут запускать Red Hat Enterprise Linux и пользоваться превосходной поддержкой Red Hat, в то время как для непроизводственных систем может использоваться система CentOS. Это решение учитывает все важные аспекты с точки зрения риска и поддержки, а также минимизирует стоимость и сложность управления.

Система CentOS стремится к полной двоичной совместимости и совместимости по спецификациям и отклонениям с системой Red Hat Enterprise Linux. Вместо того чтобы постоянно повторять "Red Hat и CentOS", в этой книге мы обычно упоминаем только одну из них. Текст зачастую в равной степени относится и к Red Hat, и к CentOS, если не указано обратное.

Другие популярные дистрибутивы также являются потомками Red Hat. Компания Oracle продает переименованную и видоизмененную версию CentOS клиентам своего программного обеспечения для обслуживания корпоративных баз данных. Дистрибутив Amazon Linux, доступный для пользователей служб Amazon Web Servises, первоначально был создан на основе системы CentOS и по-прежнему разделяет многие ее соглашения.

Большинство администраторов в какой-то момент своей карьеры обязательно столкнутся с системой, подобной Red Hat, и знакомство с ее нюансами полезно, даже если эта система не установлена в вашей организации.

Пример дистрибутива UNIX

Популярность UNIX в течение некоторого времени постепенно ослабевает, и большинство устаревших дистрибутивов UNIX (например, Solaris, HP-UX и AIX) больше не используются. Открытый исходный код системы BSD является исключением из этой тенденции и продолжает оставаться предметом культового поклонения, особенно среди экспертов по операционным системам, приверженцев открытого программного обеспечения и системных администраторов, нацеленных на безопасность. Другими словами, некоторые из ведущих операционных систем в мире полагаются на различные дистрибутивы BSD. Например, система MacOS компании Apple использует наследие BSD.



Дистрибутив FreeBSD, впервые выпущенный в конце 1993 г., является наиболее широко используемым производным инструментом BSD. В соответствии со статистикой использования он занимает 70% доли рынка версий BSD. Среди его пользователей встречаются крупные интернет-компании, такие как WhatsApp, Google и Netflix. В отличие от Linux, FreeBSD — это полная операционная система, а не только ядро. Как программное обеспечение ядра, так и пользовательское программное обеспечение лицензируются в соответствии с разрешительной лицензией BSD, что способствует развитию и расширению бизнес-сообщества.

1.5. Обозначения и типографские соглашения

Имена файлов, команды и аргументы команд, которые следует набирать на клавиатуре без изменений, даны полужирным шрифтом. Аргументы, вместо которых следует подставлять конкретные значения, даны курсивом. Например, в команде

cp *файл каталог*

предполагается, что аргумент *файл* следует заменить именем реального файла, а аргумент *каталог* — именем реального каталога.

Фрагменты сценариев и файлов конфигурации даны моноширинным шрифтом. Комментарии к интерактивным сеансам иногда сопровождаются символом комментария языка **bash** и выделяются курсивом, например:

```
$ grep Bob /pub/phonelist  # Найти номер телефона Боба
Bob Knowles 555-2834
Bob Smith 555-2311
```

Символ $ обозначает приглашение оболочки ввести данные, адресованное обычному, непривилегированному пользователю. Приглашение для привилегированного пользователя начинается символом #. Если команда является специфичной для дистрибутива или семейства дистрибутивов, символ приглашения сопровождается названием дистрибутива. Например:

```
$ sudo su - root # Стать привилегированным пользователем
# passwd         # Изменить пароль суперпользователя root
debian# dpkg -l  # Вывести инсталлированные
                 # пакеты в Debian и Ubuntu
```

Это соглашение принято в стандартных оболочках UNIX и Linux.

За исключением специальных случаев мы старались избегать использования особых шрифтов и форматов, чтобы не отвлекать читателей. Например, мы часто говорим о таких сущностях, как группа daemon, никак не выделяя их с помощью отдельного формата.

При описании синтаксиса команд мы, как правило, используем те же обозначения, что и в интерактивном руководстве:

- текст, заключенный в квадратные скобки ("[" и "]"), является необязательным;
- текст, после которого стоит многоточие ("..."), можно повторять;
- фигурные скобки ("{" и "}") указывают на то, что необходимо выбрать один из элементов, разделенных вертикальной чертой ("|").

Например, спецификации

bork [-x] {on|off} *имя_файла*

соответствует любая из следующих команд:

```
bork on /etc/passwd
bork -x off /etc/passwd /etc/smartd.conf
bork off /usr/lib/tmac
```

В выражениях с шаблонами поиска используются следующие метасимволы:

- звездочка (*) обозначает нуль или более символов;
- знак вопроса (?) обозначает один символ;
- тильда (~) обозначает начальный каталог текущего пользователя;
- выражение ~*пользователь* обозначает начальный каталог указанного пользователя.

Например, иногда мы обозначаем каталоги, где хранятся сценарии запуска (`etc/rc*.d`, `etc/rc*.d` и т.д.), сокращенным шаблоном `etc/rc*.d`.

1.6. Единицы измерения

Метрические приставки кило-, мега- и гига- определяются показателями степени числа 10 (например, один мегагерц составляет миллион герц). Однако для компьютерных типов данных используются степени числа 2. Например, один "мегабайт" памяти составляет 220, или 1 048 576 байт. Ассоциация твердотельных технологий (Solid State Technology Association) Объединенного инженерного совета по электронным устройствам (Joint Electronic Device Engineering Council — JEDEC) даже превратила эти единицы измерения в официальный стандарт Standard 100B.01, который признает их степенями двойки (не без некоторых натяжек).

Пытаясь внести ясность, Международная электротехническая комиссия (International Electrotechnical Commission — IEC) определила ряд числовых приставок, основанных именно на степенях числа 2: киби- (kibi-), меби- (mebi-), гиби- (gibi-) и так далее с аббревиатурами КиБ (Ki), МиБ (Mi) и ГиБ (Gi) соответственно. Эти единицы измерения исключают двусмысленность, но их еще только начинают использовать. Поэтому привычный набор кило-, мега- и гига-префиксов все еще в ходу в обоих смыслах: десятичном и двоичном.

Определить конкретное значение можно по контексту. Объем оперативной памяти всегда измеряется в степенях двойки, а пропускная способность сети — в степенях числа 10. Размер дисков их производители обычно указывают в десятичных единицах, а размер секторов и страниц — в двоичных.

В этой книге мы используем единицы измерения МЭК (IEC) для степеней двойки и метрические — для степеней числа 10. Метрические единицы измерения мы применяем также для приблизительных значений и в тех случаях, когда точное основание степени неясно, не зафиксировано в документах или его невозможно определить. В командах файлов конфигурации `output` и `in` мы оставляем исходные значения и указатели единиц измерений. Для обозначения бита служит буква *b*, а для байта — буква *B*. Некоторые примеры интерпретации единиц измерения приведены в табл. 1.2.

Таблица 1.2. Примеры интерпретации единиц измерения

Пример	Описание возможного варианта
1 Кбайт (kB)	Размер файла, равный 1000 байт
4 КиБ (KiB)	Общий размер страниц твердотельного диска (SSD) 4096 байт
8 Кбайт (KB)	Размер памяти — не используется в этой книге (см. описание ниже)
100 Мбайт (MB)	Номинально составляет 10^8 байт (неоднозначность, понимается по контексту)
100 Мбайт (MB)	Номинально составляет 10^8 байт, в зависимости от контекста, возможно, 99 999 744 байт[a]
1 ГиБ (GiB)	1 073 741 824 байт памяти
1 Гбит/с (Gb/s)	Скорость передачи информации в сети, равная 1 000 000 000 бит в секунду
6 Тбайт (TB)	Объем жесткого диска, равный 6 000 000 000 000 байт

[a] Число 10^8 округлено в меньшую сторону до ближайшего целого, кратного размеру 512-байтового сектора диска.

Буква K в аббревиатуре, как в случае "8 Кбайт (KB)", не является частью стандарта. Это компьютерная адаптация метрической аббревиатуры k (обозначение приставки

кило-), которая означает 1 024 в отличие от 1 000. Поскольку в аббревиатурах для многих метрических приставок уже используется прописная буква, стала возникать путаница при новом и исходном использовании буквы K в качестве множителя 1 000.

В большинстве стран эта проблема не считается важной, например в США метрические префиксы часто используются неоднозначно. В дистрибутиве Ubuntu была предпринята попытка реализовать последовательную политику использования единиц измерений, но широкой поддержки, даже в самой компании Canonical, она, похоже, не получила (подробнее см. `wiki.ubuntu.com/UnitsPolicy`).

1.7. Man-страницы и другая онлайн-документация

Традиционную онлайн-документацию образуют страницы справочного руководства, обычно называемые *man-страницами*, потому что они считываются с помощью команды **man**. (Конечно, в наши дни вся документация в той или иной форме находится в режиме онлайн.) Справочные страницы поставляются вместе с новыми пакетами программного обеспечения. Даже в эпоху Google мы продолжаем рассматривать справочные страницы как авторитетный ресурс, потому что они доступны из командной строки и, как правило, содержат полную информацию о параметрах программы, а также демонстрируют полезные примеры и связанные с ними команды.

Справочные страницы — это краткое описание отдельных команд, драйверов, форматов файлов или библиотечных процедур. Они не затрагивают более общие темы, такие как "Как установить новое устройство?" или "Почему эта система так медленно работает?"

Организация man-страниц

Системы FreeBSD и Linux разделяют `man`-страницы на разделы. Их базовая схема показана в табл. 1.3. Другие варианты UNIX иногда определяют разделы несколько иначе.

Точная структура разделов не важна для большинства тем, потому что человек находит соответствующую страницу там, где она хранится. Просто учитывайте определения разделов, если тема с тем же именем появляется в нескольких разделах. Например, **passwd** — это и команда, и конфигурационный файл, поэтому существуют соответствующие записи как в разделе 1, так и в разделе 5.

Таблица 1.3. Разделы man-страниц

Раздел	Содержание
1	Пользовательские команды и приложения
2	Системные вызовы и коды ошибок ядра
3	Библиотечные вызовы
4	Драйверы устройств и сетевые протоколы
5	Стандартные форматы файлов
6	Игры и демонстрации
7	Различные файлы и документы
8	Команды администрирования системы
9	Спецификации ядра и интерфейсы

Команда man: чтение страниц интерактивного руководства

Команда man *заголовок* форматирует конкретную страницу интерактивного руководства и выводит ее на терминал пользователя с помощью утилит more, less или другой программы постраничной разбивки, которая задана в переменной окружения PAGER. Аргумент *заголовок* — это, как правило, имя команды, устройства или файла, о которых необходимо получить справочную информацию. Поиск по разделам руководства осуществляется в порядке возрастания номеров, хотя разделы, посвященные описанию команд (1 и 8), обычно просматриваются в первую очередь.

🕮 Информацию о переменных окружения см. в разделе 7.2.

Команда man *раздел заголовок* вызывает справочную страницу из указанного раздела. Так, в большинстве систем команда man **sync** вызывает справочную страницу для команды sync, а команда **man** 2 sync — для системного вызова sync.

Команда man -k *ключевое_слово* или apropos *ключевое_слово* выводит список справочных страниц, в строке пояснений к которым имеется указанное ключевое слово.

```
$ man -k translate
objcopy (1)          - copy and translate object files
dcgettext (3)        - translate message
tr (1)               - translate or delete characters
snmptranslate (1)    - translate SNMP OID values into more useful information
tr (1p)              - translate characters
...
```

База данных ключевых слов может устаревать. При добавлении новых справочных страниц к системе вам, возможно, придется перестроить этот файл с помощью команд makewhatis (Red Hat и FreeBSD) или mandb (Ubuntu).

Хранение страниц интерактивного руководства

Неформатированная информация для справочных страниц (входные данные команды nroff) обычно хранится в подкаталогах каталога /usr/share/man. В целях экономии места на диске системы Linux сжимают страницы с помощью утилиты gzip. Команда man может очень быстро разархивировать их.

Команда man поддерживает кеш отформатированных страниц в каталогах /var/cache/man или /usr/share/man, если соответствующие каталоги доступны для записи, но эти операции рискованны с точки зрения безопасности. В большинстве систем предварительное форматирование справочных страниц выполняется однократно во время инсталляции (см. команду catman) или не выполняется совсем.

Команда man ищет страницы в ряде каталогов. В Linux-системах выяснить путь поиска позволяет команда manpath. Результат ее работы (в системе Ubuntu) обычно таков.

```
ubuntu$ manpath
/usr/local/man:/usr/local/share/man:/usr/share/man
```

Эта установка хранится в переменной среды MANPATH, и в случае необходимости ее можно изменить.

```
$ export MANPATH=/home/share/localman:/usr/share/man
```

Некоторые системы позволяют установить общесистемный параметр пути поиска для справочных страниц, который может оказаться полезным в случае, если вам придется поддерживать параллельное дерево справочных страниц, например сгенерированных кроссплатформенным менеджером пакетов OpenPKG. Если же вы хотите распростра-

нять локальную документацию в виде справочных страниц, проще всего использовать стандартный системный механизм пакетирования и выложить справочные страницы в стандартные справочные каталоги. Подробнее об этом написано в главе 6.

1.8. Другая официальная документация

Справочные страницы — это лишь малая часть официальной документации. Остальная в основном рассеяна в веб-пространстве.

Руководства по конкретным системам

Одни производители систем ведут собственные онлайн-проекты по подготовке документации, другие выпускают полезные руководства в виде объемных книг. В настоящее время нужное руководство быстрее найти в Интернете, чем в форме печатного издания. Объем и качество документации бывают разными, но большинство производителей выпускает по меньшей мере руководство по администрированию и инсталляции системы. Где найти документацию по каждому из наших примеров систем, показано в табл. 1.4.

Несмотря на полезность этой документации, она не относится к тем книгам, которые читают перед сном (хотя отдельные поставщики пишут справочные руководства, которые могут усыпить кого угодно). Прежде чем обращаться к документации поставщиков, мы обычно ищем ответы в системе Google.

Таблица 1.4. Где найти документацию от производителей операционных систем

Система	URL	Описание
Debian	`debian.org/com`	Справочник для системного администратора, поставляемый вместе с текущей версией системы
Ububtu	`help.ubuntu.com`	Дружественная к пользователю; версии LTS описаны в разделе "server guide".
RHEL	`redhat.com/docs`	Исчерпывающая документация для системных администраторов
CentOS	`wiki.cebtos.org`	Советы, подсказки и ответы на часто задаваемые вопросы
FreeBSD	`freebsd.org/docs.html`	Информацию для системного администратора см. в *FreeBSD Handbook*

Документация по конкретным пакетам

Большинство важнейших программных пакетов в мире UNIX и Linux поддерживают отдельные лица или такие сторонние организации, как Internet Systems Consortium и Apache Software Foundation. Сотрудники этих групп пишут собственную документацию, качество которой варьируется от плохого до замечательного. При этом существуют и такие прекрасно выполненные документы, как Pro Git от `git-scm./book`, которые вполне оправдывают ожидания.

В число дополнительных документов входят: технические отчеты, проектные обоснования и трактовки конкретных тем. Эти дополнительные материалы зачастую не ограничиваются простым описанием команд и могут включать самоучители и другие разделы. Многие программные пакеты, помимо справочных страниц, содержат и статьи по соответствующим темам. Например, после вызова справочной страницы для редактора **vim**

вам будет предложено не только ознакомиться с аргументами командной строки, но и перейти к разделу, из которого вы узнаете, как в действительности отредактировать файл.

Для большинства проектов по разработке программного обеспечения создаются списки рассылки для пользователей и разработчиков, а также каналы IRC для интернет-чата. Если у вас возник вопрос о конкретной конфигурации или вы обнаружили ошибку, советуем в первую очередь обращаться туда.

Книги

К самым лучшим источникам информации для системных администраторов в книжном мире можно отнести серию книг издательства O'Reilly. Начало этой серии было положено книгой *UNIX in a Nutshell*. Теперь же практически всем важным подсистемам и командам UNIX и Linux посвящены ее отдельные тома. Издательство O'Reilly публикует также книги о сетевых протоколах, операционной системе Microsoft Windows и другим темам, не связанным с UNIX. Все эти книги имеют приемлемую цену, своевременны и ориентированы на конкретную аудиторию.

Многие читатели издательства O'Reilly используют интернет-магазин Safari Books Online — службу подписки, предлагающую доступ к электронным книгам, видео и другим учебным материалам. Наряду с книгами издательства O'Reilly в этом магазине можно найти многочисленные книги других издателей.

Документы RFC

Документы из серии RFC (Request for Comments — запрос на комментарии) описывают протоколы и процедуры, используемые в Интернете. Большинство из этих документов достаточно детализированы и специализированы, но некоторые написаны в виде обзоров. Информации, которую они содержат, вполне можно доверять, но некоторые из них являются всего лишь обзорами. Фраза "эталонная реализация" означает, что программа "разработана надежным источником в соответствии со спецификациями RFC".

Авторитет RFC незыблем, а некоторых из документов этой серии достаточно полезные для системных администраторов. Подробное описание этих документов приведено в разделе 13.1. Мы будем часто цитировать их в нашей книге.

1.9. Другие источники информации

Источники, рассмотренные в предыдущем разделе, изучены экспертами и написаны авторитетными специалистами, но вряд ли это последнее слово в области управления системами UNIX и Linux. В Интернете доступны бесчисленные блоги, дискуссионные форумы и новостные ленты.

Однако, что ни говорите, но Google — лучший друг системного администратора. Если вы ищите детали определенной команды или формата файла, Google либо эквивалентная поисковая система должны быть первым ресурсом, который вы запрашиваете для любого вопроса системного администратора. Сделайте это привычкой; так вы избежите задержек и унижений, когда на ваши вопросы в интерактивном форуме отвечают с помощью ссылки на Google.[1] Если вы столкнулись *со сложным вопросом, ищите ответ в Интернете.*

[1] Или, что еще хуже, указывают ссылку на Google через сайт `lmgtfy.com`.

Сохранение актуальности

Операционные системы, инструменты и методы их поддержки быстро меняются. Советуем вам ежедневно просматривать сайты, перечисленные в табл. 1.5, чтобы быть в курсе тенденций, наблюдаемых в отрасли.

Таблица 1.5. Актуальные ресурсы

Веб-сайт	Описание
darkreading.com	Новости о средствах безопасности, тенденции и обсуждение
devopsreactions.tumblr.com	Юмор системных администраторов в анимированном виде
linux.com	Сайт консорциума Linux Foundation; форум полезен для новых пользователей
linuxfoundation.org	Некоммерческий проект компании OSS, работодателя Линуса Торвальдса (Linus Torvalds)
lwn.net	Высококачественные, своевременные статьи о Linux и OSS
lxer.com	Новости о Linux
securityfocus.com	Отчеты об уязвимостях и списки рассылки, связанные с безопасностью
@SwiftOnSecurity	Размышления от имени Тейлор Свифт (Taylor Swift) о безопасности информации (пародия)
@nixcraft	Твиты о системном администрировании UNIX и Linux
everythingsysadmin.com	Блог Томаса Лимончелли (Tomas Limonchelli), авторитетного системного администратора[a]
sysadvent.blogspot.com	Блог для системных администраторов, публикующий статьи каждый декабрь
oreilly.com/topics	Учебные ресурсы издательства O'Reilly по многим темам
schneier.com	Блог Брюса Шнайера (Bruce Schneier), эксперта по криптографии и безопасности

[a] См. также сборник Томаса Лимончелли *April Fools's RFC* на сайте rfc-humor.com

Социальные сети также полезны. Twitter и Reddit, в частности, имеют сильные сообщества с большим количеством предложений, хотя отношение сигнал-шум в этих сетях иногда может быть довольно плохим. На сайте Reddit присоединяйтесь к разделам основного форума sysadmin, linux, linuxadmin и netsec.

Практические руководства и справочные сайты

Сайты, перечисленные в табл. 1.6, содержат руководства, учебники и статьи о том, как выполнять определенные задачи в UNIX и Linux.

Таблица 1.6. Специальные форумы и справочные сайты

Веб-сайт	Описание
wiki.archlinux.org	Статьи и руководства по Arch Linux; многие из них носят более общий характер
askubuntu.com	Вопросы и ответы для пользователей и разработчиков Ubuntu
digitalocean.com	Учебники по многим темам OSS, разработки и системного администрирования[a]

Окончнаие табл. 1.6

Веб-сайт	Описание
`kernel.org`	Официальный сайт ядра Linux
`serverfault.com`	Совместно отредактированная база данных вопросов системного администрирования[6]
`serversforhackers.com`	Высококачественные видеоролики, форумы и статьи о системном администрировании

[a] См. `Digitalocean.com/community/tutorials`

[6] Также см. аналогичный сайт `stackoverflow.com`, посвященный программированию, но полезный и для системных администраторов

Сайты Stack Overflow и Server Fault, указанные в табл. 1.6 (оба входят в группу сайтов Stack Exchange), требуют более пристального взгляда. Если у вас возникла проблема, скорее всего, кто-то с ней уже сталкивался и попросил о помощи на одном из этих сайтов. Авторитетный формат вопросов и ответов, используемый сайтами Stack Exchange, хорошо подходит для тех проблем, с которыми сталкиваются системные администраторы и программисты. Рекомендуем создать учетную запись и присоединиться к этому большому сообществу.

Конференции

Отраслевые конференции — отличный способ наладить связь с другими профессионалами, следить за технологическими тенденциями, проходить учебные курсы, получать сертификаты и узнавать о последних услугах и продуктах. В последние годы резко возросло количество конференций, связанных с системным администрированием. Некоторые из наиболее известных конференций представлены в табл. 1.7.

Таблица 1.7. Конференции по системному администрированию

Конференция	Место	Время	Описание
LISA	Переменное	4 кв.	Управление инсталляцией крупных систем
Monitorama	Портленд	Июнь	Инструменты и методы мониторинга
OSCON	Переменное (США/ЕС)	2 или 3 кв.	Долгосрочная конференция O'Reilly OSS
SCALE	Пасадена	Январь	Southern California Linux Expo
DefCon	Лас-Вегас	Июль	Самый старый и самый большой съезд хакеров
Velocity	По всему миру	Переменное	Конференция O'Reilly по веб-операциям
BSDCan	Оттава	Май-июнь	Все о BSD для новичков и гуру
re:Invent	Лас-Вегас	4 кв.	AWS-конференция по облачным вычислениям в Лас-Вегасе
VMWorld	Переменное (США/ЕС)	3 или 4 кв.	Виртуализация и облачные вычисления
LinuxCon	По всему миру	Переменное	Будущее Linux
RSA	Сан-Франциско	1 кв. или 2 кв.	Промышленная криптография и безопасность информации
DevOpsDays	По всему миру	Переменное	Ряд тем для преодоления разрыва между командами разработки и групп по эксплуатации
QCon	По всему миру	Переменное	Конференция для разработчиков программного обеспечения

Meetups (`meetup.com`) — это еще один способ общения с единомышленниками. В большинстве городов США, как и во всем мире, есть группы пользователей Linux или DevOps, спонсирующие ораторов, дискуссии и хакерские семинары.

1.10. Способы поиска и установки программного обеспечения

Подготовка к работе программного обеспечения подробно рассматривается в главе 6. Но для самых нетерпеливых мы прямо здесь расскажем о том, как выяснить, что уже установлено в вашей системе, и как получить новое программное обеспечение и инсталлировать его.

В современных операционных системах программное обеспечение разделено на пакеты, которые можно инсталлировать независимо друг от друга. При стандартной установке системы используется группа "стартовых" пакетов, которую можно при необходимости расширить.

Добавочные программные продукты зачастую предоставляются также в виде предварительно скомпилированных пакетов, но далеко не во всех системах. Большая часть программного обеспечения создается независимыми группами разработчиков, выпускающими программы в виде исходных кодов. Репозитории пакетов затем берут исходные коды, компилируют их в соответствии с особенностями конкретной системы и включают в пакеты полученные бинарные файлы. Как правило, намного проще инсталлировать бинарную версию пакета, предназначенную для данной системы, чем искать и компилировать исходный код. Однако нужно учитывать, что иногда создатели пакетов на один-два выпуска отстают от текущей версии системы.

Если в двух системах используется один и тот же формат обработки пакетов, то это еще не означает, что пакеты обеих систем будут взаимозаменяемыми. Например, в системах Red Hat и SUSE применяется формат RPM, однако структура их файловых систем неодинакова. Рекомендуется всегда пользоваться пакетами, созданными для конкретной системы.

В рассматриваемых нами дистрибутивах предусмотрены отлаженные системы управления пакетами, которые включают средства доступа к репозиториям программных продуктов и поиска их в Интернете. Дистрибьюторы активно поддерживают эти репозитории от имени сообщества, которое они представляют, чтобы облегчить процесс исправления и обновления систем. Другими словами, жизнь прекрасна!

Администраторы без доступа к предварительно скомпонованным бинарным версиям пакетов должны инсталлировать системы по-старому, т.е. загружать архив исходного кода **`tar`**, а затем вручную конфигурировать, компилировать и инсталлировать систему. Иногда этот процесс проходит гладко, а иногда может превратиться в кошмар: все зависит от используемого программного обеспечения и конкретной операционной системы.

В этой книге мы предполагаем, что дополнительное программное обеспечение уже установлено, и не собираемся мучить вас шаблонными инструкциями по инсталляции каждого пакета. При необходимости мы будем указывать точные названия пакетов для выполнения конкретного проекта. Однако большей частью мы не будем повторять инструкции по инсталляции, поскольку они практически идентичны для всех пакетов.

Как определить, установлено ли программное обеспечение

По целому ряду причин может быть сложно определить, какой пакет содержит программное обеспечение, которое вам действительно нужно. Вместо того чтобы начинать с уровня пакета, проще использовать команду оболочки и выяснить, находится ли со-

ответствующий бинарный файл в вашем пути поиска. Например, следующая команда показывает, что компилятор GNU C уже установлен на этом компьютере:

```
ubuntu$ which gcc
/usr/bin/gcc
```

Если вы не можете найти нужную команду, попробуйте выполнить команду **whereis**; она ищет более широкий набор системных каталогов и не зависит от пути поиска вашей оболочки.

Другой альтернативой является невероятно полезная команда **locate**, которая справляется с предварительно скомпилированным индексом файловой системы, чтобы найти имена файлов, которые соответствуют определенному шаблону.

Система FreeBSD реализует команду **locate** как часть базовой системы. В системе Linux текущая реализация команды **locate** находится в пакете **mlocate**. В системах Red Hat и CentOS установите пакет **mlocate** с помощью команды **yum**; см. раздел 6.4.

Команда **locate** может найти любой тип файла, но это не относится к командам или пакетам. Например, если вы не были уверены, где найти подключаемый файл **signal.h**, вы можете попробовать выполнить следующую команду.

```
freebsd$ locate signal.h
/usr/include/machine/signal.h
/usr/include/signal.h
/usr/include/sys/signal.h
...
```

База данных команды **locate** периодически обновляется командой **updatedb** (в системт FreeBSD — командой **locate.updatedb**), которая запускается из программы **cron**. Поэтому результаты поиска не всегда отражают недавние изменения файловой системы.

Зная имя пакета, который ищете, вы также можете использовать утилиты для упаковки системы, чтобы напрямую проверить наличие пакета. Например, в системе Red Hat следующая команда проверяет наличие (и установленную версию) интерпретатора Python:

```
redhat$ rpm -q python
python-2.7.5-18.el7_1.1.x86_64
```

🕮 Дополнительную информацию об управлении пакетами см. в главе 6.

Вы также можете узнать, к какому пакету принадлежит определенный файл:

```
redhat$ rpm -qf /etc/httpd
HTTPD-2.4.6-31.el7.centos.x86_64

freebsd$ pkg which /usr/local/sbin/httpd
/usr/local/sbin/httpd was installed by package apache24-2.4.12

ubuntu$ dpkg-query -S /etc/apache2
apache2: /etc/apache2
```

Добавление нового программного обеспечения

Если вам необходимо установить дополнительное программное обеспечение, сначала следует определить каноническое имя соответствующего пакета программного обеспечения. Например, вам нужно будет перевести фразу "Я хочу установить **locate**" в "Мне нужно установить пакет **mlocate**" или перевести "Мне нужен справочник **named**" в "Мне нужно установить BIND". В этом может помочь целый ряд системных индек-

сов в Интернете, но Google, как правило, так же эффективен. Например, поиск "locate command" приводит вас непосредственно к нескольким соответствующим обсуждениям.

В следующих примерах показана инсталляция команды **tcpdump** для каждой из наших иллюстративных систем. Команда **tcpdump** — это инструмент для захвата пакетов, позволяющий просматривать исходные пакеты, отправляемые в систему и из системы в сети.



В системах Debian и Ubuntu используется программа APT — Debian Advanced Package Tool.

```
ubuntu# sudo apt-get install tcpdump
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  tcpdump
0 upgraded, 1 newly installed, 0 to remove and 81 not upgraded.
Need to get 0 B/360 kB of archives.
After this operation, 1,179 kB of additional disk space will be used.
Selecting previously unselected package tcpdump.
(Reading database ... 63846 files and directories currently installed.)
Preparing to unpack .../tcpdump_4.6.2-4ubuntu1_amd64.deb ...
Unpacking tcpdump (4.6.2-4ubuntu1) ...
Processing triggers for man-db (2.7.0.2-5) ...
Setting up tcpdump (4.6.2-4ubuntu1) ...
```



В системах Red Hat и CentOS аналогичная версия выглядит следующим образом.

```
redhat# sudo yum install tcpdump
Loaded plugins: fastestmirror
Determining fastest mirrors
 * base: mirrors.xmission.com
 * epel: linux.mirrors.es.net
 * extras: centos.arvixe.com
 * updates: repos.lax.quadranet.com
Resolving Dependencies
--> Running transaction check
---> Package tcpdump.x86_64 14:4.5.1-2.el7 will be installed
--> Finished Dependency Resolution
tcpdump-4.5.1-2.el7.x86_64.rpm | 387 kB 00:00
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
  Installing : 14:tcpdump-4.5.1-2.el7.x86_64 1/1
  Verifying : 14:tcpdump-4.5.1-2.el7.x86_64 1/1
Installed:
  tcpdump.x86_64 14:4.5.1-2.el7
Complete!
```



Менеджер пакетов в системе FreeBSD называется **pkg**.

```
freebsd# sudo pkg install -y tcpdump
Updating FreeBSD repository catalogue...
Fetching meta.txz:            100%    944 B    0.9kB/s    00:01
```

```
Fetching packagesite.txz:     100%    5 MiB    5.5MB/s    00:01
Processing entries: 100%
FreeBSD repository update completed. 24632 packages processed.
All repositories are up-to-date.
The following 2 package(s) will be affected (of 0 checked):

New packages to be INSTALLED:
    tcpdump: 4.7.4
    libsmi: 0.4.8_1

The process will require 17 MiB more space.
2 MiB to be downloaded.
Fetching tcpdump-4.7.4.txz:   100%    301 KiB  307.7kB/s  00:01
Fetching libsmi-0.4.8_1.txz:  100%    2   MiB    2.0MB/s  00:01
Checking integrity... done (0 conflicting)
[1/2] Installing libsmi-0.4.8_1...
[1/2] Extracting libsmi-0.4.8_1: 100%
[2/2] Installing tcpdump-4.7.4...
[2/2] Extracting tcpdump-4.7.4: 100%
```

Создание программного обеспечения из исходного кода

В качестве иллюстрации рассмотрим процесс создания версии **tcpdump** из исходного кода.

Первая задача — найти сам код. Сторонники программного обеспечения иногда хранят индекс выпусков на веб-сайте проекта, которые можно загрузить в виде архивов. Для проектов с открытым исходным кодом вы, скорее всего, найдете код в репозитории Git.

Источник **tcpdump** хранится в репозитории GitHub. Выполните клонирование репозитория в каталоге **/tmp**, создайте ветку с тегами, которую вы хотите создать, затем распакуйте, настройте, создайте и установите:

```
redhat$ cd /tmp
redhat$ git clone https://github.com/the-tcpdump-group/tcpdump.git
<status messages as repository is cloned>
redhat$ cd tcpdump
redhat$ git checkout tags/tcpdump-4.7.4 -b tcpdump-4.7.4
Switched to a new branch 'tcpdump-4.7.4'
redhat$ ./configure
checking build system type... x86_64-unknown-linux-gnu
checking host system type... x86_64-unknown-linux-gnu
checking for gcc... gcc
checking whether the C compiler works... yes
...
redhat$ make
<several pages of compilation output>
redhat$ sudo make install
<files are moved in to place>
```

Эта последовательность **configure/make/make** устанавливается для большинства программ, написанных на языке C, и работает во всех системах UNIX и Linux. Всегда полезно проверить файл **INSTALL** или **README** пакета для уточнения. Должна быть установлена среда разработки и любые необходимые для пакета требования. (В случае пакета **tcpdump** необходимой предпосылкой является установка библиотеки **libpcap** и сопутствующих библиотек.)

Зачастую возникает необходимость настроить конфигурацию сборки, поэтому используйте команду `./configure --help`, чтобы просмотреть параметры, доступные для каждого конкретного пакета. Другим полезным параметром команды `configure` является `-prefix = directory`, который позволяет скомпилировать программное обеспечение для установки где-то, кроме `/usr/local`, которое обычно является значением по умолчанию.

Установка с помощью веб-сценария

Кросс-платформенные пакеты программного обеспечения все чаще предлагают ускоренный процесс установки, который управляется сценарием оболочки, загружаемым из Интернета с помощью команд `curl`, `fetch` или `wget`.[2] Например, чтобы настроить машину как клиент Salt, вы можете выполнить следующие команды:

```
$ curl -o /tmp/saltboot -sL https://bootstrap.saltstack.com
$ sudo sh /tmp/saltboot
```

Сценарий `bootstrap` исследует локальную среду, затем загружает, инсталлирует и настраивает соответствующую версию программного обеспечения. Этот тип инсталляции особенно распространен в тех случаях, когда сам процесс сложен, но поставщик очень мотивирован, чтобы облегчить пользователям работу. (Еще один хороший пример — RVM, см. раздел 7.7.)

🕮 Дальнейшую информацию об инсталляции пакетов см. в главе 6.

Этот метод установки отлично работает, но он поднимает несколько проблем, которые стоит упомянуть. Во-первых, он не оставляет должной записи об инсталляции для будущих ссылок. Если ваша операционная система предлагает пакетную версию программного обеспечения, обычно лучше установить пакет вместо запуска веб-инсталлятора. Пакеты легко отслеживаются, обновляются и удаляются. (С другой стороны, большинство пакетов на уровне операционных систем устарели. Вероятно, вы не получите самую последнюю версию программного обеспечения.)

🕮 Более подробную информацию о цепочке доверия HTTPS см. в разделе 27.6.

Будьте очень подозрительными, если URL-адрес загрузочного сценария небезопасен (т.е. он не начинается с префикса `https:`). Незащищенный HTTP является легкоуязвимым для захвата, а URL-адреса инсталляции представляют особый интерес для хакеров, потому что они знают, что вы, скорее всего, будете работать в качестве привилегированного пользователя, независимо от того, какой код возвращается. В отличие от этого, протокол HTTPS проверяет подлинность сервера с помощью криптографической цепочки доверия. Это не вполне, но достаточно надежный способ.

Некоторые продавцы публикуют URL-адрес инсталляции по протоколу HTTP, который автоматически перенаправляет пользователя на HTTPS-версию. Это глупо и на самом деле не более безопасно, чем прямой адрес HTTP. Ничто не мешает перехвату исходного HTTP-обмена, поэтому вы, возможно, никогда не достигнете перенаправления поставщика. Однако наличие таких переадресаций означает, что стоит попробовать собственную замену `https` для `http` в небезопасных URL-адресах. Чаще всего это работает отлично.

Оболочка принимает текст сценария как свой стандартный вход, и эта функция позволяет использовать процедуры инсталляции в режиме онлайн, например:

```
$ curl -L https://badvendor.com | sudo sh
```

[2]Это все простые HTTP-клиенты, загружающие содержимое URL-адреса в локальный файл или (необязательно) распечатывающие содержимое в виде результата своей работы.

Тем не менее существует потенциальная проблема, связанная с этой конструкцией, — корневая оболочка все еще работает, даже если команда `curl` выводит частичный сценарий, а затем терпит неудачу — скажем, из-за кратковременного сбоя сети. Конечный результат непредсказуем и потенциально не очень хорош.

Нам неизвестны какие-либо документированные случаи проблем, связанных с этой причиной. Тем не менее это правдоподобный режим отказа. Более того, направление результатов работы команды `curl` в оболочку в среде системных администраторов рассматривается как типичный промах новичков, поэтому, если вы это сделаете, никому об этом не говорите.

Исправить ситуацию легко: просто сохраните сценарий во временном файле, а затем запустите сценарий на отдельном этапе после успешного завершения загрузки.

1.11. Где разместить программное обеспечение

Операционные системы и программное обеспечение могут размещаться в частных центрах обработки данных, на объектах совместного размещения, на облачной платформе или в некоторой их комбинации. Наиболее быстро растущие начинающие компании выбирают облако. У солидных предприятий, вероятно, есть центры обработки данных, поэтому они могут организовать частное облако внутри организации.

Самый практичный выбор и наша рекомендация для новых проектов — это публичные облачные провайдеры. Они предоставляют множество преимуществ по сравнению с центрами обработки данных.

- Отсутствие капитальных затрат и низкие начальные эксплуатационные расходы.
- Отсутствие необходимости устанавливать, защищать и управлять оборудованием.
- Регулировка емкости, пропускной способности и вычислительной мощности по требованию.
- Готовые решения для общих вспомогательных потребностей, таких как базы данных, балансировки нагрузки, очереди, мониторинг и т.д.
- Более дешевая и простая реализация высокодоступных или избыточных систем.

Ранние облачные системы имели репутацию слабо защищенных и низкопроизводительных, но это уже не является серьезной проблемой. В наши дни большая часть административной работы выполняется в облаке (см. главу 9 для общего введения в тему).

Наша предпочтительная облачная платформа является лидером в своей области: Amazon Web Services (AWS). Gartner, ведущая исследовательская фирма по технологиям, обнаружила, что AWS в десять раз превосходит всех конкурентов. AWS быстро внедряет инновации и предлагает гораздо более широкий спектр услуг, чем любой другой поставщик. Она также имеет отличную репутацию по обслуживанию клиентов и поддерживает большое сообщество. На первых порах AWS предлагает бесплатный уровень обслуживания, включая использование в течение года маломощного облачного сервера.

Облачная платформа Google (GCP) активно совершенствует и продает свои продукты. Некоторые утверждают, что эта технология не имеет себе равных по сравнению с другими поставщиками. Рост GCP был медленным, отчасти благодаря репутации компании Google, отказавшейся поддерживать несколько популярных служб. Тем не менее его удобные для клиента условия ценообразования и уникальные функции являются привлекательными характеристиками.

DigitalOcean — это более простая служба, целью которой является высокая производительность. Она ориентирована на разработчиков, которым DigitalOcean предла-

гает ясный интерфейс прикладного программирования, низкую цену и чрезвычайно быструю загрузку. Разработчики DigitalOcean — ярые сторонники программного обеспечения с открытым исходным кодом, а их учебники и руководства по популярным интернет-технологиям являются одними из лучших.

1.12. Специализация и смежные дисциплины

Системные администраторы работают не в вакууме; создавать и поддерживать сложную сеть должна команда экспертов. В этом разделе описываются некоторые роли, которые системные администраторы могут выполнять в соответствии со своими навыками и возможностями. Некоторые администраторы предпочитают специализироваться в одной или нескольких из этих областей.

Ваша цель как системного администратора или как профессионала, работающего в любой из этих смежных областей, — достижение целей организации. Не позволяйте политике или иерархии мешать прогрессу. Лучшие администраторы решают проблемы и свободно обмениваются информацией с другими.

Методология DevOps

🕮 Дополнительную информацию о методологии DevOps см. в главе 31.

DevOps — это не столько специфическая методология, сколько культура или философия работы. Ее цель — повышение эффективности создания и поставки программного обеспечения, особенно в крупных организациях, в которых существует множество взаимосвязанных служб и команд. Организации, внедрившие методику DevOps, стимулируют интеграцию между инженерными командами и могут не делать различия между разработкой и эксплуатацией или не придавать им значения. Эксперты, работающие в этой области, ищут неэффективные процессы и заменяют их небольшими сценариями оболочки или большими и громоздкими репозиториями Chef.

Инженеры по надежности сайтов

Инженеры по надежности сайтов ставят на первое место время безотказной и правильной работы. Мониторинг сети, развертывание программного обеспечения на производстве, координация работ, планирование будущего расширения и аварийные отключения — все это лежит в основе этих крестоносцев доступности. Головной болью инженеров по надежности сайта является единственная точка отказа.

Инженеры по безопасности

Инженеры по безопасности сосредоточены на практической, повседневной стороне информационной безопасности. Эти люди устанавливают и используют инструменты для поиска уязвимых мест и мониторинга для предотвращения сетевых атак. Они также участвуют в имитации атак, чтобы оценить эффективность методов их профилактики и обнаружения.

Сетевые администраторы

Сетевые администраторы проектируют, устанавливают, настраивают и управляют сетями. Организации, в которых функционируют центры обработки данных, скорее всего, будут использовать сетевых администраторов; это объясняется тем, что такие объекты

имеют множество физических переключателей, маршрутизаторов, брандмауэров и других устройств, которым требуется управление. Облачные платформы также предлагают множество сетевых опций, но обычно они не требуют специального администратора, поскольку большая часть работы выполняется поставщиком.

Администраторы баз данных

Администраторы баз данных являются экспертами по установке и управлению программным обеспечением баз данных. Они управляют схемами базы данных, выполняют установки и обновления, настраивают кластеризацию, выбирают параметры для оптимальной производительности и помогают пользователям формулировать эффективные запросы. Администраторы баз данных обычно владеют одним или несколькими языками запросов и имеют опыт работы с реляционными и нереляционными (NoSQL) базами данных.

Инженеры центра сетевых операций

Инженеры центра сетевых операций контролируют состояние крупных объектов в режиме реального времени и отслеживают инциденты и сбои. Они принимают запросы на устранение ошибок от пользователей, выполняют обычные обновления и координируют действия других команд. Их чаще всего можно найти наблюдающими за панелью мониторов, на которых демонстрируются графики и измерения.

Технические специалисты центров обработки данных

Технические специалисты центров обработки данных используют аппаратные средства. Они получают новое оборудование, ведут инвентарный учет, следят за жизненным циклом оборудования, устанавливают серверы в стойки, управляют кабелями, поддерживают энергопитание и кондиционирование воздуха, а также выполняют ежедневные операции по эксплуатации центра обработки данных. Системному администратору желательно дружить с техническими специалистами центров обработки данных, подпаивая их кофе, энергетическими напитками и алкоголем.

Архитекторы

Системные архитекторы обладают глубокими знаниями в более чем одной области. Они используют свой опыт для разработки распределенных систем. Их должностные инструкции могут предусматривать определение зон безопасности и сегментацию, устранение отдельных точек отказа, планирование будущего роста, обеспечение взаимодействия между несколькими сетями и третьими сторонами, а также принятие решений, касающихся всей организации. Хорошие архитекторы технически компетентны и обычно предпочитают внедрять и тестировать свои собственные проекты.

1.13. Литература

- Abbott, Martin L., and Michael T. Fisher. *The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise (2nd Edition).* Addison-Wesley Professional, 2015.
- Gancarz, Mike. *Linux and the Unix Philosophy.* Boston: Digital Press, 2003.
- Limoncelli, Thomas A., and Peter Salus. *The Complete April Fools' Day RFCs.*

- *Peer-to-Peer Communications LLC*. 2007. Юмор инженеров. Эту коллекцию историй можно свободно прочитать на сайте `rfc-humor.com`.
- Raymond, Eric S. *The Cathedral & The Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. Sebastopol, CA: O'Reilly Media, 2001.
- Salus, Peter H. The Daemon, the GNU & the Penguin: *How Free and Open Software is Changing the World. Reed Media Services*, 2008. Это увлекательная история борьбы за открытый исходный код, написанная самым известным историком UNIX, также доступна на сайте `groklaw.com` под лицензией Creative Commons. Адрес URL для самой книги довольно длинный; найдите текущую ссылку на сайте `groklaw.com` или попробуйте этот сжатый эквивалент: `tinyurl.com/d6u7j`).
- Siever, Ellen, Stephen Figgins, Robert Love, and Arnold Robbins. *Linux in a Nutshell (6th Edition)*. Sebastopol, CA: O'Reilly Media, 2009.

Системное администрирование и методология DevOps

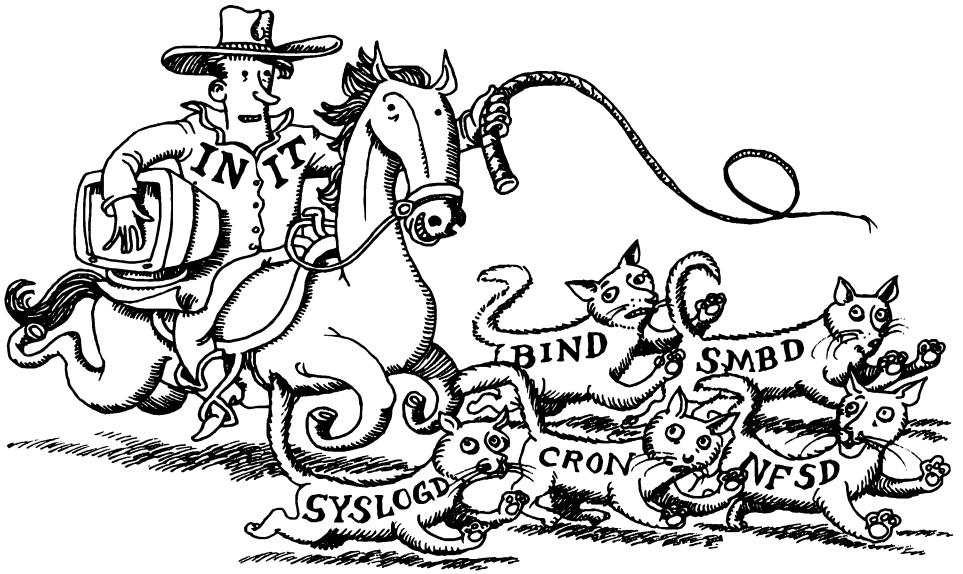
- Kim, Gene, Kevin Behr, and George Spafford. *The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win. Portland*, OR: IT Revolution Press, 2014. Введение в философию и принципы создания современной IT-организации, написанная в повествовательном стиле. Непреходящая классика.
- Kim, Gene, Jez Humble, Patrick Debois, and John Willis. *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations. Portland*, OR: IT Revolution Press, 2016.
- Limoncelli, Thomas A., Christina J. Hogan, and Strata R. Chalup. *The Practice of System and Network Administration (2nd Edition)*. Reading, MA: Addison-Wesley, 2008. Это хорошая книга, в которой очень подробно описаны организационные и процедурные аспекты системного администрирования Авторы ведут блог, посвященный системному администрированию: `everythingsysadmin.com`.
- Limoncelli, Thomas A., Christina J. Hogan, and Strata R. Chalup. *The Practice of Cloud System Administration*. Reading, MA: Addison-Wesley, 2014. Книга предыдущих авторов, посвященная распределенным системам и облачным вычислениям.

Важные инструменты

- Blum, Richard, and Christine Bresnahan. *Linux Command Line and Shell Scripting Bible (3rd Edition)*. Wiley, 2015.
- Dougherty, Dale, and Arnold Robins. *Sed & Awk (2nd Edition)*. Sebastopol, CA: O'Reilly Media, 1997. Классическая книга издательства O'Reilly о мощных и широко распространенных текстовых процессорах **sed** и **awk**.
- Kim, Peter. The Hacker *Playbook 2: Practical Guide To Penetration Testing*. CreateSpace Independent Publishing Platform, 2015.
- Neil, Drew. *Practical Vim: Edit Text at the Speed of Thought*. Pragmatic Bookshelf, 2012.
- Shotts, William E. *The Linux Command Line: A Complete Introduction*. San Francisco, CA: No Starch Press, 2012.
- Sweigart, Al. *Automate the Boring Stuff with Python: Practical Programming for Total Beginners*. San Francisco, CA: No Starch Press, 2015.

Глава 2

Загрузка и системные демоны



"Загрузка" (booting) — это стандартный термин, означающий запуск компьютера. Он представляет собой сокращенную форму слова "самозагрузка" (bootstrapping), которое подразумевает, что компьютер должен "привести себя в рабочее состояние с помощью собственных программ начальной загрузки (bootstraps)".

Процесс загрузки состоит из нескольких задач, определенных в широком смысле.

- Поиск, ввод в память и запуск кода начальной загрузки.
- Поиск, ввод в память и запуск ядра операционной системы.
- Активирование сценариев запуска и системных демонов.
- Поддержание порядка управление переходами системы из одного состояния в другое.

Действия, указанные в последнем пункте, продолжаются до тех пор, пока система остается в рабочем состоянии, поэтому граница между загрузкой и нормальной работой по своей сути немного размыта.

2.1. Обзор процесса загрузки

Процедуры запуска в последние годы сильно изменились. Появление современных BIOS с поддержкой UEFI (Unified Extensible Firmware Interface — унифицированный интерфейс расширяемой прошивки) упростило ранние этапы загрузки, по крайней мере с концептуальной точки зрения. На более поздних этапах большинство дистрибу-

тивов Linux теперь используют демон системного менеджера под названием **systemd** вместо традиционного демона UNIX **init**. Помимо всего прочего, менеджер **systemd** упрощает процесс загрузки путем добавления управления зависимостями, поддержки одновременных процессов запуска и комплексного подхода к протоколированию.

Управление загрузкой также изменялось по мере того, как системы мигрировали в облако. Дрейф в сторону виртуализации, облачных экземпляров и контейнеризации уменьшил потребность в том, чтобы администраторы касались физического оборудования. Вместо этого у нас теперь есть управление изображениями, интерфейсы API и панели управления.

Во время начальной загрузки ядро загружается в память и начинает выполняться. При этом выполняется множество задач инициализации, и затем система становится доступной для пользователей. Общий обзор этого процесса показан на рис. 2.1.



Рис. 2.1. Процессы загрузки Linux и Unix

Администраторы не имеют средств для прямого интерактивного управления большинством действий, необходимых для загрузки системы. Вместо этого администраторы могут изменять конфигурации загрузчиков, редактируя файлы конфигурации для сценариев запуска системы или изменяя аргументы, которые загрузчик передает ядру.

Прежде чем система будет полностью загружена, должны быть проверены и смонтированы файловые системы и запущены системные демоны. Этими процедурами управляет набор сценариев оболочки (иногда называемых "сценариями **init**") либо файлы, которые запускаются последовательно с помощью демона **init** или анализируются менеджером **systemd**. Точная компоновка сценариев запуска и способ их выполнения зависит от операционной системы. Подробности изложены в этой главе ниже.

2.2. Системные прошивки

При включении компьютера процессор выполняет загрузочный код, хранящийся в постоянном запоминающем устройстве. В виртуальных системах это постоянное запоминающее устройство может быть виртуальным, но концепция остается прежней.

Прошивка системы, как правило, знает обо всех устройствах, которые подключены к материнской плате, таких как контроллеры SATA, сетевые интерфейсы, USB-контроллеры и датчики для питания и температуры[1]. Помимо возможности аппаратной настройки этих устройств, прошивка позволяет вам либо сообщить о них операционной системе, либо отключить и скрыть их.

На физическом (в отличие от виртуального) оборудовании большинство прошивок предлагает пользовательский интерфейс. Тем не менее оно, как правило, слишком простое и неудобное. Для работы с ним необходимо иметь доступ к компьютеру и консоли, а также сразу же нажать на конкретную клавишу после включения системы. Ксожалению, "золотой ключик" у каждого производителя свой; проверьте, сможете ли вы разобраться в загадочной строке инструкций в тот момент, когда система включится впервые[2]. Если не сможете, попробуйте клавиши <Delete>, <Ctrl>, <F6>, <F8>, <F10> или <F11>. Для увеличения шансов на успех коснитесь клавиши несколько раз, затем удерживайте ее нажатой.

Во время обычной начальной загрузки система прошивки проверяет аппаратное обеспечение и диски, запускает простой набор проверок работоспособности, а затем ищет следующий этап кода начальной загрузки. Пользовательский интерфейс прошивки позволяет назначить загрузочное устройство, как правило, путем определения приоритетов списка доступных параметров (например, "попробуйте загрузить с DVD-привода, затем с USB-диска, а затем с жесткого диска").

В большинстве случаев системные диски входят в список вторичных приоритетов. Для загрузки с определенного диска необходимо смонтировать его как диск с самым высоким приоритетом и убедиться, что в качестве загрузочного носителя включен "жесткий диск".

BIOS или UEFI

В прошлом обычная прошивка для персонального компьютера называлась BIOS (Basic Input/Output System). Однако за последнее десятилетие термин BIOS был вытеснен более формализованным и современным стандартом — UEFI. Часто можно встретить термин "UEFI BIOS", но для ясности в этой главе мы зарезервируем термин BIOS для устаревшего стандарта. Большинство систем, реализующих UEFI, могут вернуться к устаревшей реализации BIOS, если операционная система, которую они загружают, не поддерживает UEFI.

UEFI — это текущая версия предыдущего стандарта EFI. Ссылки на имя EFI сохраняются в некоторых старых документах и даже в некоторых стандартных терминах, таких как "системный раздел EFI". Во всех, кроме самых очевидных ситуаций, эти термины можно рассматривать как эквивалентные.

В наши дни поддержка UEFI довольно типична для новых персональных компьютеров, но в мире существует очень много систем BIOS. Более того, виртуальные среды часто используют BIOS в качестве основного механизма загрузки, поэтому мир BIOS еще не находится под угрозой исчезновения.

Поскольку мы предпочли бы игнорировать BIOS и говорить только об UEFI, вполне вероятно, что вы столкнетесь с обоими типами систем еще в течение довольно долгого времени. UEFI также интегрирует в себя некоторые функции BIOS, поэтому рабочие знания BIOS могут быть весьма полезны для расшифровки документации UEFI.

[1]Виртуальные системы имитируют, что имеют такой же набор устройств.

[2]Возможно, вам будет полезно временно отключить функции управления питанием монитора.

Устаревший интерфейс BIOS

Традиционный BIOS предполагает, что загрузочное устройство начинается с записи (сектора), называемой MBR (Master Boot Record). MBR содержит первичный загрузчик ("bootblock") и простую таблицу разделов диска. Объем свободного места для загрузчика настолько мал (менее 512 байт), что он не может выполнять ничего, кроме загрузки и запуска вторичного загрузчика.

Из-за ограниченности размера ни загрузочный сектор, ни BIOS не могут обрабатывать записи любой стандартной файловой системы, поэтому вторичный загрузчик должен храниться там, где его легко найти. В одном типичном сценарии загрузочный сектор считывает информацию о разбиении диска из MBR и идентифицирует раздел диска, помеченный как "активный". Затем он считывает и выполняет вторичный загрузчик с самого начала этого раздела. Эта схема называется загрузочной записью тома (volume boot record).

🕮 Разбиение диска — это способ разделения физического диска. Детали см. в разделе 20.5.

В качестве альтернативы вторичный загрузчик может находиться в "мертвой зоне", которая расположена между MBR и началом первого раздела диска. По историческим причинам первый раздел не начинается до 64-го сектора диска, поэтому эта зона обычно содержит минимум 32 Кбайт памяти: все еще не много, но достаточно для хранения драйвера файловой системы. Эта схема хранения обычно используется загрузчиком GRUB (см. раздел 2.4).

Для успешной загрузки все компоненты загрузочной цепочки должны быть правильно установлены и совместимы друг с другом. Загрузочный сектор MBR ничего не знает об операционной системе, но, поскольку он предполагает определенное местоположение для второго этапа, может быть установлено несколько версий. Вторичный загрузчик, как правило, хорошо осведомлен об операционных системах и файловых системах (он может поддерживать несколько из них) и обычно имеет собственные параметры конфигурации.

UEFI

Спецификация UEFI включает в себя современную схему разделения диска, известную как GPT (таблица разделов GUID, где GUID (globally unique identifier) обозначает "глобально уникальный идентификатор"). UEFI также понимает файловую систему FAT (File Allocation Table), простую, но функциональную схему, впервые примененную в MS DOS. Эти функции объединяются для определения концепции системного раздела EFI (ESP — EFI System Partition). Во время загрузки микропрограмма обращается к таблице разделов GPT для идентификации ESP. Затем она считывает настроенное целевое приложение непосредственно из файла в ESP и выполняет его.

🕮 Дополнительную информацию о разделах GPT см. в разделе 20.6.

Поскольку ESP представляет собой общую файловую систему FAT, ее можно смонтировать, прочитать, записать и поддерживать в любой операционной системе. Никакие скрытые загрузочные сектора на диске не требуются.[3]

[3]По правде говоря, для облегчения взаимодействия с системами BIOS спецификация UEFI предусматривает MBR-совместимую запись в начале каждого диска. Системы BIOS не могут видеть полную таблицу разделов в стиле GPT, но они, по крайней мере, распознают диск как отформатированный. Будьте осторожны, не запускайте определенные административные инструменты для MBR на дисках GPT. Они могут неправильно интерпретировать структуру диска.

Фактически никакой загрузчик вообще не требуется. Целью загрузки UEFI может быть ядро UNIX или Linux, которое настроено для прямой загрузки UEFI, что приводит к загрузке без загрузчика. На практике, однако, большинство систем по-прежнему используют загрузчик, отчасти потому, что он упрощает поддержку совместимости с устаревшими BIOS.

Интерфейс UEFI сохраняет имя пути для загрузки из ESP в качестве параметра конфигурации. Если этот параметр не задан, используется стандартный путь, в современных системах Intel обычно это путь **/efi/boot/bootx64.efi**. Более типичный путь в системе с заданной конфигурацией (для Ubuntu и загрузчика GRUB) — **/efi/ubuntu/grubx64.efi**. Другие дистрибутивы придерживаются аналогичного соглашения.

Интерфейс UEFI определяет стандартные интерфейсы API для доступа к аппаратным средствам системы. В этом отношении он представляет собой нечто вроде миниатюрной операционной системы. Он даже обеспечивает наличие дополнительных драйверов устройств на уровне UEFI, которые записываются на языке, не зависящем от процессора, и сохраняются в ESP. Операционные системы могут использовать интерфейс UEFI или напрямую управлять оборудованием.

Поскольку UEFI имеет формальный интерфейс API, переменные UEFI можно проверять и изменять (включая записи загрузочного меню) уже в загруженной системе. Например, команда **efibootmgr -v** показывает следующую сводку конфигурации загрузки:

```
$ efibootmgr -v
BootCurrent: 0004
BootOrder: 0000,0001,0002,0004,0003
Boot0000* EFI DVD/CDROM PciRoot(0x0)/Pci(0x1f,0x2)/Sata(1,0,0)
Boot0001* EFI Hard Drive PciRoot(0x0)/Pci(0x1f,0x2)/Sata(0,0,0)
Boot0002* EFI Network PciRoot(0x0)/Pci(0x5,0x0)/MAC(001c42fb5baf,0)
Boot0003* EFI Internal Shell MemoryMapped(11,0x7ed5d000,0x7f0dcfff)/
    FvFile(c57ad6b7-0515-40a8-9d21-551652854e37)
Boot0004* ubuntu HD(1,GPT,020c8d3e-fd8c-4880-9b61-
    ef4cffc3d76c,0x800,0x100000)/File(\EFI\ubuntu\shimx64.efi
```

Команда **efibootmgr** позволяет изменить порядок загрузки, выбрать следующий настроенный параметр загрузки или даже создать и уничтожить загрузочные записи. Например, установить порядок загрузки так, чтобы сначала обращаться к системному диску, а потом — к сети, игнорируя другие параметры загрузки, можно с помощью команды

```
$ sudo efibootmgr -o 0004,0002
```

Здесь мы указываем параметры `Boot0004` и `Boot0002` из выведенных выше данных.

Возможность изменять конфигурацию UEFI из пользовательского пространства означает, что информация о конфигурации прошивки может как читаться, так и записываться — это и благословение, и проклятие. В системах, разрешающих доступ на запись по умолчанию (как правило, с помощью менеджера **systemd**), команды **rm -rf /** может быть достаточно, чтобы навсегда уничтожить систему на уровне прошивки; в дополнение к удалению файлов команда **rm** также удаляет переменные и другую информацию UEFI, доступную через **/sys**[4]. Не пытайтесь повторить это дома!

[4]Для получения дополнительной информации см. `goo.gl/QMSiSG` (ссылка на статью Phoronix).

2.3. Загрузчики

Большинство процедур начальной загрузки включают в себя выполнение загрузчика, который отличается от кода BIOS/UEFI и ядра операционной системы. Он также отделен от начального загрузочного сектора в BIOS, если вы считаете этапы.

Основное задание загрузчика — определить и загрузить соответствующее ядро операционной системы. Большинство загрузчиков также могут предоставлять пользовательский интерфейс загрузки, который позволяет выбирать, какое из нескольких возможных ядер или операционных систем вызвать.

Другая задача загрузчика — это маршалинг (marshaling) аргументов конфигурации для ядра. Ядро не имеет командной строки как таковой, но обработка его параметров запуска может показаться довольно похожей на работу с оболочкой. Например, аргумент `single` или `-s` обычно указывает ядру войти в однопользовательский режим, а не выполнять нормальный процесс загрузки.

Такие параметры могут быть жестко привязаны к конфигурации загрузчика, если вы хотите, чтобы они использовались при каждой загрузке, или предоставляться "на лету" через пользовательский интерфейс загрузчика.

В следующих нескольких разделах мы обсудим GRUB (основной загрузчик операционной системы Linux) и загрузчики, используемые с системой FreeBSD.

2.4. GRUB: универсальный загрузчик



GRUB (GRand Unified Boot), разработанный GNU Project, является загрузчиком по умолчанию для большинства дистрибутивов Linux. Линия GRUB имеет два основных направления: оригинальный GRUB, теперь называемый GRUB Legacy, и новый GRUB 2, который является текущим стандартом. Убедитесь, что вы знаете, с каким вариантом загрузчика GRUB вы имеете дело, поскольку эти две версии совершенно разные.

GRUB 2 был загрузочным менеджером по умолчанию для системы Ubuntu, начиная с версии 9.10, и недавно стал загрузочным менеджером по умолчанию для системы RHEL 7. Все наши примеры дистрибутивов Linux используют его по умолчанию. В этой книге мы обсуждаем только GRUB 2 и называем его просто GRUB.

У системы FreeBSD есть собственный загрузчик (более подробно см. раздел 2.5). Однако GRUB отлично справляется с загрузкой FreeBSD. Это может оказаться удобным, если вы планируете загружать несколько операционных систем на одном компьютере. В противном случае загрузчика FreeBSD более чем достаточно.

Конфигурация GRUB

GRUB позволяет указать такие параметры, как загрузочное ядро (указанное в качестве записи меню GRUB) и режим работы для загрузки.

Поскольку эта информация о конфигурации необходима во время загрузки, можно подумать, что она будет храниться где-то в необычном месте, например в энергонезависимой системной памяти NVRAM или в секторах диска, зарезервированных для загрузчика. На самом деле GRUB понимает большинство используемых файловых систем и обычно может самостоятельно найти корневую файловую систему. Это позволяет загрузчику GRUB читать свою конфигурацию из обычного текстового файла.

Конфигурационный файл называется `grub.cfg`, и его обычно хранят в каталоге `/boot/grub` (`/boot/grub2` в системах Red Hat и CentOS) вместе с выбором других ресурсов и модулей кода, которые GRUB может потребовать для доступа во время загрузки.[5] Изменение конфигурации загрузки — это простой вопрос об обновлении файла `grub.cfg`.

Хотя можно создать файл `grub.cfg` самостоятельно, чаще всего его проще генерировать с помощью утилиты `grub-mkconfig`, которая называется `grub2-mkconfig` в системах Red Hat и CentOS и `update-grub` в системах Debian и Ubuntu. Фактически большинство дистрибутивов предполагают, что файл `grub.cfg` может быть регенерирован по желанию, и они делают это автоматически после обновлений. Если вы не предпримете никаких шагов, чтобы предотвратить это, ваш файл `grub.cfg` будет испорчен.

Как обычно в системе Linux, дистрибутивы задают конфигурацию утилиты `grub-mkconfig` различными способами. Чаще всего конфигурация задается в каталоге `/etc/default/grub` в виде присваивания переменных `sh`. В табл. 2.1 показаны некоторые из часто изменяемых опций.

Таблица 2.1. Общие параметры конфигурации GRUB в файле `/etc/default/grub`

Обозначение переменной оболочки	Содержимое или функция
GRUB_BACKGROUND	Фоновое изображение[a]
GRUB_CMDLINE_LINUX	Параметры ядра для добавления в записи меню для Linux[6]
GRUB_DEFAULT	Номер или название элемента меню по умолчанию
GRUB_DISABLE_RECOVERY	Предотвращает создание записей режима восстановления
GRUB_PRELOAD_MODULES	Список модулей GRUB, загружаемых как можно раньше
GRUB_TIMEOUT	Секунды для отображения меню загрузки перед автозагрузкой

[a] Фоновое изображение должно иметь формат `.png`, `.tga`, `.jpg` или `.jpeg`.

[6] В табл. 2.3 в разделе 3.4 перечислены некоторые из доступных опций.

🕮 Подробнее о режимах работы см. раздел 2.7.

После редактирования каталога `/etc/default/grub` запустите утилиты `update-grub` или `grub2-mkconfig`, чтобы перевести вашу конфигурацию в правильный файл `grub.cfg`. В рамках процесса построения конфигурации эти команды проверяют загрузочные ядра системы, поэтому они могут быть полезны для запуска после внесения изменений ядра, даже если вы явно не изменили конфигурацию GRUB.

Возможно, вам потребуется отредактировать файл `/etc/grub.d/40_custom`, чтобы изменить порядок, в котором ядра указаны в меню загрузки (например, после создания настраиваемого ядра), установить пароль для загрузки или изменить имена пунктов меню загрузки. Как обычно, после внесения изменений запустите утилиты `update-grub` или `grub2-mkconfig`.

Например, вот как выглядит файл `40_custom`, который вызывает пользовательское ядро в системе Ubuntu.

[5] Если вы знакомы с соглашениями файловой системы UNIX (см. главу 5), то можете задаться вопросом, почему каталог `/boot/grub` не назван как-то более стандартно, например `/var/lib/grub` или `/usr/local/etc/grub`. Причина в том, что драйверы файловой системы, используемые во время загрузки, несколько упрощены. Загрузчики не могут обрабатывать дополнительные функции, такие как точки монтирования, когда они обходят файловую систему. Все в каталоге `/boot` должно быть простым файлом или каталогом.

```
#!/bin/sh

exec tail -n +3 $0

# This file provides an easy way to add custom menu entries. Just type
# the menu entries you want to add after this comment. Be careful not to
# change the 'exec tail' line above.

menuentry 'My Awesome Kernel' {
   set root='(hd0,msdos1)'
   linux    /awesome_kernel root=UUID=XXX-XXX-XXX ro quiet
   initrd   /initrd.img-awesome_kernel
}
```

🕮 Для получения дополнительной информации о монтировании файловых систем см. раздел 5.1.

В этом примере GRUB загружает ядро из каталога **/awesome_kernel**. Пути к ядру являются относительными и связаны с загрузочным разделом который исторически монтировался как **/boot**, но с появлением UEFI теперь, скорее всего, он является демонтированным системным разделом EFI. Для проверки вашего диска и определения состояния загрузочного раздела используйте команды **gpart show** и **mount** .

Командная строка GRUB

GRUB поддерживает интерфейс командной строки для редактирования записей в файле конфигурации "на лету" во время загрузки. Чтобы войти в режим командной строки, введите команду **c** на экране загрузки GRUB.

В командной строке можно загружать операционные системы, которые не указаны в файле **grub.cfg**, отображать системную информацию и выполнять рудиментарное тестирование файловой системы. Все, что можно сделать через файл **grub.cfg**, также можно выполнить с помощью командной строки.

Нажмите клавишу <Tab>, чтобы просмотреть список возможных команд. Некоторые наиболее полезные команды показаны в табл. 2.2.

Таблица 2.2. Команды GRUB

Команда	Функция
boot	Загружает систему из указанного образа ядра
help	Получает интерактивную помощь для команды
linux	Загружает ядро Linux
reboot	Перезагружает систему
search	Поиск устройств по файлу, метке файловой системы или UUID
usb	Проверка поддержки USB

Для получения подробной информации о GRUB и параметрах командной строки обратитесь к официальному руководству по адресу `gnu.org/software/grub/manual`.

Параметры ядра Linux

Параметры запуска ядра обычно изменяют значения параметров ядра, инструктируют ядро проверять определенные устройства, указывать путь к процессу **init** или **systemd** либо назначать определенное корневое устройство. В табл. 2.3 приведено несколько примеров.

Таблица 2.3. Примеры параметров времени загрузки ядра

Опция	Значение
`debug`	Включает отладку ядра
`init=/bin/bash`	Запускает только оболочку **`bash`**; полезна для аварийного восстановления
`root=/dev/foo`	Инструктирует ядро использовать **`/dev/foo`** в качестве корневого устройства
`single`	Загрузка в однопользовательском режиме

Если параметры ядра задаются во время загрузки, они не являются постоянными. Для того чтобы сделать изменение постоянным при перезагрузке, отредактируйте соответствующую строку ядра в файле **`/etc/grub.d/40_custom`** или **`/etc/defaults/grub`** (переменная с именем `GRUB_CMDLINE_LINUX`).

В ядро Linux постоянно добавляются заплатки безопасности, исправления ошибок и новые функции. Однако в отличие от других пакетов программного обеспечения новые версии ядра обычно не заменяют старые. Вместо этого новые ядра устанавливаются наряду с предыдущими версиями, так что в случае возникновения проблем можно вернуться к старому ядру.

Это соглашение помогает администраторам отказаться от обновления, если заплатка ядра разрушает их систему, хотя это также означает, что загрузочное меню имеет тенденцию сохранять старые версии ядра. Попробуйте выбрать другое ядро, если ваша система не будет загружаться после обновления.

Подробнее о параметрах ядра см. в главе 11.

2.5. Процесс загрузки FreeBSD



Система загрузки FreeBSD во многом похожа на GRUB, поскольку загрузчик конечной стадии (под именем **`loader`**) использует файл конфигурации на основе файловой системы, поддерживает меню и предлагает интерактивный режим командной строки. Загрузчик **`loader`** — это последнее пересечение вариантов загрузки BIOS и UEFI.

Вариант BIOS: `boot0`

Как и в случае с интерфейсом GRUB, полная среда загрузчика **`loader`** слишком велика для размещения в загрузочном секторе MBR, поэтому в BIOS постепенно загружается и запускается цепочка более сложных предварительных загрузчиков.

Интерфейс GRUB объединяет все эти компоненты под общим названием "GRUB", но в системе FreeBSD ранние загрузчики являются частью отдельной системы под названием **`boot0`**, которая используется только в системах BIOS. Система **`boot0`** имеет свои собственные опции, главным образом потому, что она хранит более поздние этапы цепочки загрузки в загрузочной записи тома (см. раздел "Устаревший интерфейс BIOS" в разделе 2.2), а не перед первым дисковым разделом.

По этой причине для загрузочной записи MBR нужен указатель на раздел, который необходимо использовать для продолжения процесса загрузки. Как правило, все это автоматически настраивается как часть процесса установки FreeBSD, но если вам когда-либо понадобится настроить конфигурацию, это можно сделать с помощью команды **`boot0cfg`**.

Вариант UEFI

Операционная система FreeBSD, использующая интерфейс UEFI, создает системный раздел EFI и устанавливает там загрузочный код в файле **/boot/bootx64.efi**.[6] Это путь по умолчанию, который проверяют системы UEFI во время загрузки (по крайней мере на современных платформах персональных компьютеров), поэтому никакой конфигурации прошивки не требуется, за исключением правильно установленных приоритетов загрузки устройств.

По умолчанию система FreeBSD не сохраняет смонтированным системный раздел EFI после загрузки. Таблицу разделов можно идентифицировать с помощью команды **gpart**.

```
$ gpart show
=>          40   134217648   ada0  GPT   (64G)
            40        1600      1   efi   (800K)
          1640   127924664      2   freebsd-ufs (61G)
     127926304     6291383      3   freebsd-swap (3.0G)
     134217687           1          - free - (512B)
```

Хотя существует возможность подключить системный раздел ESP, чтобы узнать, что в нем содержится (используя команду **mount -t msdos**), вся файловая система — это всего лишь копия образа, найденного в файле **/boot/boot1.efifat** на корневом диске. Внутри нет деталей, которые пользователь мог бы изменить.

> 🕮 Для получения дополнительной информации о монтировании файловых систем см. раздел 5.1.

Если раздел ESP поврежден или удален, его можно повторно создать, настроив раздел с помощью команды **gpart**, а затем скопировав образ файловой системы с помощью команды **dd**:

```
$ sudo dd if=/boot/boot1.efifat of=/dev/ada0p1
```

Как только начальный загрузчик первого этапа UEFI находит раздел типа `freebsd-ufs`[7], он загружает UEFI-версию программы **loader** из файла **/boot/loader.efi**. С этого момента загрузка выполняется точно так же, как под управлением BIOS: загрузчик **loader** определяет, какое ядро следует загрузить, устанавливает параметры ядра и т.д.

Конфигурация загрузчика

Загрузчик **loader** на самом деле является средой сценариев на языке Forth.[8] Внутри каталога **/boot** хранится код на языке Forth, управляющий операциями загрузчика, но он вполне самодостаточный — вам не нужно изучать Forth.

Чтобы получить значения переменных конфигурации, сценарии на языке Forth считывают файл **/boot/loader.conf**, поэтому задавать их следует именно здесь. Не путайте этот файл с файлом **/boot/defaults/loader.conf**, который содержит настройки по умолчанию и не предназначен для изменения. К счастью, присваивания переменных

[6] Не путайте каталог **/boot** в системном разделе EFI с каталогом **/boot** в корневой файловой системе FreeBSD. Они отделены друг от друга и служат различным целям, хотя, конечно, оба связаны с начальной загрузкой.

[7] Начиная с версии FreeBSD 10.1, в качестве корневого раздела в системе UEFI можно использовать ZFS.

[8] Это замечательный и интересный факт, имеющий значение лишь для людей, интересующихся историей языков программирования.

в файле **loader.conf** синтаксически похожи на стандартные операции присваивания в оболочке **sh**.

Справочные man-страницы о загрузчике **loader** и файле **loader.conf** содержат информацию обо всех параметрах загрузчика и переменных конфигурации, которые их контролируют. Некоторые из наиболее интересных вариантов защищают меню загрузки паролем, меняют экран заставки, отображаемый при загрузке, и передают параметры ядра.

Команды загрузчика loader

Загрузчик понимает различные интерактивные команды. Например, чтобы найти и загрузить альтернативное ядро, необходимо использовать последовательность следующих команд:

```
Type '?' for a list of commands, 'help' for more detailed help.
OK ls
/
  d .snap
  d dev
...
  d rescue
  l home
...
OK unload
OK load /boot/kernel/kernel.old
/boot/kernel/kernel.old text=0xf8f898 data=0x124 ... b077]
OK boot
```

Здесь мы вывели содержимое корневой файловой системы (по умолчанию), выгрузили ядро по умолчанию (**/boot/kernel/kernel**), загрузили более старое ядро (**/boot/kernel/kernel.old**) и продолжили процесс загрузки.

Для получения полной документации о доступных командах выполните команду **man loader**.

2.6. Демоны управления системой

После загрузки и завершения процесса инициализации ядро создает "спонтанные" процессы в пользовательском пространстве. Они называются *спонтанными*, потому что ядро запускает их автономно — при нормальном ходе событий новые процессы создаются только по воле существующих процессов.

Большинство спонтанных процессов действительно являются частью реализации ядра. Они не обязательно соответствуют программам в файловой системе. Они не настраиваются и не требуют внимания со стороны системного администратора. Их можно распознать в списках **ps** (см. раздел 4.3) по низким значениям параметра PID и скобкам вокруг их имен (например, [pagedaemon] в системе FreeBSD или [kdump] в системе Linux).

Исключением из этого шаблона является демон управления системой. Он имеет идентификатор процесса, равный 1, и обычно работает под именем **init**. Система дает демону **init** несколько специальных привилегий, но по большей части это просто программа на уровне пользователя, как любой другой демон.

Обязанности демона `init`

Демон `init` имеет несколько функций, но его главной целью является обеспечение того, чтобы система запускала правильные комплекты служб и демонов в любой момент времени.

Для достижения этой цели `init` поддерживает понятие режима, в котором система должна работать. Ниже перечислены некоторые общепринятые режимы[9].

- Однопользовательский режим, в котором монтируется только минимальный набор файловых систем, ни одна из служб не запущена, а на консоли запущена корневая оболочка.
- Многопользовательский режим, в котором монтируются все обычные файловые системы и запущены все настроенные сетевые службы, а также оконная система и графический менеджер входа в консоль.
- Режим сервера, аналогичный режиму многопользовательского режима, но без использования графического пользовательского интерфейса на консоли.

Каждый режим связан с определенным комплектом системных служб, а демон инициализации запускает или останавливает службы по мере необходимости, чтобы привести фактическое состояние системы в соответствие с текущим активным режимом. Режимы могут также иметь связанные с ними контрольные задачи, которые запускаются всякий раз, когда режим начинается или заканчивается.

Например, демон `init` обычно выполняет многие задачи, связанные с запуском, в качестве побочного эффекта перехода от начальной загрузки к многопользовательскому режиму. К ним могут относиться следующие задачи.

- Установка имени компьютера.
- Установка часового пояса.
- Проверка дисков с помощью команды **`fsck`**.
- Монтирование файловых систем.
- Удаление старых файлов из каталога **`/tmp`**.
- Настройка сетевых интерфейсов.
- Настройка фильтров пакетов.
- Запуск других демонов и сетевых служб.

У демона `init` очень мало встроенных знаний об этих задачах. Он просто запускает набор команд или сценариев, которые были назначены для выполнения в этом конкретном контексте.

Реализации демона `init`

В настоящее время широко используются три совершенно разных стиля в процессах системного управления.

- Стиль `init`, принятый в системе System V UNIX компании AT&T, который мы называем "традиционным стилем `init`". Этот стиль был преобладающим в системах Linux до появления менеджера **`systemd`**.

[9]Не принимайте эти имена или описания режимов слишком буквально; это всего лишь примеры общих режимов работы, которые большинство систем определяют так или иначе.

- Вариант **init**, который происходит от системы BSD UNIX и используется в большинстве BSD-систем, включая FreeBSD, OpenBSD и NetBSD. Он так же проверен временем, как и стиль SysV **init**, и имеет столько же прав называться "традиционным", но для ясности мы называем его "BSD **init**". Этот вариант довольно прост по сравнению с инициализацией стиля SysV. Мы обсудим его отдельно в разделе 2.8.
- Относительно недавно у демона **init** появился конкурент — менеджер **systemd**, целью которого является полное решение всех проблем, связанных с демонами и состоянием системы. Как следствие, менеджер **systemd** захватил значительно бо́льшую территорию, чем любая традиционная версия демона **init**. Это несколько противоречивая ситуация, о которой мы поговорим ниже. Тем не менее все наши примеры дистрибутивов Linux теперь содержат менеджер **systemd**.

Хотя эти реализации являются преобладающими в данный момент, они далеки от того, чтобы быть единственным вариантом. Например, в операционной системе macOS компании Apple используется система инициализации с именем **launchd**. Пока в системе Ubuntu не был реализован менеджер **systemd**, в ней использовался другой современный вариант демона **init** под названием Upstart.



В системах Linux теоретически можно заменить инициализацию по умолчанию в зависимости от того, какой вариант вы предпочитаете. Но на практике демон **init** настолько важен для работы системы, что многое дополнительное программное обеспечение может выйти из строя. Если вы не желаете использовать менеджер **systemd**, примите в качества стандарта дистрибутив, который его не использует.

Традиционный стиль init

В традиционном мире инициализации системные режимы (например, однопользовательские или многопользовательские) называются *уровнями выполнения*. Большинство уровней выполнения обозначаются одной буквой или цифрой.

Традиционный стиль **init** появился в начале 80-х гг. Его сторонники (т.е. противники применения менеджера **systemd**) часто ссылаются на принцип: "Не сломалось — не чини". Тем не менее традиционный стиль **init** имеет ряд заметных недостатков.

Начнем с того, что традиционный демон **init** сам по себе не настолько силен, чтобы справляться с потребностями современных операционных систем. Большинство систем, которые его используют, на самом деле имеют стандартную и фиксированную конфигурацию **init**, которая никогда не изменяется. Эта конфигурация ссылается на второй уровень сценариев оболочки, которые выполняют фактическую работу по изменению уровней выполнения и позволяют администраторам вносить изменения в конфигурацию.

Второй уровень сценариев поддерживает еще и третий уровень сценариев, связанных с демоном и системой, которые привязаны к каталогам определенного уровня, указывающим, какие службы должны работать и на каком уровне запуска. Все это немного запутанно и неизящно.

Говоря конкретно, эта система не имеет общей модели зависимостей между службами, поэтому требуется, чтобы все запуски и демонстрации выполнялись в порядке, заданном администратором. Более поздние действия не могут выполняться до тех пор, пока не будут закончены все предыдущие, поэтому выполнять их параллельно невозможно, и система тратит много времени, чтобы изменить состояние.

Менеджер systemd против остального мира

Лишь немногие проблемы в области Linux обсуждались более горячо, чем переход от традиционного демона **init** к менеджеру **systemd**. По большей части, жалобы сосредоточиваются на постоянно растущих масштабах системы.

Менеджер **systemd** осуществляет все функции **init**, ранее реализованные с помощью хитроумных уловок и приемов, и формализует их в единое целое, позволяя настраивать, получать доступ и управлять службами.

В качестве системы управления пакетами менеджер **systemd** определяет надежную модель зависимостей не только среди служб, но и среди "целей". Этот термин используется в контексте **systemd** для описания режимов работы, которые в контексте традиционного демона **init** назывались уровнями выполнения. Менеджер **systemd** не только управляет процессами параллельно, но также управляет сетевыми подключениями (**networkd**), записями журнала ядра (**journald**) и авторизацией (**logind**).

Противники использования менеджера **systemd** утверждают, что философия UNIX заключается в том, чтобы системные компоненты были небольшими, простыми и модульными. Говорят, что такой фундаментальный компонент, как **init** не должен иметь монолитного контроля над многими другими подсистемами операционной системы.

Менеджер **systemd** не только порождает сложность, но и создает потенциальные недостатки в области безопасности и препятствует разграничению между платформой операционной системы и службами, которые работают на ее основе.

🕮 Дополнительную информацию об управлении пакетами см. в главе 6.

Менеджер **systemd** также критикуют за введение новых стандартов и обязанностей в ядро Linux, качество кода, предполагаемую невосприимчивость его разработчиков к сообщениям об ошибках, дизайн его основных функций и нелепый вид. Мы не можем справедливо решить эти проблемы в этой книге, но читатели могут ознакомиться с аргументами в разделе *Arguments against systemd* по адресу `without-systemd.org`, на основном сайте противников **systemd**.

Аргументы против init

Архитектурные возражения против менеджера **systemd**, изложенные выше, являются вполне разумными. Он действительно имеет большинство типичных недостатков надстроенного программного проекта.

На практике, однако, многие администраторы очень любят использовать **systemd**, и мы относимся к этому лагерю. Оставьте на время полемику и предоставьте менеджеру **systemd** шанс завоевать ваше признание. Как только вы привыкнете к нему, вы, скорее всего, оцените его многочисленные преимущества.

По крайней мере, имейте в виду, что традиционный демон **init**, который вытесняется менеджером **systemd**, не был идеальным. Помимо всего прочего, менеджер **systemd** просто устраняет несколько ненужных различий между дистрибутивами Linux.

Дебаты действительно не имеют значения, потому что соревнование завершено. Аргументы против использования **systemd** потеряли силу, когда на него переключились системы Red Hat, Debian и Ubuntu. Многие другие дистрибутивы Linux теперь принимают **systemd** либо вольно, либо невольно.

Традиционный демон **init** по-прежнему играет определенную роль, если дистрибутив либо нацелен на небольшой размер инсталляции, либо не нуждается в расширенных функциях управления процессом **systemd**. Также существует значительное количество

реваншистов, которые презирают систему из принципа, поэтому некоторые дистрибутивы Linux обязательно будут поддерживать традиционную инициализацию в течение неопределенного срока в качестве формы протеста.

Тем не менее мы не считаем, что традиционный демон **init** имеет будущее, чтобы заслужить подробное обсуждение в этой книге. Для системы Linux мы в основном ограничиваемся менеджером **systemd**.

Мы также обсудим очень простую систему инициализации, используемую FreeBSD, в разделе 2.8.

2.7. Менеджер systemd в деталях

Конфигурация и управление системными службами — это область, в которой дистрибутивы Linux традиционно отличаются друг от друга. Менеджер **systemd** нацелен на стандартизацию этого аспекта системного администрирования, и в этой области он достиг бо́льших успехов, чем любая предыдущая альтернатива.

Системный менеджер **systemd** — это не отдельный демон, а набор программ, демонов, библиотек, технологий и компонентов ядра. Как отмечается в сообщении, опубликованном в блоге, посвященном **systemd** на странице `0pointer.de/blog`, полная сборка проекта генерирует 69 разных двоичных файлов. Подумайте об этом как о кондитерской, в которой вы обязаны съесть все конфеты.

Поскольку менеджер **systemd** сильно зависит от особенностей ядра Linux, это предложение предназначено только для Linux. Вы не увидите его в системе BSD или любом другом варианте UNIX в течение следующих пяти лет.

Модули и модульные файлы

Сущность, которой управляет менеджер **systemd**, называется *модулем* (unit)[10]. Более конкретно, модулем может быть "служба, сокет, устройство, точка монтирования, точка автоматического монтирования, файл или раздел подкачки, цель запуска, просматриваемый файловый путь, таймер, управляемый **systemd**, часть ресурса управления, группа созданных извне процессов или портал в альтернативную вселенную".[11] Хорошо, мы даже захватили часть альтернативной вселенной, но она все еще занимает много территории.

Внутри менеджера **systemd** поведение каждого модуля определяется и настраивается модульным файлом. Например, в случае службы файл модуля указывает местоположение исполняемого файла для демона, сообщает **systemd**, как запускать и останавливать службу, а также идентифицирует любые другие модули, от которых зависит служба.

Вскоре мы рассмотрим синтаксис модульных файлов, а пока приведем простой пример из системы Ubuntu. Этот файл устройства **rsync.service**; он обрабатывает запуск демона **rsync**, который отображает файловые системы.

```
[Unit]
Description=fast remote file copy program daemon
ConditionPathExists=/etc/rsyncd.conf

[Service]
ExecStart=/usr/bin/rsync --daemon --no-detach
```

[10] На жаргоне программистов их часто называют юнитами. — *Примеч. ред.*

[11] В основном, цитируется по man-странице **systemd.unit**.

```
[Install]
WantedBy=multi-user.target
```

Если вам показалось, что это напоминает файл в формате **.ini**, используемом в системах MS-DOS, значит, вы хорошо понимаете почему к **systemd** так плохо относятся. Модульные файлы могут находиться в нескольких местах. Основное место, где пакеты сохраняют свои модульные файлы во время инсталляции — **/usr/lib/systemd/system**; в некоторых системах для этого используется каталог **/lib/systemd/system**. Содержимое этого каталога считается ресурсом, поэтому вы не должны изменять его. Файлы локальных файлов и настройки могут выполняться в каталоге **/etc/systemd/system**. В каталоге **/run/systemd/system** есть также каталог модулей, который является рабочей областью для переходных модулей.

Менеджер **systemd** поддерживает телескопическое представление всех этих каталогов, поэтому они в значительной степени эквивалентны. Если возникает конфликт, то файлы в каталоге **/etc** имеют наивысший приоритет.

🕮 Для получения дополнительной информации о демоне **rsync** см. раздел 17.4.

По соглашению, модульные файлы именуются суффиксом, который зависит от типа настраиваемого устройства. Например, служебные модули имеют суффикс **.service**, а таймеры используют суффикс **.timer**. Внутри модульного файла некоторые разделы (например, `[Unit]`) относятся в общем случае ко всем типам модулей, но другие (например, `[Service]`) могут отображаться только в контексте определенного типа устройства.

Команда systemctl: управление менеджером systemd

Команда **systemctl** — это универсальная команда для изучения состояния менеджера **systemd** и внесения изменений в его конфигурацию. Как и в случае с системой Git и несколькими другими сложными пакетами программного обеспечения, первый аргумент команды **systemctl** обычно представляет собой подкоманду, которая задает общую последовательность действий, а последующие аргументы являются специфическими для этой конкретной подкоманды. Подкоманды могут быть отдельными командами верхнего уровня, но для согласованности и ясности они объединены в команду **systemctl**.

🕮 Для получения дополнительной информации о системе Git см. раздел 7.8.

Выполнение команды **systemctl** без каких-либо аргументов по умолчанию вызывает подкоманду **list-units**, в которой отображаются все загруженные и активные службы, сокеты, цели, смонтированные диски и устройства. Для того чтобы показывать только загруженные и активные службы, используйте ключ **-type = service**:

```
$ systemctl list-units --type=service
UNIT                      LOAD    ACTIVE   SUB        DESCRIPTION
accounts-daemon.service   loaded  active   running    Accounts Service
...
wpa_supplicant.service    loaded  active   running  WPA supplicant
```

Также иногда полезно видеть все инсталлированные модульные файлы, независимо от того, активны они или нет:

```
$ systemctl list-unit-files --type=service
UNIT FILE                    STATE
...
cron.service                 enabled
```

```
cryptdisks-early.service    masked
cryptdisks.service          masked
cups-browsed.service        enabled
cups.service                disabled
...
wpa_supplicant.service      disabled
x11-common.service          masked

188 unit files listed.
```

Для подкоманд, действующих на определенный модуль (например, **systemctl status**), команда **systemctl** обычно может принимать имя модуля без суффикса, указывающего тип модуля (например, **cups** вместо **cups.service**). Тем не менее тип модуля по умолчанию, с которым объединены простые имена, зависит от подкоманды.

В табл. 2.4 показаны наиболее востребованные и полезные подкоманды команды **systemctl** (для получения полного списка см. man-страницу **systemctl**).

Таблица 2.4. Наиболее распространенные подкоманды команды systemctl

Подкоманда	Функция
list-unit-files *[шаблон]*	Показывает установленные модули; существует возможность сравнения по *шаблону*
enable *модуль*	Включает *модуль* для активации при загрузке
disable *модуль*	Запрещает запуск *модуля* при загрузке
isolate *цель*	Изменяет режим работы на *целевой*
start *модуль*	Немедленно активирует *модуль*
stop *модуль*	Немедленно деактивирует *модуль*
restart *модуль*	Немедленно перезагружает (или запускает, если не работает) *модуль*
status *модуль*	Показывает состояние *модуля* и последние записи журнала
kill *шаблон*	Отправляет сигнал в *модуль*, соответствующий шаблону
reboot	Перезагрузка компьютера
daemon-reload	Перезагружает файлы *модулей* и конфигурацию **systemd**

Состояние модуля

В выводе команды **systemctl list-unit-files**, приведенном выше, мы видим, что модуль **cups.service** отключен. Для получения более подробной информации мы можем использовать команду **systemctl status**.

```
$ sudo systemctl status -l cups
cups.service - CUPS Scheduler
   Loaded: loaded (/lib/systemd/system/cups.service; disabled; vendor
     preset: enabled)
   Active: inactive (dead) since Sat 2016-12-12 00:51:40 MST; 4s ago
     Docs: man:cupsd(8)
   Main PID: 10081 (code=exited, status=0/SUCCESS)
Dec 12 00:44:39 ulsah systemd[1]: Started CUPS Scheduler.
Dec 12 00:44:45 ulsah systemd[1]: Started CUPS Scheduler.
Dec 12 00:51:40 ulsah systemd[1]: Stopping CUPS Scheduler...
Dec 12 00:51:40 ulsah systemd[1]: Stopped CUPS Scheduler.
```

Здесь команда **systemctl** показывает, что служба в настоящее время неактивна (dead), и сообщает, когда процесс был прекращен. (Для этого примера мы отключили его всего несколько секунд назад.) Он также показывает (в разделе с надписью Loaded), что служба по умолчанию была включена при запуске, но теперь она отключена.

Последние четыре строки — это последние записи журнала. По умолчанию записи журнала конденсируются так, что каждая запись занимает только одну строку. Это сжатие часто делает записи нечитаемыми, поэтому мы включили опцию **-l** для запроса полных записей. В этом случае нет никакой разницы, но это полезная привычка.

В табл. 2.5 отображаются состояния, с которыми вы чаще всего столкнетесь при проверке модулей.

Таблица 2.5. Состояние модульных файлов

Состояние	Описание
bad	У менеджера **systemd** возникла какая-то проблема; обычно это связано со сбоем модульного файла
disabled	Присутствует, но не настроен для автономного запуска
enabled	Инсталлирован и запущен; стартует автономно
indirect	Отключен, но имеет одинаковые значения в разделах Also, которые могут быть включены
linked	Модульный файл, доступный через символическую ссылку
masked	Нежелательный статус с логической точки зрения
static	Зависит от другого устройства; не требует установки

Состояния enabled и disabled применяются только к модульным файлам, которые находятся в одном из системных каталогов **systemd** (т.е. они не связаны символической ссылкой) и имеют раздел [Install] в своих модульных файлах. Включенные модули, по-видимому, действительно должны считаться "инсталлированными", что означает, что директивы в разделе [Install] были выполнены и что устройство подключено к его обычным активационным триггерам. В большинстве случаев это состояние заставляет модуль автоматически активироваться после загрузки системы.

Аналогично, состояние disabled является чем-то неправильным, потому что единственное, что фактически отключено, — это нормальный путь активации. Можно вручную активировать устройство, которое было отключено, запустив команду **systemctl**; менеджер **systemd** не будет возражать.

У многих устройств нет процедуры инсталляции, поэтому нельзя сказать, что они включены или отключены; они просто доступны. Статус таких элементов указан как static. Они активируются только вручную (**systemctl start**) или указываются в качестве зависимостей от других активных модулей.

Файлы, имеющие состояние linked, были созданы с помощью команды **systemctl link**. Эта команда создает символическую ссылку из одного из каталогов **system** менеджера **systemd** на модульный файл, который находится в другом месте в файловой системе. Такие модульные файлы могут обрабатываться командами или указываться в качестве зависимостей, но они не являются полноправными элементами системы и имеют некоторые заметные отклонения. Например, применение команды **systemctl disable** к модульному файлу в состоянии linked приводит к удалению связи и всех ссылок на него.

К сожалению, точное поведение модульных файлов в состоянии linked недостаточно хорошо документировано. Хотя идея хранить локальные модульные файлы в отдель-

ном репозитории и связывать их с менеджером **systemd** имеет определенную привлекательность, вероятно, это не лучший подход на данный момент. Просто сделайте копии.

Состояние `masked` означает "заблокирован администратором". Менеджер **systemd** знает о модуле, но ему запрещено активировать его или действовать по любой из его конфигурационных директив с помощью команды **systemctl mask**. В этом случае следует отключить модули, находящиеся в состоянии `enabled` или `linked`, с помощью команды **systemctl disable** и зарезервировать команду **systemctl mask** для модулей в состоянии `static`.

Возвращаясь к нашему исследованию службы **cups**, мы могли бы использовать следующие команды для ее повторного включения и запуска.

```
$ sudo systemctl enable cups
Synchronizing state of cups.service with SysV init with /lib/systemd/
    systemd-sysv-install...
Executing /lib/systemd/systemd-sysv-install enable cups
insserv: warning: current start runlevel(s) (empty) of script `cups'
    overrides LSB defaults (2 3 4 5).
insserv: warning: current stop runlevel(s) (1 2 3 4 5) of script `cups'
    overrides LSB defaults (1).
Created symlink from /etc/systemd/system/sockets.target.wants/cups.socket
    to /lib/systemd/system/cups.socket.
Created symlink from /etc/systemd/system/multi-user.target.wants/cups.
    path to /lib/systemd/system/cups.path.
$ sudo systemctl start cups
```

Цели

Модульные файлы могут объявлять свои отношения с другими модулями различными способами. Так, в примере, приведенном в начале раздела "Модули и модульные файлы", спецификатор `WantedBy` означает, что, если система имеет модуль **multi-user.target**, то данный модуль должен зависеть от него (**rsync.service**), когда тот находится в состоянии `enabled`.

Поскольку модули непосредственно поддерживают управление зависимостями, для реализации эквивалента уровней выполнения **init** не требуется никаких дополнительных механизмов. Для ясности менеджер **systemd** определяет отдельный класс модулей (типа **.target**), чтобы использовать их как маркеры общих режимов работы. Однако цели не имеют особой силы, за исключением управления зависимостями, доступного для любого другого модуля.

Традиционный демон **init** определяет как минимум семь уровней выполнения, но многие из них на самом деле не используются. Стремясь к ложно понятой исторической преемственности, менеджер **systemd** определяет цели как прямые аналоги уровней запуска демона **init** (**runlevel0.target** и т.д.). Он также определяет мнемонические цели для повседневного использования, такие как **poweroff.target** и **graphical.target**. В табл. 2.6 показано сопоставление между уровнями выполнения **init** и целями **systemd**.

Единственными целями, которые действительно необходимо знать, являются **multi-user.target** и **graphical.target** для повседневного использования и **rescue.target** для доступа к однопользовательскому режиму. Для того чтобы изменить текущий режим работы системы, используйте команду **systemctl isolate**:

```
$ sudo systemctl isolate multi-user.target
```

Таблица 2.6. Сопоставление между уровнями запуска `init` и системными целями

Уровень выполнения	Цель	Описание
0	`poweroff.target`	Остановка системы
emergency	`emergency.target`	Простейшая оболочка для восстановления системы
1, s, single	`rescue.target`	Однопользовательский режим
2	`multi-user.target`[a]	Многопользовательский режим (командная строка)
3	`multi-user.target`[a]	Многопользовательский сетевой режим
4	`multi-user.target`[a]	Обычно не используется `init`
5	`graphical.target`	Многопользовательский сетевой режим с графическим интерфейсом
6	`reboot.target`	Перезагрузка системы

[a] По умолчанию цель `multi-user.target` соответствует `runlevel3.target`, многопользовательскому сетевому режиму.

Подкоманда `isolate` называется так, потому что она активирует указанную цель и ее зависимости, но деактивирует все остальные модули.

В традиционном демоне `init` для изменения уровней запуска после загрузки системы используется команда `telinit`. Некоторые дистрибутивы теперь определяют `telinit` как символическую ссылку на команду `systemctl`.

Для того чтобы увидеть цель, к которой система загружается по умолчанию, запустите подкоманду `get-default`:

```
$ systemctl get-default
graphical.target
```

Большинство дистрибутивов Linux загружаются по умолчанию в модуль `graphical.target`, что не подходит для серверов, которым не нужен графический интерфейс. Но это легко изменить:

```
$ sudo systemctl set-default multi-user.target
```

Чтобы увидеть все доступные цели системы, запустите системные списки:

```
$ systemctl list-units --type = target
```

Зависимости между модулями

Пакеты программного обеспечения Linux обычно поставляются со своими собственными модульными файлами, поэтому системные администраторы не нуждаются в подробном знании всего языка конфигурации. Тем не менее им необходимы рабочие знания системы зависимостей `systemd` для диагностики и устранения проблем с зависимостями.

Во-первых, не все зависимости являются явными. Менеджер `systemd` берет на себя функции старого демона `inetd`, а также расширяет эту идею в домене межпроцессной системы связи D-Bus. Другими словами, менеджер `systemd` знает, какие сетевые порты или IPC-соединения указывают, где будет размещаться указанная служба, и может прослушивать запросы по этим каналам, не запуская службу. Если клиент выполняет материализацию, система просто запускает фактическое обслуживание и отключает соединение. Служба запускается, если она фактически используется, и остается бездействующей в противном случае.

Во-вторых, менеджер **systemd** делает некоторые предположения о нормальном поведении большинства видов единиц. Точные предположения варьируются в зависимости от типа единицы. Например, менеджер **systemd** предполагает, что среднестатистическая служба является надстройкой, которая не должна запускаться на ранних этапах инициализации системы. Отдельные блоки могут отключать эти допущения с помощью строки

```
DefaultDependencies = false
```

в разделе `[Unit]` их модульного файла; по умолчанию значение равно `true`. Чтобы увидеть точные предположения, которые применяются к каждому типу модуля, см. справочную страницу для типа **systemd.unit**, (например, **man systemd.service**).

Третий класс зависимостей — те, которые явно объявлены в разделе `[Unit]` модульных файлов. Доступные параметры показаны в табл. 2.7.

Таблица 2.7. Явные зависимости в разделе `[Unit]` модульного файла

Параметр	Значение
`Wants`	Модули, которые должны быть активированы одновременно, если это возможно, но не обязательно
`Requires`	Строгие зависимости; отказ от каких-либо предварительных условий прекращает работу этой службы
`Requisite`	Аналогично `Requires`, но модуль должен быть активным
`BindsTo`	Аналогично `Requires`, но модуль должен быть связан еще более тесно
`PartOf`	Аналогично `Requires`, но влияет только на запуск и остановку
`Conflicts`	Отрицательные зависимости; не может взаимодействовать с этими единицами

За исключением параметра `Conflicts`, все параметры в табл. 2.7 выражают основную идею о том, что настраиваемый модуль зависит от некоторого набора других модулей. Точные различия между этими параметрами являются незначительными и в первую очередь интересны разработчикам служб. Когда это возможно, предпочтение следует отдавать наименее ограничительному варианту, `Wants`.

Можно расширить группу `Wants` или `Requires`, создав файл *модульный-файл*.**wants** или *модульный-файл*.**requires** в каталоге **/etc/systemd/system** и добавив туда символические ссылки на другие модульные файлы. Еще лучше, просто дайте команде **systemctl** сделать это за вас. Например, команда

```
$ sudo systemctl add-хочет multi-user.target my.local.service
```

добавляет зависимость от модуля **my.local.service** к стандартной многопользовательской цели, гарантируя, что служба будет запускаться каждый раз, когда система переходит в многопользовательский режим.

В большинстве случаев такие специальные зависимости устанавливаются автоматически на основе раздела `[Install]` в модульных файлах. Этот раздел содержит опции `WantedBy` и `RequiredBy`, которые читаются, только если модуль включен с помощью команды **systemctl enable** или отключен с помощью команды **systemctl disable**. При включении они заставляют команду **systemctl** выполнять эквивалент **add-wants** для каждой опции `WantedBy` или **add-require** для каждой опции `RequiredBy`.

Разделы `[Install]` сами по себе не влияют на нормальную работу, поэтому, если модуль не запускается, убедитесь, что он правильно подключен и связан с символической ссылкой.

Порядок выполнения

Разумно предположить, что если модуль A требует (Requeres) наличия модуля B, то модуль B будет запущен или настроен до модуля A. Но на самом деле это не так. В менеджере **systemd** порядок, в котором блоки активируются (или деактивируются), является совершенно *отдельным вопросом*.

Когда система переходит в новое состояние, менеджер **systemd** сначала отслеживает различные источники информации о зависимости, изложенные в предыдущем разделе, чтобы определить задействованные модули. Затем он использует разделы Before и After из модульных файлов, чтобы упорядочить список заданий. Если модули не имеют разделов Before или After, они могут быть скорректированы параллельно.

Это удивительная, но достойная внимания функция дизайна. Одной из основных целей проектирования менеджера **systemd** было облегчение параллелизма, поэтому логично, что модули не получают зависимостей сериализации, если они явно не запрашивают их.

На практике раздел After используется чаще, чем Wants или Requires. Определения целей (и, в частности, обратных зависимостей, закодированных в предложениях WantedBy и RequiredBy) устанавливают общие контуры служб, работающих в каждом рабочем режиме, а отдельные пакеты беспокоятся только о своих непосредственных и прямых зависимостях.

Более сложный пример файла

Теперь внимательно рассмотрим несколько директив, используемых в модульных файлах. Вот модульный файл для веб-сервера NGINX, **nginx.service**.

```
[Unit]
Description=The nginx HTTP and reverse proxy server
After=network.target remote-fs.target nss-lookup.target

[Service]
Type=forking
PIDFile=/run/nginx.pid
ExecStartPre=/usr/bin/rm -f /run/nginx.pid
ExecStartPre=/usr/sbin/nginx -t
ExecStart=/usr/sbin/nginx
ExecReload=/bin/kill -s HUP $MAINPID
KillMode=process
KillSignal=SIGQUIT
TimeoutStopSec=5
PrivateTmp=true

[Install]
WantedBy=multi-user.target
```

Эта служба имеет тип forking, что означает, что команда запуска должна завершиться, даже если демон продолжает работать в фоновом режиме. Так как менеджер **systemd** не будет непосредственно запускать демона, тот записывает свой идентификатор PID (идентификатор процесса) в указанном PIDFile, чтобы **systemd** мог определить, какой процесс является основным экземпляром демона.

Строки Exec — это команды для запуска в различных обстоятельствах. Команды ExecStartPre запускаются до фактического запуска службы; показанные здесь команды подтверждают синтаксис конфигурационного файла NGINX и гарантируют удаление любого существующего PID-файла. ExecStart — это команда, которая фактически

запускает службу. Команда ExecReload сообщает менеджеру **systemd**, как заставить службу перечитать ее файл конфигурации. (Менеджер **systemd** автоматически устанавливает переменную среды MAINPID в соответствующее значение.)

🕮 Дополнительную информацию о сигналах см. в разделе 4.2.

Завершение этой службы обрабатывается через параметры KillMode и KillSignal, которые сообщают **systemd**, что демон службы интерпретирует сигнал QUIT как инструкцию для очистки и выхода. Строка

```
ExecStop = /bin/kill -s HUP $MAINPID
```

будет иметь по существу тот же результат. Если работа демона не закончится за количество секунд, заданных параметром TimeoutStopSec, менеджер **systemd** заставит его прекратить работу с помощью сигнала TERM, а затем неперехватываемого сигнала KILL.

Параметр PrivateTmp — попытка повысить безопасность. Он помещает каталог службы **/tmp** в место, отличающееся от фактического каталога **/tmp**, который используется всеми процессами и пользователями системы.

Локальные службы и настройки

Как показано в предыдущих примерах, довольно просто создать модульный файл для домашней службы. Просмотрите примеры в каталоге **/usr/lib/systemd/system** и адаптируйте их к своим потребностям. Для получения полного списка параметров конфигурации служб обратитесь к справочной странице для **systemd.service**. Параметры, общие для всех типов устройств, описаны на справочной странице **systemd.unit**.

Поместите свой новый файл в каталог **/etc/systemd/system**. Затем можно запустить команду

```
$ sudo systemctl enable custom.service
```

для активации зависимостей, перечисленных в разделе [Install] в служебном файле.

Как правило, не следует редактировать модульный файл, написанный не вами. Вместо этого создайте каталог конфигурации в каталоге **/etc/systemd/system/unit-file.d** и добавьте один или несколько файлов конфигурации, которые называются xxx.**conf**. Часть xxx не имеет значения; просто убедитесь, что файл имеет суффикс **.conf** и находится в нужном месте. Имя **override.conf** является стандартным.

Файлы с расширением **.conf** имеют тот же формат, что и модульные файлы, и на самом деле менеджер **systemd** сглаживает их все вместе с исходным файлом. Однако, если оба источника попытаются установить значение одного итого же параметра, переопределенные файлы имеют приоритет над исходными модульными файлами.

Следует иметь в виду, что многие параметры менеджера **systemd** могут отображаться в модульном файле более одного раза. В этих случаях множественные значения образуют список и остаются активными одновременно. Если вы задаете значение в файле **override.conf**, оно присоединяется к списку, но не заменяет существующие записи. Это может быть не тем, что вы хотите. Чтобы удалить существующие записи из списка, просто присвойте параметру пустое значение, а затем добавьте свой.

Рассмотрим пример. Предположим, что ваша организация хранит файл конфигурации NGINX в нестандартном месте, например **/usr/local/www/nginx.conf**. Вам нужно запустить демон **nginx** с параметром **-c /usr/local/www/nginx.conf**, чтобы он мог найти нужный файл конфигурации.

Вы не можете просто добавить эту опцию в файл **/usr/lib/systemd/system/nginx.service**, потому что он будет заменяться каждый раз, когда пакет NGINX обновляется или модифицируется.

Вместо этого можно использовать следующую последовательность команд.

```
$ sudo mkdir /etc/systemd/system/nginx.service.d
$ sudo cat > !$/override.conf[12]
[Service]
ExecStart=
ExecStart= /usr/sbin/nginx -c /usr/local/www/nginx.conf
<Ctrl+D>
$ sudo systemctl daemon-reload
$ sudo systemctl restart nginx.service
```

Первый параметр `ExecStart=` удаляет текущую запись, а второй устанавливает альтернативную команду запуска. Команда **systemctl daemon-reload** осуществляет повторный синтаксический разбор модульных файлов.

Тем не менее она не перезапускает демоны автоматически, поэтому вам придется повторно выполнить команду **systemctl**, чтобы изменения вступили в силу немедленно.

Эта последовательность команд представляет собой настолько распространенную идиому, что менеджер **systemctl** теперь реализует ее непосредственно:

```
$ sudo systemctl edit nginx.service
<отредактируйте замещающий файл в редакторе>
$ sudo systemctl restart nginx.service
```

Как уж было сказано, вам все равно придется перезапустить менеджер вручную.

Последнее, что нужно знать о переопределяемых файлах, заключается в том, что они не могут изменить раздел `[Install]` модульного файла. Любые сделанные вами изменения молча игнорируются. Просто добавьте зависимости напрямую с помощью команд **systemctl add-needs** или **systemctl add-require**.

Предостережения об управлении службами и запуском

Применение менеджера **systemd** имеет много архитектурных последствий, и его адаптация — непростая задача для разработчиков дистрибутивов Linux. Текущие выпуски — это в основном системы Франкенштейна, которые используют большую часть системы, но также сохраняют несколько ссылок на прошлые версии. Иногда старые версии просто еще не полностью преобразованы. В других случаях для облегчения совместимости были оставлены различные формы устаревшего кода.

Хотя менеджер **systemctl** можно и нужно использовать для управления службами и демонами, не удивительно, если вы до сих пор запускаете традиционные сценарии **init** или связанные с ними вспомогательные команды. Если вы попытаетесь использовать менеджер **systemctl** для отключения сети в системе CentOS или Red Hat, например, вы получите следующий вывод.

```
$ sudo systemctl disable network
network.service is not a native service, redirecting to /sbin/chkconfig.
Executing /sbin/chkconfig network off
```

[12] Символы **>** и **!$** — это метасимволы оболочки. Символ **>** переадресовывает вывод в файл, а действие символов **!$** распространяется до последнего компонента предыдущей командной строки, так что вам не нужно его повторно вводить. Все оболочки понимают эти обозначения. Информацию о некоторых других удобных функциях см. в разделе 7.2.

🕮 Дополнительную информацию о службе Apache см. в разделе 19.3.

Традиционные сценарии инициализации часто продолжают функционировать в системе **systemd**. Например, сценарий инициализации **/etc/rc.d/init.d/my-old-service** может быть автоматически сопоставлен с модульным файлом, таким как **my-old-service.service**, во время инициализации системы или при выполнении команды **systemctl daemon-reload**. Примером является служба Apache 2 на Ubuntu 17.04: попытка отключить **apache2.service** приводит к следующему выводу.

```
$ sudo systemctl disable apache2
Synchronizing state of apache2.service with SysV service script with
    /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install disable apache2
```

Конечный результат соответствует тому, что вы хотели, но процесс проходит довольно окольным путем.



В системах Red Hat и CentOS все еще запускается сценарий **/etc/rc.d/rc.local** во время загрузки, если вы настроите его на выполнение.[13] Теоретически можно использовать этот сценарий для взлома уязвимостей сайта или загрузки задач, если это необходимо. (На данный момент, однако, вам нужно проигнорировать хакерские возможности и сделать что-то полезное в системе, создав соответствующий набор файлов модулей.)

Некоторые задачи загрузки систем Red Hat и CentOS продолжают использовать конфигурационные файлы, найденные в каталоге **/etc/sysconfig**. Эти данные приведены в табл. 2.8.

Таблица 2.8. Файлы и подкаталоги каталога Red Hat /etc/sysconfig

Файл или каталог	Содержимое
console/	Каталог, который исторически допускал настраиваемое сопоставление клавиш
crond	Аргументы для перехода к демону **crond**
init	Конфигурация для обработки сообщений из сценариев запуска
iptables-config	Загружает дополнительные модули **iptables**, такие как NAT-помощники
network-scripts/	Сценарии аксессуаров и сетевые файлы конфигурации
nfs	Необязательные аргументы RPC и NFS
ntpd	Параметры командной строки **ntpd**
selinux	Символическая ссылка на каталог **/etc/selinux/config**[a]

[a]Устанавливает аргументы для системы SELinux или позволяет полностью отключить ее; см. раздел 3.4.

Пара пунктов табл. 2.8 заслуживают дополнительного комментария.

- Каталог сетевых сценариев содержит дополнительные материалы, относящиеся к сетевой конфигурации. Единственное, что вам может понадобиться изменить здесь, — файлы с именем **ifcfg-interface**. Например, файл **network-scripts/ifcfg-eth0** содержит параметры конфигурации для интерфейса **eth0**. Он устанавливает IP-адрес и сетевые параметры интерфейса. Дополнительная информация о настройке сетевых интерфейсов приведена в разделе 13.10.
- Файл **iptables-config** фактически не позволяет изменять правила **iptables** (брандмауэра). Это просто способ загрузки дополнительных модулей, таких как

[13]Быстрая команда **sudo chmod +x /etc/rc.d/rc.local** гарантирует, что файл будет исполняться.

трансляция сетевых адресов (NAT), если вы собираетесь пересылать пакеты или использовать систему в качестве маршрутизатора. Дополнительная информация о настройке **iptables** содержится в разделе 13.14.

Журнал systemd

Фиксация сообщений в журнале, созданных ядром, всегда был сложной задачей. Она стала еще более важной с появлением виртуальных и облачных систем, поскольку невозможно просто стоять перед консолями этих систем и следить за тем, что происходит. Часто важнейшая диагностическая информация терялась в эфире.

Менеджер **systemd** устраняет эту проблему с помощью универсальной структуры ведения журнала, которая включает все сообщения ядра и службы от начальной загрузки до окончательного завершения. Этот объект, называемый *журналом*, управляется демоном **journald**.

Системные сообщения, записанные журналом, хранятся в каталоге **/run**. Демон **rsyslog** может обрабатывать эти сообщения и хранить их в традиционных файлах журналов или пересылать их на удаленный сервер **syslog**. Можно также напрямую обращаться к журналам с помощью команды **journalctl**.

Без аргументов команда **journalctl** выводит все записи журнала (начиная с самых старых).

```
$ journalctl
-- Logs begin at Fri 2016-02-26 15:01:25 UTC, end at Fri 2016-02-26
     15:05:16 UTC. --
Feb 26 15:01:25 ubuntu systemd-journal[285]: Runtime journal is using
     4.6M (max allowed 37.0M, t
Feb 26 15:01:25 ubuntu systemd-journal[285]: Runtime journal is using
     4.6M (max allowed 37.0M, t
Feb 26 15:01:25 ubuntu kernel: Initializing cgroup subsys cpuset
Feb 26 15:01:25 ubuntu kernel: Initializing cgroup subsys cpu
Feb 26 15:01:25 ubuntu kernel: Linux version 3.19.0-43-generic (buildd@
     lcy01-02) (gcc version 4.
Feb 26 15:01:25 ubuntu kernel: Command line: BOOT_IMAGE=/boot/vmlinuz-
     3.19.0-43-generic root=UUI
Feb 26 15:01:25 ubuntu kernel: KERNEL supported cpus:
Feb 26 15:01:25 ubuntu kernel: Intel GenuineIntel
...
```

Можно настроить демон **journald** для сохранения сообщений из предыдущих загрузок. Для этого отредактируйте файл **/etc/systemd/journald.conf** и настройте атрибут `Storage`:

```
[Journal]
Storage=persistent
```

Изменив конфигурацию демона **journald**, вы получите список приоритетных загрузок.

```
$ journalctl --list-boots
-1 a73415fade0e4e7f4bea60913883d180dc Fri 2016-02-26 15:01:25 UTC
   Fri 2016-02-26 15:05:16 UTC
0 0c563fa3830047ecaa2d2b053d4e661d Fri 2016-02-26 15:11:03 UTC Fri
   2016-02-26 15:12:28 UTC
```

Затем можно получать доступ к сообщениям из предыдущей загрузки, ссылаясь на свой индекс или длинный идентификатор.

```
$ journalctl -b -1
$ journalctl -b a73415fade0e4e7f4bea60913883d180dc
```

Для того чтобы ограничиться конкретным модулем, используйте флаг **-u**.

```
$ journalctl -u ntp
-- Logs begin at Fri 2016-02-26 15:11:03 UTC, end at Fri 2016-02-26
   15:26:07 UTC. --
Feb 26 15:11:07 ub-test-1 systemd[1]: Stopped LSB: Start NTP daemon.
Feb 26 15:11:08 ub-test-1 systemd[1]: Starting LSB: Start NTP daemon...
Feb 26 15:11:08 ub-test-1 ntp[761]: * Starting NTP server ntpd
...
```

Ведение журнала **systemd** более подробно описано в главе 10.

2.8. Сценарии инициализации и запуска системы FreeBSD

Система FreeBSD использует инициализацию в стиле BSD, который не поддерживает концепцию уровней выполнения. Чтобы привести систему в полностью загруженное состояние, инициализация FreeBSD запускает программу **/etc/rc**. Эта программа является сценарием оболочки, но ее нельзя изменять напрямую. Вместо этого система **rc** реализует несколько стандартных способов расширения системы запуска и изменения конфигурации, доступных для системных администраторов и пакетов программного обеспечения.

🕮 Информацию о сценариях оболочки см. в главе 7.

Прежде всего, **/etc/rc** — это оболочка, которая запускает другие сценарии, большинство из которых находятся в каталогах **/usr/local/etc/rc.d.** и **/etc/rc.d**. Однако, прежде чем запускать любой из этих сценариев, система **rc** выполняет три файла, которые содержат информацию о конфигурации для системы.

- **/etc/defaults/config**
- **/etc/rc.conf**
- **/etc/rc.conf.local**

Эти файлы сами являются сценариями, но обычно они содержат только определения значений переменных оболочки. Затем сценарии запуска проверяют эти переменные, чтобы определить, как вести себя дальше. (Программа **/etc/rc** использует некоторые средства оболочки, чтобы гарантировать, что переменные, определенные в этих файлах, будут видны повсюду.)

🕮 Дополнительную информацию о сценариях оболочки см. в главе 7.

В файле **/etc/defaults/rc.conf** перечислены все параметры конфигурации и их настройки по умолчанию. Никогда не редактируйте этот файл, чтобы сценарий инициализации не перезаписал ваши изменения при следующем обновлении системы. Вместо этого просто переопределите значения по умолчанию, установив их снова в файлах **/etc/rc.conf** или **/etc/rc.conf.local**. На справочной странице **rc.conf** имеется обширный список переменных, которые можно задать.

Теоретически файлы **rc.conf** могут также содержать имена других каталогов, в которых для запуска сценариев можно установить значение переменной **local_startup**.

Значение по умолчанию — **/usr/local/etc/rc.d**, и мы рекомендуем оставить его неизменным.[14]

Заглянув в файл **/etc/rc.d**, можно увидеть, что существует множество различных сценариев запуска, в стандартной установке их более 150. Файл **/etc/rc** запускает эти сценарии в порядке, определенном командой **rcorder**, которая считывает сценарии и ищет информацию о зависимостях, которая была закодирована стандартным образом.

Сценарии запуска FreeBSD для разнообразных служб довольно просты. Например, верхняя часть сценария запуска **sshd** выглядит следующим образом:

```
#!/bin/sh
# PROVIDE: sshd
# REQUIRE: LOGIN FILESYSTEMS
# KEYWORD: shutdown
. /etc/rc.subr
name="sshd"
rcvar="sshd_enable"
command="/usr/sbin/${name}"
...
```

Переменная **rcvar** содержит имя переменной, которая должна быть определена в одном из сценариев **rc.conf**, в данном случае **sshd_enable**. Если вы хотите, чтобы для автоматического запуска во время загрузки использовался демон **sshd** (реальный демон, а не сценарий запуска, хотя оба называются **sshd**), поместите в файл **/etc/rc.conf** строку:

```
sshd_enable = "YES"
```

Если эта переменная имеет значение "NO" или закомментирована, сценарий **sshd** не запускает демон или не проверяет, следует ли его останавливать при выключении системы.

Команда **service** обеспечивает интерфейс реального времени в системе **rc.d** операционной системы FreeBSD.[15]

Например, чтобы остановить службу **sshd** вручную, можно запустить команду

```
$ sudo service sshd stop
```

Обратите внимание: этот метод работает только в том случае, если служба указана в файлах **/etc/rc.conf**. Если это не так, используйте подкоманды **onestop**, **onestart** или **onerestart**, в зависимости от того, что вы хотите сделать. (Команда **service**, как правило, напомнит вам об этом, если это необходимо.)

2.9. Процедуры перезагрузки и выключения

Исторически сложилось так, что машины UNIX и Linux очень строго регламентировали процесс выключения. Современные системы стали менее чувствительными, особенно если используется надежная файловая система, но всегда полезно выключить машину, когда это возможно.

---

[14]Для локальных настроек можно либо создать стандартные сценарии в стиле **rc.d**, которые находятся в каталоге **/usr/local/etc/rc.d**, либо редактировать общедоступный сценарий **/etc/rc.local**. Первый вариант предпочтительнее.

[15]Версия службы, которую использует система FreeBSD, основывается на команде **service** в системе Linux, которая управляет традиционными службами `init`.

Операционные системы прошлых лет приучали многих системных администраторов перезагружать систему как первый шаг при решении любой проблемы. Тогда это была адаптивная привычка, но в наши дни она чаще приводит к ненужным затратам времени и прерыванию обслуживания. Сосредоточьтесь на выявлении основной причины проблем, и вы, вероятно, обнаружите, что перезагружаетесь реже.

Тем не менее после изменения сценария запуска или внесения значительных изменений в конфигурацию рекомендуется выполнить перезагрузку. Эта проверка гарантирует успешную загрузку системы. Если вы обнаружили проблему, но не решали ее в течение нескольких недель, то вряд ли вспомните детали последних изменений.

Выключение физических систем

Основные обязанности, необходимые для закрытия системы, выполняет команда **halt**. Она регистрирует выключение, прекращает несущественные процессы, сбрасывает кешированные блоки файловой системы на диск и останавливает ядро. В большинстве систем команда **halt -p** начинает процесс выключения системы.

Команда **reboot** по существу идентична команде **halt**, но она заставляет машину перезагружаться, а не останавливаться.

Команда **shutdown** — это надстройка над командами **halt** и **reboot**, которая обеспечивает запланированные выключения и выдачу тревожных предупреждений для зарегистрированных пользователей. Она восходит к эпохе систем разделения времени и в настоящее время в значительной степени устарела. Команда **shutdown** не делает ничего важного, кроме выполнения команд **halt** или **reboot**, поэтому ее можно не использовать, если у вас нет многопользовательских систем.

Выключение облачных систем

Можно остановить или перезапустить облачную систему либо с сервера (с помощью команды **halt** или **reboot**, как описано в предыдущем разделе), либо с помощью веб-консоли поставщика облачных вычислений (или эквивалентного интерфейса API).

Вообще говоря, отключение системы с помощью облачной веб-консоли сродни отключению питания. Лучше, если виртуальный сервер будет управлять собственным выключением, но если он перестает отвечать на запросы, все что вам остается — отключить его с помощью веб-консоли. Что еще можно сделать?

В любом случае убедитесь, что вы понимаете, что означает выключение с точки зрения поставщика облака. Было бы очень обидно уничтожить вашу систему, когда вы хотели всего лишь перезагрузить ее.

В среде AWS операции **Stop** и **Reboot** делают именно то, что вы ожидаете. Команда **Terminate** деактивирует экземпляр и удаляет его из памяти. Если для базового устройства хранения установлено значение `delete on termination` (удалить при завершении), будет уничтожен не только ваш экземпляр, но и данные на корневом диске. Все хорошо, если вы хотели сделать именно это. Если же вы считаете, что это плохо, можно включить параметр `termination protection` (защита завершения).

2.10. Что делать, если система не грузится?

Множество проблем может помешать загрузке системы, начиная от неисправных устройств и заканчивая обновлениями ядра. Существует три основных подхода к этой ситуации, перечисленные здесь в порядке нежелательности.

- Не отлаживать; просто верните систему в хорошо известное состояние.
- Довести систему до уровня, достаточного для запуска оболочки и отладки в интерактивном режиме.
- Загрузить отдельный системный образ, смонтировать файловые системы неисправной системы и исследовать его.

Первый вариант наиболее часто используется в облаке, но он может быть полезен и на физических серверах, если у вас есть доступ к последнему образу всего загрузочного диска. Если ваша организация делает резервные копии файловой системы, восстановление всей системы может создать больше проблем, чем хотелось бы. Мы обсуждаем вариант восстановления всей системы в режиме восстановления облачных систем немного ниже.

В оставшихся двух подходах основное внимание уделяется предоставлению вам доступа к системе, определению основной проблемы и исправлению любых проблем. Загрузка неисправной системы в оболочку на сегодняшний день является предпочтительным вариантом, но проблемы, которые происходят на ранних этапах загрузки, могут привести к нарушению этого подхода.

Режим "загрузка в оболочку" известен как однопользовательский режим или режим спасения (rescue mode). Системы, которые используют менеджер **systemd**, имеют еще более примитивный вариант, доступный в форме аварийного режима (emergency mode); он концептуально похож на однопользовательский режим, но делает абсолютный минимум работы перед запуском оболочки.

Поскольку однопользовательский режим, режим спасения и аварийный режим не настраивают сеть и не запускают связанные с сетью службы, обычно, чтобы их использовать, необходим физический доступ к консоли. В результате однопользовательский режим обычно недоступен для облачных систем. Мы рассмотрим некоторые варианты восстановления испорченных облачных образов чуть ниже.

Однопользовательский режим

В однопользовательском режиме, также известном как **rescue.target**, для систем, использующих менеджер **systemd**, запускается только минимальный набор процессов, демонов и служб. Корневая файловая система смонтирована (как правило, **/usr**), но сеть остается неинициализированной.

Во время загрузки вы запрашиваете однопользовательский режим, передавая аргумент ядру, обычно **single** или **-s**. Это можно сделать с помощью интерфейса командной строки загрузчика.

В некоторых случаях он может быть настроен автоматически в качестве опции меню загрузки. Если система уже запущена, ее можно перевести в однопользовательский режим с помощью команды **shutdown** (FreeBSD), **telinit** (традиционный **init**) или **systemctl** (**systemd**).

🕮 Дополнительную информацию об учетной записи **root** см. в главе 3.

Системы Sane перед запуском однопользовательской корневой оболочки запрашивают корневой пароль. К сожалению, это означает, что сбросить забытый корневой пароль в однопользовательском режиме практически невозможно. Если вам нужно сбросить пароль, придется получить доступ к диску с помощью отдельного загрузочного носителя.

🕮 Дополнительную информацию о файловых системах и мониторинге см. в главе 5.

Из однопользовательской оболочки можно выполнять команды так же, как при входе в систему с полной загрузкой. Однако иногда монтируется только корневой раздел;

чтобы использовать программы, которые не находятся в каталогах **/bin**, **/sbin** или **/etc**, необходимо смонтировать другие файловые системы вручную.

Часто можно найти указатели на доступные файловые системы, просмотрев каталог **/etc/fstab**. В системе Linux можно выполнить команду **fdisk -l** (буква **L** в нижнем регистре), чтобы просмотреть список разделов диска локальной системы. Аналогичная процедура для системы FreeBSD заключается в том, чтобы выполнить команду **camcontrol devlist** для идентификации дисковых устройств, а затем — команду **fdisk -s** *устройство* для каждого диска.

Во многих однопользовательских средах корневой каталог файловой системы начинает монтироваться в состоянии "только для чтения". Если каталог **/etc** является частью корневой файловой системы (обычный случай), то будет невозможно редактировать многие важные файлы конфигурации. Чтобы устранить эту проблему, необходимо начать сеанс в однопользовательском режиме, перемонтировав каталог в режиме чтения и записи. В системе Linux этот трюк обычно выполняет команда

```
# mount -o rw,remount /
```

В системах FreeBSD возможность повторного подключения является неявной, когда вы повторяете монтирование, но необходимо явно указать исходное устройство. Например,

```
# mount -o rw /dev/gpt/rootfs /
```

RHEL 

Однопользовательский режим в системах Red Hat и CentOS немного более агрессивен, чем обычно. К моменту достижения командной строки эти системы уже пытались подключить все локальные файловые системы. Хотя это умолчание обычно полезно, при несправной файловой системе могут возникнуть проблемы. В этом случае можно загрузиться в аварийном режиме, добавив команду **systemd.unit=emergency.target** к аргументам ядра из загрузчика (обычно GRUB). В этом режиме локальные файловые системы не монтируются и запускается всего несколько основных служб.

🕮 Для получения дополнительной информации о файловых системах и их монтировании см. главу 5.

Команда **fsck** запускается во время обычной загрузки для проверки и восстановления файловых систем. В зависимости от того, какую файловую систему вы используете для корневого каталога, вам может потребоваться запустить **fsck** вручную, когда вы подключаете систему в однопользовательском или аварийном режиме. Для получения более подробной информации о команде **fsck** обратитесь к разделу 20.10.

Однопользовательский режим — это просто точка на обычном пути загрузки, поэтому можно выйти из однопользовательской оболочки, введя команду **exit** или нажав клавиши <Ctrl+D>, чтобы продолжить загрузку. Вы также можете нажать клавиши <Ctrl+D> в момент запроса пароля, чтобы полностью обойти однопользовательский режим.

Однопользовательский режим в системе FreeBSD



Система FreeBSD включает однопользовательский вариант в меню загрузки.

```
1. Boot Multi User [Enter]
2. Boot Single User
3. Escape to loader prompt
4. Reboot
```

```
Options
5. Kernel: default/kernel (1 of 2)
6. Configure Boot Options...
```

Одна приятная особенность однопользовательского режима в FreeBSD заключается в том, что он спрашивает, какую программу использовать в качестве оболочки. Просто нажмите <Enter> для оболочки **/bin/sh**.

Выбрав вариант 3, вы перейдете в среду командной строки загрузочного уровня, реализованную финальным загрузчиком FreeBSD **loader**.

Однопользовательский режим с загрузчиком GRUB



В системах, использующих менеджер **systemd**, можно загрузиться в режиме восстановления, добавив команду **systemd.unit=rescue.target** в конец существующей строки ядра Linux. Чтобы изменить параметры загрузки, выделите нужное ядро на заставке GRUB и нажмите клавишу <e>. Аналогично для загрузки в аварийном режиме используйте команду **systemd.unit=emergency.target**.

Вот пример типичной конфигурации:

```
linux16/vmlinuz-3.10.0-229.el7.x86_64 root=/dev/mapper/rhel_rhel-root
ro crashkernel=auto rd.lvm.lv=rhel_rhel/swap rd.lvm.lv=rhel_rhel/root
rhgb quiet LANG=en_US.UTF-8 systemd.unit=rescue.target
```

Для того чтобы запустить систему после внесения изменений, нажмите <Ctrl+X>.

Восстановление облачных систем

Из-за природы облачных систем невозможно подключить монитор или USB-накопитель при возникновении проблем с загрузкой. Облачные провайдеры делают все возможное, чтобы облегчить решение проблем, но основные ограничения остаются.

🕮 Для более широкого ознакомления с облачными вычислениями см. главу 9.

Резервные копии важны для всех систем, но для облачных серверов сделать такую копию очень просто. Провайдеры взимают дополнительную плату за резервные копии, но они недороги. Чаще создавайте образы (snapshot) и вы всегда будете иметь разумный системный образ, чтобы вернуться к нему в короткие сроки.

С философской точки зрения вы, вероятно, ошибаетесь, если вашим облачным серверам требуется отладка при загрузке. Неисправные физические серверы можно сравнить с домашними животными, которых лечат, если они заболели, но больной крупный рогатый скот не лечат, а просто забивают. Ваши облачные серверы — это крупный рогатый скот; если они плохо работают, замените их заведомо рабочими копиями. Использование этого подхода помогает не только избежать критических сбоев, но и облегчает масштабирование и миграцию системы.

Тем не менее вам неизбежно придется попытаться восстановить облачные системы или диски, поэтому мы кратко обсудим этот процесс ниже.

В рамках AWS однопользовательские и аварийные режимы недоступны. Однако файловые системы EC2 могут быть подключены к другим виртуальным серверам, если они поддерживаются устройствами Elastic Block Storage (EAS). Это значение по умолчанию установлено для большинства экземпляров EC2, поэтому вполне вероятно, что при необходимости вы

сможете использовать этот метод. Понятно, что это похоже на загрузку с USB-накопителя, так что можно сориентироваться на загрузочном диске физической системы.

Вот что надо сделать.

1. Запустите новый экземпляр в области видимости экземпляра, с которым возникли проблемы. В идеале этот экземпляр восстановления должен запускаться с одного и того же базового образа и использовать тот же тип экземпляра, что и неисправная система.
2. Остановите проблематичный экземпляр. (Но будьте осторожны, чтобы не выполнить операцию `terminate`, потому что она удаляет образ загрузочного диска.)
3. С помощью веб-консоли AWS или CLI отсоедините том от проблемной системы и присоедините его к системе восстановления.
4. Войдите в систему восстановления. Создайте точку монтирования и смонтируйте том, а затем сделайте все, что необходимо для устранения проблемы. Затем отключите том. (Не выходит? Убедитесь, что вы не находитесь в одном из каталогов этого тома.)
5. В консоли AWS отсоедините том от экземпляра восстановления и подключите его к проблематичному экземпляру. Запустите проблематичный экземпляр и надейтесь на лучшее.

Дроплеты компании DigitalOcean предлагают консоль с поддержкой VNC, к которой можно получить доступ через Интернет, хотя использовать веб-приложения в некоторых браузерах довольно неудобно. Компания DigitalOcean не предоставляет возможности отсоединить устройства хранения и перенести их в систему восстановления, как это делает Amazon. Вместо этого большинство системных образов позволяют загружаться с альтернативного ядра восстановления.

Чтобы получить доступ к ядру восстановления, сначала отключите дроплеты, а затем смонтируйте ядро восстановления и перезагрузитесь. Если все пойдет хорошо, виртуальный терминал предоставит вам доступ к однопользовательскому режиму. Более подробные инструкции для этого процесса доступны на сайте `digitalocean.com`.

Проблемы с загрузкой в экземпляре Google Compute Engine сначала должны быть исследованы путем изучения информации о последовательном порте экземпляра:

```
$ gcloud compute instances get-serial-port-output экземпляр
```

Такую же информацию можно получить через веб-консоль GCP.

Процесс перемещения диска, аналогичный описанному выше для облака Amazon, также доступен в Google Compute Engine. Вы используете CLI для удаления диска из неработающего экземпляра и загрузки нового экземпляра, который монтирует диск в качестве дополнительной файловой системы. Затем можно запускать проверки файловой системы, изменять параметры загрузки и при необходимости выбирать новое ядро. Этот процесс подробно описан в документации Google по адресу `cloud.google.com/compute/docs/troubleshooting`.

Глава 3

Управление доступом и привилегии суперпользователя



Эта глава посвящена контролю доступа, под которым, в отличие от концепции безопасности, мы подразумеваем механизм принятия решений, связанных с безопасностью, реализуемый ядром и его делегатами. В главе 27, посвященной безопасности, рассматривается более общий вопрос о том, как настроить систему или сеть, чтобы свести к минимуму вероятность нежелательного доступа злоумышленников.

Контроль доступа — это область активных исследований, которая уже давно является одной из главных проблем разработки операционных систем. За последнее десятилетие мир UNIX и Linux пережил кембрийский взрыв новых возможностей в этой области. Первичным драйвером этого всплеска стало появление API-интерфейсов ядра, которые позволяют сторонним модулям увеличивать или заменять традиционную систему управления доступом UNIX. Этот модульный подход создает множество новых возможностей; контроль доступа теперь так же открыт для изменений и экспериментов, как и любой другой аспект UNIX.

Тем не менее традиционная система остается стандартом UNIX и Linux, и она подходит для большинства установок. Даже системные администраторы, которые хотят выйти на новые рубежи, должны овладеть основами.

3.1. Стандартное управление доступом в UNIX

Стандартная модель контроля доступа в системе UNIX в течение десятилетий практически не изменилась. С некоторыми усовершенствованиями она по-прежнему является стандартной для распространенных дистрибутивов операционных систем. Схема соответствует таким основным правилам.

- Решения по управлению доступом зависят от того, какой пользователь пытается выполнить операцию или в некоторых случаях от членства этого пользователя в группе UNIX.
- Объекты (например, файлы и процессы) имеют владельцев. Владельцы имеют широкий (но не обязательно неограниченный) контроль над своими объектами.
- Вы являетесь владельцами создаваемых вами объектов.
- Специальная учетная запись пользователя под названием `root` может действовать как владелец любого объекта.
- Только пользователь `root` может выполнять особо важные административные операции.[1]

Некоторые системные вызовы (например, `settimeofday`) ограничены привилегиями пользователя `root`; реализация просто проверяет подлинность текущего пользователя и отклоняет операцию, если пользователь не является привилегированным (`root`). Другие системные вызовы (например, `kill`) реализуют различные вычисления, которые включают как сравнение прав собственности, так и проверку специальных условий для пользователя `root`. Наконец, файловые системы имеют свои собственные системы управления доступом, которые они реализуют в сотрудничестве с уровнем ядра VFS. Они, как правило, более сложны, чем элементы управления доступом в других местах ядра. Например, файловые системы гораздо чаще используют группы UNIX для контроля доступа.

Усложнение этой картины состоит в том, что ядро и файловая система тесно переплетаются. Например, вы управляете и общаетесь с большинством устройств через файлы, которые представляют их в каталоге `/dev`. Поскольку файлы устройств являются объектами файловой системы, они подчиняются семантике управления доступом к файловой системе. Ядро использует этот факт в качестве основной формы контроля доступа для устройств.

🕮 Дополнительную информацию о файлах устройств см. в разделе 5.4.

Контроль доступа к файловой системе

В стандартной модели каждый файл имеет как владельца, так и группу, иногда называемую "групповым владельцем". Владелец может устанавливать разрешения для файла. В частности, владелец может установить их настолько ограничительно, что никто другой не сможет получить к нему доступ. Мы больше говорим о разрешениях файлов в главе 5 (см. раздел 5.5).

[1] Имейте в виду, что мы здесь описываем оригинальный дизайн системы контроля доступа. В наши дни не все эти утверждения остаются истинными в буквальном смысле. Например, процесс Linux, который имеет соответствующие возможности (см. раздел 3.3), теперь может выполнять некоторые операции, которые ранее были ограничены только компетенцией пользователя `root`.

Хотя владельцем файла всегда является один человек, многие могут быть владельцами файлов, если они все являются частью одной группы. Группы традиционно определяются в файле **/etc/group**, но в настоящее время информация о группе часто сохраняется в системе сетевых баз данных, такой как LDAP; для получения дополнительной информации см. главу 17.

🕮 Для получения дополнительной информации о группах см. раздел 8.5.

Владелец файла определяет, что могут сделать с ним члены групп. Эта схема позволяет делиться файлами между членами одного и того же проекта.

Владельца файла можно определить с помощью команды **ls -l**:

```
$ ls -l ~garth/todo
-rw-r----- 1 garth staff 1259 May 29 19:55 /Users/garth/todo
```

Данный файл принадлежит пользователю **garth** и группе **staff**. Буквы и тире в первом столбце символизируют разрешения на файл; для получения подробной информации о том, как декодировать эту информацию, обратитесь к разделу 5.5. В данном случае коды означают, что пользователь **garth** может читать или записывать файл и что члены группы **staff** могут его прочитать.

🕮 Дополнительную информацию о файлах **passwd** и **group** см. в главе 8.

И ядро, и файловая система отслеживают владельцев и группы как числа, а не как текстовые имена. В подавляющем большинстве случаев идентификационные номера пользователей (короткие идентификаторы UID) сопоставляются с именами пользователей в файле **/etc/passwd**, а идентификационные номера групп (GID) сопоставляются с именами групп в файле **/etc/group**. (Для получения информации о более сложных возможностях см. главу 17.)

Текстовые имена, соответствующие идентификаторам UID и GID, определяются только для удобства пользователей системы. Когда команды, такие как **ls**, должны отображать информацию о владельце в удобочитаемом формате, они ищут каждое имя в соответствующем файле или базе данных.

Владение процессом

Владелец процесса может отправлять сигналы процесса (см. раздел 4.2), а также может понизить (повысить) приоритет планирования процесса. Процессы на самом деле имеют несколько идентификаторов, связанных с ними: реальный, текущий и сохраненный UID; реальный, текущий и сохраненный GID; а в системе Linux — "идентификатор файловой системы", который используется только для определения разрешений доступа к файлам. Вообще говоря, реальные номера используются для учета (и в настоящее время в значительной степени являются рудиментарными), а текущие номера — для определения разрешений доступа. Реальные и текущие номера, как правило, одинаковы.

Сохраненные UID и GID — это парковочные места для идентификаторов, которые в настоящее время не используются, но остаются доступными для вызова из процесса. Сохраненные идентификаторы позволяют программе повторно входить в привилегированный режим работы и выходить из него; эта предосторожность уменьшает риск непреднамеренного неправильного поведения.

Идентификатор UID файловой системы обычно описывается как деталь реализации сетевой файловой системы NFS. Обычно он совпадает с текущим идентификатором UID.

Учетная запись суперпользователя root

Учетная запись **root** — это запись всемогущего административного пользователя UNIX. Она также называется учетной записью суперпользователя, хотя фактическое имя пользователя — **root** (корень).

Определяющей характеристикой учетной записи **root** является ее идентификатор UID, равный 0. Ничто не мешает вам изменять имя пользователя в этой учетной записи или создавать дополнительные учетные записи, идентификаторы UID которых равны 0; однако это плохие идеи.[2] Такие изменения обычно создают непреднамеренные нарушения безопасности системы. Они также создают путаницу, когда другим людям приходится иметь дело со странным способом настройки вашей системы.

🕮 Подробнее о системе NFS см. в главе 21.

Традиционная система UNIX позволяет суперпользователю (т.е. любому процессу, для которого текущий идентификатор UID равен 0) выполнять любую допустимую операцию для любого файла или процесса.[3]

Перечислим некоторые примеры привилегированных операций.

- Создание файлов устройств.
- Установка системных часов.
- Повышение лимитов использования ресурсов и приоритетов процессов.
- Изменение имени хоста системы.
- Настройка сетевых интерфейсов.
- Открытие привилегированных сетевых портов (с номерами меньше 1024).
- Выключение системы.

Примером полномочий суперпользователя является способность принадлежащего ему процесса изменять свои идентификаторы UID и GID. Примером может служить программа входа в систему **login** и ее эквиваленты в виде графического пользовательского интерфейса; процесс, который запрашивает ваш пароль при входе в систему, изначально запускается с правами пользователя **root**. Если введенный вами пароль и имя пользователя являются законными, то программа входа в систему **login** изменяет свои UID и GID на ваши UID и GID и запускает среду оболочки или графический пользовательский интерфейс. Как только процесс **root** изменит своего владельца и станет обычным пользовательским процессом, он не сможет восстановить прежнее привилегированное состояние.

Установка флагов setuid и setgid

Традиционный контроль доступа UNIX дополняется системой подстановки идентификаторов, которая реализована ядром и файловой системой, взаимодействующими между собой. Эта схема позволяет запускать специально выделенные исполняемые файлы с повышенными разрешениями, обычно с правами пользователя **root**. Это дает возможность разработчикам и администраторам создавать структурированные способы, с помощью которых непривилегированные пользователи могут выполнять привилегированные операции.

---

[2]Дженнин Таунсенд, одна из наших рецензентов, прокомментировала это так: "Настолько плохие идеи, что я боюсь даже упоминать их, чтобы кого-нибудь не спровоцировать".

[3]Здесь ключевое слово — "допустимая". Некоторые операции (например, выполнение файла, для которого не установлен бит разрешения выполнения) запрещены даже суперпользователю.

Когда ядро запускает исполняемый файл с установленными битами разрешения `setuid` или `setgid`, оно изменяет текущий идентификатор UID или GID результирующего процесса на UID или GID файла, содержащего образ программы, а не на UID и GID пользователя, который ввел команду. Таким образом, привилегии пользователя распространяются только на выполнение этой конкретной команды.

Например, пользователи должны иметь возможность изменять свои пароли. Однако, поскольку пароли (традиционно) хранятся в защищенном файле **`/etc/master.passwd`** или **`/etc/shadow`**, пользователям нужна команда **`passwd`**, чтобы обеспечить доступ. Команда **`passwd`** проверяет, кто ее запускает, и соответственно настраивает свое поведение: пользователи могут изменять только собственные пароли, но пользователь **`root`** может изменять любой пароль.

Программы, запускаемые с флагом `setuid`, особенно для пользователя **`root`**, могут создавать проблемы безопасности. Команды с установленным флагом `setuid`, поставляемые с системой, теоретически защищены; однако в прошлом уже были обнаружены бреши в системе безопасности и, несомненно, некоторые будут обнаружены в будущем.

Самый верный способ минимизировать количество *проблем*, связанных с флагом `setuid`, — минимизировать количество *программ*, запускаемых с этим флагом. Подумайте дважды, прежде чем использовать программное обеспечение, которое должно устанавливать флаг `setuid`, и избегайте применения флага `setuid` в вашем собственном домашнем программном обеспечении. Никогда не используйте флаг `setuid` в программах, которые не были написаны специально для этого.

Вы можете отключить установку флагов `setuid` и `setgid` в отдельных файловых системах, указав параметр `nosuid` при их монтировании. Рекомендуется использовать этот параметр в файловых системах, которые содержат домашние каталоги пользователей или монтируются из менее надежных административных доменов.

🕮 Дополнительную информацию о возможностях монтирования файловой системы см. в разделе 20.10.

3.2. Управление учетной записью root

Доступ к учетной записи **`root`** необходим для администрирования системы. Кроме того, он представляет собой точку опоры для системы безопасности. Правильное использование этой учетной записи является решающим навыком.

Вход в учетную запись root

Поскольку **`root`** — это просто еще один пользователь, большинство систем позволяют вам напрямую входить в учетную запись **`root`**. Однако это оказывается плохой идеей, поэтому в системе Ubuntu такие действия запрещены по умолчанию.

Во-первых, учетные записи **`root`** не содержат информации о том, какие операции выполнялись с правами **`root`**. Это плохо, когда вы понимаете, что вчера испортили что-то в 3:00 утра и не можете вспомнить, что именно вы изменили; это еще хуже, если доступ был несанкционированным, и вы пытаетесь выяснить, что нарушитель сделал с вашей системой. Другим недостатком является то, что сценарий входа в систему не содержит записей о том, кто на самом деле выполняет данную работу. Если у нескольких пользователей есть доступ к учетной записи **`root`**, вы не сможете определить, кто ее использовал и когда.

По этим причинам большинство систем позволяют отключать вход в учетные записи **`root`** на терминалах, через оконные системы и по всей сети — везде, кроме системной

консоли. Мы предлагаем вам использовать эти возможности. Чтобы узнать, как реализовать эту политику в вашей конкретной системе, см. раздел 17.3.

Если у пользователя **root** есть пароль (т.е. учетная запись **root** не отключена, см. ниже), этот пароль должен быть очень сложным. Для получения дополнительных комментариев относительно выбора пароля см. раздел 27.3.

Команда su: замена идентификатора пользователя

Лучше получать доступ к учетной записи **root** с помощью команды **su**. При вызове без аргументов она выдает приглашение на ввод пароля пользователя **root**, а затем запускает оболочку с привилегированными правами. Эти права будут сохраняться, пока вы не завершите работу (нажав комбинацию клавиш <Ctrl+D> или выполнив команду **exit**). Команда **su** не фиксирует действия, производимые привилегированным пользователем, но добавляет запись в журнальный файл с указанием, кто и когда вошел в систему под именем **root**.

Команда **su** способна также подставлять вместо имени **root** имена других пользователей. Иногда единственный способ помочь пользователю в решении проблемы — войти с помощью команды **su** в его среду и воспроизвести ситуацию.

Зная чей-либо пароль, можно непосредственно зарегистрироваться в системе под его именем, введя команду **su -** *имя_пользователя*. В ответ будет выдан запрос на ввод пароля. При использовании в качестве ключа команды **su** дефиса (-) оболочка перейдет в режим регистрации.

Точная реализация этого режима зависит от конкретной оболочки, но обычно в этом режиме меняются количество и имена файлов запуска, считываемых интерпретатором. Например, в режиме регистрации оболочка **bash** считывает файл **~/.bash_profile**, а вне этого режима — файл **~/.bashrc**. При диагностике проблем конкретного пользователя режим регистрации позволяет максимально точно воспроизвести среду, в которой он работал.

В одних системах пароль **root** позволяет регистрироваться с помощью команд **su** или **login** под любым именем. В других нужно сначала стать пользователем **root**, применив в явном виде команду **su**, а затем с помощью этой же команды перейти в другую учетную запись. В таком случае вводить пароль не потребуется.

Рекомендуем сделать правилом при вводе команды указывать полное имя, например **/bin/su** или **/usr/bin/su**, а не просто **su**. Это послужит определенной защитой от тех программ с именем **su**, которые преднамеренно были изменены злоумышленником, планировавшим собрать хороший "урожай" паролей[4].

В некоторых системах, чтобы использовать команду **su**, необходимо быть членом группы wheel.

Команду **su** в значительной степени может заменить программа **sudo** (см. следующий раздел), саму же команду **su** лучше всего оставить для аварийных ситуаций.

Программа sudo: ограниченный вариант команды su

Без одной из усовершенствованных систем управления доступом, описанных в разделе 3.4, трудно разрешить кому-то выполнять какую-то задачу (например, резервные

[4]По аналогичной причине настоятельно рекомендуем не включать запись "." (текущий каталог) в переменную **path** (которую можно увидеть, набрав команду **echo $PATH**). Несмотря на очевидное удобство, подобная конфигурация приводит к тому, что можно непреднамеренно запустить "специальную" версию какой-нибудь системной команды, которую злоумышленник оставил в качестве приманки. Пользователю **root** следует быть бдительным вдвойне.

копии), не предоставляя этому пользователю права свободно запускать систему. И если учетная запись **root** используется несколькими администраторами, на самом деле у вас будет только смутное представление о том, кто ее использует или что они сделали.

Наиболее широко используемым решением этих проблем является программа под названием **sudo**, которая в настоящее время поддерживается Тоддом Миллером (Todd Miller). Эта программа работает на всех наших иллюстративных системах и также доступна в форме исходного кода на сайте `sudo.ws`. Мы рекомендуем использовать ее в качестве основного метода доступа к учетной записи **root**.

Программа **sudo** принимает в качестве аргумента командную строку, выполняемую с правами **root** (или правами любого другого ограниченного пользователя). Программа **sudo** проверяет файл **/etc/sudoers** (**/usr/local/etc/sudoers** в системе FreeBSD), в котором перечислены люди, имеющие разрешение использовать программу **sudo** и команды, которые им можно запускать на каждом хосте. Если предлагаемая команда разрешена, программа **sudo** запрашивает собственный пароль пользователя и выполняет команду.

В течение пяти минут после запуска программа **sudo** позволяет выполнять команды без обязательного введения пароля (продолжительность этого периода можно менять), после чего программа **sudo** перейдет в режим ожидания. Этот тайм-аут служит скромной защитой от пользователей с привилегиями **sudo**, которые оставляют терминалы без присмотра.

Программа **sudo** ведет журнал выполненных командных строк, регистрирует хосты, на которых они выполнялись, людей, которые их выполняли, каталоги, из которых они были выполнены, и время, в которое они были выполнены. Эта информация может быть зарегистрирована в системном журнале или помещена в файл по вашему выбору. Мы рекомендуем использовать механизм Syslog для пересылки записей журнала на безопасный центральный сервер.

Запись журнала для выполнения команды **sudo /bin/cat /etc/sudoers** пользователя `randy` может выглядеть так:

```
Dec 7 10:57:19 tigger sudo: randy: TTY=ttyp0 ; PWD=/tigger/users/randy;
   USER=root ; COMMAND=/bin/cat /etc/sudoers
```

Пример конфигурации

Файл **sudoers** разработан таким образом, что одна версия может использоваться сразу на нескольких разных хостах. Вот типичный пример.

```
# Define aliases for machines in CS & Physics departments
Host_Alias  CS = tigger, anchor, piper, moet, sigi
Host_Alias  PHYSICS = eprince, pprince, icarus

# Define collections of commands
Cmnd_Alias  DUMP = /sbin/dump, /sbin/restore
Cmnd_Alias  WATCHDOG = /usr/local/bin/watchdog
Cmnd_Alias  SHELLS = /bin/sh, /bin/dash, /bin/bash

# Permissions
mark, ed    PHYSICS = ALL
herb        CS = /usr/sbin/tcpdump : PHYSICS = (operator) DUMP
lynda       ALL = (ALL) ALL, !SHELLS
%wheel      ALL, !PHYSICS = NOPASSWD: WATCHDOG
```

Первые два набора строк определяют группы хостов и команды, которые упоминаются в дальнейших спецификациях разрешений, включенных в файл. Списки могут быть включены буквально в спецификации, но псевдонимы облегчают чтение и понимание файла **sudoers**; они также упрощают обновление файла в будущем. Также воз-

можно определить псевдонимы для пользователей и групп пользователей, для которых могут выполняться команды.

🕮 Дополнительную информацию о файле **syslog** см. в главе 10.

Каждая строка спецификации разрешений содержит следующую информацию.

- Пользователи, к которым относится запись.
- Хосты, на которые распространяется эта запись.
- Команды, которые могут выполняться указанными пользователями.
- Пользователи, которым разрешается выполнять команды.

Первая строка разрешений применяется к пользователям `mark` и `ed` на машинах в группе `PHYSICS` (`eprince`, `pprince` и `icarus`). Встроенный командный псевдоним `ALL` позволяет им выполнять любую команду. Поскольку в круглых скобках не указан список пользователей, программа **sudo** будет запускать команды с правами **root**.

Вторая строка разрешений позволяет пользователю `herb` запускать утилиту **tcpdump** на машинах группы `CS` и команды вывода дампа на машинах группы `PHYSICS`. Однако команды вывода дампа могут выполняться только с правами пользователя `operator`, а не суперпользователя. Реальная командная строка, которую должен был бы ввести пользователь `herb`, выглядела бы так:

```
ubuntu$ sudo -u operator /usr/sbin/ dump 0u /dev/sda1
```

Пользователь `lynda` может запускать команды как любой пользователь на любой машине, за исключением того, что она не может запускать несколько общих оболочек. Означает ли это, что пользователь `lynda` действительно не имеет доступа к корневой оболочке? Конечно, нет:

```
ubuntu$ cp -p /bin/sh/tmp/sh
ubuntu$ sudo /tmp/sh
```

Вообще говоря, любая попытка разрешить "все команды, кроме..." обречена на провал, по крайней мере в техническом смысле. Тем не менее может оказаться целесообразным настроить файл **sudoers** таким образом, чтобы напомнить о том, что вызывать оболочку с правами **root** крайне нежелательно.

Последняя строка позволяет пользователям группы `wheel` выполнять локальную команду **watchdog** с правами **root** на всех машинах, кроме `eprince`, `pprince` и `icarus`. Кроме того, для выполнения этой команды не требуется пароль.

Обратите внимание на то, что команды в файле **sudoers** указаны с полными именами путей, чтобы люди не могли выполнять свои собственные программы и сценарии с правами **root**. Хотя в вышеприведенном примере это не показано, можно указать аргументы, допустимые для каждой команды.

Чтобы вручную изменить файл **sudoers**, используйте команду **visudo**, которая проверяет, что никто другой не редактирует файл, вызывает для него редактор (**vi** или любой редактор, указанный вами в переменной среды `EDITOR`), а затем проверяет синтаксис отредактированного файла перед его установкой. Этот последний шаг особенно важен, потому что неправильный файл **sudoers** может помешать вам снова исправить ошибку.

Преимущества и недостатки программы sudo

Использование программы **sudo** имеет следующие преимущества.

- Благодаря ведению журнала команд значительно улучшается учет.
- Пользователи могут выполнять определенные задачи, не имея неограниченных прав **root**.

- Реальный пароль **root** может быть известен только одному или двум людям.[5]
- Использование программы **sudo** быстрее, чем вызов **su** или вход в систему с правами **root**.
- Привилегии можно отменить без необходимости изменения пароля пользователя **root**.
- Поддерживается канонический список всех пользователей с правами **root**.
- Уменьшается вероятность того, что корневая оболочка останется без присмотра.
- Один файл может управлять доступом ко всей сети.

У программы **sudo** есть и несколько недостатков. Хуже всего то, что любое нарушение безопасности личной учетной записи **sudoer** может быть эквивалентно нарушению самой учетной записи **root**. Вы не можете сделать многого, чтобы противостоять этой угрозе, кроме как предостеречь пользователей, перечисленных в файле **sudoers**, чтобы они защитили свои учетные записи, поскольку они будут иметь учетную запись **root**. Вы также можете регулярно запускать взломщик паролей для пользователей, перечисленных в **sudoers**, чтобы убедиться, что они выбрали правильные пароли. Все комментарии по выбору пароля см. в разделе 27.3.

Регистрацию команд в программе **sudo** можно легко обойти с помощью таких трюков, как временный выход в оболочку из разрешенной программы, или команды **sudo sh** и **sudo su**. (Такие команды отображаются в журналах, поэтому вы по крайней мере узнаете, что они были запущены.)

Программа sudo и расширенный контроль доступа

Программе **sudo** как способ разделения привилегий корневой учетной записи превосходит многие из систем управления доступом, описанных в разделе 3.4.

- Вы точно определяете, как будут разделены привилегии. Ваше подразделение может быть более грубым или более тонким, чем привилегии, определенные для вас с помощью готовой системы.
- Простые конфигурации — самые распространенные — просты в настройке, обслуживании и понимании.
- Программа **sudo** работает на всех системах UNIX и Linux. Вам не нужно беспокоиться об управлении различными решениями на разных платформах.
- Вы можете использовать один файл конфигурации во всей организации.
- Вы бесплатно получаете качественные протоколы.

Поскольку система уязвима для катастрофической компрометации, если корневая учетная запись будет взломана, основным недостатком контроля доступа на основе программы **sudo** является то, что потенциальная поверхность атаки расширяется и распространяется на учетные записи всех администраторов.

Программа **sudo** хорошо работает как инструмент благонамеренных администраторов, которым необходим общий доступ к привилегиям **root**. Она также отлично подходит для того, чтобы остальные пользователи могли выполнить несколько конкретных операций. Несмотря на синтаксис конфигурации, который предполагает иное, программа **sudo**, к сожалению, не является безопасным способом определить ограниченные области автономии или вынести определенные операции за пределы безопасной области.

🕮 Для получения дополнительной информации о взломе пароля см. раздел 27.5.

[5]Или даже никому, если у вас есть правильная система хранения паролей.

Даже не пытайтесь использовать эти конфигурации. Если вам нужна эта функциональная возможность, лучше всего подключить одну из систем управления доступом, описанных в разделе 3.4.

Типичная настройка

За многие годы система конфигурации **sudo** накопила много функций. Она была модифицирована, чтобы учесть множество необычных ситуаций и крайних случаев. В результате текущая документация создает впечатление сложности, которое не обязательно оправданно.

Поскольку важно, чтобы программа **sudo** была надежным и безопасным инструментом, естественно задаться вопросом, можете ли вы подвергать свои системы дополнительному риску, если не используете ее расширенные функции и не устанавливаете правильные значения для всех параметров. Ответ: нет.

90% содержимого файла **sudoers** выглядит примерно так:

```
User_Alias    ADMINS = alice, bob, charles
ADMINS        ALL = (ALL) ALL
```

Это совершенно респектабельная конфигурация, и во многих случаях нет необходимости ее усложнять. Мы упоминали несколько дополнительных функций, которые вы можете использовать в следующих разделах, но все они являются инструментами для решения проблем, которые полезны для конкретных ситуаций. Для общей надежности больше ничего не требуется.

Управление окружением

Многие команды проверяют значения переменных окружения и изменяют свое поведение в зависимости от того, что они находят. В случае, если команды выполняются с правами **root**, этот механизм может быть полезным и многообещающим способом атаки.

Например, допустим, что несколько команд запускают программу, указанную в переменной окружения `EDITOR`, для редактирования текстового файла. Если эта переменная указывает на вредоносную программу хакера вместо текстового редактора, вполне вероятно, что в конечном итоге вы запустите эту программу с правами **root**.[6]

Чтобы минимизировать этот риск, поведение программы **sudo** по умолчанию заключается в передаче только минимального, дезинфицированного множества переменных окружения для команд, которые она запускает. Если вам нужны дополнительные переменные окружения, которые нужно передать, вы можете их перечислить, добавив в список `env_keep` файла **sudoers**. Например, строки

```
Defaults    env_keep += "SSH_AUTH_SOCK"
Defaults    env_keep += "DISPLAY XAUTHORIZATION XAUTHORITY"
```

сохраняют несколько переменных окружения, используемых X Windows и механизмом пересылки ключей SSH.

Для разных пользователей или групп можно настроить разные списки `env_keep`, но конфигурация быстро усложнится. Мы предлагаем придерживаться единого универсального списка и быть относительно консервативным с исключениями, которые вы закрепляете в файле **sudoers**.

---

[6] Поясним, что сценарий в этом случае заключается в том, что ваша учетная запись была скомпрометирована, но злоумышленник не знает ваш фактический пароль и поэтому не может напрямую запускать программу **sudo**. К сожалению, это обычная ситуация, все, что требуется, — это терминальное окно, оставленное на мгновение без присмотра.

Если необходимо сохранить переменную среды, которая не указана в файле **sudoers**, ее можно явно установить в командной строке **sudo**. Например, команда

```
$ sudo EDITOR=emacs vipw
```

редактирует файл системного пароля с помощью редактора **emacs**. Эта функция имеет некоторые потенциальные ограничения, но они не распространяются на пользователей, которые могут запускать все команды.

Программа sudo без паролей

Часто можно с сожалением видеть, что программа **sudo** настроена на выполнение команды с правами **root** без необходимости вводить пароль. Для справки, эта конфигурация задается с помощью ключевого слова NOPASSWD в файле **sudoers**. Например:

```
ansible      ALL = (ALL) NOPASSWD: ALL      # Не делайте этого
```

Иногда это делается из-за лени, но, как правило, основная причина — желание выполнить какие-то операции без санкции программы **sudo**. Наиболее распространенными случаями являются удаленная настройка с помощью такой системы, как Ansible, или запуск команд из планировщика **cron**.

🕮 Дополнительную информацию о системе Ansible см. в главе 23.

Излишне говорить, что эта конфигурация опасна, поэтому избегайте ее, если можете. По крайней мере ограничьте выполнение небезопасного доступа к определенному набору команд, если сможете.

Другим вариантом, который хорошо работает в контексте удаленного выполнения, является замена введенных вручную паролей аутентификации с помощью программы **ssh-agent** и перенаправленных SSH-ключей. Вы можете настроить этот метод аутентификации с помощью модуля PAM на сервере, где будет выполняться программа **sudo**.

Большинство систем не включают модуль PAM, который по умолчанию использует аутентификацию на основе SSH, но он легко доступен. Найдите пакет pam_ssh_agent_auth.

Пересылка ключей SSH имеет собственный набор проблем безопасности, но это, безусловно, лучше, чем полное отсутствие аутентификации.

Приоритет

Вызов программы **sudo** потенциально может быть рассмотрен несколькими записями в файле **sudoers**. Например, рассмотрим следующую конфигурацию:

```
User_Alias    ADMINS = alice, bob, charles
User_Alias    MYSQL_ADMINS = alice, bob

%wheel        ALL = (ALL) ALL
MYSQL_ADMINS  ALL = (mysql) NOPASSWD: ALL
ADMINS        ALL = (ALL) NOPASSWD: /usr/sbin/logrotate
```

🕮 Дополнительную информацию о конфигурации PAM см. в разделе 17.3.

Здесь администраторы могут запускать команду **logrotate** как обычный пользователь без предоставления пароля. Администраторы MySQL могут выполнять любую команду от имени пользователя **mysql** без пароля. Любой человек в группе wheel может выполнять любую команду под любым идентификатором UID, но сначала он должен пройти аутентификацию с помощью пароля.

Если пользователь находится в группе `wheel`, она потенциально охватывается каждой из последних трех строк. Откуда вы знаете, какая из них будет определять поведение **sudo**?

Правило заключается в том, что **sudo** всегда подчиняется последней соответствующей строке, причем совпадение определяется всеми четырьмя параметрами: именем пользователя, хостом, именем целевого пользователя и командой. Каждый из этих элементов должен соответствовать строке конфигурации, или строка просто игнорируется.

Поэтому исключения `NOPASSWD` должны соответствовать их более общим аналогам, как показано выше. Если бы порядок последних трех строк был отменен, то бедной Алисе[7] пришлось бы вводить пароль независимо от того, какую команду **sudo** она пыталась бы запустить.

Программа sudo без терминала управления

Помимо проблемы аутентификации без пароля, автоматическое выполнение программы **sudo** (например, из планировщика **cron**) часто происходит без обычного терминала управления. В этом нет ничего неправильного, но это странная ситуация, которую программа **sudo** может обнаружить и отклонить, если в файле **sudoers** включена опция `requiretty`.

Этот параметр не задается по умолчанию с точки зрения программы **sudo**, но некоторые дистрибутивы операционных систем включают его в файлы **sudoers** по умолчанию, поэтому его стоит проверить и удалить. Найдите строку, имеющую вид

```
Defaults   requiretty
```

и поменяйте в ней значение на противоположное:

```
Defaults  !requiretty
```

Опция `requiretty` предлагает небольшую символическую защиту от определенных сценариев атаки. Тем не менее ее легко обойти, таким образом, она дает мало реальных преимуществ в плане безопасности. По нашему мнению, опцию `requiretty` необходимо отключить, потому что это общий источник проблем.

Конфигурация sudo на уровне сайта

Поскольку файл **sudoers** включает текущий хост в качестве подходящего критерия для строк конфигурации, вы можете использовать один главный файл **sudoers** в административном домене (т.е. в области, где гарантируется эквивалентность имен хостов и учетных записей пользователей). Этот подход немного усложняет исходные настройки **sudoers**, но это отличная идея по нескольким причинам. Вам следует это попробовать.

Основное преимущество этого подхода заключается в том, что нет никакой тайны в том, кто и какие имеет разрешения и для каких хостов. Все записано в одном авторитетном файле. Например, когда администратор покидает вашу организацию, нет необходимости отслеживать все хосты, на которых у этого пользователя могли быть права **sudo**. Когда необходимы изменения, вы просто изменяете главный файл **sudoers** и рассылаете его заново.

Естественным следствием этого подхода является то, что разрешения **sudo** могут быть лучше выражены в терминах учетных записей пользователей, а не групп UNIX. Например, запись

```
%wheel      ALL = ALL
```

[7] Традиционный персонаж в описаниях криптографических протоколов. — *Примеч. ред.*

имеет некоторую интуитивную привлекательность, но она отменяет перечисление привилегированных пользователей на каждой локальной машине. Вы не можете взглянуть на эту строку и определить, к кому она относится, не подходя к рассматриваемой машине. Поскольку идея состоит в том, чтобы сохранить всю соответствующую информацию в одном месте, лучше избегать такого типа группировки при совместном использовании файла **sudoers** в сети. Конечно, если членство в вашей группе тесно координируется по всей организации, группы можно использовать без проблем.

Распределение файла **sudoers** лучше всего достигается с помощью более широкой системы управления конфигурацией, как описано в главе 23. Однако если вы еще не достигли такого уровня, вы можете легко разослать его самостоятельно. Однако будьте осторожны: установка поддельного файла **sudoers** — это быстрый путь к катастрофе. Это также удобный файл для использования в системе мониторинга целостности файлов; см. раздел 28.8.

В случае отсутствия системы управления конфигурацией лучше всего использовать сценарий извлечения информации, который заканчивается планировщиком **cron** на каждом хосте. Используйте команду **scp** для копирования текущего файла **sudoers** из известного центрального репозитория, а затем проверьте его с помощью команды **visudo -c -f** *newsudoers* перед установкой, чтобы убедиться, что формат является приемлемым для локальной программы **sudo**. Команда **scp** проверяет ключ хоста удаленного сервера, гарантируя, что файл **sudoers** поступает с вашего хоста, а не с поддельного сервера.

При совместном использовании файла **sudoers** спецификация имени хоста может быть немного сложной. По умолчанию программа **sudo** использует вывод команды **hostname** в качестве текста, который нужно сравнить. В зависимости от соглашений, используемых в вашей организации, это имя может включать или не включать часть домена (например, `anchor` или `anchor.cs.colorado.edu`). В любом случае имена хостов, указанные в файле **sudoers**, должны совпадать с именами, которые возвращают все хосты. (Вы можете включить опцию `fqdn` в файле **sudoers**, чтобы попытаться преобразовать локальные имена хостов в их полные формы.)

Совпадение имен хостов сложнее проверяется в облаке, где имена экземпляров часто по умолчанию имеют алгоритмически сгенерированные шаблоны. Программа **sudo** понимает простые символы шаблонов соответствия (подстановки) в именах хостов, поэтому рассмотрите возможность использования схемы именования, которая включает в себя некоторые указания на классификацию безопасности каждого хоста с точки зрения программы **sudo**.

Кроме того, можно использовать виртуальные сетевые функции облачного провайдера для разделения хостов по IP-адресу, а затем сопоставлять IP-адреса вместо имен хостов из файла **sudoers**.

Отключение учетной записи root

Если ваша организация стандартизирует использование программы **sudo**, вы вряд ли будете использовать реальные пароли **root**. Большая часть вашей административной команды никогда не будет иметь возможности использовать их.

Этот факт ставит вопрос о том, нужен ли пароль **root** вообще. Если вы решите, что не нужен, вы можете полностью отключить вход в систему с правами **root**, установив зашифрованный пароль **root** равным * или другой произвольной строке фиксированной длины. В системе Linux, команда **passwd -l** блокирует учетную запись, добавляя знак ! к зашифрованному паролю с эквивалентными результатами.

Символы * и ! — просто условности; никакое программное обеспечение не проверяет их явным образом. Их эффект заключается в задании некорректных хешей паролей. В результате пароль **root** просто не проходит проверку.

Основной эффект блокировки учетной записи **root** заключается в том, что пользователь **root** не может войти в систему, даже с консоли. Ни один из пользователей не может успешно запустить программу **su**, потому что для этого также требуется проверка пароля **root**. Однако учетная запись **root** продолжает существовать, и все программное обеспечение, которое обычно выполняется с правами **root**, продолжает это делать. В частности, программа **sudo** работает как обычно.

Основное преимущество отключения учетной записи **root** заключается в том, что вам не нужно записывать пароль **root** и управлять им. Вы также устраняете возможность взлома пароля **root**, но это более приятный побочный эффект, чем убедительная причина для перехода без пароля. Редко используемые пароли уже имеют низкий риск компрометации.

Особенно полезно иметь реальный пароль **root** на физических компьютерах (в отличие от облачных или виртуальных экземпляров, см. главы 9 и 24). Если проблемы с оборудованием или конфигурацией мешают процессу **sudo** или загрузке, реальным компьютерам необходимо будет уделить внимание. В этих случаях удобно иметь традиционную учетную запись **root** как аварийный резерв.



Система Ubuntu поставляется с отключенной учетной записью **root**, и весь административный доступ осуществляется через программу **sudo** или ее эквивалент с графическим пользовательским интерфейсом. Если хотите, можете установить пароль **root** на Ubuntu, а затем разблокировать учетную запись с помощью команды **sudo passwd -u root**.

Системные учетные записи, отличные от root

Пользователь **root** обычно является единственным, кто имеет особый статус с точки зрения ядра, но в большинстве систем существуют и другие псевдопользователи. Вы можете идентифицировать эти фиктивные учетные записи по их идентификаторам UID, которые обычно не превышают 100. Чаще всего идентификаторы UID менее 10 относятся к системным учетным записям, а UID от 10 до 100 — к псевдопользователям, связанным с определенными компонентами программного обеспечения.

Обычно принято заменять звездочкой зашифрованное поле пароля этих специальных пользователей в файле **shadow** или **master.passwd**, чтобы под их учетными записями нельзя было войти в систему. В качестве их системных оболочек должны быть установлены программы **/bin/false** или **/bin/nologin**, чтобы защитить от удаленных попыток входа в систему, которые используют альтернативы паролей, такие как файлы ключей SSH.

Как и в случае с учетными записями пользователей, в большинстве систем определяются множество связанных с системой групп, которые имеют одинаково низкие идентификаторы GID.

🕮 Дополнительную информацию о файлах **shadow** и **master.passwd** см. в разделе 8.3.

Файлам и процессам, являющимся частью операционной системы, которые не должны принадлежать к категории **root**, иногда в качестве владельцев назначаются пользователи **bin** или **daemon**. Теория заключалась в том, что это соглашение поможет избежать угроз безопасности, связанных с владением правами **root**. Однако это не слишком веский аргумент, и в современных системах часто используется только учетная запись **root**.

Основное преимущество псевдоучетных записей и псевдогрупп заключается в том, что с их помощью можно безопаснее обеспечить доступ к определенным группам ресурсов, чем с помощью учетной записи **root**. Например, базы данных часто реализуют сложные системы управления доступом. С точки зрения ядра они работают как псевдопользователь, такой как `mysql`, который владеет всеми ресурсами, связанными с базой данных.

Сетевая файловая система (NFS) использует учетную запись `nobody` для представления пользователей **root** в других системах. Для удаленных суперпользователей, которые должны быть лишены своих привилегированных полномочий, удаленный UID, равный 0, должен быть сопоставлен с чем-то другим, кроме локального UID, равного 0. Учетная запись `nobody` действует как общее имя для этих удаленных суперпользователей. В системе NFSv4 учетная запись `nobody` может применяться к удаленным пользователям, которые также не соответствуют действительной локальной учетной записи.

🕮 Дополнительную информацию об учетной записи `nobody` см. в разделе 21.2.

Поскольку учетная запись `nobody` должна представлять обобщенного и относительно бесправного пользователя, она не должна владеть файлами. Если учетная запись `nobody` владеет файлами, значит, удаленные суперпользователи могут их контролировать.

3.3. Расширения стандартной модели контроля доступа

В предыдущих разделах описаны основные концепции традиционной модели управления доступом. Несмотря на то, что эту модель можно описать на нескольких страницах, она выдержала испытание временем, потому что проста, предсказуема и способна обрабатывать требования средней организации. Все варианты UNIX и Linux продолжают поддерживать эту модель, и она по-прежнему остается стандартной и наиболее широко распространенной.

В современных операционных системах эта модель включает в себя ряд важных усовершенствований. Текущий статус-кво обеспечивают три уровня программного обеспечения.

- Стандартная модель, описанная в этом пункте.
- Расширения, которые обобщают и точно настраивают базовую модель.
- Расширения ядра, которые реализуют альтернативные подходы.

Эти категории являются не архитектурными слоями, а скорее историческими артефактами. Ранние производные системы UNIX использовали стандартную модель, но ее недостатки были широко признаны даже тогда. Со временем сообщество начало разрабатывать обходные пути для решения нескольких более насущных проблем. В интересах поддержания совместимости и поощрения широкого распространения изменения обычно были структурированы как усовершенствования традиционной системы. Некоторые из этих настроек (например, PAM) теперь считаются стандартами UNIX.

За последнее десятилетие в области модуляризации систем контроля доступа были достигнуты большие успехи. Эта эволюция позволяет еще более радикально изменить контроль доступа. Мы рассмотрим некоторые из наиболее распространенных подключаемых параметров для Linux и FreeBSD, начиная с раздела 3.4.

Пока мы рассмотрим некоторые из более прозаических расширений, которые связаны с большинством систем. Во-первых, мы рассмотрим проблемы, которые пытаются решить эти расширения.

Недостатки стандартной модели

Несмотря на свою элегантность, стандартная модель имеет некоторые очевидные недостатки.

- Начнем с того, что учетная запись **root** представляет собой потенциально единственную точку отказа. Если она будет скомпрометирована, целостность всей системы будет нарушена, и, по существу, нет ограничений на ущерб, который может нанести злоумышленник.
- Единственный способ разделить привилегии учетной записи **root** — писать программы с флагом `setuid`. К сожалению, как показывает постоянный поток обновлений программного обеспечения, связанных с безопасностью, написать безопасное программное обеспечение сложно. Любая программа, использующая флаг `setuid`, является потенциальной целью.
- Стандартная модель мало что может сказать о безопасности в сети. Нет компьютера, к которому непривилегированный пользователь имел бы физический доступ и которому можно было бы доверять, чтобы точно представлять владельцев запускаемых процессов. Кто скажет, что кто-то не переформатировал диск и не установил свою собственную операционную систему с идентификатором UID по своему выбору?
- В стандартной модели определение группы является привилегированной операцией. Например, универсальный пользователь не может выразить намерение, что только пользователи **alice** и **bob** должны иметь доступ к определенному файлу.
- Поскольку многие правила управления доступом встроены в код отдельных команд и демонов (классический пример — программа **passwd**), невозможно переопределить поведение системы без изменения исходного кода и перекомпиляции. В реальном мире это непрактично и провоцирует ошибки.
- Стандартная модель также практически не поддерживает аудит или ведение журнала. Вы можете видеть, к каким группам UNIX принадлежит пользователь, но не можете определить, что такое членство в группах позволяет пользователю делать. Кроме того, нет реального способа отслеживать использование повышенных привилегий или посмотреть, какие операции они выполнили.

PAM: подключаемые модули аутентификации

Учетные записи пользователей традиционно защищены паролями, хранящимися (в зашифрованном виде) в файлах **/etc/shadow** или **/etc/master.passwd** или в эквивалентной сетевой базе данных.

Многим программам может потребоваться проверка учетных записей, включая **login**, **sudo**, **su** и любую программу, которая требует регистрации на рабочей станции с графическим пользовательским интерфейсом.

🕮 Дополнительную информацию о файлах **shadow** и **master.passwd** см. в разделе 8.3.

Эти программы действительно не должны иметь жестко запрограммированных ожиданий о том, как должны шифроваться и проверяться пароли. В идеале они даже

не должны предполагать, что пароли вообще используются. Что делать, если вы хотите использовать биометрическую идентификацию, систему идентификации сети или какую-либо двухфакторную аутентификацию? На помощь приходят подключаемые модули аутентификации PAM!

PAM (Pluggable Autentication Modules) — это оболочка для различных библиотек аутентификации, реализующих разные методы. Системные администраторы определяют методы аутентификации, которые они хотят использовать в системе, а также соответствующие контексты для каждого из них. Программы, требующие аутентификации пользователя, просто вызывают систему PAM, а не реализуют собственные методы аутентификации.

В свою очередь PAM вызывает библиотеку аутентификации, указанную системным администратором. Строго говоря, PAM — это технология аутентификации, а не технология контроля доступа. Следовательно, вместо ответа на вопрос "Имеет ли пользователь X разрешение на выполнение операции Y?", она помогает ответить на вопрос "Как узнать, что это действительно пользователь X?"

PAM является важным компонентом цепочки контроля доступа в большинстве систем, а конфигурация PAM является общей административной задачей. Более подробную информацию о PAM вы можете найти в разделе 17.3.

Kerberos: сетевая криптографическая аутентификация

Как и PAM, протокол Kerberos занимается аутентификацией, а не контролем доступа как таковым. Однако, если PAM является основой проверки подлинности, Kerberos является специфическим методом аутентификации. В организациях, которые используют Kerberos, PAM и Kerberos, как правило, работают вместе, PAM — это оболочка, а Kerberos — фактическая реализация.

Протокол Kerberos использует доверенную стороннюю организацию (сервер) для аутентификации всей сети. Вы не аутентифицируете себя на машине, которую используете, а предоставляете свои учетные данные службе Kerberos. Затем Kerberos выдает криптографические данные, которые вы можете представить другим службам в качестве доказательства вашей подлинности.

Kerberos — это зрелая технология, которая широко используется десятилетиями. Это стандартная система аутентификации, используемая системой Windows, которая является частью системы Active Directory Microsoft. Больше о технологии Kerberos написано в разделе 17.3.

Списки управления доступом к файловой системе

Поскольку управление доступом к файловой системе крайне важно для систем UNIX и Linux, это одна из главных целей их разработки. Наиболее распространенным дополнением была поддержка списков управления доступом (access control lists — ACL), обобщение традиционной модели прав пользователь/группа/другое, которая позволяет устанавливать разрешения для нескольких пользователей и групп одновременно.

Списки ACL являются частью реализации файловой системы, поэтому они должны явно поддерживаться любой файловой системой, которую вы используете. Тем не менее теперь все основные файловые системы UNIX и Linux в той или иной форме поддерживают ACL.

Поддержка ACL обычно осуществляется в одной из двух форм: ранний вариант стандарта POSIX, который так и не дошел до формального принятия, но был широко реа-

лизован в любом случае, и система, стандартизованная NFSv4, которая адаптирует ACL Microsoft Windows. Оба стандарта ACL описаны более подробно в разделе 5.6.

🕮 Дополнительную информацию об NFS см. в главе 21.

Возможности Linux



Системы мандатов (capability systems) делят полномочия учетной записи **root** на несколько (около 30) отдельных разрешений.

Версия системы мандатов в Linux основана на черновике проекта POSIX 1003.1e, который используется, несмотря на то, что он официально не был одобрен в качестве стандарта. Помимо прочего, теоретики считают, что эта зомби-система не соответствует академической концепции системы мандатов. Однако это не важно; система существует, и Linux называет ее мандатной, поэтому мы подчиняемся.

Мандаты могут быть унаследованы от родительского процесса. Они также могут быть включены или отключены атрибутами, установленными в исполняемом файле, в процессе, напоминающем выполнение программы с флагом `setuid`. Процессы могут отказаться от мандатов, которые они не планируют использовать.

Традиционные полномочия **root** — это просто объединение всех мандатов, поэтому существует довольно прямое сопоставление между традиционной моделью и мандатной. Мандатная модель является более детальной.

В качестве примера укажем, что мандат Linux под названием `CAP_NET_BIND_SERVICE` контролирует возможность процесса связывания с привилегированными сетевыми портами (номера которых меньше 1024). Некоторым демонам, которые традиционно работают с правами **root**, нужен только этот конкретный мандат. В системе мандатов такой демон может теоретически работать как непривилегированный пользователь и получать мандат на возможность привязки к порту из исполняемого файла. Пока демон не проверяет явно, что он работает от имени пользователя **root**, он даже не должен знать о нем.

Так ли все это в реальном мире? Совсем нет. Как это обычно бывает, мандаты эволюционировали и стали более мощной технологией, чем система взаимодействия с пользователями. Они широко используются системами более высокого уровня, такими как AppArmor (см. раздел 3.4) и Docker (см. главу 25), но редко используются самостоятельно.

Для администраторов полезно просмотреть man-страницу **capabilities(7)**, чтобы понять, что включено в каждую из категорий мандатов.

Пространства имен Linux



Система Linux может распределять процессы по иерархическим разделам (пространствам имен), из которых видна только часть системных файлов, сетевых портов и процессов. Помимо прочего, эта схема действует как форма превентивного контроля доступа. Вместо того, чтобы основывать решения управления доступом на потенциально тонких критериях, ядро просто отрицает существование объектов, которые не видны изнутри данной области.

Внутри раздела применяются обычные правила контроля доступа, и в большинстве случаев процессы, которые были ограничены, даже не знают об этом. Поскольку ограничение является необратимым, внутри раздела процессы могут выполняться с правами **root**, не опасаясь, что они могут поставить под угрозу другие части системы.

Этот умный трюк является одним из оснований для программной контейнеризации и его наиболее известной реализации Docker. Полная система намного сложнее и вклю-

чает в себя расширения, такие как доступ к файловой системе для копирования на запись. Нам еще немного нужно сказать о контейнерах в главе 25.

В качестве формы контроля доступа пространство имен является относительно грубым подходом. Конструкция правильно настроенных гнезд для процессов, в которых можно жить, также несколько сложна.

В настоящее время эта технология применяется в первую очередь к дополнительным службам, а не к внутренним компонентам операционной системы.

3.4. Современный контроль доступа

Учитывая широкий диапазон вычислительных сред и смешанный успех усилий по продвижению стандартной модели, разработчики ядра неохотно выступали в качестве посредников в более широких дискуссиях по контролю доступа. В мире Linux ситуация активизировалась в 2001 г., когда Агентство национальной безопасности США предложило интегрировать свою систему Enhanced Linux (SELinux) в ядро в качестве стандартного средства.

По нескольким причинам сторонники ядра сопротивлялись этому слиянию. Вместо того чтобы использовать SELinux или другую альтернативную систему, они разработали интерфейс уровня ядра API Linux Security Modules, позволяющий системам управления доступом интегрироваться в качестве загружаемых модулей ядра.

Системы на основе LSM не действуют, если пользователи не загрузили и не включили их. Этот факт снижает барьеры для включения в стандартное ядро, и система Linux теперь поставляется с SELinux и четырьмя другими системами (AppArmor, Smack, TOMOYO и Yama), готовыми к работе.

Разработки на стороне BSD примерно совпадают с разработками Linux, в основном благодаря работе Роберта Уотсона (Robert Watson) над TrustedBSD. Этот код был включен в систему FreeBSD, начиная с версии 5. Он также предоставляет технологию песочницы приложений, используемую в системах MacOS и iOS от Apple.

Если одновременно задействованы несколько модулей управления доступом, для их разрешения должна быть утверждена отдельная операция. К сожалению, система LSM требует явного сотрудничества между активными модулями, и ни один из существующих модулей не включает эту функцию. На данный момент системы Linux фактически ограничены выбором одного дополнительного модуля LSM.

Отдельные экосистемы

Контроль доступа по своей сути относится к уровню ядра. За исключением списков управления доступом к файловой системе (см. раздел 5.6), по существу, среди систем в отношении альтернативных механизмов контроля доступа нет стандартизации. В результате каждое ядро имеет свой собственный набор доступных реализаций, и ни одна из них не является кросс-платформенной.



Поскольку дистрибутивы Linux имеют общую линию ядра, все они теоретически совместимы со всеми различными предложениями обеспечения безопасности Linux. Однако на практике это не так: эти системы нуждаются в поддержке на уровне пользователя в виде дополнительных команд, модификаций компонентов уровня пользователя и профилей защиты для демонов и служб. Следовательно, у каждого дистрибутива есть только один или два механизма контроля доступа, которые он активно поддерживает.

Обязательный контроль доступа

Стандартная модель UNIX рассматривается как форма "избирательного контроля доступа", поскольку позволяет владельцам контролируемых доступом объектов устанавливать права на них. Например, вы можете разрешить другим пользователям просматривать содержимое вашего домашнего каталога или написать программу с флагом `setuid`, которая позволяет другим людям отправлять сигналы вашим процессам.

Избирательное управление доступом не обеспечивает особой гарантии безопасности данных на пользовательском уровне. Поскольку пользователи имеют возможность устанавливать разрешения самостоятельно; никто не знает, что они могут делать со своими собственными файлами. Даже самые благонамеренные и обученные пользователи могут ошибаться.

Системы мандатного управления доступом (mandatory access control — MAC) позволяют администраторам писать политики контроля доступа, которые переопределяют или дополняют избирательные разрешения традиционной модели. Например, вы можете установить правило, что домашние каталоги пользователей доступны только их владельцам. Тогда не важно, что пользователь делает приватную копию конфиденциального документа и небрежно относится к разрешениям документа; в любом случае никто ничего не может увидеть в домашнем каталоге этого пользователя.

Мандаты MAC — это эффективная технология для реализации моделей безопасности, таких как система многоуровневой безопасности Министерства обороны. В этой модели политики безопасности контролируют доступ в соответствии с воспринимаемой секретностью контролируемых ресурсов. Пользователям назначается уровень безопасности из структурированной иерархии. Они могут читать и писать документы на том же уровне секретности или ниже, но не могут получить доступ к документам с более высоким уровнем секретности. Например, пользователь с уровнем "секретно" может читать и писать секретные документы, но не может читать документы, которые классифицируются как совершенно секретные.

Если вы не обрабатываете секретные данные для государственного органа, маловероятно, что вы когда-нибудь столкнетесь или захотите развернуть такие всеобъемлющие модели безопасности. Чаще всего система MAC используется для защиты отдельных служб, в противном случае они остаются вне доступа пользователей.

Хорошо реализованная политика MAC основывается на принципе наименьших привилегий (разрешая доступ только тогда, когда это необходимо), поскольку правильно спроектированный брандмауэр позволяет проходить только определенным признанным службам и клиентам. Система MAC может помешать компрометации системы с уязвимым программным обеспечением (например, на основе переполнения буфера), ограничив объем нарушения до нескольких конкретных ресурсов, требуемых этому программному обеспечению.

К сожалению, система MAC стала чем-то вроде модного слова, синонимом "расширенного контроля доступа". Даже общий API-интерфейс безопасности FreeBSD называется интерфейсом MAC, несмотря на то, что некоторые дополнения не предлагают никаких реальных функций MAC.

Доступные системы MAC варьируются от глобальной замены стандартной модели до облегченных расширений, которые предназначены для конкретных предметных областей и сценариев использования. Общим для реализаций MAC является то, что они обычно добавляют в систему контроля доступа централизованные, написанные администратором (или поставляемые поставщиком) политики, а также обычное сочетание разрешений файлов, списков управления доступом и атрибутов процесса.

Независимо от области действия, системы MAC представляют собой потенциально значительное отклонение от стандартных систем, которое может оказаться неожиданным для программ, основанных на стандартной модели безопасности UNIX. Прежде чем перейти к полномасштабному развертыванию MAC, убедитесь, что вы понимаете правила ведения журнала модуля и знаете, как идентифицировать и устранять проблемы, связанные с MAC.

Контроль доступа на основе ролей

Еще одна функция, которая обычно проверяется системами контроля доступа, — это контроль доступа на основе ролей (также известный как RBAC), теоретическая модель, формализованная в 1992 г. Дэвидом Феррариоло (David Ferraiolo) и Риком Куном (Rick Kuhn). Основная идея состоит в том, чтобы добавить слой косвенности в управление доступом к вычислениям.

Разрешения назначаются не пользователям непосредственно, а промежуточным конструкциям, называемым *ролями*, а роли, в свою очередь, назначаются пользователям. Чтобы принять решение, связанное с управлением доступом, система перечисляет роли текущего пользователя и проверяет, имеют ли какие-либо из этих ролей соответствующие разрешения.

Роли похожи на разные группы UNIX, но они носят более общий характер, поскольку могут использоваться вне контекста файловой системы. Роли также могут образовывать иерархии, что значительно упрощает администрирование. Например, вы можете определить роль "старшего администратора", которая имеет все разрешения "администратора" плюс дополнительные разрешения X, Y и Z.

Многие версии UNIX, включая Solaris, HP-UX и AIX, включают в себя некоторую форму встроенной системы RBAC. Системы Linux и FreeBSD не имеют четкого, собственного средства RBAC. Однако она встроена в несколько более полных вариантов системы MAC.

SELinux: улучшенная безопасность Linux

SELinux является одной из старейших реализаций системы Linux MAC и является продуктом Агентства национальной безопасности США. В зависимости от точки зрения, этот факт может быть источником уверенности или подозрительности.[8]

Система SELinux использует максималистский подход и реализует почти все возможности MAC и RBAC, которые можно себе представить. Несмотря на то что она внедрена в нескольких дистрибутивах, ею, как правило, сложно управлять и устранять неполадки. Приведем анонимную цитату из старой версии страницы SELinux в системе Wikipedia, которая передает разочарование, испытываемое многими системными администраторами.

> *Как ни странно, хотя заявленное предназначение SELinux заключается в том, чтобы облегчить создание индивидуализированных политик управления доступом, специально предназначенных для практики и правил хранения данных в организациях, ее программные инструменты настолько скудны и неудобны, что поставщики выживают главным образом благодаря консалтингу, который обычно принимает форму дополнительных изменений в политике безопасности шаблонов.*

[8]Если вы склонны к подозрениям, то стоит отметить, что в качестве части дистрибутива ядра Linux кодовая база SELinux открыта для проверки.

Несмотря на сложность управления, внедрение системы SELinux медленно расширяется, особенно в таких средах, как правительственные органы, финансы и здравоохранение, которые имеют высокие и конкретные требования к безопасности. Она также является стандартной частью платформы Android.

Наше общее мнение относительно системы SELinux заключается в том, что она способна принести больше вреда, чем пользы. К сожалению, этот вред может проявляться не только как потерянное время и испорченное настроение системных администраторов, но и, как это ни парадоксально, в качестве недостатков в области безопасности. Сложные модели трудно обсуждать, а SELinux — это не ровное игровое поле; хакеры, которые специализируются на этой системе, понимают ее гораздо лучше, чем средний системный администратор.

В частности, разработка политики SELinux является сложной задачей. Например, для защиты нового демона политика должна тщательно перечислить все файлы, каталоги и другие объекты, к которым процесс должен получить доступ. Для сложного программного обеспечения, такого как **sendmail** или **httpd**, эта задача может быть довольно сложной. Например, одна компания предлагает трехдневный курс по разработке политики.

К счастью, многие общие политики доступны в режиме онлайн, и большинство дистрибутивов SELinux имеют разумные значения по умолчанию. Их можно легко установить и настроить для конкретной среды. Полномасштабный редактор политики, целью которого является облегчение применения политики, можно найти на сайте `seedit.sourceforge.net`.

RHEL  Система SELinux хорошо поддерживается системами Red Hat (и, следовательно, CentOS) и Fedora. Система Red Hat устанавливает ее по умолчанию.

 Системы Debian и SUSE Linux также имеют некоторую доступную поддержку SELinux, но вы должны установить дополнительные пакеты, а конфигурация по умолчанию является более слабой.

 Система Ubuntu наследует некоторую поддержку SELinux от Debian, но в последних версиях Ubuntu используется система AppArmor (см. далее). Некоторые рудиментарные пакеты, относящиеся к SELinux, по-прежнему доступны, но они, как правило, не обновляются.

Файл **/etc/selinux/config** — это элемент управления верхнего уровня для систем SELinux. В нем есть интересные строки:

```
SELINUX=enforcing
SELINUXTYPE=targeted
```

Первая строка имеет три возможных значения: `enforcing`, `permissive` или `disabled`. Параметр `enforcing` гарантирует, что загруженная политика применяется и запрещает нарушения. Параметр `permissive` допускает нарушения, но регистрирует их в журнале `syslog`, что полезно для отладки и разработки политики. Параметр `disabled` полностью отключает систему SELinux.

Имя `SELINUXTYPE` представляет собой имя используемой базы данных политик. Это, по сути, имя подкаталога внутри каталога **/etc/selinux**. В один и тот же момент времени применяться может только одна политика, а доступные наборы политик зависят от системы.

RHEL  По умолчанию стратегия системы Red Hat задается параметром `targeted` и направлена на обеспечение дополнительной безопасности нескольких демонов, которые она явно защищает, предоставляя остальную часть системы самой

себе. Раньше существовала отдельная политика, называемая `strict`, которая применяла стратегию MAC ко всей системе, но эта политика была объединена с политикой `targeted`. Чтобы получить полную систему MAC, удалите модули `unconfinedus` и `unconfineduser` с помощью команды **`semodule -d`**.

Система Red Hat также определяет политику **`mls`**, которая реализует многоуровневую защиту DoD. Она устанавливается отдельно с помощью команды **`yum install selinux-policy-mls`**.

Если вы заинтересованы в разработке собственных политик SELinux, обратите внимание на утилиту **`audit2allow`**. Она создает определения политики из журналов нарушений. Идея заключается в том, чтобы разрешить защиту подсистемы, регистрируя, но не запрещая ее нарушения. Затем можно подключить подсистему и создать политику, которая, по существу, позволяет этой подсистеме все. К сожалению, трудно гарантировать полное покрытие всех путей кода с помощью такого подхода, поэтому автоматически созданные профили вряд ли будут идеальными.

AppArmor



Система AppArmor является продуктом компании Canonical, Ltd., выпускающей дистрибутив Ubuntu. Она поддерживается системами Debian и Ubuntu, но также была принята в качестве стандарта дистрибутивами SUSE. Системы Ubuntu и SUSE позволяют устанавливать ее по умолчанию, хотя набор защищенных служб не является обширным.

Система AppArmor реализует стратегию MAC и предназначена в качестве дополнения к традиционной системе управления доступом UNIX. Несмотря на то что возможна любая конфигурация, AppArmor не предназначена для системы, ориентированной на пользователя. Ее основная цель — обеспечение безопасности, т.е. ограничение ущерба, который могут нанести отдельные программы, если они будут скомпрометированы или запущены.

Защищенные программы по-прежнему подвергаются всем ограничениям, налагаемым стандартной моделью, но, кроме того, ядро фильтрует их действия через назначенный и специфичный для приложения профиль AppArmor. По умолчанию система AppArmor отклоняет все запросы, поэтому профиль должен явно указывать все, что разрешено для процесса.

Программы без профилей, таких как пользовательские оболочки, не имеют особых ограничений и работают так, как если бы система AppArmor не была установлена.

Эта роль обеспечения безопасности службы, по сути, является той же конфигурацией, которая реализована системой SELinux в целевой среде Red Hat. Тем не менее система AppArmor разработана специально для обеспечения безопасности, поэтому она скрывает некоторые из самых загадочных нюансов SELinux.

Профили AppArmor хранятся в файле **`/etc/apparmor.d`**, и относительно понятны даже без подробного знания системы. Например, вот профиль для демона **`cups-browsed`**, входящего в систему печати в операционной системе Ubuntu:

```
#include <tunables/global>

/usr/sbin/cups-browsed {

    #include <abstractions/base>
    #include <abstractions/nameservice>
```

```
    #include <abstractions/cups-client>
    #include <abstractions/dbus>
    #include <abstractions/p11-kit>

    /etc/cups/cups-browsed.conf r,
    /etc/cups/lpoptions r,
    /{var/,}run/cups/certs/* r,
    /var/cache/cups/* rw,
    /tmp/** rw,

     # Site-specific additions and overrides. See local/README.
     #include <local/usr.sbin.cups-browsed>
}
```

Большая часть этого кода является модульным шаблоном. Например, этот демон должен выполнять поиск имен хостов, поэтому профиль интерполирует путь **abstractions/nameservice** и предоставляет доступ к библиотекам разрешений имен, файлам **/etc/nsswitch.conf** и **/etc/hosts**, сетевым портам, используемым протоколом LDAP и т.д.

Информация профилирования, специфичная для этого демона, состоит (в данном случае) из списка файлов, к которым может обращаться демон, а также разрешений для каждого файла. Синтаксис сопоставления шаблонов немного отличается: символы ** могут соответствовать нескольким компонентам пути, а `{var/,}` соответствует строке `var/` в соответствующем месте пути.

Даже этот простой профиль устроен довольно сложно. Если раскрыть все инструкции **#include**, то профиль имеет длину около 750 строк. (А ведь мы выбрали этот пример ради его краткости. Увы!)

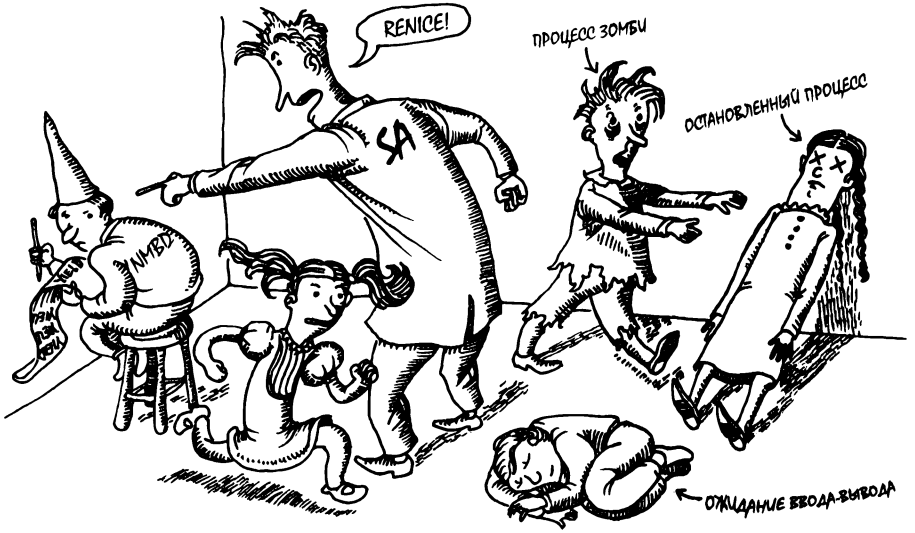
Система AppArmor ссылается на файлы и программы по имени пути, что делает профили удобочитаемыми и независимыми от какой-либо конкретной реализации файловой системы. Однако этот подход является компромиссным. Например, система AppArmor не распознает жесткие ссылки, указывающие на один и тот же объект.

3.5. Литература

- Ferraiolo, David F., D. Richard Kuhn, and Ramaswamy Chandramouli. *Role-Based Access Control (2nd Edition)*. Boston, MA: Artech House, 2007.
- Haines, Richard. *The SELinux Notebook (4th Edition)*. 2014. Сборник информации, относящейся к системе SELinux, наиболее близкий к официальной документации. Доступен для загрузки на сайте `freecomputerbooks.com`.
- Vermeulen, Sven. *SELinux Cookbook*. Birmingham, UK: Packt Publishing, 2014. Книга, содержащая самые разнообразные советы по работе с системой SELinux. В ней описаны модели обеспечения безопасности как служб, так и пользователей.

Глава 4

Управление процессами



Процесс представляет собой выполняемую программу. Это абстракция, с помощью которой можно управлять памятью, временем работы процессора и ресурсами ввода-вывода.

Это аксиома философии UNIX, позволяющая как можно больше работать в контексте процессов, а не ядра. Системные и пользовательские процессы соответствуют одним и тем же правилам, поэтому вы можете использовать один набор инструментов для управления ими.

4.1. Компоненты процесса

Процесс состоит из адресного пространства и набора структур данных внутри ядра. Адресное пространство представляет собой набор страниц памяти, которые ядро выделило для использования процессу.[1] Эти страницы содержат код и библиотеки, которые выполняются процессом, переменные процесса, его стеки и различную дополнительную информацию, необходимую ядру во время выполнения. Виртуальное адресное пространство процесса выделяется в физической памяти случайным образом и отслеживается таблицами страниц ядра.

В структурах данных ядра хранится всевозможная информация о каждом процессе. К наиболее важным относятся следующие сведения:

- таблица распределения памяти, выделенной процессу;
- текущее состояние процесса (неактивен, приостановлен, выполняется и т.п.);
- приоритет процесса;

[1] Страницы — это базовые блоки памяти, размер которых составляет от 1 до 8 Кбайт.

- информация о ресурсах, используемых процессом (центральный процессор, память и т.д.);
- информация о файлах и сетевых портах, открытых процессом;
- маска сигналов (запись о том, какие сигналы блокируются);
- имя владельца процесса.

Поток — это контекст выполнения процесса. Каждый процесс имеет как минимум один поток, но некоторые процессы могут иметь несколько потоков. Каждый поток, действуя в адресном пространстве внешнего процесса, имеет свой собственный стек и регистры центрального процессора

В современных компьютерных системах используются несколько центральных процессоров и несколько ядер внутри каждого центрального процессора. Такие многопоточные приложения, как BIND и Apache, извлекают максимальную пользу из мультиядерных систем благодаря тому, что эти приложения могут отрабатывать несколько запросов одновременно.

Многие характеристики процесса непосредственно влияют на его выполнение. В частности, имеет значение, сколько времени выделяется ему центральным процессором, к каким файлам он имеет доступ и т.д. В следующих подразделах рассмотрим наиболее интересные с точки зрения системного администратора характеристики процессов. Они поддерживаются во всех версиях систем UNIX и Linux.

Идентификатор процесса PID

Ядро назначает каждому процессу уникальный идентификатор. Большинство команд и системных вызовов, работающих с процессами, требует указания конкретного идентификатора, чтобы был ясен контекст операции. Идентификаторы PID присваиваются по порядку по мере создания процессов.



В настоящее время система Linux использует концепцию пространства имен процесса, которая еще больше ограничивает способность процессов видеть друг друга и влиять друг на друга. Контейнерные реализации используют эту функцию для разделения процессов. Один из побочных эффектов заключается в том, что процесс может иметь разные PID в зависимости от пространства имен наблюдателя. Это похоже на эйнштейновскую относительность для идентификаторов процессов. Для получения дополнительной информации см. главу 25.

Идентификатор родительского процесса PPID

Ни в UNIX, ни в Linux нет системного вызова, который бы инициировал новый процесс для выполнения конкретной программы. Для того чтобы породить новый процесс, существующий процесс должен клонировать сам себя. Клон может заменить выполняемую программу другой.

В операции клонирования исходный процесс называют *родительским*, а его клон — *дочерним*. Помимо собственного идентификатора, каждый дочерний процесс имеет атрибут PPID (Parent Process ID), который совпадает с идентификатором породившего его родительского процесса[2].

[2]По крайней мере первоначально. Если родительский процесс по какой-то причине завершается раньше потомка, демон `init` или `sytemd` (процесс с номером 1) подставляет себя на место предка (подробности описаны в разделе 4.2).

Идентификатор родительского процесса — весьма полезная информация, если приходится иметь дело с неизвестным (и, возможно, нестандартно ведущим себя) процессом. Отслеживание истоков процесса может облегчить понимание его назначения и значимости.

🕮 Дополнительную информацию об идентификаторах UID см. в разделе 8.2.

Идентификатор пользователя UID и текущий идентификатор пользователя EUID

UID (User ID) — это идентификатор пользователя, создавшего данный процесс, точнее, копия значения UID родительского процесса. Менять атрибуты процесса могут только его создатель (владелец) и суперпользователь.

🕮 Дополнительную информацию об установке флага `setuid` см в разделе 3.1.

EUID (Effective User ID) — это текущий пользовательский идентификатор процесса, предназначенный для того, чтобы определить, к каким ресурсам и файлам у процесса есть право доступа в данный момент. У большинства процессов значения UID и EUID одинаковы. Исключение составляют программы, у которых установлен бит смены идентификатора пользователя (`setuid`).

Зачем нужны идентификаторы UID и EUID? Просто чтобы разграничить понятия идентификации и прав доступа. К тому же программы с установленным битом `setuid` не всегда выполняются с расширенными привилегиями. В большинстве систем значение EUID можно устанавливать, чтобы предоставлять процессу дополнительные полномочия, и сбрасывать, чтобы отбирать их.

В большинстве систем хранится начальный идентификатор, т.е. копия значения EUID, которое имел процесс в начальной точке. Если процесс не удалит сохраненный идентификатор, его можно будет использовать в качестве реального или текущего идентификатора. Благодаря этому "консервативно" написанная программа с установленным битом `setuid` может большую часть времени работать с обычными привилегиями, переходя в режим расширенных привилегий лишь в определенные моменты.



В системе Linux есть еще и нестандартный параметр FSUID, определяющий возможности работы с файловой системой, но вне ядра он используется редко и не переносится на другие системы UNIX.

Идентификатор группы (GID) и текущий идентификатор группы (EGID)

🕮 Дополнительную информацию о группах см. в разделе 8.2.

GID (Group ID) — это идентификатор группы, к которой принадлежит владелец процесса. Идентификатор EGID связан с атрибутом GID так же, как с атрибутом UID, т.е. они будут отличаться, если программа запускается с установленным битом `setgid`. В ядре назначение сохраненного GID для каждого процесса аналогично назначению сохраненного атрибута UID.

В значительной степени атрибут GID процесса является рудиментарным. С точки зрения определения прав доступа процесс одновременно может относиться к нескольким группам. Список групп хранится отдельно от значений GID и EGID. При анализе прав доступа проверяется текущий идентификатор и дополнительный список групп, но не значение GID.

Единственная ситуация, в которой атрибут GID имеет реальное значение, — создание процессом новых файлов. В зависимости от установленных прав доступа в данной файловой системе новые файлы могут принимать атрибут GID создающего их процесса. Подробнее эти вопросы рассмотрены в разделе 5.5.

Фактор уступчивости

Приоритет процесса определяет, какую долю времени центрального процессора получает программа. Ядро применяет динамический алгоритм вычисления приоритетов, учитывающий, сколько времени центрального процессора уже использовал процесс и сколько времени он ожидает своей очереди. Кроме того, учитывается заданный администратором так называемый *фактор уступчивости*, который определяет, в какой степени программа может "делиться" процессором с другими пользователями. Подробнее этот механизм рассматривается в разделе 4.4.

Управляющий терминал

📖 Дополнительную информацию о стандартных каналах связи см в разделе 7.2.

С большинством процессов, не являющихся демонами, связан управляющий терминал, который определяет базовую конфигурацию стандартных каналов ввода, вывода и ошибок. От управляющего терминала зависит также распределение сигналов в ответ на события клавиатуры, например нажатие клавиш <Ctrl+C>, о чем пойдет речь в разделе 4.2.

4.2. Жизненный цикл процесса

Для создания нового процесса существующий процесс, как правило, клонирует сам себя с помощью системного вызова **fork**.[3] В результате формируется копия исходного процесса, имеющая лишь некоторые отличия. В частности, новому процессу присваивается собственный идентификатор PID, и учет ресурсов ведется для него независимо от предка.

Системный вызов **fork** имеет уникальное свойство: он возвращает два разных значения. В дочернем процессе это число 0, а в родительском — идентификатор PID процесса-потомка. Поскольку в остальном процессы идентичны, они должны проверять результат вызова, чтобы определить, какую роль следует играть в дальнейшем.

После завершения системного вызова **fork** дочерний процесс обычно запускает новую программу с помощью одного из системных вызовов семейства **exec**. Все вызовы этого семейства заменяют программу, выполняемую процессом, и устанавливают сегменты памяти в исходное состояние. Формы вызовов **exec** различаются только способами указания аргументов командной строки и переменных среды, передаваемых новой программе.

При загрузке системы ядро самостоятельно запускает несколько процессов. Наиболее важный из них — демон **init** или **systemd** с номером процесса, всегда равным 1. Эти процессы отвечают за выполнение сценариев запуска системы, хотя харак-

[3] Технически в системах Linux используется системный вызов **clone**, расширение системного вызова **fork**, которое обрабатывает потоки и включает дополнительные функции. Системный вызов **fork** остается в ядре для обеспечения обратной совместимости, но на самом деле он выполняет системный вызов **clone**.

тер их действий в UNIX и Linux различается. Все процессы, кроме тех, что создаются ядром, являются потомками этих процессов. Дополнительная информация о загрузке и демоне **init** содержится в главе 2.

Демон **init** (или **systemd**) играет и другую важную роль в управлении процессами. Завершающийся процесс вызывает функцию **_exit**, чтобы уведомить ядро о своей готовности прекратить работу. В качестве параметра функции **_exit** передается код завершения — целое число, обозначающее причину прекращения процесса. По общепринятому соглашению, нуль свидетельствует об успешном завершении процесса.

Ядро системы требует, чтобы, прежде чем процесс окончательно исчезнет, его удаление было подтверждено родительским процессом с помощью системного вызова **wait**. Этот вызов возвращает код завершения потомка (или сообщает о причине его уничтожения, если завершение не было естественным) и в случае необходимости статистику использования ресурсов.

Описанный механизм работает нормально, если родительский процесс завершается позже порожденных им процессов и выполняет системные вызовы **wait** для их уничтожения. Если же родительский процесс завершается раньше срока, то ядро распознает, что вызова **wait** не последует, и переназначает все "осиротевшие" процессы демону **init** (или **systemd**). Он берет их под свой контроль, осуществляя для каждого из них вызов **wait**.

Сигналы

Сигналы — это запросы на прерывание, реализуемые на уровне процессов. Существуют свыше тридцати различных сигналов, и они находят самое разное применение.

- Сигналы могут посылаться между процессами как средство коммуникации.
- Сигналы могут посылаться драйвером терминала для уничтожения или приостановки процессов, когда пользователь нажимает специальные комбинации клавиш, такие как <Ctrl+C> или <Ctrl+Z>[4].
- Сигналы могут посылаться в самых разных целях пользователем или администратором с помощью команды **kill**.
- Сигналы могут посылаться ядром, когда процесс выполняет недопустимую инструкцию, например деление на нуль.
- Сигналы могут посылаться ядром для уведомления процесса о "представляющем интерес" событии, таком как прекращение дочернего процесса или доступность данных в канале ввода-вывода.

🕮 Дамп памяти — это файл, содержащий образ памяти процесса. Его можно использовать для отладки.

Когда поступает сигнал, возможен один из двух вариантов развития событий. Если процесс назначил сигналу подпрограмму обработки, то после вызова ей предоставляется информация о контексте, в котором был сгенерирован сигнал. В противном случае ядро выполняет от имени процесса действия, заданные по умолчанию. Эти действия зависят от сигнала. Многие сигналы приводят к завершению процесса, а в некоторых случаях при этом еще и создаются дампы памяти.

[4]Функции, связанные с комбинациями клавиш <Ctrl+Z> или <Ctrl+C>, могут назначаться другим клавишам с помощью команды **stty**, но на практике такое встречается очень редко. В этой главе мы подразумеваем, что с данными клавишами связаны их стандартные функции.

Процедура вызова обработчика называется *перехватом сигнала*. Когда выполнение обработчика завершается, процесс возобновляется с той точки, где был получен сигнал.

Для того чтобы определенные сигналы не поступали в программу, нужно задать их игнорирование или блокирование. Игнорируемый сигнал просто пропускается и не влияет на работу процесса. Блокируемый сигнал ставится в очередь на обработку, но ядро не требует от процесса никаких действий до явного разблокирования сигнала. Обработчик вызывается для разблокированного сигнала только один раз, даже если в течение периода блокировки поступило несколько аналогичных сигналов.

В табл. 4.1 перечислены сигналы, которые должны быть известны любому системному администратору. Традиционно имена сигналов записываются прописными буквами. Иногда к именам добавляется префикс `SIG` (например, `SIGHUP`).

Таблица 4.1. Сигналы, которые должен знать каждый администратор[a]

№[б]	Имя	Описание	Реакция по умолчанию	Перехватывается?	Блокируется?	Дамп памяти?
1	HUP	Отбой	Завершение	Да	Да	Нет
2	INT	Прерывание	Завершение	Да	Да	Нет
3	QUIT	Выход	Завершение	Да	Да	Да
9	KILL	Уничтожение	Завершение	Нет	Нет	Нет
10	BUS	Ошибка на шине	Завершение	Да	Да	Да
11	SEGV	Ошибка сегментации	Завершение	Да	Да	Да
15	TERM	Запрос на завершение	Завершение	Да	Да	Нет
17	STOP	Остановка	Остановка	Нет	Нет	Нет
18	TSTP	Сигнал остановки, посылаемый с клавиатуры	Остановка	Да	Да	Нет
19	CONT	Продолжение после остановки	Игнорируется	Да	Нет	Нет
28	WINCH	Изменение окна	Игнорируется	Да	Да	Нет
30	USR1	Определяется пользователем	Завершение	Да	Да	Нет
31	USR2	Определяется пользователем	Завершение	Да	Да	Нет

[a] Список названий сигналов и номеров также можно получить с помощью встроенной в оболочку **bash** команды **kill -l**.

[б] Зависит от используемой системы. Подробнее см. файл **/usr/include/signal.h** или man-страницы интерактивного руководства для системного вызова `signal()`.

Существуют и другие сигналы, не показанные в табл. 4.1; большинство из них сообщает о загадочных ошибках, например "неверная инструкция". По умолчанию такие сигналы, как правило, приводят к завершению программы и созданию дампа ядра. Перехват и блокирование сигналов обычно разрешены, так как есть достаточно "умные" программы, устраняющие последствия ошибок.

Сигналы `BUS` и `SEGV` также посылаются при возникновении ошибок. Мы включили их в таблицу, поскольку они чрезвычайно распространены: обычно программа аварийно завершается именно из-за них. Сами по себе эти сигналы не имеют диагностической ценности. Они лишь указывают на факт неправильного обращения к памяти.

Сигналы `KILL` и `STOP` нельзя ни перехватить, ни заблокировать, ни проигнорировать. Сигнал `KILL` приводит к уничтожению процесса, которому он посылается, а сиг-

нал STOP приостанавливает выполнение процесса до получения сигнала CONT. Сигнал CONT можно перехватить и проигнорировать, но нельзя заблокировать.

Сигнал TSTP представляет собой более "гибкую" версию сигнала STOP. Проще всего описать его как запрос на остановку. Он генерируется драйвером терминала при нажатии пользователем комбинации клавиш <Ctrl+Z>. Программы, перехватывающие этот сигнал, обычно выполняют операции очистки, а затем посылают сами себе сигнал STOP. С другой стороны, программы могут игнорировать сигнал TSTP, чтобы их нельзя было остановить командой с клавиатуры.

Хотя назначение сигналов KILL, INT, TERM, HUP и QUIT может показаться одинаковым, в действительности они совершенно разные.

- Сигнал KILL не блокируется и приводит к безусловному завершению процесса на уровне ядра. По сути, процесс не успевает даже принять этот сигнал.
- Сигнал INT посылается драйвером терминала при нажатии пользователем комбинации клавиш <Ctrl+C> и служит запросом на завершение текущей операции. Перехватив этот сигнал, простые программы должны завершить работу или позволить уничтожить себя стандартному обработчику сигнала. Программы, в которых есть интерактивный режим командной строки, должны прекратить текущую операцию, выполнить очистку и снова перейти в режим ожидания.
- Сигнал TERM представляет собой запрос на завершение программы. Предполагается, что процесс, получивший этот сигнал, осуществляет очистку и завершается.
- У сигнала HUP есть две распространенные интерпретации. Во-первых, многие демоны воспринимают его как команду сброса. Если демон способен повторно прочесть свой конфигурационный файл и адаптироваться к изменениям без перезапуска, сигнал HUP позволяет менять его поведение.
- Во-вторых, этот сигнал иногда генерируется драйвером терминала при попытке уничтожить связанные с терминалом процессы. В основном это поведение сохранилось со времен использования проводных соединений терминалов и модемов. Отсюда и название "отбой".
- Оболочки семейства C (**tcsh** и другие) обычно делают фоновые процессы невосприимчивыми к сигналу HUP, чтобы они могли продолжать свою работу, даже когда пользователь выходит из системы. Пользователи оболочек семейства Bourne (**ksh**, **bash** и так далее) могут эмулировать такое поведение с помощью команды **nohup**.
- Сигнал QUIT напоминает сигнал TERM, за исключением того, что по умолчанию стандартный обработчик создает дамп памяти.

Сигналы USR1 и USR2 не имеют стандартного назначения. Ими можно пользоваться в различных целях. Например, веб-сервер Apache интерпретирует сигнал HUB как запрос на немедленный перезапуск. Сигнал USR1 инициирует более плавный переход, в ходе которого разрешается закончить сеансы связи существующего клиента.

Команда **kill**: отправка сигналов

Команду **kill** чаще всего используют для уничтожения процессов. Эта команда может послать процессу любой сигнал, но по умолчанию это сигнал TERM. Команду **kill** могут выполнять как рядовые пользователи (для своих собственных процессов), так и суперпользователь (для любого процесса). Она имеет следующий синтаксис:

kill [*-сигнал*] *pid*

где *сигнал* — это номер или символическое имя посылаемого сигнала (см. табл. 4.1), а *pid* — идентификационный номер целевого процесса.

Команда **kill** без номера сигнала не гарантирует, что процесс будет уничтожен, поскольку сигнал TERM можно перехватывать, блокировать и игнорировать. Команда

```
$ kill -9 pid
```

"гарантированно" уничтожает процесс, так как сигнал с номером 9 (KILL) не перехватывается. Используйте команду **kill -9** только в случае, если "вежливый" запрос на завершение программы не был выполнен. Мы написали слово "гарантированно" в кавычках, так как иногда процессы переходят в такое состояние, в котором их нельзя завершить даже таким способом (обычно это связано с блокировкой ввода-вывода, например при остановке жесткого диска). Единственный выход в такой ситуации — перезагрузка.

Команда **killall** уничтожает процессы, заданные именем. Например, следующая команда уничтожает все процессы веб-сервера Apache.

```
$ sudo killall httpd
```

Команда **pkill** осуществляет поиск процессов, заданных именами (или другими атрибутами, например EUID), и посылает найденным процессам сигнал. Например, следующая команда посылает сигнал TERM всем процессам, выполняемым от имени пользователя ben.

```
$ sudo pkill -u ben
```

Состояния процессов и потоков

Как показано в предыдущем разделе, процесс может быть приостановлен сигналом STOP и возвращен в активную нагрузку с сигналом CONT. Состояние приостановления или выполнения применяется к процессу в целом и наследуется всеми потоками процесса.[5]

Даже при номинальном запуске потоки часто должны ждать, пока ядро завершит какую-то фоновую работу для них, прежде чем они смогут продолжить выполнение. Например, когда поток считывает данные из файла, ядро должно запрашивать соответствующие блоки диска, а затем упорядочивать их содержимое для доставки в адресное пространство процесса запроса. В течение этого времени запрашивающий поток входит в состояние краткосрочного спящего режима, в котором он не может быть выполнен. Однако другие потоки в одном процессе могут продолжать работать.

Существуют процессы, которые называют "спящими" (например, в выводе результатов работы команды **ps** — см. следующий раздел). Поскольку атрибут сна относится к потоку, этот термин немного обманчив. Обычно процесс считается спящим, когда все его потоки являются спящими. Разумеется, различие остается спорным в случае однопоточных процессов, которые представляют собой наиболее распространенный случай.

Интерактивные оболочки и системные демоны проводят бо́льшую часть времени, ожидая ввода данных с терминала или сетевых подключений. Поскольку спящий поток эффективно блокируется до тех пор, пока его запрос не будет удовлетворен, его процесс обычно не получает никакого процессорного времени, если он не получает сигнал или ответ на один из своих запросов ввода-вывода.

🕮 Дополнительную информацию об инсталляции файловой системы NFS с параметром hard см. в разделе 21.4.

[5]Аналогичным образом можно управлять отдельными потоками. Однако эти объекты в первую очередь представляют интерес для разработчиков; системные администраторы не должны беспокоиться по этому поводу.

Некоторые операции могут привести к тому, что процессы или потоки войдут в состояние беспробудного сна. Это состояние обычно является переходным и не наблюдается в результате работы команды **ps** (оно обозначается буквой `D` в столбце `STAT`, см. табл. 4.2). Однако несколько специфических ситуаций могут привести к тому, что это состояние станет постоянным. Наиболее распространенная причина связана с проблемами сервера в файловой системе NFS, инсталлированной с параметром `hard`. Поскольку процессы в состоянии беспробудного сна не могут просыпаться даже для обслуживания сигнала, они не могут быть прекращены. Чтобы избавиться от них, вы должны устранить основную проблему или перезагрузить компьютер.

4.3. Команда ps: текущий контроль процессов

Команда **ps** — основной инструмент, которым системный администратор пользуется для текущего контроля процессов. Версии этой команды различаются аргументами и выходным форматом, но, по сути, выдают одну и ту же информацию. В основном, различие в версиях — это следствие разных путей развития систем UNIX. Кроме того, поставщики систем могут настраивать эту команду с учетом конфигурации системы, так как она тесно связана с особенностями обработки процессов в ядре и поэтому часто отражает изменения в ядре.

С помощью команды **ps** можно получить информацию об идентификаторах PID, UID, приоритете и управляющем терминале того или иного процесса. Она также позволяет выяснить, какой объем памяти использует процесс, сколько времени центрального процессора заняло его выполнение, каково его текущее состояние (выполняется, остановлен, простаивает и т.д.). Процессы-зомби в листинге команды обозначаются как `<exiting>` или `<defunct>`.

Команда **ps** безнадежно усложнилась за последние несколько лет. Некоторые поставщики оставили попытки стандартизировать выходной формат этой команды и сделали ее полностью конфигурируемой. Проведя небольшую настройку, можно получить практически любые требуемые результаты.



В системе Linux команда **ps** является настоящим хамелеоном и понимает наборы опций из ряда других систем. В отличие от остальных команд системы UNIX, команда **ps** в системе Linux воспринимает флаги командной строка как с дефисом, так и без дефиса, хотя их интерпретация может оказаться разной. Например, результат выполнения команды **ps -a** отличается от результата выполнения команды **ps a**.

Не пугайтесь всех этих сложностей: пусть они будут кошмаром разработчиков ядра, а не системных администраторов! Для повседневной работы достаточно знать лишь несколько наиболее важных опций команды **ps**.

Получить список всех процессов, выполняющихся в системах, можно с помощью команды **ps aux**. Ключ **a** означает, что мы хотим увидеть все процессы, ключ **x** — что мы хотим увидеть даже процессы, отсоединенные от управляющего терминала, а ключ **u** обеспечивает фильтрование по имени или идентификатору пользователя, который запустил программу. Ниже показаны результаты работы команды **ps aux** в системе Red Hat.

```
redhat$ ps aux
  USER  PID  %CPU %MEM    VSZ  RSS  TTY   STAT  TIME COMMAND
  root    1   0.1  0.2   3356  560    ?   S     0:00 init [5]
  root    2     0    0      0    0    ?   SN    0:00 [ksoftirqd/0]
  root    3     0    0      0    0    ?   S<    0:00 [events/0]
```

```
root     4     0    0     0    0    ?   S<    0:00 [khelper]
root     5     0    0     0    0    ?   S<    0:00 [kacpid]
root    18     0    0     0    0    ?   S<    0:00 [kblockd/0]
root    28     0    0     0    0    ?   S     0:00 [pdflush]
...
root   196     0    0     0    0    ?   S     0:00 [kjournald]
root  1050     0  0.1  2652  448    ?   S<s   0:00 udevd
root  1472     0  0.3  3048 1008    ?   S<s   0:00 /sbin/dhclient -1
root  1646     0  0.3  3012 1012    ?   S<s   0:00 /sbin/dhclient -1
root  1733     0    0     0    0    ?   S     0:00 [kjournald]
root  2124     0  0.3  3004 1008    ?   Ss    0:00 /sbin/dhclient -1
root  2182     0  0.2  2264  596    ?   Ss    0:00 syslogd -m 0
root  2186     0  0.1  2952  484    ?   Ss    0:00 klogd -x
root  2519   0.0  0.0 17036  380    ?   Ss    0:00 /usr/sbin/atd
root  2384     0  0.6  4080 1660    ?   Ss    0:00 /usr/sbin/sshd
root  2419     0  1.1  7776 3004    ?   Ss    0:00 sendmail: accept
...
```

Команды, имена которых заключены в квадратные скобки, в действительности являются не командами, а потоками ядра, запланированными в качестве процессов. Описание полей приведено в табл. 4.2.

Еще одна полезная аббревиатура, **lax**, предоставляет дополнительную техническую информацию. Ключи **a** и **x** описаны выше (отображают все процессы), а ключ **l** означает выбор "длинного" формата вывода данных. Команда **ps lax** выполняется быстрее, чем команда **ps aux**, так как не сопоставляет идентификаторы процессов с именами пользователей. Это может оказаться весьма важным фактором, если система уже перегружена каким-то процессом.

Таблица 4.2. Пояснения к результатам работы команды ps aux

Поле	Содержимое
USER	Имя владельца процесса
PID	Идентификатор процесса
%CPU	Доля времени центрального процессора (в процентах), выделенная процессу
%MEM	Часть реальной памяти (в процентах), используемая процессом
VSZ	Виртуальный размер процесса
RSS	Размер резидентного набора (количество страниц памяти)
TTY	Идентификатор управляющего терминала
STAT	Текущий статус процесса: R — выполняется, D — ожидает записи на диск, S — неактивен (< 20 с), T — приостановлен, Z — зомби. Дополнительные флаги: W — процесс выгружен на диск, < — процесс имеет повышенный приоритет, N — процесс имеет пониженный приоритет, L — некоторые страницы блокированы в ядре, s — процесс является лидером сеанса.
TIME	Количество времени центрального процессора, затраченное на выполнение процесса
COMMAND	Имя и аргументы команды[a]

[a] Программы могут модифицировать эту информацию, так что она не всегда точно представляет реальную командную строку.

Ниже приведены результаты работы аббревиатуры **ps lax**. Обратите внимание на дополнительные поля PPID (идентификатор родительского процесса), NI (фактор уступчивости) и WCHAN (ресурс, которого ожидает процесс).

```
redhat$ ps lax
   F UID   PID PPID PRI  NI  VSZ  RSS  WCHAN    STAT  TIME COMMAND
   4   0     1    0  16   0 3356  560  select   S     0:00 init [5]
   1   0     2    1  34  19    0    0  ksofti   SN    0:00 [ksoftirqd/0
   1   0     3    1   5 -10    0    0  worker   S<    0:00 [events/0]
   1   0     4    3   5 -10    0    0  worker   S<    0:00 [khelper]
   5   0  2186    1  16   0 2952  484  syslog   Ss    0:00 klogd -x
   5  32  2207    1  15   0 2824  580       -   Ss    0:00 portmap
   5  29  2227    1  18   0 2100  760  select   Ss    0:00 rpc.statd
   1   0  2260    1  16   0 5668 1084       -   Ss    0:00 rpc.idmapd
   1   0  2336    1  21   0 3268  556  select   Ss    0:00 acpid
   5   0  2384    1  17   0 4080 1660  select   Ss    0:00 sshd
   1   0  2399    1  15   0 2780  828  select   Ss    0:00 xinetd -sta
   5   0  2419    1  16   0 7776 3004  select   Ss    0:00 sendmail: a
   ...
```

В командах с длинными списками аргументов вывод командной строки может быть отключен. Чтобы отобразить на выходе больше столбцов, добавьте флаг **w**. Чтобы снять ограничения с ширины столбца, добавьте флаг **w** дважды; это удобно для тех процессов, которые имеют исключительно длинные аргументы командной строки, например некоторых Java-приложений.

Администраторам часто необходимо идентифицировать PID процесса. Вы можете найти PID, применив команду **grep** к результатам работы команды **ps**:

```
$ ps aux | grep sshd
root      6811 0.0 0.0  78056 1340  ?   Ss 16:04 0:00 /usr/sbin/sshd
bwhaley 13961 0.0 0.0 110408 868 pts/1 S+ 20:37 0:00 grep /usr/sbin/sshd
```

Обратите внимание на то, что результаты работы команды **ps** включают в себя команду **grep**, так как процесс **grep** был активен во время выполнения **ps**. Вы можете удалить эту строку из вывода с помощью команды **grep -v**:

```
$ ps aux | grep -v grep | grep sshd
root       6811 0.0 0.0   78056 1340?        Ss 16:04 0:00 /usr/sbin/sshd
```

Можно также определить PID процесса с помощью команды **pidof**:

```
$ pidof /usr/sbin/sshd
6811
```

Кроме того, можно применить утилиту **pgrep**:

```
$ pgrep sshd
6811
```

Команда **pidof** и утилита **pgrep** демонстрируют все процессы, соответствующие переданной строке. Часто обычная команда **grep** обеспечивает максимальную гибкость, хотя ее результаты могли бы быть чуть подробнее.

4.4. Интерактивный мониторинг процессов с помощью команды top

Команды, аналогичные **ps**, позволяют сделать моментальный снимок состояния системы, но получить полную картину всего происходящего в ней довольно сложно. Для

этого служит свободно распространяемая утилита **top**, которая работает во многих системах, регулярно обновляя сводку активных процессов и используемых ими ресурсов. Рассмотрим пример.

```
redhat$ top
top - 20:07:43 up 1:59, 3 users, load average: 0.45, 0.16, 0.09
Tasks: 251 total, 1 running, 250 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.7 us, 1.2 sy, 0.0 ni, 98.0 id, 0.0 wa, 0.0 hi, 0.2 si, 0.0 st
KiB Mem : 1013672 total,  128304 free, 547176 used, 338192 buff/cache
KiB Swap: 2097148 total, 2089188 free,   7960 used. 242556 avail Mem

PID    USER   PR  NI    VIRT    RES    SHR  S %CPU %MEM  TIME+   COMMAND
2731   root   20   0  193316  34848  15184  S  1.7  3.4  0:30.39  Xorg
25721  ulsah  20   0  619412  27216  17636  S  1.0  2.7  0:03.67  konsole
25296  ulsah  20   0  260724   6068   3268  S  0.7  0.6  0:17.78  prlcc
  747  root   20   0    4372    604    504  S  0.3  0.1  0:02.68  rngd
  846  root   20   0  141744    384    192  S  0.3  0.0  0:01.74  prltoolsd
 1647  root   20   0  177436   3656   2632  S  0.3  0.4  0:04.47  cupsd
10246  ulsah  20   0  130156   1936   1256  R  0.3  0.2  0:00.10  top
    1  root   20   0   59620   5472   3348  S  0.0  0.5  0:02.09  systemd
    2  root   20   0       0      0      0  S  0.0  0.0  0:00.02  kthreadd
    3  root   20   0       0      0      0  S  0.0  0.0  0:00.03  ksoftirqd/0
    5  root    0 -20       0      0      0  S  0.0  0.0  0:00.00  kworker/0:+
    7  root   rt   0       0      0      0  S  0.0  0.0  0:00.20  migration/0
    8  root   20   0       0      0      0  S  0.0  0.0  0:00.00  rcu_bh
    9  root   20   0       0      0      0  S  0.0  0.0  0:00.00  rcuob/0
...
```

По умолчанию эта информация обновляется каждые десять секунд. Наиболее активные процессы отображаются первыми. Команда **top** позволяет также посылать процессам сигналы и использовать команду **renice**, чтобы пользователь мог наблюдать за тем, как его действия влияют на общее состояние системы.

Сводная информация в первых нескольких строках верхнего выхода — одно из первых мест для анализа состояния системы. Оно в сжатом виде отражает загрузку системы, использование памяти, количество процессов и использование процессора.

В многоядерных системах использование центрального процессора усредняется по всем ядрам. В системе Linux нажмите клавишу 1 (номер один), пока команда **top** предлагает переключиться между ядрами. В системе FreeBSD для достижения того же эффекта выполните команду **top -P**.[6]

Суперпользователь может выполнять команду **top** с параметром **-q**, чтобы довести его до максимально возможного приоритета. Этот параметр может быть полезен, если вы пытаетесь отследить процесс, который уже привел систему в состояние сбоя.

Нам также нравится программа **htop**, кросс-платформенный интерактивный просмотрщик процессов с открытым исходным кодом, предлагающий больше возможностей и имеющий более удобный интерфейс, чем **top**. Она пока недоступна в качестве пакета в наших иллюстративных системах, но вы можете загрузить двоичную или исходную версию с веб-сайта разработчика по адресу `hisham.hm/htop`.

[6] В системах FreeBSD можно установить переменную окружения **TOP**, чтобы передать дополнительные аргументы команде **top**. Мы рекомендуем использовать параметр **-H**, показать все потоки для многопоточных процессов, а не просто сводную информацию, а также параметр **-P** для отображения всех ядер процессора. Добавьте в файл инициализации оболочки строку **TOP="-HP"** , чтобы эти изменения сохранялись между сеансами оболочки.

4.5. Команды nice и renice: изменение приоритета выполнения

Фактор уступчивости — это число, по которому ядро определяет свою политику в отношении процессов, конкурирующих за право доступа к центральному процессору.[7] Чем выше фактор уступчивости, тем ниже приоритет процесса и наоборот, отсюда и название термина. Низкое или отрицательное значение означает использование высокого приоритета: процесс ведет себя не слишком уступчиво.

В настоящее время администраторам редко приходится определять приоритеты вручную. Сегодня, когда на рабочих столах стоят намного более быстродействующие компьютеры, системный планировщик, как правило, обслуживает все процессы весьма оперативно. Добавление классов планирования предоставляет разработчикам дополнительные средства управления в тех случаях, когда важна быстрая ответная реакция.

Диапазон допустимых значений фактора уступчивости зависит от используемой системы. В частности, в системе Linux используется диапазон от –20 до +19, а в системе FreeBCD — от –20 до +20.

Если пользователь не предпринимает специальных мер, дочерний процесс наследует приоритет своего родительского процесса. Владелец процесса может увеличить фактор уступчивости, но не может уменьшить его, даже чтобы вернуться к стандартному значению. Это не позволяет процессам с низким приоритетом порождать высокоприоритетных потомков. Однако суперпользователь может устанавливать произвольные значения фактора уступчивости.

К сожалению, уровень производительности подсистемы ввода-вывода растет не так стремительно, как быстродействие центральных процессоров, поэтому жесткие диски стали основным узким местом в большинстве операционных систем. Фактор уступчивости никак не влияет на подсистемы управления памятью и вводом-выводом, поэтому даже низкоприоритетный процесс способен монополизировать эти ресурсы или захватить непропорционально большую их часть.

Фактор уступчивости можно установить при создании процесса. Это делается с помощью команды **nice**. Команда **renice** позволяет изменять приоритет выполняемого процесса. Первая из этих команд принимает в качестве аргумента строку запуска процесса, а вторая — идентификатор процесса либо имя пользователя.

Приведем примеры.

```
$ nice -n 5 ~/bin/longtask  // Понижаем приоритет (увеличиваем
                            // фактор уступчивости) на 5
$ sudo renice -5 8829       // Задаем фактор уступчивости равным -5
$ sudo renice 5 -u boggs    // Задаем фактор уступчивости процессов
                            // пользователя "boggs" равным 5
```

К сожалению, в системах по-разному реализован способ установки желаемого приоритета; более того, даже в рамках одной и той же системы не согласованы механизмы действия команд **nice** и **renice**. Ситуацию усложняет то, что существует версия команды **nice**, встроенная в оболочки языка C и ряд других популярных интерпретаторов (за исключением **bash**). Если не указать полное имя команды, будет вызвана именно встроенная версия, а не системная. Для того чтобы избежать неоднозначности, рекомендуется использовать полный путь к системной версии команды, который можно найти в файле **/usr/bin/nice**.

[7] Команда **nice** определяет только приоритет использования центрального процессора. Для управления приоритетами ввода-вывода используется команда **ionice**.

Все эти варианты приведены в табл. 4.3. Элемент *приор* для среды, в которой вызывается команда `nice` или `renice`, означает абсолютное значение фактора уступчивости, а *инкр* — относительное. Знаки "плюс" в команде **nice** обязательны только для интерпретатора команд; в остальных случаях они игнорируются.

Таблица 4.3. Как выражаются приоритеты в различных версиях команд `nice` и `renice`

Система	Диапазон	ОС-команда `nice`	Команда `nice` в оболочке `csh`	Команда `renice`
Linux	-20–19	**-n** *инкр*	*+инкр* или *-инкр*	*приор* или **-n** *приор*
FreeBSD	-20–20	**-n** *инкр*	*+инкр* или *-инкр*	*инкр* или **-n** *инкр*

4.6. Файловая система `/proc`



Версии команд **ps** и **top** считывают информацию о состоянии процессов из каталога **/proc** — псевдофайловой системы, в которой ядро помещает большой объем интересной информации о состоянии системы.

Несмотря на имя **/proc** (и имя базового типа файловой системы — "proc"), хранящаяся в этом каталоге информация не ограничивается одними лишь процессами — здесь хранится вся информация о состоянии и статистические сведения, генерируемые ядром. Некоторые параметры можно даже изменять, записывая новые значения в соответствующий файл каталога **/proc**, — ряд примеров приведен в разделе 11.4.

Хотя большую часть информации проще получать с помощью таких интерфейсных команд, как **vmstat** и **ps**, некоторые данные придется считывать непосредственно из каталога **/proc**. Следовательно, целесообразно просмотреть его, чтобы ознакомиться со всеми помещенными в него файлами. Команда **man proc** позволяет ознакомиться с рядом полезных советов и приемов.

Поскольку ядро создает содержимое файлов каталога **/proc** на лету (во время их считывания), большинство из них выглядят пустыми при их открытии с помощью команды **ls -l**. Для просмотра действительного содержимого этих файлов придется прибегнуть к командам **cat** или **less**. Однако будьте осторожны: некоторые файлы содержат двоичные данные либо ссылаются на двоичные данные, непосредственный просмотр которых может поставить в тупик эмулятор терминала.

Информация, относящаяся к конкретным процессам, распределена по подкаталогам, названным по идентификаторам процессов. Например, каталог **/proc/1** всегда содержит информацию о демоне **init**. Наиболее полезные файлы с информацией о процессах перечислены в табл. 4.4.

Таблица 4.4. Файлы с информацией о процессах каталога `/proc` (нумерованные подкаталоги)

Файл	Содержимое
cgroup	Группа управления, которой принадлежит процесс
cmd	Команда или программа, выполняемая процессом
cmdline[a]	Полная командная строка процесса (разделенная нулями)
cwd	Символическая ссылка на текущий каталог процесса
environ	Переменные среды процесса (разделенные нулями)
exe	Символическая ссылка на файл, который должен выполняться

Окончание табл. 4.4

Файл	Содержимое
`fd`	Подкаталог, содержащий ссылки на дескрипторы каждого открытого файла
`fdinfo`	Подкаталог, содержащий дополнительную информацию о дескрипторах каждого открытого файла
`maps`	Информация отображения памяти (сегменты совместного использования, библиотеки и т.п.)
`ns`	Подкаталог, содержащий ссылку на пространство имен каждого открытого файла
`root`	Символическая ссылка на корневой каталог процесса (определенный командой `chroot`)
`stat`	Информация об общем состоянии процесса (для ее получения лучше всего использовать команду `ps`)
`statm`	Информация об использовании памяти

[a]Может быть недоступна, если запись информации о процессе выполняется из памяти.

Отдельные компоненты внутри файлов `cmdline` и `environ` разделены символами нуля, а не символами перевода строки. Для того чтобы их содержимое было более читабельным, его можно фильтровать с помощью команды `tr "\000" "\n"`.

В подкаталоге `fd` открытые файлы представлены символическими ссылками. Дескрипторы файлов, которые связаны с каналами или сетевыми сокетами, не имеют связанных с ними имен файлов. Вместо этого в качестве целевой ссылки ядро предоставляет обобщенное описание.

Файл `maps` полезен при определении библиотек, с которыми связана или от которых зависит та или иная программа.



В системе FreeBSD также используется файловая система на основе каталога `/proc`. Однако она объявлена устаревшей. По соображениям совместимости ее все еще можно смонтировать, но только не по умолчанию. Для этого можно выполнить команду[8]

```
freebsd$ sudo mount -t procfs proc /proc
```

Схема файловой системы похожа на версию файловой системы `procfs` в системе Linux, но не совпадает с ней. Информация о процессе включает в себя его состояние, символическую ссылку на выполняемый файл, детали виртуальной памяти, выделенной процессу, а также другую подробную информацию (см. также справочную страницу `man procfs`).

4.7. Команды strace и truss: отслеживание сигналов и системных вызовов

Иногда определение действий, действительно выполняемых данным процессом, может быть затруднительным. Умозаключение приходится делать на основе косвенных данных, полученных от файловой системы и с помощью таких средств, как программа `ps`.

Если источников информации недостаточно, можно непосредственно следить за процессами с помощью команд `strace` (в системе Linux она предоставляется в виде дополнительного пакета) или `truss` (в системе FreeBSD). Эти команды отображают каждый системный вызов, выполняемый процессом, и каждый получаемый им сигнал.

[8]Чтобы автоматически монтировать файловую систему `/proc` во время загрузки, добавьте в файл в `/etc/fstab` строку `proc /proc procfs rw 0 0`.

Команды **strace** и **truss** можно связать с выполняемым процессом, последить за ним в течение некоторого времени, а затем отключиться от процесса, не прерывая его[9].

Хотя системные вызовы выполняются на сравнительно низком уровне абстракции, обычно вывод команды **strace** позволяет получить достаточно полную информацию об активности процесса. Например, следующий журнал был получен в результате выполнения команды **strace** применительно к активной копии команды **top** (выполняемой с идентификатором PID, равным 5810).

```
redhat$ sudo strace -p 5810
gettimeofday( {1116193814, 213881}, {300, 0} )                = 0
open("/proc", O_RDONLY|O_NONBLOCK|O_LARGEFILE|O_DIRECTORY)    = 7
fstat64(7, {st_mode=S_IFDIR|0555, st_size=0, ...} )           = 0
fcntl64(7, F_SETFD, FD_CLOEXEC)                               = 0
getdents64(7, /* 36 entries */, 1024)                         = 1016
getdents64(7, /* 39 entries */, 1024)                         = 1016
stat64("/proc/1", {st_mode=S_IFDIR|0555, st_size=0, ...} )    = 0
open("/proc/1/stat", O_RDONLY)                                = 8
read(8, "1 (init) S 0 0 0 0 -1 4194560 73"..., 1023)          = 191
close(8)                                                      = 0
...
```

Команда **strace** не только отображает имя каждого системного вызова, выполненного процессом, но и раскрывает аргументы и отображает результирующий код, возвращенный ядром.

В этом примере команда **top** начинает свою работу с проверки текущего значения времени. Затем она открывает каталог **/proc** и считывает его содержимое, тем самым получая список процессов, выполняемых в текущий момент. Команда **top** обращается к статической копии каталога, представляющей процесс **init**, а затем открывает **/proc/1/stat**, чтобы прочесть информацию о состоянии этого процесса.

Вывод результатов системного вызова часто позволяет выявлять ошибки, которые не сообщаются самим процессом. Например, ошибки разрешения файловой системы или конфликты сокетов обычно довольно очевидны в выводе команд **strace** или **truss**. Ищите системные вызовы, возвращающие показания ошибок, и проверяйте ненулевые значения.

Команда **strace** снабжена флагами, которые описаны на соответствующей man-странице. Например, флаг **-f** используется для разветвленных процессов, и его полезно применять для отслеживания демонов (например, **httpd**), которые порождают множество дочерних процессов. Опция **-e trace=file** позволяет отображать только файловые операции, что особенно удобно для определения местоположения "неуловимых" файлов конфигурации.

Ниже приведен аналогичный пример использования команды **truss** в системе FreeBSD. В данном случае мы хотим увидеть, что делает команда **cp** при копировании файлов.

```
freebsd$ truss cp /etc/passwd /tmp/pw
...
lstat("/etc/passwd",{ mode=-rw-r--r-- ,inode=13576,size=2380,
    blksize=4096 }) = 0 (0x0)
umask(0x1ff)                                    = 18 (0x12)
umask(0x12)                                     = 511 (0x1ff)
```

[9]По крайней мере, как правило. Команда **strace** может прерывать системные вызовы. В этих случаях отслеживаемый процесс должен быть подготовлен для повторного запуска этих вызовов. Таково стандартное правило выполнения программ в среде UNIX, но оно не всегда соблюдается.

```
fstatat(AT_FDCWD,"/etc/passwd",{ mode=-rw-r--r-- ,inode=13576,
    size=2380,blksize=4096 },0x0) = 0 (0x0)
stat("/tmp/pw",0x7fffffffe440) ERR#2 'No such file or directory'
openat(AT_FDCWD,"/etc/passwd",O_RDONLY,00)          = 3 (0x3)
openat(AT_FDCWD,"/tmp/pw",O_WRONLY|O_CREAT,0100644) = 4 (0x4)
mmap(0x0,2380,PROT_READ,MAP_SHARED,3,0x0) = 34366304256
    (0x800643000)
write(4,"# $FreeBSD: releng/11.0/etc/mast"...,2380) = 2380 (0x94c)
close(4)                                            = 0 (0x0)
close(3)                                            = 0 (0x0)
...
```

После выделения памяти и открытия зависимостей библиотеки (не показано) команда **cp** использует системный вызов **lstat** для проверки текущего состояния файла **/etc/passwd**. Затем она применяет команду **stat** к пути предполагаемой копии **/tmp/pw**. Этот файл еще не существует, поэтому команда **stat** дает сбой, а команда **truss** сообщает об ошибке "Нет такого файла или каталога".

Затем команда **cp** выполняет системный вызов **openat** (с параметром O_RDONLY) для чтения содержимого файла **/etc/passwd**, за которым следует команда **openat**, применяемая к пути **/tmp/pw** с параметром O_WRONLY для создания нового файла назначения. Затем она отображает содержимое файла **/etc/passwd** в память (с помощью команды **mmap**) и записывает данные с помощью команды **write**. Наконец, команда **cp** выполняет уборку, закрывая оба дескриптора файлов.

Трассировка системных вызовов — мощный инструмент для отладки в руках системных администраторов. Переходите к этим инструментам после того, как исчерпаете более традиционные способы, такие как просмотр файлов журнала и настройка процесса для подробного вывода. Не пугайтесь плотного вывода; обычно достаточно сосредоточиться на читаемых человеком частях.

4.8. Процессы, вышедшие из-под контроля

Процессы, вышедшие из-под контроля, — это процессы, которые поглощают значительно больше ресурсов системы, диска или сети системы, чем требует их обычная роль или ожидаемое поведение. Иногда такие программы имеют свои собственные ошибки, которые привели к искажению поведения. В других случаях они не справляются надлежащим образом с восходящими отказами и застревают в замкнутых циклах. Например, процесс может повторно запускать одну и ту же неудачную операцию снова и снова, перегружая центральный процессор. В еще одной категории случаев ошибки не существует, но программное обеспечение просто неэффективно в реализации и жадно поглощает ресурсы системы.

Все эти ситуации заслуживают внимания со стороны системного администратора, причем не только потому, что процесс, вышедший из-под контроля, скорее всего, работает некорректно, но также и потому, что он обычно мешает работе других процессов, которые выполняются в системе.

Граница между патологическим и нормальным поведением при большой нагрузке является неопределенной. Часто первый шаг в диагностике заключается в том, чтобы выяснить, какое из этих явлений вы фактически наблюдаете. Как правило, системные процессы должны вести себя разумно, поэтому очевидное неправильное поведение со стороны одного из этих процессов автоматически вызывает подозрение. Пользовательские процессы, такие как веб-серверы и базы данных, могут быть просто перегружены.

Вы можете определить процессы, которые используют чрезмерное процессорное время, просматривая результаты работы команд **ps** или **top**. Также проверьте средние значения загрузки системы, о которых сообщает команда **uptime**. Традиционно эти значения определяют среднее количество процессов, которые были запущены за предыдущие 1-, 5- и 15-минутные интервалы. В системе Linux среднее значение нагрузки также учитывает занятость, вызванную дисковым трафиком и другими формами ввода-вывода.

Для систем с привязкой к центральному процессору средние нагрузки должны быть меньше общего количества ядер центрального процессора, доступных в вашей системе. Если это не так, система перегружена. В системе Linux общее использование центрального процессора можно выяснить с помощью команд **top** или **ps**, которые позволяют определить, зависят ли средние значения нагрузки от загрузки процессора или системы ввода-вывода. Если загрузка процессора составляет около 100%, это, вероятно, является узким местом.

Процессы, которые избыточно используют физическую память системы, могут вызвать серьезные проблемы с производительностью. Вы можете проверить размер памяти, используемой процессами, выполнив команду **top**. Столбец VIRT показывает общий объем виртуальной памяти, выделяемый каждым процессом, а столбец RES показывает часть этой памяти, отображаемую в настоящее время на определенные страницы памяти ("резидентный набор").

Оба эти числа могут включать общие ресурсы, такие как библиотеки, и, следовательно, потенциально вводить в заблуждение. Более прямой показатель потребления памяти, используемой процессом, указан в столбце DATA, который по умолчанию не отображается. Чтобы добавить этот столбец в таблицу вывода команды **top**, нажмите клавишу <f> сразу после запуска команды **top** и выберите пункт DATA из списка, нажав клавишу пробела. Значение DATA указывает объем памяти в сегментах данных и стека каждого процесса, поэтому он относительно специфичен для отдельных процессов (без учета разделов общей памяти). Следите за ростом этого показателя с течением времени, а также его абсолютной величиной. В системе FreeBSD существует эквивалентный показатель SIZE, который отображается по умолчанию.

Постарайтесь лучше понять, что происходит, прежде чем прекращать процесс, который по всей видимости вышел из-под контроля. Лучший путь для решения проблемы и предотвращения ее повторения — это живой пример, который можно исследовать. Как только вы уничтожите неудачный процесс, большинство доступных доказательств исчезнет.

Помните о возможности взлома. Вредоносная программа, как правило, не проверяется на правильность в различных средах, поэтому более вероятно, что в среднем возникнет какое-то вырожденное состояние. Если вы подозреваете неладное, получите трассировку системного вызова с помощью команд **strace** или **truss**, чтобы понять, что делает данный процесс (например, взламывает пароли) и где хранятся его данные.

Процессы, вышедшие из-под контроля, могут заполнить всю файловую систему, что вызовет многочисленные проблемы. Когда файловая система заполняется, на консоль записывается большое количество сообщений, и попытки записи в файловую систему будут вызывать сообщения об ошибках.

Первое, что нужно сделать в этой ситуации, — определить, какая файловая система заполнена и какой файл ее заполняет. Команда **df -h** показывает использование диска файловой системы в единицах, читаемых человеком. Найдите файловую систему, запол-

ненную на 100% или более.[10] Используйте команду **du -h** в идентифицированной файловой системе, чтобы определить, какой из каталогов использует наибольшее пространство. Повторяйте выполнять команду **du** до тех пор, пока не будут обнаружены большие файлы.

Команды **df** и **du** описывают использование диска по-разному. Команда **df** сообщает о дисковой памяти, используемой смонтированной файловой системой, в соответствии с показателями, записанными в метаданных файловой системы. Команда **du** суммирует размеры всех файлов в данном каталоге. Если файл отсоединен (удален) от файловой системы, но по-прежнему ссылается на какой-то запущенный процесс, команда **df** сообщает о занимаемом пространстве, а команда **du** — нет. Это несоответствие сохраняется до тех пор, пока дескриптор открытого файла не будет закрыт или файл не будет усечен. Если вы не можете определить, какой процесс использует файл, попробуйте запустить команды **fuser** и **lsof** (они подробно описаны в разделе 5.2), чтобы получить дополнительную информацию.

4.9. Периодические процессы

Часто бывает полезно иметь сценарий или команду, выполняемую без вмешательства человека. К таким ситуациям относятся запланированное создание резервных копий, операции по обслуживанию базы данных или выполнение ночных пакетных заданий. Как это типично для UNIX и Linux, существует более чем один способ достижения этой цели.

Демон cron: команды расписания

Демон **cron** является традиционным инструментом для запуска команд в соответствии с заданным расписанием. Он запускается, когда система загружается, и выполняется до тех пор, пока система работает. Существует несколько реализаций демона **cron**, но, к счастью для администраторов, синтаксис и функциональность различных версий почти идентичны.

RHEL  По причинам, которые остаются неясными, демон **cron** в системе Red Hat был переименован в **crond**. Однако это все тот же самый **cron**, который все мы знаем и любим.

Демон **cron** считывает файлы конфигурации, содержащие списки команд и время, в которое они должны быть вызваны. Командные строки выполняются оболочкой **sh**, поэтому почти все, что вы можете сделать вручную из оболочки, также можно выполнить с помощью демона **cron**. При желании можно настроить **cron** для использования другой оболочки.

Файл конфигурации **cron** называется **crontab** (сокращение слов "cron table"). Файл **crontab** для отдельных пользователей хранится в каталоге **/var/spool/cron** (Linux) или **/var/cron/tabs** (FreeBSD). Существует не более одного файла **crontab** для каждого пользователя. Файлы **crontab** представляют собой текстовые файлы с именами пользователей, которым они принадлежат. Демон **cron** использует эти имена файлов (и их владельца), чтобы выяснить, какой идентификатор UID применять при запуске команд, содержащихся в каждом файле. Команда **crontab** передает файлы **crontab** в этот каталог и выгружает их оттуда.

[10] В большинстве реализаций около 5% памяти на диске резервируется "на крайний случай", но процесс, запущенный с правами **root**, может заполнить даже это пространство, и в результате система сообщает, что степень заполнения диска превышает 100%.

Демон **cron** пытается минимизировать время, затрачиваемое на повторный синтаксический анализ файлов конфигурации и создания расписания запуска задач. Команда **crontab** помогает поддерживать эффективность демона **cron**, уведомляя **cron** при изменении файлов **crontab**. Следовательно, нельзя редактировать файлы **crontab** напрямую, потому что этот подход может привести к тому, что демон **cron** не заметит ваши изменения. Если вы столкнулись с ситуацией, когда демон **cron**, похоже, не признает модифицированный файл **crontab**, сигнал HUP, отправленный в процесс **cron**, в большинстве систем заставит его перезагрузиться.

Демон **cron** обычно работает молча, но большинство версий могут вести файл журнала (обычно **/var/log/cron**), в котором перечислены команды, которые были выполнены, и моменты времени, когда они запускались. Взгляните на файл журнала **cron**, если у вас возникли проблемы с работой демона **cron** и вы не можете понять, почему.

🕮 Дополнительную информацию о журнале syslog см. в главе 10.

Формат файлов crontab

Все файлы **crontab** в системе имеют одинаковый формат. Комментарии вводятся после знака фунта (#) в первом столбце строки. Каждая строка без комментария содержит шесть полей и представляет одну команду:

минута час день месяц день_недели команда

Первые пять полей сообщают демону **cron**, когда нужно выполнить команду. Они разделены пробелами, но внутри поля *команды* пробелы передаются прямо в оболочку. Поля в спецификации времени интерпретируются, как показано в табл. 4.5. Запись в файле **crontab** часто называют *заданием планировщика*.

Таблица 4.5. Спецификации времени в файле crontab

Поле	Описание	Диапазон
минута	Минута часа	от 0 до 59
час	Час дня	0 до 23
день	День месяца	с 1 по 31
месяц	Месяц года	с 1 по 12
день недели	День недели	0 до 6 (0 = воскресенье)

Каждое из связанных с временем полей может содержать следующую информацию.

- Звездочка, которая может означать любое значение.
- Единственное целое число, которое означает точно заданное значение.
- Два целых числа, разделенных дефисом, соответствующие диапазону значений.
- Диапазон, за которым следует косая черта и значение шага, например 1-10/2.
- Список целых чисел или диапазонов, разделенных запятыми, соответствующих любому значению.

Например, спецификация времени

```
45    10    *    *    1-5
```

означает "10:45, с понедельника по пятницу". Подсказка: никогда не используйте звездочки во всех полях, если не хотите, чтобы команда запускалась каждую минуту, что полезно только при тестировании сценариев. Одна минута — минимальный шаг, доступный для работы демона **cron**.

Диапазоны времени в файле **crontab** могут включать значение шага. Например, ряд `0,3,6,9,12,15,18` можно записать более кратко как `0-18/3`. Можно также использовать трехбуквенную текстовую мнемонику для имен месяцев и дней, но не в сочетании с диапазонами. Насколько нам известно, эта функция работает только с английскими именами.

Существует потенциальная двусмысленность между полями *день_недели* и день месяца. Каждый день — это и день недели и день месяца. Если указаны значения как *день_недели*, так и *день*, то *день* должен удовлетворять только одному из двух условий, которые необходимо выбрать.

Например, спецификация времени

```
0,30 * 13 * 5
```

означает "каждые полчаса в пятницу и каждые полчаса 13-го числа", а не "каждые полчаса в пятницу 13-го".

Команда представляет собой командную строку оболочки **sh**, которая должна быть выполнена. Это может быть любая допустимая команда оболочки и ее не нужно брать в кавычки. Предполагается, что команда продолжается до конца строки и может содержать пробелы или знаки табуляции.

Знаки процента (`%`) указывают символы новой строки в поле *команда*. В фактическую команду включается только текст, стоящий до первого знака `%`. Остальные строки передаются *команде* в качестве стандартного ввода. Используйте обратную косую черту (\) в качестве управляющего символа в командах с значащим знаком процента, например `date + \%s`.

Хотя оболочка **sh** участвует в выполнении команды, она не действует как оболочка регистрации (login shell) и не читает содержимое файлов **~/.profile** или **~/.bash_profile**. В результате переменные окружения команды могут быть настроены несколько иначе, чем ожидаемые. Если команда работает нормально из-под оболочки, но терпит сбой при включении в файл **crontab**, вероятным виновником является окружение. Если это необходимо, всегда можете обернуть свою команду сценарием, устанавливающим соответствующие переменные окружения.

Мы также рекомендуем использовать полный путь к команде, чтобы задание выполнялось должным образом, даже если параметр `PATH` не установлен так, как ожидалось. Например, следующая команда каждую минуту записывает дату и время безотказной работы в файл в домашнем каталоге пользователя:

```
* * * * * echo $(/bin/date) - $(/usr/bin/uptime) >> ~/uptime.log
```

Кроме того, вы можете явно задать переменные среды в верхней части файла **crontab**:

```
PATH = /bin:/USR/bin

* * * * * echo $(date) - $(uptime) >> ~/uptime.log
```

Вот еще несколько примеров корректных записей в файле **crontab**:

```
*/10 * * * 1,3,5 echo ruok | /usr/bin/nc localhost 2181 |
    mail -s "TCP-port 2181 status" ben@admin.com
```

Эта строка отправляет результаты проверки подключения к порту 2181 каждые 10 минут по понедельникам, средам и пятницам. Поскольку демон **cron** выполняет команду посредством оболочки **sh**, специальные символы оболочки, такие как каналы и перенаправления, функционируют должным образом.

```
0 4 * * Sun (/usr/bin/mysqlcheck -u maintenance --optimize --all-databases)
```

Эта запись запускает программу обслуживания **mysqlcheck** по воскресеньям в 4:00 утра. Поскольку выход не сохраняется в файле или отбрасывается иным образом, он будет отправлен по электронной почте владельцу файла **crontab**.

```
20 1 * * * find /tmp -mtime +7 -type f -exec rm -f { } ';'
```

Эта команда выполняется каждую ночь в 1:20. Она удаляет все файлы в каталоге **/tmp**, которые не были изменены за семь дней. Символы **';'** в конце строки означают конец аргументов подкоманды для поиска.

Демон **cron** не пытается компенсировать команды, которые не были запущены, пока система не работала. Тем не менее он хорошо разбирается в настройках времени, например при переходе на летнее и зимнее время.

Если ваше задание **cron** является сценарием, обязательно сделайте его исполняемым (с помощью команды **chmod +x**), иначе демон **cron** не сможет его выполнить. В качестве альтернативы настройте команду **cron** для непосредственного выполнения вашего сценария прямо из командной оболочки (например, **bash -c ~/bin/myscript.sh**).

Управление файлом crontab

Команда **crontab** *имя_файла* устанавливает файл с именем имя_файла в качестве вашего файла заданий, заменяя любую предыдущую версию. Команда **crontab -e** проверяет копию вашего файла **crontab**, применяет к нему свой редактор (заданный переменной среды EDITOR), а затем повторно передает его в каталог **crontab**. Команда **crontab -l** выводит содержимое вашего файла **crontab** на стандартный вывод, а **crontab -r** удаляет его, оставляя вас без файла **crontab** вообще.

Суперпользователь может предоставить аргумент имени пользователя для редактирования или просмотра файлов **crontab** других пользователей. Например, команда **crontab -r jsmith** стирает файл **crontab**, принадлежащий пользователю jsmith, а команда **crontab -e jsmith** редактирует его. Система Linux допускает использование как имени пользователя, так и имени файла в одной и той же команде, поэтому имя пользователя должно быть указано с префиксом **-u** для устранения неоднозначности (например, **crontab -u jsmith crontab.new**).

Без аргументов командной строки большинство версий команды **crontab** пытаются читать файлы **crontab** со стандартного ввода. Если вы ввели эту команду случайно, не пытайтесь выйти с помощью клавиш <Ctrl+D>; если вы сделаете это, то сотрете файл **crontab**. Вместо этого используйте клавиши <Ctrl+C>. Система FreeBSD требует, чтобы вы предоставили тире в качестве аргумента имени файла, чтобы заставить команду **crontab** обратить внимание на ее стандартный ввод. Остроумно!

Многие организации испытывают малозаметные, но повторяющиеся сетевые сбои, которые возникают, потому что системные администраторы настроили демон **cron** для выполнения одной и той же команды на сотнях машин в одно и то же время, вызывая задержки или чрезмерную нагрузку. Проблема синхронизации часов с сетевым протоколом NTP усугубляет проблему. Эту проблему легко исправить с помощью сценария случайной задержки.

Демон **cron** регистрирует свою деятельность в журнале **syslog**, используя функцию **cron**, причем большинство сообщений отправляется на уровне **info**. Конфигурации журнала **syslog**, заданные по умолчанию, обычно отправляют данные журнала **cron** в отдельный файл.

Другие файлы crontab

Помимо пользовательских файлов crontab, команда **cron** также подчиняется системным записям, занесенным в файл **/etc/crontab** и в каталог **/etc/cron.d**. Эти файлы имеют немного отличающийся формат от файлов **crontab** для каждого пользователя: они позволяют командам запускаться от имени произвольного пользователя. Дополнительное поле имени пользователя находится перед именем команды. Поле имени пользователя отсутствует в файлах crontab, поскольку имя файла **crontab** и так предоставляет эту информацию.

В общем случае, файл **/etc/crontab** — это файл для системных администраторов, который можно сохранить вручную, тогда как **/etc/cron.d** — это своего рода хранилище, в которое пакеты программного обеспечения могут устанавливать любые записи конфигурации демона **cron**, которые им могут понадобиться. Файлы в каталоге **/etc/cron.d** по умолчанию называются именами пакетов, которые их установили, но в демоне **cron** это соглашение не применяется.



В дистрибутивах Linux также предустановлены записи конфигурации демона **cron**, которые запускают сценарии в наборе известных каталогов, тем самым обеспечивая еще один способ для пакетов программного обеспечения устанавливать периодические задания без какого-либо редактирования файла **crontab**. Например, сценарии в каталогах **/etc/cron.hourly**, **/etc/cron.daily** и **/etc/cron.weekly** запускаются ежечасно, ежедневно и еженедельно соответственно.

Контроль доступа к демону cron

Два файла конфигурации указывают, какие пользователи могут отправлять файлы **crontab**: в системе Linux — **/etc/cron.{allow,deny}**, а в системе FreeBSD — **/var/cron/{allow,deny}**. Многие стандарты безопасности требуют, чтобы файлы **crontab** был доступны только для обслуживания учетных записей или для пользователей, имеющих на это обоснованное право. Файлы **allow** и **deny** способствуют соблюдению этих требований.

Если файл **cron.allow** существует, то он содержит список всех пользователей, которые могут отправлять файлы **crontab**, по одному в строке. Никто другой не может вызвать команду **crontab**. Если файл **cron.allow** не существует, то проверяется файл **cron.deny**. Это тоже список пользователей, но имеющий противоположный смысл: доступ разрешен каждому, за исключением перечисленных пользователей.

Если ни файл **cron.allow**, ни файл **cron.deny** не существуют, то по умолчанию (при отсутствии общепринятого соглашения это может произойти случайно) система либо разрешает всем пользователям вводить файлы **crontab**, либо предоставляет доступ к файлу **crontab** только суперпользователю. На практике начальная конфигурация обычно включается в инсталляцию операционной системы по умолчанию, поэтому вопрос о том, как команда **crontab** может вести себя без файлов конфигурации, является спорным. Большинство конфигураций по умолчанию позволяют всем пользователям получать доступ к демону **cron**.

Важно отметить, что в большинстве систем управление доступом осуществляется программой **crontab**, а не **cron**. Если пользователь может записать файл **crontab** в соответствующий каталог другими способами, то демон **cron** будет слепо исполнять содержащиеся в нем команды. Поэтому очень важно назначить владельцем пользователя **root** для каталогов **/var/spool/cron** и **/var/cron/tabs**. Дистрибутивы операционных систем всегда правильно устанавливают разрешения по умолчанию.

Системные таймеры

В соответствии с задачей дублировать функции всех других подсистем Linux команда **systemd** включает в себя концепцию таймеров, которые активируют данную службу **systemd** по предопределенному расписанию. Таймеры мощнее, чем записи **crontab**, но их сложнее настраивать и ими труднее управлять. Некоторые дистрибутивы Linux (например, CoreOS) полностью отказались от демона **cron** в пользу таймеров **systemd**, но наши иллюстративные системы продолжают использовать демон **cron** и запускать его по умолчанию.

🕮 Введение в **systemd** и **units** см. в главе 2.

У нас нет полезных советов относительно выбора между таймерами **systemd** и файлами **crontab**. Используйте то, что вы предпочитаете для любой заданной задачи. К сожалению, у вас на самом деле нет возможности стандартизовать одну или другую систему, потому что программные пакеты добавляют свои задания в произвольную систему по своему выбору. Вам всегда нужно проверять обе системы, когда вы пытаетесь выяснить, как выполняется конкретное задание.

Структура таймеров systemd

Системный таймер состоит из двух файлов:

- модуль таймера, который описывает расписание и устройство для активации;
- модуль службы, в котором указаны сведения о том, что нужно запустить.

В отличие от записей **crontab**, таймеры **systemd** могут быть описаны как в абсолютных календарных терминах ("среда в 10:00 утра"), так и в терминах, относящихся к другим событиям ("через 30 секунд после загрузки системы"). Параметры объединяются, чтобы позволить создавать сложные выражения, которые не имеют ограничений, характерных для демона **cron**. Параметры выражения времени описаны в табл. 4.6.

Таблица 4.6. Типы таймеров systemd

Тип	Время
`OnActiveSec`	Относительно времени, в которое активируется сам таймер
`OnBootSec`	Относительно времени загрузки системы
`OnStartupSec`	Относительно времени, когда система была запущена
`OnUnitActiveSec`	Относительно времени, в течение которого указанное устройство было активным
`OnUnitInactiveSec`	Относительно времени, в течение которого указанное устройство неактивно
`OnCalendar`	Определенный день и время

Как показывают их имена, значения для этих параметров таймера указаны в секундах. Например, `OnActiveSec=30` означает 30 секунд после включения таймера. Фактически это значение может быть любым действительным выражением времени **systemd**, как более подробно описано ниже.

Пример таймера systemd

Системы Red Hat и CentOS включают предварительно сконфигурированный таймер **systemd**, который удаляет временные файлы системы один раз в день. Ниже мы более подробно рассмотрим пример.

Вначале перечислим все определенные командой **systemctl** таймеры. (Мы приводим вывод в виде таблицы, чтобы сделать его доступным для чтения. Обычно каждый таймер создает одну длинную строку вывода.)

```
redhat$ systemctl list-timers
NEXT         Sun 2017-06-18 10:24:33 UTC
LEFT         18h left
LAST         Sat 2017-06-17 00:45:29 UTC
PASSED       15h ago
UNIT         systemd-tmpfiles-clean.timer
ACTIVATES    systemd-tmpfiles-clean.service
```

В выводе перечисляются имя таймера и имя активируемой службы. Так как это системный таймер по умолчанию, файл модуля находится в стандартном каталоге **/usr/lib/systemd/system**. Вот что записано в модуле таймера:

```
redhat$ cat /usr/lib/systemd/system/systemd-tmpfiles-clean.timer
[Unit]
Description=Daily Cleanup of Temporary Directories
[Timer]
OnBootSec=15min
OnUnitActiveSec=1d
```

Таймер сначала активируется через 15 мин. после загрузки, а затем запускается один раз в день. Обратите внимание на то, что необходим какой-то триггер для начальной активации (здесь — `OnBootSec`). Не существует единой спецификации, которая обеспечивала бы эффект "каждые X минут" самостоятельно.

Проницательные читатели заметят, что таймер фактически не указывает, какой модуль нужно запустить. По умолчанию таймер **systemd** ищет модуль службы с тем же именем, что и таймер. Целевой модуль можно явно задать с помощью параметра `Unit`. В этом случае соответствующий модуль службы не имеет никаких сюрпризов.

```
redhat$ cat /usr/lib/systemd/system/systemd-tmpfiles-clean.service
[Unit]
Description=Cleanup of Temporary Directories
DefaultDependencies=no
Conflicts=shutdown.target
After=systemd-readahead-collect.service systemd-readahead-replay.service
   local-fs.target time-sync.target
Before=shutdown.target

[Service]
Type=simple
ExecStart=/usr/bin/systemd-tmpfiles --clean
IOSchedulingClass=idle
```

Целевую службу можно запустить напрямую (т.е. независимо от таймера) с помощью команды **systemctl start systemd-tmpfiles-clean**, как и любую другую службу. Этот факт значительно облегчает отладку запланированных задач, которые могут быть источником значительных административных сложностей при использовании демона **cron**.

Для того чтобы создать собственный таймер, создайте файлы **.timer** и **.service** в каталоге **/etc/systemd/system**. Если хотите, чтобы таймер запускался при загрузке, добавьте следующий текст:

```
[Install]
WantedBy = multi-user.target
```

в конце файла блока таймера. Не забудьте включить таймер во время загрузки с помощью команды **systemctl enable**. (Вы также можете сразу запустить таймер с помощью команды **systemctl start**.)

Параметр таймера `AccuracySec` задерживает его активацию случайным количеством времени в пределах указанного временного окна. Эта функция удобна, когда таймер работает на большой группе сетевых машин, и вы хотите, чтобы все таймеры не срабатывали в один и тот же момент. (Напомним, что с помощью демона **cron** для достижения этого эффекта необходимо использовать сценарий случайной задержки.)

По умолчанию параметр `AccuracySec` равен 60 сек. Если вы хотите, чтобы ваш таймер выполнялся в точно запланированное время, используйте настройку `AccuracySec = 1ns`. (Точность, равная наносекунде, вероятно, будет достаточной. Впрочем, обратите внимание на то, что вы на самом деле не получите точность наносекунды.)

Определение времени в таймере systemd

Таймеры позволяют гибко задавать даты, время и интервалы. Страница руководства **systemd.time** является авторитетным справочником по грамматике спецификаций.

Для задания относительных моментов можно использовать интервальные выражения вместо секунд, таких как те, которые используются в качестве значений параметров `OnActiveSec` и `OnBootSec`. Например, можно использовать все указанные ниже формы:

```
OnBootSec=2h 1m
OnStartupSec=1week 2days 3hours
OnActiveSec=1hr20m30sec10msec
```

Пробелы во временны*х* выражениях являются необязательными. Минимальная точность — наносекунды, но если ваш таймер срабатывает слишком часто (более одного раза каждые две секунды), система временно отключает его.

В дополнение к запуску с периодическими интервалами таймеры могут быть запланированы для активации в определенное время с помощью параметра `OnCalendar`. Эта функция предлагает самое близкое соответствие синтаксису традиционного задания **cron**, но его синтаксис более выразителен и гибкий. В табл. 4.7 приведены некоторые примеры спецификаций времени, которые могут использоваться как значение параметра `OnCalendar`.

Во временных выражениях звездочки являются заполнителями, которые соответствуют любому допустимому значению. Как и в файлах **crontab**, косая черта означает приращение значения. Точный синтаксис немного отличается от используемого в файлах **crontab**: там увеличиваемый объект должен быть диапазоном (например, `9-17/2`, "каждые два часа между 9:00 и 17:00"), а временны*е* выражения таймера **systemd** принимают только начальное значение (например, `9/2`, "каждые два часа, начиная с 9:00 утра").

Таблица 4.7. Примеры кодирования времени и даты в таймере systemd

Спецификация времени	Значение
`2017-07-04`	4 июля 2017 года в 00:00:00 (полночь)
`Fri-Mon *-7-4`	4 июля каждый год, но только если он попадает в интервал с пятницы по понедельник
`Mon-Wed *-*-* 12:00:00`	Понедельник, вторник и среда в полдень
`Mon 17:00:00`	Понедельник в 5:00.
`weekly`	По понедельникам в 00:00:00 (полночь)
`monthly`	В первый день месяца в 00:00:00 (полночь)
`*:0/10`	Каждые 10 минут, начиная с 0-й минуты
`*-*-* 11/12:10:0`	В 11:10 и 23:10 каждый день

Временные таймеры

С помощью команды **systemd-run** можно запланировать выполнение команды в соответствии с любым из стандартных типов таймера **systemd**, но не создавая файлы таймера и службы, предназначенные для конкретной задачи. Например, можно извлекать файлы из репозитория Git каждые десять минут:

```
$ systemd-run --on-calendar '*:0/10' /bin/sh -c "cd /app && git pull"
Running timer as unit run-8823.timer.
Will run service as unit run-8823.service.
```

Таймер **systemd** возвращает идентификатор временного модуля, который можно вывести с помощью команды **systemctl**. (Еще раз, напомним, что мы отредактировали вывод, приведенный ниже.)

```
$ systemctl list-timers run-8823.timer
NEXT     Sat 2017-06-17 20:40:07 UTC
LEFT     9min left
LAST     Sat 2017-06-17 20:30:07 UTC
PASSED   18s ago

$ systemctl list-units run-8823.timer
UNIT         run-8823.timer
LOAD         loaded
ACTIVE       active
SUB          waiting
DESCRIPTION /bin/sh -c "cd /app && git pull"
```

Чтобы отменить и удалить временный таймер, просто остановите его, выполнив команду **stop systemct**:

```
$ sudo systemctl stop run-8823.timer
```

Команда **systemd-run** создает таймер и модульные файлы в подкаталогах **/run/systemd/system**. Однако временные таймеры после перезагрузки не сохраняются. Чтобы сделать их постоянными, нужно найти их в каталоге **/run**, настроить их по мере необходимости и инсталлировать в каталоге **/etc/systemd/system**. Обязательно остановите временный таймер перед запуском или включением постоянной версии.

Общее использование запланированных задач

В этом разделе мы рассмотрим несколько общих задач, которые часто автоматизируются с помощью демона **cron** или таймера **systemd**.

Отправка почты

Следующая запись `crontab` реализует простую электронную почту. Вы можете использовать такую запись, чтобы автоматически отправлять результаты ежедневного отчета или результаты выполнения команды. (Строки были обрезаны, чтобы поместиться на странице. На самом деле это одна длинная строка.)

```
30 4 25 * * /usr/bin/mail -s "Time to do the TPS reports"
   ben@admin.com%TPS reports are due at the end of the month! Get
   busy!%%Sincerely,%cron%
```

Обратите внимание на использование символа `%` как для выделения команды из входного текста, так и для отметки окончаний строки внутри ввода. Это сообщение отправляется по электронной почте в 4:30 утра 25 числа каждого месяца.

Очистка файловой системы

Когда программа аварийно завершается, ядро может создать файл дампа памяти (обычно называемый **core**.*pid*, **core** или *программа*.**core**), содержащий снимок адресного пространства программы. Файлы дампа памяти полезны для разработчиков, но для системных администраторов они обычно являются пустой тратой места на диске.

Пользователи часто не знают о файлах дампа, поэтому они не склонны отключать их создание или удалять их самостоятельно. Вы можете использовать задание **cron** для очистки этих файлов дампа или других остатков, оставшихся после неправильных или сбойных процессов.

Ротация файла журнала

Системы различаются по качеству управления файлами журнала, и вам, вероятно, придется настроить параметры по умолчанию, чтобы они соответствовали вашим локальным правилам. Ротация файла журнала означает разделение его на сегменты по размеру или по дате с сохранением нескольких старых версий журнала. Поскольку ротация журнала является повторяющимся и регулярно происходящим событием, это идеальная задача, которую нужно планировать (подробнее см. раздел 10.5).

Выполнение пакетных заданий

Некоторые длительные вычисления лучше всего запускать в виде пакетных заданий. Например, сообщения могут накапливаться в очереди или базе данных. Вы можете использовать задание **cron** для одновременного извлечения, преобразования и загрузки всех сообщений в очереди в другое место, например хранилище данных.

Некоторые базы данных извлекают выгоду из текущего обслуживания. Например, распределенная база данных с открытым исходным кодом Cassandra имеет функцию восстановления, которая синхронизирует узлы в кластере. Эти задачи обслуживания являются хорошими кандидатами для выполнения с помощью демона **cron** или таймера **systemd**.

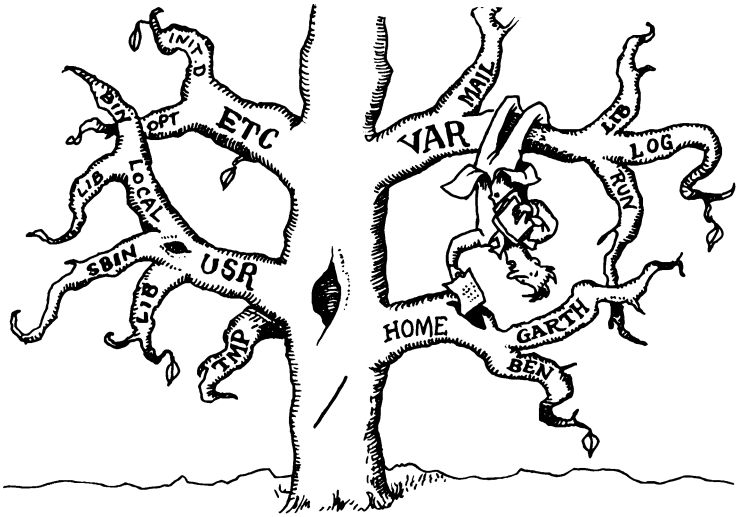
Резервное копирование и зеркалирование

Вы можете использовать запланированную задачу для автоматического резервного копирования каталога в удаленную систему. Мы рекомендуем запускать полное резервное копирование один раз в неделю, накапливая инкрементные различия каждую ночь. Запускайте резервное копирование поздно ночью, когда нагрузка на систему, вероятно, будет низкой.

Зеркала — это побайтовые копии файловых систем или каталогов, размещенных в другой системе. Они могут использоваться как форма резервного копирования или как способ сделать файлы доступными в нескольких местах. Веб-сайты и репозитории программного обеспечения часто зеркалируются, чтобы повысить отказоустойчивость системы и обеспечить более быстрый доступ для физически удаленных пользователей. Используйте периодическое выполнение команды **rsync** для поддержания зеркал в актуальном состоянии.

Глава 5

Файловая система



Ответьте, не раздумывая, на вопрос: что из перечисленного ниже можно считать элементами файловой системы?

- Процессы
- Аудиоустройства
- Структуры данных ядра и параметры настройки
- Каналы межзадачного взаимодействия

Если речь идет о системах UNIX или Linux, ответ будет таков: все перечисленное выше. Ну и, конечно же, в файловую систему входят собственно файлы [1].

Основным назначением файловой системы является упорядочение хранимых ресурсов системы. Однако программистам не хотелось каждый раз заново изобретать колесо при управлении объектами других типов. Очень часто оказывалось удобным представлять такие объекты в виде элементов файловой системы. Подобный унифицированный подход имеет как преимущества (единый программный интерфейс, легкий доступ из интерпретатора команд), так и недостатки (реализация файловых систем по методу доктора Франкенштейна), но независимо от того, нравится он вам или нет, именно такой подход применяется в системе UNIX (а значит, и в системе Linux).

Файловую систему можно представить состоящей из четырех основных компонентов:

- пространство имен — методы именования объектов и организации их в виде единой иерархии;

[1] Точнее говоря, эти элементы представлены внутри файловой системы. В большинстве случаев файловая система служит "местом встречи" клиентов с драйверами и серверами, которые им требуются.

- API[2] — набор системных вызовов для перемещения между объектами и управления ими;
- модель безопасности — схема защиты, сокрытия и совместного использования объектов;
- реализация — программный код, который связывает логические модели с дисковой подсистемой.

В современных ядрах систем определен абстрактный интерфейс, позволяющий работать с различными файловыми системами. Одни части файлового дерева обрабатываются традиционной дисковой подсистемой, другие управляются специальными драйверами ядра. Например, за работу файловых систем — как сетевых (Network File System — NFS), так и общих межсетевых (Common Internet File System — CIFS), — отвечает драйвер, который перенаправляет запросы серверу, расположенному на другом компьютере.

К сожалению, концептуальные границы нечетко очерчены, поэтому имеется слишком много "особых" случаев. К примеру, файлы устройств позволяют программам взаимодействовать с драйверами ядра. Они не являются файлами данных, но обрабатываются файловой системой, а их характеристики записываются на диск.

Еще одним усложняющим фактором является то, что ядро поддерживает несколько типов дисковых файловых систем. Доминирующими стандартами в данный момент являются системы ext4, XFS и UFS, а также ZFS и Btrfs компании Oracle. Однако существует и множество других файловых систем, таких как VxFS компании Veritas и JFS компании IBM.

Есть много реализаций файловых систем с другой семантикой, таких как FAT и NTFS (используются в системе Microsoft Windows) и 9660 (применяется на компакт-дисках).

Организация файловой системы — очень обширная тема, которую мы рассмотрим с нескольких точек зрения. В этой главе рассказывается, где обычно хранятся файлы, и описываются характеристики файлов, а также рассматриваются биты разрешения и ключевые команды, позволяющие задавать атрибуты файлов. Технические подробности, связанные с файловыми системами, например разбиение дисков, рассматриваются в главе 20.

Файловая система NFS, обеспечивающая совместное использование и удаленный доступ к файлам в системах UNIX и Linux описывается в главе 21. В главе 22 рассматривается аналогичная система, применяемая в системе Windows.

🕮 Дополнительную информацию о конкретных файловых системах см. в разделе 20.9.

При таком разнообразии реализаций файловых систем может показаться странным, что данная глава посвящена всего лишь одной файловой системе. Однако на отдельных реализациях можно специально не останавливаться, поскольку в большинстве современных файловых систем либо предприняты попытки обеспечить более эффективное и надежное применение традиционных функциональных возможностей, либо дополнительные функциональные возможности реализованы в качестве отдельного уровня, работающего поверх семантики стандартной файловой системы (некоторые файловые системы реализуют оба направления). Как бы то ни было, слишком много программных пакетов ориентировано на описанную в этой главе модель, чтобы ею можно пренебречь.

[2]Интерфейс прикладных программ (Application Programming Interface — API) — общий термин для описания набора подпрограмм, организованных в виде библиотеки или совокупности библиотек и доступных программистам.

5.1. Имена путей

Файловая система — это единая иерархическая структура, которая начинается с каталога / и разветвляется, охватывая произвольное число подкаталогов. Каталог самого верхнего уровня называется *корневым*. Эта моноиерархическая система отличается от используемой в системе Windows, где применяется понятие пространства имен, основанное на принципе деления диска на разделы.

В графических пользовательских интерфейсах каталоги часто называются *папками*, даже в системах Linux. Папки и каталоги — одно и то же. Папка — это термин систем Windows и macOS. Тем не менее стоит отметить, что слово "папка" часто вызывает раздражение у некоторых экспертов. Не используйте его в технических контекстах, если не готовы ловить на себе иронические взгляды.

Цепочка имен каталогов, через которые необходимо пройти для доступа к заданному файлу, вместе с именем этого файла образуют путь к файлу. Путь может быть абсолютным (например, `/tmp/foo`) или относительным (например, `book4/filesystem`). Последние интерпретируются начиная с текущего каталога. Возможно, многие считают, что текущий каталог задается интерпретатором команд. На самом деле текущий каталог есть у каждого процесса.

Термины *имя файла* и *путь* в той или иной степени являются взаимозаменяемыми (по крайней мере так они трактуются в данной книге). Соответственно, имена файлов и пути бывают абсолютными и относительными.

Файловое дерево может иметь произвольную глубину, однако каждый компонент имени файла должен состоять не более чем из 255 символов. Существует также ограничение на длину пути, который вы можете передавать ядру в качестве аргумента системного вызова (4095 байт в Linux и 1024 байт в BSD). Для того чтобы получить доступ к файлу, полное имя которого превышает эти ограничения, необходимо с помощью команды `cd` перейти в промежуточный каталог, а затем воспользоваться относительным путем к файлу.

5.2. Монтирование и демонтирование файловой системы

Файловое дерево формируется из отдельных частей, называемых *файловыми системами*, каждая из которых содержит корневой каталог и список его подкаталогов и файлов. Как правило, значение термина становится ясным из контекста, но для ясности мы будем использовать термин "файловое дерево" для обозначения всей схемы, а файловой системой будем называть ветви, присоединенные к этому дереву.

Некоторые файловые системы представляют собой разделы диска или логические тома с памятью на дисках, но, как уже было сказано, файловая система может принимать облик всего, что подчиняется определенному интерфейсу прикладного программирования: сетевых файловых систем, компонентов ядра, эмуляторов резидентных дисков и т.д. В большинстве ядер есть даже оригинальная файловая система, позволяющая монтировать отдельные файлы, как если бы они были физическими устройствами, и создавать образы файловых систем без необходимости в перераспределении дисковой памяти. Системы Linux даже отдельные части файлового дерева интерпретируют как файловые системы. Этот прием позволяет дублировать, перемещать и скрывать фрагменты файлового дерева.

В большинстве случаев файловые системы присоединяются к файловому дереву с помощью команды **mount**[3]. Эта команда связывает каталог существующего файлового дерева, называемый *точкой монтирования*, с корневым каталогом новой файловой системы. На время монтирования доступ к прежнему содержимому точки монтирования становится невозможным. Впрочем, в большинстве случаев точка монтирования — это пустой каталог.

Например, команда

```
$ sudo mount /dev/sda4 /users
```

подключает к точке монтирования, заданной в виде пути **/users**, файловую систему, расположенную в дисковом разделе (устройстве) **/dev/sda4**. Просмотреть содержимое файловой системы по окончании монтирования можно с помощью команды **ls /users**.

В некоторых системах программа **mount** — это просто оболочка, вызывающая команды, специфичные для файловой системы, такие как **mount.ntfs** или **mount_smbfs**. При необходимости эти вспомогательные команды можно выполнять напрямую; иногда они предлагают дополнительные параметры, которые оболочка **mount** не понимает. С другой стороны, универсальной команды **mount** вполне достаточно для повседневного использования.

Чтобы увидеть все файловые системы, смонтированные к данному моменту, команду **mount** можно выполнить без каких-либо аргументов. В системах Linux может быть 30 или более файловых систем, большинство из которых представляют собой различные интерфейсы ядра.

Файловые системы, которые обычно монтируются в системе, перечислены в файле **/etc/fstab**. Информация в этом файле позволяет автоматически проверять (с помощью команды **fsck**) и монтировать (с помощью команды **mount**) файловые системы во время загрузки с указанными параметрами. Файл **fstab** также служит документацией для компоновки файловых систем на диске и включает короткие команды, такие как **mount /usr**. Информацию о команде **fstab** см. в разделе 20.10.

Файловые системы демонтируются командой **umount**. "Занятую" файловую систему демонтировать невозможно. В ней не должно быть ни открытых файлов, ни выполняющихся процессов с их текущими каталогами. Если демонтируемая файловая система содержит исполняемые программы, они не должны быть запущены.



В системе Linux предусмотрена возможность "ленивого" демонтирования (с помощью команды **umount -l**), которая удаляет файловую систему из иерархии имен, но в действительности не демонтирует ее до тех пор, пока все существующие файловые ссылки не будут закрыты. Полезность этой опции достаточно спорна. Во-первых, нет никакой гарантии, что существующие ссылки когда-либо будут закрыты сами по себе, кроме того, "наполовину демонтированное" состояние файловой системы может нарушать семантическую целостность для использующих ее программ. Они могут выполнять операции чтения и записи с существующими дескрипторами файлов, но не могут открывать новые файлы или выполнять какие-либо другие операции файловой системы.

Команда **umount -f** предназначена для принудительного демонтирования занятой файловой системы и поддерживается во всех рассмотренных нами системах. Однако не

[3]Мы говорим "в большинстве случаев", поскольку в файловой системе ZFS принят несколько другой подход к монтированию и демонтированию, не говоря уже о других аспектах администрирования файловых систем. Подробнее см. раздел 20.12.

стоит применять ее в системах, не использующих NFS. Кроме того, она может не применяться к определенным типам файловых систем (например, к таким, как системы XFS или ext4, которые ведут системные журналы).

Если файловая система, которую вы пытаетесь демонтировать, занята, вместо использования команды **umount -f** выполните команду **fuser**, чтобы узнать, какие процессы работают с файловой системой. Команда **fuser -c** *точка_монтирования* выводит идентификаторы всех процессов, обращающихся к файлам или каталогам указанной файловой системы, а также ряд буквенных кодов, которые отображают природу этой активности. Например,

```
freebsd$ fuser -c /usr/home
/usr/home: 15897c  87787c  67124x  11201x  11199x  11198x  972x
```

Буквенные коды зависят от конкретной системы. В данном примере используется система FreeBSD. В этой системе буква c означает, что процесс имеет свой рабочий каталог в файловой системе, а буква x означает выполняемую программу. Однако детали обычно не имеют большого значения; главное для нас — идентификаторы процессов (PID).

Для того чтобы в точности определить, что собой представляют эти процессы, выполните команду **ps**, передав ей список идентификаторов, о которых сообщила команда **fuser**.

```
nutrient:~$ ps up "87787 11201"
USER  PID  %CPU  %MEM  STARTED     TIME  COMMAND
fnd 11201   0.0   0.2  14Jul16  2:32.49  ruby: slave_audiochannelbackend
fnd 87787   0.0   0.0  Thu07PM  0:00.93  -bash (bash)
```

Список идентификаторов взят в кавычки, чтобы интерпретатор передал его команде **ps** как один аргумент.



В системах Linux можно избежать необходимости задания идентификаторов (PID) в команде **ps**, выполнив команду **fuser** с флагом **-v**. Этот вариант команды генерирует более читабельный результат, который включает имя команды.

```
$ fuser -cv /usr
                USER  PID  ACCESS  COMMAND
     /usr root  444  ....m   atd
                root  499  ....m  sshd
                root  520  ....m   lpd
...
```

Буквенные коды в столбце ACCESS такие же, как в результате выполнения команды **fuser -c**.

Более удачной альтернативой команде **fuser** является утилита **lsof**. Утилита **lsof** — более сложная программа, чем **fuser**, и, соответственно, результаты ее работы более содержательны. Она по умолчанию инсталлируется во всех наших иллюстративных системах Linux, а в системе FreeBSD доступна в виде пакета.



В системе Linux сценарии, предназначенные для поиска конкретной информации об использовании файловых систем процессами, располагают возможностью непосредственного чтения файлов, хранящихся в каталоге **/proc**. Однако команда **lsof -F**, форматирующая вывод команды **lsof** для облегчения синтаксического анализа, — более простое и переносимое решение. Для запроса только той информации, которая действительно необходима, следует использовать дополнительные флаги командной строки.

5.3. Структура файлового дерева

Файловые системы в системе UNIX никогда не были хорошо организованы. В ней одновременно используется много разных, несовместимых соглашений об именовании файлов. Во многих случаях файлы группируются по выполняемым функциям, независимо от того, как часто они изменяются. Это затрудняет обновление операционной системы. Например, каталог **/etc** содержит ряд файлов, которые никогда не меняются, а также локальные файлы. Как определить, какие файлы нельзя трогать при обновлении системы? Это нужно просто знать... или верить, что программа инсталляции примет правильные решения.

Системные администраторы, стремящиеся к логической стройности, иногда поддаются соблазну улучшить структуру файлового дерева, принятую по умолчанию. К сожалению, файловое дерево имеет много скрытых зависимостей, поэтому попытки изменить его структуру созда.т новые проблемы. Просто предоставьте операционной системе самой решать, где разместить системные пакеты. Если же вам предлагают изменить местоположение пакетов, всегда соглашайтесь на вариант, принятый по умолчанию, иначе у вас возникнут неприятности.

🕮 Дополнительную информацию о настройке конфигурации ядер см. в главе 11.

Корневая файловая система содержит корневой каталог и минимальный набор файлов и подкаталогов. Файл ядра находится в недрах корневой файловой системы, но не имеет стандартного имени или точного местоположения; в системе BSD это даже не один файл, а, скорее, набор компонентов.

Частью корневой файловой системы являются также каталог **/etc** для критических системных файлов и файлов конфигурации, каталоги **/sbin** и **/bin** — для важных утилит и иногда каталог **/tmp** — для временных файлов. В прошлом каталог **/dev** представлял собой реальный каталог, включенный в корневую файловую систему, но в сейчас он является виртуальной файловой системой, которая монтируется отдельно. (Подробнее см. в разделе 11.3.)

Одни системы хранят совместно используемые библиотечные файлы и прочие важные программы (например, препроцессор языка C) в каталоге **/lib** или **/lib64**. Другие переместили эти элементы в каталог **/usr/lib**, оставив для каталога **/lib** роль символической ссылки.

Огромное значение имеют также каталоги **/usr** и **/var**. В первом хранится большинство стандартных программ и другие полезные компоненты, в частности интерактивная документация и библиотеки. Совсем не обязательно, чтобы каталог **/usr** был отдельной файловой системой, однако для удобства администрирования его, как правило, монтируют именно так. Для того чтобы система могла загружаться в многопользовательском режиме, необходимы оба каталога — **/usr** и **/var**.

В прошлом было принято разбивать системный диск на разделы и размещать в них отдельные части файлового дерева (в основном в каталогах **/usr**, **/var** и **/tmp**). Эта практика не исчезла полностью и в настоящее время, но основной тенденцией стало использование одной большой корневой файловой системы. Большие жесткие диски и возрастающая сложность реализаций файловых систем снизили ценность разбиения дисков на разделы.

🕮 Побудительные причины для разбиения дисков и его основные правила см. в разделе 20.6.

Чаще всего разбиение диска используют для того, чтобы одна часть файлового дерева не занимала все доступное пространство и не вызывала сбой системы. Соответственно, чаще всего разбиению подвергаются каталог `/var` (содержащий файлы журнала, размеры которых со временем растут и могут вызвать проблемы), `/tmp` и рабочие каталоги пользователей. Специальные файловые системы могут хранить большие объемы информации, например библиотеки исходных кодов и базы данных.

Наиболее важные стандартные каталоги перечислены в табл. 5.1.

Таблица 5.1. Стандартные каталоги и их содержимое

Каталог	Содержимое
`/bin`	Команды операционной системы ядра
`/boot`	Ядро и файлы для его загрузки
`/compat`	Файлы и каталоги в системе FreeBSD, обеспечивающие бинарную совместимость с системой Linux
`/dev`	Файлы устройств: дисков, принтеров, псевдотерминалов и т.д.
`/etc`	Важные файлы запуска и конфигурации системы
`/home`	Стандартные домашние каталоги пользователей
`/lib`	Библиотеки, совместно используемые библиотеки и команды, применяемые в каталогах `/bin` и `/sbin`
`/media`	Точки монтирования файловых систем на съемных носителях
`/mnt`	Временные точки монтирования
`/opt`	Программные пакеты необязательных приложений (которые пока не находят широкого применения)
`/proc`	Информация о всех выполняющихся процессах
`/root`	Домашний каталог суперпользователя (часто просто `/`)
`/run`	Точки встречи для выполняемых программ (идентификаторы PID, сокеты и т.д.).
`/sbin`	Команды, необходимые для обеспечения минимальной работоспособности системы[3]
`/srv`	Поля, зарезервированные для распределения через веб и другие серверы
`/sys`	Интерфейсы разных ядер (Linux)
`/tmp`	Временные файлы, которые могут удаляться при перезагрузке
`/usr`	Иерархия дополнительных файлов и программ
`/usr/bin`	Большинство команд и исполняемых файлов
`/usr/include`	Файлы заголовков, предназначенные для компиляции C-программ
`/usr/lib`	Библиотеки и вспомогательные файлы для стандартных программ
`/usr/local`	Локальные программы (программы, создаваемые или устанавливаемые локальным пользователем); отражает структуру каталога `/usr`
`/usr/sbin`	Менее важные команды системного администрирования
`/usr/share`	Элементы, общие для различных систем
`/usr/share/man`	Страницы интерактивной документации
`/usr/src`	Исходные коды нелокальных программных пакетов (не находит широкого применения)
`/usr/tmp`	Дополнительный каталог для временных файлов, которые могут сохраняться при перезагрузке

Окончание табл. 5.1

Каталог	Содержимое
`/var`	Системные данные и конфигурационные файлы
`/var/adm`	Разное: журнальные файлы, записи об инсталляции системы, административные компоненты
`/var/log`	Системные журнальные файлы
`/var/run`	Те же функции. что и в каталоге `/run`, а также символические ссылки
`/var/spool`	Буферные каталоги для принтеров, электронной почты и т.д.
`/var/tmp`	Каталог для временного хранения файлов (после перезагрузки файлы не исчезают)

[a] В прошлом отличительная особенность команд в каталоге `/sbin` обычно состояла в том, что они связаны со статическими версиями системных библиотек и, следовательно, не имеют многочисленных зависимостей от других частей системы. В настоящее время все бинарные модули связаны динамически и поэтому между каталогами `/bin` и `/sbin` реальной разницы нет.

Во многих системах man-страница **`hier`** содержит общие рекомендации по формированию файловой системы. Однако не стоит ожидать, что реальная система во всех отношениях согласует свои действия с "генеральным планом".



Стандарт Filesystem Hierarchy Standard пытается кодифицировать, рационализировать и объяснить стандартные каталоги для систем семейства Linux.[4] Это отличный ресурс для консультаций, если вы столкнулись с необычной ситуацией и должны выяснить, куда сохранить какой-нибудь объект.

Несмотря на свой статус "стандарта", это скорее отражение реальной практики, чем регламентирующий документ. К тому же в последнее время он не часто обновляется, поэтому не описывает точное расположение файловой системы, обнаруженное в современных дистрибутивах

5.4. Типы файлов

В большинстве реализаций файловых систем определены семь типов файлов. Даже если разработчики добавляют в файловую систему что-нибудь новое и необычное (например, информацию о процессах в каталог **`/proc`**), им приходится маскировать это под файлы стандартных типов.

- обычные файлы;
- каталоги;
- файлы символьных устройств;
- файлы блочных устройств;
- локальные сокеты;
- именованные каналы (FIFO);
- символические ссылки.

Определить тип существующего файла можно с помощью команды **`file`**. Эта команда не только определяет стандартные типы файлов, но и распознает их форматы.

```
$ file /usr/include
/usr/include: directory
```

[4] См. `wiki.linuxfoundation.org/en/FHS`.

```
$ file /bin/sh
/bin/sh: ELF 64-bit LSB executable, x86-64, version 1 (FreeBSD),
   dynamically linked, interpreter /libexec/ld-elf.so.1, for FreeBSD 11.0
   (1100122), FreeBSD-style, stripped
```

Многословная информация о файле **/bin/sh** означает всего лишь "это исполняемая команда."

Другая возможность исследования файлов предоставляется командой **ls -ld**. Флаг **-l** подразумевает вывод подробной информации, а флаг **-d** заставляет команду **ls** выводить информацию о каталоге, а не его содержимое.

Первый символ в строке вывода обозначает тип объекта. В следующем примере выдается информация о каталоге **/usr/include**.

```
$ ls -ld /usr/include
drwxr-xr-x   27 root   root      4096 Jul 15 20:57 /usr/include
```

Возможные коды для представления различных типов файлов перечислены в табл. 5.2.

Таблица 5.2. Кодирование типов файлов в листинге команды ls

Тип файла	Символ	Создается командой	Удаляется командой
Обычный файл	-	Редакторы, **cp** и др.	**rm**
Каталог	d	**mkdir**	**rmdir, rm -r**
Файл символьного устройства	c	**mknod**	**rm**
Файл блочного устройства	b	**mknod**	**rm**
Локальный сокет	s	**socket(2)**	**rm**
Именованный канал	p	**mknod**	**rm**
Символическая ссылка	l	**ln -s**	**rm**

Как видно из табл. 5.2, команда **rm** является универсальным средством удаления файлов. Но как удалить файл, имя которого, скажем, **-f**? В большинстве файловых систем это абсолютно корректное имя, но команда **rm -f** не сделает то, что нужно, поскольку выражение **-f** будет воспринято как флаг команды. Выйти из положения можно, либо указав более полное имя (например, **./-f**), либо воспользовавшись специальным аргументом **--**, который сообщит команде **rm** о том, что остальная часть командной строки представляет собой имя файла, а не список аргументов: **rm -- -f**.

Похожая проблема возникает с файлами, в именах которых присутствуют управляющие символы, поскольку такие имена трудно или вообще невозможно воспроизвести с помощью клавиатуры. В подобной ситуации нужно воспользоваться метасимволами интерпретатора команд, чтобы задавать не имя целиком, а лишь его шаблон. Указывайте также опцию **-i**, чтобы команда **rm** требовала подтвердить удаление каждого файла. Это позволит не удалить нужные файлы, соответствующие шаблону. Вот как, к примеру, можно удалить файл с оригинальным именем **foo**<Ctrl+D>**bar**.

```
$ ls
foo?bar foose kde-root

$ rm -i foo*
rm: remove 'foo\004bar'? y
rm: remove 'foose'? n
```

Команда **ls** отображает вместо управляющего символа знак вопроса, что иногда сбивает с толку. Если забыть, что символ "?" тоже является подстановочным знаком интерпретатора, и попытаться выполнить команду **rm foo?bar**, можно удалить более одного файла (правда, не в данном примере). Ценность флага **-i** очень велика!

Команда **ls -b** выводит управляющие символы в виде восьмеричных чисел. Это свойство может оказаться полезным, если необходимо специально идентифицировать эти символы. Например, <Ctrl+A> — это 1 (\001 в восьмеричной системе счисления), <Ctrl+B> — это 2 и т.д. в алфавитном порядке. Страница **man ascii** и раздел Википедии о системе ASCII содержат прекрасную таблицу управляющих символов и их восьмеричных эквивалентов.

Если имена файлов совсем уж сложно набирать, прибегните к команде **rm -i ***. Еще одна возможность удаления файлов с причудливыми именами — использование другого интерфейса файловой системы, такого как режим представления каталогов интерпретатора **emacs** или визуального средства, подобного Nautilus.

Обычные файлы

Обычный файл — это просто последовательность байтов. Файловые системы не налагают ограничения на его структуру. Текстовые документы, файлы данных, программные файлы, библиотеки функций и многое другое — все это хранится в обычных файлах. К их содержимому возможен как последовательный, так и прямой доступ.

Каталоги

Каталог хранит именованные ссылки на другие файлы. Он создается командой **mkdir** и удаляется (при условии, что он пуст) командой **rmdir**. Непустые каталоги можно удалить командой **rm -r**.

Специальные ссылки "." и ".." обозначают сам каталог и его родительский каталог соответственно. Такие ссылки нельзя удалить. Поскольку корневой каталог находится на вершине иерархии, ссылка "/.." в нем эквивалентна ссылке "/." (и обе эти ссылки эквивалентны "/").

Жесткая ссылка

Имя файла в действительности хранится в родительском каталоге, а не в самом файле. На файл можно ссылаться из нескольких каталогов одновременно и даже из нескольких элементов одного и того же каталога, причем у всех ссылок могут быть разные имена. Это создает иллюзию того, что файл одновременно присутствует в разных каталогах.

Эти дополнительные "жесткие" (фиксированные) ссылки (которые следует отличать от символических, или "мягких") можно считать синонимами для исходных файлов, и с точки зрения файловой системы все ссылки на файл эквивалентны. Файловая система подсчитывает количество ссылок на каждый файл и при удалении файла не освобождает блоки данных до тех пор, пока не будет удалена последняя ссылка на него. Ссылки не могут указывать на файл, находящийся в другой файловой системе.

Жесткие ссылки создаются командой **ln** и удаляются командой **rm**. Синтаксис команды **ln** легко запомнить, поскольку она является "зеркальным отражением" команды **cp**. Команда **cp oldfile newfile** создает копию файла **oldfile** с именем **newfile**, а команда **ln oldfile newfile** преобразует имя **newfile** в дополнительную ссылку на файл **oldfile**.

В большинстве файловых систем жесткие ссылки можно создавать не только на файлы, но и на каталоги, но это делается довольно редко. Например, ссылки на каталоги часто приводят к вырожденным условиям, таким как зацикливание и неоднозначность. В большинстве случаев лучше использовать символические ссылки (см. ниже).

Для того чтобы узнать, сколько ссылок существует на данный файл, используйте команду `ls -l`. (См. ниже пример использования команды `ls`.) Дополнительные подробности можно найти в разделе 5.3. Кроме того, рекомендуем обратить внимание на комментарии по поводу команды `ls -i` в разделе 5.3, которые помогут идентифицировать жесткие ссылки.

Важно понимать, что жесткие ссылки не являются особым типом файлов, просто файловая система позволяет создавать несколько ссылок на один файл. Не только содержимое, но и атрибуты файла, в частности права доступа и идентификатор владельца, являются общими для всех ссылок.

Файлы символьных и блочных устройств

🕮 Устройства и драйверы более подробно рассматриваются в главе 11.

Файлы устройств позволяют программам получать доступ к аппаратным средствам и периферийному оборудованию системы. Ядро включает (или загружает) специальные программы (драйверы), которые во всех деталях "знают", как взаимодействовать с каждым из имеющихся устройств, поэтому само ядро может оставаться относительно абстрактным и независимым от оборудования.

Драйверы устройств образуют стандартный коммуникационный интерфейс, который воспринимается пользователем как совокупность обычных файлов. Получив запрос к файлу символьного или блочного устройства, файловая система передает этот запрос соответствующему драйверу. Важно отличать *файлы* устройств от *драйверов* этих устройств. Файлы сами по себе не являются драйверами. Их можно рассматривать как шлюзы, через которые драйвер принимает запросы.

Разница между символьными и блочными устройствами слишком тонкая и не стоит внимания. В прошлом некоторые типы аппаратных средств могли быть представлены файлами любого типа, но в современных системах такая конфигурация встречается редко. На практике, в системе FreeBSD блочных устройств вообще нет, хотя упоминания о них на справочных страницах и в заголовочных файлах еще встречаются.

Файлы устройств характеризуются двумя номерами: старшим и младшим. Старший номер устройства позволяет ядру определить, к какому драйверу относится файл, а младший номер, как правило, идентифицирует конкретное физическое устройство. Например, старший номер устройства 4 в системе Linux соответствует драйверу последовательного порта. Таким образом, первый последовательный порт (`/dev/tty0`) будет иметь старший номер 4 и младший номер 0.

Драйверы могут интерпретировать переданные им младшие номера устройств как угодно. Например, драйверы накопителей на магнитных лентах с помощью этого номера определяют, необходимо ли перемотать ленту после закрытия файла устройства.

В далеком прошлом `/dev` играл роль общего каталога, а файлы устройств, которые в нем хранились, создавались с помощью команды `mknod` и удалялись командой `rm`. К сожалению, эта "сырая" система плохо справлялась с безбрежным морем драйверов и типов устройств, которые появились в последние десятилетия. Кроме того, она способствовала возникновению разного рода потенциальных конфигурационных нестыковок: например, файлы устройств ссылались на несуществующие устройства, устройства оказывались недоступными, поскольку они не имели файлов устройств, и т.д.

В наши дни в большинстве систем реализована некоторая форма автоматического управления файлами устройств, которая позволяет системе играть более активную роль в настройке конфигурации собственных файлов устройств. Подробнее о методах решения этой задачи в разных системах написано в главе 11.

Локальные сокеты

Установленные посредством сокетов соединения позволяют процессам взаимодействовать, не подвергаясь влиянию других процессов. В системе UNIX поддерживается несколько видов сокетов, использование которых, как правило, предполагает наличие сети.

🕮 Дополнительную информацию о системе Syslog см. в главе 10.

Локальные сокеты доступны только на локальном компьютере, и обращение к ним осуществляется через специальные объекты файловой системы, а не через сетевые порты. Иногда такие сокеты называют UNIX-сокетами. В качестве примеров стандартных средств, использующих локальные сокеты, можно назвать системы X Window и Syslog.

Локальные сокеты создаются с помощью системного вызова **socket**. Когда с обеих сторон соединение закрыто, сокет можно удалить командой **rm** или с помощью системного вызова **unlink**.

Именованные каналы

Подобно локальным сокетам, именованные каналы обеспечивают взаимодействие двух процессов, выполняемых на одном компьютере. Такие каналы еще называют файлами FIFO (First In, First Out — "первым поступил, первым обслужен"). Они создаются командой **mknod** и удаляются командой **rm**.

Символические ссылки

Символическая, или "мягкая", ссылка позволяет вместо имени файла указывать его псевдоним. Столкнувшись при поиске файла с символической ссылкой, ядро извлекает хранящееся в ней имя. Разница между жесткими и символическими ссылками состоит в том, что жесткая ссылка является прямой, т.е. указывает непосредственно на индексный дескриптор файла, тогда как символическая ссылка указывает на файл по имени. Файл, адресуемый символической ссылкой, и сама ссылка представляют собой разные объекты файловой системы.

Символические ссылки создаются командой **ln -s** и удаляются командой **rm**. Они могут содержать произвольное имя, т.е. разрешается указывать на файлы, хранящиеся в других файловых системах, и даже на несуществующие файлы. Иногда несколько символических ссылок образуют петлю.

Символическая ссылка может хранить как полное, так и сокращенное имя файла.

Например, команда

```
$ sudo ln -s archived/secure /var/data/secure
```

посредством относительного пути связывает имя **/var/data/secure** с именем **/var/data/archived/secure**. Как видно из следующего результата выполнения команды **ls**, она создает символическую ссылку **/var/data/secure** с целью **archived/secure**.

```
$ ls -l /var/data/secure
lrwxrwxrwx 1 root root 18 2009-07-05 12:54 /var/data/secure -> archived/secure
```

Каталог **/var/data** можно переместить куда угодно, но символическая ссылка останется корректной.

Разрешения файла, которые команда **ls** показывает для символической ссылки, `lrwxrwxrwx`, являются фиктивными значениями. Разрешение на создание, удаление или последующую ссылку контролируется содержащим каталогом, тогда как права на чтение, запись и выполнение в целевой ссылке предоставляются собственными разрешениями. По этой причине символические ссылки не содержат никакой информации о разрешении.

Часто ошибочно думают, будто первый аргумент команды **ln -s** должен преобразовываться с учетом текущего каталога. На самом деле он не раскрывается командой **ln** как имя файла, а просто записывается в виде литеральной строки, которая становится объектом символической ссылки.

5.5. Атрибуты файлов

В традиционной модели файловой системы UNIX и Linux каждому файлу соответствует набор из девяти битов режима. Они определяют, какие пользователи имеют право читать файл, записывать в него данные или запускать его на выполнение. Вместе с другими тремя битами, которые в основном влияют на работу исполняемых файлов, этот набор образует код, или режим, доступа к файлу.

Двенадцать битов режима хранятся вместе с четырьмя дополнительными битами, определяющими тип файла. Эти четыре бита устанавливаются при создании файла и не подлежат изменению. Биты режима могут изменяться владельцем файла или суперпользователем с помощью команды **chmod** ("change mode" — изменить режим). Просмотр значений этих битов осуществляется с помощью команды **ls -l** (или **ls -ld** в случае каталога). Пример приведен далее в этой главе.

Биты режима

Девять битов режима определяют, кто и какие операции может выполнять над файлом. Традиционная система UNIX не позволяет устанавливать режим на уровне отдельного пользователя (хотя теперь списки управления доступом поддерживаются во всех основных системах). Подробнее это описано в разделе 5.6. Вместо этого три различных набора битов определяют права доступа, предоставляемые владельцу файла, членам группы, которой принадлежит файл, и прочим пользователям (именно в таком порядке)[5]. Каждый набор состоит из трех битов: бита чтения, записи и выполнения (в том же порядке).

Код доступа удобно записывать в виде восьмеричного числа, так как каждая цифра в нем представляется тремя битами. Три старших бита (в коде доступа им соответствуют восьмеричные значения 400, 200 и 100) служат для управления доступом к файлу его владельца. Вторые три бита (40, 20 и 10) задают доступ для членов группы. Последние три бита (4, 2 и 1) определяют доступ к файлу остальных пользователей. Старший бит каждой триады — это бит чтения, средний — бит записи, младший — бит выполнения.

Если пользователь попадает только в одну из двух-трех категорий прав доступа, ему предоставляются только самые ограниченные права. Например, права владельца файла

[5]Если считать владельца "пользователем", а все остальные "другими", то можно запомнить порядок наборов разрешений, используя имя Hugo. Буквы **u**, **g** и **o** также являются буквенными кодами, используемыми мнемонической версией команды **chmod**.

всегда определяются триадой битов владельца и никогда — битами группы. Возможна ситуация, когда другие пользователи имеют больше прав, чем владелец, но такая конфигурация используется редко.

Для обычного файла бит чтения означает возможность открывать и читать файл. Бит записи позволяет изменять содержимое файла, в том числе полностью стирать его. Правом удаления и переименования файла управляют биты, заданные для его родительского каталога, поскольку именно там хранится имя файла.

Установкой бита выполнения задается разрешение запускать файл. Существует два типа исполняемых файлов: бинарные файлы, которые выполняются непосредственно центральным процессором, и сценарии, обрабатываемые интерпретатором команд или какой-нибудь другой аналогичной программой. В соответствии с принятыми соглашениями, сценарии начинаются со строки примерно такого вида.

```
#!/usr/bin/perl
```

Здесь задается соответствующий интерпретатор. Текстовые исполняемые файлы, в которых не указан конкретный интерпретатор, считаются сценариями **bash** или **sh**[6].

Бит выполнения для каталога (в этом контексте его часто называют *битом поиска или сканирования*) разрешает переходить в каталог или проходить через него при доступе к файлу, но без получения списка содержимого каталога. Получение списка становится возможным, если для каталога задана комбинация битов чтения и выполнения. Комбинация битов записи и выполнения позволяет создавать, удалять и переименовывать файлы в данном каталоге.

Традиционную девятибитовую модель режимов способны усложнить и переопределить такие расширения, как списки управления доступом (см. раздел 5.6), система SELinux (раздел 3.4), т.е. Linux с улучшенной безопасностью (см. раздел 22.9), и "бонусные" биты режимов, определяемые отдельными файловыми системами (см. последний подраздел данного раздела). Если окажется, что вы не можете объяснить поведение своей файловой системы, проверьте, оказывает ли на нее влияние один из перечисленных выше факторов.

Биты setuid и setgid

Биты, которым в коде доступа соответствуют восьмеричные значения `4000` и `2000`, — это биты смены идентификатора пользователя (`setuid`) и идентификатора группы (`setgid`). Если эти биты установлены для исполняемых файлов, они позволяют программам получать доступ к файлам и процессам, которые при прочих обстоятельствах недоступны пользователю, выполняющему эти программы. Подробнее этот механизм был описан в начале главы 3.

Если бит `setgid` установлен для каталога, то создаваемые в нем файлы будут принимать идентификатор группы каталога, а не группы, в которую входит владелец файла. Это упрощает пользователям, принадлежащим к одной группе, совместный доступ к каталогам. Следует также учитывать, что для исполняемого файла бит `setgid` имеет другое значение, но неоднозначность практически никогда не возникает.

---

[6]Если содержимое файла начинается с символов **#!**, он обрабатывается ядром напрямую. Однако если интерпретатор команд не указан или указан неправильно, ядро откажется выполнять файл. В этом случае текущий интерпретатор предпримет вторую попытку выполнить сценарий, вызвав команду **/bin/sh**, которая обычно связана с оболочкой **Almquist** или **bash** (см. раздел 7.3). Свен Машек (Sven Mascheck) ведет очень подробную страницу, посвященную истории, реализации и кроссплатформным вопросам, связанным с использованием символов **#!** (так называемые shebang).

Дополнительный бит

Бит, которому в коде доступа соответствует восьмеричное значение 1000, называется *дополнительным* (sticky bit). В первых UNIX-системах дополнительный бит запрещал выгрузку программ из памяти. Сегодня он утратил свое значение и попросту игнорируется.

Если дополнительный бит установлен для каталога, то файловая система позволит удалять и переименовывать его файлы только владельцу каталога, владельцу файла или суперпользователю. Иметь одно лишь право записи в каталог недостаточно. Такая мера позволяет несколько лучше защитить каталоги вроде **/tmp**.

Команда ls: просмотр атрибутов файла

Файловая система хранит для каждого файла около сорока информационных полей; большая часть из них используется самой файловой системой. Администратора, в основном, интересует количество жестких ссылок, владелец, группа, код доступа, время последнего обращения и последней модификации, размер и тип файла. Всю эту информацию можно получить с помощью команды **ls -l** (или команды **ls -ld** в случае каталога; без флага **-d** команда **ls** перечисляет содержимое каталога).

Для каждого файла хранится также время последнего изменения атрибутов. Традиционное название этого поля ("ctime" от "change time" — время изменения) многих вводит в заблуждение, так как они полагают, что это время создания файла. На самом деле здесь зафиксировано время последнего изменения одного из атрибутов файла (владелец, код доступа и так далее), но не его содержимого.

Рассмотрим пример.

```
$ ls -l /usr/bin/gzip
-rwxr-xr-x  4  root  wheel 37432 Nov 11 2016 /usr/bin/gzip
```

В первом поле задается тип файла и режим доступа к нему. Поскольку первый символ — дефис, значит, перед нами обычный файл (см. табл. 5.2).

Следующие девять символов — это три набора битов режима. Порядок наборов таков: владелец, группа, другие пользователи. В листинге команды **ls** биты режима представляются буквами r, w и x (чтение, запись и выполнение). В показанном примере владелец имеет все права доступа к файлу, а остальные пользователи — только право на чтение и выполнение.

Если бы был установлен бит setuid, то вместо буквы x, обозначающей право владельца на выполнение, стояла бы буква s. В случае бита setgid вместо буквы x для группы тоже была бы указана буква s. Последний бит режима (право выполнения для других пользователей) представляется буквой t, когда для файла задан sticky-бит. Если бит setuid/setgid или sticky-бит установлен, а надлежащий бит выполнения — нет, эти биты представляются символами S и T, что указывает на игнорирование данных атрибутов вследствие ошибки.

Второе поле листинга представляет собой счетчик ссылок на файл. В данном случае здесь стоит цифра 4, свидетельствующая о том, что **/usr/bin/gzip** — одно из четырех имен файла (три других варианта этого файла — **gunzip**, **gzcat** и **zcat** — находятся в каталоге **/usr/bin**). Каждый раз при создании жесткой ссылки на файл этот счетчик увеличивается на единицу. Символические ссылки в счетчике не учитываются.

Любой каталог имеет минимум две жесткие ссылки: из родительского каталога и из специального файла "." внутри самого каталога.

Следующие два поля определяют владельца и группу файла. В данном примере файл принадлежит пользователю root и одноименной группе. В действительности ядро хранит эти данные не в строковом виде, а в виде идентификаторов. Если символьные версии идентификаторов определить невозможно, в этих полях будут отображаться числа. Так происходит, когда запись пользователя или группы была удалена из файла, соответственно, **/etc/passwd** или **/etc/group**. Не исключено также, что возникла ошибка в базе данных LDAP (описана в главе 17), если она используется.

В следующем поле отображается размер файла в байтах. Рассматриваемый файл имеет размер 37432 байт. Далее указана дата последней модификации файла: 11 ноября 2016 г. В последнем поле листинга приведено имя файла: **/usr/bin/gzip**.

Для файла устройства команда **ls** выдает несколько иную информацию.

```
$ ls -l /dev/tty0
crw--w----. 1 root tty 4, 0 Aug 3 15:12 /dev/tty0
```

Большинство полей осталось теми же, но вместо размера в байтах показаны старший и младший номера устройства. Имя **/dev/tty0** относится в данной системе (Red Hat) к первой виртуальной консоли, управляемой драйвером устройства 4 (драйвер терминала). Точка в конце названия режима означает отсутствие списка управления доступом (ACL, см. раздел 5.6).

При поиске жестких ссылок может пригодиться команда **ls -i**, отображающая для каждого файла номер его индексного дескриптора. Не вдаваясь в детали реализации файловой системы, скажем, что этот номер представляет собой индекс таблицы, в которой перечислены все файлы системы. Именно на индексные дескрипторы ссылаются файловые записи каталогов. Жесткие ссылки, указывающие на один и тот же файл, будут иметь одинаковый номер. Для того чтобы составить представление о "паутине" ссылок, воспользуйтесь командой **ls -li** для определения числа ссылок и номера индексного дескриптора какого-нибудь файла, а затем найдите его "двойников" с помощью команды **find**[7].

Другими важными опциями команды **ls** являются **-a**, отображающая все записи в каталоге (даже те файлы, имена которых начинаются с точки), **-t**, выполняющая сортировку файлов по времени изменения (или **-tr**, осуществляющая сортировку в обратном хронологическом порядке), **-F**, отображающая имена файлов с выделением каталогов и исполняемых файлов, **-R**, выводящая рекурсивный список, и **-h**, которая отображает размеры файлов в удобной для человеческого восприятия форме (например, 8K или 53M).

В большинстве версий **ls** теперь по умолчанию используются файлы цветового кодирования, если ваша программа терминала поддерживает эту возможность (в большинстве случаев эта возможность существует). Команда **ls** указывает цвета в соответствии с ограниченной и абстрактной палитрой ("красный", "синий" и т. д.), и конечная программа должна отображать эти запросы в определенные цвета. Вам может потребоваться настроить оба варианта **ls** (переменные окружения LSCOLORS или LS_COLORS). Кроме того, вы можете просто удалить конфигурацию по умолчанию для раскраски (обычно **/etc/profile.d/colorls***), чтобы полностью исключить цвета.

Команда chmod: изменение прав доступа

Код доступа к файлу можно изменить с помощью команды **chmod**. Такое право есть лишь у владельца файла и суперпользователя. В первых UNIX-системах код доступа

[7]Попробуйте применить команду **find** *точка_монтирования* **-xdev -inum** *индексный_дескриптор* **-print**.

задавался в виде восьмеричного числа. В современных версиях поддерживается также система мнемонических обозначений. Первый способ удобнее для системного администратора, но при этом можно задать только абсолютное значение кода доступа. В случае использования мнемонического синтаксиса разрешается сбрасывать и устанавливать отдельные биты режима.

Первым аргументом команды **chmod** является спецификация прав доступа. Второй и последующий аргументы — это имена файлов, права доступа к которым подлежат изменению. При использовании восьмеричной формы записи первая цифра относится к владельцу, вторая — к группе, а третья — к другим пользователям. Если необходимо задать биты `setuid` или `setgid` или дополнительный бит, следует указывать не три, а четыре восьмеричные цифры: первая цифра в этом случае будет соответствовать трем специальным битам.

В табл. 5.3 показано восемь возможных комбинаций для каждого трехбитового набора, где символы `r`, `w` и `x` обозначают право чтения, записи и выполнения соответственно.

Таблица 5.3. Коды доступа в команде `chmod`

Восьмеричное число	Двоичное число	Режим доступа	Восьмеричное число	Двоичное число	Режим доступа
0	000	`---`	4	100	`r--`
1	001	`--x`	5	101	`r-x`
2	010	`-w-`	6	110	`rw-`
3	011	`-wx`	7	111	`rwx`

Например, команда **`chmod 711 myprog`** предоставляет владельцу все права, а остальным пользователям — только право выполнения[8].

При использовании мнемонического синтаксиса вы объединяете множество исполнителей (`u` — пользователь, `g` — группа или `o` — другой) с оператором (`+` — добавить, `-` — удалить и `=` — присвоить) и набором прав доступа. Более подробное описание мнемонического синтаксиса можно найти на man-странице команды **chmod**, но синтаксис всегда лучше изучать на примерах. В табл. 5.4 приведено несколько примеров мнемонических спецификаций.

Таблица 5.4. Примеры мнемонических спецификаций команды `chmod`

Спецификация	Назначение
`u+w`	Владельцу файла дополнительно дается право записи
`ug=rw,o=r`	Владельцу и членам группы дается право чтения/записи, остальным пользователям — право чтения
`a-x`	У всех пользователей отбирается право выполнения
`ug=srx,o=`	Владельцу и членам группы дается право чтения/выполнения, устанавливаются также биты SUID и SGID; остальным пользователям запрещается доступ к файлу
`g=u`	Членам группы предоставляются такие же права, что и владельцу

[8] Если файл **`myprog`** является сценарием интерпретатора команд, должно быть задано также право чтения для остальных пользователей. Файлы сценариев, запускаемые интерпретатором, открываются и читаются в текстовом виде, а бинарные файлы выполняются непосредственно ядром, поэтому для них не нужно задавать право чтения.

Символ **u** ("*u*ser") обозначает владельца файла, **g** ("*g*roup") — группу, **o** ("*o*thers") — других пользователей, **a** ("*a*ll") — всех пользователей.



В системах Linux у существующего файла можно "заимствовать" права доступа. Например, при выполнении команды **chmod --reference=filea fileb** код доступа для файла **fileb** будет установлен таким же, как у файла **filea**.

При наличии опции **-R** команда **chmod** будет рекурсивно обновлять права доступа ко всем файлам указанного каталога и его подкаталогов. Здесь удобнее всего придерживаться мнемонического синтаксиса, чтобы менялись только те биты, которые заданы явно. Например, команда

```
$ chmod -R g+w mydir
```

добавляет групповое право записи к каталогу **mydir** и его содержимому, не затрагивая остальные права.

При установке битов выполнения будьте осторожны с выполнением команды **chmod -R**. Ведь применительно к каталогу и файлу бит выполнения интерпретируется по-разному. Следовательно, вряд ли реальный результат выполнения команды **chmod -R a-x** будет соответствовать ожидаемому вами. Для выбора исключительно обычных файлов применяйте команду **find**.

```
$ find mydir -type f -exec chmod a-x {} ';'
```

Команды chown и chgrp: смена владельца и группы

Команды **chown** и **chgrp** предназначены для изменения владельца файла и группы соответственно. Синтаксис этих команд подобен синтаксису команды **chmod**, за исключением того, что первый аргумент содержит нового владельца или группу соответственно.

Для того чтобы изменить группу файла, необходимо либо быть владельцем файла и входить в назначаемую группу, либо быть суперпользователем. Правила изменения владельца в целом усложнились и зависят от конкретной системы. В большинстве систем предусмотрены средства настройки поведения команды **chown** в зависимости от выполняемого процесса.

Как и команда **chmod**, команды **chown** и **chgrp** имеют флаг **-R**, который задает смену владельца или группы не только самого каталога, но и всех его подкаталогов и файлов. Например, последовательность команд

```
$ sudo chown -R matt ~matt/restore
$ sudo chgrp -R staff ~matt/restore
```

можно применить для определения прав доступа к файлам для пользователя `matt`, которые были восстановлены из резервной копии. Если вы устанавливаете домашний (исходный) каталог пользователя, не следует пытаться выполнять команду **chown** для файлов, имена которых начинаются с точки.

```
$ sudo chown -R matt ~matt/.*
```

Дело в том, что указанному шаблону соответствует также файл **~matt/..**, поэтому будет изменен владелец родительского каталога и, возможно, домашних каталогов других пользователей.

Команда **chown** может изменить владельца файла и группу одновременно с помощью такого синтаксиса.

```
chown пользователь:группа файл ...
```

Например:

```
$ sudo chown -R matt:staff ~matt/restore
```

На самом деле можно не указывать пользователя или группу, что делает команду **chgrp** излишней. Если включить двоеточие, но не назвать конкретную группу, то версия команды **chown** в системе Linux использует группу по умолчанию для пользователя.

Некоторые системы принимают обозначение *пользователь.группа* как эквивалентное обозначению *пользователь:группа*. Это всего лишь историческое наследие.

Команда umask: задание стандартных прав доступа

Встроенная команда **umask** позволяет менять стандартный режим доступа к создаваемым файлам. Каждый процесс имеет собственный атрибут **umask**; встроенная в интерпретатор команда **umask** устанавливает собственное значение **umask**, которое затем наследуется выполняемыми вами командами.

Значение **umask** задается в виде трехзначного восьмеричного числа, которое соответствует аннулируемым правам доступа. При создании файла код доступа к нему устанавливается равным разнице между величиной, которую запрашивает создающая файл программа, и значением **umask**. Возможные комбинации битов режима для команды `umask` перечислены в табл. 5.5.

Таблица 5.5. Схема кодирования значения umask

Восьмеричное число	Двоичное число	Режим доступа	Восьмеричное число	Двоичное число	Режим доступа
0	000	rwx	4	100	-wx
1	001	rw-	5	101	-w-
2	010	r-x	6	110	--x
3	011	r--	7	111	---

Например, команда **umask 027** предоставляет все права владельцу файла, запрещает читать файл членам группы и не дает никаких прав другим пользователям. Стандартное значение **umask** равно, как правило, `022`, т.е. право модификации файла предоставляется только его владельцу.

🕮 Дополнительную информацию о стандартных конфигурационных сценариях см. в главе 8.

В стандартной модели управления доступом невозможно заставить пользователей установить определенное значение **umask**, потому что они всегда могут установить его иначе. Однако вы можете поместить подходящие значения по умолчанию в образцы файлов запуска, которые вы даете новым пользователям. Если вам требуется больше контроля над разрешениями на создаваемые пользователем файлы, вам необходимо будет перейти в обязательную систему контроля доступа, такую как SELinux; см. раздел 3.4.

Дополнительные флаги в системе Linux

В файловых системах Linux определен ряд вспомогательных флагов, устанавливая которые можно запрашивать особые режимы обработки файлов. Например, один флаг делает файл доступным только для дополнения, а другой — недоступным для изменения и удаления.

Флаги имеют двоичные значения, символизируя наличие или отсутствие заданного файла. Базовая файловая система должна поддерживать соответствующую возможность,

поэтому не все флаги могут использоваться во всех файловых системах. Кроме того, некоторые флаги являются экспериментальными, нереализованными или предназначенными только для чтения.

В системе Linux для их просмотра и изменения атрибутов файлов предназначены специальные команды **lsattr** и **chattr**. В табл. 5.6 перечислены некоторые важные флаги.

Таблица 5.6. Дополнительные флаги файловых систем Linux

Флаг	FS[a]	Назначение
A	XBE	Никогда не обновлять время доступа (`st_atime`; в целях повышения производительности)
a	XBE	Разрешить запись в файл только в режиме добавления[б]
C	B	Запрещение копирования при записи
c	B	Сжатие содержимого
D	BE	Включить режим синхронной записи изменений каталога
d	XBE	Не создавать резервные копии (команда **dump** будет игнорировать такой файл)
i	XBE	Сделать файл неизменяемым и неудаляемымб
j	E	Ведение журнала для регистрации изменения данных и метаданных
S	XBE	Включить режим синхронной записи изменений (без буферизации)
X	B	Отмена сжатия данных, если этот режим включен по умолчанию

[a] X = XFS, B = Btrfs, E = ext3 и ext4.

[б] Может быть установлен только суперпользователем.

Как и следовало ожидать от такого случайного набора функций, значение этих флагов для администраторов меняется. Главное, чтобы помнить, что если какой-то конкретный файл ведет себя странно, проверьте его с помощью команды **lsattr**, чтобы узнать, включен ли один или несколько флагов.

Отказ от учета времени последнего доступа (флаг **A**) может повысить производительность в некоторых ситуациях. Однако его значение зависит от реализации и доступа к файловой системе; вам придется провести самостоятельное тестирование производительности. Кроме того, современные ядра теперь по умолчанию устанавливают файловые системы с параметром `relatime`, что сводит к минимуму обновления в переменной `st_atime` и делает флаг **A** в значительной степени устаревшим.

Флаги, запрещающие изменение и разрешающие только добавление (**i** и **a**), были в значительной степени задуманы как способы сделать систему более устойчивой к хакерским подделкам или враждебному коду. К сожалению, они могут запутать программное обеспечение и защищать только от хакеров, которые плохо знают, как использовать команду **chattr -ia**. Реальный опыт показал, что эти флаги чаще используются хакерами, чем для защиты от них.

🕮 Дополнительную информацию об управлении конфигурацией см. в главе 23.

Мы видели несколько случаев, когда системные администраторы использовали флаг **i** (запрет изменений) для предотвращения изменений, которые в противном случае были бы внесены системой управления конфигурацией, такой как Ansible или Salt. Излишне говорить, что это создает путаницу, когда детали забыты, и никто не может понять, почему управление конфигурацией не работает. Никогда не делайте этого — про-

сто подумайте о стыде, который испытает ваша мать, когда узнает, что вы натворили. Устраните проблему в системе управления конфигурацией — слушайтесь свою маму.

Флаг, отменяющий резервное копирование (`d`), потенциально может представлять интерес для системных администраторов, но поскольку он является рекомендацией, убедитесь, что ваша резервная система его поддерживает.

Флаги, которые влияют на ведение журнала и поддержку синхронности (`D`, `j` и `S`), существуют в основном для поддержки баз данных. Они не являются общедоступными для системных администраторов. Все эти параметры могут значительно снизить производительность файловой системы. Кроме того, известно, что вмешательство в синхронизацию записи затрудняет работу программы `fsck` в файловых системах ext*.

5.6. Списки управления доступом

Традиционная девятибитовая система контроля доступа "владелец/группа/другие" позволяет решать большинство задач администрирования. Хотя у нее есть явные ограничения, она хорошо согласуется с традициями (кое-кто может сказать "с устаревшими традициями") простоты и предсказуемости UNIX.

Списки управления доступом или ACL (access control lists) — более мощный и в то же время более сложный метод управления доступом к файлам. Каждому файлу или каталогу может соответствовать список ACL, в котором перечислены правила установки прав доступа к данному файлу или каталогу. Каждое правило в списке ACL называется *записью управления доступом* (access control entry — ACE).

Запись управления доступом идентифицирует пользователя или группу, к которой она применяется, и определяет набор привилегий, которыми наделяются эти пользователи. Списки ACL не имеют фиксированной длины и могут содержать определения прав множества пользователей или групп. Большинство операционных систем ограничивает длину отдельного списка ACL, но это ограничение может быть довольно слабым (обычно 32 записи), которое достигается не часто.

Более сложные системы позволяют администраторам указывать частичные наборы прав или отрицательные права, а некоторые из систем поддерживают наследование, которое позволяет определять права доступа для передачи их вновь создаваемым элементам файловых систем.

📖 Дополнительную информацию об управлении конфигурацией см. в главе 23.

Предупреждение

Списки ACL получили широкое распространение и поэтому будут занимать все наше внимание в остальной части этой главы. Однако ни один из перечисленных ниже фактов не следует толковать как поощрение их использовать. Списки ACL имеют собственную нишу, но она находится вне основного направления администрирования UNIX и Linux.

Списки ACL существуют прежде всего для обеспечения совместимости с системами Windows и удовлетворения потребностей малого сегмента предприятий, которые фактически требуют гибкости на уровне ACL. Они не являются блестящим следующим поколением средств управления доступом и не предназначены для вытеснения традиционной модели.

Сложность ACL создает несколько потенциальных проблем. Списки ACL утомительны в использовании, но также могут вызывать неожиданные взаимодействия с системами

резервного копирования, которые не предназначены для работы с ACL, сетевыми файловыми службами и даже с простыми программами, такими как текстовые редакторы.

Поддержка списков ACL также, как правило, усложняется по мере увеличения количества записей. В реальном мире они часто содержат рудиментарные записи, а также записи, которые служат только для компенсации проблем, вызванных предыдущими записями. Эти сложные списки можно реорганизовать и упростить, но это рискованно и требует много времени, поэтому делается редко.

В прошлом копии этой главы, которые мы отправляли профессиональным администраторам для просмотра, часто возвращались с такими заметками, как "Эта часть выглядит отлично, но я ничего не могу сказать, потому что я никогда не использовал ACL".

Типы ACL

В качестве доминирующих стандартов для систем UNIX и Linux появились два вида списков ACL: POSIX ACL и NFSv4 ACL.

Версия POSIX восходит к спецификации, созданной в середине 1990-х годов. К сожалению, ни один фактический стандарт не был принят, и первоначальные реализации сильно различались. В наши дни мы находимся в лучшей ситуации. Все системы свелись в основном к синхронизации ACL POSIX и общему набору команд, **`getfacl`** и **`setfacl`**, для управления ими.

В первом приближении модель ACL POSIX просто расширяет традиционную систему разрешений UNIX `rwx` для обеспечения разрешений для нескольких групп и пользователей.

После появления списков POSIX ACL для систем UNIX и Linux все более распространенным явлением стал обмен файловыми системами с операционной системой Windows, которая имеет свой собственный набор соглашений ACL. Здесь ситуация усложняется, поскольку Windows порождает множество различий, которые не встречаются ни в традиционной модели UNIX, ни в эквиваленте ACL POSIX. Списки ACL Windows также семантически более сложны; например, они допускают отрицательные разрешения (записи отрицания) и имеют сложную схему наследования.

Архитекторы четвертой версии NFS — общего протокола обмена файлами — хотели внедрить списки ACL как сущности первого класса. Из-за разделения UNIX/Windows и несоответствий между реализациями ACL UNIX было ясно, что системы на концах NFSv4-соединения часто могут иметь разные типы. Каждая система может понимать ACL, NFSv4, POSIX ACL, Windows ACL или вообще не ACL.

🕮 Дополнительную информацию о протоколе NFSC см. в главе 21.

Стандарт NFSv4 должен быть совместим со всеми этими различными мирами, не вызывая слишком много сюрпризов или проблем безопасности. Учитывая это ограничение, неудивительно, что списки ACL NFSv4 по сути являются объединением всех существующих систем. Они являются строгим надмножеством списков ACL POSIX, поэтому любой ACL POSIX может быть представлен как ACL NFSv4 без потери информации. В то же время в ACL NFSv4 содержатся все биты разрешений, найденные в системах Windows, и у них также есть большинство семантических функций Windows.

Реализация списков ACL

Теоретически ответственность за поддержку и функционирование систем ACL можно было бы переложить на ряд других компонентов операционной системы. Системы

ACL могли бы быть реализованы ядром от имени всех файловых систем, отдельными файловыми системами или, может быть, такими высокоуровневыми программами, как серверы NFS и SMB.

На практике поддержка ACL зависит от операционной и файловой систем. Файловая система, которая поддерживает списки ACL в одной операционной системе, может не поддерживать их в другой или может предусматривать несколько другую реализацию, поддерживаемую другими командами.

Демоны службы файлов отображают собственную схему ACL своего узла (или схемы) в соответствии с соглашениями, соответствующими файловому протоколу: NFSv4 ACL для NFS и ACL для Windows для SMB. Детали этого отображения зависят от реализации файлового сервера. Как правило, правила сложны и настраиваются с помощью опций конфигурации.

Поскольку реализации списков ACL зависят от конкретной файловой системы и операционные системы поддерживают несколько реализаций файловых систем, во многих системах предусматривается поддержка нескольких типов списков ACL. Даже отдельно взятая файловая система может предложить несколько вариантов ACL, как в системе ZFS. Если возможно использование нескольких систем ACL, команды по управлению ими могут быть одинаковыми или различными — все зависит от конкретной системы.

Поддержка ACL в системе Linux



Система Linux стандартизировала списки ACL в стиле POSIX. Протокол NFSv4 ACL на уровне файловой системы не поддерживается, хотя, конечно, системы Linux могут монтировать и совместно использовать файловые системы NFSv4 по сети.

Преимущество этой стандартизации заключается в том, что почти все файловые системы Linux теперь включают поддержку ACL POSIX, включая XFS, Btrfs и семейство ext*. Даже ZFS, чья собственная система ACL поддерживает протокол NFSv4, была перенесена в Linux с помощью POSIX ACL. Стандартные команды **`getfacl`** и **`setfacl`** могут использоваться везде, независимо от типа базовой файловой системы. (Тем не менее иногда следует проверять, что для монтирования файловой системы используется правильная опция монтирования. Файловые системы обычно поддерживают параметр `acl`, параметр `noacl` или оба, в зависимости от их значений по умолчанию.)

🕮 Дополнительную информацию о сервере SMB см. в главе 22.

Система Linux имеет набор команд (**`nfs4_getfacl`**, **`nfs4_setfacl`** и **`nfs4_editfacl`**) для обработки списков файлов ACL NFSv4, установленных на серверах NFS. Однако эти команды не могут применяться к локально сохраненным файлам. Более того, они редко включаются в дистрибутивы программного обеспечения по умолчанию; вам придется устанавливать их отдельно.

Поддержка ACL в системе FreeBSD



Система FreeBSD поддерживает как ACL POSIX, так и ACFS NFSv4. Его родные команды **`getfacl`** и **`setfacl`** были расширены, чтобы включить ACL в стиле NFSv4. Поддержка ACL NSFv4 появилась относительно недавно (с 2017 г.).

Файловые системы UFS и ZFS поддерживают ACL в стиле NFSv4, причем UFS также поддерживает ACL POSIX. Потенциальным источником путаницы здесь является

файловая система ZFS, которая поддерживает только протокол NFSv4 в системе BSD (и ее предшественнике — системе Solaris) и только POSIX в системе Linux.

Для файловой системы UFS следует использовать один из вариантов монтирования `acls` или `nfsv4acls`, чтобы указать, чего именно вы хотите. Эти варианты являются взаимоисключающими.

Обзор POSIX ACL

По существу, списки POSIX ACL Linux представляют собой простое расширение стандартной 9-битовой UNIX-модели прав доступа. Система поддерживает только права чтения, записи и выполнения. Такие дополнительные атрибуты, как `setuid` и дополнительные биты, обрабатываются исключительно традиционными битами определения режима.

Списки ACL позволяют устанавливать биты `rwx` независимо для любой комбинации пользователей и групп. Возможный вид отдельных записей в ACL приведен в табл. 5.7.

Таблица 5.7. Записи, которые могут встретиться в списке управления доступом

Формат	Пример	Получатель права доступа
`user::`*права_доступа*	`user::rw-`	Владелец файла
`user:`*имя_пользователя*`:`*права_доступа*	`user:trent:rw-`	Конкретный пользователь
`group::`*права_доступа*	`group::r-x`	Группа, которая владеет файлом
`group:`*имя_группы*`:`*права_доступа*	`group:staff:rw-`	Конкретная группа
`other::`*права_доступа*	`other::---`	Всех другие пользователи
`mask::`*права_доступа*	`mask::rwx`	Все, кроме владельца и других пользователей[a]

[a]Иногда маски не совсем понятны; пояснения к ним даны далее в этом разделе.

Пользователей и группы можно задавать именами или идентификаторами UID/GID. Точное число записей, которое может содержать список ACL, зависит от реализации файловой системы, но, как правило, оно не меньше 32. Вероятно, это практический предел для поддержки ACL.

Взаимодействие между традиционными режимами и списками ACL

При использовании списков ACL файлы сохраняют свои исходные биты режима, но при этом автоматически поддерживается целостность атрибутов, и два набора прав доступа не могут вступить в конфликт. Ниже приведен пример (используется синтаксис команд системы Linux) автоматического обновления записей ACL в ответ на изменения, выполненные "старой" командой **chmod**.

```
$ touch example
$ ls -l example
-rw-rw-r-- 1 garth garth   0 Jun 14 15:57 example
$ getfacl example
# file: example
# owner: garth
# group: garth
user::rw
group::rw
other::r--
$ chmod 640 example
```

```
$ ls -l example
-rw-r----- 1 garth garth   0 Jun 14 15:57 example
$ getfacl --omit-header example[9]
user::rw-
group::r--
other::---
```

Эта принудительная целостность атрибутов позволяет более старым программам, которые даже не подозревают о существовании списков ACL, достаточно успешно работать в среде, использующей эти списки. Однако существует и определенная проблема. Несмотря на то что ACL-запись `group::` в приведенном примере вроде и отслеживает средний набор традиционных битов режима, это не всегда так.

Для того чтобы понять, в чем здесь дело, предположите, что унаследованная программа очищает биты разрешения записи во всех трех наборах прав традиционного режима (например, **chmod ugo-w** *файл*). Понятно, что целью выполнения этой команды является запрет выполнения записи в файле кем-либо. Но что произойдет, если результирующий список ACL будет выглядеть следующим образом?

```
user::r--
group::r--
group:staff:rw-
other::r--
```

С точки зрения унаследованной программы файл выглядит неизменяемым, хотя в действительности он доступен для записи любому члену группы `staff`. Подобная ситуация нежелательна. Для снижения вероятности недоразумений приняты следующие правила.

- По определению записи `user::` и `other::` списка ACL идентичны битам режима "владелец" и "другие" традиционного режима файла. Изменение режима ведет к изменению соответствующих записей ACL и наоборот.
- Во всех случаях эффективными правами доступа, предоставляемыми владельцу файла и пользователям, которые не указаны как-либо иначе, являются те, которые указаны в соответствующих записях `user::` и `other::` списка ACL.
- Если файл не имеет явно определенного списка ACL или имеет ACL, который состоит только из одной записи `user::`, одной записи `group::` и одной записи `other::`, эти записи идентичны трем наборам традиционных битов режима. Именно такой случай продемонстрирован в приведенном выше примере применения команды **getfacl**. (Подобный ACL называют минимальным, и он не требует действительной реализации в виде логически отдельного ACL.)
- В более сложных списках ACL традиционные биты режима соответствуют специальной записи ACL, называемой "маской", а не записи `group::`. Маска ограничивает доступ так, что ACL может предоставлять права всем названым пользователям, *всем* названым группам *и* группе, заданной по умолчанию.

Иначе говоря, маска определяет верхний предел прав доступа, которые система ACL может присваивать отдельным группам и пользователям. Концептуально эта маска аналогична команде **umask**, за исключением того, что маска ACL действует всегда и указывает предоставленные, а не отобранные права. Записи ACL для названых пользователей, названых групп и группы, заданной по умолчанию, могут содержать права доступа, отсутствующие в маске, но ядро будет просто их игнорировать.

[9]Это пример из системы Linux. В системе FreeBSD для подавления вывода комментариев на дисплей версия команды **getfacl** использует флаг **-q**, а не **--omit-header**.

В результате традиционные биты режима не в состоянии уменьшить права, глобально предоставленные списком ACL. Более того, удаление бита из части группы традиционного режима ведет к удалению соответствующего бита в маске ACL и, следовательно, к лишению этих прав всех пользователей, кроме владельца файла и тех, кто относится к категории "другие".

При расширении списка ACL, приведенного в предыдущем примере, для включения в него записей конкретного пользователя и группы команда **setfacl** автоматически определяет соответствующую маску.

```
$ ls -l example
-rw-r-----    1 garth garth         0 Jun 14 15:57 example
$ setfacl -m user::r,user:trent:rw,group:admin:rw example
$ ls -l example
-r--rw----+   1 garth garth 0 Jun 14 15:57 example
$ getfacl --omit-header example
user::r--
user:trent:rw-
group::r--
group:admin:rw-
mask::rw-
other::---
```

Параметр **-m** для команды **setfacl** означает "изменить": он добавляет записи, которых еще нет, и настраивает те, которые уже существуют. Обратите внимание: команда **setfacl** генерирует маску, которая позволяет вступить в действие всем правам, предоставленным в списке ACL. Для определения маски вручную нужно включить ее в список записей ACL, переданный команде **setfacl**, либо использовать опцию **-n**, чтобы помешать ее повторному генерированию командой **setfacl**.

Обратите внимание на то, что в результате выполнения второй команды **ls -l** (после команды **setfacl**) в конце режима файла выводится знак "+", означающий, что с ним теперь связан реальный список ACL. Первая команда **ls -l** не выводит знак "+", поскольку на данный момент список ACL является "минимальным".

Следует иметь в виду, что применение традиционной команды **chmod** для манипулирования правами доступа группы в файле, связанном с ACL, сказывается только на маске. Продолжим рассмотрение предыдущего примера.

```
$ chmod 770 example
$ ls -l example
-rwxrwx---+ 1 garth staff 0 Jun 14 15:57 example
$ getfacl --omit-header example
user::rwx
user:trent:rw-
group::r--
group:admin:rw-
mask::rwx
other::---
```

В данном случае результат выполнения команды **ls** вводит в заблуждение. Несмотря на обширные права, предоставленные группе, ни один из ее членов в действительности не обладает правами выполнения файла на одном лишь основании принадлежности к группе. Для того чтобы предоставить это право, придется отредактировать список ACL вручную.

Чтобы полностью удалить ACL и вернуться к стандартной системе разрешений UNIX, используйте команду **setfacl -bn**. (Строго говоря, флаг **-n** нужен только в си-

стеме FreeBSD. Без него команда **setfacl** в системе FreeBSD создает набор рутинных масок, которая позже испортит изменения в групповом режиме. Однако в системе Linux флаг **-n** можно установить, не создавая проблем.)

Определение прав доступа

При попытке получения процессом доступа к файлу идентификатор UID сравнивается с UID владельца файла. Если они совпадают, права доступа определяются правами `user::` в списке ACL. В противном случае при наличии соответствия записи конкретного пользователя в ACL права определяются этой записью и маской ACL.

При отсутствии подходящей записи конкретного пользователя файловая система пытается найти подходящую запись группы, предоставляющую запрошенные права. Эти записи также обрабатываются в сочетании с маской ACL. При отсутствии подходящей записи права доступа определяются записью `other::`.

Наследование списков ACL

Кроме типов записей, перечисленных в табл. 5.7, списки ACL каталогов могут содержать записи, определенные по умолчанию, которые передаются спискам ACL новых файлов и подкаталогов внутри них. Подкаталоги получают свои записи как в форме записей активного ACL, так и в форме записей, заданных по умолчанию. Следовательно, исходные заданные по умолчанию записи могут со временем распространяться по нескольким уровням иерархии каталогов.

При копировании записей, заданных по умолчанию, связь между родительским и дочерними списками ACL не сохраняется. Изменения в заданных по умолчанию записях родительского каталога не отражаются в списках ACL существующих подкаталогов.

С помощью команды **setfacl -dm** можно установить записи ACL по умолчанию. Кроме того, записи, заданные по умолчанию, можно включить в обычные записи списка управления доступом, поставив перед ними префикс `:`.

Если каталог имеет *хотя бы одну* запись по умолчанию, он должен включать полный набор значений по умолчанию для записей `user::`, `group ::`, `other ::` и `mask ::`. Команда **setfacl** будет заполнять любые записи по умолчанию, которые вы не укажете путем копирования их из текущего списка разрешений ACL, создавая маску как обычно.

Списки NFSv4 ACL



В этом разделе рассмотрим характеристики списков NFSv4 ACL и синтаксис команд на примере системы FreeBSD. Они не поддерживаются системой Linux (демоны которой отличаются от служб NFS).

С точки зрения структуры списки NFSv4 ACL аналогичны спискам ACL в системе Windows. Основное различие между ними состоит в спецификации категории, к которой относится запись контроля прав доступа.

В списках ACL обеих систем запись хранится в виде строки. Для списков Windows ACL эта строка обычно содержит идентификатор защиты Windows (Windows security identifier — SID), в то время как NFSv4-строка, как правило, имеет такой формат: `user:`*`имя_пользователя`* или `group:`*`имя_группы`*. Она также может иметь один из специальных маркеров: `owner@`, `group@` или `everyone@`. В действительности эти "специальные" записи можно считать самыми популярными, поскольку они связаны с битами режима доступа, установленными для каждого файла.

Такие системы, как Samba, которые разделяют файлы между системами UNIX и Windows, должны обеспечивать определенный способ преобразования данных между Windows и NFSv4.

Модель прав доступа систем Windows и NFSv4 отличается большей детализацией по сравнению с традиционной UNIX-моделью "чтение-запись-выполнение". Рассмотрим ее основные отличительные черты.

- В системе NFSv4 разрешение на создание файлов в каталоге отличается от разрешения создавать подкаталоги.
- В системе NFSv4 предусмотрен отдельный бит разрешения на "добавление".
- В системе NFSv4 предусмотрены отдельные биты разрешения на чтение и запись для данных, файловых атрибутов, расширенных атрибутов и ACL.
- В системе NFSv4 реализовано управление возможностью пользователей менять владельца файла посредством стандартной системы ACL. В традиционной модели UNIX возможность менять владельца файла обычно имеет только суперпользователь.

В табл. 5.8 приведены различные разрешения (права доступа), которые могут назначаться в системе NFSv4, а также однобуквенные коды, используемые для их представления.

Таблица 5.8. Права доступа к файлам в системе NFSv4

Код	Имя	Права доступа
r	read_data	Чтение данных файла или получение содержимого каталога
w	write_data	Запись данных в файл или создание файла в каталоге
x	execute	Выполнение файла как программы
p	append_data	Добавление данных в файл или создание подкаталога в каталоге
D	delete_child	Удаление дочернего каталога в каталоге
d	delete	Удаление
a	read_attributes	Чтение нерасширенных атрибутов
A	write_attributes	Запись нерасширенных атрибутов
R	read_xattr	Чтение именованных ("расширенных") атрибутов
W	write_xattr	Запись именованных ("расширенных") атрибутов
c	read_acl	Чтение списка управления доступом
C	write_acl	Запись списка управления доступом
o	write_owner	Изменение владельца
s	synchronize	Допускает запросы на синхронный ввод-вывод (обычно игнорируется)

Несмотря на то что в системе NFSv4 модель прав доступа достаточно детализирована, отдельные разрешения должны быть очевидными, т.е. не должны требовать дополнительных разъяснений. Разрешение "synchronize" позволяет клиенту указать, что модификации файла должны быть синхронизированы, т.е. вызовы, связанные с записью данных, не должны завершаться до тех пор, пока данные не будут реально сохранены на диске.

Расширенный атрибут — это именованная часть данных, которая сохраняется вместе с файлом. Большинство современных файловых систем поддерживают такие атрибуты, хотя они пока не нашли широкого применения. В настоящее время расширенные атри-

буты в основном используются для сохранения самих списков ACL. Однако модель прав доступа в системе NFSv4 предусматривает обработку списков ACL отдельно от других расширенных атрибутов.

В системе FreeBSD владелец файла всегда имеет разрешения `read_acl`, `write_acl`, `read_attributes` и `write_attributes`, даже если в списках управления доступом заданы другие значения.

Объекты NFSv4, для которых можно задавать права доступа

В дополнение к обычным спецификаторам `user:`*`имя_пользователя`* или `group:`*`имя_группы`* в системе NFSv4 определено еще несколько объектов, которые могут быть указаны в списке ACL. Самыми важными из них являются `owner@`, `group@` и `everyone@`, которые связаны с традиционными категориями в девятибитовой модели прав доступа.

Система NFSv4 отличается от POSIX по ряду признаков. Прежде всего, тем, что NFSv4 не имеет стандартного объекта, используемого по умолчанию для управления наследованием списков ACL. Вместо этого любая отдельная запись управления доступом (ACE) может быть помечена как передающаяся по наследству (см. ниже раздел "Наследование списков ACL в протоколе NFSv4"). Кроме того, система NFSv4 не использует маску для согласования прав доступа, заданных в режиме файла с помощью списка ACL. Режим, необходимый для согласования с параметрами, задаваемыми для `owner@`, `group@` и `everyone@`, а также файловые системы, которые реализуют списки NFSv4 ACLs, должны сохранять эту совместимость при обновлении режима либо списка ACL.

Определение прав доступа в протоколе NFSv4

Система NFSv4 отличается от POSIX тем, что в записи ACE указывается только частичный набор разрешений. Каждая запись ACE является либо разрешающей, либо запрещающей; она действует больше как маска, чем авторитетная спецификация всех возможных разрешений. Множественные ACE могут применяться к любой заданной ситуации.

При принятии решения о разрешении конкретной операции файловая система считывает список ACL по порядку, обрабатывая записи до тех пор, пока все запрошенные разрешения не будут предоставлены или какое-либо запрошенное разрешение будет отклонено. Рассматриваются только те записи, чья организация строки совместима с идентификатором текущего пользователя. Этот процесс итеративной оценки означает, что значения `owner@`, `group@` и `all@` не являются точными аналогами соответствующих битов традиционного режима. Список ACL может содержать множественные копии этих элементов, и их приоритет определяется порядком появления в ACL, а не соглашением. В частности, значение `everyone@` действительно относится ко всем пользователям, а не только к тем, которые не заданы более конкретно.

Файловая система может "пройти" список NFSv4 ACL до конца, не получив определенного ответа на запрос прав доступа. В этом случае стандарт NFSv4 считает полученный результат неопределенным, однако большинство существующих реализаций выберут доступ с отказом, во-первых, потому, что такое соглашение используется в системе Windows, и, во-вторых, потому, что это единственный вариант, который имеет смысл.

Наследование списков ACL в протоколе NFSv4

Подобно спискам POSIX ACL, списки NFSv4 ACL позволяют созданным объектам наследовать записи управления доступом от включающего их каталога. Однако система

NFSv4 при небольшом выигрыше в возможностях гораздо более запутана по сравнению с POSIX. Ниже перечислены причины такой запутанности.

- Любая запись может быть отмечена как наследуемая. Признаки наследования для создаваемых подкаталогов (`dir_inherit` или `d`) и создаваемых файлов (`file_inherit` или `f`) устанавливаются по отдельности.
- Можно применять различные записи управления доступом для новых файлов и новых каталогов путем создания отдельных записей управления доступом в родительском каталоге с соответствующими признаками. Можно также применить одну запись ACE ко всем новым дочерним записям (любого типа) за счет включения обоих флагов `d` и `f`.
- С точки зрения определения прав доступа записи управления доступом оказывают одинаковое влияние на родительский (исходный) каталог независимо от того, наследуемый он или нет. Если вы хотите применить запись к дочернему, но не к родительскому каталогу, включите для соответствующей записи ACE флаг `inherit_only` (`i`).
- Новые подкаталоги обычно наследуют две копии каждой записи ACE: с отключенными флагами наследования (применяется к самому подкаталогу) и с включенным флагом `inherit_only` (устанавливает новый подкаталог для распространения своих передаваемых по наследству записей ACE). Вы можете подавить создание этой второй записи ACE включением флага `no_propagate` (`n`) для копии записи ACE родительского каталога. Конечный результат состоит в том, что запись ACE распространяется только на прямых потомков исходного каталога.
- Не путайте распространение записей управления доступом с настоящим наследованием. Установка связанного с наследованием флага для записи ACE просто означает, что данная запись ACE будет скопирована в новые записи. Она не создает никаких постоянных отношений между родителем и его потомком. Если вы позже измените записи ACE для родительского каталога, дочерние каталоги при этом не будут обновлены.

Различные флаги наследования приведены в табл. 5.9.

Таблица 5.9. Флаги наследования в системе NFSv4 ACE

Код	Имя	Назначение
`f`	`file_inherit`	Распространение записи ACE на создаваемые файлы
`d`	`dir_inherit`	Распространение записи ACE на создаваемые подкаталоги
`i`	`inherit_only`	Распространение, но не применение к текущему каталогу
`n`	`no_propagate`	Распространение на новые подкаталоги, но не на их потомков

Просмотр списков NFSv4 ACL

Система FreeBSD расширила стандартные команды **`setfacl`** и **`getfacl`**, используемые для работы со списками POSIX ACL, чтобы обеспечить дополнительную возможность обработки списков NFSv4 ACL. Например, вот как выглядит список ACL для только что созданного каталога.

```
freebsd$ mkdir example
freebsd$ ls -ld example
drwxr-xr-x 2 garth staff 2 Aug 16 18:52 example/
```

```
$ getfacl -q example
       owner@:rwxp--aARWcCos:------:allow
       group@:r-x---a-R-c--s:------:allow
    everyone@:r-x---a-R-c--s:------:allow
```

Флаг **-v** запрашивает подробные имена разрешений.

```
freebsd$ getfacl -qv example
   owner@:read_data/write_data/execute/append_data/read_attributes/
       write_attributes/read_xattr/write_xattr/read_acl/write_acl/
       write_owner/synchronize::allow
   group@:read_data/execute/read_attributes/read_xattr/read_acl/
       synchronize::allow
   everyone@:read_data/execute/read_attributes/read_xattr/read_acl/
       synchronize::allow
```

🕮 Мы отформатировали этот вывод и расставили косые черты, чтобы его было легче читать.

Может показаться, что создаваемый каталог получает сложный список ACL, но на самом деле это не так: список ACL — всего лишь девятибитовый код режима, отображаемый в первой строке результата, преобразуемого в формат списка ACL. Совсем не обязательно хранить реальный список ACL в файловой системе, поскольку список ACL и режим доступа эквивалентны. (Такие списки ACL называются *тривиальными*.) Если бы каталог имел реальный список ACL, команда **ls**, чтобы обозначить наличие списка ACL, отображала бы биты режима со знаком "+" в конце (например, `drwxr-xr-x+`).

Каждое нумерованное выражение представляет одну запись управления доступом. Вот как выглядит ее формат.

объект:права_доступа:флаги_наследования:тип

Поле *объект* может занимать одно из таких ключевых слов, как `owner@`, `group@`, `everyone@`, или элемент в такой форме: `user:`*имя_пользователя* либо `group:`*имя_группы*. Оба поля *права_доступа* и *флаги_наследования* представляют собой списки опций, разделенные косыми чертами в развернутом виде или в стиле битовых масок команды **ls** в лаконичном виде. Поле *тип* в записи ACE задает один из двух возможных вариантов: разрешение или отказ.

Использование двоеточия в качестве разделителя в поле *объект* усложняет синтаксический разбор вывода команды **getfacl** независимо от используемого формата. Если вам необходимо обрабатывать списки ACL программно, лучше всего использовать модульный интерфейс прикладного программирования, а не сценарий синтаксического разбора выходной информации.

Взаимодействие между списками ACL и традиционными режимами

Режим и список ACL должны оставаться совместимыми, поэтому при корректировании одного из этих вариантов другой обновляется автоматически. Файловая система может определить соответствующий режим для заданного списка ACL, но ее алгоритм генерирования и обновления списков ACL в ответ на изменения режимов остается слишком сложным, особенно для существующих ACL. В частности, для категорий `owner@`, `group@` и `everyone@` система может сгенерировать несколько наборов записей (и притом противоречивых), которые будут зависеть от порядка их агрегирования.

Лучше всего никогда не изменять режим файла или каталога после применения списка ACL.

Модификация списков ACL NFSv4

Поскольку система разрешений обеспечивает согласованность между режимом файла и его ACL, все файлы имеют как минимум тривиальный список ACL. Следовательно, изменения списков ACL всегда обновляются. Изменения списка ACL осуществляются с помощью команды **setfacl**, почти так же, как и в режиме ACL POSIX Основное отличие состоит в том, что порядок записей списка управления доступом является значимым для протокола NFSv4, поэтому может потребоваться вставить новые записи в определенную точку в существующем списке ACL. Это можно сделать это с помощью флага **-a**:

setfacl -a *позиция записи файл ...*

Здесь *позиция* — это индекс существующей записи управления доступом (нумеруется начиная с нуля), перед которой должны быть вставлены новые записи. Например, команда

```
$ setfacl -a 0 user: ben: full_set :: deny ben_keep_out
```

устанавливает запись управления доступом в файле `ben_keep_out`, которая запрещает все разрешения пользователю `ben`. Обозначение **full_set** — это сокращенное обозначение, включающее все возможные разрешения. (В настоящее время эти разрешения имеют вид **rwxpDdaARWcCos**, сравните их с табл. 5.8.)

Поскольку новая запись управления доступом вставлена в нулевое положение, она проверяется первой и имеет приоритет над более поздними записями. Пользователь `ben` будет лишен доступа к файлу, даже если, например, разрешение `everyone@` предоставляет доступ другим пользователям. Для идентификации разрешений можно использовать длинные имена, такие как `write_data`. Отделите несколько длинных имен косой чертой. Не следует смешивать однобуквенные коды и длинные имена в одной команде.

Как и в списках POSIX ACL, для добавления новых записей в конец существующего списка ACL можно использовать флаг **-m**. Что касается сложных изменений в существующих списках ACL, лучше всего записать ACL в текстовый файл, отредактировать записи управления доступом в текстовом редакторе, а затем перезагрузить весь список ACL. Например:

```
$ getfacl -q file> /tmp/file.acl
$ vi /tmp/file.acl                    # Внесите необходимые изменения
$ setfacl -b -M /tmp/file.acl file
```

Параметр **-b** команды **setfacl** удаляет существующий список ACL перед добавлением записей управления доступом, перечисленных в файле **file.acl**. Это позволяет удалять записи, просто стирая их в текстовом файле.

Глава 6

Инсталляция и управление программным обеспечением



Инсталляция программных средств, настройка их конфигурации и управление ими — основные обязанности системных администраторов. Они отвечают на запросы пользователей инсталлировать и настроить конфигурацию программ, усовершенствовать средства защиты их данных и устранить прорехи в системе безопасности, а также управляют переходом к новым версиям программ, которые, с одной стороны, предлагают новые возможности, а с другой стороны, чреваты проблемами несовместимости. Администраторам обычно приходится выполнять перечисленные ниже задачи:

- осуществлять автоматизированную инсталляцию операционной системы на группу компьютеров;
- выполнять настройку операционных систем в локальных средах;
- следить за выходом исправлений и своевременно обновлять с их помощью систему и приложения;
- управлять добавочными программными пакетами.

Процесс настройки конфигурации готового дистрибутива или программного пакета, направленный на удовлетворение всех потребностей пользователей (и не допускающий нарушения локальных условий защиты, размещения файлов и топологии сети), часто называют *локализацией*. В этой главе рассматриваются методики, которые позволяют

упростить инсталляцию программного обеспечения, в том числе в крупных системах. Мы также рассмотрим процедуру инсталляции для каждого из наших примеров операционных систем, включая некоторые возможности автоматизации, использующие распространенные инструменты (предназначенные для конкретной платформы).

6.1. Инсталляция операционных систем

Процедура инсталляции дистрибутивов Linux и FreeBSD довольно проста. Обычно для инсталляции необходимо загрузиться с внешнего USB-устройства или оптического носителя, ответить на ряд вопросов, выбрать конфигурацию разделов диска и указать программе-инсталлятору, какие программные пакеты нужно установить. Такие системы, как Ubuntu и OpenSolaris, включают опцию "live", которая позволяет запускать операционную систему, не инсталлируя ее в действительности на локальный диск.

Инсталляция базовой операционной системы с локального носителя представляет собой довольно тривиальную операцию благодаря приложениям с графическим пользовательским интерфейсом, которые обеспечивают выполнение всех этапов этого процесса. В табл. 6.1 перечислены указатели на подробные инструкции по инсталляции для каждого из наших примеров дистрибутивов.

Таблица 6.1. Документация по инсталляции

Дистрибутив	Источник документации
Red Hat	`redhat.com/docs/manuals/enterprise`
CentOS	`wiki.centos.org/Manuals/Release/Notes/CentOS7`
Debian	`debian.org/releases/stable/istallmanual`
Ubuntu	`help.ubuntu.com/community/Installation`
FreeBSD	`freebsd.org/doc/handbook/bsdinstall.html`

Загрузка по сети на персональном компьютере

Любой, кому придется инсталлировать систему сразу на нескольких компьютерах, столкнется с недостатками инсталляции в интерактивном режиме. Это большие затраты времени, предрасположенность к ошибкам и скучная необходимость повторения стандартного процесса инсталляции на сотнях систем. Свести к минимуму человеческий фактор можно благодаря контрольному списку локализации, хотя даже его использование не сможет полностью защитить от всевозможных ошибок.

Для того чтобы ослабить воздействие перечисленных выше факторов, большинство систем включает возможности инсталляции по сети, которые упрощают проведение крупномасштабных инсталляций. В самых распространенных методах для загрузки системы без использования физического носителя используются сетевые протоколы DHCP и TFTP, после чего из сетевого сервера извлекаются файлы инсталляции через протоколы HTTP, NFS или FTP.

Предзагрузочная среда выполнения (Preboot eXecution Environment — PXE) позволяет полностью выполнить инсталляцию операционной системы без вмешательства пользователей. Она была разработана компанией Intel как стандарт, позволяющий загружать системы через сетевой интерфейс. Особенно хорошо она зарекомендовала себя в виртуальных средах.

Стандарт PXE действует подобно миниатюрной операционной системе, размещенной в схеме ПЗУ на вашей сетевой плате. Он предлагает системе BIOS использовать его сетевые особенности посредством стандартизованного интерфейса API. При таком взаимодействии один загрузчик может загружать по сети операционную систему на любом персональном компьютере, понимающем стандарт PXE, не устанавливая при этом какие-либо специальные драйвера для каждой сетевой платы.

🕮 Дополнительную информацию о протоколе DHCP см. в разделе 13.6.

Внешняя (сетевая) часть протокола PXE не содержит никаких сложностей и подобна процедурам загрузки по сети, используемым в других архитектурах. Компьютер осуществляет широковещательную рассылку специального DHCP-запроса с установленным PXE-флагом, а DHCP-сервер или прокси-сервер возвращает DHCP-пакет, содержащий значения PXE-параметров (имя загрузочного сервера плюс имя загрузочного файла). Клиент получает от сервера свой загрузочный файл по протоколу TFTP (возможно использование многоадресной версии этого протокола), после чего запускает его. Процедура загрузки PXE проиллюстрирована на рис. 6.1.

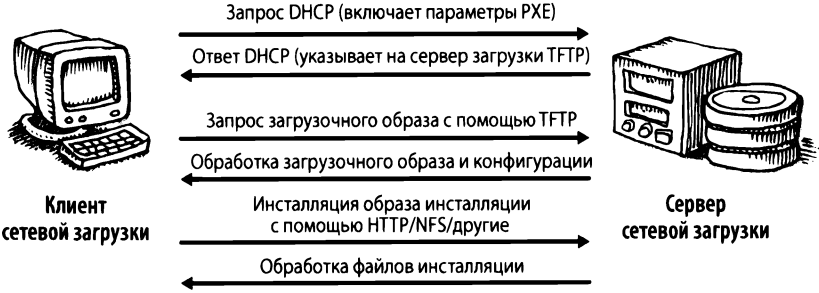


Рис. 6.1. Процесс загрузки и инсталляции PXE

DHCP, TFTP и файловые серверы могут быть расположены на разных хостах. Загрузочный файл, предоставленный TFTP, содержит меню с указателями на доступные загрузочные образы операционной системы, которые затем могут быть извлечены с файлового сервера с помощью HTTP, FTP, NFS или другого сетевого протокола.

Загрузка PXE чаще всего используется в сочетании с инструментами автоматической установки, такими как утилита `kickstart` в системе Red Hat или система предварительной инсталляции Debian, описанные в следующих разделах. Можно также использовать PXE для загрузки бездисковых систем, таких как тонкие клиенты.

Сервер Cobbler, рассматриваемый ниже, содержит средства, которые упрощают сетевую загрузку. Тем не менее вам все же понадобятся знания рабочих инструментов, которые лежат в основе Cobbler, начиная с PXE.

Настройка PXE

Наиболее широко используемой системой загрузки PXE является PXELINUX, автором которой является Питер Авлин (H. Peter Anvin). Она является частью набора загрузочных устройств SYSLINUX для каждого случая. Ее можно найти на сайте `syslinux.org`. Другим вариантом является система iPXE (`ipxe.org`), поддерживающая дополнительные режимы начальной загрузки, включая поддержку беспроводных сетей.

Система PXELINUX содержит загрузочный файл, который инсталлируется в каталог **tftpboot** сервера TFTP. Персональный компьютер загружает загрузочный файл и его конфигурацию с сервера; конфигурация определяет, какое ядро необходимо использовать. Эта цепочка событий может быть выполнена без вашего вмешательства; как вариант, можно создать специальное меню загрузки.

🕮 Дополнительную информацию о сервере программного обеспечения DHCP см. в разделе 17.3.

PXELINUX использует для своих загрузок PXE API и поэтому не зависит от аппаратной части в течение всего процесса загрузки. Более того, она может загружать не только Linux, но и другие операционные системы. С помощью системы PXELINUX можно инсталлировать системы FreeBSD и другие операционные системы, включая Windows.

На стороне сервера лучше всего использовать DHCP-сервер организации ISC (Internet Systems Consortium). В качестве альтернативы можно использовать Dnsmasq (`goo.gl/FNk7a`), упрощенный сервер, поддерживающий DNS, DHCP и сетевую загрузку.

Использование Kickstart — автоматизированного инсталлятора Red Hat и CentOS

RHEL 

Kickstart — это утилита, предназначенная для автоматической инсталляции программного обеспечения Red Hat. Она представляет собой сценарный интерфейс к стандартному инсталлятору Red Hat — программе Anaconda — и зависит как от версии дистрибутива, так и от имеющихся пакетов RPM. Утилита Kickstart достаточно гибкая и автоматически распознает аппаратное обеспечение системы, поэтому она может одинаково хорошо работать как на реальных, так и на виртуальных машинах. Инсталляцию с помощью утилиты Kisckstart можно осуществлять с оптического носителя, локального жесткого диска, а также серверов NFS, FTP и HTTP.

Создание файла конфигурации для утилиты Kickstart

Работа утилиты Kickstart контролируется конфигурационным файлом, который обычно называется **ks.cfg**. Формат этого файла довольно прост. Для тех, кто не любит работать с текстовыми файлами, существует графическая утилита **system-config-kickstart**, упрощающая настройку конфигурационного файла.

Конфигурационный файл утилиты Kickstart состоит из трех упорядоченных частей. Первая из них — это раздел команд, где задаются такие установки, как язык, раскладка клавиатуры и часовой пояс. Здесь же с помощью параметра **url** задается источник, из которого загружается дистрибутив (в показанном ниже примере это компьютер **installserver**).

Рассмотрим пример законченного раздела команд.

```
text
lang en_US                          # lang используется при инсталляции...
langsupport en_US                   # ... а langsupport – на этапе выполнения
keyboard us                         # Используется американская раскладка
timezone --utc America/EST          # --utc означает, что аппаратный таймер
                                    # настроен по Гринвичу
mouse
rootpw --iscrypted $6$NaCl$X5jRlREy9DqNTCXjHp075/
```

```
reboot                              # Перезагрузка после инсталляции. Рекомендуем.
bootloader --location=mbr           # Загрузчик по умолчанию — MBR
install                             # Инсталляция новой системы, не модификация
url --url http://installserver/redhat
clearpart --all --initlabel         # Очистка всех существующих разделов
part / --fstype ext3 --size 4096
part swap --size 1024
part /var --fstype ext3 -size 1 --grow
network --bootproto dhcp

auth --useshadow --enablemd5
firewall --disabled
xconfig --defaultdesktop=GNOME --startxonboot --resolution 1280x1024
     --depth 24
```

По умолчанию утилита Kickstart работает в графическом режиме, что препятствует ее автоматическому запуску. Ключевое слово `text` в начале файла исправляет данную ситуацию.

Параметр `rootpw` задает пароль суперпользователя на новом компьютере. По умолчанию пароль указывается в текстовом виде, что создает проблему с точки зрения безопасности. Всегда пользуйтесь опцией `--iscrypted`, чтобы задать зашифрованный пароль. Для того чтобы зашифровать пароль, следует использовать команду **`openssl passwd -1`**. Этот параметр устанавливает на всех ваших системах один и тот же пароль. Для того чтобы изменить пароль на этапе сборки, следует выполнить процесс после загрузки.

Директивы `clearpart` и `part` задают список разделов с указанием их размеров. С помощью опции `--grow` можно отвести всю оставшуюся область жесткого диска под один из разделов. Так можно облегчить подгонку систем с разными размерами жестких дисков. Такие усовершенствованные опции разбиения на разделы, как использование менеджера логических томов (Logical Volume Manager — LVM), поддерживаются утилитой Kickstart, но не инструментом **`system-config-kickstart`**. Обратитесь к встроенной в систему Red Hat оперативно-доступной документации за полным списком параметров форматирования дисков.

Во втором разделе файла приведен список инсталлируемых пакетов, начинающийся с директивы `%packages`. В списке могут быть указаны отдельные пакеты, коллекции (например, `@ GNOME`) либо запись `@ Everything`, обозначающая весь имеющийся набор пакетов. В первом случае задается лишь имя пакета, без номера версии и расширения **`.rpm`**. Приведем пример.

```
%packages
@ Networked Workstation
@ X Window System
@ GNOME
mylocalpackage
```

В третьем разделе конфигурационного файла указывается произвольный набор команд интерпретатора, которые подлежат выполнению утилитой Kickstart. Существует два возможных набора команд. Один из них начинается с директивы `%pre` и содержит команды, выполняемые перед началом инсталляции. Команды, выполняемые по завершении инсталляции, помечаются директивой `%post`. В обоих случаях возможности системы по распознаванию имен компьютеров ограничены, так что лучше пользоваться IP-адресами. Кроме того, команды второго набора выполняются в среде **`chrooted`**, поэтому они не имеют доступа к инсталляционному носителю.

Файл **ks.cfg** довольно легко создавать программно. Один из вариантов — использовать библиотеку Pykickstart Python, которая может читать и писать конфигурации утилиты Kickstart.

Например, предположим, что вы хотели установить разные наборы пакетов на серверах и клиентах, и у вас также есть два отдельных физических местоположения, которые требуют немного разных настроек. Вы можете использовать библиотеку `pykickstart` для написания сценария, который преобразует главный набор параметров в набор из четырех отдельных файлов конфигурации, один для серверов и один для клиентов в каждом офисе.

Изменение дополнения к пакетам может быть просто вопросом изменения основного файла конфигурации, а не изменения всех возможных файлов конфигурации. Могут быть даже случаи, когда вам нужно создавать индивидуальные файлы конфигурации для определенных хостов. В этой ситуации вы, конечно, хотите, чтобы окончательные файлы **ks.cfg** автоматически генерировались.

Создание сервера Kickstart

Утилита Kickstart ожидает, что ее инсталляционные файлы расположены так же, как и на инсталляционном компакт-диске. На сервере ее пакеты должны находиться в каталоге **RedHat/RPMS**. Если инсталляция выполняется по сети с помощью протоколов FTP, NFS или HTTP, можно либо скопировать содержимое дистрибутива (оставив дерево неизменным), либо просто использовать образы дистрибутивов ISO. Кроме того, в библиотеку можно добавлять свои собственные пакеты. Однако есть ряд нюансов, о которых следует помнить.

Получив команду инсталлировать все пакеты (запись `@ Everything` в разделе пакетов файла **ks.cfg**), утилита сначала устанавливает базовые пакеты, а затем — пользовательские, причем в алфавитном порядке. Если какой-то пользовательский пакет зависит от других пакетов, не входящих в базовый набор, назовите его наподобие **zzmypackage.rpm**, чтобы он инсталлировался последним.

Если не требуется инсталлировать все пакеты, укажите нужные в разделе `%packages` файла **ks.cfg** либо добавьте их в одну или несколько коллекций. Коллекции обозначаются записями вида `@ GNOME` и представляют собой предопределенные наборы пакетов, компоненты которых перечислены в файле **RedHat/base/comps** на сервере. К сожалению, формат этого файла плохо документирован. Коллекции задаются в строках, которые начинаются с цифры 0 или 1; единица указывает на то, что коллекция выбрана по умолчанию.

В принципе, менять стандартные коллекции нежелательно. Мы рекомендуем оставить их в том виде, в котором они определены в системе Red Hat, и указывать дополнительные пакеты непосредственно в файле **ks.cfg**.

🕮 Дополнительную информацию о PXE см. в разделе "Настройка PXE".

Задание пользовательского конфигурационного файла

После создания конфигурационного файла необходимо заставить утилиту Kickstart использовать его. Это можно сделать несколькими способами. Официальный способ — загрузиться с внешнего носителя (USB и DVD) и запросить инсталляцию Kickstart, введя **linux ks** в начальной строке приглашения `boot:`. Кроме того, можно использовать загрузку PXE.

Если не указать дополнительных аргументов, система определит свой сетевой адрес по протоколу DHCP. Затем она узнает имя загрузочного DHCP-сервера и загрузочного файла, попытается смонтировать серверный каталог через NFS и установит загрузочный файл в качестве конфигурационного. При отсутствии сведений о загрузочном файле система будет искать файл **/kickstart/***IP_адрес_хоста*-**kickstart**.

Другой способ задания конфигурационного файла заключается в указании пути к нему в виде аргумента опции **inst.ks**.[1] Здесь есть несколько возможных вариантов. Команда

```
boot: linux ks=http:сервер:/path
```

заставляет утилиту Kickstart загрузить конфигурационный файл по протоколу HTTP, а не через NFS.

Для того чтобы вообще не использовать загрузочный носитель, вам нужно научиться работать с протоколом PXE. Подробно о нем речь шла в начале этой главы.

Автоматизированная инсталляция систем Debian и Ubuntu



Инсталлятор систем Debian и Ubuntu **debian-installer** рекомендуется использовать в качестве варианта предварительной автоматизированной инсталляции системы Ubuntu. Как и в случае Kickstart в системе Red Hat, файл предварительной конфигурации отвечает на вопросы, задаваемые инсталлятором.

Все интерактивные части инсталлятора Debian используют утилиту **debconf**, чтобы решить, какие вопросы нужно ставить и какие ответы использовать по умолчанию. Предоставив утилите **debconf** базу данных с заранее сформулированными вопросами, вы можете полностью автоматизировать работу инсталлятора. Вы можете или сгенерировать базу данных вручную (она представляет собой обычный текстовый файл), или выполнить интерактивную инсталляцию на тестовой системе, а затем передать свои ответы утилите **debconf** с помощью следующих команд.

```
$ sudo debconf-get-selections --installer > preseed.cfg
$ sudo debconf-get-selections >> preseed.cfg
```

Создайте конфигурационный файл, который будет доступен по сети, и передайте его в ядро во время инсталляции с помощью следующего аргумента ядра.

```
preseed/url=http://хост/путь/к/файл_предварительной_инсталляции
```

Синаксис файла предварительной инсталляции, обычно именуемого **preseed.cfg**, довольно прост и во многом сходен с файлом **ks.cfg** в системе Red Hat. Следующий пример представлен в сокращенном виде.

```
d-i debian-installer/locale string en_US
d-i console-setup/ask_detect boolean false
d-i console-setup/layoutcode string us
d-i netcfg/choose_interface select auto
d-i netcfg/get_hostname string unassigned-hostname
d-i netcfg/get_domain string unassigned-domain
...
d-i partman-auto/disk string /dev/sda
d-i partman-auto/method string lvm
d-i partman-auto/choose_recipe select atomic
```

[1] До появления системы RHEL 7 этот параметр назывался **ks**. В настоящее время используются оба варианта названия, но в будущем от имени **ks** будут постепенно отказываться.

```
...
d-i passwd/user-fullname string Daffy Duck
d-i passwd/username string dduck
d-i passwd/user-password-crypted password $6$/mkq9/$G//i6tN.
    x6670.95lVSM/
d-i user-setup/encrypt-home boolean false
tasksel tasksel/first multiselect ubuntu-desktop
d-i grub-installer/only_debian boolean true
d-i grub-installer/with_other_os boolean true
d-i finish-install/reboot_in_progress note
xserver-xorg xserver-xorg/autodetect_monitor boolean true
...
```

Ряд опций в этом листинге просто запрещают диалоги, которые обычно требуют взаимодействия с пользователем. Например, опция `console-setup/ask_detect` запрещает выбор раскладки клавиатуры.

Такая конфигурация старается идентифицировать сетевой интерфейс, который в действительности подключен к сети (`choose_interface select auto`) и получает сетевую информацию через протокол динамического конфигурирования хоста DHCP. Предполагается, что значения системного имени хоста и домена предоставляются протоколом DHCP и не переопределяются.

При выполнении "предварительной" инсталляции нельзя использовать существующие разделы диска: в этом случае либо задействуется существующее свободное пространство, либо выполняется перераспределение всего диска. Наличие строк `partman*` свидетельствует о том, что для сегментирования дисковой памяти используется пакет `partman-auto`. Если система имеет несколько дисков, то для инсталляции вы должны указать нужный диск. В противном случае (т.е. для единственного диска) используется значение **`/dev/sda`**.

Предлагается несколько "рецептов" сегментирования дисковой памяти:

- вариант `atomic` помещает все системные файлы в один раздел;
- вариант `home` создает отдельный раздел для каталога **`/home`**;
- вариант `multi` создает отдельные разделы для каталогов **`/home`**, **`/usr`**, **`/var`** и **`/tmp`**.

Профили пользователей можно создавать посредством ряда директив `passwd`. Как в случае конфигурации Kickstart, мы настоятельно рекомендуем использование зашифрованных (хешированных) паролей. Файлы предварительной инсталляции часто хранятся на HTTP-серверах и могут быть обнаружены любопытными пользователями. (Безусловно, хешированный пароль постоянно является объектом для грубых силовых атак.)

Опция выбора задачи (`tasksel`) позволяет указать тип подлежащей инсталляции Ubuntu-системы из предложенных вариантов: `standard`, `ubuntu-desktop`, `dns-server`, `lamp-server`, `kubuntu-desktop`, `edubuntu-desktop` и `xubuntu-desktop`.

Представленный выше пример файла предварительной инсталляции взят из документации по инсталляции системы Ubuntu, доступной по адресу: `help.ubuntu.com`. Руководство содержит полную документацию по синтаксису и применению файла предварительной инсталляции.

Несмотря на то что происхождение системы Ubuntu не связано с "родословной" Red Hat, на ее собственный базовый инсталлятор "привита" совместимость с управляющими файлами Kickstart. Кроме того, Ubuntu включает утилиту **`system-config-kickstart`** для создания этих файлов. В инсталляторе Kickstart для системы Ubuntu опускается ряд

важных функций, которые поддерживаются Red Hat-инсталлятором Anaconda (например, LVM и конфигурация брандмауэра). Если у вас нет веской причины для выбора Kickstart, мы все же рекомендуем использовать инсталлятор Debian.

Сетевая загрузка с помощью Cobbler, сервера инициализации Linux с открытым исходным кодом

На сегодняшний день самым простым способом сетевой загрузки является использование сервера Cobbler — проекта, первоначально написанного Майклом де Хааном (Michael DeHaan), плодовитым разработчиком программного обеспечения с открытым исходным кодом. Cobbler улучшает утилиту **kickstart**, устраняя некоторые из ее самых утомительных и повторяющихся административных элементов. Он объединяет все важные функции сетевой загрузки, включая DHCP, DNS и TFTP, и помогает управлять образами операционной системы, используемыми для создания физических и виртуальных машин. Сервер Cobbler включает в себя командные строки и веб-интерфейсы для администрирования.

Шаблоны — это, пожалуй, самая интересная и полезная функция сервера Cobbler. Вам часто понадобятся разные настройки быстрой и предварительной инсталляции для разных профилей хоста. Например, у вас могут быть веб-серверы в двух центрах обработки данных, которые, помимо сетевых параметров, требуют одинаковой конфигурации. Вы можете использовать фрагменты Cobbler для обмена разделами конфигурации между двумя типами хостов.

Фрагмент (snippet) — это всего лишь набор команд оболочки. Например, следующий фрагмент добавляет открытый ключ к авторизованным ключам SSH для пользователя root:

```
mkdir -p --mode=700 /root/.ssh
cat >> /root/.ssh/authorized_keys << EOF
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDKErzVdarNkL4bzAZotSzU/
... Rooy2R6TCzc1Bt/oqUK1R1kuV
EOF
chmod 600 /root/.ssh/authorized_keys
```

Вы сохраняете фрагмент кода в каталоге фрагментов Cobbler, а затем ссылаетесь на него в шаблоне быстрой инсталляции. Например, если вы сохранили фрагмент, показанный выше, как **root_pubkey_snippet**, то можете ссылаться на него в шаблоне следующим образом.

```
%post
SNIPPET::root_pubkey_snippet
$kickstart_done
```

Используйте шаблоны Cobbler для настройки разделов диска, условной установки пакетов, настройки часовых поясов, добавления пользовательских репозиториев пакетов и выполнения любых других требований к локализации.

Cobbler также может создавать новые виртуальные машины под различными гипервизорами. Он может интегрироваться с системой управления конфигурацией для предоставления машин после их загрузки.

Пакеты Cobbler доступны в стандартных хранилищах для наших образцов дистрибутивов Linux. Вы также можете получить пакеты и документацию из проекта Cobbler GitHub на сайте `cobbler.github.io`.

Автоматизированная инсталляция системы FreeBSD

Утилита **bsdinstall** в системе FreeBSD является текстовым инсталлятором, который запускается при загрузке компьютера с установочного компакт-диска или DVD-диска FreeBSD. Ее средства автоматизации рудиментарны по сравнению с быстрой инсталляцией Red Hat или предварительной инсталляцией Debian, а документация ограничена. Лучшим источником информации является справочная страница **bsdinstall**.

Создание настраиваемого, не привлекающего к себе внимание инсталляционного образа — утомительное дело, которое включает в себя следующие этапы.

1. Загрузка последней версии ISO (образа компакт-диска) с сайта **ftp.freebsd.org**.
2. Распаковка образа ISO в локальный каталог.
3. Внесение необходимых изменений в клонированный каталог.
4. Создание нового образа ISO из вашей индивидуальной компоновки и запись его на носители или создание загрузочного образа PXE для сетевой загрузки.

Версия системы FreeBSD в архиве **tar** понимает формат ISO в дополнение ко многим другим форматам, поэтому вы можете просто извлечь файлы образа компакт-диска в пустой каталог. Создайте подкаталог перед извлечением, потому что файл ISO распаковывается в текущий каталог.

```
freebsd$ sudo mkdir FreeBSD
freebsd$ sudo tar xpCf FreeBSD FreeBSD-11.0.iso
```

После того как вы извлекли содержимое образа, вы можете настроить их, чтобы отобразить нужные параметры установки. Например, вы можете добавить собственные DNS-резолверы, отредактировав файл **FreeBSD/etc/resolv.conf**, чтобы включить в него свои собственные серверы имен.

Обычно программа **bsdinstall** требует от пользователей выбора таких параметров, как тип используемого терминала, сопоставление клавиатуры и нужный стиль разбиения диска. Вы можете обойти интерактивные вопросы, поместив файл с именем **installerconfig** в каталог **etc** образа системы.

Формат этого файла описан на странице руководства **bsdinstall**. Он имеет два раздела.

- Преамбула, которая устанавливает определенные параметры установки.
- Сценарий оболочки, который выполняется после завершения установки.

Мы отсылаем вас к странице руководства, вместо того чтобы излагать ее содержание. Среди других настроек она содержит опции для непосредственного подключения к корню ZFS и к другим настраиваемым схемам разбиения.

Как только ваши настройки будут выполнены, вы можете создать новый файл ISO с помощью команды **mkisofs**. Создайте образ PXE или запишите ISO на оптический носитель для автоматической установки.

Проект mfsBSD (`mfsbsd.vx.sk`) представляет собой набор сценариев, которые генерируют PXE-совместимый образ ISO. Базовый образ FreeBSD 11 занимает не менее 47 МиБ. Исходный текст сценариев см. по адресу `github.com/mmatuska/mfsbsd`.

6.2. Управление пакетами

Программное обеспечение систем UNIX и Linux (исходные коды, файлы сборки, документация и шаблоны конфигурации) традиционно распространялись в виде сжатых

архивов (`.tar.gz` или `.tgz`). Для разработчиков это удобно, но для пользователей и администраторов создает сложности. Содержимое этих архивов нужно было компилировать и собирать для каждой системы и каждой версии. Это утомительный и уязвимый для ошибок процесс.

Со временем появились системы управления пакетами, упрощающие и облегчающие работу по управлению программным обеспечением. Пакеты включают все файлы, необходимые для запуска части программного обеспечения, включая предварительно скомпилированные двоичные файлы, информацию о зависимостях и шаблоны конфигурационных файлов, которые могут быть настроены администраторами. Возможно, самое главное, что системы упаковки пытаются сделать процесс установки максимально атомарным. Если во время установки возникает ошибка, пакет может быть отменен или загружен повторно. Новые версии программного обеспечения могут быть установлены с помощью простого обновления пакета.

Установщики пакетов обычно осведомлены о файлах конфигурации и, как правило, не перезаписывают локальные настройки, сделанные системным администратором. Они либо создают резервные копии существующих файлов конфигурации, которые меняют, либо предоставляют примеры файлов конфигурации под другим именем. Если вы обнаружите, что недавно установленный пакет испортил что-то в вашей системе, вы можете теоретически вернуть его обратно, чтобы восстановить свою систему в исходное состояние. Конечно, теория отличается от практики, поэтому не пытайтесь это делать на производственной системе, предварительно не проверив.

Системы управления пакетами определяют модель зависимостей, которая позволяет разработчикам пакетов обеспечивать правильную установку библиотек и инфраструктуры поддержки, от которых зависят их приложения. К сожалению, графики зависимостей иногда несовершенны. В случае неудачи администраторы могут оказаться в сложной ситуации, когда невозможно обновить пакет из-за несовместимости версий между его зависимостями. К счастью, последние версии программного обеспечения для управления пакетами, по-видимому, менее восприимчивы к этому эффекту.

В ходе установки пакеты могут запускать сценарии в разные моменты времени, поэтому они могут делать гораздо больше, чем просто выгружать новые файлы. Пакеты часто добавляют новых пользователей и группы, запускают проверки работоспособности и настраивают параметры в соответствии с окружающей средой.

К сожалению, версии пакетов не всегда соответствуют версиям программного обеспечения, которое они устанавливают. Например, рассмотрим следующий пакет RPM для приложения **`docker-engine`**:

```
$ rpm -qa | grep -i docker
docker-engine-1.13.0-1.el7.centos.x86_64
$ docker version | grep Version
Version: 1.13.1
```

Сам пакет объявляет версию 1.13.0, но двоичные отчеты демона **`docker`** сообщают о версии 1.13.1. В этом случае специалисты, поддерживающие распространение дистрибутивов, учитывают изменения и увеличивают версию младшего пакета. Имейте в виду, что строка версии пакета не обязательно является точным указанием версии программного обеспечения, которая фактически установлена.

Вы можете создавать пакеты для облегчения распространения ваших собственных локализаций или программного обеспечения. Например, вы можете создать пакет, который при инсталляции считывает информацию о локализации для машины (или полу-

чает ее из центральной базы данных) и использует эту информацию для настройки локальных файлов конфигурации.

Вы также можете связывать локальные приложения в пакеты (в комплекте с зависимостями) или создавать пакеты для сторонних приложений, которые обычно не распространяются в формате пакета. Вы можете обновлять свои пакеты и использовать механизм зависимостей для автоматического обновления компьютеров при выпуске новой версии пакета локализации. Мы рекомендуем использовать программу **fpm**, Effing Package Manager, которая является самым простым способом начать сбор пакетов для нескольких платформ. Вы можете найти ее на сайте `github.com/jordansissel/fpm`.

Вы также можете использовать механизм зависимости для создания групп пакетов. Например, вы можете создать пакет, который не устанавливает ничего, но зависит от многих других пакетов. Установка пакета с включенными зависимостями приводит к тому, что все пакеты будут установлены за один шаг.

6.3. Системы управления пакетами для Linux

В системах Linux широко распространены два формата пакетов. В системах Red Hat, CentOS, SUSE и Amazon Linux, а также в большинстве других дистрибутивов применяется диспетчер пакетов RPM (Red Hat Package Manager). В системах Debian и Ubuntu используются пакеты отдельного формата **.deb**. Оба формата функционально идентичны.

Системы упаковки RPM и **.deb** теперь работают в виде двухуровневых средств управления конфигурацией. На нижнем уровне находятся средства, которые инсталлируют, деинсталлируют и запрашивают пакеты: **rpm** для RPM и **dpkg** для **.deb**.

Над этими командами находятся системы, которые знают, как нужно производить поиск пакетов в Интернете, анализировать зависимости между пакетами и модернизировать все пакеты в системе. Система **yum** (Yellowdog Updater, Modified) работает с системой RPM. Система Advanced Package Tool (APT) первоначально была создана для работы с пакетами **.deb**, а сейчас она может работать также с пакетами RPM.

На следующих двух страницах мы рассмотрим команды низкого уровня **rpm** и **dpkg**. В разделе 6.4 мы обсудим комплексные системы обновления APT и **yum**, которые основываются на этих низкоуровневых объектах. Ваши ежедневные действия по администрированию обычно включают инструменты высокого уровня, но вам иногда приходится пробираться в глубокие дебри **rpm** и **dpkg**.

Команда rpm: управление пакетами RPM

RHEL 

Команда **rpm** инсталлирует, проверяет и запрашивает состояние пакетов. Когда-то она еще и создавала пакеты, однако сейчас эта функция отведена команде **rpmbuild**. Тем не менее опции **rpm** по-прежнему имеют сложные взаимодействия и вместе могут использоваться только в некоторых комбинациях. Воспринимать утилиту **rpm** нужно так, будто это несколько разных команд с одним и тем же именем.

Режим, который вы выбираете для работы **rpm** (например, **-i** или **-q**), определяет, к каким функциям этой утилиты вы хотите обратиться. Команда **rpm --help** перечислит все опции, разбивая их по режимам. Если вам часто приходится иметь дело с пакетами RPM, нужно будет внимательно прочесть главную страницу.

Обычно используются опции **-i** (install), **-U** (upgrade), **-e** (erase) и **-q** (query). Последняя опция является довольно сложной в том плане, что она служит для включения остальных опций; чтобы сформировать определенный запрос, вам нужно предоставить дополнительный флаг командной строки. Например, команда **rpm -qa** отображает список всех пакетов, инсталлированных в системе.

Рассмотрим небольшой пример. Предположим, требуется инсталлировать новую версию пакета OpenSSH, поскольку появилось сообщение о том, что в предыдущей версии выявлена брешь. После того как пакет загружен на локальный компьютер, для его инсталляции достаточно ввести команду **rpm -U**.

```
redhat$ sudo rpm -U openssh-6.6.1p1-33.el7_2.x86_64.rpm
error: failed dependencies:
openssh = 6.6.1p1-23 is needed by openssh-clients-6.6.1p1-23
openssh = 6.6.1p1-23 is needed by openssh-server-6.6.1p1-23
```

Гм... Похоже, не все так просто! Как видите, инсталлированная в настоящий момент версия 6.6.1p1-23 связана с рядом других пакетов. Команда **rpm** не позволяет обновить пакет OpenSSH до версии 6.6.1p1-33, так как это изменение затрагивает другие пакеты. Этот тип конфликта встречается постоянно, поэтому приходится разрабатывать такие системы, как APT и **yum**. В реальном мире мы вряд ли пытались бы распутывать зависимости вручную, однако в качестве примера сделаем это для утилиты **rpm**.

Можно прибегнуть к принудительному обновлению с помощью опции **--force**, но это вряд ли оправданно. Информация о зависимостях присутствует здесь для того, чтобы сэкономить ваше время и избавить вас от лишних проблем, а не для того, чтобы запутать вас. Для любого системного администратора нет ничего хуже, чем нарушение работы SSH в дистанционной системе:

```
redhat$ rpm -q --whatrequires openssh
openssh-server-6.6.1p1-23.el7_2.x86_64
openssh-clients-6.6.1p1-23.el7_2.x86_64
```

Теперь предположим, что обновленные копии необходимых пакетов получены. Можно инсталлировать их последовательно, но команда **rpm** все берет на себя. Достаточно указать список пакетов в командной строке, и команда **rpm** отсортирует их в соответствии с имеющимися зависимостями.

```
redhat$ sudo rpm -U openssh-*
...
redhat$ rpm -q openssh
openssh-6.6.1p1-33.el7_3
```

Круто! Похоже, это сработало. Обратите внимание: **rpm** понимает, о каком пакете мы говорим, даже если мы не указали полное имя или версию пакета. (К сожалению, **rpm** не перезапускает оболочку **sshd** после установки. Вам нужно будет вручную перезагрузить ее, чтобы завершить обновление.)

Команда dpkg: управление пакетами .deb



В системе Debian аналогом команды **rpm** является команда **dpkg**. К полезным ее опциям относятся **--install**, **--remove**, а **-l** перечисляет пакеты, инсталлированные в системе. Обратите внимание на то, что команда **dpkg --install**, выполненная в отношении пакета, уже находящегося в системе, перед инсталляцией удаляет предыдущую версию пакета.

С помощью команды **dpkg -l | grep** *пакет* легко определить, инсталлирован ли уже указанный пакет. Например, чтобы отыскать HTTP-сервер, выполните следующую команду.

```
ubuntu$ dpkg -l | grep -i http
ii lighttpd 1.4.35-4+deb8u1 amd64       fast webserver with minimal
    memory footprint
```

В результате будет найдена программа **lighttpd** — облегченный веб-сервер с превосходным открытым исходным кодом. Начальные буквы `ii` означают, что заданная программа уже инсталлирована.

Предположим, что группа, занимающаяся вопросами безопасности системы Ubuntu, выпустила исправление к редактору **nvi**. После загрузки исправления нужно выполнить команду **dpkg** для ее установки. Из показанного ниже примера видно, что эта команда гораздо более многословна, чем **rpm**, и сообщает о том, что именно она делает.

```
ubuntu$ sudo dpkg --install ./nvi_1.81.6-12_amd64.deb
 (Reading database ... 24368 files and directories currently installed.)
 Preparing to replace nvi 1.79-14 (using ./nvi_1.81.6-12_amd64.deb) ...
 Unpacking replacement nvi ...
 Setting up nvi (1.81.6-12) ...
 Checking available versions of ex, updating links in /etc/alternatives ...
 (You may modify the symlinks there yourself if desired - see 'man ln'.)
 Leaving ex (/usr/bin/ex) pointing to /usr/bin/nex.
 Leaving ex.1.gz (/usr/share/man/man1/ex.1.gz) pointing to /usr/share/
    man/man1/nex.1.gz.
 ...
```

Теперь можно ввести команду **dpkg -l**, чтобы узнать, все ли прошло нормально. Флаг **-l** допускает наличие шаблона поиска, благодаря чему можно получить информацию только по редактору **nvi**.

```
ubuntu$ dpkg -l nvi
   Name    Version      Description
ii nvi     1.81.6-12    4.4BSD re-implementation of vi.
```

Инсталляция завершилась успешно.

6.4. Использование высокоуровневых систем управления пакетами в системе Linux

Системы управления пакетами, такие как APT и **yum**, а также Red Hat Network, ставят перед собой следующие задачи:

- упростить определение местонахождения и загрузку пакетов;
- автоматизировать процесс обновления или модернизации систем;
- способствовать управлению зависимостей между пакетами.

Очевидно, что, помимо команд, на стороне клиентов эти системы должны выполнять множество других функций. Все они требуют, чтобы компании, обеспечивающие поддержку дистрибутивов, организовывали свои предложения в согласованном порядке, чтобы клиенты могли иметь осознанный доступ к их программному обеспечению.

Так как ни один поставщик не в состоянии предложить программы для Linux на любой вкус, каждая система допускает существование множества хранилищ программного обеспечения. Хранилища могут быть локальными по отношению к вашей сети, поэтому

эти системы предлагают первоклассную базу, чтобы вы могли создать собственную систему внутреннего распределения.

RHEL Служба Red Hat Network неразрывно связана с дистрибутивом Red Hat Enterprise Linux. Это коммерческая служба, пользование которой стоит денег, зато она обладает привлекательным интерфейсом и более мощными возможностями в плане автоматизации, чем APT и **yum**. Служба Red Hat Network — это улучшенная открытая версия дорогостоящего сервера Satellite Server. Клиент может ссылаться на хранилища **yum** и APT, и эта возможность позволяет таким дистрибутивам, как CentOS, адаптировать клиентский графический интерфейс под нестандартное использование.

Система APT документирована гораздо лучше, чем Red Hat Network, и к тому же бесплатна. Она также является более гибкой в плане настройки. Система APT разрабатывалась для системы Debian и программного обеспечения **dpkg**, но была расширена и теперь может работать с RPM. Доступными являются все версии, работающие со всеми нашими примерами дистрибутивов. На данный момент система APT более других подходит на роль универсального стандарта для распространения программного обеспечения.

Программа **yum** — это аналог APT, предназначенный для RPM. Она также является стандартным администратором пакетов для Red Hat Enterprise Linux и CentOS, хотя может работать в любой системе, основанной на RPM, при условии что вы сможете указать ей хранилища, имеющие соответствующий формат.

Мы предпочитаем APT, и считаем, что это разумный выбор, если нужно настроить собственную автоматизированную сеть распределения пакетов. Более подробно об этом можно прочитать в разделе "Создание локального зеркала хранилища" далее в этой главе.

Хранилища пакетов

Дистрибьюторы Linux поддерживают хранилища программного обеспечения, которые работают совместно с выбранными ими системами управления пакетами. Конфигурация, выбираемая по умолчанию для системы управления пакетами, обычно ссылается на один или несколько хорошо известных веб- или FTP-серверов, находящихся под управлением дистрибьюторов.

Однако из этого факта нельзя сразу понять, что может содержаться в таких хранилищах. Должны ли они включать только наборы пакетов, принятых в качестве официальных, основных, версий? Официальные версии плюс текущие обновления системы защиты? Современные версии всех пакетов, которые существовали в официальных выпусках? Полезные программы сторонних производителей, которые официально не поддерживаются дистрибьютором? Исходный код? Бинарные файлы для многочисленных архитектур аппаратных средств? Если вы запускаете **apt upgrade** или **yum upgrade**, чтобы обновить систему, что именно под этим подразумевается?

Вообще, системы управления пакетами должны отвечать на все эти вопросы и облегчать организациям выбор специфических профилей, которые они хотят включить в их "мир программного обеспечения". Следующие концепции помогут структурировать этот процесс.

- Выпуск (release) — это самодостаточное отображение окружения пакета. Прежде чем наступила эпоха Интернета, именованные выпуски операционных систем были более или менее постоянными и были связаны с одним определенным моментом времени; исправления системы создавались отдельно. В наши дни выпуск пред-

ставляет собой более расплывчатое понятие. Выпуски выходят во время обновления пакетов. Некоторые выпуски, такие как Red Hat Enterprise Linux, предназначены специально для того, чтобы задержать выходы новых версий; по умолчанию в них включаются только обновления защиты. Остальные выпуски, такие как бета-версии, меняются часто и существенно. Однако во всех случаях выпуск является базовой линией, трендом, той мерой, "до которой я хочу обновить свою систему".

- Компонент (component) — подборка программ в рамках выпуска. Дистрибутивы имеют различные отличия, однако общим является отличие между основным программным обеспечением, предлагаемым дистрибьютором, и дополнительным программным обеспечением, предлагаемым энтузиастами. Другое отличие, присущее миру Linux, кроется между свободными частями открытого исходного кода выпуска и частями, связанными с некоторым соглашением о коммерческом использовании.
- Отдельного внимания со стороны администратора заслуживают минимально активные компоненты, которые включают только исправления в системе защиты. Некоторые выпуски позволяют комбинировать компонент защиты с постоянным базовым компонентом для создания относительно стабильной версии дистрибутива.
- Архитектура (architecture) — это специфический класс аппаратных средств (оборудования). Предполагается, что компьютеры, относящиеся к некоторому классу архитектуры, будут иметь одинаковые характеристики, позволяющие запускать на них одинаковые исполняемые файлы. Архитектуры являются специфическими экземплярами выпусков (например, "Ubuntu Karmic Koala for x86_64"). Так как компоненты являются подразделениями выпусков, для каждого из них существует соответствующий экземпляр, характерный для данной архитектуры.
- Индивидуальные пакеты являются элементами, составляющими компоненты, а следовательно, и выпуски. Пакеты обычно являются специфическими для архитектуры и выходят в виде версии, не зависящей от главного выпуска и остальных пакетов. Соответствие между пакетами и выпусками является неявным в том плане, как производится настройка сетевого хранилища.

Существование компонентов, которые не поддерживаются дистрибьютором (например, universe и multiverse для системы Ubuntu), поднимает вопрос о том, как эти компоненты относятся к основному выпуску операционной системы. Можно ли о них говорить как о настоящих компонентах специфического выпуска или же они представляют собой нечто другое?

С точки зрения управления пакетами ответ прост: universe — это настоящий компонент. Они связаны со специфическим выпуском и выходят вместе с ним. Разделение управления интересно с точки зрения администрирования, но не влияет на системы управления пакетами, за исключением ситуаций, в которых администратор должен вручную добавлять несколько хранилищ.

Служба RHN: Red Hat Network

RHEL

После того как система Red Hat перестала входить в группу потребительских товаров Linux, Red Hat Network стала платформой для управления системой для Red Hat Enterprise Linux. Подписываясь на нее, вы приобретаете право доступа к Red Hat Network. В простейшем виде служба Red Hat Network представляет собой популярный веб-портал и список рассылки.

> В этом смысле она мало чем отличается от тех служб уведомления об исправлениях, которые уже много лет эксплуатируются различными поставщиками UNIX-систем. Дополнительные возможности открываются лишь при условии, что вы готовы за них заплатить. Информацию и текущие расценки можно узнать на сайте **rhn.redhat.com**.

Служба Red Hat Network предлагает веб-ориентированный интерфейс для загрузки новых пакетов (можно также работать в режиме командной строки). После того как вы зарегистрируетесь, система будет получать все необходимые исправления в автоматическом режиме.

Недостатком такого подхода является то, что решения об обновлении системы принимаются за вас. Определите для себя, в какой степени следует доверять разработчикам системы Red Hat (а также тех программ, которые они предлагают в виде пакетов).

Разумным компромиссом будет выделение одного из компьютеров в качестве сервера автоматических обновлений. Периодически можно создавать образы серверной системы и проверять, в какой степени они подходят для внутреннего распространения.

APT: усовершенствованное средство управления пакетами

APT (Advanced Packaging Tool) — это одна из зрелых систем управления пакетами. С помощью всего лишь одной команды **apt** можно модернизировать всю систему и программное обеспечение, а также (с помощью Red Hat Network) постоянно поддерживать актуальными ваши версии без вмешательства со стороны человека.

Первое правило, которого следует придерживаться при использовании системы APT в системе Ubuntu (это касается всех средств управления пакетами этой системы), требует игнорировать утилиту **dselect**, являющуюся клиентской надстройкой системы. Речь не идет о том, что утилита **dselect** плохая, просто ее интерфейс неудачен и может напугать новичка. В документации вы можете найти рекомендации в пользу применения **dselect**, тем не менее не принимайте их во внимание и используйте **apt**.

В случае типичной инсталляции Ubuntu, которая загружается с одного из стандартных зеркал, узнать список доступных пакетов можно на веб-сайте **packages.ubuntu.com**. Этот веб-сайт имеет удобную поисковую систему. Если же создается внутренний сервер APT (см. раздел "Создание локального зеркала хранилища" ниже в этой главе), то, конечно, администратор сам знает, какие пакеты доступны, и может сформировать их список.

Дистрибутивы обычно имеют пустые пакеты, которые существуют для того, чтобы формировать списки зависимых пакетов. Утилита **apt** автоматически загружает и инсталлирует зависимые пакеты, благодаря чему можно легко устанавливать или обновлять блоки пакетов. Например, инсталлируя пакет **gnome-desktop-environment**, можно быть уверенным в том, что установлено все необходимое для пользовательского интерфейса GNOME.

Система APT включает в себя набор низкоуровневых команд, таких как **apt-get** и **apt-cache**, которые для большинства целей обернуты командой **apt**. Обертка является более поздним дополнением к системе, поэтому вы по-прежнему будете видеть ссылки на низкоуровневые команды в Интернете и в документации. В первом приближении команды, которые выглядят одинаково, на самом деле являются одной и той же командой. Например, нет никакой разницы между **apt-install** и **apt-get install**.

Если файл **/etc/apt/sources.list** (см. его подробное описание ниже в этой главе) настроен, а имя необходимого пакета известно, то осталось лишь выполнить команду **apt update**, чтобы обновить информацию о пакетах. После этого от имени привилегированного пользователя нужно ввести команду **apt install** *имя_пакета*, которая,

собственно, и осуществит инсталляцию пакета. Эта же команда обновит пакет, если он уже инсталлирован.

Предположим, требуется установить новую версию пакета **sudo**, в котором устранена очередная брешь. Сначала не помешает выполнить команду **apt-get update**.

```
debian$ sudo apt update
Get:1 http://http.us.debian.org stable/main Packages [824kB]
Get:2 http://non-us.debian.org stable/non-US/main Release [102B]
...
```

Теперь можно приступить к получению пакета. Обратите внимание на то, что мы используем команду **sudo** в процессе инсталляции нового пакета **sudo** — утилита **apt** способна обновлять даже те пакеты, которые в настоящий момент используются.

```
debian$ sudo apt install sudo
Reading Package Lists... Done
Building Dependency Tree... Done
1 packages upgraded, 0 newly installed, 0 to remove and 191 not upgraded.
Need to get 0B/122kB of archives. After unpacking 131kB will be used.
(Reading database ... 24359 files and directories currently installed.)
Preparing to replace sudo 1.6.2p2-2 (using .../sudo_1.8.10p3-1+deb8u3_
   amd64.deb) ...
Unpacking replacement sudo ...
Setting up sudo (1.8.10p3-1+deb8u3) ...
Installing new version of config file /etc/pam.d/sudo ...
```

Настройка конфигурации хранилища

Настроить конфигурацию системы APT несложно. Хорошие инструкции можно найти в документации по управлению пакетами в системе Ubuntu по адресу:

`help.ubuntu.com/community/AptGet/Howto`

Самый важный конфигурационный файл утилиты называется **/etc/apt/sources.list**. В нем сообщается, где искать пакеты. В каждой строке файла указывается следующее.

- Тип пакета. В настоящее время это `deb` или `deb-src` для пакетов в стиле Debian либо `rpm` или `rpm-src` — для RPM.
- URL-адрес файла, компакт-диска, сервера HTTP или FTP, где находятся пакеты.
- Дистрибутив (на самом деле — название выпуска), если нужно работать с несколькими версиями пакета. Дистрибьюторы используют его для главных выпусков, однако вы можете использовать его для других целей, если вам нужны внутренние системы распространения.[2]
- Возможный список компонентов, т.е. категорий пакетов в рамках дистрибутива.

Стандартная конфигурация вполне приемлема, если только не нужно создавать собственное хранилище пакетов. Исходные пакеты загружаются при использовании строк, начинающихся с `deb-src`.

Работая в системе Ubuntu, вы почти наверняка захотите включить в нее компонент universe, который предоставляет доступ к открытым программным средствам большого

[2]Дистрибьюторы используют поле *дистрибутив* для определения основных выпусков, но для внутренних систем распространения вы можете использовать его, как сами захотите.

(по объему) мира Linux. Пакеты multiverse включают такие неоткрытые исходные тексты, как некоторые утилиты и компоненты VMware.

В процессе редактирования файла **sources.list** вам следует перенастроить отдельные записи для указания адреса зеркала, задав более близко расположенный сервер. Полный список зеркал Ubuntu находится по адресу: **launchpad.net/ubuntu/+archivemirrors**. Это динамический (и длинный) список зеркал, который регулярно изменяется, поэтому обязательно отслеживайте изменения между выпусками.

Убедитесь, что в качестве источника указан элемент **security.ubuntu.com**, и тогда вы точно получите доступ к самым последним исправлениям, связанным с мерами по укреплению безопасности.

Пример файла /etc/apt/sources.list

В показанном ниже примере в качестве источника пакетов для загрузки основных компонентов Ubuntu используется адрес **us.archive.ubuntu.com** (эти компоненты полностью поддерживаются командой разработчиков Ubuntu). Кроме того, этот список (**sources.list**) включает неподдерживаемые (но с открытым исходным кодом) пакеты universe и небесплатные неподдерживаемые пакеты в компоненте multiverse. В каждом компоненте также предусмотрено хранилище для обновлений, т.е. пакетов с исправленными ошибками. Наконец, последние шесть строк предназначены для обновлений, связанных с безопасностью.

```
# Общий формат: тип URL-адрес дистрибутив [компоненты]
deb http://archive.ubuntu.com/ubuntu xenial main restricted
deb-src http://archive.ubuntu.com/ubuntu xenial main restricted
deb http://archive.ubuntu.com/ubuntu xenial-updates main restricted
deb-src http://archive.ubuntu.com/ubuntu xenial-updates main restricted
deb http://archive.ubuntu.com/ubuntu xenial universe
deb-src http://archive.ubuntu.com/ubuntu xenial universe
deb http://archive.ubuntu.com/ubuntu xenial-updates universe
deb-src http://archive.ubuntu.com/ubuntu xenial-updates universe
deb http://archive.ubuntu.com/ubuntu xenial multiverse
deb-src http://archive.ubuntu.com/ubuntu xenial multiverse
deb http://archive.ubuntu.com/ubuntu xenial-updates multiverse
deb-src http://archive.ubuntu.com/ubuntu xenial-updates multiverse
deb http://archive.ubuntu.com/ubuntu xenial-backports main restricted
   universe multiverse
deb-src http://archive.ubuntu.com/ubuntu xenial-backports main restricted
   universe multiverse
deb http://security.ubuntu.com/ubuntu xenial-security main restricted
deb-src http://security.ubuntu.com/ubuntu xenial-security main restricted
deb http://security.ubuntu.com/ubuntu xenial-security universe
deb-src http://security.ubuntu.com/ubuntu xenial-security universe
deb http://security.ubuntu.com/ubuntu xenial-security multiverse
deb-src http://security.ubuntu.com/ubuntu xenial-security multiverse
```

Поля *дистрибутив* и *компоненты* помогают утилите **apt** ориентироваться в иерархии файловой системы хранилища Ubuntu, которая характеризуется стандартизованным размещением. Корневой дистрибутив для каждого выпуска может быть помечен как `thrusty`, `xenial` или `yaketty`. Доступными компонентами обычно являются `main`, `universe`, `multiverse` и `restricted`. Если вас устраивает наличие неподдерживаемых

(и с ограниченной лицензией в случае multiverse) программ в вашей среде, добавьте хранилища universe и multiverse.

После обновления файла sources.list выполните команду **apt-get update**, чтобы заставить систему APT учесть ваши изменения.

Создание локального зеркала хранилища

Если вы планируете применять утилиту **apt** для большого числа компьютеров, то потребуется локальное кеширование пакетов — загружать копии всех пакетов для каждого компьютера было бы неразумно. Совсем не трудно сконфигурировать зеркало хранилища, которое было бы удобным для локального администрирования. От вас требуется лишь отслеживать обновления с помощью исправлений, связанных с безопасностью.

Лучше всего для такой работы использовать удобный пакет **apt-mirror**, который доступен на сайте apt-mirror.github.io. Вы можете также инсталлировать пакет из компонента universe с помощью команды **sudo apt install apt-mirror**.

После инсталляции пакета **apt-mirror** в каталоге **/etc/apt** вы найдете файл **mirror.list**. Это теневая версия **sources.list**, но она используется только как источник для зеркального отображения операций. По умолчанию **mirror.list** содержит все хранилища для запуска версии Ubuntu.

Для того чтобы в действительности воспроизвести хранилища в **mirror.list**, необходимо запустить пакет **apt-mirror** с правами root.

```
ubuntu$ sudo apt-mirror
Downloading 162 index files using 20 threads...
Begin time: Sun Feb 5 22:34:58 2017
[20]... [19]... [18]... [17]... [16]... [15]... [14]...
```

По умолчанию пакет **apt-mirror** помещает свои копии хранилища в каталог **/var/spool/apt-mirror**. Можно закомментировать строку set base_path в файле **mirror.list**, но не забудьте создать в новом корневом каталоге зеркала подкаталоги **mirror**, **skel** и **var**.

Пакету **apt-mirror** требуется много времени для первого прохода, поскольку ему приходится отображать множество гигабайтов данных (на данный момент приблизительно 40 Гбайт для выпуска Ubuntu). Последующие проходы протекают быстрее и должны быть реализованы автоматически (из демона **cron**). Для того чтобы избавиться от уже устаревших файлов, можете запустить сценарий **clean.sh** из подкаталога **var** своего зеркала.

Для того чтобы приступить к использованию своего зеркала, воспользуйтесь (с помощью своего любимого веб-сервера) базовым каталогом через протокол HTTP. Мы любим использовать символические ссылки на веб-корень. Вот пример.

```
ln -s /var/spool/apt-mirror/us.archive.ubuntu.com/ubuntu /var/www/ubuntu
```

Для того чтобы заставить клиентов использовать свое локальное зеркало, отредактируйте файлы **sources.list** так, как если бы выбрали нелокальное зеркало.

Автоматизация работы системы APT

Утилиту **apt** можно запускать по графику с помощью демона **cron**. Даже если пакеты не инсталлируются автоматически, полезно регулярно выполнять команду **apt update**, чтобы следить за обновлениями сводных файлов.

Команда **apt upgrade** загрузит и инсталлирует новые версии любых пакетов, имеющихся на локальном компьютере. Обратите внимание на то, что команда **apt upgrade** определена немного иначе, чем низкоуровневая команда **apt-get upgrade**. Тем не менее она точно соответствует вашим целям. (Она эквивалентна команде **apt-get dist-upgrade --with-new-pkgs**.) Команда **apt upgrade** может удалить пакеты, которые утилита посчитает несовместимыми с модернизированной системой, так что к этому нужно быть готовым.

Если вы любите риск, то можете даже позволить автоматическое обновление с зеркала дистрибутивов. Для этого предназначен параметр **-y** утилиты **apt-get**, благодаря которому на любой вопрос будет выдано оптимистичное "Да!" Имейте в виду, что некоторые обновления (например, пакеты ядра) могут не вступать в силу до тех пор, пока система не будет перезагружена.

Обычно не рекомендуется автоматически загружать обновления непосредственно с зеркал дистрибутивов. Другое дело, когда имеются внутренние серверы APT и система управления версиями. Следующий маленький сценарий позволит клиенту поддерживать синхронизацию с сервером APT.

```
# apt update && apt upgrade -y
```

Этот сценарий может запускаться демоном **cron** по расписанию. Можно также создать ссылки на него в сценариях запуска системы (см. главу 2), чтобы обновления выполнялись на этапе начальной загрузки.

Если процедуры обновления запускаются на большом числе компьютеров, то с помощью демона **cron** необходимо распределить их по времени, чтобы не перегрузить сеть.

Если вы не вполне доверяете источнику пакетов, можно автоматически загружать их, но не инсталлировать. Это позволяет делать опция **--download-only** утилиты **apt-get**. Просмотрите полученные пакеты и определите сами, какие из них инсталлировать. Загруженные пакеты находятся в каталоге **/var/cache/apt**, который со временем может серьезно разрастись. Удаляйте неиспользуемые файлы командой **apt-get autoclean**.

Система yum: управление выпусками для RPM

Система **yum** (Yellowdog Updater, Modified) представляет собой администратор метапакетов, основанный на RPM. Называть **yum** клоном **apt**, пожалуй, неправильно — в плане тематики и реализации они очень похожи, но на практике **yum** проще и медленнее.

Команда **yum-arch** на стороне сервера компилирует базу данных заголовочной информации из большого набора пакетов (нередко из целого выпуска). После этого база данных заголовков совместно используется пакетами посредством протокола HTTP. Клиенты используют команду **yum** для выбора и инсталляции пакетов; **yum** выявляет ограничения зависимостей и выполняет дополнительные действия, необходимые для завершения процесса инсталляции требуемых пакетов. Если запрошенный пакет зависит от других пакетов, **yum** загружает и инсталлирует и эти пакеты.

Сходство между **apt** и **yum** распространяется на опции командной строки, которые понятны им обоим. Например, **yum install foo** загружает и инсталлирует самую свежую версию пакета **foo** (и его зависимости, если это необходимо). Однако существует как минимум одно "предательское" отличие: **apt update** обновляет кеш информации о пакетах **apt**, а **yum update** — каждый пакет в системе (аналогично команде **apt-get update**). Более того, есть еще команда **yum upgrade**, которая делает то же, что и **yum update**, но устаревшими приемами.

Команда **yum** не рассматривает частичные имена пакетов, если не включить символы универсализации оболочки (такие, как * и ?). Например, **yum update 'lib*'** обновляет все пакеты, имена которых начинаются с "**lib**". Не забывайте заключать символы универсализации в кавычки, чтобы избежать возникновения ошибок.

В отличие от **apt**, программа **yum** во время запуска по умолчанию сверяет информацию о пакетах, хранящуюся в кеше, с содержимым сетевого хранилища. Для того чтобы отменить этот процесс, используйте опцию **-C**, в результате чего **yum makecache** будет обновлять локальный кеш (на это уйдет некоторое время). К сожалению, опции **-C** недостаточно, чтобы повысить производительность медлительной **yum**.

Конфигурационным файлом **yum** является **/etc/yum.conf**. Он включает общие опции и указатели на хранилища пакетов. Можно активизировать одновременно множество хранилищ; каждое хранилище может быть связано с множеством URL-адресов.

Замена для **yum** под названием DNF (для Dandified Yum) находится в активной разработке. Эта программа уже используется как менеджер пакетов по умолчанию для системы Fedora и в конечном итоге полностью заменит **yum**. Система DNF поддерживает более высокое разрешение зависимостей и, помимо всего прочего, улучшенный интерфейс прикладного программирования. Посетите сайт **dnf.baseurl.org**, чтобы узнать больше.

6.5. Управление программным обеспечением в системе FreeBSD



У системы FreeBSD есть возможность управления пакетами для нескольких выпусков, но она еще только переходит на полностью пакетно-ориентированную модель дистрибуции, в которой большинство элементов основной операционной системы определяются как пакеты. В последних выпусках системы FreeBSD имеется программное обеспечение, разделенное по трем основным категориям.

- Базовая система, которая включает в себя набор основных программных средств и утилит.
- Набор бинарных пакетов, управляемых командой **pkg**.
- Отдельная система портов, которая загружает исходный код, применяет исправления FreeBSD, затем собирает и устанавливает ее.

Начиная с версии FreeBSD 11, границы между этими территориями стали еще более запутанными. Базовая система уже упакована, но старая схема управления базовой системой как единым целым все еще существует. Многие программные пакеты могут быть установлены либо как двоичные пакеты, либо как порты с аналогичными результатами, но с разными последствиями для будущих обновлений. Однако взаимная совместимость еще не обеспечена; некоторые вещи могут устанавливаться только как порт или как пакет.

Одна из целей версии FreeBSD 12 заключается в том, чтобы более решительно переключить систему на универсальное управление пакетами. Базовая система и порты могут продолжать существовать в той или иной форме (в настоящее время слишком рано говорить точно, как все будет работать), но будущее направление ясно.

Соответственно, попробуйте, по возможности, управлять дополнительным программным обеспечением с помощью **pkg**. Избегайте портов, если требуемое программное обеспечение не имеет упакованной версии или вам нужно настроить параметры времени компиляции.

Еще одним особенным событием "железного века" UNIX является требование FreeBSD о том, чтобы добавляемые пакеты были локальными, хотя они компилируются системой FreeBSD и выпускаются как часть официального хранилища пакетов. Пакеты устанавливают двоичные файлы в каталог **/usr/local**, а большинство файлов конфигурации хранятся в каталоге **/usr/local/etc**, а не **/etc**.

Базовая система

Базовая система обновляется как единое целое и функционально отличается от любых дополнительных пакетов (по крайней мере теоретически). Базовая система поддерживается в хранилище Subversion. Вы можете просмотреть исходное дерево, включая все ветви источника, на сайте `svnweb.freebsd.org`.

Определены несколько ветвей разработки.

- Ветвь `CURRENT` предназначена только для активных целей развития. Она первой получает новые функции и исправления, но не получила широкого распространения от сообщества пользователей.
- На ветви `STABLE` регулярно обновляются улучшения, предназначенные для следующего крупного выпуска. Она содержит новые функции, но поддерживает совместимость с пакетом и проходит тестирование. Оно может содержать ошибки или повреждать изменения и рекомендуется, только если вы не боитесь риска.
- Ветвь `RELEASE` разворачивается из ветки `STABLE`, когда достигается цель выпуска. Она остается в основном статичной. Единственными обновлениями для ветки `RELEASE` являются исправления слабостей системы безопасности и исправления серьезных ошибок. Официальные образы ISO находятся на ветки `RELEASE`, и она является единственной, рекомендованной для использования в производственных системах.

Просмотрите текущую ветвь вашей системы с помощью команды **uname -r**.

```
$ uname -r
11.0-RELEASE
```

Выполните команду **freebsd-update**, чтобы ваша система обновлялась с последними пакетами. Получение обновлений и их установка являются отдельными операциями, но вы можете объединить их в одну командную строку:

```
$ sudo freebsd-update fetch install
```

Эта команда извлекает и устанавливает последние базовые двоичные файлы. Она доступна только для ветки `RELEASE`; двоичные файлы для ветвей `STABLE` и `CURRENT` не создаются. Этот же инструмент можно использовать для обновления версий системы. Например:

```
$ sudo freebsd-update -r 11.1-RELEASE upgrade
```

Менеджер пакетов pkg в системе FreeBSD

Программа **pkg** является интуитивно понятной и быстродействующей. Это самый простой способ установить программное обеспечение, которое еще не включено в базовую систему. Используйте команду **pkg help** для быстрой ссылки на доступные подкоманды или команду **help pkg** для отображения справочной страницы для определенной подкоманды.

В табл. 6.2 перечислены некоторые из наиболее часто используемых подкоманд.

Таблица 6.2. Примеры подкоманд pkg

Команда	Что это делает
pkg install -y *пакет*	Выполняет инсталляцию без вопросов "Вы уверены?"
pkg backup	Делает резервную копию базы локальных пакетов
pkg info	Перечисляет все установленные пакеты
pkg info *пакет*	Показывает расширенную информацию о пакете
pkg search -i *пакет*	Выполняет поиск хранилища, в котором находится пакет (без учета регистра)
pkg audit -F	Показывает пакеты с известными слабостями системы безопасности
pkg which *файл*	Показывает, какому пакету принадлежит названный файл
pkg autoremove	Удаляет неиспользуемые пакеты
pkg delete *пакет*	Удаляет пакет (тот же, что и **remove**)
pkg clean -ay	Удаляет кешированные пакеты из **/var/cache/pkg**
pkg update	Обновляет локальную копию каталога пакетов
pkg upgrade	Обновляет пакеты до последней версии

Когда вы устанавливаете пакеты с помощью команды **pkg install**, программа **pkg** проверяет локальный каталог пакетов, а затем загружает запрошенный пакет из хранилища на сайте `pkg.FreeBSD.org`. Как только пакет установлен, он регистрируется в базе данных SQLite, хранящейся в файле **/var/db/pkg/local.sqlite**.

Старайтесь не удалять этот файл, чтобы система не потеряла информацию о том, какие пакеты были установлены. Создавайте резервные копии базы данных с помощью подкоманды **pkg backup**.

Подкоманда **pkg version**, предназначенная для сравнения версий пакета, имеет своеобразный синтаксис. Он использует символы **=**, **<** и **>**, чтобы показывать пакеты, которые являются текущими, старше последней доступной версии или новее, чем текущая версия. Используйте следующую команду для отображения пакетов с обновлениями:

```
freebsd$ pkg version -vIL=
dri-11.2.2,2       <   needs updating (index has 13.0.4,2)
gbm-11.2.2         <   needs updating (index has 13.0.4)
harfbuzz-1.4.1     <   needs updating (index has 1.4.2)
libEGL-11.2.2      <   needs updating (index has 13.0.4_1)
```

Эта команда сравнивает все установленные пакеты с индексом (**-I**) в поисках тех пакетов, которые не являются (-L) текущей версией (**=**), и выводит на экран подробные сведения (**-v**).

Программа **pkg** выполняет поиск пакетов быстрее, чем Google. Например, команда **pkg search dns** находит все пакеты, содержащие в имени слово **dns**. Поисковый термин является регулярным выражением, поэтому вы можете искать что-то вроде **pkg search ^apache**. Более подробная информация содержится в справке, которую выдает команда **pkg help search**.

Коллекция портов

Порты FreeBSD представляют собой коллекцию всего программного обеспечения, которое FreeBSD может создать на основе исходных кодов. После инициализации дерева портов вы найдете все доступное программное обеспечение в категоризированных подкаталогах **/usr/ports**. Чтобы инициализировать дерево портов, используйте утилиту **portsnap**:

```
freebsd$ portsnap fetch extract
```

Чтобы обновить дерево портов в одной команде, используйте команду **portsnap fetch update**.

Загрузка метаданных портов занимает некоторое время. Загрузка включает указатели на исходный код для всех портов, а также любые связанные с ними исправления для совместимости с FreeBSD. Когда установка метаданных будет завершена, вы можете искать программное обеспечение, а затем создавать и устанавливать все, что вам нужно.

Например, оболочка **zsh** не включена в базу FreeBSD. Используйте утилиту **whereis** для поиска **zsh**, затем выполните сборку и установку из дерева портов:

```
freebsd$ whereis zsh
bash: /usr/ports/shells/zsh
freebsd$ cd /usr/ports/shells/zsh
freebsd$ make install clean
```

Чтобы удалить программное обеспечение, установленное через систему портов, запустите команду **make deinstall** из соответствующего каталога.

Существует несколько способов обновления портов, но мы предпочитаем утилиту **portmaster**. Сначала установите **portmaster** из коллекции портов:

```
freebsd$ cd /usr/ports/ports-mgmt/portmaster
freebsd$ make install clean
```

Выполните команду **portmaster -L**, чтобы увидеть все порты, имеющие доступные обновления, и сразу обновите их с помощью команды **portmaster -a**.

Вы также можете устанавливать порты через **portmaster**. Фактически это несколько более удобно, чем типичный процесс на основе команды **make**, потому что вам не нужно покидать свой текущий каталог. Чтобы установить программу **zsh**, выполните следующую команду:

```
freebsd$ portmaster shells/zsh
```

Если вам нужно освободить место на диске, очистите рабочие каталоги портов с помощью **portmaster -c**.

6.6. Локализация и настройка конфигурации программного обеспечения

Адаптация компьютеров к вашей локальной (или облачной) среде является одним из основных заданий системного администрирования. Для решения этих задач нужен структурированный и воспроизводимый способ.

В этой книге мы вернемся к данной теме в главах 23 и 26. Системы управления конфигурацией — это ваши инструменты для установки и настройки программного обеспечения воспроизводимым образом. Они являются основным ключом к правильной локализации.

Если отложить проблемы с реализацией в сторону, откуда вы знаете, правильно ли настроена ваша локальная среда? Перечислим несколько моментов, которые следует учитывать.

- У пользователей нет привилегий системного администратора. Необходимость в них в ходе выполнения обычных операций вызывает подозрение и наверняка свидетельствует о том, что кто-то имеет злой умысел.
- У пользователей нет намерений причинить вред системе. Вы должны организовать защиту системы таким образом, чтобы она была защищена от неумышлен-

ных ошибок и не передавала первому встречному привилегии системного администратора.

- Прежде чем наказывать пользователей, изредка делающих что-то не так, нужно провести с ними беседу. На неумелые действия администратора пользователи зачастую реагируют неадекватно. Следовательно, если ваши действия мешают пользователям работать, то это свидетельствует о проблемах в архитектуре.
- Старайтесь ориентироваться на заказчика. Беседуйте с пользователями и постарайтесь узнать, какие задачи для них являются сложными. Попытайтесь сделать эти задачи гораздо проще.
- Ваши предпочтения — только ваши, и ничьи более. Пользователи должны иметь собственные предпочтения. По мере возможности такие варианты нужно предлагать всегда и везде.
- Если ваши решения, принимаемые в административных целях, влияют на способность пользователей обучаться системе, постарайтесь их изменить. Расскажите о них пользователям.
- Постоянно обновляйте вашу локальную документацию и делайте ее легкодоступной. Более подробно об этом можно прочитать в разделе 31.2.

Организация локализации

Если в компании тысяча компьютеров с различными конфигурациями, администратор вынужден тратить большую часть времени на выяснение причин того, почему такая-то проблема возникла только на этом компьютере. Думаете, решением проблемы будет унификация всех конфигураций? Не спешите с выводами! Ограничения, существующие в реальном мире, а также различные нужды ваших пользователей делают это невозможным.

Между администрированием *многочисленных* и *бесчисленных* конфигураций есть разница. Важно разделить вашу установку на управляемые части. Вы увидите, что одни части локализации применяются ко всем управляемым компьютерам, другие — только к некоторым из них, а третьи — только к отдельным компьютерам.

В процессе проектирования системы локализации следует убедиться в том, что все исходные данные хранятся в системе управления изменениями. Так вы сможете всегда знать, какие изменения были тщательно проверены и готовы к развертыванию. Кроме того, это позволит вам идентифицировать пользователя, являющегося автором проблематичных изменений. Чем больше человек участвуют в этом процессе, тем более важным является последнее утверждение.

Структурные изменения

Помимо выполнения инсталляций с чистого листа, вам придется также постоянно загружать и инсталлировать обновления. Следует иметь в виду, что на разных компьютерах установлены различные сроки действия и предъявляются разные требования к стабильности и периоду работоспособности.

Не стоит развертывать новые выпуски программного обеспечения на всех компьютерах сразу. Наоборот, нужно делать это с учетом нужд каждой группы и потратить некоторое время на выявление проблем на самых ранних этапах, чтобы не допустить серьезного повреждения системы. Важные серверы нельзя обновлять до тех пор, пока не будет уверенности в надежности производимых изменений, и избегайте пятниц, если вы не готовы проводить долгие выходные перед монитором.

Свои преимущества есть и у разделения базового выпуска операционной системы и выпуска локализации. В зависимости от того, какую степень стабильности необходимо достичь в вашей среде, вы можете использовать второстепенные локальные выпуски только для того, чтобы устранить ошибки. Наш опыт подсказывает, что добавление новых функций небольшими порциями приводит к более стабильному выполнению операций, чем при постановке изменений в очередь на следующие крупные выпуски, что сразу же снизит качество обслуживания. Этот принцип тесно связан с идеей непрерывной интеграции и развертывания (см. главу 26).

Ограничение количества выпусков

Часто имеет смысл ограничивать максимальное количество выпусков, с которыми вы хотите работать в любой момент времени. Некоторые администраторы придерживаются мнения, что если программное обеспечение работает надежно, то ничего изменять в нем не нужно. Они полагают, что не имеющая под собой почвы модернизация систем стоит времени и денег и что новейший выпуск слишком часто содержит много ошибок. Те же, кто вооружается этими принципами на практике, должны быть готовы к тому, что активные выпуски придется коллекционировать во вместительном каталоге.

В отличие от них, администраторы, следящие за новинками, утверждают, что в случайной коллекции выпусков, вышедших много лет тому назад, слишком трудно разбираться, а управлять ею — тем более. Их козырем являются исправления защиты, которые должны обычно применяться универсальным образом и в строгом порядке. Инсталлировать исправления в устаревших версиях операционной системы часто бывает просто невозможно, поэтому администраторы сталкиваются с выбором: не инсталлировать обновления на некоторые компьютеры или полностью переходить на новые внутренние выпуски. Это не сулит ничего хорошего.

Ни одна из этих перспектив не является доказуемо верной, хотя мы стараемся придерживаться мнения, что лучше использовать ограниченное количество выпусков. Лучше выполнять модернизацию по вашему собственному плану, чем руководствоваться чьим-то планом.

Тестирование

Важно протестировать изменения, пока их еще можно отменить. По меньшей мере, это означает тестирование собственных настроек. Но в той же степени это касается и системного программного обеспечения. Один известный поставщик UNIX однажды выпустил исправление, которое, будучи примененным определенным образом, выполняло команду `rm -rf /`. Представьте последствия установки такого исправления во всей организации без предварительного тестирования.

Тестирование особенно полезно, когда применяются средства, способные автоматически устанавливать исправления (как большинство систем управления пакетами, рассмотренных в этой главе). Никогда не подключайте компьютеры, ответственные за выполнение ключевых задач, к службе обновления, спонсируемой поставщиком. Постарайтесь подключить к этой службе простой компьютер и уже с него распространяйте изменения на остальные компьютеры в вашей организации.

🕮 Дополнительную информацию о выявлении проблем см. в разделе 31.2.

Если вы предвидите, что заранее запланированное обновление или изменение может вызвать у пользователей некоторые проблемы, сообщите им о ваших планах и позвольте

им связаться с вами, если у них возникнут какие-либо трудности. Позаботьтесь о том, чтобы пользователи имели возможность оперативно сообщать о замеченных ошибках.

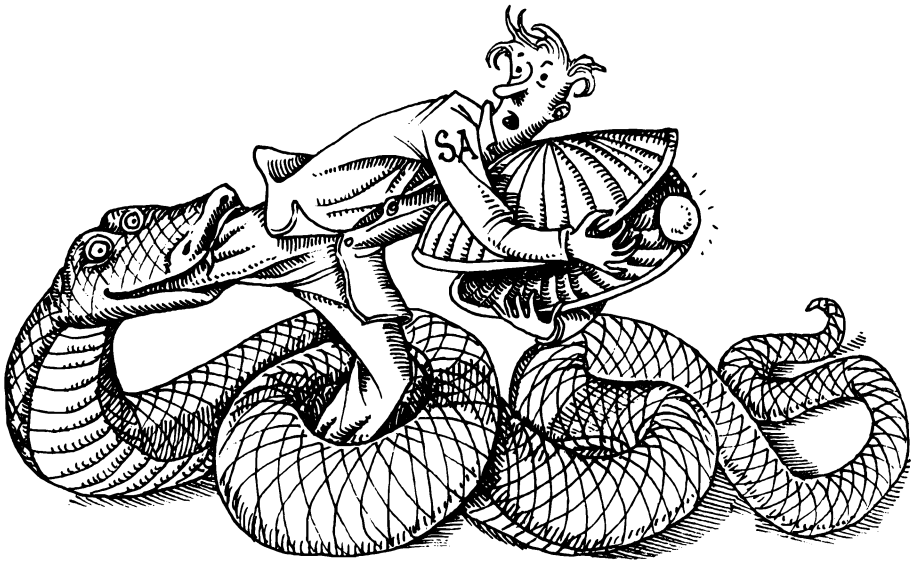
В транснациональной компании можно подключить к тестированию иностранные офисы. Участие пользователей из других стран особенно важно в случае многоязыковых систем. Если никто в американском офисе не говорит по-японски, необходимо связаться с офисом в Токио и попросить их проверить все, что касается поддержки раскладки кандзи (kanji). Вообще, на удивление, большое число системных параметров зависит от региональных особенностей. К примеру, при тестировании программного обеспечения целесообразно проверять, установлена ли новая версия кодировки UTF-8, позволяющая правильно отображать символы разных языков.

6.7. Литература

- Intel Corporation and SystemSoft. *Preboot Execution Environment (PXE) Specification, v2.1.* 1999. `pix.net/software/pxeboot/archive/pxespec.pdf`.
- Lawson, Nolan. *What it feels like to be an open-source maintainer.* `wp.me/p1t8Ca-1ry`.
- *PXELinux Questions.* `syslinux.zytor.com/wiki/index.php/PXELINUX`.
- Rodin, Josip. *Debian New Maintainers' Guide.* `debian.org/doc/maint-guide`. В этом документе содержится полезная информация о пакетах **.deb**. См. также главы 2 и 7 справочника по системе Debian.

Глава 7

Сценарии и командная оболочка



Масштабируемый подход к управлению системой требует, чтобы административные изменения структурировались, воспроизводились и реплицировались на нескольких компьютерах. В реальном мире это означает, что эти изменения должны быть подкреплены программным обеспечением, а не выполняться администраторами, работающими по контрольным спискам, или, что еще хуже, по памяти.

Сценарии стандартизируют административные обязанности и освобождают время администраторов для более важных и интересных задач. Сценарии также служат своего рода упрощенной документацией, поскольку они записывают шаги, необходимые для выполнения конкретной задачи.

Основной альтернативой сценариям системных администраторов является использование систем управления конфигурациями, описанных в главе 23. Эти системы предлагают структурированный подход к администрированию, который хорошо масштабируется для облака и сетей машин. Однако они более сложны, более формальны и менее гибки, чем простые обычные сценарии. На практике большинство администраторов используют комбинацию управления сценариями и конфигурацией. Каждый подход имеет свои сильные стороны, и они хорошо работают вместе.

В этой главе мы кратко рассмотрим языки сценариев **sh**, Python и Ruby. Мы дадим несколько важных советов по использованию оболочки, а также обсудим регулярные выражения как общую технологию.

7.1. Основы сценариев

В этой главе рассматриваются различные варианты написания сценариев и языковые особенности. Эта информация полезна, но более важным, чем любая из этих деталей, является вопрос о том, как включить сценарии (или, иначе говоря, автоматизацию) в вашу ментальную модель системного администрирования.

Создание микросценариев

Начинающие системные администраторы часто откладывают изучение сценариев, пока не столкнутся с особенно сложной или утомительной работой. Например, необходимо автоматизировать резервное копирование определенного типа, чтобы оно выполнялось регулярно и чтобы данные резервного копирования хранились в двух разных центрах обработки данных. Или, возможно, есть конфигурация облачного сервера, которая была бы полезной для создания, инициализации и развертывания с помощью одной команды.

Подобные разумные сценарии могут оставить впечатление, что сценарий — это секретное оружие, которое должно приводиться в готовность только тогда, когда на горизонте возникает большая задача. Допустим, что создание и отладка вашего первого 100-строчного сценария заняли несколько дней. Вы же не можете тратить целые дни на каждую маленькую задачу... Не так ли?

На самом деле вы достигнете большей эффективности, сэкономив несколько нажатий клавиш здесь и несколько команд там. Сценарии типа Marquee, которые являются частью формальных процедур вашего сайта, — это лишь видимая часть айсберга. Ниже ватерлинии лежат многие более мелкие формы автоматизации, которые одинаково полезны для системных администраторов. Как правило, подходите к каждой задаче с вопросом "Как справиться с этой проблемой, если она возникнет снова?"

Большинство администраторов сохраняют набор коротких сценариев для личного использования (вроде скриплетов) в своих каталогах `~/bin`. Используйте эти наспех сделанные сценарии, чтобы устранить проблемы, с которыми вы сталкиваетесь в повседневной работе. Они обычно достаточно короткие, чтобы можно было их понять с первого взгляда, поэтому им не нужна документация, кроме простой инструкции об использовании. Обновляйте их по мере изменения ваших потребностей.

Для сценариев оболочки у вас также есть возможность определять функции, которые находятся внутри конфигурационных файлов (например, `.bash_profile`), а не в автономных файлах сценариев. Функции оболочки работают аналогично автономным сценариям, но не зависят от вашего пути поиска и автоматически перемещаются вместе с вами везде, куда простирается среда оболочки.

В качестве иллюстрации рассмотрим простую функцию оболочки Bash, выполняющую резервное копирование файлов в соответствии со стандартизованным соглашением об именах.

```
function backup () {
   newname=$1.`date +%Y-%m-%d.%H%M.bak`;
   mv $1 $newname;
   echo "Backed up $1 to $newname.";
   cp -p $newname $1;
}
```

Несмотря на функциональный синтаксис, этот код используется как сценарий или любая другая команда.

```
$ backup afile
Backed up afile to afile.2017-02-05.1454.bak.
```

Основным недостатком функций оболочки является то, что они хранятся в памяти и каждый раз при запуске новой оболочки интерпретироваться должны заново. Но на современном оборудовании эти затраты незначительны.

В меньшем масштабе можно использовать псевдонимы, которые на самом деле являются очень короткой разновидностью скриплета. Их можно определить либо с помощью функций оболочки, либо с помощью встроенной функции вашей оболочки (обычно называемой `alias`). Чаще всего они устанавливают аргументы по умолчанию для отдельных команд. Например,

```
alias ls = 'ls -Fh'
```

заставляет команду **`ls`** выделять имена каталогов и исполняемых файлов и выводить понятные для человека размеры файлов для длинных листингов (например, `2.4M`).

Хорошо изучите несколько инструментов

Системные администраторы сталкиваются с большим количеством программного обеспечения. Они не могут быть экспертами во всем, поэтому, как правило, собирают документацию, экспериментируют и поверхностно изучают новые пакеты программного обеспечения для настройки в локальной среде. Лень — это добродетель.

Тем не менее некоторые темы заслуживают детального изучения, поскольку они повышают вашу силу и эффективность. В частности, вы должны хорошо знать оболочку, текстовый редактор и язык сценариев.[1] Прочитайте руководства от корки до корки, а затем регулярно читайте книги и блоги. Всегда есть чему поучиться.

Использование технологий, подобных этим, требует предварительного изучения по нескольким причинам. В качестве инструментов они довольно абстрактны; трудно представить все, что они могут сделать, не читая подробности. Вы не можете использовать функции, о которых не знаете.

Другая причина, по которой эти инструменты вознаграждают затраты на обучение, заключается в том, что они практичны; большинство функций потенциально ценны для большинства администраторов. Сравните это со средним демоном сервера, где ваша основная задача часто заключается в том, чтобы определить 80% функций, которые не имеют отношения к вашей ситуации.

Оболочка или редактор — это инструмент, который вы используете постоянно. Постепенное улучшение вашего мастерства с помощью этих инструментов приводит не только к повышению производительности, но и к большему удовольствию от работы. Никто не любит тратить время на повторяющиеся детали.

Автоматизируйте все, что возможно

Сценарии оболочки не являются единственной возможностью системных администраторов использовать автоматизацию. Существует целый мир программируемых систем — просто следите за ними. Активно эксплуатируйте эти объекты и используйте их для повышения производительности труда — совместите инструменты со своим рабочим процессом.

---

[1] Забегая вперед, укажем, что это, вероятно, должны быть Bash, **`vim`** и Python.

Например, авторы английского издания создали эту книгу в среде Adobe InDesign, которая якобы является графическим приложением. Однако она также допускает использование сценариев, написанных на языке JavaScript, поэтому мы создали библиотеку сценариев InDesign для реализации и обеспечения соблюдения многих наших соглашений.

Такие возможности существуют повсюду.

- Приложения Microsoft Office программируются на Visual Basic или C#. Если ваша работа включает анализ или отчетность, делайте отчеты TPS (Testing Procedure Specification) самостоятельно.
- Большинство приложений Adobe допускают использование сценариев.
- Если в ваши обязанности входит работа с базами данных, вы можете автоматизировать многие обычные задачи с помощью хранимых процедур SQL. Некоторые базы данных поддерживают дополнительные языки; например, база PostgreSQL поддерживает язык Python.
- PowerShell — это основной инструмент сценариев для систем Microsoft Windows. Сторонние дополнения, такие как AutoHotKey, значительно облегчают автоматизацию приложений Windows.
- В системах macOS некоторые приложения могут управляться с помощью языка сценариев AppleScript. На системном уровне используйте приложение "Automator", "Службы" и действия с папками, чтобы автоматизировать различные задачи и подключить языки традиционного языка к графическому интерфейсу.

В мире системного администрирования некоторые подсистемы имеют собственные подходы к автоматизации. Многие другие подсистемы хорошо сочетаются с системами автоматизации общего назначения, такими как Ansible, Salt, Chef и Puppet, описанными в главе 23. Для всего остального существуют сценарии общего назначения.

Избегайте преждевременной оптимизации

Нет никакого реального различия между сценарием и программой. Разработчики языка иногда обижаются, когда их детища попадают в категорию сценариев, и не только потому, что ярлык указывает на определенную нехватку полноты, но также потому, что некоторые языки сценариев в прошлом заслужили репутацию плохо организованных.

Впрочем, нам все еще нравится термин *программирование сценариев* (scripting); он означает использование программного обеспечения в качестве своего рода универсального клея, связывающего различные команды, библиотеки и файлы конфигурации в одно функциональное целое.

Административные сценарии должны подчеркивать эффективность программы и ясность кода, а не вычислительную эффективность. Это не повод быть неаккуратным, подтверждение того, что сценарии редко выполняются за одну-две секунды. Оптимизация может иметь удивительно низкую отдачу даже для сценариев, которые регулярно запускаются из демона `cron`.

Выберите правильный язык сценариев

В течение долгого времени стандартным языком для административных сценариев был сценарий, определенный оболочкой `sh`. Сценарии оболочки обычно используются для легких задач, таких как автоматизация последовательности команд или сборка нескольких фильтров для обработки данных.

Оболочка всегда доступна, поэтому сценарии оболочки относительно переносимы и имеют несколько зависимостей, отличных от команд, которые они вызывают. Независимо от того, выбираете ли вы оболочку, оболочка может выбрать вас: большинство сред включают в себя огромный набор существующих сценариев **sh**, а администраторы часто обязаны читать, понимать и настраивать эти сценарии.

Как язык программирования **sh** является несколько неэлегантным. Синтаксис является специфическим, а оболочке не хватает расширенных возможностей текстовой обработки современных языков — функции, которые часто особенно полезны для системных администраторов.

Язык Perl, разработанный в конце 1980-х годов, стал важным шагом вперед для разработки административных сценариев. Либеральный синтаксис, обширная библиотека пользовательских модулей и встроенная поддержка регулярных выражений сделали его фаворитом системных администраторов на протяжении многих лет. Язык Perl разрешает (а некоторые говорят, поощряет) определенный стиль кодирования "полный вперед — к черту торпеды"[2]. Существуют разные мнения по поводу того, является это преимуществом или недостатком.

В наши дни язык Perl известен как Perl 5. Эту версию следует отличать от переработанной и несовместимой версии Perl 6, которая наконец была выпущена после 15 лет подготовительных работ. К сожалению, Perl 5 немного устарел по сравнению с более новыми языками, а использование Perl 6 пока еще недостаточно распространено для того, чтобы мы рекомендовали его в качестве безопасного выбора. Возможно, мир полностью отказался от языка Perl. На данном этапе мы предлагаем избегать языка Perl, если вы начинаете новый проект.

Языки JavaScript и PHP лучше всего известны как языки для веб-разработки, но их можно использовать в качестве инструментов для общего использования. К сожалению, оба языка имеют недостатки дизайна, которые ограничивают их привлекательность, и им не хватает многих сторонних библиотек, на которые полагаются системные администраторы.

Если вы пришли из мира веб-разработки, у вас может возникнуть соблазн применить существующие навыки PHP или JavaScript для системного администрирования. Мы рекомендуем не делать этого. Код есть код, но проживание с другими системными администраторами в одной экосистеме приносит множество долгосрочных преимуществ. (По крайней мере, избегая PHP, вам не придется терпеть насмешки, участвуя в вики-конференциях локальных системных администраторов.)

Python и Ruby — современные, универсальные языки программирования, которые хорошо подходят для административной работы. Эти языки являются результатом нескольких десятилетий развития языкового дизайна по сравнению с оболочкой, а их средства обработки текста настолько сильны, что **sh** может тихо курить в сторонке.

Основной недостаток как Python, так и Ruby заключается в том, что их среды могут быть немного неудобными для настройки, особенно когда вы начинаете использовать сторонние библиотеки, которые скомпилировали компоненты, написанные на языке C. Оболочка обходит эту проблему, отказавшись от модульной структуры и сторонних библиотек.

В отсутствие внешних ограничений Python является наиболее широко используемым языком сценариев для системных администраторов. Он тщательно разработан, широко используется и поддерживается другими пакетами. В табл. 7.1 приведены некоторые общие примечания о других языках.

[2]Цитата героя Гражданской войны в США адмирала Дэвида Фаррагута (David Farragut). — *Примеч. ред.*

Таблица 7.1. Шифрование на языке сценариев

Language	Designer	Описание
Bourne shell	Стефан Борн (Stephen Bourne)	Простая серия команд, переносимые сценарии
bash	Брайан Фокс (Brian Fox)	Похожа на оболочку, написанную Борном, но более приятная, хотя и менее переносимая
C shell	Билл Джой (Bill Joy)	Никогда не используйте для создания сценариев; см. сноску в разделе 7.2
JavaScript	Брендан Эйх (Brendan Eich)	Веб-разработка, сценарий приложений
Perl	Ларри Уолл (Larry Wall)	Быстрые и однострочные сценарии, обработка текста
PHP	Расмус Лендорф (Rasmus Lerdorf)	Вы плохо себя вели и заслуживаете наказания
Python	Гвидо ван Расмуссен (Guido van Rossum)	Сценарии общего назначения, обработка данных
Ruby	"Мац" Мацумото ("Matz" Matsumoto)	Универсальные сценарии, веб

Следуйте рекомендациям

Несмотря на то что фрагменты программ в этой главе снабжены некоторыми комментариями и сообщениями о правильном применении сценариев, они являются слишком схематичными. Существует множество книг о написании качественных программ, но все же будет не лишним изложить здесь некоторые основные рекомендации по написанию сценариев.

- При запуске с неадекватными аргументами сценарий должен вывести сообщение о его корректном применении и завершиться. Можно также реализовать возможность получения справочной информации (**`--help`**).
- Проверяйте входные данные и выходные значения на корректность. Например, прежде чем выполнять команду **`rm -rf`** для каталога, следует сначала дважды проверить, что полученный в результате путь будет соответствовать ожидаемому образцу.
- Сценарий должен возвращать надлежащий код завершения: нуль — в случае успешного выполнения и ненулевое значение — при неудачном исходе. Необязательно сопровождать каждый вид отказа уникальным кодом завершения, однако следует поинтересоваться, какая информация была бы полезной для вызывающих процедур.
- Используйте соответствующие соглашения о присвоении имен для переменных, сценариев и подпрограмм. Приведите в соответствие соглашения, действующие в языке, в остальной части кодовой основы вашего сайта и, что самое важное, других переменных и функциях, определенных в текущем проекте. Используйте принцип смешанного регистра букв или знаки подчеркивания, чтобы сделать читабельными длинные имена.[3]
- Для переменных используйте имена, которые отражают хранимые в них значения, но старайтесь все же давать имена покороче. Например, не `number_of_lines_of_input`, а `n_lines`.

[3]Имена самих сценариев также имеют большое значение. В этом контексте в роли пробелов дефисы предпочтительнее, чем знаки подчеркивания (например, **`system-config-printer`**).

- Разработайте руководство по стилю, чтобы вы и ваши коллеги могли писать код, используя одни и те же соглашения. Такое руководство облегчит вам понимание кода, написанного другими программистами, а им — написанного вами.[4]
- Начинайте каждый сценарий с блока комментария, в котором будет сказано, что делает данный сценарий и какие параметры он принимает. Не забудьте указать свое имя и текущую дату. Если сценарий требует, чтобы в системе были инсталлированы нестандартные инструменты, библиотеки или модули, обязательно перечислите их.
- Комментируйте свой код, используя такую степень детализации, какую посчитаете полезной, с учетом временного "отхода" от этого кода и с последующим возвратом к нему через месяц или два. Имеет смысл также комментировать выбор алгоритма, причины неиспользования более очевидного способа кодирования, необычные пути в коде, все "трудные места", возникшие в период разработки.
- Не загромождайте код бесполезными комментариями; отнеситесь к написанию комментариев с точки зрения потенциального читателя: пишите лаконично, грамотно и "по делу".
- Лучше запускать сценарии от имени суперпользователя `root`. Не устанавливайте для них атрибут `setuid`, поскольку такие сценарии довольно трудно сделать совершенно безопасными. Для того чтобы реализовать надлежащие стратегии управления контролем доступа, используйте команду **sudo**.
- Не пишите сценариев, которые вы не понимаете. Администраторы часто рассматривают сценарии как авторитетную документацию о том, как следует выполнять определенную процедуру. Не устанавливайте вводящий в заблуждение пример, распространяя полусырые сценарии.
- Не стесняйтесь адаптировать код из существующих сценариев для своих нужд. Только не занимайтесь программированием в стиле "копируй, вставляй и молись", если вы не понимаете код. Потратьте время, чтобы понять это. Это время никогда не пропадает зря.
- В оболочке **bash** используйте ключ **-x**, чтобы отобразить команды, прежде чем они будут выполнены, и ключ **-n**, чтобы проверить синтаксис команд без их выполнения.
- Помните, что, выполняя Python-сценарий, вы находитесь в режиме отладки, если явно не отключили его с помощью аргумента `-O` в командной строке. Это означает, что еще до вывода диагностических выходных данных вы можете протестировать специальную переменную `__debug__`.

Том Кристиансен (Tom Christiansen) предлагает следующие пять золотых правил для генерирования полезных сообщений об ошибках.

- Сообщения об ошибках должны направляться в канал **STDERR**, а не **STDOUT**.
- Включайте в сообщение об ошибке имя программы.
- Указывайте виновника (имя функции или операции) ошибочного действия.
- Если к сбою приводит системный вызов, включите строку **perror**.
- Если код завершения некоторой операции не равен нулю, обеспечьте выход из программы.

[4] С другой стороны, разработка руководства по стилю позволит привлечь к себе внимание членов вашей команды на несколько недель. Не ссорьтесь из-за руководства по стилю; старайтесь найти взаимопонимание и избегайте длительных переговоров по поводу использования квадратных скобок и запятых. Главное — убедиться, что все согласны с соглашениями об именах.

7.2. Основы работы с оболочками

Система UNIX всегда предлагала пользователям выбор оболочек, но одна из версий оболочки Bourne, **sh**, была стандартной для каждой системы UNIX и Linux. Код для оригинальной оболочки Bourne всегда был защищен лицензией AT&T, поэтому в наши дни **sh** чаще всего проявляется в форме оболочки Almquist (известной как **ash**, **dash** или просто **sh**) или же "возрожденной оболочки Bourne" с именем **bash**.

Оболочка Almquist — это воспроизведение оригинальной оболочки Bourne без излишеств. По современным меркам ее едва ли можно использовать в качестве стартовой оболочки (login shell). Она предназначена только для эффективного выполнения сценариев оболочки **sh**.

Оболочка **bash** фокусируется на интерактивной работе. На протяжении многих лет она включала в себя большинство полезных функций, созданных для других оболочек. Она по-прежнему запускает сценарии, предназначенные для оригинальной оболочки Bourne, но не особенно удобна для сценариев. Некоторые системы (например, Debian) включают в себя как **bash**, так и **dash**. Другие полагаются на оболочку **bash** как для создания сценариев, так и для интерактивного использования.

Существуют разные варианты оболочки Bourne, в частности **ksh** (оболочка Korn) и родственная ей оболочка **zsh**, которая обеспечивает широкую совместимость с **sh**, **ksh** и **bash**, а также имеет множество интересных функций, включая исправление орфографии и улучшенные средства универсализации (globbing). Насколько нам известно, она не используется как оболочка по умолчанию в любой системе, но у нее есть что-то вроде культа.

Исторически BSD-производные системы благоприятствовали интерактивной оболочке на языке C, **csh**. В настоящее время она чаще всего встречается в виде расширенной версии **tcsh**. Несмотря на ранее широко распространенное применение **csh** в качестве стартовой оболочки, ее не рекомендуется использовать в качестве языка сценариев.[5]

tcsh — это превосходная и широко доступная оболочка, но она не является производной от **sh**. Оболочки сложны; если вы не знакомы с оболочкой, нет смысла изучать одну оболочку для сценариев, а вторую — с различными функциями и синтаксисом — для ежедневного использования. Придерживайтесь современной версии **sh** как оболочки двойного назначения.

В наши дни среди всех вариантов оболочки **sh** оболочка **bash** является универсальным стандартом. Чтобы легко перемещаться между различными системами, стандартизируйте свою личную среду в **bash**.



В системе FreeBSD оболочка **tcsh** используется по умолчанию, а оболочка **bash** поставляется как часть базовой системы. Но это легко исправить: выполните команду **sudo pkg install bash** для инсталляции оболочки **bash** и используйте команду **chsh**, чтобы переключить оболочку. Вы можете установить **bash** как значение по умолчанию для новых пользователей, выполнив команду **adduser -C**.[6]

Прежде чем приступать к детализации сценариев оболочки, мы должны рассмотреть некоторые основные функции и синтаксис оболочки.

---

[5]Подробное объяснение того, почему это так, содержится в классическом заявлении Тома Кристиансена *Программирование для csh считается вредным* (Csh Programming Considered Harmful). Оно широко воспроизводится в Интернете, например на сайте **harmful.cat-v.org/software/csh**.

[6]Изменение значения по умолчанию может показаться самонадеянным, но стандартная система FreeBSD отсылает новых пользователей к оболочке Almquist **sh**, так что им некуда идти — только вверх.

Материал в этом разделе относится к основным интерактивным оболочкам в линейке **sh** (включая **bash** и **ksh**, но не **csh** или **tcsh**), независимо от точной платформы, которую вы используете. Попробуйте формы, с которыми вы еще не знакомы, и поэкспериментируйте с ними!

Редактирование команд

Мы наблюдали, как многие редактируют командные строки с помощью клавиш управления курсором. Вы ведь так не поступаете в своем текстовом редакторе, верно?

Если вы предпочитаете редактор **emacs**, то все основные команды редактора `emacs` доступны при редактировании уже выполненного ряда команд (т.е. "предыстории"). Нажав комбинацию клавиш <Ctrl+E>, вы переместитесь в конец строки, а с помощью комбинации клавиш <Ctrl+A> — в начало. Нажатие комбинации клавиш <Ctrl+P> позволяет вернуться к предыдущим командам и вызвать их для редактирования. С помощью комбинации клавиш <Ctrl+R> вы сможете шаг за шагом "прокрутить" свою "предысторию" и найти старые команды.

Если вы предпочитаете использовать редактор `vi`, переключите свою командную оболочку в режим **vi**.

```
$ set -o vi
```

В режиме **vi** редактирование является модальным, но вы начинаете работать в режиме ввода команд. Для того чтобы выйти из режима ввода, нажмите вначале клавишу <Esc>, а затем клавишу <i>, чтобы вернуться в него. В режиме редактирования нажатие клавиши <w> переместит вас вперед на одно слово; использование комбинации клавиш <f+X> позволит найти следующий символ "X" в строке и т.д. "Прогуляться" по командам, зафиксированным в "предыстории", можно с помощью клавиш <Esc> и <k>. Решили вернуться в режим редактирования **emacs**? Выполните следующую команду.

```
$ set -o emacs
```

Каналы и перенаправление потоков

Каждому процессу доступны по меньшей мере три информационных канала: "стандартный ввод" (**STDIN**), "стандартный вывод" (**STDOUT**) и "стандартная ошибка" (**STDERR**). Эти каналы устанавливаются ядром системы "от имени процесса", и поэтому сам процесс не обязательно знает их направленность. Они, например, могут быть связаны с окном терминала, файлом, подключением к сети или каналом, принадлежащим другому процессу.

Для систем UNIX и Linux используется унифицированная модель ввода-вывода, в которой каждому каналу присваивается целое число, именуемое *файловым дескриптором*. Точное число (номер), назначаемое каналу, обычно не имеет значения, но каналам **STDIN**, **STDOUT** и **STDERR** гарантированно соответствуют файловые дескрипторы **0**, **1** и **2** соответственно, чтобы обеспечить безопасное обращение к ним по номерам. В контексте интерактивного окна терминала канал **STDIN** обычно считывает данные с клавиатуры, а оба канала **STDOUT** и **STDERR** записывают свои выходные данные на экран.

Большинство команд системы Unix принимают входные данные из канала **STDIN**. Выходная информация записывается ими в канал **STDOUT**, а сообщения об ошибках — в канал **STDERR**. Это соглашение позволяет объединять команды подобно строительным блокам для организации конвейерной обработки данных.

Командная оболочка интерпретирует символы "<", ">" и ">>" как инструкции по изменению направления передаваемых командой данных в файл или принимаемых

данных из файла. Символ "<" связывает канал **STDIN** с содержимым некоторого существующего файла. Символы ">" и ">>" перенаправляют поток **STDOUT**; причем символ ">" используется для замены содержимого файла, а ">>" — для добавления данных в его конец. Например, команда

```
$ grep bash /etc/passwd > /tmp/bash-users
```

копирует строки, содержащие слово "bash" из файла **/etc/passwd** в файл **/tmp/bash-users** (при необходимости файл будет создан). Следующая команда упорядочивает содержимое этого файла и выводит результат на терминал.

```
$ sort < /tmp/bash-users[7]
root:x:0:0:root:/root:/bin/bash
...
```

Для того чтобы перенаправить потоки **STDOUT** и **STDERR** в одно и то же место, используйте символ ">&". Для того чтобы перенаправить только поток **STDERR**, используйте вариант "2>".

На примере команды **find** можно показать, почему важно обрабатывать потоки **STDOUT** и **STDERR** отдельно. Дело в том, что она формирует выходные данные по обоим каналам, особенно в случае ее выполнения непривилегированным пользователем. Например, при выполнении команды

```
$ find / -name core
```

обычно генерируется так много сообщений об ошибках "permission denied" (отсутствие прав доступа), что настоящие результаты поиска теряются в "шуме". Чтобы отбросить все сообщения об ошибках, можно использовать такой вариант.

```
$ find / -name core 2> /dev/null
```

В этой версии команды **find** только настоящие совпадения (где пользователь имеет разрешение на чтение родительского каталога) выводятся в окно терминала. Чтобы сохранить список совпадающих путей в файле, выполните такую команду.

```
$ find / -name core > /tmp/corefiles 2> /dev/null
```

Эта командная строка отправляет совпадающие пути в файл `/tmp/corefiles`, отбрасывая все ошибки и ничего не посылая в окно терминала.

Для того чтобы связать канал **STDOUT** одной команды с каналом **STDIN** другой, используйте символ "|". Рассмотрим несколько примеров.

```
$ find / -name core 2> /dev/null | less
```

Первая команда выполняет операцию **find** из предыдущего примера, но посылает список найденных файлов не в файл, а в программу постраничного вывода **less**.

```
$ ps -ef | grep httpd
```

Здесь команда **ps** генерирует список процессов, из которого команда **grep** выбирает строки, содержащие слово **httpd**. Результат выполнения команды **grep** никуда больше не перенаправляется, поэтому совпадающие строки попадают в окно терминала.

```
$ cut -d: -f7 < /etc/passwd | sort -u
```

Здесь команда **cut** выбирает из файла **/etc/passwd** пути к оболочке каждого пользователя. Список оболочек затем отправляется через команду **sort -u**, чтобы сформировать отсортированный список уникальных значений.

Для того чтобы последующая команда выполнялась только в случае успешного выполнения предыдущей, можно разделить эти команды символом "**&&**". Например, командная строка

```
$ mkdir foo && cd foo
```

[7] По правде говоря, команда **sort** может работать с файлами напрямую, поэтому символ < в этом контексте является необязательным. Он используется здесь для иллюстрации.

пытается создать каталог с именем **foo** и в случае успеха выполняет команду **cd**. В данном случае успех выполнения команды **mkdir** будет зафиксирован при получении кода завершения, равного нулю. Использование для этой цели символа, означающего операцию "логическое И", может сбить с толку тех, кто в других языках программирования использовал вычисления в сокращенной форме записи. Если кому-то это непонятно, не стоит слишком долго здесь застревать; а просто примите это как идиому командной оболочки.

И наоборот, символ "||" обеспечит выполнение следующей команды только в том случае, если предыдущая команда не выполнится (т.е. сгенерирует ненулевое значение кода завершения).

```
$ cd foo || echo "No such directory"
```

Для того чтобы разбить команду на несколько строк, для наглядности отделяя тем самым код обработки ошибок от остальной части командного конвейера, можно использовать в сценариях обратную косую черту.

```
cp --preserve --recursive /etc/* /spare/backup \
   || echo "Did NOT make backup"
```

Для получения обратного эффекта, т.е. объединения нескольких команд в одной строке, можно использовать в качестве разделителя точку с запятой.

```
$ mkdir foo; cd foo; touch afile
```

Использование переменных и кавычек

В операциях *присваивания* имена переменных никак не выделяются, но предваряются знаком доллара при *ссылке* на их значения.

```
$ etcdir='/etc'
$ echo $etcdir
/etc
```

Не ставьте до и после знака равенства (=) пробелы, в противном случае оболочка ошибочно примет имя вашей переменной за имя команды.

При ссылке на переменную можно заключить ее имя в фигурные скобки, чтобы синтаксический анализатор (да и сам человек) не сомневался в том, где заканчивается имя переменной и начинается другой текст; например, используйте запись `${etcdir}` вместо `$etcdir`. Обычно без фигурных скобок можно обойтись, но они могут оказаться полезными в случае, если вам понадобится раскрывать переменные в строках с двойными кавычками. Часто нужно сделать так, чтобы после значения переменной стояли буквы или знаки препинания. Например,

```
$ echo "Saved ${rev}th version of mdadm.conf."
Saved 8th version of mdadm.conf.
```

Для имен переменных командной оболочки не существует стандартного соглашения, но обычно предполагается, что имена, полностью написанные прописными буквами, принадлежат переменным среды или переменным, считанным из файлов глобальной конфигурации. И чаще всего все буквы в именах локальных переменных — строчные, а отдельные их компоненты разделяются символами подчеркивания. Вообще имена переменных чувствительны к регистру букв.

Командная оболочка интерпретирует строки, заключенные в одинарные и двойные кавычки, почти одинаково. Различие состоит в том, что строки в двойных кавычках служат субъектами для универсализации файловых имен (замены реальных символов в имени и расширении файла универсальными, т.е. такими метасимволами, как "*" и "?") и для раскрытия переменных (замены переменных их значениями). Например,

```
$ mylang="Pennsylvania Dutch"
$ echo "I speak ${mylang}."
I speak Pennsylvania Dutch.
$ echo 'I speak ${mylang}.'
I speak ${mylang}.
```

Обратные апострофы (также известные как обратные галочки, левые кавычки, левые одиночные кавычки или открывающие кавычки) интерпретируются аналогично двойным кавычкам, но несут при этом дополнительную нагрузку. Эта нагрузка состоит в интерпретации содержимого такой строки, как команды оболочки, выполнении этой команды и замене строки результатом выполнения этой команды.

```
$ echo "There are `wc -l < /etc/passwd` lines in the passwd file."
There are 28 lines in the passwd file.
```

Переменные окружения

При запуске процесса UNIX он получает список аргументов командной строки, а также набор переменных окружения. Большинство оболочек показывают вам текущее окружение в ответ на команду **printenv**:

```
$ printenv
EDITOR = VI
USER = garth
ENV = /gome/garth/.bashrc
LSCOLORS = exfxgxgxdxgxgxbxbxcxcx
PWD = /mega/Documents/Projects/Code/spl
HOME = /home/garth
... <total of about 50>
```

По соглашению имена переменных окружения набираются прописными буквами, но формально это не требуется.

Программы, которые вы запускаете, могут обращаться к этим переменным и соответствующим образом изменять их поведение. Например, программа **vipw** проверяет переменную среды EDITOR, чтобы определить, какой текстовый редактор будет работать для вас.

Переменные окружения автоматически импортируются в пространство имен переменных **sh**, поэтому их можно установить и прочитать с помощью стандартного синтаксиса. Чтобы превратить переменную оболочки в переменную среды, используйте команду **export** *имя_переменной*. Вы также можете комбинировать этот синтаксис со значением присваивания, как показано здесь:

```
$ export EDITOR = nano
$ vipw
<запускает редактор nano>
```

Несмотря на то что они называются переменными окружения, эти значения не существуют в каком-то абстрактном месте вне пространства и времени. Оболочка передает снимок текущих значений в любую запущенную вами программу, но при этом нет никаких открытых соединений. Более того, каждая оболочка или программа и каждое окно терминала имеют свою собственную отдельную копию окружения, которая может быть изменена отдельно.

Команды для переменных окружения, которые вы хотите настроить во время входа в систему, должны быть включены в ваш файл **~/.profile** или **~/.bash_profile**. Другие переменные среды, такие как PWD для текущего рабочего каталога, автоматически поддерживаются оболочкой.

Команды фильтрации

Любая корректная команда, которая считывает данные из канала **STDIN** и записывает данные в канал **STDOUT**, может использоваться в качестве фильтра (т.е. компонента конвейера) для обработки данных. В этом разделе мы кратко рассмотрим некоторые из наиболее широко используемых команд фильтра, но список практически бесконечен. Команды фильтрации отличаются настолько сильным "коллективизмом", что порой даже трудно продемонстрировать их индивидуальное использование.

Большинство команд фильтра принимают одно или несколько имен файлов в командной строке. Только если вы не укажете файл, они прочитают данные из стандартного входного потока.

Команда cut: разбивка строк на поля

Команда **cut** выводит выбранные части входных строк. Она чаще всего выделяет поля с разделителями, как в примере, показанном ниже, но также может возвращать сегменты, определяемые границами столбцов. Разделителем по умолчанию служит символ <Tab>, но вы можете изменить разделители с помощью опции **-d**. Параметр **-f** определяет, какие поля включать в вывод.

Пример использования команды **cut** приведен ниже в разделе, посвященном команде **uniq**.

Команда sort: сортировка строк

Команда **sort** сортирует входные строки. Звучит просто, да? Хотя на самом деле не все так безоблачно: существуют потенциальные нюансы, связанные с точно заданными частями сортируемых строк (т.е. с использованием сортировочного ключа) и порядком сортировки. Наиболее распространенные варианты применения этой команды показаны в табл. 7.2, в отношении же остальных случаев обратитесь к соответствующей man-странице.

Таблица 7.2. Ключи команды sort

Ключ команды	Значение
-b	Игнорировать ведущие пробельные символы
-f	Сортировать независимо от регистра букв
-h	Сортировать "читабельные" числа (например, 2Мб)
-k	Указывать столбцы, которые формируют сортировочный ключ
-n	Сравнивать поля как целые числа
-r	Изменить порядок сортировки на противоположный
-t	Установить разделитель полей (по умолчанию — пробельный символ)
-u	Выводить только уникальные записи

На примерах приведенных ниже команд демонстрируется различие между числовой и лексикографической сортировкой (последняя действует по умолчанию). В обеих командах используются ключи **-t:** и **-k3,3** для сортировки файла **/etc/group** по его третьему полю (в качестве разделителя используется двоеточие), содержащему идентификатор (ID) группы. Первая команда реализует числовую сортировку, а вторая — лексикографическую (в алфавитном порядке).

```
$ sort -t: -k3,3 -n /etc/group[8]
root:x:0:
bin:x:1:daemon
daemon:x:2:
...
$ sort -t: -k3,3 /etc/group
root:x:0:
bin:x:1:daemon
users:x:100:
...
```

Также полезна опция **-h**, реализующая числовую сортировку, которая понимает суффиксы, такие как M для мега и G для гига. Например, следующая команда правильно сортирует размеры подкаталогов в каталоге **/usr**, сохраняя корректность вывода.

```
$ du -sh /usr/* | sort -h
16K   /usr/locale
128K  /usr/local
648K  /usr/games
15M   /usr/sbin
20M   /usr/include
117M  /usr/src
126M  /usr/bin
845M  /usr/share
1.7G  /usr/lib
```

Команда uniq: вывод уникальных строк

Команда **uniq** по существу подобна команде **sort -u**, но у нее есть некоторые полезные ключи, которые команда **sort** не эмулирует. Так, ключ **-c** используется для подсчета количества экземпляров каждой строки, ключ **-d** — для отображения только строк, имеющих дубликаты, а ключ **-u** — для отображения только строк, не имеющих дубликаты. При этом входные данные должны быть предварительно отсортированы, обычно путем "пропускания" через команду **sort**.

Например, по результатам выполнения следующей команды видно, что 20 пользователей в качестве стартовой оболочки используют /bin/bash, а остальные 12 — **/bin/false**. (Последний результат характерен либо для псевдопользователей, либо для пользователей, учетные записи которых были заблокированы.)

```
$ cut -d: -f7 /etc/passwd | sort | uniq -c
  20 /bin/bash
  12 /bin/false
```

Команда wc: подсчет строк, слов и символов

Подсчет количества строк, слов и символов в файле — еще одна распространенная операция, удобным средством реализации которой и является команда wc (*w*ord *c*ount). Выполненная без каких-либо ключей, она выведет все три результата подсчета.

```
$ wc /etc/passwd
32 77 2003 /etc/passwd
```

В контексте написания сценариев команду **wc** часто применяют только с одним из ключей (**-l**, **-w** или **-c**), чтобы результат ее выполнения состоял из одного числа. Эту форму чаще всего можно встретить внутри обратных апострофов, и тогда результат может быть сохранен или использован по назначению.

---

[8] Команда **sort** принимает ключ **-k3** (а не **-k3,3**), но, вероятно, она не делает то, что вы ожидаете. Без номера завершающего поля ключ сортировки действует до конца строки.

Команда tee: копирование входного потока в два места

Командный конвейер, как правило, действует прямолинейно, но зачастую полезно распараллелить поток данных, т.е. "перехватить" его и отправить копию в файл или окно терминала. Это можно сделать с помощью команды `tee`, которая отправляет свой стандартный входной поток как в стандартный выходной канал, так и в файл, указанный в командной строке. Представьте действие этой команды в виде тройника в трубопроводе (*tee* (англ.) — тройник. — *Примеч. пер.*).

Устройство **`/dev/tty`** можно считать синонимом для текущего терминала. Например, команда

```
$ find / -name core | tee /dev/tty | wc -l
```

выводит найденные путевые имена файлов `core` и результат подсчета их количества.

Часто работа конвейера с командой **`tee`**, выводящей результаты и в файл, и в окно терминала (для проверки), занимает много времени. Вы можете понаблюдать за первыми результатами, чтобы убедиться в том, что все работает как надо, а затем смело уходите: результаты сохранятся в файле.

Команды head и tail: чтение файла с начала или с конца

Просмотр строк с начала или конца файла — обычная административная операция. Команды **`head`** и **`tail`** отображают по умолчанию десять строк, но вы можете использовать ключ, задающий желаемое количество строк.

Для интерактивного использования команда **`head`** уже несколько устарела, и вместо нее (в этом контексте) часто используется команда **`less`**, которая разбивает файлы для отображения по страницам. Однако команду **`head`** можно успешно применять в сценариях и с другой целью.

С командой **`tail`** используется ключ **`-f`**, который чрезвычайно полезен для системных администраторов. Вместо немедленного завершения после вывода требуемого количества строк команда **`tail -f`** ожидает поступления новых строк, которые будут добавлены в конец файла, а затем выведены по назначению, что очень удобно для мониторинга журналов регистрации. Однако следует иметь в виду, что программа, которая реализует запись данных в файл, может буферизировать свои выходные данные. Даже если строки будут добавляться регулярно (с точки зрения логики), они могут стать видимыми только фрагментами объемом 1 или 4 КиБ.[9]

Команды **`head`** и **`tail`** получают несколько имен файлов из командной строки. Даже команда **`tail -f`** допускает использование нескольких файлов, и это довольно удобно; когда появляются новые результаты, подлежащие выводу на экран, команда **`tail`** выводит имя файла, содержащего эти результаты.

Для остановки процесса мониторинга нажмите комбинацию клавиш <Ctrl+C>.

Команда grep: поиск текста

При выполнении команды **`grep`** по мере просмотра входного текста выводятся строки, которые совпадают с заданным образцом (шаблоном). Свое название эта команда получила "в наследство" от команды **`g`**`/регулярное_выражение/`**`p`**) из старого (и ныне действующего) редактора **`ed`**, используемого в самых первых версиях UNIX.

"Регулярные выражения" (regular expressions) — это шаблоны, предназначенные для поиска совпадающего с ними текста и написанные на стандартном языке шаблонов.

[9] Информацию о единицах измерения см. в разделе 1.6 главы 1.

Они представляют собой универсальный стандарт, используемый большинством программ при поиске по совпадению с шаблоном, хотя и с небольшими различиями в реализациях. Странное имя команды уходит своими корнями в оригинальные изыскания соответствующих разделов теории вычислений. Более детально синтаксис регулярных выражений рассматривается в разделе 7.4.

Подобно большинству фильтров, команда **grep** имеет множество ключей. Например, ключ **-c** позволяет выводить количество совпавших строк, ключ **-i** служит для игнорирования регистра букв при сопоставлении, а ключ **-v** предназначен для вывода отличающихся (а не совпавших) строк. Очень полезным является ключ **-l** (прописная буква "L"), который заставляет команду `grep` выводить только имена файлов, содержащие совпавшие с шаблоном строки, а не все совпавшие строки.

Например, выполнение команды

```
$ sudo grep -l mdadm /var/log/*
/var/log/auth.log
/var/log/syslog.0
```

демонстрирует, что записи с именем утилиты **mdadm** нашлись в двух различных файлах системных журналов.

Команду **grep** можно считать классической реализацией базового механизма использования регулярных выражений, но в некоторых версиях предлагается выбор других диалектов. Например, Linux-команда **grep -p** служит для поиска выражений в стиле языка Perl, хотя в справочных страницах можно найти неопределенное предупреждение о том, что такой вариант носит "экспериментальный характер". Для полной уверенности в своих сценариях просто используйте язык Ruby, Python или Perl.

Если вы фильтруете с помощью команд **tail -f** и **grep**, то добавьте параметр **--line-buffered**, чтобы убедиться, что вы увидите каждую соответствующую строку, как только она станет доступной

```
$ tail -f /var/log/messages | grep --line-buffered ZFS
May 8 00:44:00 nutrient ZFS: vdev state changed, pool_
guid=10151087465118396807 vdev_guid=7163376375690181882
...
```

7.3. Написание сценариев для оболочки sh

Оболочка **sh** отлично подходит для простых сценариев, которые автоматизируют то, что вы в противном случае вводили бы в командной строке. Ваши навыки командной строки переносятся на сценарии **sh**, и наоборот, что помогает вам извлечь максимальную отдачу от времени обучения, которое вы вкладываете в изучение оболочки **sh**. Но как только сценарий **sh** получает более 50 строк или когда вам нужны функции, которых нет в оболочке **sh**, приходит время переходить на язык Python или Ruby.

Для сценариев есть смысл ограничить себя диалектом, понятным исходной оболочке Bourne, которая является стандартом IEEE и POSIX. Оболочки, совместимые с **sh**, часто дополняют эту базовую линейку языковыми функциями. Целесообразно применять эти расширения, если вы делаете это намеренно и хотите использовать конкретный интерпретатор. Однако чаще всего сценарии в конечном итоге используют эти расширения непреднамеренно, и затем администраторы удивляются, когда их сценарии не работают в других системах.

В частности, не предполагайте, что версией оболочки **sh** в вашей системе всегда является **bash**, или даже, что оболочка **bash** доступна. В 2006 году система Ubuntu заменила **bash** на **dash** в качестве интерпретатора сценариев по умолчанию, и в рамках этого процесса преобразования был составлен удобный список команд, за которыми нужно следить. Вы можете найти его на сайте **wiki.ubuntu.com/DashAsBinSh**.

Выполнение

В оболочке **sh** комментарии начинаются со знака # и продолжаются до конца строки. Как и в командной строке, вы можете разбить одну логическую строку на несколько физических строк, обозначив переход на новую строку символом обратной косой черты (\). И, наоборот, можно поместить на одной строке несколько операторов, разделив их точкой с запятой.

Сценарий для оболочки sh может состоять только из последовательности командных строк. Например, следующий сценарий **helloworld** просто выполняет команду **echo**.

```
#!/bin/sh
echo "Hello, world!"
```

Первая строка содержит сочетание символов #!, которое означает, что данный текстовый файл является сценарием, предназначенным для интерпретации командной оболочкой из каталога **/bin/sh** (которая сама может служить ссылкой на оболочку **dash** и **bash**) При принятии решения о том, как выполнить этот файл, ядро найдет соответствующий синтаксис. С точки зрения оболочки, выполняющей этот сценарий, первая строка представляет собой просто комментарий.

Теоретически, если ваша оболочка **sh** находится в другом месте, вам придется отредактировать первую строку. Тем не менее многие существующие сценарии предполагают, что она находится в каталоге **/bin/sh**, который системы вынуждены поддерживать, хотя бы через ссылку.

Если вам нужен сценарий для запуска в оболочке **bash** или в другом интерпретаторе, который может не иметь одного и того же командного пути в каждой системе, вы можете использовать каталог **/usr/bin/env** для поиска переменной среды **PATH** для определенной команды.[10] Например,

```
#! /usr/bin/env ruby
```

является распространенной идиомой для запуска сценариев на языке Ruby. Подобно каталогу **/bin/sh**, каталог **/usr/bin/env** является настолько широко распространенным вариантом, что все системы обязаны его поддерживать.

Для того чтобы подготовить этот файл к выполнению, достаточно установить его бит, "отвечающий" за выполнение (см. раздел 5.5).

```
$ chmod +x helloworld
$ ./helloworld[11]
Hello, world!
```

[10]Поиск путей имеет последствия для безопасности, особенно при запуске сценариев в среде **sudo**. Дополнительную информацию об обработке переменных окружения в среде **sudo** см. в разделе 3.2.

[11]Если ваша оболочка понимает команду **helloworld** без префикса **./**, это означает, что в вашем пути поиска указан текущий каталог (**.**). Это плохо, поскольку дает другим пользователям возможность устраивать для вас "ловушки" в надежде, что вы будете пытаться выполнить определенные команды при переходе к каталогу, в котором они имеют доступ для записи.

Можно также непосредственно запустить (активизировать) оболочку в качестве интерпретатора сценария.

```
$ sh helloworld
Hello, world!
$ source helloworld
Hello, world!
```

Первая команда выполнит сценарий **helloworld** в новом экземпляре оболочки **sh**, а вторая — заставит вашу начальную оболочку прочитать содержимое файла, а затем выполнить его. Второй вариант полезен в случае, когда сценарий устанавливает переменные среды или выполняет другие операции настройки, применимые только к текущей оболочке. Обычно этот вариант используется при написании сценариев, включающих содержимое файла конфигурации, написанного в виде ряда присваиваний значений переменным.[12]

🕮 Дополнительную информацию о битах разрешения можно прочитать в разделе 5.5.

Если вы пришли из мира Windows, то вам привычно использовать понятие расширения файла, по которому можно судить о типе файла, а также о том, можно ли его выполнить. В мире UNIX и Linux признак того, может ли файл быть выполнен (и если да, то кем), содержится в специальных битах полномочий. При желании вы можете наделить свои **bash**-сценарии суффиксом **.sh**, который бы напоминал вам об их типе, но тогда при выполнении соответствующей команды вам придется вводить суффикс **.sh**, поскольку UNIX не интерпретирует расширения специальным образом.

От команд к сценариям

Прежде чем переходить к особенностям написания сценариев для оболочки **sh**, остановимся на методике этого процесса. Многие пишут эти сценарии так же, как на языке Python или Ruby, т.е. используя какой-нибудь текстовый редактор. Однако удобнее рассматривать в качестве интерактивной среды разработки сценариев режим с приглашением на ввод команды.

Предположим, например, что у вас есть журналы регистрации, именованные с суффиксами **.log** и **.LOG** и разбросанные по всей иерархической системе каталогов. Допустим также, что вы хотите изменить эти суффиксы, переведя их в верхний регистр. Прежде всего, попытаемся отыскать все эти файлы.

```
$ find . -name '*log'
.do-not-touch/important.log
admin.com-log/
foo.log
genius/spew.log
leather_flog
...
```

Ой! Похоже на то, что нам надо включить в шаблон точку и при поиске игнорировать каталоги. Нажмите комбинацию клавиш <Ctrl+P>, чтобы вернуться в режим командной строки, а затем модифицируйте ее.

```
$ find . -type f -name '*.log'
.do-not-touch/important.log
foo.log
```

[12] Синонимом для команды **source** служит команда в виде точки (dot-команда), например **. helloworld**.

```
genius/spew.log
...
```

Отлично, это уже выглядит лучше. Правда, каталог **.do-not-touch** (т.е. "не трогать") вызывает смутное чувство тревоги; но мы можем избавиться от него.

```
$ find . -type f -name '*.log' | grep -v .do-not-touch
foo.log
genius/spew.log
...
```

Прекрасно, мы получили абсолютно точный список файлов, которые должны быть переименованы. Попробуем сгенерировать несколько новых имен.

```
$ find . -type f -name '*.log' | grep -v .do-not-touch | while read fname
> do
> echo mv $fname `echo $fname | sed s/.log/.LOG/`
> done
mv foo.log foo.LOG
mv genius/spew.log genius/spew.LOG
...
```

Да, это именно те команды, которые позволят переименовать нужные файлы. Как же их выполнить? Мы могли бы снова вызвать уже выполненную команду и отредактировать команду **echo**, чтобы заставить оболочку **sh** выполнять команды **mv**, а не просто выводить их. Ведь передача команд в отдельную копию оболочки **sh** — более надежный вариант работы, который к тому же требует меньшего объема редактирования.

Нажав комбинацию клавиш <Ctrl+P>, мы обнаруживаем, что оболочка **bash** заботливо свернула наш мини-сценарий в одну-единственную строку. К этой "уплотненной" командной строке мы просто добавляем канал, передающий выходные данные команде **sh -x**.

```
$ find . -type f -name '*.log' | grep -v .do-not-touch | while read fname;
do echo mv $fname `echo $fname | sed s/.log/.LOG/`; done | sh -x
+ mv foo.log foo.LOG
+ mv genius/spew.log genius/spew.LOG
...
```

Ключ `-x` команды **bash** обеспечивает вывод каждой команды перед ее выполнением.

Мы завершили переименование файлов, но нам хотелось бы сохранить этот сценарий, чтобы использовать его снова. Встроенная в **sh** команда **fc** по своему действию во многом аналогична нажатию комбинации клавиш <Ctrl+P>, но вместо возврата последней команды в командную строку она передает команду в заданный вами редактор. Добавьте в свой файл строку идентификационного комментария, поместите сохраненный файл в приемлемый для вас каталог (например, **~/bin** или **/usr/local/bin**), сделайте файл исполняемым, и вы получите настоящий сценарий.

Итак, подытожим.

1. Разрабатывайте сценарий (или его часть) в виде конвейера команд, причем пошагово и в режиме выполнения командных строк.
2. Пересылайте результат в стандартный выходной поток, проверяя правильность работы используемых вами команд.
3. На каждом этапе используйте буфер ранее выполненных команд для их появления в командной строке и инструменты редактирования — для их модификации.

4. Пока вы получаете неправильные результаты, можно считать, что вы, по сути, ничего не сделали, и до тех пор, пока команды не заработают так, как надо, ничего (из уже сделанного) не надо удалять.
5. Если результат выглядит правильным, выполните команды на реальном примере, чтобы убедиться, что все получилось так, как ожидалось.
6. Используйте команду **fc**, чтобы зафиксировать свою работу в редакторе, оформите ее соответствующим образом и сохраните.

В приведенном выше примере мы вывели командные строки, а затем направили их в подоболочку для выполнения. Этот метод не является универсально применимым, но часто оказывается полезным. В качестве альтернативного варианта можно фиксировать результаты, перенаправляя их в файл. Терпеливо добивайтесь получения нужных результатов и, пока не увидите их, не выполняйте никаких потенциально деструктивных действий.

Ввод и вывод данных

Команда **echo** не отличается сложностью, но зато проста в применении. Для получения большего контроля над выводом данных используйте команду **printf**. Она не очень удобна, поскольку предполагает, что вы должны в явном виде указывать в нужных для вас местах символы перехода на новую строку (\n), но позволяет использовать символ табуляции и другие средства форматирования результата. Сравните результаты выполнения следующих двух команд.

```
$ echo "\taa\tbb\tcc\n"
\taa\tbb\tcc\n
$ printf "\taa\tbb\tcc\n"
  aa    bb    cc
```

Иногда работа команд **echo** и **printf** поддерживается на уровне операционной системы (обычно соответствующие им исполняемые файлы хранятся в каталогах **/bin** и **/usr/bin** соответственно). Хотя эти команды и встроенные в оболочку утилиты в целом подобны, они могут иметь незначительные отличия, особенно это касается команды **printf**. Вы можете либо придерживаться **sh**-синтаксиса, либо вызывайте "внешнюю" команду **printf**, указывая ее полный путь.

Для того чтобы сформировать для пользователя приглашение ввести данные, можно использовать команду read.

```
#!/bin/sh

echo -n "Enter your name: "
read user_name

if [ -n "$user_name" ]; then
     echo "Hello $user_name!"
     exit 0
else
     echo "Greetings, nameless one!"
     exit 1
fi
```

Ключ **-n** в команде **echo** подавляет привычный переход на новую строку, но здесь была бы кстати команда **printf**. Мы еще рассмотрим вкратце синтаксис оператора **if**, но его действие здесь, с нашей точки зрения, очевидно. Ключ **-n** в операторе **if** обе-

спечит значение истины, если его строковый аргумент не окажется нулевым. Вот как выглядит результат выполнения этого сценария.

```
$ sh readexample
Enter your name: Ron
Hello Ron!
```

Пробелы в именах файлов

Именование файлов и каталогов, по существу, не регламентируется, за исключением того, что имена ограничены по длине и не должны содержать косой черты или нули. В частности, допускаются пробелы. К сожалению, система UNIX имеет давнюю традицию разделения аргументов командной строки на пробелы, поэтому устаревшее программное обеспечение имеет тенденцию давать сбой, когда в именах файлов появляются пробелы.

Пробелы в именах файлов были обнаружены прежде всего в файловых системах, совместно используемых с компьютерами Mac и персональными компьютерами, но теперь они внедрились в культуру UNIX и также встречаются в некоторых стандартных пакетах программного обеспечения. Выхода нет: административные сценарии должны быть готовы обрабатывать пробелы в именах файлов (не говоря уже об апострофах, звездочках и различных других угрожающих пунктуационных метках).

В оболочке и в сценариях можно указывать имена файлов с пробелами, чтобы держать их части вместе. Например, команда

```
$ less "My spacey file"
```

считает файл **My spacey file** в качестве единственного аргумента команды **less**. Вы также можете избежать отдельных пробелов с помощью обратной косой черты:

```
$ less My\ spacey\ file
```

Функция завершения имени файла большинства оболочек (зачастую привязанная к клавише <Tab>) обычно автоматически добавляет обратную косую черту. Когда вы пишете сценарии, полезным оружием, о котором нужно знать, является опция **-print0**. В сочетании с параметром **xargs -0** она делает комбинацию **find /xargs** корректной, независимо от пробелов, содержащихся в именах файлов. Например, команда

```
$ find / home-type f -size + 1M -print0 | xargs -0 ls -l
```

выводит на экран длинный список всех файлов в каталоге **/home** размером более одного мегабайта.

Функции и аргументы командной строки

Аргументы командной строки служат для сценария переменными с числовыми именами: `$1` — первый аргумент командной строки, `$2` — второй и т.д. Аргумент `$0` содержит имя, по которому был вызван сценарий, например **./bin/example.sh**, т.е. это не фиксированное значение.

Переменная `$#` содержит количество переданных аргументов командной строки, а переменная `$*` — все эти аргументы. Ни одна из этих переменных не учитывает аргумент `$0`. Рассмотрим пример использования этих аргументов.

```
#!/bin/sh

show_usage() {
     echo "Usage: $0 source_dir dest_dir" 1>&2
```

```
    exit 1
}

# Здесь начинается основная программа
if [ $# -ne 2 ]; then
    show_usage
else # There are two arguments
    if [ -d $1 ]; then
        source_dir=$1
    else
        echo 'Invalid source directory' 1>&2
        show_usage
    fi
    if [ -d $2 ]; then
        dest_dir=$2
    else
        echo 'Invalid destination directory' 1>&2
        show_usage
    fi
fi

printf "Source directory is ${source_dir}\n"
printf "Destination directory is ${dest_dir}\n"
```

Если вы вызываете сценарий без аргументов или с недействительными аргументами, сценарий должен вывести короткое сообщение об ошибке, чтобы напомнить вам, как его использовать. В приведенном выше примере сценарий принимает два аргумента, подтверждает, что оба аргумента являются каталогами, и выводит на экран их имена. Если аргументы недействительны, сценарий печатает сообщение об использовании и выходит с ненулевым кодом возврата. Если вызывающий сценарий проверяет код возврата, он будет знать, что этот сценарий не удалось выполнить правильно.

Мы создали отдельную функцию **show_usage** для печати сообщения об использовании. Если впоследствии сценарий будет обновлен, чтобы принимать дополнительные аргументы, сообщение об использовании должно быть изменено только в одном месте. Обозначение `1>&2` в строках, которые отображают сообщения об ошибках, выводит результаты в поток **STDERR**.

```
$ mkdir aaa bbb
$ sh showusage aaa bbb
Source directory is aaa
Destination directory is bbb
$ sh showusage foo bar
Invalid source directory
Usage: showusage source_dir dest_dir
```

Аргументы для функций оболочки **sh** рассматриваются как аргументы командной строки. Первый аргумент — `$1` и т.д. Как было показано выше, `$0` остается именем сценария.

Чтобы сделать пример более надежным, мы могли бы заставить программу `show_usage` получать в качестве аргумента код ошибки. Это позволит возвращать более определенный код для каждого типа сбоя. Следующий фрагмент кода показывает, как это может выглядеть.

```
show_usage() {
     echo "Usage: $0 source_dir dest_dir" 1>&2
     if [ $# -eq 0 ]; then
          exit 99 # Exit with arbitrary nonzero return code
     else
          exit $1
     fi
}
```

В этой версии подпрограммы аргумент не является обязательным. Внутри символы `$#` сообщают, сколько аргументов было передано. Если не указан более конкретный код, сценарий заканчивает работу с кодом 99. Однако конкретное значение, например

```
show_usage 5
```

приводит к тому, что сценарий завершает работу после вывода сообщения об использовании именно с этим кодом. (Переменная оболочки `$?` содержит статус выхода последней выполненной команды, независимо от того, используется она внутри сценария или в командной строке.)

В оболочке **sh** сильна аналогия между функциями и командами. Вы можете определить полезные функции в файле **~/.bash_profile** (**~/.profile** для обычной оболочки **sh**), а затем использовать их в командной строке, как если бы они были командами. Например, если ваш сайт стандартизован на сетевом порту 7988 для протокола SSH (форма "безопасность через безвестность"), вы можете определить функцию

```
ssh() {
 /usr/bin/ssh -p 7988 $*
}
```

в каталоге **~/.bash_profile**, чтобы функция команда **ssh** всегда выполнялась с параметрами **-p 7988**.

Как и многие оболочки, **bash** имеет механизм псевдонимов, который может воспроизвести этот пример более точно, при этом функция становится все более универсальной и мощной.

Поток управления

Выше в этой главе мы уже рассмотрели несколько конструкций `if-then` и `if-then-else` (они работают вполне ожидаемым образом). Терминатором (признаком конца) для оператора `if` служит оператор `fi`. Для образования цепочки `if`-операторов можно использовать ключевое слово `elif`, означающее "`else if`".

```
if [ $base -eq 1 ] && [ $dm -eq 1 ]; then
     installDMBase
elif [ $base -ne 1 ] && [ $dm -eq 1 ]; then
     installBase
elif [ $base -eq 1 ] && [ $dm -ne 1 ]; then
     installDM
else
     echo '==> Installing nothing'
fi
```

Как и специальный `[]`-синтаксис для выполнения операций сравнения, так и "параметрические" имена операторов целочисленного сравнения (например, **-eq**) являются наследием утилиты **/bin/test** из ранней командной оболочки Стивена Борна. В дей-

ствительности квадратные скобки — это не что иное, как условное обозначение вызова утилиты **test**; они не являются частью оператора **if**.[13]

В табл. 7.3 собраны операторы оболочки **bash**, позволяющие сравнивать числа и строки. В отличие от языка Perl, в оболочке **bash** используются текстовые операторы для чисел и символические операторы для строк.

Таблица 7.3. Элементарные операторы сравнения оболочки sh

Строки	Числа	Истина, если
`x = y`	`x -eq y`	x равно y
`x != y`	`x -ne y`	x не равно y
`x < y`	`x -lt y`	x меньше y
`x <= y`	`x -le y`	x меньше или равно y
`x >`[a] `y`	`x -gt y`	x больше y
`x >= y`	`x -ge y`	x больше или равно y
`-n x`	-	x не равно значению `null`
`-z x`	-	x равно значению `null`

[a]Здесь должна использоваться обратная косая черта или двойные квадратные скобки, чтобы предотвратить интерпретацию символа > в символ перенаправления ввода-вывода.

В оболочке **bash** предусмотрены возможности оценки свойств файлов (снова-таки как освобождение от наследства **/bin/test**). Некоторые из операторов тестирования и сравнения файлов приведены в табл. 7.4.

Таблица 7.4. Операторы проверки файлов оболочки sh

Оператор	Истина, если
`-d` *файл*	Файл *файл* существует и является каталогом
`-e` *файл*	Файл *файл* существует
`-f` *файл*	Файл *файл* существует и является обычным
`-r` *файл*	У вас есть право доступа для чтения файла
`-s` *файл*	Файл *файл* существует и он не пустой
`-w` *файл*	У вас есть право доступа для записи в файл
файл1 `-nt` *файл2*	Файл *файл1* новее, чем *файл2*
файл1 `-ot` *файл*	Файл *файл1* старше, чем *файл2*

Несмотря на всю полезность формы **elif**, зачастую с точки зрения ясности программного кода лучше использовать структуру **case**. Ниже показан ее синтаксис на примере функции, которая централизует процесс регистрации сценария. Конкретные варианты описываются закрывающими скобками после каждого условия и двумя точками с запятой, завершающими блок операторов, который должен быть выполнен при реализации заданного условия. Оператор **case** завершается ключевым словом **esac**.

```
# Уровень протоколирования устанавливается в глобальной
# переменной LOG_LEVEL. Возможные варианты перечислены в порядке
# от самого строгого до наименее строгого: Error, Warning, Info и Debug.

logMsg {
    message_level=$1
```

[13]Сейчас эти операции встроены в оболочку и уже не запускают на выполнение утилиту **/bin/test**.

```
    message_itself=$2
    if [ $message_level -le $LOG_LEVEL ]; then
         case $message_level in
              0) message_level_text="Error" ;;
              1) message_level_text="Warning" ;;
              2) message_level_text="Info" ;;
              3) message_level_text="Debug" ;;
              *) message_level_text="Other"
         esac
         echo "${message_level_text}: $message_itself"
    if
  }
```

Эта функция иллюстрирует общепринятую парадигму "уровня регистрации" (log level), используемую многими приложениями административного характера. Код этого сценария позволяет генерировать сообщения на различных уровнях детализации, но действительно регистрируются или отрабатываются только те из них, которые "проходят" глобально устанавливаемый порог `$LOG_LEVEL`. Чтобы прояснить важность каждого сообщения, его текст предваряется меткой, описывающей соответствующий уровень регистрации.

Циклы

В оболочке **sh** конструкция `for...in` позволяет упростить выполнение некоторых действий для группы значений или файлов, особенно при универсализации файловых имен, т.е. замене реальных символов в имени и расширении универсальными (например, "`*`" и "`?`") с целью формирования целых списков имен файлов. Шаблон `*.sh` в приведенном ниже цикле `for` позволяет обработать целый список совпадающих с ним (шаблоном) имен файлов из текущего каталога. Оператор `for`, проходя по этому списку, по очереди присваивает имя каждого файла переменной `script`.

```
#!/bin/sh

suffix=BACKUP--`date +%Y-%m-%d-%H%M`

for script in *.sh; do
    newname="$script.$suffix"
    echo "Copying $script to $newname..."
    cp -p $script $newname
done
```

Вывод выглядит следующим образом:

```
$ sh forexample
Copying rhel.sh to rhel.sh.BACKUP--2017-01-28-2228...
Copying sles.sh to sles.sh.BACKUP--2017-01-28-2228...
...
```

В раскрытии имени файла нет ничего магического; все работает в точном соответствии с тем, как написано в командной строке. Другими словами, сначала имя файла раскрывается (т.е. шаблон заменяется существующим именем), а затем обрабатывается интерпретатором в развернутом виде. Имена файлов можно также вводить статически, как показано ниже:

```
for script in rhel.sh sles.sh; do
```

В действительности любой список имен, содержащих пробельные символы (включая содержимое переменной), обрабатывается как объект циклической конструкции `for... in`. Вы также можете опустить список полностью (вместе с ключевым словом `in`), и в этом случае цикл неявно выполняет итерации по аргументам командной строки сценария (на верхнем уровне) или аргументы, переданные функции:

```
#!/bin/sh

for file; do
    newname="${file}.backup"
    echo "Copying $file to $newname..."
    cp -p $file $newname
done
```

Циклы в оболочке **bash**, в отличие от традиционной оболочки **sh**, ближе к циклам из традиционных языков программирования, в которых задается стартовое выражение, инкрементация и условие окончания цикла. Например:

```
# bash-specific
for (( i=0 ; i < $CPU_COUNT ; i++ )); do
    CPU_LIST="$CPU_LIST $i"
done
```

На примере следующего сценария иллюстрируется цикл `while`, который часто применяется для обработки аргументов командной строки и чтения строк файла.

```
#!/bin/sh

exec 0<$1
counter=1
while read line; do
   echo "$counter: $line"
   counter=$((counter + 1))
done
```

Вот как выглядит результат выполнения этого сценария.

```
$ sh whileexample /etc/passwd
1: root:x:0:0:Superuser:/root:/bin/bash
2: bin:x:1:1:bin:/bin:/bin/bash
3: daemon:x:2:2:Daemon:/sbin:/bin/bash
...
```

В этом сценарии есть интересные особенности. Оператор `exec` переопределяет стандартный выходной поток сценария так, чтобы входные данные считывались из файлов, имена которых должны определяться в соответствии с первым аргументом командной строки.[14] Если окажется, что такой файл не существует, данный сценарий сгенерирует ошибку.

Оператор `read` в конструкции `while` на самом деле является встроенным в оболочку, но действует подобно внешней команде. Внешние команды можно также помещать в конструкцию `while`. Цикл `while` в такой форме завершится тогда, когда внешняя команда возвратит ненулевое значение кода завершения.

Выражение `$((counter++))` выглядит несколько странно. Обозначение `$((...))` говорит о вычислении выражения. Кроме того, оно делает необязательным использование

[14] В зависимости от вызова, оператор **exec** может иметь и такое значение: "Остановить этот сценарий и передать управление другому сценарию или выражению". В этом состоит еще одна странность оболочки: доступ к обеим упомянутым здесь функциям реализуется через один и тот же оператор.

символа $ для обозначения имен переменных. Это выражение заменяется результатом арифметического вычисления.

Обозначение $((...)) применяется также в контексте двойных кавычек. В оболочке **bash**, поддерживающей оператор постинкрементации из языка C++, тело цикла можно свернуть в одну строку

```
while read line; do
   echo "$((counter++)): $line"
done
```

Арифметика

Все переменные в оболочке **sh** представляют собой строковые значения, поэтому оболочка **sh** не делает различия в присваиваниях между числом 1 и символьной строкой "1". Различие состоит в использовании переменных. Следующий код иллюстрирует это различие.

```
#!/bin/sh

a=1
b=$((2))

c=$a+$b
d=$((a + b))

echo "$a + $b = $c \t(plus sign as string literal)"
echo "$a + $b = $d \t(plus sign as arithmetic addition)"
```

Этот сценарий выводит следующий результат.

```
1 + 2 = 1+2 (plus sign as string literal)
1 + 2 = 3   (plus sign as arithmetic addition)
```

Обратите внимание на то, что знак "плюс" в присваивании переменной $c не работает как оператор конкатенации для строк. Он остается всего лишь литеральным символом, так что эта строка эквивалентна следующей.

```
c="$a+$b"
```

Для того чтобы выполнить вычисления, необходимо заключить выражение в двойные скобки: $((...)), как показано выше в присваивании переменной $d. Однако даже эта мера предосторожности не позволяет получить в переменной $d числового значения; это значение по-прежнему хранится в виде строки "3".

В оболочке **sh** реализован обычный ассортимент операторов: арифметических, логических и отношений (подробнее см. соответствующие man-страницы).

7.4. Регулярные выражения

Как мы уже упоминали в разделе 7.2, регулярные выражения являются стандартизованными шаблонами, которые анализируют и манипулируют текстом. Например, регулярное выражение

```
I sent you a che(que|ck) for the gr[ae]y-colou?red alumini?um.
```

соответствует предложениям, использующим либо американские, либо британские орфографические правила.

Регулярные выражения поддерживаются большинством современных языков, хотя одни языки "принимают их ближе к сердцу", чем другие. Они также используются в таких UNIX-командах, как **grep** и **vi**. Регулярные выражения настолько распространены, что их название в оригинальной литературе часто сокращали до слова "regex" (*reg*ular *ex*pressions). О том, как использовать их немалые возможности, написаны уже целые книги.[15]

Сопоставление имен файлов и их раскрытие, выполненное оболочкой в процессе интерпретации таких командных строк, как **wc -l *.pl**, *не является* формой сопоставления с регулярным выражением. Это совсем другая система, именуемая "универсализацией файловых имен с помощью оболочки" (shell globbing), в которой используется другой, причем более простой, синтаксис.

Сами по себе регулярные выражения не являются частью языка написания сценариев, но они настолько полезны, что заслуживают подробного рассмотрения в любом обсуждении "сценарной" темы, а уж тем более в этом разделе.

Процесс сопоставления

Код, который использует регулярное выражение, пытается сопоставить одну заданную текстовую строку с одним заданным шаблоном. Сопоставляемая текстовая строка может иметь очень большой размер и содержать встроенные символы перехода на новую строку. Зачастую регулярное выражение удобно использовать для сопоставления с содержимым целого файла или документа.

Для того чтобы механизм сопоставления мог доложить о благоприятном исходе, шаблон поиска должен целиком совпасть с непрерывной областью исследуемого текста. При этом шаблон может совпасть в любом месте исследуемого текста. Обнаружив успешное совпадение, вычислитель возвращает текст совпадения вместе со списком совпадений для любых частей (подразделов) шаблона, ограниченных специальным образом.

Литеральные символы

В общем случае символы регулярного выражения при сопоставлении должны соответствовать самим себе. Так, шаблон

```
I am the walrus
```

совпадает со строкой "I am the walrus" и только с этой строкой. Поскольку совпадение может быть обнаружено в любом месте исследуемого текста, этот шаблон может дать успешный результат совпадения со строкой

```
I am the egg man. I am the walrus. Koo koo ka-choo!
```

Однако реальное совпадение ограничено фрагментом "I am the walrus". Процесс сопоставления чувствителен к регистру букв.

Специальные символы

В табл. 7.5 собраны значения некоторых распространенных специальных символов, которые могут использоваться в регулярных выражениях (вообще, этих специальных символов гораздо больше).

Многие такие специальные конструкции, как + и |, оказывают влияние на сопоставление подстрок слева или справа. В общем случае подстроку может составлять один символ, или "подшаблон", заключенный в круглые скобки, или класс символов, заклю-

[15] Ссылки на литературу приведены в конце главы.

ченный в квадратные скобки. Однако для символа "|" подстрока распространяется неограниченно как влево, так и вправо. Если вы хотите ограничить область действия вертикальной черты, заключите ее и обе подстроки в их собственный набор круглых скобок. Например, шаблон

```
I am the (walrus|egg man)\.
```

Таблица 7.5. Распространенные специальные символы регулярных выражений

Символ	С чем совпадает или что делает
`.`	Совпадает с любым символом
`[`*символы*`]`	Совпадает с любым символом из заданного набора
`[^`*символы*`]`	Совпадает с любым символом не из заданного набора
`^`	Совпадает с началом строки
`$`	Совпадает с концом строки
`\w`	Совпадает с любым алфавитно-цифровым символом (аналогично набору `[A-Za-z0-9_]`)
`\s`	Совпадает с любым пробельным символом (аналогично набору `[ \f\t\n\r]`)[a]
`\d`	Совпадает с любой цифрой (аналогично набору `[0-9]`)
`|`	Совпадает с элементом либо слева, либо справа
`(`*выражение*`)`	Ограничивает область действия, группирует элементы, позволяя формировать группы совпадений с "захватом" и без "захвата"
`?`	Позволяет одно повторение (или ни одного) предшествующего элемента
`*`	Позволяет множество повторений (или ни одного) предшествующего элемента
`+`	Позволяет одно или больше повторений предшествующего элемента
`{`*n*`}`	Ровно n повторений предшествующего элемента
`{`*min*`, }`	Не менее *min* повторений (обратите внимание на запятую)
`{`*min*`,`*max*`}`	Любое число (от *min* до *max*) повторений

[a]Пробел, символ прогона страницы, табуляции, новой строки или возврата каретки.

даст совпадение либо со строкой "I am the walrus.", либо со строкой "I am the egg man.". Этот пример демонстрирует также способ "экранирования" специальных символов (в данном случае точки) с помощью обратной косой черты (\). Шаблон

```
(I am the (walrus|egg man)\. ?){1,2}
```

совпадает со всеми последующими строками.

- I am the walrus.
- I am the egg man.
- I am the walrus. I am the egg man.
- I am the egg man. I am the walrus.
- I am the egg man. I am the egg man.
- I am the walrus. I am the walrus.

Приведенный выше шаблон также совпадет со строкой "I am the walrus. I am the egg man. I am the walrus.", хотя количество повторений явно ограничено числом два. Дело в том, что шаблон не обязательно должен совпадать полностью с исследуемым текстом. В данном случае после обнаружения совпадений этого регулярного выражения с двумя предложениями дальнейшее его сопоставление прекратилось с объявлением об успеш-

ном завершении. После выполнения условия, заданного в регулярном выражении, уже не важно, что в данном тексте есть еще одно совпадение с шаблоном.

Часто метасимвол регулярного выражения "*" (квантификатор, равный нулю или больше нуля) путают с символом "*", используемым командной оболочкой для универсализации файловых имен. В системе регулярных выражений используйте метасимвол "*" в случае, если для совпадения приемлема любая последовательность символов (включая их отсутствие).

Примеры использования регулярных выражений

В США почтовые индексы (zip-коды) имеют либо пять цифр, либо пять цифр с последующим тире и еще четырьмя цифрами. При создании регулярного выражения для zip-кода необходимо обеспечить сопоставление пятизначного номера. Следующее регулярное выражение отвечает всем этим требованиям.

```
^\d{5}$
```

Символы ^ и $ сопоставляются с началом и концом обрабатываемого текста, не соответствуя при этом реальным символам в тексте; они представляют собой "утверждения нулевой длины" (т.е. это не символы, а лишь "позиции" в тексте). Эти символы гарантируют, что только те текстовые строки, которые состоят ровно из пяти цифр, должны совпадать с регулярным выражением (строки большей длины не должны давать совпадения). Сочетание символов \d обеспечивает совпадение с цифрой, а квантификатор {5} выражает требование совпадения ровно пяти цифр.

Для того чтобы можно было выявить либо пятизначный почтовый индекс, либо расширенный zip-код (zip+4), добавим в качестве необязательной части тире и четыре дополнительные цифры.

```
^\d{5}(-\d{4})?$
```

Круглые скобки объединяют в группу тире и дополнительные цифры, чтобы они считались одной необязательной единицей. Например, это регулярное выражение не даст совпадения с пятизначным почтовым индексом, за которым следует "голое" тире (без последующих цифр). Если тире существует, также должно существовать четырехзначное расширение, в противном случае совпадение зафиксировано не будет.

Рассмотрим классический пример использования регулярного выражения.

```
M[ou]'?am+[ae]r ([AEae]l[- ])?[GKQ]h?[aeu]+([dtz][dhz]?){1,2}af[iy]
```

Оно дает совпадения со многими вариантами произношения имени бывшего лидера Ливии Муаммара Каддафи (Moammar Gadhafi), например с такими, как:

- Muammar al-Kaddafi (BBC);
- Moammar Gadhafi (Associated Press);
- Muammar al-Qadhafi (Al-Jazeera);
- Mu'ammar Al-Qadhafi (U.S. Department of State).

Вы понимаете, почему каждый из приведенных вариантов успешно совпал с шаблоном?[16]

Приведенное выше регулярное выражение также иллюстрирует, насколько быстро могут быть достигнуты пределы удобочитаемости. Многие системы регулярных выра-

[16]Обратите внимание: это регулярное выражение должно быть гибким при сопоставлении. Многие шаблоны, которые не являются законными написаниями, также соответствуют шаблону: например, "Mo'ammer el Qhuuuzzthaf".

жений (включая используемую в Perl) поддерживают опцию x, которая игнорирует литеральные пробельные символы в шаблоне и легитимизирует комментарии, разрешая "растягивать" шаблон и располагать его на нескольких строках. Затем вы можете использовать пробельные символы для выделения логических групп и "прояснения отношений" между ними подобно тому, как это делается в процедурных языках.

```
M [ou] '? a m+ [ae] r     # Имя: Mu'ammar, Moamar и т.д.
\s                        # Пробельный, но не литеральный пробел
(                         # Группа для необязательного префикса-фамилии
   [AEae] l               #    Al, El, al или el
   [-\s]                  #    Тире или пробел
)?
[GKQ] h? [aeu]+           # Начальный слог фамилии: Kha, Qua и т.д.
(                         # Группа для согласных в начале 2-го слога
   [dtz] [dhz]?           #    dd, dh и т.д.
)+                        # Группа может встречаться дважды
af [iy]                   # Последний оператор afi или afy
```

Такой способ описания регулярного выражения, вероятно, слегка облегчит понимание, но все же и он в состоянии замучить потенциальных читателей. Поэтому лучше использовать иерархический подход и строить множество небольших сопоставлений вместо попытки описать все возможные ситуации в одном огромном регулярном выражении.

Захваты

В случае если совпадение происходит, каждый набор круглых скобок становится "группой захвата", которая фиксирует реально совпавший текст. То, как именно эти элементы становятся доступными для вас, зависит от конкретной реализации и контекста. В большинстве случаев доступ к результатам можно получить в виде списка или последовательности пронумерованных переменных.

Поскольку круглые скобки могут быть вложенными, как узнать, чему соответствует каждое совпадение? Ответ простой: совпадения выстраиваются в результате в том же порядке, в котором располагаются открывающие скобки. Количество "захватов" равно количеству открывающих скобок, независимо от роли (или ее отсутствия), которую играла каждая взятая в скобки группа в реальном совпадении. Если взятая в скобки группа не используется (например, при сопоставлении регулярного выражения `Mu('')?ammar` со строкой "Muammar"), соответствующий "захват" будет пустым.

Если обнаружится несколько совпадений группы, в качестве результата возвращается содержимое только последнего совпадения. Например, при шаблоне

```
(I am the (walrus|egg man)\. ?){1,2}
```

и таком варианте обрабатываемого текста

```
I am the egg man. I am the walrus.
```

существуют два результата (по одному для каждого набора круглых скобок).

```
I am the walrus.
walrus
```

Обратите внимание на то, что обе захваченные группы в действительности совпали дважды. Однако был захвачен только последний текст, совпавший с каждым набором круглых скобок.

Жадность, леность и катастрофический поиск с возвратом

Регулярные выражения сопоставляются слева направо. Если каждый компонент шаблона перед переходом к следующему компоненту стремится "захватить" максимально возможную строку, то такая характеристика поведения называется *жадностью* (greediness).

Если анализатор регулярных выражений достигает состояния, из которого сопоставление выполнить уже невозможно, он немного "откатывается" назад от кандидата на совпадение и заставляет одного из "жадных" компонентов отказаться от части текста. Например, рассмотрим регулярное выражение `a*aa` применительно к входному тексту "аааааа".

Вначале анализатор присваивает весь входной текст компоненту `a*`, поскольку этот компонент является "жадным". Когда окажется, что больше букв "а" не осталось, анализатор перейдет к поиску совпадений со следующим компонентом регулярного выражения. Поскольку им является `a` и во входном тексте больше нет ничего, что можно было бы сопоставить с `a`, пора делать "откат". Компонент `a*` вынужден отказаться от одной из букв "а", с которыми уже было зафиксировано совпадение.

Теперь анализатор может сопоставлять компонент `a*a`, но он по-прежнему не может это сделать для последней буквы "а" в шаблоне. Поэтому он снова откатывается и "отнимает" вторую букву "а" из "добычи" компонента `a*`. На этот раз вторая и третья буквы "а" в шаблоне имеют буквы "а" для фиксации совпадения, и на этом обработка текста завершена.

Этот простой пример иллюстрирует некоторые важные общие моменты. Прежде всего, "жадное" сопоставление с "откатами" делает использование таких простых шаблонов, как `<img.*></tr>`, дорогим удовольствием при обработке целых файлов.[17] Действие порции `.*` начинается с сопоставления всего содержимого от первой найденной подстроки `<img` до конца входного текста, и только благодаря повторным откатам область поиска сузится, чтобы "подобраться" к локальным тегам.

Более того, порция `></tr>`, с которой связан этот шаблон, — это *последнее возможное* допустимое вхождение искомого элемента во входном тексте, что, наверняка, совсем не отвечает вашим намерениям. Вероятно, вы хотели отыскать совпадение с подстрокой `<img>`, за которой следует тег `</tr>`. Тогда лучше переписать этот шаблон в виде `<img[^>]*></tr>`, что позволит распространить совпадение с начальными групповыми символами только до конца текущего тега благодаря явно заданной невозможности пересечь границу, обозначенную правой угловой скобкой.

Можно также использовать "ленивые" (в противоположность "жадным") операторные выражения: `*?` вместо `*` и `+?` вместо `+`. Эти варианты квантификаторов ограничивают количество вхождений искомых символов во входном тексте до минимально возможного. Во многих ситуациях эти операторы работают эффективнее и дают результаты, более близкие к желаемым, чем "жадные" варианты.

Однако обратите внимание на то, что "ленивые" квантификаторы (в отличие от "жадных") могут давать различные совпадения; разница состоит не просто в используемой реализации. В нашем HTML-примере "ленивый" шаблон выглядел бы так: `<img.*?></tr>`. Но даже в этом случае элемент `.*?` мог бы стать причиной внесения в результат ненужных нам символов ">", поскольку следующим после тега `<img>` может быть не тег `</tr>`. А такой результат вас, скорее всего, не устроит.

[17] Несмотря на то что в этом разделе в качестве примеров обрабатываемого текста показаны фрагменты HTML-кода, регулярные выражения — не самый подходящий инструмент в данном случае (что не преминули отметить и наши рецензенты). Языки Ruby и Python имеют прекрасные расширения, которые анализируют HTML-документы надлежащим образом. Вы можете получить доступ к интересующим вас порциям с помощью Xpath- или CSS-селекторов. Подробнее о модульных репозиториях соответствующих языков можно узнать, обратившись к странице Википедии, посвященной языку запросов Xpath (XML Path Language) и соответствующим модулям в репозиториях.

Шаблоны с несколькими секциями групповых символов могут "спровоцировать" анализатор регулярных выражений на экспоненциальное поведение, особенно в случае, если порции текста могут совпадать с несколькими шаблонными выражениями, и тем более в случае, если искомый текст в действительности не соответствует шаблону. Эта ситуация не является такой уж необычной, как может показаться, главным образом тогда, когда шаблон сопоставляется с HTML-кодом. Очень часто вам придется искать конкретные теги, за которыми следуют другие теги, возможно, разделенные третьими тегами, и так далее, одним словом, вам придется создавать задание, которое потребует, чтобы анализатор проверил массу возможных комбинаций.

Большой специалист в области регулярных выражений Ян Гойвертс (Jan Goyvaerts) называет этот эффект *катастрофическим откатом* (catastrophic backtracking) и описывает его в своем блоге (за подробностями и некоторыми удачными решениями обращайтесь по адресу: `regular-expressions.info/catastrophic.html`).

Из всего сказанного выше можно сделать такие выводы.

- Если вы можете организовать построчное сопоставление шаблона, а не пофайловое (т.е. с просмотром сразу всего файла), значит, вы значительно уменьшаете риск получения, мягко говоря, низкой производительности.
- Несмотря на то что регулярные выражения являются "жадными" по умолчанию, вам, скорее всего, такие не нужны. Используйте "ленивые" операторы.
- Все экземпляры специальных символов "`.*`" по существу подозрительны и должны быть тщательно исследованы.

7.5. Программирование на языке Python

Python и Ruby — интерпретируемые языки с выраженным объектно-ориентированным уклоном. Оба они широко используются в качестве языков сценариев общего назначения и имеют обширные библиотеки сторонних модулей. Мы обсудим Ruby более подробно в разделе 7.6.

Язык Python предлагает простой синтаксис, который обычно довольно легко отслеживать, даже при чтении кода других людей.

Мы рекомендуем всем системным администраторам свободно владеть языком Python. Это один из лучших среди современных языков системного администрирования и общего назначения. Он также широко поддерживается в качестве связующего средства для использования в других системах (например, в базе данных PostgreSQL и среде разработки Xcode от Apple). Он отлично взаимодействует с интерфейсом прикладного программирования REST и имеет хорошо разработанные библиотеки для машинного обучения, анализа данных и вычислений.

Страсти по Python 3

Python уже успел стать основным языком написания сценариев в мире, когда в 2008 году появилась версия Python 3. В этой версии разработчики решили отказаться от обратной совместимости с Python 2, чтобы можно было реализовать группу скромных, но фундаментальных изменений и исправлений в языке, особенно в области интернационализированной обработки текста.[18]

[18]Точный список изменений в версии Python 3 не является предметом нашего обсуждения, но вы можете найти их описание на странице **`docs.python.org/3.0/whatsnew/3.0.html`**.

К сожалению, развертывание версии Python 3 потерпело фиаско. Обновления языка были вполне разумными, но не обязательными для среднестатистического программиста на Python с существующей базой поддерживаемого кода. В течение долгого времени разработчики сценариев избегали использования Python 3, потому что их любимые библиотеки не поддерживали его, в то время как авторы библиотек не поддерживали Python 3, потому что их клиенты все еще использовали Python 2.

Даже в самых благоприятных обстоятельствах трудно подтолкнуть большое и взаимозависимое сообщество пользователей к такого рода разрыву. В случае Python 3 борьба шла в течение большей части десятилетия. Однако по состоянию на 2017 год эта ситуация, по-видимому, меняется.

Библиотеки совместимости, которые позволяют одному и тому же Python-коду работать в любой версии языка, в какой-то степени облегчили переход. Но даже сейчас версия Python 3 остается менее распространенной, чем Python 2.

На момент написания этой статьи книги `py3readiness.org` сообщает, что только 17 из 360 лучших библиотек Python остаются несовместимыми с Python 3. Однако длинный хвост непереносимого программного обеспечения хорошо отрезвляет: только немногим более 25% библиотек, хранящихся на сайте `pypi.python.org` (индекс Python Package, или PyPI) работает под управлением версии Python 3.[19] Конечно, многие из этих проектов устарели и больше не поддерживаются, но 25% по-прежнему является относительно небольшим числом.

Python 2 или Python 3?

Замедленный переход привел к тому, что версии Pythons 2 и 3 стали рассматриваться как отдельные языки. Вам не нужно делать взаимоисключающий выбор — в любой системе можно запускать обе версии одновременно без конфликтов.

Все наши иллюстративные системы устанавливают версию Python 2 по умолчанию, обычно как **`/usr/bin/python2`** с символической ссылкой из **`/usr/bin/python`**. Python 3 может быть установлен как отдельный пакет; двоичный код называется **`python3`**.

RHEL 

Проект Fedora работает над тем, чтобы сделать Python 3 своей версией по умолчанию, а системы Red Hat и CentOS намного отстают и даже не определяют предварительно построенный пакет для версии Python 3. Однако вы можете выбрать один из репозиториев EPEL Fedora (Extra Packages for Enterprise Linux). Для получения инструкций по доступу к этому репозиторию, обратитесь в раздел FAQ на сайте `fedoraproject.org/wiki/EPEL` для получения инструкций по доступу к этому репозиторию. Его легко настроить, но точные команды зависят от версии.

Если вы впервые приступаете к разработке сценариев или к программированию на языке Python, имеет смысл перейти непосредственно к Python 3. Именно его синтаксис мы демонстрируем в этой главе, хотя на самом деле разница между Python 2 и Python 3 в наших простых примерах заключается только в строках `print`.

Для существующего программного обеспечения вы можете использовать любую версию Python, которую предпочитает программное обеспечение. Если ваш выбор сложнее, чем просто новый или старый код, обратитесь к вики-системе Python на `wiki.python.org/moin/Python2orPython3`, в которой содержится отличная коллекция задач, решений и рекомендаций.

[19] Для получения актуальной статистики см. сайт **`caniusepython3.com`**.

Краткое введение в язык Python

Для более полного ознакомления с языком Python мы рекомендуем 5-е издание двухтомника *Изучаем Python* Марка Лутца (Mark Lutz). Полную ссылку можно найти в разделе "Литература".

Приведем короткий сценарий "Hello, world!", который нужно сохранить в файле **helloworld**.

```
#!/usr/local/bin/python3
print ("Hello, world!")
```

Чтобы запустить его, установите бит выполнения или вызовите интерпретатор **python3** напрямую.

```
$ chmod +x helloworld
$ ./helloworld
Hello world!
```

Самый крупный разрыв Python с традиционным стилем программирования заключается в том, что отступы имеют логический смысл. Для разграничения блоков в языке Python не используются фигурные и квадратные скобки или ключевые слова `begin` и `end`. Блоки автоматически формируются из выражений, находящихся на одном уровне отступов. Точный стиль отступов (пробелы или табуляция, а также глубина отступов) значения не имеет.

Создание блоков на языке Python лучше всего показано на примере. Рассмотрим простое выражение `if-then-else`:

```
import sys

a = sys.argv[1]

if a == "1":
       print ('a равно 1')
       print ('Это все еще раздел then инструкции if.')
else:
       print ('a это', a)
       print ('Это все еще раздел else инструкции if.')
print ('Это - вывод после инструкции if.')
```

Первая строка импортирует модуль `sys`, который содержит массив `argv`. Две ветки инструкции `if` состоят из двух строк, каждая с отступом до одного уровня. (Двоеточие в конце строки обычно обозначает начало блока и связано с отступом, который следует за ним.) Окончательный оператор печати находится вне контекста оператора `if`.

```
$ python3 blockexample 1
a равно 1
Это все еще раздел then инструкции if.
Это - вывод после инструкции if.
$ python3 blockexample 2
a это 2
Это все еще раздел else инструкции if.
Это - вывод после инструкции if.
```

Соглашение об отступах в языке Python является менее гибким средством форматирования кода, но оно уменьшает беспорядок, связанный с использованием скобок и точек с запятой. Это требует перестройки от тех, кто привык к традиционным разделителям, но большинство людей в конечном итоге приходят к выводу, что им это нравится.

Функция `print` в языке Python принимает произвольное количество аргументов. Она вставляет пробел между каждой парой аргументов и автоматически передает новую строку. Вы можете подавлять или изменять эти символы, добавляя опции `end =` или `sep =` в конец списка аргументов.

Например, строка

```
print ("один", "два", "три", sep = "-", end = "! \n")
```

производит вывод

```
один-два-три!
```

Комментарии начинаются с символа # и простираются до конца строки, как и в языках **sh**, Perl и Ruby.

Можно разделить длинные строки, поставив обратные косые черты в местах разрыва строк. Когда вы это делаете, значение имеют только отступы первой строки. Вы можете отступать в строках продолжения настолько далеко, насколько вам нравится. Строки с несбалансированными круглыми, квадратными или фигурными скобками автоматически сигнализируют о продолжении даже в отсутствие обратных косых черт, но вы можете включить обратную косую черту, если это упрощает структуру кода.

Некоторые операции вырезания и вставки преобразуют символы табуляции в пробелы, и, если вы не знаете, чего хотите, это может привести вас в бешенство. Золотое правило гласит: "Никогда не смешивайте табуляцию и пробелы; используйте для отступов или то, или другое". Многие программные средства делают традиционное предположение о том, что табуляции эквивалентны восьми пробелам, что является слишком большим отступом для читаемого кода. Большинство специалистов из сообщества Python, похоже, предпочитают пробелы и четырехсимвольные отступы.

Впрочем, большинство текстовых редакторов имеют варианты, которые могут помочь сохранить ваше душевное равновесие, либо путем запрета табуляции в пользу пробелов, либо путем разного отображения пробелов и табуляции. В крайнем случае вы можете перевести табуляцию в пробелы с помощью команды **expand**.

Объекты, строки, числа, списки, словари, кортежи и файлы

Все типы данных в языке Python являются объектами, и это дает им большую мощность и гибкость, чем в большинстве других языков.

В языке Python списки заключены в квадратные скобки и индексируются с нуля. Они по существу похожи на массивы, но могут содержать объекты любого типа.[20]

В языке Python есть кортежи, которые, по сути, являются неизменными списками. Кортежи эффективнее, чем списки, и полезны для представления постоянных данных. Синтаксис кортежей такой же, как и для списков, за исключением того, что в качестве разделителей используются круглые, а не квадратные скобки. Поскольку выражение `(thing)` выглядит как простое алгебраическое выражение, кортежи, содержащие только один элемент, должны содержать запятую, играющую роль маркера, чтобы устранить неоднозначность: `(thing, )`.

Приведем несколько основных переменных и типов данных в языке Python:

```
name = 'Гвен'
rating = 10
characters = ['Губка Боб', 'Патрик', 'Сквидвард']
```

[20]Однородный и более эффективный тип массива реализован в модуле **array**, но для большинства целей мы рекомендуем использовать списки.

```
elements = ('литий', 'углерод', 'бор')
print ("Имя:\t%s\nРейтинг:\t%d" % (name, rating))
print ("Персонажи:\t%s" % characters)
print ("Кумир:\t%s" % characters[0])
print ("Элементы:\t%s" % (elements, ))
```

В этом примере получается следующий результат:

```
$ python3 objects
Имя:    Гвен
Рейтинг:        10
Персонажи:      ['Губка Боб', 'Патрик', 'Сквидвард']
Кумир:  Губка Боб
Элементы:       ('литий', 'углерод', 'бор')
```

Обратите внимание на то, что стандартное преобразование строки для типов списка и кортежей представляет их в том виде, в котором они были введены в исходном коде.

Переменные в языке Python не являются синтаксически помеченными или не имеют объявления типа, но объекты, к которым они относятся, имеют базовый тип. В большинстве случаев Python не конвертирует типы автоматически, но отдельные функции или операторы могут это сделать. Например, вы не можете объединить строку и число (с помощью оператора +) без явного преобразования числа в его строковое представление. Тем не менее операторы и инструкции форматирования приводят все к строковой форме.

Как показано выше, каждый объект имеет строковое представление. Словари, списки и кортежи рекурсивно формируют свои строковые представления, создавая их составные элементы и комбинируя эти строки с соответствующей пунктуацией.

Оператор форматирования строк `%` очень похож на функцию `sprintf` из языка C, но его можно использовать везде, где может появляться строка. Это двоичный оператор, который берет строку слева и значения, которые нужно вставить справа. Если необходимо вставить более одного значения, значения должны быть представлены как кортеж.

Словарь в языке Python (также известный как хеш или ассоциативный массив) представляет собой набор пар ключ-значение. Хеш можно представить как массив, чьи индексы (ключи) являются произвольными значениями; они не должны быть цифрами. Однако на практике цифры и строки являются общими ключами.

Словарные литералы заключены в фигурные скобки, каждая пара ключей-значений разделяется двоеточием. При использовании словари работают так же, как списки, за исключением того, что индексы (ключи) могут быть объектами, отличными от целых чисел.

```
ordinal = { 1 : 'первый', 2 : 'второй', 3 : 'третий' }
print("Упорядоченные словарные литералы: ", ordinal)
print("Первый литерал:", ordinal[1])
```

```
$ python3 dictionary
Упорядоченные словарные литералы:  {1: 'первый', 2: 'второй', 3: 'третий'}
Первый литерал: первый
```

Python обрабатывает открытые файлы как объекты со ассоциированными методами. В соответствии со своим названием метод `readline` читает одну строку, поэтому в приведенном ниже примере считываются и выводятся на печать две строки из файла **/etc/passwd**.

```
f = open('/etc/passwd', 'r')
print(f.readline(), end="")
print(f.readline(), end="")
f.close()
```

```
$ python3 fileio
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
```

Переход на новую строку в конце вызовов функции `print` подавляется с помощью параметра `end = ""`, потому что каждая строка уже содержит символ перехода на новую строку из исходного файла. Python не разделяет их автоматически.

Пример проверки ввода

В следующем сценарии показана общая схема проверки ввода в языке Python. Он также демонстрирует определение функций и использование аргументов командной строки, а также пару других специфических особенностей.

```
import sys
import os

def show_usage(message, code = 1):
    print(message)
    print("%s: исходный_каталог целевой_каталог" % sys.argv[0])
    sys.exit(code)

if len(sys.argv) != 3:
    show_usage("Нужны 2 аргумента; вы ввели один %d" % (len(sys.argv) - 1))
elif not os.path.isdir(sys.argv[1]):
    show_usage("Исходный каталог не найден")
elif not os.path.isdir(sys.argv[2]):
    show_usage("Целевой каталог не найден")

source, dest = sys.argv[1:3]
print("Исходный каталог:", source)
print("Целевой каталог", dest)
```

Помимо импорта модуля `sys`, мы также импортируем модуль `os` для доступа к процедуре `os.path.isdir`. Обратите внимание на то, что импорт не сокращает ваш доступ к любым символам, определенным модулями; вы должны использовать полностью квалифицированные имена, которые начинаются с имени модуля.

Определение подпрограммы `show_usage` предоставляет значение по умолчанию для кода выхода, если вызывающий объект явно не указывает этот аргумент. Поскольку все типы данных являются объектами, аргументы функции эффективно передаются по ссылке.

В первой позиции список `sys.argv` содержит имя сценария, поэтому его длина больше, чем количество аргументов командной строки, которые были фактически предоставлены. Форма `sys.argv[1:3]` — это список фрагментов. Любопытно, что срезы не включают элемент в дальнем конце указанного диапазона, поэтому этот фрагмент включает только элементы `sys.argv[1]` и `sys.argv[2]`. Чтобы включить второй и последующие аргументы, можно просто написать `sys.argv[1:]`.

Как и в **sh**, в языке Python есть специальное условие "else if"; ключевым словом является `elif`. Но в нем нет явной инструкции `case` или `switch`.

Параллельное присваивание переменных `source` и `dest` немного отличается от некоторых языков тем, что сами переменные не входят в список. Python допускает параллельные назначения в любой форме.

В языке Python используются одинаковые операторы сравнения для числовых и строковых значений. Оператор сравнения "не равен" имеет вид `!=`, но в языке нет

унарного оператора !; для этой цели используется ключевое слово not. Существуют также булевы операторы and и or.

Циклы

В приведенном ниже фрагменте используется конструкция for...in для обхода элементов в диапазоне от 1 до 10.

```
for counter in range(1, 10):
    print(counter, end=" ")
print()                          # Последний переход на новую строку
```

Как и в срезе массива в предыдущем примере, правая конечная точка диапазона фактически в него не входит. Вывод включает только числа от 1 до 9:

```
1 2 3 4 5 6 7 8 9
```

Это единственный тип цикла в языке Python, но это очень мощный инструмент. В языке Python есть несколько функций, которые отличают его конструкцию for от других языков.

- В числовых диапазонах нет ничего особенного. Любой объект может поддерживать итерационную модель Python как обычно. Можно перебирать строку (по символам), список, файл (по символам, строкам или блокам), список и т. д.
- Итераторы могут принимать несколько значений, и можно иметь несколько переменных цикла. Присваивание в верхней части каждой итерации действует так же, как обычные множественные присваивания в языке Python. Эта функция особенно хороша для обхода словарей.
- В циклах for и while в конце могут быть разделы else, которые выполняются только в том случае, если цикл завершается нормально, в отличие от выхода из цикла по инструкции break. Поначалу это кажется неестественным, но такой подход позволяет довольно элегантно обрабатывать некоторые варианты использования.

В приведенном ниже примере сценарий принимает регулярное выражение в командной строке и сопоставляет его с списком гномов из сказки о Белоснежке, а также с цветами их костюмов. На экран выводится первое совпадение с частями, которые соответствуют регулярному выражению, окруженному символами подчеркивания.

```
import sys
import re

suits = {
    'Bashful':'yellow', 'Sneezy':'brown', 'Doc':'orange', 'Grumpy':'red',
    'Dopey':'green', 'Happy':'blue', 'Sleepy':'taupe'
}
pattern = re.compile("(%s)" % sys.argv[1])

for dwarf, color in suits.items():
    if pattern.search(dwarf) or pattern.search(color):
        print("%s's dwarf suit is %s." %
              (pattern.sub(r"_\1_", dwarf), pattern.sub(r"_\1_", color)))
        break
else:
    print("No dwarves or dwarf suits matched the pattern.")
```

Ниже приведен вариант вывода:

```
$ python3 dwarfsearch '[aeiou]{2}'
Sl_ee_py's dwarf suit is t_au_pe.

$ python3 dwarfsearch 'ga|gu'
No dwarves or dwarf suits matched the pattern.
```

Присваивание переменной `suits` демонстрирует синтаксис Python для шифрования символьных словарей. Метод `suits.items()` является итератором для пар ключ-значение — обратите внимание на то, что на каждой итерации мы извлекаем как имя гнома, так и цвет его костюма. Если вы хотите перебирать только ключи, вы можете просто написать `dwarf in suits`.

Язык Python реализует обработку регулярных выражений с помощью модуля `re`. В языке нет встроенных функций для работы с регулярными выражениями, поэтому обработка регулярных выражения в языке Python немного более ограниченная, чем, скажем, в языке Perl. Здесь шаблон регулярного выражения изначально компилируется из первого аргумента командной строки, окруженного скобками (для формирования группы захвата). Затем строки тестируются и модифицируются поисковыми и вспомогательными методами объекта `regex`. Вы также можете вызвать `re.search` и другие функции непосредственно, предоставляя регулярное выражение для использования в качестве первого аргумента.

Символы `\1` в строке подстановки представляют собой обратную ссылку на содержимое первой группы захвата. Странно выглядящий префикс `r`, который предшествует строке подстановки (`r"_\1_"`), подавляет нормальную подстановку управляющих символов в строковых константах (`r` обозначает "raw"). Без этого шаблон замены состоял бы из двух символов подчеркивания, окружающих символ с цифровым кодом `1`.

Следует отметить, что в словарях нет определенного порядка обхода. Если вы запустите поиск гномов во второй раз, то можете получить другой ответ:

```
$ python3 dwarfsearch '[aeiou] {2}'
Dopey's dwarf suit is gr_ee_n.
```

7.6. Программирование на языке Ruby

Язык Ruby, разработанный и поддерживаемый японским разработчиком Юкихиро Мацумото (Yukihiro "Matz" Matsumoto), обладает многими функциями, общими с языком Python, включая широко распространенный подход "Все является объектом". Первоначально выпущенный в середине 1990-х годов, язык Ruby не получил известности, пока десятилетия спустя не появилась платформа веб-разработки Rails.

В умах людей язык Ruby по-прежнему тесно связан с сетью, но в самом языке нет ничего веб-специфичного. Он хорошо подходит для написания сценариев общего назначения. Однако язык Python, вероятно, все же лучший выбор для основного языка сценариев, хотя бы из-за его более широкой популярности.

Хотя язык Ruby во многом эквивалентен языку Python, он является более гибким. Например, классы Ruby остаются открытыми для модификации другим программным обеспечением, и сообщество программистов на языке Ruby не возражает против расширений, которые изменяют стандартную библиотеку.

Язык Ruby нравится тем, у кого есть склонность к "синтаксическому сахару", т.е. особенностям, которые на самом деле не меняют базовый язык, но позволяют более четко и четко выражать код.

В среде Rails, например, строка

```
due_date = 7.days.from_now
```

создает объект класса `Time` без ссылки на имена любых связанных с временем классов или выполнения явных арифметических операций над датой и временем. Платформа Rails определяет объект `days` как расширение класса `Fixnum`, представляющего целые числа. Этот метод возвращает объект `Duration`, который действует как число; используется в качестве значения, эквивалентного 604800, т.е. количеству секунд в семи днях. Если проверить его в отладчике, он опишет себя как "7 дней".[21]

Язык Ruby упрощает разработку "доменных языков" (например, DSL), мини-языков, которые на самом деле являются подмножеством языка Ruby, но которые рассматриваются как специализированные системы конфигурации. Например, язык Ruby DSL используется в средствах настройки Chef и Puppet.

🕮 Дополнительную информацию об инструментах Chef и Puppet см. в главе 23.

Инсталляция

В некоторых системах язык Ruby установлен по умолчанию, а в некоторых — нет. Тем не менее он всегда доступен как пакет, часто в нескольких версиях.

На сегодняшний день (версия 2.3) язык Ruby поддерживает относительно хорошую совместимость со старым кодом. В отсутствие конкретных предупреждений обычно лучше всего установить самую последнюю версию.

К сожалению, большинство системных пакетов отстают на несколько выпусков от релизов языка Ruby. Если ваша библиотека пакетов не включает текущую версию (проверьте сайт `ruby-lang.org`, чтобы определить, так ли это), установите самую свежую версию через RVM; не пытайтесь делать это сами.

🕮 Подробнее об RVM см. в разделе 7.7.

Краткое введение в язык Ruby

Поскольку языки Ruby и Python настолько похожи, некоторые фрагменты кода на языке Ruby могут показаться похожими на фрагменты кода из раздела о языке Python, приведенные ранее в этой главе.

```
#!/usr/bin/env ruby

print "Hello, world!\n\n"

name = 'Gwen'
rating = 10
characters = [ 'SpongeBob', 'Patrick', 'Squidward' ]
elements = { 3 => 'lithium', 7 => 'carbon', 5 => 'boron' }

print "Name:\t", name, "\nRating:\t", rating, "\n"
print "Characters:\t#{characters}\n"
print "Elements:\t#{elements}\n\n"

element_names = elements.values.sort!.map(&:upcase).join(', ')
print "Element names:\t", element_names, "\n\n"

elements.each do |key, value|
    print "Atomic number #{key} is #{value}.\n"
end
```

[21] Эта форма полиморфизма является общей для языков Ruby и Python. Его часто называют "утиной типизацией": если объект ходит как утка и крякает как утка, вам не нужно беспокоиться о том, действительно ли это утка.

Вывод выглядит следующим образом:

```
Hello, world!
Name:         Gwen
Rating:       10
Characters:   ["SpongeBob", "Patrick", "Squidward"]
Elements:     {3=>"lithium", 7=>"carbon", 5=>"boron"}

Element names: BORON, CARBON, LITHIUM

Atomic number 3 is lithium.
Atomic number 7 is carbon.
Atomic number 5 is boron.
```

Как и Python, язык Ruby использует скобки для разграничения массивов и фигурные скобки для разделения словарных литералов. (В языке Ruby они называются "хешами".) Оператор `=>` отделяет каждый хеш-ключ от его соответствующего значения, а пары ключ-значение отделяются друг от друга запятыми. В языке Ruby нет кортежей.

Функция `print` в языке Ruby — это функция (или, точнее, глобальный метод), как и в языке Python 3. Однако, если вы хотите использовать переход на новую строку, вы должны явно указать это.[22] Кроме того, круглые скобки, обычно встречающиеся вокруг аргументов в вызовах функций, в Ruby являются необязательными. Разработчики обычно не включают их, если они не помогают прояснить код или устранить неоднозначность. (Обратите внимание на то, что некоторые из вызовов функции печати включают в себя несколько аргументов, разделенных запятыми.)

В нескольких случаях мы использовали фигурные скобки `#{}` скобки для интерполяции значений переменных, заключенные в строках с двумя кавычками. Такие скобки могут содержать произвольный код Ruby; любое значение, созданное кодом, автоматически преобразуется в тип строки и вставляется во внешнюю строку. Вы также можете конкатенировать строки с помощью оператора +, но интерполяция обычно более эффективна.

Строка, вычисляющая `element_names`, иллюстрирует еще несколько возможностей языка Ruby:

```
element_names = elements.values.sort! .map (&: upcase) .join (',')
```

Это серия вызовов методов, каждый из которых работает с результатом, возвращаемым предыдущим методом, подобно конвейеру. Например, метод `values` объекта `elements` создает массив строк, которые затем сортируются в алфавитном порядке функцией `sort!`.[23] Метод `map` класса `Array` вызывает метод `upcase` для каждого элемента, а затем снова собирает все результаты в новый массив. Наконец, метод `join` объединяет элементы этого массива, чередующиеся с запятыми, для создания строки.

Блоки

В предыдущем коде текст между `do` и `end` представляет собой блок, известный в других языках как лямбда-функция, замыкание или анонимная функция.[24]

[22] Есть также функция **puts**, которая добавляет символы перехода на новую строку автоматически, но она, возможно, слишком изощренная. Если вы попытаетесь самостоятельно добавить символ перехода на новую строку, функция **puts** не будет вставлять эти символы автоматически.

[23] Восклицательный знак в имени функции **sort!** предупреждает вас, что при использовании этого метода нужно быть осторожным. Это не синтаксическое правило, а просто часть имени метода. В данном случае функция **sort!** сортирует массив на месте. Существует также метод **sort** без восклицательного знака, который возвращает элементы в новом отсортированном массиве.

[24] Фактически в языке Ruby существуют три объекта этого общего типа, известные как блоки, процедуры и лямбда. Различия между ними незначительны и не важны для этого обзора.

```
elements.each do |key, value|
   print "Atomic number #{key} is #{value}.\n"
end
```

Данный блок принимает два аргумента, которые он называет `key` и `value`, и выводит их значения на экран.

Оператор `each` выглядит как функция языка, но это всего лишь метод, определяемый хешами. Оператор `each` принимает блок как аргумент и вызывает его один раз для каждой пары ключ-значение, содержащей хеш. Этот тип итерационной функции, используемой в сочетании с блоком, очень характерен для кода на языке Ruby. Имя `each` является стандартным именем обобщенных итераторов, но многие классы определяют более конкретные версии, такие как `each_line` или `each_character`.

В языке Ruby есть альтернативный синтаксис для блоков, в котором в качестве разделителей используются фигурные скобки, а не `do...end`. Он означает точно то же самое, но больше похож на часть выражения. Например,

```
characters.map {| c | c.reverse} # ["boBegnopS", "kcirtaP", "drawdiuqS"]
```

Эта форма функционально идентична `character.map(&:reverse)`, но вместо того, чтобы просто указывать методу `map`, какой метод вызывать, мы включили явный блок, вызывающий обратный метод.

Значение блока — это значение последнего выражения, которое оно вычисляет до завершения. Удобно, что почти все в языке Ruby является выражением (т.е. "фрагментом кода, который может быть выполнен для создания значения"), включая структуры управления, такие как `case` (аналог конструкции `switch` в большинстве языков) и `if-else`. Значения этих выражений отражают значение, полученное в зависимости от того, какой из разделов `case` или ветви инструкции `if-else` был выполнен.

Блоки имеют много применений, помимо итераций. Они позволяют одной функции выполнять процедуры настройки и удаления от имени другого раздела кода, поэтому они часто представляют собой многоэтапные операции, такие как транзакции баз данных или операции с файловой системой.

Например, следующий код открывает файл `/etc/passwd` и выводит строку, определяющую учетную запись `root`:

```
open '/etc/passwd', 'r' do |file|
   file.each_line do |line|
       print line if line.start_with? 'root:'
   end
end
```

Функция `open` открывает файл и передает его объект `IO` во внешний блок. После того как блок завершит работу, файл откроется автоматически. Нет необходимости в отдельной операции закрытия (хотя она существует, если вы хотите ее использовать), и файл закрывается независимо от того, как завершается внешний блок.

Применяемая здесь постфиксная конструкция `if` может быть знакома тем, кто использовал язык Perl. Это хороший способ выразить простые условные обозначения без сокрытия основного действия. Здесь сразу видно, что внутренний блок представляет собой цикл, который выводит некоторые строки.

В случае, если структура аргумента `line` функции `print` не ясна, в нее снова включаются необязательные круглые скобки. Оператор `if` имеет самый низкий приоритет и использует один метод вызова для обоих вариантов:

```
print (line) if line.start_with? ('root:')
```

Как и в случае метода `sort!`, знак вопроса представляет собой соглашение об именах методов, возвращающих логические значения.

Синтаксис для определения именованной функции несколько отличается от синтаксиса для блока.

```
def show_usage(msg = nil)
   STDERR.puts msg if msg
   STDERR.puts "Usage: #{$0} filename ..."
exit 1
end
```

Скобки все еще являются необязательными, но на практике они всегда отображаются в этом контексте, если функция не принимает аргументов. Здесь аргумент `msg` по умолчанию равен нулю.

Глобальная переменная `$0` является довольно загадочной и содержит имя, по которому вызывается текущая программа. (Традиционно этим именем является первый аргумент массива `argv`. Однако по соглашению в языке Ruby `ARGV` содержит только фактические аргументы командной строки.)

Как и в языке C, вы можете обрабатывать небулевые значения, как если бы они были булевыми, что показано здесь в форме `if msg`. Однако отображение в языке Ruby для этого преобразования немного необычное: все, кроме `nil` и `false`, считается истинным. В частности, `0` — это истинное значение. (На практике эта возможность часто оказывается удобной.)

Символы и хеши опций

В языке Ruby широко используется необычный тип данных, называемый *символом* и обозначенный двоеточием, например `::example`. О символах можно думать как о непреложных строках. Они обычно используются как ярлыки или известные хеш-ключи. Внутренне язык Ruby реализует их как числа, поэтому они быстро хешируются и сравниваются.

Символы так часто используются в качестве хеш-ключей, что в версии Ruby 2.0 определили альтернативный синтаксис для хеш-литералов, чтобы уменьшить количество знаков препинания. Стандартный хеш

```
h = { :animal => 'cat', :vegetable => 'carrot', :mineral => 'zeolite' }
```

можно написать в стиле Ruby 2.0 следующим образом:

```
h = { animal: 'cat', vegetable: 'carrot', mineral: 'zeolite' }
```

Вне этого хеш-литерального контекста символы сохраняют свои префиксы `:` везде, где они появляются в коде. Например, вот как получить конкретные значения из хеша:

```
healthy_snack = h[:vegetable] # 'carrot'
```

В языке Ruby принято своеобразное, но мощное соглашение для обработки параметров в вызовах функций. Если вызываемая функция запрашивает такое поведение, язык Ruby собирает завершающие аргументы вызова, которые напоминают хешированные пары, в новый хеш. Затем он передает этот хеш функции в качестве аргумента. Например, в выражении, допустимом на платформе Rails,

```
file_field_tag :upload, accept: 'application/pdf', id: 'commentpdf'
```

функция `file_field_tag` получает только два аргумента — символ `:upload` и хеш, содержащий ключи `:accept` и `:id`. Поскольку хеши не имеют жесткого порядка, неважно, в каком порядке появляются параметры.

Этот тип гибкой обработки аргументов проявляется в стандарте языка Ruby и другими способами. Библиотеки Ruby, включая стандартную библиотеку, как правило, делают все возможное, чтобы принять максимально широкий диапазон входных данных. Скаляры, массивы и хеши часто являются равноправными аргументами, и многие функции можно вызывать с помощью блоков или без них.

Регулярные выражения в языке Ruby

В отличие от Python, язык Ruby имеет немного "синтаксического сахара" для работы с регулярными выражениями. Язык Ruby поддерживает традиционную `/.../` нотацию для литералов регулярных выражений, а содержимое может содержать управляющие последовательности символов `#{}`, похожие на строки с двумя кавычками.

Кроме того, в языке Ruby определен оператор `=~` (и его отрицание `!~`) для проверки соответствия между строкой и регулярным выражением. Он возвращает либо индекс первого совпадения, либо `nil`, если нет совпадения.

```
"Hermann Hesse" =~ /H[aeiou]/ # => 0
```

Для того чтобы получить доступ к компонентам соответствия, явно вызовите метод `match` регулярного выражения. Он возвращает либо `nil` (если соответствий нет), либо объект, к которому можно получить доступ в виде массива компонентов.

```
if m = /(^H\w*)\s/.match("Heinrich Hoffmeyer headed this heist")
puts m[0] # 'Heinrich'
end
```

Рассмотрим предыдущую программу для определения цвета костюма гнома на языке Ruby.

```
suits = {
   Bashful: 'yellow', Sneezy: 'brown', Doc: 'orange', Grumpy: 'red',
   Dopey: 'green', Happy: 'blue', Sleepy: 'taupe'
}

abort "Usage: #{$0} pattern" unless ARGV.size == 1
pat = /(#{ARGV[0]})/

matches = suits.lazy.select {|dwarf, color| pat =~ dwarf || pat =~ color}

if matches.any?
   dwarf, color = matches.first
   print "%s\'s dwarf suit is %s.\n" %
         [ dwarf.to_s.sub(pat, '_\1_'), color.sub(pat, '_\1_') ]
else
   print "No dwarves or dwarf suits matched the pattern.\n"
end
```

Метод `select`, примененный к коллекции, создает новую коллекцию, включающую только те элементы, для которых предоставленный блок имеет значение как `true`. В этом случае совпадения представляют собой новый хеш, содержащий только пары, для которых либо ключ, либо значение соответствует шаблону поиска. Поскольку мы применили ленивый вызов (`lazy`), фильтрация на самом деле не произойдет, пока мы не попытаемся извлечь значения из результата. Фактически этот код проверяет столько пар, сколько необходимо для поиска соответствия.

Вы заметили, что оператор `=~` сопоставления с образцом использовался на символах, которые представляют имена гномов? Он работает, потому что оператор `=~` достаточно умен, чтобы перед сопоставлением преобразовать символы в строки. К сожалению, при

использовании шаблона подстановки мы должны явно выполнить это преобразование (с помощью метода `to_s`); метод `sub` определен только для строк, поэтому нам нужна настоящая строка для его вызова.

Обратите внимание также на параллельное присваивание гнома и цвета. Метод `matches.first` возвращает двухэлементный массив, который Ruby распаковывает автоматически.

Оператор `%` для строк работает аналогично такому же оператору в языке Python; это версия функции `sprintf` в языке Ruby. Здесь есть два компонента для заполнения, поэтому мы передаем значения как двухэлементный массив.

Язык Ruby как фильтр

Язык Ruby можно использовать без сценария, помещая изолированные выражения в командную строку. Это простой способ сделать быстрые преобразования текста (правда, язык Perl по-прежнему намного лучше справляется с этой задачей).

Используйте параметры командной строки **`-p`** и **`-e`** для циклического обхода потока **`STDIN`**, выполните простое выражение для каждой строки (представленное как переменная `$_`) и распечатайте результат. Например, следующая команда переводит строку **`/etc/passwd`** в верхний регистр:

```
$ ruby -pe '$_.tr!("a-z", "A-Z")' /etc/passwd
NOBODY:*:-2:-2:UNPRIVILEGED USER:/VAR/EMPTY:/USR/BIN/FALSE
ROOT:*:0:0:SYSTEM ADMINISTRATOR:/VAR/ROOT:/BIN/SH
...
```

Команда **`ruby -a`** включает режим автоматического разделения, отделяющий входные строки от полей, которые хранятся в массиве с именем `$F`. По умолчанию разделителем является пробел, но вы можете установить другой шаблон разделителя с помощью опции **`-F`**.

Режим разделения удобен для использования в сочетании с параметром **`-p`** или его вариантом **`-n`**, который не подразумевает автоматического вывода на экран. В приведенной ниже команде конструкция **`ruby -ane`** используется для создания версии файла **`passwd`**, которая включает только имена пользователей и оболочки.

```
$ ruby -F: -ane 'print $F[0], ":", $F[-1]' /etc/passwd
nobody:/usr/bin/false
root:/bin/sh
...
```

Только бесстрашный программист может использовать параметр **`-i`** в сочетании с параметром **`-pe`** для редактирования файлов на месте. Язык Ruby читает содержимое файлов, представляет их строки для редактирования и сохраняет результаты в исходные файлы. Вы можете задать параметр **`-i`**, который сообщает языку Ruby, как создать резервную копию исходной версии каждого файла. Например, **`-i.bak`** создает копию файла **`passwd`** с именем **`passwd.bak`**. Остерегайтесь! Если вы не создадите резервный шаблон, вы вообще не получите резервные копии. Обратите внимание на то, что между параметром **`-i`** и суффиксом нет пробела.

7.7. Управление библиотекой и средой для Python и Ruby

Языки имеют много одинаковых вопросов управления пакетами и версиями и часто решают их аналогичным образом. Поскольку языки Python и Ruby в этой области похожи друг на друга, мы обсуждаем их в этом разделе вместе.

Поиск и установка пакетов

Самое основное требование — обеспечить простой и стандартизированный способ обнаружения, получения, установки, обновления и распространения дополнительного программного обеспечения. У языков Ruby и Python есть централизованные хранилища для этой цели, у Ruby — на сайте `rubygems.org`, в у Python — на сайте `pypi.python.org`.

В мире Ruby пакеты называются "драгоценными камнями" (gems), а команда для управлениям пакетами также называется **`gem`**. Команда **`gem search regex`** показывает доступные пакеты с соответствующими именами, а **`gem install`** *`имя_пакета`* загружает и инсталлирует пакет. С помощью параметра **`--user-install`** можно инсталлировать приватную копию вместо изменения модификации набора пакетов в системе.

Эквивалент в языке Python называется **`pip`** (**`pip2`** или **`pip3`**, в зависимости от того, какие версии Python установлены). Не все системы включают его по умолчанию. Системы, которые этого не делают, обеспечивают доступ к нему как к отдельному пакету на уровне операционной системы. Как и в случае с пакетами языка Ruby, основными командами являются **`pip search`** и **`pip install`**. Опция **`--user`** устанавливает пакеты в ваш домашний каталог.

Утилиты **`gem`** и **`pip`** понимают зависимости между пакетами, по крайней мере на базовом уровне. Когда вы устанавливаете пакет, вы неявно запрашиваете, чтобы все пакеты, от которых он зависит, также были установлены (если они еще не инсталлированы).

В базовой среде Ruby или Python может быть установлена только одна версия пакета. Если вы переустановите или обновите пакет, старая версия будет удалена.

У вас часто есть выбор для установки пакета **`gem`** или **`pip`** с помощью стандартного языкового механизма (**`gem`** или **`pip`**) или пакета уровня операционной системы, который хранится в стандартном хранилище вашего поставщика. Пакеты операционной системы с большей вероятностью будут устанавливаться и запускаться без проблем, но они с меньшей вероятностью будут обновляться. Ни один из вариантов не имеет явного преимущества.

Создание воспроизводимых сред

Программы, библиотеки и языки развивают сложные сети зависимостей, поскольку они эволюционируют вместе во времени. Производственный сервер может зависеть от десятков или сотен таких компонентов, каждый из которых имеет свои собственные ожидания относительно среды установки. Как определить, какая комбинация версий библиотеки создаст гармоничную среду? Как убедиться, что конфигурация, которую вы тестировали в лаборатории разработки, является той же самой, которая развертывается в облаке? В общем, как сделать так, чтобы управление всеми этими частями не было большой проблемой?

Языки Python и Ruby имеют стандартизированный способ для выражения зависимости между пакетами. В обеих системах разработчики пакетов создают текстовый файл в корне проекта, который перечисляет его зависимости. В языке Ruby файл называется **`Gemfile`**, а в языке Python — **`requirements.txt`**. Оба формата поддерживают гибкие спецификации версий для зависимостей, поэтому пакеты могут заявить, что они совместимы с "любой версией пакета **`simplejson`** версии 3 или выше" или "Rails 3, но не Rails 4". Также можно указать точное требование к версии для любой зависимости.

Оба формата файлов позволяют указать источник для каждого пакета, поэтому зависимости не обязательно должны распространяться через стандартный пакет хранилища. Поддерживаются все распространенные источники: от веб-адресов до локальных файлов и хранилищ GitHub.

Инсталлируйте пакет зависимостей Python с параметром **pip install -r requirements.txt**. Хотя команда **pip** отлично справляется с определением спецификаций отдельных версий, она, к сожалению, не может самостоятельно решать сложные отношения зависимостей между пакетами. Разработчикам иногда приходится настраивать порядок, в котором пакеты упоминаются в файле **requirements.txt** для достижения удовлетворительного результата. Кроме того, новые выпуски пакетов могут нарушить равновесие версии, хотя это происходит редко.

Команда **pip freeze** распечатывает текущий пакет пакетов Python в формате **requirements.txt**, указывая точную версию для каждого пакета. Эта функция может быть полезна для репликации текущей среды на производственном сервере.

В мире Ruby прямым аналогом команды **pip -r** является команда **gem install -g Gemfile**. Однако в большинстве случаев для управления зависимостями лучше использовать менеджер управления пакетами Bundler. Выполните команду **gem install bundler**, чтобы установить его (если он еще не установлен в системе), а затем запустите установку пакета из корневого каталога проекта, который вы настраиваете.[25]

У менеджера Bundler есть несколько интересных особенностей.

- Он действительно выполняет рекурсивное управление зависимостями, поэтому, если существуют пакеты, которые взаимно совместимы и удовлетворяют всем ограничениям, менеджер Bundler может найти его самостоятельно.
- Он автоматически записывает результаты расчетов версий в файл **Gemfile.lock**. Поддержание этой контекстной информации позволяет менеджеру Bundler обрабатывать обновления в файле **Gemfile** надежно и эффективно. При переносе на новую версию Gemfile менеджер Bundler изменяет только пакеты, в которых он нуждается.
- Поскольку файл **Gemfile.lock** ассоциирован со средой, запуск установки пакета на сервере развертывания автоматически воспроизводит среду пакета, найденную в среде разработки.[26]
- В режиме развертывания (**bundle install --deployment**) менеджер Bundler устанавливает отсутствующие пакеты в каталог локального проекта, помогая изолировать проект от любых будущих изменений. Затем можно использовать команду **bundle exec** для запуска определенных команд в этой гибридной среде пакетов.[27]

Несколько сред

Команда **pip** и **bundle** успешно осуществляют управление зависимостями для отдельных программ Python и Ruby, но что, если две программы на одном сервере имеют противоречивые требования?

В идеале каждая программа в производственной среде будет иметь собственную библиотечную среду, которая не зависит от системы и всех других программ.

---

[25] Пакеты в языке Ruby могут содержать команды на уровне оболочки. Однако у них обычно нет справочных страниц; для получения подробных сведений о пакете выполните команду **bundle help** или обратитесь к полной документации на сайте **bundler.io**..

[26] Или, по крайней мере, это поведение по умолчанию. При необходимости в файле **Gemfile** легко указывать различные требования к средам разработки и развертывания.

[27] Некоторые программные пакеты, такие как Rails, являются совместимыми с менеджером Bundler и могут использовать локально установленные пакеты даже без выполнения команды **exec bundle**.

Пакет `virtualenv`: виртуальные среды для языка Python

Пакет **`virtualenv`** языка Python создает виртуальные среды, которые находятся в своих собственных каталогах.[28] Чтобы настроить новую среду, после установки пакета просто запустите команду **`virtualenv`** с именем пути.

```
$ virtualenv myproject
New python executable in /home/ulsah/myproject/bin/python
Installing setuptools, pip, wheel...done.
```

Каждая виртуальная среда имеет каталог **`bin/`**, содержащий двоичные файлы для виртуальных сред Python и PIP. Когда вы запускаете один из этих двоичных файлов, вы автоматически помещаетесь в соответствующую виртуальную среду. Установите пакеты в среду, как обычно, запустив копию команды **`pip`** в виртуальной среде.

Для того чтобы запустить виртуализованную программу Python из демона **`cron`** или из сценария запуска системы, явно укажите путь к правильной копии **`python`**. (В качестве альтернативы поместите путь в строку сценария.)

При интерактивной работе в оболочке можно запустить сценарий **`bin/activate`** виртуальной среды, чтобы установить версии утилит **`python`** и **`pip`** в виртуальной среде по умолчанию. Сценарий перестраивает переменную `PATH` вашей оболочки. Для того чтобы покинуть виртуальную среду, используйте команду **`deactivate`** .

Виртуальные среды привязаны к определенным версиям Python. Во время создания виртуальной среды вы можете установить связанный двоичный код Python с помощью параметра **`--python virtualenv`**. В результате будет установлен бинарный файл Python.

RVM: менеджер `enVironment` для языка Ruby

В мире Ruby все похоже, но несколько сложнее. Выше было указано, что менеджер Bundler может кешировать локальные копии пакетов Ruby от имени конкретного приложения. Это разумный подход при перемещении проектов в производство, но это не так удобно для интерактивного использования. Он также предполагает, что вы хотите использовать установленную версию Ruby.

Те, кто хочет получить более общее решение, должны подумать о менеджере RVM, сложном и довольно неудобном виртуализаторе среды, который использует несколько особенностей оболочки. Справедливости ради отметим, что менеджер RVM является чрезвычайно отполированным примером "кривых костылей". На практике он работает плавно.

Менеджер RVM управляет как версиями Ruby, так и несколькими коллекциями пакетов, и позволяет переключаться между ними на ходу. Например, команда

```
$ rvm ruby-2.3.0@ulsah
```

активирует Ruby версии 2.3.0 и набор пакетов под названием `ulsah`. Ссылки на утилиты **`ruby`** или **`gem`** теперь разрешаются в рамках указанных версий. Этот пример также работает для программ, установленных пакетами, такими как **`bundle`** и **`rails`**. К счастью, управление пакетами не изменилось; просто используйте пакет или комплект, как обычно, и все вновь установленные пакеты автоматически попадут в нужное место.

Процедура установки менеджера RVM включает выбор сценария Bash из Интернета и его локальное выполнение. В настоящее время используются команды

```
$ curl -o /tmp/install -sSL https://get.rvm.io
$ sudo bash/tmp/install stable
```

[28] Как и в случае с другими командами, связанными с языком Python, существуют версии команды **`virtualenv`** с числовыми суффиксами, которые поступают с определенными версиями Python.

но все же проверьте текущую версию и криптографическую подпись на сайте `rvm.io`.[29] Обязательно проведите инсталляцию с помощью программы **sudo**, как показано выше; если вы этого не сделаете, менеджер RVM настроит частную среду в вашем домашнем каталоге. (Это не страшно, но ничто в производственной системе не должно ссылаться на ваш домашний каталог.) Кроме того, вам нужно будет добавить авторизованных пользователей RVM в группу UNIX `rvm`.

После первоначальной установки RVM не используйте **sudo** при установке пакетов или изменении конфигураций RVM. Менеджер RVM контролирует доступ через членство в группе `rvm`.

Менеджер RVM выполняет свою работу автоматически, манипулируя переменными среды оболочки и путем поиска. Следовательно, он должен быть включен в вашу среду во время запуска оболочки при входе в систему. Когда вы устанавливаете RVM на системном уровне, он включает сценарий **rvm.sh** с соответствующими командами в файл **/etc/profile.d**. Некоторые оболочки автоматически запускают эту заглушку. Если это не так, необходимо явно выполнить команду `source`, которую вы можете добавить в файлы запуска вашей оболочки:

```
source /etc/profile.d/rvm.sh
```

Менеджер RVM никак не изменяет исходную установку Ruby, в частности сценарии, начинающиеся с

```
#!/usr/bin/env ruby
```

Символы `#!` в стандартном Ruby видят только системные пакеты. Следующий вариант более гибкий:

```
#!/usr/bin/env ruby
```

Он находит команду **ruby** в соответствии с контекстом RVM пользователя, который его запускает.

Команда **rvm install** устанавливает новые версии языка Ruby. Эта функция RVM позволяет легко устанавливать несколько разных версий Ruby. Ее следует предпочесть собственным пакетам Ruby вашей операционной системы, которые редко обновляются. Команда **rvm install** загружает двоичные файлы, если они доступны. Если нет, она устанавливает необходимые пакеты операционной системы, а затем строит Ruby из исходного кода.

Вот как мы можем настроить развертывание приложения Rails, которое, как известно, совместимо с Ruby 2.2.1.

```
$ rvm install ruby-2.2.1
Searching for binary rubies, this might take some time.
No binary rubies available for: ubuntu/15.10/x86_64/ruby-2.2.1.
Continuing with compilation. Please read 'rvm help mount' to get more
   information on binary rubies.
Checking requirements for ubuntu.
Installing required packages: gawk, libreadline6-dev, zlib1g-dev,
   libncurses5-dev, automake, libtool, bison, libffi-dev.....
Requirements installation successful.
Installing Ruby from source to: /usr/local/rvm/rubies/ruby-2.2.1, this
   may take a while depending on your cpu(s)...
...
```

[29] См. комментарии о том, почему наши команды не соответствуют рекомендациям RVM, в разделе 1.10.

Если вы установили RVM, как описано выше, система Ruby устанавливается в каталоге **/usr/local/rvm** и доступна для всех учетных записей в системе.

Для поиска версии RVM следует использовать команду **rvm first known**, которая, как известно, знает, как выполнять загрузку и сборку. Команда **rvm list** выводит список уже установленных и доступных для использования пакетов RVM.

```
$ cd myproject.rails
$ rvm ruby-2.2.1@myproject --create --default --ruby-version
ruby-2.2.1 - #gemset created /usr/local/rvm/gems/ruby-2.2.1@myproject
ruby-2.2.1 - #generating myproject wrappers.........
$ gem install bundler
Fetching: bundler-1.11.2.gem (100%)
Successfully installed bundler-1.11.2
1 gem installed
$ bundle
Fetching gem metadata from https://rubygems.org/..........
Fetching version metadata from https://rubygems.org/...
Fetching dependency metadata from https://rubygems.org/..
Resolving dependencies......
...
```

Строка **ruby-2.2.1@myproject** задает версии Ruby и комплекта пакетов. Флаг **--create** создает комплект пакетов, если он еще не существует. Флаг **--default** устанавливает эту комбинацию по умолчанию, а флаг **-ruby-version** записывает имена интерпретатора Ruby и комплекта пакетов в файлы **.ruby-version** и **.ruby-gemset** в текущем каталоге.

Если файл **.*-version** существует, то менеджер RVM автоматически считывает и оценивает их при работе со сценариями в этом каталоге. Эта функция позволяет каждому проекту указывать свои собственные требования и освобождает вас от необходимости помнить, что с ним связано.

Чтобы запустить пакет в запрошенной среде (как описано в файлах **.ruby-version** и **.ruby-gemset**), выполните команду

rvm in */путь/к/каталогу* do *команда-запуска аргументы_запуска ...*

Это удобный синтаксис для использования при запуске заданий из сценариев запуска или демона **cron**. Он не зависит от текущего пользователя, настраивающего RVM, или от конфигурации RVM текущего пользователя.

В качестве альтернативы можно указать явную среду для команды, например:

rvm ruby-2.2.1@myproject **do** *команда-запуска аргументы_запуска ...*

Однако существует и третий вариант — запустить двоичный код **ruby** изнутри оболочки, поддерживаемой RVM для этой цели. Например, команда

/usr/local/rvm/wrappers/ruby-2.2.1@myproject/ruby ...

автоматически переносит вас в мир Ruby 2.2.1.

7.8. Контроль версий с помощью системы Git

Ошибки неизбежны. Важно следить за изменениями конфигурации и кода, чтобы, когда эти изменения вызовут проблемы, можно было легко вернуться к известному благополучному состоянию. Системы контроля версий — это программные средства, которые отслеживают, архивируют и предоставляют доступ к нескольким версиям файлов.

Системы контроля версий решают ряд проблем. Во-первых, они определяют организованный способ отслеживания истории изменений в файле, чтобы эти изменения можно было понять в контексте и чтобы можно было восстановить более ранние версии. Во-вторых, они расширяют концепцию версий выше уровня отдельных файлов. Связанные группы файлов могут быть сопоставлены вместе с учетом их взаимозависимостей. Наконец, системы контроля версий координируют действия нескольких редакторов, поэтому условия гонки не могут привести к тому, что чьи-либо изменения будут окончательно потеряны[30], и поэтому несовместимые изменения от нескольких редакторов не станут активными одновременно.

Самой популярной в настоящее время системой является Git, единолично созданная Линусом Торвальдсом (Linus Torvalds). Линус создал систему Git для управления исходным кодом ядра Linux из-за его разочарования в системах контроля версий, которые существовали в то время. В настоящее время он является таким же вездесущим и влиятельным, как Linux. Трудно сказать, какие из этих изобретений Линуса оказали большее влияние на мир.

Большинство современных программ разработано с помощью системы Git, и, как результат, администраторы сталкиваются с ним ежедневно. Вы можете найти, загрузить и внести вклад в проекты с открытым исходным кодом на GitHub, GitLab и других сайтах коллективной разработки. Вы также можете использовать систему Git для отслеживания изменений в сценариях, коде управления конфигурацией, шаблонах и любых других текстовых файлах, которые необходимо отслеживать с течением времени. Мы используем Git для отслеживания содержания этой книги. Он хорошо подходит для совместной работы и совместного использования, что делает его важным инструментом для сайтов, которые охватывают DevOps.

🕮 Для получения дополнительной информации о DevOps см. раздел 31.1.

Прелесть системы Git состоит в том, что у нее нет выделенного центрального хранилища. Чтобы получить доступ к хранилищу, клонируйте его (включая всю его историю) и носите с собой, как улитка свою раковину. Ваши фиксации в хранилище — это локальные операции, поэтому они выполняются быстро, и вам не нужно беспокоиться о связи с центральным сервером.

Система Git использует интеллектуальную систему сжатия, чтобы снизить стоимость хранения всей истории, и в большинстве случаев эта система достаточно эффективна.

Система Git отлично подходит для разработчиков, потому что они могут сворачивать исходный код на ноутбук и работать без подключения к сети, сохраняя при этом все преимущества контроля версий. Когда придет время интегрировать работу нескольких разработчиков, их изменения могут быть интегрированы из одной копии хранилища в другую любым способом, который подходит для рабочего процесса организации. Всегда можно развернуть две копии хранилища обратно в их общее состояние предка, независимо от того, сколько изменений и итераций произошло после раскола.

Использование локального хранилища Git — большой шаг вперед в управлении версиями — или, возможно, более точно, это большой шаг назад, но в хорошем смысле. Ранние системы контроля версий, такие как RCS и CVS, использовали локальные хранилища, но не могли обеспечивать совместную работу, слияние изменений и не-

[30]Например, предположим, что системные администраторы Алиса и Боб оба редактируют один и тот же файл и каждый из них вносит некоторые изменения. Алиса сохраняет файл первой. Когда Боб сохраняет свою копию файла, он перезаписывает версию Алисы. После того как Алиса выйдет из редактора, ее изменения полностью и бесследно исчезнут.

зависимую разработку проектов. В настоящее время установка файлов под контролем версий — это быстрая, простая и локальная операция. В то же время все расширенные возможности совместной работы Git доступны для использования в соответствующих ситуациях.

Система Git обладает сотнями функций и может быть довольно сложной в использовании. Тем не менее большинство пользователей Git обойдутся лишь несколькими простыми командами. Особые ситуации лучше всего обрабатывать путем поиска в Google описания того, что вы хотите сделать (например, "git undo last commit"). В первую очередь Google, конечно же, предложит посетить сайт Stack Overflow и найти дискуссию, которая точно соответствует вашей ситуации. Прежде всего, не паникуйте. Даже если вам кажется, что вы испортили хранилище и удалили результаты работы за последние несколько часов, в системе Git, скорее всего, есть скрытая копия. Вам просто нужно использовать флан `reflog` и найти ее.

Прежде чем вы начнете использовать систему Git, укажите свое имя и адрес электронной почты:

```
$ git config --global user.name "John Q. Ulsah"
$ git config --global user.email "ulsah@admin.com"
```

Эти команды создают ini-файл конфигурации Git **`~/.gitconfig`**, если он еще не существует. Более поздние команды **`git`** читают этот файл для настройки конфигурации. Система Git предоставляет пользователям широкие возможности для настройки рабочего процесса.

Простой пример Git

Мы разработали простое хранилище примеров для поддержки некоторых сценариев оболочки. На практике вы можете использовать Git для отслеживания кода управления конфигурацией, шаблонов инфраструктуры, специальных сценариев, текстовых документов, статических веб-сайтов и всего, что вам нужно для работы в течение долгого времени.

Следующие команды создают новое хранилище Git и заполняют его:

```
$ pwd
/home/bwhaley
$ mkdir scripts && cd scripts
$ git init
Initialized empty Git repository in /home/bwhaley/scripts/.git/
$ cat > super-script.sh << EOF
> #!/bin/sh
> echo "Hello, world"
> EOF
$ chmod +x super-script.sh
$ git add .
$ git commit -m "Initial commit"
[master (root-commit) 9a4d90c] super-script.sh
  1 file changed, 0 insertions(+), 0 deletions(-)
  create mode 100755 super-script.sh
```

В приведенной выше последовательности команда **`git init`** создает инфраструктуру хранилища, создавая каталог **`.git`** в каталоге **`/home/bwhaley/scripts`**. После того как вы создадите исходный сценарий "hello, world", добавьте команду **`git add`**. Она копирует его в "индекс" Git, который является промежуточной областью для предстоящей фиксации.

Индекс — это не просто список файлов для фиксации; это дерево файлов, каждое из которых является таким же реальным, как текущий рабочий каталог и содержимое храни-

лища. Файлы в индексе имеют содержимое, и в зависимости от того, какие команды вы выполняете, это содержимое может отличаться как от хранилища, так и от рабочего каталога. Команда **git add** на самом деле просто означает "**cp** из рабочего каталога в индекс".

Команда **git commit** вводит содержимое индекса в хранилище. Для каждой фиксации требуется сообщение журнала. Флаг **-m** позволяет включить сообщение в командной строке. Если вы оставите это, команда **git** запустит для вас редактор.

Теперь внесите изменения и проверьте их в хранилище.

```
$ vi super-script.sh
$ git commit super-script.sh -m "Made the script more super"
[master 67514f1] Made the script more super
  1 file changed, 1 insertions(+), 0 deletions(-)
```

Именование модифицированных файлов в командной строке **git commit** обходит обычное использование Git индекса и создает вариант, который включает только изменения в указанные файлы. Существующий индекс остается неизменным, и система Git игнорирует любые другие файлы, которые могут быть изменены.

Если изменение связано с несколькими файлами, существуют несколько вариантов. Если вы точно знаете, какие файлы были изменены, вы всегда можете их перечислить в командной строке, как показано выше. Если вы ленивы, то можете выполнить команду **git commit -a**, чтобы система Git добавляла все измененные файлы в индекс перед выполнением фиксации. Однако у этого последнего варианта есть пара подводных камней.

Во-первых, могут быть изменены файлы, которые вы не хотите включать в фиксацию. Например, если у сценария **super-script.sh** был файл конфигурации, и вы изменили этот файл конфигурации для отладки, вы можете не захотеть перенести измененный файл обратно в хранилище.

Вторая проблема заключается в том, что команда **git commit -a** выбирает только изменения в файлах, которые в настоящее время находятся под управлением системы контроля версий. Он не отображает новые файлы, которые вы, возможно, создали в рабочем каталоге.

Для обзора состояния системы Git вы можете выполнить команду **get status**. Эта команда сообщает вам о новых, измененных и индексированных файлах. Например, предположим, что вы добавили сценарий **more-scripts/another-script.sh**. Система Git может показать следующее.

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)
      modified: super-script.sh
Untracked files:
   (use "git add <file>..." to include in what will be committed)
      more-scripts/
      tmpfile
no changes added to commit (use "git add" and/or "git commit -a")
```

Сценарий **another-script.sh** не указан по имени, потому что система Git еще не видит каталог `more-scripts`, который его содержит. Вы можете видеть, что сценарий **super-script.sh** был изменен, а также увидеть файл **tmpfile**, который, вероятно, не должен быть включен в хранилище. Вы можете выполнить команду **git diff super-script.sh**, чтобы увидеть изменения, внесенные в сценарий. Команда **git** предлагает команды для следующих операций, которые вы можете выполнить.

Предположим, вы хотите отследить изменения в файле **super-script.sh** отдельно от вашего нового файла **another-script.sh**.

```
$ git commit super-script.sh -m "The most super change yet"
Created commit 6f7853c: The most super change yet
 1 files changed, 1 insertions(+), 0 deletions(-)
```

Для того чтобы искоренить файл **tmpfile** из системы Git, создайте или отредактируйте файл **.gitignore** и поместите в него имя файла. Это заставляет систему Git игнорировать файл **tmpfile** раз и навсегда. Шаблоны в тоже работают.

```
$ echo tmpfile >> .gitignore
```

Наконец, зафиксируйте все сделанные изменения.

```
$ git add .
$ sudo git commit -m "Ignore tmpfile; Add another-script.sh to the repo"
Created commit 32978e6: Ignore tmpfile; add another-script.sh to the repo
 2 files changed, 2 insertions(+), 0 deletions(-)
 create mode 100644 .gitignore
 create mode 100755 more-scripts/another-script.sh
```

Обратите внимание на то, что сам файл **.gitignore** становится частью управляемого набора файлов, который обычно вы хотите. Зачастую приходится повторно добавлять файлы, которые уже находятся под контролем, поэтому команда **git add** — это простой способ сказать: "Я хочу, чтобы образ нового хранилища выглядел как рабочий каталог, за исключением того, что указано в **.gitignore**". В этой ситуации нельзя просто выполнить команду **git commit -a**, потому что она не найдет файлы **another-script.sh** и **.gitignore**; эти файлы являются новыми для системы Git и поэтому должны быть явно добавлены.

Ловушки Git

Для того чтобы заставить вас думать, что система Git управляет файлами разрешений, а также их содержимым, она показывает вам режимы файлов при добавлении новых файлов в хранилище. Это неправда; система Git не отслеживает режимы, владельцев или время модификации.

Система Git *отслеживает* бит исполнения. Если вы выполняете сценарий с установленным битом исполнения, любые будущие клоны также исполняются. Однако не следует ожидать, что система Git будет отслеживать право собственности или статус "только для чтения". Следствием является то, что вы не можете рассчитывать на использование системы Git для восстановления сложных иерархий файлов в ситуациях, когда важны права собственности и разрешения.

Другим следствием является то, что в хранилище Git никогда нельзя включать простые текстовые пароли и другие секреты. Они не только открыты для всех, кто имеет доступ к хранилищу, но также могут быть непреднамеренно распакованы в форме, доступной миру.

Коллективное кодирование с помощью системы Git

Появление и быстрый рост сайтов коллективной разработки, таких как GitHub и GitLab, является одним из самых важных тенденций в новейшей компьютерной истории. Миллионы проектов программного обеспечения с открытым исходным кодом создаются и прозрачно управляются огромными сообществами разработчиков, которые используют все мыслимые языки. Программное обеспечение никогда не было проще создавать и распространять.

GitHub и GitLab — это, по сути, хранилища Git с множеством дополнительных функций, связанных с коммуникацией и рабочим процессом. Хранилище может создать любой человек. Хранилища доступны как благодаря команде `git`, так и в Интернете. Веб-интерфейс дружелюбен и предлагает функции поддержки сотрудничества и интеграции.

Опыт коллективного кодирования может быть несколько пугающим для новичков, но на самом деле это не сложно, как только будут поняты некоторые основные термины и методология.

- "Master" — это имя по умолчанию, назначенное первой ветви в новом хранилище. Большинство программных проектов по умолчанию используют это имя в качестве основной линии разработки, хотя у некоторых главная ветвь может отсутствовать. Главной ветвью обычно управляют, чтобы поддерживать текущий, но работоспособный код; разработка нового кода происходит в другом месте. Последняя фиксация называется *вершиной главной ветви*.
- Ветвление (fork) в системе GitHub представляет собой моментальный снимок хранилища в определенный момент времени. Ветвления возникают, когда у пользователя нет разрешения на изменение основного хранилища, но он хочет внести изменения, либо для дальнейшей интеграции с основным проектом, либо для создания совершенно отдельного пути разработки.
- Запрос на включение (pull request) — это запрос на объединение изменений из одной ветви или ветвления в другую. Они читаются сторонними разработчиками целевого проекта и могут быть приняты для включения кода других пользователей и разработчиков. Каждый запрос на включение также является нитью дискуссии, поэтому комментировать предполагаемые обновления кода могут как владелец, так и любой другой человек.
- Разработчик (committer) и специалист по эксплуатации (maintainer) — это лицо, у которого есть доступ к хранилищу для записи. Для крупных проектов с открытым исходным кодом этот очень желанный статус предоставляется только доверенным разработчикам, которые имеют долгую историю вкладов.

Вам часто придется обращаться в хранилище GitHub или GitLab, чтобы найти или обновить часть программного обеспечения. Убедитесь, что вы просматриваете главное хранилище, а не случайное ветвление. Обратите внимание на ближайшую метку "ответвление от" и следуйте за ней.

Будьте осторожны при оценке нового программного обеспечения с этих сайтов. Ниже приведены несколько вопросов, которые стоит обдумать, прежде чем запускать случайную часть нового программного обеспечения на своих компьютерах.

- Сколько участников принимали участие в разработке?
- Означает ли история фиксации, что разработка является недавней и регулярной?
- Какова лицензия и совместима ли она с потребностями вашей организации?
- На каком языке написано программное обеспечение, и знаете ли вы, как им управлять?
- Достаточно ли документации для эффективного использования программного обеспечения?

Большинство проектов имеют определенную стратегию ветвления, на которую они полагаются, чтобы отслеживать изменения программного обеспечения. Некоторые сторонники настаивают на строгом соблюдении выбранной ими стратегии, а другие — более снисходительны. Одной из наиболее широко используемых является модель Git Flow, раз-

работанная Винсентом Дриссеном (Vincent Driessen); дополнительную информацию см. на сайте `goo.gl/GDaF`. Прежде чем вносить свой вклад в проект, ознакомьтесь с методами его разработки, чтобы помочь специалистам, которые его сопровождают.

Прежде всего, помните, что разработчики с открытым исходным кодом часто не получают никакой платы. Участвуя в коллективной разработке и поддерживая проекты, они ценят терпение и вежливость.

7.9. Литература

- Brooks, Frederick P. JR. *The Mythical Man-Month: Essays on Software Engineering.* Reading, MA: Addison-Wesley, 1995.
- Chacon, Scott, and Straub, Ben. *Pro Git, 2nd edition. 2014.* `git-scm.com/book/en/v2`. Исчерпывающая книга о Pro Git, свободно опубликованная по лицензии Creative Commons.

Оболочки и сценарии оболочки

- Robbins, Arnold, and Nelson H. F. Beebe. *Classic Shell Scripting.* Sebastopol, CA: O'Reilly Media, 2005. Книга, посвященная традиционному (и переносимому) диалекту оболочки Bourne. Она также содержит довольно много информации об утилитах **sed** и **awk**.
- Powers, Shelley, Jerry Peek, Tim O'Reilly, and Mike Loukides. *Unix Power Tools, (3rd Edition)*, Sebastopol, CA: O'Reilly Media, 2002. Классическая книга о системе UNIX, охватывающая много вопросов, в том числе сценарии для оболочки **sh** и работу с командной строкой. Некоторые разделы несколько устарели, но материал, касающийся оболочек, остается актуальным.
- Sobell, Mark G. A *Practical Guide to Linux Commands, Editors, and Shell Programming.* Upper Saddle River, NJ: Prentice Hall, 2012. Книга заслуживает внимания, поскольку в ней описываются оболочки **tcsh** и **bash**.
- Shotts, William E., Jr. *The Linux Command Line: A Complete Introduction.* San Francisco, CA: No Starch Press, 2012. Книга посвящена оболочке **bash**, но в ней также есть материал об интерактивной работе и программировании. Большая часть материала относится к системам UNIX и Linux.
- Blum, Richard, and Christine Bresnahan. *Linux Command Line and Shell Scripting Bible (3rd Edition).* Indianapolis, IN: John Wiley & Sons, Inc. 2015. Книга посвящена, в основном, оболочкам, в частности оболочке **bash**.
- Cooper, Mendel. *Advanced Bash-Scripting Guide.* `www.tldp.org/LDP/abs/html`. Свободно распространяемая и легко доступная в Интернете книга. Несмотря на ее название, новички ее легко поймут благодаря множеству хороших примеров.

Регулярные выражения

- Friedl, Jeffrey. *Mastering Regular Expressions (3rd Edition)*, Sebastopol, CA: O'Reilly Media, 2006.
- Goyvaerts, Jan and Steven Levithan. *Regular Expressions Cookbook.* Sebastopol, CA: O'Reilly Media, 2012.

- Goyvaerts, Jan. `regular-expressions.info`. Подробный интерактивный источник информации о регулярных выражениях с учетом всевозможных диалектов.
- Krumins, Peteris. *Perl One-Liners: 130 Programs That Get Things Done*. San Francisco, CA: No Starch Press, 2013.

Python

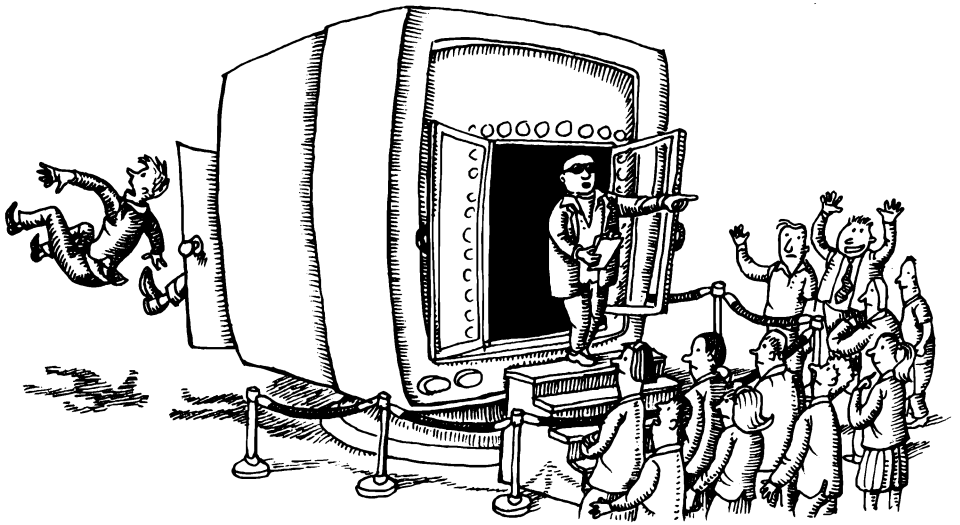
- Sweigart, Al. *Automate the Boring Stuff with Python: Practical Programming for Total Beginners*. San Francisco, CA: No Starch Press, 2015. Удачное введение в программирование на языке Python 3 и программирование в целом. Содержит множество примеров системного администрирования.(Эл Свейгарт. *Автоматизация рутинных задач с помощью Python: практическое руководство для начинающих*, пер. с англ., изд. "Диалектика", 2016.)
- Pilgrim, Mark. *Dive Into Python*. Berkeley, CA: Apress, 2004. Классическая книга о языке Python 2, свободно доступная на веб-сайте `diveintopython.net`.
- Pilgrim, Mark. *Dive Into Python 3*. Berkeley, CA: Apress, 2009. Dive Into Python updated for Python 3. Книга свободно доступна на веб-сайте `diveintopython3.net`.
- Lutz, Mark. *Learning Python, 5th Edition*. O'Reilly, 2013. Фундаментальная книга о языке Python.(Марк Лутц. *Изучаем Python, 5-е издание*, в двух томах, пер. с англ., изд. "Диалектика", 2019.)
- Ramalho, Luciano. *Fluent Python*. Sebastopol, CA: O'Reilly Media, 2015. Advanced, idiomatic Python 3.
- Beazley, David, and Brian K. Jones. *Python Cookbook (3rd Edition)*, Sebastopol, CA: O'Reilly Media, 2013. Covers Python 3.
- Gift, Noah, and Jeremy M. Jones. *Python for Unix and Linux System Administrators*, Sebastopol, CA: O'Reilly Media, 2008.

Ruby

- Flanagan, David, and Yukihiro Matsumoto. *The Ruby Programming Language*. Sebastopol, CA: O'Reilly Media, 2008. Классическая, исчерпывающая и хорошо написанная книга о языке Ruby из первых рук. Она уже немного устарела и не учитывает Ruby 2.0 и более новые версии; однако языковые различия между ними являются минимальными.
- Black, David A. *The Well-Grounded Rubyist (2nd Edition)*. Shelter Island, NY: Manning Publications, 2014. Не бойтесь названия, которое может отпугнуть новичков; это хорошая, основательная книга о языке Ruby 2.1.
- Thomas, Dave. *Programming Ruby 1.9 & 2.0: The Pragmatic Programmer's Guide (4th Edition)*. Pragmatic Bookshelf, 2013. Classic and frequently updated.
- Fulton, Hal. *The Ruby Way: Solutions and Techniques in Ruby Programming (3rd Edition)*. Upper Saddle River, NJ: Addison-Wesley, 2015. Еще один классический и современный справочник по языку Ruby, с легким философским уклоном.

Глава 8

Управление учетными записями пользователей



Современные вычислительные среды состоят из физического оборудования, облачных систем и виртуальных хостов. Гибкость этой гибридной инфраструктуры порождает растущую потребность в централизованном и структурированном управлении учетными записями. Системные администраторы должны понимать как традиционную модель учетных записей, используемую UNIX и Linux, так и способы расширения этой модели для интеграции с такими службами каталогов, как LDAP и Microsoft Active Directory.

Правильное управление учетными записями является ключевым фактором безопасности системы. Редко используемые учетные записи, а также учетные записи с легко угадываемыми паролями являются основными целями для злоумышленников. Даже если вы используете автоматизированные инструменты вашей системы для добавления и удаления пользователей, важно понимать изменения, которые осуществляют эти инструменты. По этой причине мы начинаем обсуждение управления учетными записями с простых файлов, которые необходимо изменить, чтобы добавить пользователей автономного компьютера. В последующих разделах мы рассмотрим команды управления более высокого уровня, которые поставляются с нашими демонстрационными примерами операционных систем, и файлы конфигурации, которые контролируют их поведение.

В большинстве систем также есть простые инструменты с графическим пользовательским интерфейсом для добавления и удаления пользователей, но они, как правило, не поддерживают дополнительные функции, такие как пакетный режим или расширенная локализация. Эти инструменты достаточно простые, поэтому мы не считаем целесообразным детально анализировать их работу и в этой главе будем использовать командную строку.

В этой главе мы сосредоточимся исключительно на добавлении и удалении пользователей. Многие темы, связанные с управлением учетными записями, описаны в других главах (здесь вы найдете соответствующие ссылки).

- Подключаемые модули аутентификации (pluggable authentication modules — PAM) для шифрования паролей и обеспечения их стойкости, описаны в главе 17 (см. раздел 17.3).
- Хранилища паролей описаны в главе 27 (см. раздел 27.4).
- Службы каталогов, такие как OpenLDAP и Active Directory, рассматриваются в главе 17 (см. раздел 17.2).
- Наконец, политические и правовые вопросы являются основными темами главы 31.

8.1. Основы управления учетными записями

По существу, пользователь представляет собой всего лишь число, 32-битное целое число без знака, известное как идентификатор пользователя, ID или UID. Почти все, что связано с управлением учетными записями пользователей, вращается вокруг этого числа.

Система определяет интерфейс прикладного программирования (через стандартные подпрограммы библиотеки C), который осуществляет прямое и обратное отображение между идентификаторами UID и более полными наборами сведений о пользователях. Например, функция **getpwuid()** принимает UID в качестве аргумента и возвращает соответствующую запись, содержащую имена пользователя и его домашнего каталога. Аналогично функция **getpwnam()** просматривает эту же информацию по имени учетной записи.

Традиционно эти библиотечные функции получали аргументы непосредственно из текстового файла **/etc/passwd**. Со временем они начали поддерживать дополнительные источники информации, такие как сетевые информационные базы данных (например, LDAP) и файлы с защитой от чтения, в которых зашифрованные пароли можно было бы хранить более надежно.

Эти уровни абстракции (которые часто устанавливаются в файле **nsswitch.conf**) позволяют функциям более высокого уровня функционировать без достоверного знания используемого метода управления базой данных. Например, когда вы входите в систему как `dotty`, процесс регистрации (Window Server, **login**, **getty** или что-то еще) применяет к этому имени функцию **getpwnam()**, а затем сравнивает пароль, который вы вводите, с его зашифрованной записью, возвращаемой библиотекой, независимо от его фактического источника.

Мы начинаем с подкаталога **/etc/passwd**, который по-прежнему поддерживается везде. Другие варианты воспроизводят эту модель, если не по форме, то по духу.

🕮 Дополнительную информацию о файле **nsswitch.conf** см. в разделе 17.3.

8.2. Файл /etc/passwd

Файл **/etc/passwd** содержит список пользователей, которые известны системе. Его можно расширить или заменить службой каталогов, поэтому его можно считать полным и достоверным только в автономных системах.

Традиционно зашифрованный пароль каждого пользователя также хранился в файле **/etc/passwd**, который был доступен для чтения. Однако появление более мощных процессоров повысило вероятность взлома этих открытых паролей. В ответ системы

UNIX и Linux переместили пароли в отдельный файл (**/etc/master.passwd** в FreeBSD и **/etc/shadow** в Linux), который защищен от чтения. В наши дни сам файл **passwd** содержит только формальную запись, чтобы отметить прежнее местоположение поля для ввода пароля (x в Linux и * в FreeBSD).

В процессе регистрации пользователя система обращается к файлу **/etc/passwd** в поисках идентификатора пользователя и его домашнего каталога, а также другой информации. Каждая строка файла описывает одного пользователя и содержит семь следующих полей, разделенных двоеточиями:

- регистрационное имя;
- шифрованный пароль или "заполнитель" пароля (вопросы применения шифрованных паролей описаны далее в этом разделе);
- идентификатор пользователя UID;
- идентификатор группы по умолчанию GID;
- поле GECOS (полное имя, номер офиса, рабочий и домашний телефоны);
- домашний каталог;
- регистрационная оболочка.

Вот примеры правильно составленных записей файла **/etc/passwd**.

```
root:x:0:0:The System,,x6096, :/:/bin/sh
jl:!:100:0:Jim Lane,ECOT8-3,,:/staff/jl:/bin/sh
dotty:x:101:20::/home/dotty:/bin/tcsh
```

🕮 Дополнительную информацию о файле **nsswitch.conf** см. в разделе 17.3.

Если пользовательские учетные записи совместно используются с помощью службы каталогов, например LDAP, в файле **passwd** появляются специальные записи, которые начинаются с символа "+" или "-". Эти записи сообщают системе, как интегрировать данные службы каталогов в содержимое файла **/etc/passwd**. Эта интеграция также может быть реализована в файле **/etc/nsswitch.conf**.

В следующих разделах мы рассмотрим поля файла **/etc/passwd** более подробно.

Регистрационное имя

Регистрационные имена (называемые также именами пользователей) должны быть уникальными и в зависимости от системы могут иметь ограничения на длину и набор символов. В настоящее время во всех версиях систем UNIX и Linux эта длина не превышает 32 символа.

Пользовательские имена не могут содержать двоеточия и символы новой строки, поскольку эти символы применяются в качестве разделителей полей и записей соответственно в файле **passwd**. В зависимости от системы на пользовательские имена могут быть наложены и другие ограничения. В системе Ubuntu эти ограничения наиболее слабые, поскольку они допускают имена, начинающиеся или целиком состоящие из чисел и других специальных символов.[1] По многочисленным причинам, которые было бы слишком долго объяснять, мы рекомендуем ограничивать допустимый диапазон регистрационных имен алфавитно-цифровыми символами, использовать только нижний регистр и начинать имена с буквы.

Регистрационные имена чувствительны к регистру. Нам неизвестны случаи, когда смешение регистров в именах приводило бы к возникновению проблем, но имена,

[1] По какой-то непостижимой причине допустимый набор символов в Unicode включает эмотиконы. Это делает нас ☺.

состоящие из строчных букв, считаются традиционными, к тому же их проще вводить. Если, например, такие регистрационные имена, как `john` и `John`, будут принадлежать различным людям, проблемы обязательно возникнут.

Следует выбирать такие регистрационные имена, которые легко запомнить, поэтому имя, состоящее из случайной последовательности букв, является не самым удачным вариантом. Поскольку регистрационные имена часто используются в адресах электронной почты, полезно определить стандартную процедуру их формирования. Пользователям должна быть предоставлена возможность делать обоснованные предположения о регистрационных именах друг друга. Имена, фамилии, инициалы и сочетания этих элементов — вот приемлемые варианты для схем формирования имен.[2]

Любая жестко заданная схема в конечном счете приводит к появлению дубликатов или слишком длинных имен, поэтому иногда придется делать исключения. Выберите стандартный способ разрешения конфликтов, добавив, например, в конец имени цифру.

В крупных организациях в адресах электронной почты часто используются полные имена (например, `John.Q.Public@mysite.com`), благодаря чему регистрационные имена скрываются от внешнего мира. Это прекрасная идея, но, чтобы облегчить работу администраторам, необходимо установить четкое и понятное соответствие между регистрационными именами и реальными именами пользователей.

В заключение отметим: необходимо, чтобы имя пользователя на всех компьютерах было одинаковым. Это удобно как самому пользователю, так и администратору.

Зашифрованные пароли

Исторически системы шифровали пароли пользователей с помощью алгоритма DES. По мере увеличения вычислительной мощности эти пароли стали тривиальными для взлома. Затем системы перешли на скрытые пароли и криптографию на основе алгоритма MD5. Теперь, когда в алгоритме MD5 были обнаружены значительные недостатки, текущим стандартом стало хеширование паролей с помощью криптографической "соли" на основе алгоритма SHA-512 (см. документ *Guide to Cryptography* на веб-сайте `owasp.org`).

Наши иллюстративные системы поддерживают множество алгоритмов шифрования, но все они по умолчанию соответствуют алгоритму SHA-512. Если вы не обновляете системы из более старых версий, вам не нужно обновлять выбор алгоритма.



В системе FreeBSD алгоритм, заданный по умолчанию, можно модифицировать с помощью файла **`/etc/login.conf`**.



В системах Debian и Ubuntu алгоритм, заданный по умолчанию, можно было модифицировать с помощью файла **`/etc/login.defs`**, но после появления подключаемых модулей аутентификации PAM, этот подход стал устаревшим. Политики установления паролей по умолчанию, включая выбор алгоритма хеширования, можно найти в файле **`/etc/pam.d/common-passwd`**.

RHEL 

В системах алгоритм шифрования паролей можно по-прежнему установить в файле **`/etc/login.defs`** с помощью команды **`authconfig`**, как показано ниже:

```
$ sudo authconfig --passalgo=sha512 --update
```

[2]Стандарт RFC5321 требует, чтобы локальная часть адреса (т.е. часть перед знаком `@`) считалась чувствительной к регистру. Остальная часть адреса обрабатывается в соответствии со стандартами DNS, который нечувствителен к регистру. К сожалению, это различие является тонким, и оно не реализуется повсеместно. Помните также, что многие устаревшие системы электронной почты возникли до появления сообщества IETF.

Изменение алгоритма шифрования паролей не приводит к обновлению существующих паролей, поэтому пользователи должны вручную изменить свои пароли, прежде чем новый алгоритм вступит в действие. Для того чтобы аннулировать старый пароль и принудительно внести обновления, используйте команду

```
$ chage -d 0 имя_пользователя
```

Качество пароля — еще одна важная проблема. Теоретически более длинные пароли более безопасны, как и пароли, содержащие разнотипные символы (например, прописные буквы, знаки препинания и цифры).

Большинство систем позволяют устанавливать стандарты построения паролей для пользователей, но имейте в виду, что пользователи могут умело обойти эти требования, если найдут их чрезмерными или обременительными. Стандарты, используемые нашими иллюстративными системами, приведены в табл. 8.1.

Таблица 8.1. Стандарты качества паролей

Система	Требования по умолчанию	Место установки

Red Hat CentOS	Больше 8 символов с требованием уровня сложности	`/etc/login.defs`
		`/etc/security/pwquality.conf`
		`/etc/pam.d/system-auth`

Debian Ubuntu	Больше 6 символов с требованием уровня сложности	`/etc/login.defs` `/etc/pam.d/common-password`
FreeBSD	Без ограничений	`/etc/login.conf`

🕮 Дополнительную информацию о выборе паролей см. в разделе 27.3.

Требования к качеству пароля — это вопрос дебатов, но мы рекомендуем вам определить приоритет по сложности.[3] Двенадцать символов — это минимальная длина для будущего пароля; обратите внимание, что это значительно больше, чем длина, заданная по умолчанию в любой системе. В вашей организации могут существовать общие стандарты качества пароля. Если это так, отложите эти настройки.

Если вы решили обойти свои системные инструменты для добавления пользователей и вместо этого изменить файл **`/etc/passwd`** вручную (выполнив команду **`vipw`** — см. раздел 8.6), чтобы создать новую учетную запись, поместите в зашифрованное поле пароля символ * (FreeBSD) или x (Linux). Эта мера предотвратит несанкционированное использование учетной записи до тех пор, пока вы или пользователь не установите реальный пароль.

Шифрованные пароли имеют постоянную длину (86 символов для SHA-512, 34 символа для MD5 и 13 символов для DES) независимо от длины незашифрованного пароля. Пароли шифруются в сочетании со случайной "солью", так что одному паролю могут соответствовать разные зашифрованные формы. Если два пользователя выбирают один и тот же пароль, этот факт обычно не может быть обнаружен путем проверки зашифрованных паролей.

Пароли, зашифрованные алгоритмом MD5 в файле теневого пароля, всегда начинаются с символов `$1$` или `$md5$`. Пароли Blowfish начинаются с символов `$2$`, пароли SHA-256 — с `$5$`, а пароли SHA-512 — с `$6$`.

[3] Дополнительную информацию по этой проблеме см. по адресу `xkcd.com/comics/password_strength.png`.

Идентификатор пользователя

🕮 Дополнительную информацию об учетной записи пользователя root см. в разделе 3.1.

По определению идентификатор пользователя root равен нулю. Большинство систем также определяют псевдопользователей, например bin и daemon, как владельцев команд или файлов конфигурации. Эти фиктивные регистрационные имена принято указывать в начале файла **/etc/passwd**, присваивая им низкие идентификаторы UID и назначая для них фиктивную оболочку (например, **/bin/false**), чтобы никто не мог войти в систему под этими именами.

Для того чтобы зарезервировать достаточно места для будущих фиктивных пользователей, мы рекомендуем назначать UID для реальных пользователей начиная с 1000 или выше. (Желаемый диапазон для новых UID можно указать в файлах конфигурации с помощью параметров команды **useradd**.) По умолчанию наши системы ссылок Linux присваивают идентификаторы UID, начиная с 1000. Система FreeBSD присваивает первому реальному пользователю UID, равный 1001, а затем добавляет единицу для каждого нового пользователя.

Не используйте UID повторно, даже когда пользователи покидают вашу организацию и вы удаляете их учетные записи. Эта предосторожность предотвращает путаницу, которая может возникнуть при дальнейшем восстановлении файлов из резервных копий, в которых пользователи могут быть идентифицированы с помощью UID, а не по регистрационному имени.

Идентификаторы UID должны быть уникальными в масштабе всей организации. Иначе говоря, конкретный UID должен ссылаться на одно и то же регистрационное имя и одного и того же человека на всех машинах, которые разрешено использовать этому человеку. Неспособность поддерживать уникальные UID может привести к проблемам безопасности в таких системах, как NFS, а также к путанице, когда пользователи переходят из одной рабочей группы в другую.

Трудно поддерживать уникальные UID, если группы машин управляются разными людьми или организациями. Эти проблемы носят технический и политический характер. Лучшее решение — иметь центральный сервер базы данных или каталог, который содержит запись для каждого пользователя и обеспечивает уникальность.

Более простая схема — назначить каждой группе внутри организации собственный диапазон UID и позволить каждой группе управлять собственным диапазоном. Это решение ограничивает пространство UID, но не устраняет параллельную проблему с уникальными регистрационными именами. Независимо от вашей схемы, основной задачей является согласованность подхода. Если согласованность невозможна, уникальность UID становится второй по важности целью.

Популярной системой управления и распространения информации об учетных записях, которая зарекомендовала себя в крупных организациях, является протокол Lightweight Directory Access Protocol (LDAP). Он кратко описан в этой главе и более подробно рассматривается в разделе 17.2.

Идентификатор группы по умолчанию

Как и идентификатор пользователя (UID), идентификатор группы (GID) является 32-битовым целым числом. Идентификатор 0 зарезервирован для группы с именем root, system или wheel. Как и в случае идентификаторов пользователей, системы используют несколько предопределенных групп для собственных нужд. Увы, в этом вопросе согласованности среди коллег — производителей систем не наблюдается. Например,

группа `bin` имеет идентификатор GID, равный 1, в системах Red Hat, CentOS и GID, равный 2, — в системах Ubuntu, Debian и GID, равный 7, — в системе FreeBSD.

В те далекие времена, когда компьютеры были еще дорогим удовольствием, группы использовались для учета машинного времени, чтобы время работы центрального процессора, время, затраченное на регистрацию, и использованное дисковое пространство оплачивал соответствующий отдел. В настоящее время группы используются, в основном, для организации совместного доступа к файлам.

🕮 Дополнительную информацию о каталогах с установленным битом `setgid` см. в разделе 5.5.

Группы определяются в файле **`/etc/group`**, а поле идентификатора группы в файле **`/etc/passwd`** задает стандартный ("фактический") идентификатор GID на момент регистрации пользователя в системе. Этот идентификатор не играет особой роли при определении прав доступа; он используется лишь при создании новых файлов и каталогов. Новые файлы обычно включаются в фактическую группу своего владельца, но если вы хотите разделять файлы с другими участниками проектной группы, не забудьте вручную изменить для этих файлов владельца группы.

При этом следует иметь в виду, что если у каталогов установлен бит `setgid` (02000) или файловая система смонтирована с опцией **`grpid`**, новые файлы включаются в группу родительского каталога.

Поле GECOS

Это поле иногда используется для хранения персональной информации о каждом пользователе. Оно является наследием некоторых из ранних версий системы UNIX, использовавших для разных целей семейство операционных систем General Electric Comprehensive Operating Systems. Оно не имеет четко определенного синтаксиса. В принципе структура поля GECOS может быть произвольной, а ее элементы разделяются запятыми и размещаются в следующем порядке:

- полное имя (часто используется только это поле);
- номер офиса и здания;
- рабочий телефон;
- домашний телефон.

🕮 Дополнительную информацию о базе данных LDAP см в разделе 17.2.

Информацию, содержащуюся в поле GECOS, можно изменять с помощью команды **`chfn`**. Эта команда используется для ведения и обновления списка телефонных номеров, но ею часто злоупотребляют: например, пользователь может изменить информацию так, что она станет нецензурной или некорректной. В некоторых системах можно установить ограничения, указав, какие поля может модифицировать команда **`chfn`**. Администраторы сетей большинства университетских городков совсем отключают команду **`chfn`**. Во многих системах команда **`chfn`** "понимает" только файл **`passwd`**, поэтому, если для управления регистрационной информацией вы используете базу данных LDAP или какую-либо иную службу каталогов, команда **`chfn`** может не работать вовсе.

Домашний каталог

Войдя в систему, пользователь попадает в свой домашний каталог. Именно в домашних каталогах оболочки регистрации ищут информацию, связанную с учетной записью,

псевдонимы оболочки и переменные окружения, а также ключи SSH, сертификаты серверов и другие параметры.

Следует иметь в виду, что если домашние каталоги смонтированы вне файловой системы, то в случае проблем с сервером или с самой сетью они могут оказаться недоступными. Если на момент регистрации этот каталог отсутствует, выводится сообщение вроде `no home directory` (домашний каталог отсутствует) и пользовательские данные помещаются в каталог `/`.[4] В качестве альтернативы регистрацию можно отключить совсем с учетом системной конфигурации. Более подробно домашние каталоги описываются в разделе 8.6.

Регистрационная оболочка

В качестве регистрационной оболочки, как правило, задается интерпретатор команд, но в принципе это может быть любая программа. По умолчанию для системы FreeBSD используется интерпретатор **sh**, а для системы Linux — интерпретатор **bash** (GNU-версия оболочки **sh**).

В некоторых системах пользователи могут менять интерпретатор с помощью команды **chsh**, но, подобно **chfn**, эта команда может не работать, если для управления регистрационной информацией вы используете базу данных LDAP или какую-либо иную службу каталогов. Если вы используете файл **/etc/passwd**, системный администратор всегда может заменить интерпретатор пользователя, отредактировав файл **passwd** с помощью программы **vipw**.

8.3. Файлы /etc/shadow

Файл скрытых паролей доступен для чтения только суперпользователю и предназначен для хранения зашифрованных паролей подальше от любопытных глаз и программ взлома. В нем также содержится учетная информация, которая отсутствует в исходном файле **/etc/passwd**. В настоящее время механизм скрытых паролей используется по умолчанию практически во всех системах.

Файл **shadow** не включает в себя файл **passwd**, и последний не генерируется автоматически при изменении файла **shadow**. Оба файла необходимо хранить независимо друг от друга (или использовать команду наподобие **useradd** для автоматического управления файлами). Как и **/etc/passwd**, файл **/etc/shadow** содержит одну строку для каждого пользователя. Каждая строка состоит из 9 полей, разделенных двоеточиями:

- регистрационное имя;
- зашифрованный пароль;
- дата последнего изменения пароля;
- минимальное число дней между изменениями пароля;
- максимальное число дней между изменениями пароля;
- количество дней до истечения срока действия пароля, когда выдается предупреждение;

[4] Это сообщение появляется при регистрации в системе через консоль или терминал, но не через экранный диспетчер, такой как **xdm**, **gdm** или **kdm**. В последнем случае пользователь вообще будет немедленно выведен из системы, поскольку программа-диспетчер не сможет осуществить запись в нужный каталог (например, **~/.gnome**).

- количество дней по истечении срока действия пароля, когда учетная запись аннулируется;
- дата истечения срока действия учетной записи;
- зарезервированное поле, в настоящее время пустое.

Лишь первые два поля обязательно должны быть заполнены. Поля дат в файле **/etc/shadow** задаются в виде числа дней (а не секунд), прошедших с 1 января 1970 года, что не является стандартным способом вычисления времени в системах UNIX и Linux.[5]

Типичная запись выглядит так.

```
millert:$6$iTEFbMTM$CXmxPwErbEef9RUBvf1zv8EgXQdaZg2eOd5uXyvt4sFzi6G41
    IqavLilTQgniAHm3Czw/LoaGzoFzaMm.YwOl/:16971:0:180:14:::
```

Приведем более подробное описание некоторых полей.

- Регистрационное имя совпадает с именем из файла **/etc/passwd**. Оно связывает записи файлов **passwd** и **shadow**.
- Зашифрованный пароль идентичен тому, который ранее хранился в файле **/etc/passwd**.
- Третье поле содержит дату последнего изменения пользователем своего пароля. Это поле обычно заполняется командой **passwd**.
- В четвертом поле задано количество дней, спустя которые пользователь сможет снова изменить пароль. Это не позволяет пользователям немедленно возвращаться к привычным паролям после их обязательного изменения. Такая возможность кажется нам опасной, если в течение указанного времени будет обнаружено нарушение безопасности системы. Поэтому мы рекомендуем устанавливать данное поле равным нулю.
- В пятом поле задано максимальное число дней между двумя изменениями пароля. С помощью этой установки администраторы заставляют пользователей менять свои пароли (подробнее механизм устаревания паролей описан в разделе 27.4). В системе Linux максимальное время жизни пароля определяется суммой значений данного и седьмого (льготный период) полей.
- В шестом поле задано количество дней, оставшихся до момента устаревания пароля, когда программа **login** должна предупреждать пользователя о необходимости сменить пароль.
- В восьмом поле задан день, в который истекает срок действия учетной записи (считая от 1 января 1970 года). По прошествии этой даты пользователь не сможет зарегистрироваться в системе, пока администратор не сбросит значение поля. Если поле оставлено пустым, учетная запись всегда будет активной.[6]
- Чтобы установить поле даты истечения срока, можно использовать команду **usermod** (даты здесь должны быть в формате *гггг-мм-дд*).
- Девятое поле зарезервировано на будущее.[7]

Теперь, когда назначение полей понятно, вернемся к рассмотренному выше примеру.

```
millert:$6$iTEFbMTM$CXmxPwErbEef9RUBvf1zv8EgXQdaZg2eOd5uXyvt4sFzi6G41
    IqavLilTQgniAHm3Czw/LoaGzoFzaMm.YwOl/:17336:0:180:14:::
```

[5] При этом вы можете преобразовать секунды в дни с помощью команды **expr `date+%s` / 86400**.

[6] Седьмое поле в книге не описано.— *Примеч. ред.*

[7] Возможно, это поле никогда не будет использовано.

В этой записи говорится о том, что пользователь millert последний раз менял свой пароль 19 июня 2017 года. Следующий раз пароль должен быть изменен через 180 дней. За две недели до этого пользователь начнет получать предупреждения о необходимости сменить пароль. Учетная запись не имеет срока действия.

С помощью утилиты **pwconv** можно согласовать содержимое файлов **shadow** и **passwd**, выявляя и удаляя пользователей, не указанных в файле **passwd**.

8.4. Файлы /etc/master.passwd и /etc/login.conf в системе FreeBSD



Появление модулей PAM и наличие аналогичных команд управления пользователями в системах FreeBSD и Linux сделали управление учетными записями относительно согласованным между платформами, по крайней мере на самом верхнем уровне. Однако между ними существуют некоторые различия на уровне реализации.

Файл /etc/master.passwd

В системе FreeBSD "реальным" файлом паролей является **/etc/master.passwd**, который доступен только для чтения. Файл **/etc/passwd** предназначен для обратной совместимости и не содержит паролей (вместо этого он содержит символы * в качестве заполнителей).

Чтобы изменить файл паролей, выполните команду **vipw**. Эта команда вызывает ваш редактор для копии **/etc/master.passwd**, затем инсталлирует новую версию и повторно генерирует файл **/etc/passwd**, чтобы отразить все изменения. (Команда **vipw** является стандартной для всех систем UNIX и Linux, но особенно важно использовать ее в системе FreeBSD, потому что файлы с двойным паролем должны оставаться синхронизированными (см. раздел 8.6).)

Кроме полей файла **passwd** файл **master.passwd** содержит три дополнительных поля. К сожалению, по умолчанию они зажаты между полем GID и полем GECOS, заданными по умолчанию, поэтому форматы файлов напрямую не совместимы. Дополнительные три поля перечислены ниже:

- класс регистрации;
- время изменения пароля;
- время истечения срока действия пароля.

Класс регистрации (если он указан) относится к записи в файле **/etc/login.conf**. Этот класс определяет ограничения потребления ресурсов и управляет множеством других параметров (см. следующий раздел).

Поле времени изменения пароля отражает возраст пароля. Он содержит время в секундах с момента UNIX, после которого пользователь будет вынужден изменить свой пароль. Вы можете оставить это поле пустым, указав, что пароль будет действовать вечно.

Время истечения срока действия учетной записи означает время и дату (в секундах, как и для момента истечения срока действия пароля), по прошествии которого истекает срок действия учетной записи пользователя. Пользователь не может войти в систему после этой даты, если поле не будет сброшено администратором. Если это поле оставлено пустым, учетная запись останется действительной.

Файл /etc/login.conf

Файл **/etc/login.conf** в системе FreeBSD задает параметры учетной записи для пользователей и групп пользователей. Его формат состоит из пар ключ-значение с двоеточием и булевых флагов.

Когда пользователь входит в систему, поле входа в каталог **/etc/master.passwd** определяет, какую запись из файла **/etc/login.conf** применять. Если в записи пользователя **master.passwd** не указан класс входа в систему, используется класс по умолчанию.

Элемент **login.conf** может установить любое из следующих значений.

- Ограничения ресурсов (максимальный размер процесса, максимальный размер файла, количество открытых файлов и т.д.).
- Сеансовые ограничения (с какого момента и насколько долго разрешены регистрационные имена).
- Переменные окружения по умолчанию.
- Пути по умолчанию (PATH, MANPATH и т.д.).
- Расположение файла с сообщениями для всех пользователей.
- Хост и контроль доступа на основе подсистемы TTY.
- Стандартная маска для команды **umask**.
- Контроль учетных записей (в основном заменяется модулем PAM pam_passwdqc).

Следующий пример демонстрирует переопределение нескольких значений по умолчанию. Он предназначен для назначения системных администраторов.

```
sysadmin:\
   :ignorenologin:\
   :requirehome@:\
   :maxproc=unlimited:\
   :openfiles=unlimited:\
   :tc=default:
```

Пользователи, имеющие класс регистрации sysadmin, могут войти в систему, даже если их имя указано в файле **/var/run/nologin** и у них может не быть рабочего домашнего каталога (этот параметр позволяет регистрацию, когда система NFS не работает). Пользователи класса sysadmin могут запускать любое количество процессов и открывать любое количество файлов.[8] Последняя строка записывает информацию в запись default.

Хотя система FreeBSD имеет разумные стандартные значения, может возникнуть необходимость обновить файл **/etc/login.conf**, чтобы установить предупреждения об истечении времени ожидания или об истечении срока действия пароля. Например, чтобы установить время ожидания равным 15 мин. и выдавать предупреждения за семь дней до истечения срока действия паролей, необходимо добавить следующие элементы в запись default:

```
: warnpassword=7d:\
: idletime=15m:\
```

Если вы изменяете файл **/etc/login.conf**, вы также должны выполнить следующую команду для компиляции ваших изменений в хешированную версию файла, которую система фактически использует в повседневной работе:

```
$ cap_mkdb /etc/login.conf
```

[8] По-прежнему существуют технические ограничения на общее количество процессов и открытых файлов, которые может поддерживать ядро, но искусственно установленных ограничений нет.

8.5. Файл /etc/group

Файл **/etc/group** содержит имена UNIX-групп и списки членов каждой группы. Вот как выглядит фрагмент файла **group** из системы FreeBSD.

```
wheel:*:0:root
sys:*:3:root,bin
operator:*:5:root
bin:*:7:root
ftp:*:14:dan
staff:*:20:dan,ben,trent
nobody:*:65534:lpd
```

Каждая строка представляет одну группу и содержит следующие четыре поля:

- имя группы;
- зашифрованный пароль или заполнитель;
- идентификатор группы;
- список членов, разделенный запятыми (пробелов быть не должно).

Как и в файле **/etc/passwd**, поля разделяются двоеточиями. Длина имени группы не должна превышать 8 символов из соображений совместимости, хотя во многих системах такого ограничения нет.

Несмотря на возможность установить пароль группы, позволяющий любому пользователю присоединиться к группе, выполнив команду **newgrp**, она используется редко. Пароль можно установить с помощью команды **gpasswd**; а его зашифрованная форма в системе Linux сохраняется в файле **/etc/gshadow**.

Имена групп и их идентификаторы должны быть одинаковыми на всех компьютерах, получающих совместный доступ к файлам через сетевую файловую систему. Этого трудно достичь в гетерогенной среде, поскольку в разных операционных системах используются разные идентификаторы для одних и тех же групп.

Если в файле **/etc/passwd** пользователь объявлен членом определенной группы по умолчанию, а в файле **/etc/group** — нет, пользователь все равно включается в группу. На этапе регистрации членство в группе проверяется по обоим файлам, но лучше все же согласовывать их содержимое.

В некоторых старых системах ограничивается количество групп, к которым может принадлежать пользователь. В современных ядрах систем Linux и FreeBSD ограничений нет вообще.

Для того чтобы избежать конфликтов, связанных с идентификаторами стандартных групп, рекомендуем выбирать идентификаторы локальных групп, начиная со значения 1000.

Согласно традиции UNIX новые пользователи добавляются в группу, название которой отражает их категорию, например “students” или “finance”. Однако следует учитывать, что подобная традиция повышает вероятность доступа пользователей к файлам друг друга из-за неаккуратной установки прав.

Чтобы этого избежать, рекомендуем создавать для каждого пользователя уникальную группу с помощью утилит **useradd** и **adduser** (т.е. группу, имя которой совпадает с именем пользователя и содержит только данного пользователя). Это соглашение намного проще выполнить, чем обеспечить согласование между идентификаторами пользователя и группы.

Чтобы пользователи могли обмениваться файлами с помощью группового механизма, создайте отдельные группы. Идея персональных групп состоит в том, чтобы не препятство-

вать использованию групп как таковых, а просто создать более ограничительную группу *по умолчанию* для каждого пользователя, чтобы файлы не были предоставлены для совместного пользователя непреднамеренно. Можно также ограничить доступ к вновь созданным файлам и каталогам, установив маску **umask** по умолчанию вашего пользователя в файле запуска по умолчанию, например **/etc/profile** или **/etc/bashrc** (см. раздел 8.6).

🕮 Дополнительную информацию о программе **sudo** см. в разделе 3.2.

Членство в группах также может служить маркером для других контекстов или привилегий. Например, вместо того, чтобы вводить имя пользователя каждого системного администратора в файл **sudoers**, вы можете настроить программу **sudo** так, чтобы все члены группы "admin" автоматически имели привилегии **sudo**.



В системе Linux для создания, изменения и удаления групп предусмотрены команды **groupadd**, **groupmod** и **groupdel**.

Система FreeBSD для выполнения всех этих функций использует команду **pw**. Чтобы добавить пользователя `dan` в группу `staff`, а затем проверить, что изменение было правильно реализовано, необходимо выполнить следующие команды:

```
$ sudo pw groupmod staff -m dan
$ pw groupshow staff
staff:*:20:dan,evi,garth,trent,ben
```

8.6. Подключение пользователей вручную: основные действия

Прежде чем создавать учетную запись для нового пользователя в крупной коммерческой компании, правительственном учреждении или учебном заведении, важно заставить его подписать соглашение о правилах работы в системе. (Как?! У вас нет такого соглашения? Немедленно прочтите материал в разделе 31.9, чтобы узнать, для чего оно нужно и как его составлять.)

У пользователей нет веских причин подписывать такое соглашение, поэтому в интересах администратора убедить их сделать это, чтобы иметь рычаги влияния. После того как учетная запись создана, добиться подписи будет трудно, так что лучше получить ее заранее. Если существует такая возможность, прежде чем создавать учетную запись, получите подпись пользователя на бумажном документе.

Процесс подключения нового пользователя состоит из ряда этапов, определяемых системными требованиями; два связаны с формированием пользовательской среды, а еще несколько этапов может понадобиться для целей системного администрирования.

- Обязательные этапы:
  - отредактировать файлы **passwd** и **shadow** (или файл **master.passwd** в системе FreeBSD), чтобы создать учетную запись пользователя;
  - добавить запись нового пользователя в файл **/etc/group** (в действительности это не необходимо, а желательно);
  - задать первоначальный пароль;
  - создать для нового пользователя домашний каталог, назначив его владельца с помощью команды **chown** и задав режим доступа с помощью команды **chmod**;
  - настроить роли и права доступа (если вы используете RBAC; см. чуть ниже).

- Пользовательские этапы:
  - скопировать в домашний каталог пользователя стандартные конфигурационные сценарии.
- Административные этапы:
  - получить подпись пользователя на соглашении о правилах работы;
  - проверить правильность создания учетной записи;
  - добавить контактную информацию о пользователе, а также сведения о статусе учетной записи в административную базу данных.

Для выполнения этого списка действий необходимо иметь сценарий или утилиту, и во всех наших иллюстративных системах они предусмотрены в виде команд **adduser** и **useradd**.

Редактирование файлов passwd и group

Ручное редактирование файлов **passwd** и **group** уязвимо для ошибок и неэффективно, поэтому мы рекомендуем инструменты высокого уровня, такие как **useradd**, **adduser**, **usermod**, **pw** и **chsh**.

Если вам придется добавлять пользователя вручную, используйте команду **vipw**, чтобы отредактировать файлы **passwd** и **shadow** (в системе FreeBSD — файл **master.passwd**). По умолчанию выбирается редактор **vi**, но эту установку можно изменить, задав новое значение переменной среды EDITOR.[9] Более важно то, что команда **vipw** блокирует редактируемый файл, позволяя только одному пользователю редактировать его. Это не допускает никаких конфликтов между действиями разных пользователей.



В системах Linux после редактирования файла **passwd** команда **vipw** автоматически напомнит пользователю о необходимости отредактировать файл **shadow**. Для этого используется команда **vipw -s**.



В системе FreeBSD команда **vipw** редактирует файл **master.passwd**, а не **/etc/passwd**. После внесения изменений команда **vipw** запускает выполнение команды **pwd_mkdb** для генерирования производного файла **passwd** и двух хешированных версий **master.passwd** (одна из них содержит зашифрованные пароли, доступные для чтения только пользователю root, а вторая не содержит паролей и доступна для чтения любому пользователю).

Например, выполнение команды **vipw** и добавление следующей строки приведет к определению учетной записи с именем whitney.

```
whitney:*:1003:1003::0:0:Whitney Sather, AMATH 3-27, x7919,:
    /home/staff/whitney:/bin/sh
```

Обратите внимание на звездочку в поле зашифрованного пароля. Она предотвращает использование учетной записи, пока не будет установлен настоящий пароль с помощью команды **passwd** (см. следующий раздел).

Затем отредактируйте **/etc/group**, выполнив команду **vigr**. Добавьте строку для новой персональной группы, если ваша организация использует их, и добавьте имя пользователя для каждой группы, в которой пользователь должен иметь членство.

---

[9] Если сначала выполнить команду **vipw** (или **vigr**), системы Ubuntu и Debian предложат вам выбрать одну из команд **vim.basic**, **vim.tiny**, **nano** или **ed**. Если впоследствии вы передумаете, выполните команду **select-editor**.

Как и в случае с командой **vipw**, использование команды **vigr** гарантирует, что изменения, внесенные в файл **/etc/group**, являются корректными и атомарными. После сеанса редактирования команда **vigr** должна попросить вас выполнить команду **vigr -s** для редактирования файла теневой группы (**gshadow**). Если вы не хотите устанавливать пароль для группы, что необычно, вы можете пропустить этот шаг.



Чтобы внести изменения в файл **/etc/group**, в системе FreeBSD используется команда **pw groupmod**.

Задание пароля

Правила выбора удачных паролей приведены в разделе 27.3.

Установите пароль для нового пользователя с помощью следующей команды.

```
$ sudo passwd новое-имя-пользователя
```

В некоторых автоматизированных системах при добавлении новых пользователей необязательно вводить начальный пароль. Задание пароля в таких случаях происходит при первой регистрации. Несмотря на удобство, этот вариант приводит к образованию огромной дыры в системе безопасности; тот, кто может угадать новые регистрационные имена (или узнать их в файле **/etc/passwd**), может выполнить "пиратские" действия по отношению к учетным записям, прежде чем соответствующие пользователи получат возможность зарегистрироваться.



Помимо других функций команда **pw** в системе FreeBSD генерирует и устанавливает случайные пароли пользователей.

```
$ sudo pw usermod raphael -w random
   Password for 'raphael' is: 1n3tcYu1s
```

Мы не являемся горячими сторонниками использования случайных паролей. Однако они бывают полезными в качестве временных паролей, которые подлежат замене после использования.

Создание домашнего каталога пользователя и инсталляция конфигурационных файлов

Новые домашние каталоги пользователей создаются с помощью команд **useradd** и **adduser**, но вы вряд ли захотите выполнять двойную проверку прав доступа и файлов запуска для новых учетных записей.

Домашний каталог создать легко. Если вы забыли это сделать при создании учетной записи нового пользователя, исправить положение можно с помощью простой команды **mkdir**. Вам также понадобится установить права владения на новый каталог и права доступа к нему, но лучше всего сделать это после инсталляции локальных конфигурационных файлов.

Имена таких файлов традиционно начинаются с точки и заканчиваются суффиксом **rc** (сокращение от "run command" — команда запуска) — "отголосок" операционной системы CTSS. Предшествующая точка заставляет команду **ls** не включать эти файлы в листинги каталогов, если только не указана опция **-a**. Мы рекомендуем предоставлять стандартные файлы запуска для всех интерпретаторов команд, которые популярны в ваших системах, чтобы пользователи по умолчанию продолжали работать в корректной среде даже при смене командной оболочки. Наиболее часто встречающиеся конфигурационные файлы перечислены в табл. 8.2.

Таблица 8.2. Распространенные конфигурационные файлы

Команда	Имя файла	Применение
Все оболочки	`.login_conf`	Настройки параметров регистрации пользователя по умолчанию (FreeBSD)
sh	`.profile`	Установка пути поиска, типа терминала и среды
bash[a]	`.bashrc`	Установка типа терминала (если это необходимо)
	`.bash_profile`	Установка опций программ **biff** и **mesg** Установка переменных окружения Установка псевдонимов команд Установка пути поиска Установка атрибута **umask** для управления правами доступа Установка переменной `CDPATH` для поиска имен файлов Установка переменных `PS1` (строка приглашения) и `HISTCONTROL`
csh/tcsh	`.login`	Считывается всеми зарегистрированными экземплярами интерпретатора **csh**
	`.cshrc`	Считывается всеми экземплярами интерпретатора **csh**
vi/vim	`.exrc/.vimrc`	Установка опций редакторов **vi/vim**
emacs	`.emacs`	Установка опций редактора **emacs** и функциональная привязка его клавиш
git	`.gitconfig`	Установка параметров пользователя, редактора, цвета и псевдонимов в системе Git
GNOME	`.gconf`	Среда GNOME: конфигурация среды пользователя посредством команды **gconf**
	`.gconfpath`	Путь для дополнительного варианта конфигурации среды пользователя посредством команды **gconf**
KDE	`.kde/`	Среда KDE: каталог файлов конфигурации

[a] Оболочка **bash** также читает файлы **.profile** или **/etc/profile** в режиме эмуляции **sh**. Файл **.bash_profile** считывается командами регистрации, а файл **.bashrc** считывается с помощью интерактивных оболочек без регистрации.

Образцы файлов запуска традиционно хранятся в каталоге **/etc/skel**. Если вы будете редактировать примеры конфигурационных файлов, каталог **/usr/local/etc/skel** — самое удачное место для хранения модифицированных копий.

Конфигурационные файлы и каталоги, перечисленные для настольных сред GNOME и KDE, представляют собой лишь вершину айсберга. В частности. обратите внимание на утилиту **gconf**, предназначенную для хранения предпочтений в выборе приложений для программ под управлением GNOME (подобно тому, как это реализовано в системном реестре Windows).

🕮 Дополнительную информацию об атрибуте **umask** см. в разделе 5.5.

Не забудьте задать разумное значение **umask** (рекомендуем 077, 027 или 022, в зависимости от "дружелюбности" среды и размеров организации). Если вы не используете индивидуальные группы, рекомендуем установить для **umask** значение 077, поскольку оно предоставляет владельцу полный доступ, а для группы и вообще для всех остальных — никакого доступа.

В зависимости от интерпретатора, в каталоге **/etc** могут содержаться системные конфигурационные файлы, обрабатываемые раньше пользовательских. Например, интерпретаторы **bash** и **sh** читают файл **/etc/profile** до начала обработки файла

~/.profile и **~/.bash_profile**. В эти файлы стоит помещать стандартные установки, необходимые для функционирования вашего хоста, но следует иметь в виду, что пользователи могут переопределять свои установки в собственных конфигурационных файлах. Информацию о других интерпретаторах можно найти в документации (см. соответствующие **man**-страницы).



По соглашению, система Linux сохраняет фрагменты загрузочных файлов в каталоге **/etc/profile.d**. Хотя имя каталога происходит из соглашений, принятых для оболочки sh, файл **/etc/profile.d** может фактически включать фрагменты для нескольких разных оболочек. Конкретные целевые оболочки различаются суффиксами имен файлов (***.sh**, ***.csh** и т.д.). В оболочке не встроена поддержка файла **profile.d**; фрагменты просто исполняются сценарием запуска по умолчанию в каталоге **/etc** (например, **/etc/profile** в случае оболочки **sh** или **bash**).

Разделение файлов запуска, заданных по умолчанию, на фрагменты облегчает модульность и позволяет программным пакетам включать собственные значения по умолчанию на уровне оболочки. Например, фрагменты **colorls.*** указывают, как правильно раскрасить вывод команды **ls**, чтобы сделать его нечитаемым на темном фоне.

Установка прав доступа и владения

После установки домашнего каталога переходите к пользователю и убедитесь, что он имеет соответствующие права доступа. Команда

```
$ sudo chown -R новый_пользователь:новая_группа ~новый_пользователь
```

должна надлежащим образом установить права владения. Обратите внимание на невозможность использования команды

```
$ sudo chown -R новый_пользователь:новая_группа ~новый_пользователь/.*
```

применительно к файлам, имена которых начинаются с точки, поскольку *новый_пользователь* станет владельцем не только собственных файлов, но и родительского каталога "**..**" (например, /home). Это распространенная и опасная ошибка.

Конфигурирование ролей и административных привилегий

Управление доступом на основе ролей (role-based access control — RBAC) позволяет "подогнать" системные права к отдельным пользователям и обеспечивается на многих наших примерах систем. Политика RBAC не является традиционной частью модели управления доступом UNIX или Linux, но если на вашем хосте она используется, то конфигурация ролей должна быть частью процесса добавления пользователей. Модель RBAC подробно описана в разделе 3.4.

🕮 Подробнее о законах SOX и GLBA написано в главе 31.

Принятые в США закон Сарбейнса–Оксли (Sarbanes–Oxley Act), закон о преемственности страхования и отчетности в области здравоохранения (Health Insurance Portability and Accountability Act — HIPAA) и закон Грэмма–Лича–Блайли (Gramm–Leach–Bliley Act), предназначенные, в частности, для защиты информации заказчика, полученной или используемой финансовыми организациями США, от кражи, неавторизованного доступа либо злоупотребления, усложнили многие аспекты системного администрирования в корпоративной сфере, включая управление пользователями. Поэтому использование ролей

может быть вашим единственным жизнеспособным вариантом, позволяющим выполнить некоторые требования, устанавливаемые законами SOX, HIPAA и GLBA.

Заключительные действия

Для того чтобы удостовериться, что новая учетная запись сконфигурирована надлежащим образом, завершите сеанс работы (т.е. выйдите из системы), а затем снова войдите в систему под именем нового пользователя и выполните следующие команды.

```
$ pwd          # Для проверки домашнего каталога
$ ls -la       # Для проверки владельца/группы файлов конфигурации
```

Вам необходимо уведомить новых пользователей об их регистрационных именах и начальных паролях. Многие хосты отправляют эту информацию по электронной почте, но из соображений безопасности это не приветствуется. Делайте это при личном контакте или по телефону. Но если вам приходится добавлять сразу 500 новичков на университетские компьютеры CS-1, переложите эту задачу на преподавателей! Если вам приходится распространять пароли через электронную почту, проследите за тем, чтобы срок их действия истекал через несколько дней, если они не используются и не были изменены.

🕮 Подробнее о заключении письменных контрактов с пользователями можно прочитать в разделе 31.9.

Если в вашей организации требуется, чтобы пользователи подписывали соответствующее соглашение, убедитесь в реализации этого действия до создания их учетных записей. Такая проверка предотвратит возможные упущения и усилит правовую основу санкций, которые, возможно, вам придется позже применить. Кроме того, это подходящий момент для того, чтобы ознакомить пользователей с дополнительной документацией о локальных настойках.

Напомните новым пользователям о необходимости немедленного изменения их паролей. При желании это можно обеспечить путем установки действия паролей в течение некоторого короткого времени. Еще один вариант состоит в создании сценария, который бы проверял новых пользователей и удостоверялся в том, что их зашифрованные пароли были изменены[10].

Если вы знаете пользователей лично, то в такой среде относительно несложно проследить, кто использует систему и с какой целью. Но если в подчинении администратора большой и часто меняющийся штат пользователей, необходим более формальный способ контроля учетных записей. Ведение базы данных с контактной информацией и сведениями о статусе учетных записей поможет в случае необходимости, легко выяснить, кто есть кто и зачем тому или иному пользователю понадобилась учетная запись.

8.7. Добавление пользователей с помощью сценариев: useradd, adduser и newusers

Во всех иллюстративных системах сценарии **useradd** и **adduser** реализуют одну и ту же базовую процедуру, описанную выше. Но ее можно сконфигурировать под конкретную среду. К сожалению, в каждой системе существуют свои представления о том, что именно

[10]Поскольку один и тот же пароль может иметь множество зашифрованных представлений, этот метод проверяет только сам факт изменения пользователем своего пароля, а не то, что пароль реально был заменен другим паролем.

следует настраивать, где следует реализовать настройки и каково должно быть стандартное поведение, поэтому мы описываем эти подробности в соответствующих разделах.

В табл. 8.3 приведены команды и файлы конфигурации, имеющие отношение к управлению пользователями.

Таблица 8.3. Команды и файлы конфигурации для управления пользователями

Система	Команды	Конфигурационные файлы
Все версии Linux	`useradd`, `usermod`, `userdel`	`/etc/login.defs` `/etc/default/useradd`
Debian/Ubuntu[a]	`adduser`, `deluser`	`/etc/adduser.conf` `/etc/deluser.conf`
FreeBSD	`adduser`, `rmuser`	`/etc/login.defs`

[a] В этот набор входят не только стандартные функции Linux, но и некоторые дополнительные функции.

Команда `useradd` в системе Linux



В системе Linux предусмотрен сценарий `useradd`, извлекающий параметры конфигурации из файлов `/etc/login.defs` и `/etc/default/useradd`.

Файл `login.defs` предназначен для решения таких проблем, как старение пароля, выбор алгоритмов шифрования, расположение файлов почтовых каталогов и предпочтительные диапазоны UID и GID. Редактирование файла `login.defs` выполняется вручную. Полезным средством для понимания параметров являются комментарии.

Параметры, хранящиеся в файле `/etc/default/useradd`, задают расположение домашних каталогов и оболочку по умолчанию для новых пользователей. Эти значения устанавливаются по умолчанию с помощью команды `useradd`. Например, команда `useradd -D` выводит на экран текущие значения, а параметр `-D` в сочетании с другими флагами задает значения определенных опций. Например, команда

```
$ sudo useradd -D -s /bin/bash
```

устанавливает `bash` как оболочку по умолчанию.

Типичными опциями по умолчанию являются размещение новых пользователей в отдельных группах, использование алгоритма шифрования SHA-512 для паролей и заполнение домашних каталогов новых пользователей загрузочными файлами из каталога `/etc/skel`.

Основная форма команды `useradd` принимает имя новой учетной записи в командной строке:

```
$ sudo useradd hilbert
```

Эта команда создает в файле `/etc/passwd` запись, подобную показанной ниже, а также соответствующую запись в файле `shadow`:

```
hilbert:x:1005:20::/home/hilbert:/bin/sh
```

По умолчанию команда `useradd` блокирует новую учетную запись. Для фактического использования учетной записи необходимо назначить реальный пароль.

Более реалистичный пример показан ниже. Мы указываем, что первичная группа пользователя `hilbert` должна иметь имя "hilbert" и что он также должен быть добавлен в группу "faculty". Мы переопределяем местоположение и оболочку основного каталога по умолчанию и попросим команду `useradd` создать домашний каталог, если он еще не существует:

```
$ sudo useradd -c "David Hilbert" -d/home/math/hilbert -g hilbert
    -G faculty -m -s /bin/tcsh hilbert
```

Эта команда создает следующую запись в файле **passwd**:

```
hilbert:x:1005:30:David Hilbert:/home/math/hilbert:/bin/tcsh
```

Назначенный идентификатор UID превышает наивысший UID, существующий в системе, а соответствующая запись в файле **shadow** имеет вид:

```
hilbert::14322:0:99999:7:0::
```

Символ (символы) заполнения пароля в файлах **passwd** и **shadow** различаются в зависимости от операционной системы. Команда **useradd** также добавляет пользователя `hilbert` в соответствующие группы в файле **/etc/group**, создает каталог **/home/math/hilbert** с надлежащими владельцами и заполняет его файлами из каталога **/etc/skel**.

Команда adduser в системах Debian и Ubuntu



В дополнение к семейству команд **useradd**, линия Debian также предоставляет несколько сценариев более высокого уровня для этих команд в виде **adduser** и **deluser**. Эти дополнительные команды настраиваются в файле **/etc/adduser.conf**, где можно указать такие параметры:

- правила размещения домашних каталогов: по группам, по имени пользователя и т.д.;
- настройки разрешения для новых домашних каталогов;
- диапазоны UID и GID для системных и общих пользователей;
- возможность создания отдельных групп для каждого пользователя;
- квоты диска (только булевские, к сожалению);
- сопоставление имен пользователей и групп с помощью регулярных выражений.

Другие типичные параметры команды **useradd**, такие как правила для паролей, устанавливаются как параметры модуля PAM, который выполняет обычную аутентификацию пароля. (Подключаемые модули аутентификации PAM обсуждаются в разделе 17.3.) У команд **adduser** и **deluser** есть аналоги для групп: **addgroup** и **delgroup**.

Команда adduser в системе FreeBSD



Система FreeBSD поставляется со сценариями оболочки **adduser** и **rmuser**, которые можно использовать в готовом виде или модифицировать в соответствии с вашими потребностями. Сценарии построены на основе средств, предоставленных командой **pw**.

При желании команду **adduser** можно использовать интерактивно. По умолчанию она создает записи пользователей и групп и домашний каталог. Можно указать сценарию файл после флага **-f**, содержащий список учетных записей для создания, или ввести каждого пользователя в интерактивном режиме.

Например, процесс создания нового пользователя `raphael` выглядит так.

```
$ sudo adduser
Username: raphael
Full name: Raphael Dobbins
Uid (Leave empty for default): <return>
Login group [raphael]: <return>
```

```
Login group is raphael. Invite raphael into other groups? []: <return>
Login class [default]: <return>
Shell (sh csh tcsh bash rbash nologin) [sh]: bash
Home directory [/home/raphael]: <return>
Home directory permissions (Leave empty for default): <return>
Use password-based authentication? [yes]: <return>
Use an empty password? (yes/no) [no]: <return>
Use a random password? (yes/no) [no]: yes
Lock out the account after creation? [no]: <return>
Username      : raphael
Password      : <random>
Full Name     : Raphael Dobbins
Uid           : 1004
Class         :
Groups        : raphael
Home          : /home/raphael
Home Mode     :
Shell         : /usr/local/bin/bash
Locked        : no
OK? (yes/no)  : yes
adduser: INFO: Successfully added (raphael) to the user database.
adduser: INFO: Password for (raphael) is: RSCAds5fy0vxOt
Add another user? (yes/no): no
Goodbye!
```

Команда `newusers` в системе Linux: добавление пользователей пакетом



При выполнении Linux-команды **newusers** одновременно создается несколько учетных записей из содержимого подготовленного текстового файла. Это не вполне формальный способ, но им удобно пользоваться в случае, если нужно добавить много пользователей сразу, например при создании учетных записей для учебной группы. Команда **newusers** в качестве аргумента получает входной файл, состоящий из строк, подобно файлу **/etc/passwd**, за исключением того, что его поле пароля содержит начальный пароль, заданный открытым текстом. Поэтому хорошо подумайте о защите такого файла.

Команда **newusers** принимает связанные со сроком действия пароля параметры, установленные в файле **/etc/login.defs**, но не копирует стандартные конфигурационные файлы, как это делает команда **useradd**. Единственный файл, который предоставляется в этом случае, — файл **.xauth**.

В университетских системах действительно важно применять "пакетный" сценарий **adduser**, который использует список студентов из данных регистрации для генерирования входной информации для команды **newusers**. Для этого необходимо иметь имена пользователей, подготовленные в соответствии с локальными правилами при гарантии их уникальности (на локальном уровне), а также метод случайного генерирования паролей при увеличивающихся значениях идентификаторов пользователей и групповых идентификаторов для каждого студента. Возможно, вам лучше написать собственную оболочку для команды **useradd** на языке Python, чем пытаться приспособить свои нужды к тому, что предлагает команда **newusers**.

8.8. Безопасное удаление учетных записей пользователей и файлов

Когда пользователь покидает организацию, его учетная запись и файлы должны быть удалены из системы. Эта процедура включает удаление всех ссылок на регистрационное имя, которые были введены вручную или с помощью команд **useradd** и **rmuser**. При ручном удалении учетной записи последовательность операций будет примерно такой:

- удалить записи пользователя из локальных баз данных и телефонных списков;
- удалить пользовательские почтовые псевдонимы из файла либо организовать перенаправление поступающих ему сообщений;
- удалить пользовательские задания из файла **crontab**, из очереди команды **at**, а также из заданий на печать;
- уничтожить пользовательские процессы, которые еще выполняются;
- удалить записи пользователя из файлов **passwd**, **shadow**, **group** и **gshadow**;
- удалить домашний каталог пользователя;
- удалить почтовый каталог пользователя (если почта хранится локально);
- удалить записи в совместно используемых календарях, системах бронирования номеров и пр.
- удалить или преобразовать права владения любыми списками рассылки, используемыми удаленным пользователем.

Перед тем как удалять домашний каталог пользователя, необходимо переместить из него в другие каталоги все файлы, которые нужны остальным пользователям. Поскольку не всегда можно с уверенностью сказать, какие файлы понадобятся, а какие — нет, лучше предварительно скопировать пользовательские домашний и почтовый каталоги на какой-то носитель.

После удаления учетной записи пользователя убедитесь, что в системе не осталось файлов с его идентификатором. Чтобы узнать точный путь к этим файлам, воспользуйтесь командой **find** с аргументом **-nouser**. Поскольку команда **find** может вести поиск на сетевых серверах, проверяйте файловые системы по отдельности, задавая опцию **-xdev**.

```
$ sudo find filesystem -xdev -nouser
```

Если в организации за пользователями закреплены отдельные рабочие станции, эффективнее всего переинсталлировать систему, прежде чем предоставлять ее новому пользователю. Но не забудьте сбросить на системный жесткий диск локальные файлы, которые могут понадобиться в будущем.[11]

Каждая система из наших примеров имеет команды, с помощью которых она автоматизирует процесс удаления пользователя. Если эти команды не отвечают вашим высоким требованиям, можете дополнить их функциональные возможности предоставлением расширенного списка мест хранения данных о пользователях.



В системах Debian и Ubuntu команда **deluser** представляет собой сценарий на языке Perl, который вызывает обычную команду **userdel**, аннулируя все результаты работы команды **adduser**. При этом вызывается сценарий **user/local/sbin/deluser.local**, если таковой существует, чтобы реализовать несложную операцию локализации. Файл конфигурации **/etc/deluser.conf** позволяет установить следующие опции:

[11] Помните о лицензионных ключах!

- удалять ли домашний и почтовый каталоги пользователя;
- резервировать ли файлы пользователя и куда поместить резервные копии;
- удалять ли все файлы, которыми владел пользователь;
- удалять ли группу, если в данный момент она не имеет членов.

RHEL  В системе Red Hat предусмотрен сценарий **userdel.local**, но не предусмотрены префиксные и постфиксные сценарии для автоматизации операций по резервированию файлов пользователя, подлежащего удалению.



Сценарий **rmsuser** в системе FreeBSD представляет собой превосходный инструмент для удаления файлов и процессов пользователей. С ним не могут сравниться никакие программы **userdel** других поставщиков.

8.9. Блокирование регистрационных имен пользователей

Иногда возникает необходимость временно отключить учетную запись пользователя. Проще всего сделать это, поставив звездочку (*) или какой-то другой символ в поле зашифрованного пароля в файле **/etc/shadow** или **/etc/master.passwd**. Эта мера препятствует большинству типов доступа, управляемых паролем, поскольку дешифровка больше не может привести к получению какого-либо приемлемого пароля.



Система FreeBSD позволяет блокировать учетные записи с помощью команды **pw**. Например, команда

```
$ sudo pw lock пользователь
```

помещает строку *LOCKED* в начало хешированного пароля, блокируя учетную запись.



Во всех дистрибутивах Linux, рассматриваемых нами в качестве примеров, команды **usermod -L** *пользователь* и **usermod -U** *пользователь* позволяют легко соответственно блокировать и разблокировать пароли. Флаг **-L** просто помещает символ "!" перед зашифрованным паролем в файле **/etc/shadow**, а флаг **-U** удаляет его.

К сожалению, модификация пароля пользователя просто приводит к блокированию регистрации. При этом пользователь не уведомляется о блокировке пароля и не получает разъяснений, почему его учетная запись больше не работает. Альтернативный способ блокировать регистрационные имена состоит в замене интерпретатора команд пользователя программой, которая выдает сообщение, поясняющее, почему учетная запись отключена, и содержащее инструкции по исправлению ситуации. Затем программа завершается и вместе с ней заканчивается сеанс регистрации.

Этот метод имеет как преимущества, так и недостатки. Те формы доступа, которые проверяют пароли, но не обращают внимания на интерпретатор команд, не будут заблокированы. Многие демоны, предоставляющие доступ к системе без регистрации (например, **ftpd**), проверяют, упомянут ли интерпретатор пользователя в файле **/etc/shells**. Если он там не указан, вход в систему будет запрещен (именно это нам и требуется). К сожалению, этот метод нельзя назвать универсальным. Поэтому, если для блокирования учетных записей вы решите модифицировать интерпретатор команд, вам придется провести дополнительные исследования своей системы.

Кроме того, поясняющее сообщение нельзя увидеть, если пользователь пытается зарегистрироваться в системе в оконной среде или посредством эмулятора терминала, который не позволяет увидеть сообщения после выхода из системы.

8.10. Уменьшение риска с помощью модулей PAM

Подключаемые модули аутентификации (Pluggable Authentication Modules — PAM) подробно описаны в разделе 17.3. В них сосредоточено управление системными средствами аутентификации посредством стандартных библиотечных утилит, чтобы такие программы, как **login**, **sudo**, **passwd** и **su**, не должны были предоставлять собственный код аутентификации. Модули PAM уменьшают риск, присущий отдельно взятым программным продуктам, позволяют администраторам устанавливать политики безопасности, действующие в рамках всего хоста сети, а также определяют простой способ добавления в систему новых методов аутентификации.

Добавление и удаление учетных записей пользователей не предусматривает изменение конфигурации модулей PAM, но используемые для этих целей инструменты должны действовать по правилам PAM и в соответствии с требованиями PAM. Кроме того, многие параметры конфигурации PAM подобны тем, которые используются командами **useradd** или **usermod**. Если вы измените какой-нибудь параметр (как описано ниже в этой главе), а команда **useradd**, казалось бы, не обратит на это внимания, проверьте, не аннулировала ли системная конфигурация PAM ваше новое значение.

8.11. Централизация управления учетными записями

В средних и крупных корпорациях всех типов (академических, частных или государственных) важно использовать определенную форму централизованного управления учетными записями. Каждый пользователь должен иметь на хосте уникальное, удобное и безопасное регистрационное имя, идентификатор (UID) и пароль. Администраторам нужна централизованная система, которая бы немедленно обеспечивала повсеместное распространение изменений (например, аннулирование конкретной учетной записи).

Такая централизация может быть достигнута различными способами, большинство из которых (включая систему Active Directory компании Microsoft) в той или иной степени (от бесплатных инсталляций с открытым исходным кодом до дорогих коммерческих систем) использует упрощенный протокол доступа к сетевым каталогам (Lightweight Directory Access Protocol — LDAP).

Протокол LDAP и служба Active Directory

🕮 Подробнее о протоколе LDAP и его реализациях рассказывается в разделе 17.2.

Протокол LDAP представляет собой обобщенный репозиторий (наподобие базы данных), который может хранить данные, связанные с управлением пользователями, а также другие типы данных. Он использует иерархическую модель "клиент-сервер", которая поддерживает несколько серверов и несколько одновременно работающих клиентов. Одно из самых больших преимуществ LDAP как централизованного репозитория для учетных данных состоит в том, что он может обеспечить в системах уникальность идентификаторов UID и GID. Кроме того, он хорошо стыкуется с Windows, хотя обратное утверждение справедливо лишь в самой малой степени.

Служба Active Directory компании Microsoft использует протоколы LDAP и Kerberos (сетевой протокол аутентификации) и может управлять множеством типов данных, включая данные пользователей. Она ведет себя несколько "эгоистично", навязывая "свою линию" при взаимодействии с UNIX- или Linux-репозиториями LDAP. Если вам нужна единая система аутентификации для хоста, который включает Windows-компьютеры, а также

UNIX- и Linux-системы, то, возможно, проще всего позволить службе Active Directory использовать ваши UNIX-базы данных LDAP в качестве вспомогательных серверов.

Подробнее вопросы интеграции систем UNIX и Linux со службами LDAP, Kerberos и Active Directory см. в главе 17.

Системы "единого входа"

Системы с однократной идентификацией пользователей (single sign-on — SSO) уравновешивают удобства пользователя с проблемами безопасности. Их основная идея состоит в том, чтобы пользователь, один раз зарегистрировавшись (в ответ на соответствующее приглашение на веб-странице или в окне Windows), мог затем переходить, к примеру, из одного раздела в другой без повторной аутентификации. Иначе говоря, пользователь получает аутентификационный мандат (обычно в неявном виде, чтобы не требовалось больше никакого активного управления), который затем можно использовать для доступа к другим компьютерам и приложениям. Пользователь должен помнить только одну последовательность (а не много), состоящую из регистрационного имени и пароля.

Эта схема позволяет усложнять мандаты (поскольку пользователю не нужно помнить о них и вообще иметь с ними дело), которые теоретически повышают степень сложности. Однако влияние скомпрометированной учетной записи в этом случае усиливается, поскольку одно регистрационное имя предоставляет взломщику доступ сразу к нескольким компьютерам и приложениям. Системы SSO переносят "арену действий" из настольного компьютера (пока вы спокойно регистрировались) в зону особой уязвимости. Кроме того, узким местом системы становится сервер аутентификации. Если он "упадет", вся работа организации остановится.

Несмотря на то что в основе системы SSO лежит простая идея, она предполагает большую внутреннюю сложность, поскольку различные приложения и компьютеры, к которым пользователь хотел получить доступ, должны понимать процесс аутентификации и SSO-мандаты.

Существует ряд систем SSO с открытым исходным кодом:

- JOSSO, сервер SSO с открытым исходным кодом, написанный на языке Java;
- служба CAS (Central Authentication Service), созданная Йельским университетом (на языке Java);
- продукт Snibboleth, система SSO с открытым кодом, распространяемая по лицензии Apache 2.

Существуют также коммерческие системы, большинство из которых интегрировано с семействами программных продуктов, предназначенными для управления учетными данными (о них пойдет речь в следующем разделе).

Системы управления учетными данными

Управление учетными данными (identity management) — это последний писк моды в области управления пользователями. Если говорить проще, то этот термин означает идентификацию пользователей конкретной системы, аутентифицирование их личностей и предоставление привилегий на основе этих аутентифицированных личностей. Стандартизация этой деятельности проводится такими организациями, как World Wide Web Consortium и The Open Group.

Коммерческие системы управления учетными данными сочетают несколько ключевых концепций UNIX в виде графического интерфейса, "приправленного" терми-

нами из сферы маркетинга. Основным элементом для всех этих систем является база данных, которая содержит информацию, связанную с аутентификацией и авторизацией пользователей, часто хранимую в формате LDAP. Управление реализуется на базе таких понятий, как группы UNIX, а ограниченные административные права усиливаются посредством таких утилит, как `sudo`. Большинство таких систем было разработано с целью урегулирования требований, диктуемых с точки зрения возможности идентификации, использования контрольных записей и слежения за ними.

Создано много коммерческих систем в этой сфере, например: Identity Management (Oracle), Sun Identity Management Suite[12], Courion, Avatier Identity Management Suite (AIMS) и BMC Identity Management Suite. Оценивая системы управления учетными данными, ищите следующие функции.

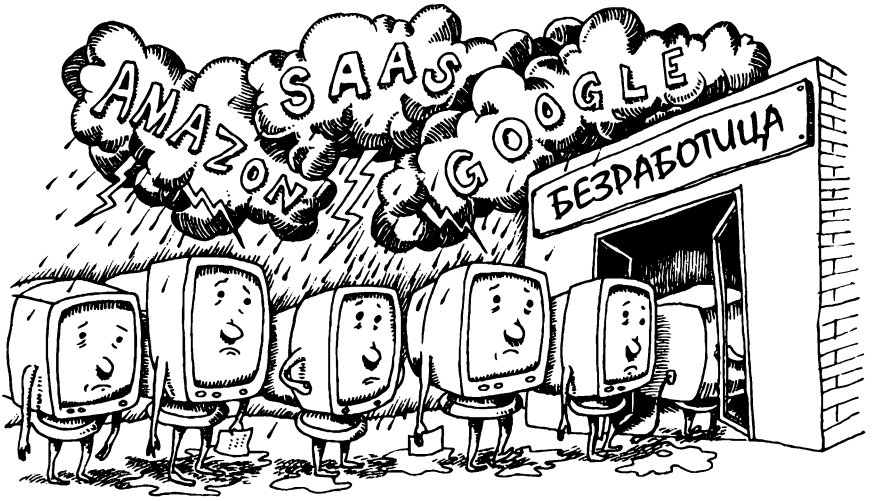
- Контроль:
  - безопасный веб-интерфейс для управления, доступного как изнутри, так и снаружи организации;
  - интерфейс, при использовании которого наем менеджеров может потребовать, чтобы учетные записи предоставлялись в соответствии с ролью;
  - возможность согласования с базами данных кадров для автоматического удаления доступа для служащих, которые уволены или сокращены.
- Управление учетными записями:
  - генерирование глобальных уникальных идентификаторов пользователей;
  - возможность создания, изменения и удаления пользовательских учетных записей в организации на оборудовании и операционных системах всех типов;
  - возможность простого отображения имен всех пользователей, которые имеют определенный набор привилегий;
  - простой способ отобразить всех пользователей, имеющих определенные привилегии, а также все привилегии, предоставленные конкретному пользователю;
  - управление доступом на основе ролей, включая инициализацию учетных записей пользователей с использованием ролевого принципа; создание исключений для предоставления прав на основе ролей, включая организацию действий для рассмотрения исключений;
  - регистрация с перестраиваемой конфигурацией всех изменений и действий администратора и создание реконфигурируемых отчетов на основе регистрационных данных (по пользователю, по дате и пр.).
- Удобство использования:
  - возможность для пользователей менять их пароли глобально в одной операции;
  - возможность разрешить пользователям изменять (и восстанавливать) собственные пароли при соблюдении правил выбора сильных паролей.

Рассмотрите также, как реализована система с точки зрения того, какие в действительности выполняются аутентификации и авторизации. Требуется ли системе повсеместная инсталляция пользовательских агентов или возможно самостоятельное согласование с базовыми системами?

[12]Теперь, когда компания Oracle купила Sun, не ясно, выживет ли эта система как продукт после завершения процесса поглощения.

Глава 9

Облачные вычисления



Облачные вычисления — это практика лизинга компьютерных ресурсов из пула общей емкости. Облачные службы предоставляют пользователям ресурсы по их требованию и получают за это установленную цену. Компании, использующие облако, быстрее выходят на рынок, демонстрируют большую гибкость и несут меньшие капитальные и эксплуатационные расходы, чем предприятия, управляющие традиционными центрами обработки данных.

Облако — это реализация "коммунальных вычислений", впервые задуманная ученым-программистом Джоном Мак-Карти, который описал эту идею в одной беседе в Массачусетском технологическом институте в 1961 году. Многочисленные технологические достижения, появившиеся за прошедшее время, помогли воплотить эту идею в жизнь. Перечислим лишь некоторые из них.

- Программное обеспечение виртуализации надежно выделяет ресурсы центральных процессоров, памяти, хранилища и сети по требованию.
- Надежные слои безопасности изолируют пользователей и виртуальные машины друг от друга, даже если они совместно используют базовое оборудование.
- Стандартизированные аппаратные компоненты позволяют создавать центры обработки данных с огромными мощностями, ресурсами хранения и системами охлаждения.
- Надежная глобальная сеть соединяет все это вместе.

Облачные провайдеры используют эти и многие другие инновации. Они предлагают множество услуг, начиная от размещения частных серверов и заканчивая полностью управляемыми приложениями. Ведущие поставщики облачных технологий являются конкурентоспособными, высокодоходными и быстро растущими компаниями.

В этой главе излагаются мотивы перехода в облако, описываются некоторые из основных поставщиков облачных вычислений, рассматриваются самые важные облачные службы и предлагаются советы по контролю затрат. В качестве еще более краткого введения в разделе 9.4 показано, как создавать облачные серверы из командной строки.

Управление облачными серверами упоминается еще в нескольких разделах этой книги, которые перечислены в таблице 9.1.

Таблица 9.1. Разделы, в которых рассматриваются темы, связанные с облачными вычислениями

Раздел	Название и тема раздела или подраздела
2.10	*Восстановление облачных систем* (проблемы, связанные с загрузкой для облака)
13.15	*Облачные сети* (сеть TCP/IP для облачных платформ)
19.3	*Веб-хостинг в облаке*
24.6	*Packer* (использование инструмента Packer для создания изображений ОС для облака)
25.4	*Контейнерная кластеризация и управление* (особенно в разделе, посвященном AWS ECS)
26.4	*Непрерывная интеграция и развертывание на практике* (пример конвейера CI/CD, использующего облачные службы)
28.7	*Инструменты мониторинга коммерческих приложений* (инструменты мониторинга для облака)

Кроме того, облачные системы часто упоминаются в главе 23.

9.1. Облако в контексте

Переход от серверов в частных центрах обработки данных к современному вездесущему облаку был быстрым и драматичным. Давайте посмотрим на причины этого массового движения.

Облачные провайдеры создают технически развитую инфраструктуру, которую большинство компаний не могут себе позволить по финансовым соображениям. Они размещают свои центры обработки данных в районах с недорогой электроэнергией и многочисленными сетевыми коммутаторами. Они разрабатывают пользовательские серверные шасси, которые максимизируют энергоэффективность и минимизируют стоимость эксплуатации. Они используют специально созданную сетевую инфраструктуру со специальным аппаратным и программным обеспечением, точно настроенным на их внутренние сети. Они проводят интенсивную автоматизацию, чтобы обеспечить быстрое расширение и уменьшить вероятность человеческой ошибки.

Благодаря всем этим инженерным усилиям (не говоря уже об экономии за счет масштаба) стоимость распределенных вычислительных услуг для облачного провайдера намного ниже, чем для типичной компании с небольшим центром обработки данных. Экономия затрат отражается как на цене облачных услуг, так и на прибыли поставщиков.

На этой аппаратной основе созданы элементы управления, упрощающие и облегчающие настройку инфраструктуры. Облачные поставщики предлагают как интерфейсы прикладного программирования, так и инструменты, ориентированные на пользователя, которые контролируют предоставление и освобождение ресурсов. В результате весь цикл жизненного цикла системы или группы систем, распределенных в виртуальной сети, может быть автоматизирован. Эта концепция носит название "инфраструктура как код" и резко контрастирует с процедурами закупки и подготовки ручного сервера прошлых времен.

Гибкость — еще одна важная движущая сила внедрения облаков. Поскольку облачные системы могут предоставляться и освобождаться автоматически, любая компания, имеющая циклический спрос, может оптимизировать эксплуатационные расходы за счет добавления дополнительных ресурсов в периоды максимального использования и удаления дополнительной емкости, когда она больше не нужна. Встроенные функции автоматического масштабирования, доступные на некоторых облачных платформах, упрощают этот процесс.

Облачные провайдеры имеют глобальное присутствие на рынке. Приложив определенные усилия по планированию и проектированию, предприятия могут выйти на новые рынки, предложив услуги в нескольких географических регионах. Кроме того, выполнить аварийное восстановление в облаке проще, чем в частных центрах обработки данных, поскольку резервные системы могут физически находиться в разных местах.

🕮 Дополнительную информацию о методике DevOps см. в разделе 31.1.

Все эти характеристики хорошо сочетаются с методикой системного администрирования DevOps, делающей акцент на гибкости и повторяемости. В облаке вы больше не ограничены медленными процессами закупок или подготовки, и почти все можно автоматизировать.

Тем не менее от вас требуется определенный умственный скачок, потому что вы больше не контролируете свое собственное оборудование. Эту ситуацию хорошо описывает одна промышленная метафора: серверы следует рассматривать как скот, а не как домашних животных. Домашнему животному дают имя, его любят и лелеют. Когда домашнее животное болеет, его несут к ветеринару и лечат. Напротив, крупный рогатый скот — это товар, который пасется, продается и перемещается в больших количествах. Заболевший крупный рогатый скот пристреливают.

Облачный сервер — это всего лишь один из членов стада, иначе пришлось бы игнорировать базовый факт облачных вычислений: облачные системы являются эфемерными, и они могут выйти из строя в любое время. Планируйте эту ситуацию, и вы будете более успешны при работе с отказоустойчивой инфраструктурой.

Несмотря на все свои преимущества, облако не является панацеей для быстрого снижения затрат или повышения производительности. Непростой перенос существующего корпоративного приложения из центра обработки данных в облачный провайдер (по принципу "поднять и перенести") вряд ли будет успешным без тщательного планирования. Для облака нужны другие процессы эксплуатации, которые требуют обучения и тестирования. Кроме того, большинство корпоративных программ предназначено для статических сред, но отдельные системы в облаке следует рассматривать как недолговечные и ненадежные. Система называется облачной, если она надежна даже перед лицом непредвиденных событий.

9.2. Выбор облачной платформы

На выбор поставщика облачных услуг влияет множество факторов. Вероятно, определенную роль будут играть стоимость, прошлый опыт, совместимость с существующими технологиями, безопасность, требования соответствия и внутренняя политика. На процесс отбора также может повлиять репутация, размер провайдера, функции и, конечно же, маркетинг.

К счастью, существует много облачных провайдеров. Мы решили сосредоточиться только на трех основных провайдерах облачных услуг: Amazon Web Services (AWS), Google

Cloud Platform (GCP) и DigitalOcean (DO). В этом разделе мы упомянем несколько дополнительных вариантов. Основные игроки в этой области перечислены в табл. 9.2.

Таблица 9.2. Наиболее широко используемые облачные платформы

Провайдер	Отличительные черты
Amazon Web Services	Огромный размер. Быстрое внедрение инноваций. Может быть дорогой. Сложная
DigitalOcean	Простая и надежная. Имеет удобный интерфейс прикладного программирования. Хороша для разработки
Google Cloud Platform	Технически сложная и быстро совершенствующаяся. Делает акцент на производительность. Имеет широкий спектр услуг по передаче больших данных
IBM Softlayer	Больше похожа на хостинг, чем на облачную платформу. Имеет глобальную частную сеть
Microsoft Azure	Вторая по размеру, но далеко отстает от первой. Имеет историю отключений. Возможно, заслуживает внимания со стороны магазинов Microsoft
OpenStack	Модулярная DIY платформа с открытым исходным кодом для создания частных облаков. Имеет AWS-совместимые интерфейсы прикладного программирования
Rackspace	Открытые и частные облака, реализующие проекты OpenStack. Предлагает управляемые службы для AWS и Azure. Имеет фанатичных сторонников
VMware vCloud Air	Заумный сервис для публичных, частных и гибридных облаков. Использует технологию VMware. Вероятно, обречен

Публичные, частные и гибридные облака

В публичном облаке поставщик контролирует все физическое оборудование и предоставляет доступ к системам через Интернет. Это снимает с пользователей обязанность инсталлировать и обслуживать оборудование, но за счет меньшего контроля над функциями и характеристиками платформы. AWS, GCP и DO являются общедоступными облачными провайдерами.

Частные облачные платформы аналогичны, но размещаются в собственном центре обработки данных организации или управляются поставщиком от имени одного клиента. Серверы в частном облаке являются единственными арендаторами, а не делят его с другими клиентами, как в общедоступном облаке.

Частные облака обеспечивают гибкость и программный контроль, так же, как и обычные облака. Они привлекательны для организаций, которые уже инвестировали значительный капитал в оборудование, а также для инженеров, особенно тех, которые ценят полный контроль над своей средой.

OpenStack — это ведущая система с открытым исходным кодом, используемая для создания частных облаков. Она получает финансовую и техническую поддержку от таких предприятий, как AT & T, IBM и Intel. Rackspace сама по себе является одним из крупнейших участников OpenStack.

Комбинация публичных и частных облаков называется *гибридным облаком*. Гибриды могут быть полезными, когда предприятие сначала переходит с локальных серверов в публичное облако, для добавления временной емкости обработки пиковых нагрузок и реализации других сценариев. Администраторы должны быть осторожными: работа с двумя различными облачными платформами в тандеме непропорционально увеличивает сложность.

Облако VMware vSphere Air, основанное на технологии виртуализации vSphere, представляет собой удобное гибридное облако для клиентов, которые уже используют вирту-

ализацию VMware в своем локальном центре обработки данных. Эти пользователи могут прозрачно перемещать приложения в инфраструктуру vCloud Air и из нее.

Термин *публичное облако* является немного неудачным и вызывает опасения, связанные с безопасностью. На самом деле пользователи публичных облаков изолированы друг от друга несколькими слоями аппаратной и программной виртуализации. Частное облако практически не имеет преимущества в области безопасности над публичным облаком.

Кроме того, работа с частным облаком — это сложная и дорогостоящая перспектива, которую нельзя предпринимать необдуманно. Только крупнейшие и наиболее совершенные организации имеют инженерные ресурсы и финансовые средства, необходимые для внедрения надежного, безопасного частного облака. В то же время, после реализации функции частного облака обычно не соответствуют тем, которые предлагаются коммерческими облаками.

Для большинства организаций мы рекомендуем публичное облако над частными или гибридными опциями. Публичные облака предлагают наивысшую ценность и простоту администрирования. В оставшейся части этой книги мы ограничимся публичными облаками. В следующих нескольких разделах представлен краткий обзор каждой из приведенных выше платформ.

Amazon Web Services

AWS (Amazon Web Services) предлагает множество услуг: от виртуальных серверов (EC2) до управляемых баз данных, хранилищ данных (RDS и Redshift) и бессерверных функций, выполняемых в ответ на события (Lambda). Каждый год компания AWS выпускает сотни обновлений и новых функций. Она имеет самое большое и самое активное сообщество пользователей. AWS — безусловно, самая крупная компания в области облачных вычислений.

С точки зрения большинства пользователей, AWS имеет практически неограниченные возможности. Однако новые учетные записи имеют ограничения, которые определяют, какой объем вычислительной мощности вы можете запросить. Эти ограничения защищают как Amazon, так и вас, поскольку затраты могут быстро вырваться из-под контроля, если услуги не будут должным образом управляться. Чтобы увеличить свои лимиты, вы должны заполнить форму на сайте поддержки AWS. Ограничения, связанные с каждой службой, перечислены в документации по лимитам услуг.

Онлайновая документация AWS, расположенная по адресу aws.amazon.com/documentation, является авторитетной, всеобъемлющей и хорошо организованной. Это должно быть первое место, которое вы рассматриваете при исследовании конкретной услуги. Белые страницы, посвященные безопасности, пути миграции и архитектуры, неоценимы для тех, кто заинтересован в создании надежных облачных сред.

Google Cloud Platform

Если AWS является действующим чемпионом в области облачных вычислений, то компания Google является его потенциальным преемником. Он конкурирует за клиентов, используя такие нечестные трюки, как снижение цен и прямое обращение к болевым точкам AWS.

Спрос на инженеров настолько ожесточен, что Google, как известно, браковала бывших сотрудников AWS. В прошлом компания Google проводила вечеринки параллельно с конференцией AWS re:Invent в Лас-Вегасе, пытаясь привлечь как талантливых людей, так и пользователей. По мере эскалации облачных войн клиенты в конечном итоге выигрывают от этого конкурса в виде более низких затрат и улучшенных функций.

Google использует самую передовую глобальную сеть в мире, которая приносит пользу облачной платформе. Центры обработки данных Google — это технологические чудеса, которые предлагают множество инноваций для повышения энергоэффективности и снижения эксплуатационных затрат[1]. Компания Google относительно прозрачна в своих операциях, а ее вклад с открытым исходным кодом помогает развивать облачную индустрию.

Несмотря на свою техническую подкованность, по каким-то причинам компания Google является приверженцем публичного облака, а не лидером. Когда оно появилось в 2011-2012 гг.[2], GCP уже опаздывала на игру. Ее службы имеют много одинаковых функций (и часто одни и те же имена), эквивалентных функциям AWS. Если вы знакомы с AWS, то обнаружите, что веб-интерфейс GCP внешне несколько отличается от AWS, но их функциональные возможности поразительно похожи.

Мы ожидаем, что в последующие годы GCP получит определенную долю рынка, поскольку компания Google улучшает свою продукцию и укрепляет доверие клиентов. Она наняла некоторые из самых ярких умов в отрасли, которые должны разработать некоторые инновационные технологии. Как потребители, мы все выиграем от этого.

DigitalOcean

DigitalOcean — это другая разновидность публичного облака. В то время как AWS и GCP конкурируют за крупные предприятия и стартапы, ориентированные на рост, DigitalOcean привлекает маленьких клиентов с более простыми потребностями. Название этой стратегии — минимализм. Нам нравится применять DigitalOcean для экспериментов и проверок концепций.

DigitalOcean предлагает центры обработки данных в Северной Америке, Европе и Азии. В каждом из этих регионов есть несколько центров, но они не связаны напрямую и поэтому не могут считаться зонами доступности (см. раздел 9.3). В результате построить глобальные высокодоступные производственные службы на основе DigitalOcean значительно сложнее, чем на основе AWS или Google.

Серверы DigitalOcean называются *дроплетами* (droplet). Они просто управляются из командной строки или веб-консоли и быстро загружаются. DigitalOcean поставляет образы для всех наших демонстрационных операционных систем, кроме Red Hat. Он также имеет несколько образов для популярных приложений с открытым исходным кодом, таких как Cassandra, Drupal, Django и GitLab.

Платформа DigitalOcean также имеет функции балансировки нагрузки и хранения блоков. В главе 26 мы приводим пример предоставления балансировщика нагрузки DigitalOcean с двумя дроплетами с использованием инструмента обеспечения инфраструктуры Terraform компании HashiCorp.

9.3. Основы работы с облачными службами

Облачные службы слабо группируются по трем категориям.

- Инфраструктура как услуга (Infrastructure-as-s-Service — IaaS), в которой пользователи запрашивают исходные ресурсы вычислений, памяти, сети и хранилища.

[1]См. сайт `Google.com/about/datacenters`, на котором можно найти фотографии и факты о том, как работают центры обработки данных Google.

[2]Компания Google выпустила другие облачные продукты уже в 2008 г., включая App Engine, первый продукт платформы. Однако стратегия Google и бренд GCP не были очевидны до 2012 г.

Обычно они поставляются в виде виртуальных частных серверов, а также VPS. Пользователи IaaS отвечают за управление всем: операционными системами, сетями, системами хранения и собственным программным обеспечением.

- Платформа как услуга (Platform-as-s-Service — PaaS), в которой разработчики представляют свои специализированные приложения, упакованные в формате, указанном поставщиком. Затем поставщик запускает код от имени пользователя. В этой модели пользователи несут ответственность за свой собственный код, а поставщик управляет операционной системой и сетью.
- Программное обеспечение как услуга (Software-as-s-Service — SaaS) — самая широкая категория, в которой поставщик размещает программное обеспечение и управляет им, а пользователи платят абонентскую плату за доступ. Пользователи не поддерживают ни операционную систему, ни приложение. Почти любое размещенное веб-приложение (например, WordPress) попадает в категорию SaaS.

В табл. 9.3 показано, как каждая из этих абстрактных моделей разбивается на слои, связанные с полным развертыванием.

Таблица 9.3. На каких слоях вы отвечаете за управление?

Слой	Локальный[a]	IaaS	PaaS	SaaS
Приложение	✓	✓	✓	
Базы данных	✓	✓	✓	
Время выполнения приложения	✓	✓	✓	
Операционная система	✓	✓		
Виртуальная сеть, хранилище и серверы	✓	✓		
Платформа виртуализации	✓			
Физические серверы	✓			
Системы хранения	✓			
Физическая сеть	✓			
Мощность, пространство и охлаждение	✓			

[a] Локальный: локальные серверы и сеть

IaaS — инфраструктура как услуга (виртуальные серверы).

PaaS — платформа как услуга (например, Google App Engine).

SaaS — программное обеспечение как услуга (например, большинство веб-служб).

Из этих вариантов IaaS является наиболее подходящим для системного администрирования. В дополнение к определению виртуальных компьютеров, поставщики IaaS виртуализируют связанные с ними аппаратные элементы, такие как диски (в настоящее время используется более общий термин — "блочные устройства хранения данных") и сети. Виртуальные серверы могут посещать виртуальные сети, для которых вы указываете топологию, маршруты, адресацию и другие характеристики.

В большинстве случаев эти сети являются частными для вашей организации. IaaS также может включать в себя другие основные службы, такие как базы данных, очереди, хранилища ключей-значений и вычислительные кластеры. Эти функции в совокупности создают полную замену традиционного центра обработки данных (и во многих случаях даже улучшение).

PaaS — это область больших обещаний, которая еще не полностью реализована. Текущие предложения, такие как AWS Elastic Beanstalk, Google App Engine и Heroku, имеют экологические ограничения или нюансы, которые делают их непрактичными (или неполными) для использования в загруженных производственных средах. Снова и снова мы видели, как бизнес перерастает эти услуги. Однако новым услугам в этой области уделяется большое внимание. В ближайшие годы мы ожидаем значительных улучшений.

Облачные поставщики широко различаются по своим функциям и деталям реализации, но концептуально многие услуги очень похожи. В следующих разделах описываются облачные службы в целом, но поскольку AWS является лидером в этом пространстве, мы иногда принимаем его номенклатуру и условные обозначения в качестве значений по умолчанию.

Доступ к облаку

Первичный интерфейс большинства облачных провайдеров — это своего рода веб-интерфейс. Новые системные администраторы должны использовать эту веб-консоль для создания учетной записи и для настройки своих первых ресурсов.

Облачные провайдеры также определяют интерфейсы прикладного программирования, имеющие доступ к тем же базовым функциям, что и к веб-консоли. В большинстве случаев у них также есть стандартная оболочка командной строки, переносимая большинством систем для этих интерфейсов прикладного программирования.

Даже ветераны системного администрирования часто используют веб-графические интерфейсы. Тем не менее важно также дружить с инструментами командной строки, поскольку они легче справляются с автоматизацией и повторяемостью. Используйте сценарии, чтобы избежать утомительного и вялого процесса запроса всего и вся через браузер.

Облачные поставщики также поддерживают комплекты разработки программного обеспечения (SDK) для многих популярных языков, чтобы помочь разработчикам использовать их интерфейсы прикладного программирования. Сторонние инструменты используют SDK для упрощения или автоматизации определенных наборов задач. Вы, несомненно, столкнетесь с этими SDK, если напишете свои собственные инструменты.

Обычно для доступа к системам UNIX и Linux, работающим в облаке, используется алгоритм SSH с аутентификацией открытого ключа. Для получения дополнительной информации об эффективном использовании SSH см. раздел 27.7.

Некоторые поставщики облачных вычислений позволяют получить доступ к сеансу консоли через веб-браузер, что может быть особенно полезным, если вы ошибочно заблокируете себя с помощью правила брандмауэра или сломанной конфигурации SSH. Однако это не означает представление реальной консоли системы, поэтому вы не можете использовать эту функцию для отладки начальной загрузки или проблем с BIOS.

Регионы и зоны доступности

Облачные провайдеры поддерживают центры обработки данных по всему миру. Несколько стандартных терминов описывают особенности, связанные с географией.

Регион — это область, в которой облачный провайдер поддерживает центры обработки данных. В большинстве случаев регионы названы в честь территории предполагаемого обслуживания, хотя сами центры обработки данных более сконцентрированы.

Например, регион Amazon `us-east-1` обслуживается центрами обработки данных в северной Вирджинии.[3]

Некоторые провайдеры также имеют зоны доступности (или просто зоны), которые представляют собой совокупность центров обработки данных в регионе. Зоны внутри региона просматриваются в сетях с высокой пропускной способностью и низкой задержкой, поэтому межзональная связь выполняется быстро, хотя и не обязательно дешево. Например, мы наблюдали межзонную задержку длительностью менее 1 мс.

Зоны обычно проектируются так, чтобы быть независимыми друг от друга с точки зрения мощности и охлаждения, и они географически распределены, так что стихийное бедствие, которое затрагивает одну зону, имеет низкую вероятность воздействия на других людей в том же регионе.

Регионы и зоны имеют основополагающее значение для создания высокодоступных сетевых услуг. В зависимости от требований доступности вы можете развертывать свои ресурсы в нескольких зонах и регионах, чтобы минимизировать последствия сбоя в центре обработки данных или в географической области. Отключения зоны доступности возможны, но редки; региональные сбои происходят еще реже. Большинство служб, предоставляемых облачными провайдерами, знают о зонах и используют их для достижения встроенной избыточности.

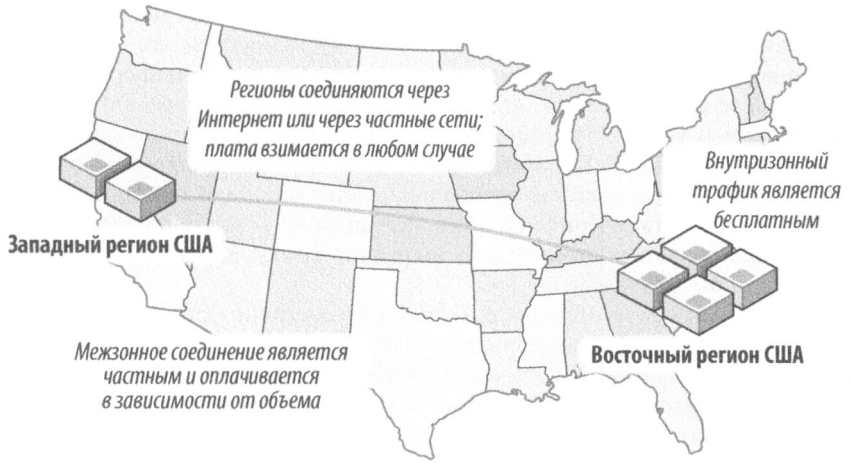


Рис. 9.1. Серверы, распределенные между несколькими регионами и зонами

Многоуровневые развертывания являются более сложными из-за физических расстояний между регионами и связанной с ними большей задержки. Некоторые поставщики облачных технологий используют более быструю и надежную межрегиональную сеть, чем другие. Если ваш сайт обслуживает пользователей по всему миру, качество сети вашего облачного поставщика имеет первостепенное значение.

Выбирайте регионы в зависимости от географической близости к вашей пользовательской базе. Для сценариев, в которых разработчики и пользователи находятся в разных географических регионах, подумайте над тем, чтобы ваши системы разработки были близки к разработчикам, а производственные системы ближе к пользователям.

[3]Для сигнала, передаваемого по оптическому волокну, требуется около 5 мс, чтобы преодолеть 1000 км, поэтому регионы размером с восточное побережье США с точки зрения производительности вполне приемлемы. Сетевое подключение, доступное для дата-центра, важнее, чем его точное местоположение.

Для сайтов, предоставляющих услуги для глобальной пользовательской базы, работа в нескольких регионах может существенно повысить производительность для конечных пользователей. Запросы могут направляться на региональные серверы каждого клиента, используя географическое DNS-разрешение, которое определяет местоположение клиентов по их исходным IP-адресам.

Большинство облачных платформ имеют регионы в Северной Америке, Южной Америке, Европе и странах Азиатско-Тихоокеанского региона. Только AWS и Azure непосредственно присутствуют в Китае. Некоторые платформы, в частности AWS и vCloud, имеют регионы, совместимые со строгими федеральными требованиями ITAR, принятыми в США.

Виртуальные частные серверы

Флагманской службой облака является виртуальный частный сервер, виртуальная машина, работающая на аппаратном обеспечении провайдера. Виртуальные частные серверы иногда называются *экземплярами*. Вы можете создать столько экземпляров, сколько вам нужно, запустив свою предпочтительную операционную систему и приложения, а затем закрыть экземпляры, когда они больше не нужны. Вы платите только за то, что используете, и обычно не несете авансовых расходов.

Поскольку экземпляры являются виртуальными машинами, их мощность процессора, память, размер диска и сетевые настройки могут настраиваться, когда экземпляр создается и даже корректируется постфактум. Открытые облачные платформы определяют предустановленные конфигурации, называемые *типами экземпляров*. Они варьируются от однопроцессорных узлов с 512 Мбайт памяти до больших систем со многими ядрами процессора и несколькими ТиБ памяти. Некоторые типы экземпляров сбалансированы для общего использования, а другие специализируются на приложениях с процессором, памятью, диском или сетью. Конфигурации экземпляров — это одна из областей, в которой поставщики облачных вычислений энергично конкурируют, чтобы соответствовать потребностям рынка.

Экземпляры создаются из образов, сохраненного состояния операционной системы, которое содержит (как минимум) корневую файловую систему и загрузчик. Образ может также включать в себя тома дисков для дополнительных файловых систем и других настраиваемых параметров. Вы можете легко создавать собственные образы с помощью собственного программного обеспечения и настроек.

Все наши иллюстративные операционные системы используются широко, поэтому облачные платформы обычно предоставляют для них официальные образы.[4] Многие сторонние поставщики программного обеспечения также поддерживают образы облаков, которые имеют предустановленное программное обеспечение для облегчения принятия клиентами.

Также легко создавать свои собственные образы. Подробнее о том, как создавать образы виртуальной машины в пакете, см. раздел 24.5.

Сети

Облачные провайдеры позволяют создавать виртуальные сети с настраиваемыми топологиями, которые изолируют ваши системы друг от друга и от Интернета. На платформах, которые предлагают эту функцию, вы можете установить диапазоны адресов ва-

[4] В настоящее время вы должны создать свой собственный образ FreeBSD, если используете Google Compute Engine.

ших сетей, определить подсети, настроить маршруты, установить правила брандмауэра и построить VPN для подключения к внешним сетям. С сетью и эксплуатацией более крупных и более сложных облачных приложений связаны определенные издержки.

🕮 Дополнительную информацию о сети VPN см. в разделе 27.9.

Вы можете сделать ваши серверы доступными в Интернете путем аренды публично маршрутизируемых адресов от вашего провайдера. У всех поставщиков есть большой пул таких адресов, из которых пользователи могут выбирать. В качестве альтернативы серверам может быть предоставлен только частный адрес RFC1918 в адресном пространстве, выбранном для вашей сети, что делает их общедоступными.

🕮 Дополнительную информацию о частных адресах RFC1918 см. в разделе 13.4.

Системы без общедоступных адресов напрямую не доступны из Интернета, даже для администраторов. Вы можете получить доступ к таким узлам через сервер перехода или хост-бастион, который открыт для Интернета, или через сеть VPN, которая подключается к вашей облачной сети. Для безопасности лучше, чтобы внешняя зона вашей виртуальной империи была как можно меньше.

Хотя все это звучит многообещающе, у вас меньше возможностей управлять виртуальными сетями, чем традиционными, и вы должны подчиняться прихотям и капризам набора функций, предоставленных вашим выбранным провайдером. Это особенно раздражает, когда новая функция не может взаимодействовать с вашей частной сетью. (Мы смотрим на тебя, Amazon!)

Для получения подробной информации о сети TCP/IP в облаке перейдите в раздел 13.15.

Хранилище

Важной частью облачных вычислений является хранение данных. Облачные провайдеры имеют самые большие и самые современные системы хранения данных на планете, поэтому вам будет трудно обеспечить их объем и возможности в частном центре обработки данных. Поставщики облачных вычислений взимают плату с объема данных, которые вы храните. Они очень мотивированы, чтобы дать вам как можно больше возможностей для хранения ваших данных.[5]

Перечислим несколько наиболее важных способов хранения данных в облаке.

- Объектные хранилища содержат коллекции дискретных объектов (по существу, файлов) в плоском пространстве имен. Объектные хранилища могут вмещать практически неограниченный объем данных с исключительно высокой надежностью, но относительно низкой производительностью. Они предназначены в основном для чтения. Файлы в хранилище объектов доступны через сеть с помощью протокола HTTPS. Примеры — AWS S3 и Google Cloud Storage.
- Блочные устройства хранения — это виртуализированные жесткие диски. Они могут быть настроены по вашему выбору и подключены к виртуальному серверу, подобно томам SAN в традиционной сети. Вы можете перемещать тома между узлами и настраивать свои профили ввода-вывода. Примеры — AWS EBS и Google Cloud Storage.

[5]Например, компания AWS предлагает посетить вашу организацию на машине AWS Snowmobile, представляющей собой транспортный контейнер длиной 45 футов, который буксируется грузовым тягачом и способен перевести 100 ПиБ из вашего дата-центра в облако.

- Эфемерное хранилище — это локальное дисковое пространство на виртуальном частном сервере (Virtual Private Server — VPS), созданное на основе дисков на главном сервере. Они обычно бывают быстрыми и емкими, но при удалении VPS данные теряются, поэтому эфемерное хранилище лучше всего использовать для временных файлов. Примеры — тома хранилища экземпляров на AWS и локальные SSD на GCP.

В дополнение к этим службам хранения данных облачные провайдеры обычно предлагают множество бесплатных служб баз данных, которые вы можете получить через сеть. Реляционные базы данных, такие как MySQL, PostgreSQL и Oracle, выполняются как службы AWS Relational Database Service. Они предлагают встроенную мультизонную избыточность и шифрование данных при хранении.

Распределенные аналитические базы данных, такие как AWS Redshift и GCP BigQuery, предлагают невероятную рентабельность инвестиций; обе эти базы заслуживают внимательного изучения, прежде чем строить собственное дорогостоящее хранилище. Поставщики облаков также предлагают обычный ассортимент баз данных в оперативной памяти и NoSQL, таких как Redis и **memcached**.

Идентификация и авторизация

Администраторам, разработчикам и другим техническим сотрудникам необходимо управлять облачными службами. В идеальном случае средства управления доступом должны соответствовать принципу наименьших привилегий: каждый директор может иметь доступ только к объектам, которые имеют к нему отношение, и не более того. В зависимости от контекста такие спецификации контроля доступа могут стать довольно сложными.

Компания AWS исключительно сильна в этой области. Ее служба под названием Identity and Access Management (IAM) определяет не только пользователей и группы, но и роли для систем. Например, серверу могут быть назначены политики, позволяющие его программному обеспечению запускать и останавливать другие серверы, хранить и извлекать данные в хранилище объектов или взаимодействовать с очередями — все с автоматической ротацией ключей. Служба IAM также имеет интерфейс прикладного программирования для управления ключами, который поможет вам безопасно хранить секреты.

Другие облачные платформы имеют меньше возможностей авторизации. Неудивительно, что служба Azure основана на службе Active Directory Microsoft. Она хорошо сочетается с сайтами, для которых существует интегрированный каталог. Служба контроля доступа Google, также называемая IAM, относительно грубая и неполная по сравнению со службой компании Amazon.

Автоматизация

Инструменты API и CLI, созданные поставщиками облачных технологий, являются основными строительными блоками пользовательской автоматизации, но они часто неуклюжи и непрактичны для организации больших коллекций ресурсов. Например, что делать, если вам нужно создать новую сеть, запустить несколько экземпляров VPS, предоставить базу данных, настроить брандмауэр и, наконец, подключить все эти компоненты? С точки зрения облачного API это был бы сложный сценарий.

AWS CloudFormation стала первой службой для решения этой проблемы. Она принимает шаблон в формате JSON или YAML, который описывает требуемые ресурсы и связанные

с ними детали конфигурации. Вы отправляете шаблон в службу CloudFormation, который проверяет его на наличие ошибок, сортирует зависимости между ресурсами и создает или обновляет конфигурацию облака в соответствии с вашими спецификациями.

Шаблоны CloudFormation являются мощными, но подвержены ошибкам в человеческих руках из-за строгих требований синтаксиса. Полный шаблон является невыносимо многословным, и людям сложно даже читать его. Вместо того, чтобы писать эти шаблоны вручную, мы предпочитаем автоматически генерировать их с помощью библиотеки Python под названием Troposphere от Марка Пика (Mark Peek) (см. `Github.com/cloudtools/troposphere`).

Третья сторонняя служба также нацелена на эту проблему. Служба Terraform с открытым исходным кодом компании HashiCorp, является средством для построения и изменения инфраструктуры независимо от особенностей облака.

Как и в службе CloudFormation, вы описываете ресурсы в настраиваемом шаблоне, а затем позволяете службе Terraform создавать правильные вызовы API для реализации вашей конфигурации.

Затем вы можете проверить свой конфигурационный файл на управление версиями и управлять инфраструктурой с течением времени.

9.4. Облака: быстрый запуск VPS на платформе

Облако — отличная песочница, в которой можно изучить системы UNIX и Linux. Этот короткий раздел поможет вам инсталлировать и запускать виртуальные серверы на AWS, GCP или DigitalOcean. В качестве системных администраторов мы активно используем командную строку (в отличие от веб-графических интерфейсов) для взаимодействия с облаком, поэтому иллюстрируем здесь использование именно этих инструментов.

Веб-службы Amazon

Для того чтобы использовать AWS, сначала настройте учетную запись на сайте `aws.amazon.com`. Создав учетную запись, немедленно следуйте инструкциям AWS Trusted Advisor, чтобы настроить свою учетную запись в соответствии с предлагаемыми рекомендациями. Затем вы можете перейти к отдельным консолям обслуживания для EC2, VPC и т.д.

Каждая служба AWS имеет специальный пользовательский интерфейс. Когда вы войдете в веб-консоль, вы увидите вверху список служб. Внутри Amazon каждая служба управляется независимой командой, и, к сожалению, данный интерфейс отражает этот факт. Хотя это решение помогло развивать AWS-службы, это приводит к несколько фрагментированному опыту взаимодействия. Некоторые интерфейсы более удобны и интуитивны, чем другие.

Для того чтобы защитить свою учетную запись, включите многофакторную аутентификацию (MFA) для пользователя `root`, а затем создайте привилегированного пользователя IAM для повседневного использования. Мы также обычно настраиваем псевдоним, чтобы пользователи могли получить доступ к веб-консоли без ввода номера учетной записи. Эта опция находится на начальной странице службы IAM.

В следующем разделе мы представляем официальный инструмент **`aws`** — интерфейс командной строки, написанный на Python. Новые пользователи также могут воспользоваться службой быстрого запуска Amazon Lightsail, целью которой является запуск экземпляра EC2 с минимальными усилиями.

Интерфейс aws: управление подсистемами AWS

Программа **aws** — это унифицированный интерфейс командной строки для служб AWS. Он управляет экземплярами, сохраняет резервные копии, редактирует записи DNS и выполняет большинство других задач, отображаемых в веб-консоли. Инструмент основан на замечательной библиотеке Boto, SDK Python для AWS API и работает в любой системе с действующим интерпретатором Python.

Установите этот инструмент с помощью команды **pip**:

```
$ pip install awscli
```

Для того чтобы использовать **aws**, сначала выполните аутентификацию в интерфейсе прикладного программирования AWS, используя пару случайных строк, называемых *идентификатором ключа доступа* и *секретным ключом доступа*. Создайте эти учетные данные в веб-консоли IAM, а затем скопируйте и вставьте их на местном уровне.

При выполнении команды **aws configure** вам будет предложено установить учетные данные API и регион по умолчанию:

```
$ aws configure
AWS Access Key ID: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [us-east-1]: <return>
Default output format [None]: <return>
```

Эти настройки сохраняются в файле **~/.aws/config**. Пока вы настраиваете среду, мы также рекомендуем вам настроить функцию автозаполнения оболочки **bash**, чтобы легче было обнаружить подкоманды. Дополнительная информация представлена в документах AWS CLI.

Первый аргумент команды **aws** называет конкретную службу, которой вы хотите манипулировать; например, **ec2** для действий, которые управляют веб-службой Elastic Compute Cloud. Вы можете добавить подсказку по ключевым словам в конце любой команды, чтобы увидеть инструкции. Так, **aws help**, **aws ec2 help** и **aws ec2 describe-instance help** помогают создавать полезные справочные страницы.

Создание экземпляра EC2

🕮 Дополнительную информацию о команде **pip** см. в разделе 7.6.

Для создания и запуска экземпляров EC2 используйте команду **aws ec2 run-instances**. Хотя вы можете создавать несколько экземпляров с помощью одной команды (используя параметр **--count**), экземпляры должны иметь общую конфигурацию. Вот минимальный пример полной команды:

```
$ aws ec2 run-instances --image-id ami-d440a6e7
    --instance-type t2.nano --associate-public-ip-address
    --key-name admin-key
# вывод см. ниже
```

В этом примере указаны следующие данные конфигурации.

- Образ базовой системы представляет собой версию CentOS 7, выпущенную Amazon, с именем `ami-d440a6e7`. (AWS называет эти образы AMI — Amazon Machine Images.)
- Как и другие объекты AWS, имена образов, к сожалению, не мнемонические; вы должны искать идентификаторы в веб-консоли EC2 или в командной строке (**aws ec2 describe-images**) для их декодирования.

- Тип экземпляра — `t2.nano`, который в настоящее время является наименьшим типом экземпляра. Он имеет одно ядро центрального процессора и 512 Мбайт оперативной памяти. Сведения о доступных типах экземпляров можно найти в веб-консоли EC2.
- Для управления доступом SSH также назначается предварительно сконфигурированная пара ключей. Вы можете создать пару ключей с помощью команды **ssh-keygen** (см. раздел 27.7), а затем загрузить открытый ключ в консоль AWS EC2.

Результат работы команды **aws ec2 run-instance** показан ниже. Это формат JSON, поэтому он легко распознается другим программным обеспечением. Например, после запуска экземпляра сценарий может извлекать IP-адрес экземпляра и настраивать DNS, обновлять систему инвентаризации или координировать запуск нескольких серверов.

```
$ aws ec2 run-instances ... # Те же команды, что и выше
{
    "OwnerId": "188238000000",
    "ReservationId": "r-83a02346",
    "Instances": [
        ...
        "PrivateIpAddress": "10.0.0.27",
        "InstanceId": "i-c4f60303",
        "ImageId": "ami-d440a6e7",
        "PrivateDn   sName": "ip-10-0-0-27.us-west-2.compute.internal",
        "KeyName":  "admin-key",
        "SecurityGroups": [
            {
                "GroupName": "default",
                "GroupId": "sg-9eb477fb"
            }
        ],
        "SubnetId": "subnet-ef67938a",
        "InstanceType": "t2.nano",
    ...
}
```

По умолчанию экземпляры EC2 в подсетях VPC не имеют подключенных общедоступных IP-адресов, что делает их доступными только из других систем в пределах одного VPC. Чтобы использовать экземпляры непосредственно из Интернета, задействуйте параметр **--associate-public-ip-address**, как показано в нашем примере. Вы можете узнать назначенный IP-адрес постфактум с помощью команды **aws ec2 describe-instances** или путем поиска экземпляра в веб-консоли.

Брандмауэры в EC2 называются группами безопасности. Поскольку мы не указали здесь группу безопасности, AWS принимает группу `default`, которая не дает доступа. Чтобы подключиться к экземпляру, настройте группу безопасности, чтобы разрешить выполнение алгоритма SSH с вашего IP-адреса. В сценариях реального мира структура группы безопасности должна быть тщательно спланирована во время проектирования сети. Мы обсуждаем группы безопасности в разделе 13.15.

Команда **aws configure** устанавливает область по умолчанию, поэтому вам не нужно указывать регион для экземпляра, если вы не хотите задать что-то, отличное от значения, принятого по умолчанию. AMI, пара ключей и подсеть относятся к региону, и интерфейс **aws** жалуется, если их нет в указанном вами регионе. (В этом конкретном случае AMI, пара ключей и подсеть находятся в регионе `us-east-1`.)

Обратите внимание на поле `InstanceId` в списке результатов, которое является уникальным идентификатором для нового экземпляра. Команда **`aws ec2 describe-instance -instance-id id`** выводит подробную информацию о существующем экземпляре, а команда **`aws ec2 describe-instances`** — обо всех экземплярах в регионе.

Когда экземпляр запущен и группа безопасности по умолчанию настроена для передачи трафика по TCP-порту 22, вы можете использовать алгоритм SSH для входа в систему. Большинство AMI настроены с учетной записью `nonroot` с привилегиями **sudo**. Для системы Ubuntu имя пользователя — `ubuntu`; для CentOS — `centos`. FreeBSD и Amazon Linux используют имя `ec2-user`. В документации для выбранного вами AMI должно быть указано имя пользователя, если оно не является одним из них.

Правильно настроенные образы позволяют использовать только открытые ключи для аутентификации SSH, а не пароли. После того как вы вошли в систему с секретным ключом SSH, у вас будет полный доступ к **sudo** без необходимости пароля. Мы рекомендуем отключить пользователя по умолчанию после первой загрузки и создать личные учетные записи.

Просмотр журнала консоли

Отладка низкоуровневых проблем, таких как проблемы при запуске и ошибки диска, может быть сложной задачей без доступа к консоли экземпляра. Интерфейс EC2 позволяет получить вывод консоли экземпляра, который может быть полезен, если экземпляр находится в состоянии ошибки или, по-видимому, зависает. Вы можете сделать это через веб-интерфейс или с помощью команды **`aws ec2 get-console-output`**, как показано ниже.

```
$ aws ec2 get-console-output --instance-id i-c4f60303
{
    "InstanceId": "i-c4f60303",
    "Timestamp": "2015-12-21T00:01:45.000Z",
    "Output": "[ 0.000000] Initializing cgroup subsys cpuset\r\n[
        0.000000] Initializing cgroup subsys cpu\r\n[ 0.000000]
        Initializing cgroup subsys cpuacct\r\n[ 0.000000] Linux version
        4.1.7-15.23.amzn1.x86_64 (mockbuild@gobi-build-60006)
        (gcc version 4.8.3 20140911 (Red Hat 4.8.3-9)) #1 SMP Mon Sep
        14 23:20:33 UTC 2015\r\n
    ...
}
```

🕮 Дополнительную информацию об управлении пользователями см. в главе 8.

Полный журнал, конечно, намного дольше, чем этот фрагмент. В дампе JSON содержимое журнала, к сожалению, склеивается в одну строку. Для лучшей читаемости отформатируйте его с помощью команды **sed**.

```
$ aws ec2 get-console-output --instance-id i-c4f60303 | sed
    's/\\r\\n/\\n/g'
{
    "InstanceId": "i-c4f60303",
    "Timestamp": "2015-12-21T00:01:45.000Z",
    "Output": "[ 0.000000] Initializing cgroup subsys cpuset
        [ 0.000000] Initializing cgroup subsys cpu
        [ 0.000000] Initializing cgroup subsys cpuacct
        [ 0.000000] Linux version 4.1.7-15.23.amzn1.x86_64
            (mockbuild@gobi-build-60006) (gcc version 4.8.3 20140911
```

```
            (Red Hat 4.8.3-9)) #1 SMP Mon Sep 14 23:20:33 UTC 2015
    ...
}
```

Эта запись журнала поступает непосредственно из процесса загрузки Linux. В приведенном выше примере показано несколько строк с момента инициализации экземпляра. В большинстве случаев вы найдете самую интересную информацию в конце журнала.

Остановка и завершение работы экземпляров

Когда вы закончите работать с экземпляром, вы можете остановить (stop) его, чтобы закрыть экземпляр, но сохранить его для последующего использования, или завершить (terminate) его, чтобы полностью удалить экземпляр. По умолчанию завершение также полностью освобождает корневой диск экземпляра. После прекращения работы экземпляр не может быть восстановлен даже с помощью AWS.

```
$ aws ec2 stop-instances --instance-id i-c4f60303
{
    "StoppingInstances": [
        {
             "InstanceId": "i-c4f60303",
             "CurrentState": {
                 "Code": 64,
                 "Name": "stopping"
             },
             "PreviousState": {
                  "Code": 16,
                  "Name": "running"
             }
        }
    ]
}
```

Обратите внимание на то, что виртуальные машины не меняют состояние мгновенно; им требуется перезагрузка. Следовательно, существуют переходные состояния, такие как начало (starting) и остановка (stopping). Обязательно учитывайте их в любых сценариях, которые вы можете написать.

Google Cloud Platform

Для того чтобы начать работу с платформой GCP, создайте учетную запись на сайте `cloud.google.com`. Если у вас уже есть идентификатор Google, вы можете зарегистрироваться в той же учетной записи.

Службы GCP работают в разделе, известном как *проект*. В каждом проекте есть отдельные пользователи, данные о счетах и учетные данные API, поэтому вы можете достичь полного разделения между разрозненными приложениями или областями бизнеса. Создав свою учетную запись, создайте проект и включите отдельные службы GCP в соответствии с вашими потребностями. Google Compute Engine, служба VPS, является одной из первых служб, которые вы, возможно, захотите включить.

Настройка `gcloud`

Программа `gcloud`, приложение на языке Python, является инструментом CLI для GCP. Это компонент Google Cloud SDK, который содержит множество библиотек и инструментов для взаимодействия с GCP. Чтобы установить его, следуйте инструкциям по установке на сайте `cloud.google.com/sdk`.

Первое действие должно состоять в том, чтобы настроить среду, выполнив команду **gcloud init**. Эта команда запускает небольшой локальный веб-сервер, а затем открывает ссылку браузера для отображения пользовательского интерфейса Google для аутентификации. После аутентификации через веб-браузер **gcloud** попросит вас (в оболочке) выбрать профиль проекта, зону по умолчанию и другие значения по умолчанию. Настройки сохраняются в файле **~/.config/gcloud/**.

Выполните команду **gcloud help** для получения общей информации или **gcloud -h** для краткого описания использования. Также доступна помощь по подкоманде; например, команда **gcloud help compute** показывает справочную страницу для службы Compute Engine.

Запуск экземпляра на GCE

В отличие от команд **aws**, которые немедленно возвращают управление, команда **gcloud compute** работает синхронно. Например, когда вы выполняете команду **create** для запроса нового экземпляра, команда **gcloud** делает необходимый вызов API, а затем ждет, пока экземпляр не будет настроен и запущен, а затем возвращает его. Это соглашение позволяет избежать необходимости опроса состояния экземпляра после его создания.[6]

Для того чтобы создать экземпляр, сначала определите имя или псевдоним образа, которое вы хотите загрузить:

```
$ gcloud compute images list --regexp 'debian.*'
NAME                         PROJECT        ALIAS     DEPRECATED STATUS
debian-7-wheezy-v20160119    debian-cloud   debian-7              READY
debian-8-jessie-v20160119    debian-cloud   debian-8              READY
```

Затем создайте и загрузите экземпляр, указав его имя и образ.

```
$ gcloud compute instances create ulsah --image debian-8
# ожидает экземпляра для запуска...
NAME   ZONE           MACHINE_TYPE   INTERNAL_IP  EXTERNAL_IP     STATUS
ulsah  us-central1-f  n1-standard-1  10.100.0.4   104.197.65.218  RUNNING
```

Обычно результаты работы этой команды содержит столбец, который показывает, является ли экземпляр вытесняемым, но в этом случае он был пустым, и мы удалили его, чтобы уместить вывод на странице. Вытесняемые экземпляры менее дороги, чем стандартные экземпляры, но они могут работать только 24 часа и их работа может быть прекращена в любое время, если Google нуждается в ресурсах для другой цели. Они предназначены для долговременных операций, которые могут допускать перерывы, например задания пакетной обработки.

Вытесняемые экземпляры аналогичны спот-экземплярам EC2 в том смысле, что вы платите дисконтированную ставку за остальную резервную емкость. Тем не менее мы обнаружили, что вытесняемые экземпляры Google более разумны и проще в управлении, чем спот-экземпляры AWS. Однако долговременные стандартные экземпляры остаются наиболее подходящим выбором для большинства задач.

Программа **gcloud** инициализирует экземпляр публичным и частным IP-адресом. Вы можете использовать публичный IP-адрес с алгоритмом SSH, но программа **gcloud** имеет полезную оболочку для упрощения входа в систему SSH:

```
$ gcloud compute ssh ulsah
Last login: Mon Jan 25 03:33:48 2016
ulsah:~$
```

[6] Выполните команду **aws ec2 wait** для получения информации об опросе событий или состояний в AWS EC2.

DigitalOcean

С объявленным временем загрузки 55 с виртуальные серверы DigitalOcean (дроплеты) — это самый быстрый путь к корневой оболочке. Стоимость начального уровня составляет 5 долл. США в месяц, поэтому они также не разорят вас.

🕮 Дополнительную информацию о настройке "драгоценных камней" Ruby см. в разделе 7.6.

После создания учетной записи вы можете управлять своими дроплетами через вебсайт DigitalOcean. Тем не менее нам удобнее применять буксир, инструмент командной строки, написанный на языке Ruby, который использует опубликованный API DigitalOcean. Предположим, что у вас есть Ruby и его менеджер библиотеки, **gem**, установленный в локальной системе. Для инсталляции программы **tugboat** просто выполните команду **gem install tugboat**.

Необходимо выполнить несколько этапов настройки. Сначала создайте пару криптографических ключей, которые вы можете использовать для контроля доступа к дроплетам.

```
$ ssh-keygen -t rsa -b 2048 -f ~/.ssh/id_rsa_do
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase): <return>
Enter same passphrase again: <return>
Your identification has been saved in /Users/ben/.ssh/id_rsa_do.
Your public key has been saved in /Users/ben/.ssh/id_rsa_do.pub.
```

Скопируйте содержимое файла открытого ключа и вставьте его в веб-консоль DigitalOcean (в настоящее время в разделе Setting⇨Security). В рамках этого процесса назначьте короткое имя открытому ключу.

Затем подключите программу **tugboat** API DigitalOcean, введя токен доступа, который вы получите с веб-сайта. Программа **tugboat** сохраняет токен для будущего использования в файле **~/.tugboat**.

🕮 Дополнительную информацию об алгоритме SSH см. в разделе 22.7.

```
$ tugboat authorize
Note: You can get your Access Token from https://cloud.digitalocean.com/
settings/tokens/new
Enter your access token: e9dff1a9a7ffdd8faf3…f37b015b3d459c2795b64
Enter your SSH key path (defaults to ~/.ssh/id_rsa): ~/.ssh/id_rsa_do
Enter your SSH user (optional, defaults to root):
Enter your SSH port number (optional, defaults to 22):
Enter your default region (optional, defaults to nyc1): sfo1
...
Authentication with DigitalOcean was successful.
```

Чтобы создать и запустить дроплеты, сначала укажите имя образа системы, который вы хотите использовать в качестве базовой. Например, так.

```
$ tugboat images | grep -i ubuntu
16.04.1 x64 (slug: , id: 21669205, distro: Ubuntu)
16.04.1 x64 (slug: , id: 22601368, distro: Ubuntu)
16.04.2 x64 (slug: ubuntu-16-04-x64, id: 23754420, distro: Ubuntu)
16.04.2 x32 (slug: ubuntu-16-04-x32, id: 23754441, distro: Ubuntu)
...
```

Вам также нужен цифровой идентификатор DigitalOcean для ключа SSH, вставленного в веб-консоль.

```
$ tugboat keys
SSH Keys:
Name: id_rsa_do, (id: 1587367), fingerprint:
    bc:32:3f:4d:7d:b0:34:ac:2e:3f:01:f1:e1:ea:2e:da
```

Этот вывод показывает, что числовой идентификатор для ключа с именем `id_rsa_do` равен 1587367. Создайте и запустите дроплеты следующим образом:

```
$ tugboat create -i ubuntu-16-04-x64 -k 1587367 ulsah-ubuntu
queueing creation of droplet 'ulsah-ubuntu'...Droplet created!
```

Здесь аргумент **-k** — это идентификатор ключа SSH, а последний аргумент — короткое имя для дроплета, которое вы можете назначить по своему усмотрению.

Как только дроплет загрузится, вы можете войти в систему с помощью команды **tugboat ssh**.

```
$ tugboat ssh ulsah-ubuntu
Droplet fuzzy name provided. Finding droplet ID...done, 23754420
     (ubuntu-16-04-x64)
Executing SSH on Droplet (ubuntu-16-04-x64)...
This droplet has a private IP, checking if you asked to use the Private IP...
You didn't! Using public IP for ssh...
Attempting SSH: root@45.55.1.165
Welcome to Ubuntu 16.04 ((GNU/Linux 4.4.0-28-generic x86_64)
root@ulsah-ubuntu:~#
```

Вы можете создать столько дроплетов, сколько вам нужно, но имейте в виду, что вам будет выставлен счет за каждый из них, даже если он выключен. Чтобы отключить дроплет, выполните команду **tugboat snapshot** *имя-дроплета имя-снимка*, чтобы сделать снимок памяти системы, и команду **tugboat destroy** *имя-дроплета*, чтобы отключить дроплет. Позднее вы можете воссоздать дроплет, используя снимок в качестве исходного образа.

9.5. Контроль затрат

Новички в облачных вычислениях часто наивно предполагают, что крупномасштабные системы будут значительно дешевле работать в облаке, чем в дата-центре. Это ожидание может быть связано с превратной интерпретацией низкой стоимости часа работы с экземпляром на облачной платформе. Кроме того, возможно, пользователи поддаются уговорам облачных маркетологов, чьи тематические исследования всегда показывают огромную экономию.

Независимо от их источника, мы обязаны отбросить надежды и оптимизм. По нашему опыту, новые пользователи облачных вычислений часто удивляются, когда затраты быстро растут.

Тарифы облака обычно состоят из нескольких компонентов.

- Вычислительные ресурсы виртуальных частных серверов, балансировщиков нагрузки и всего остального, которые потребляют циклы процессора для запуска ваших служб. Цена взимается за час использования.
- Передача данных в Интернете (как входящая, так и исходящая), а также трафик между зонами и регионами. Цены взимаются за двоичные гига- или терабайты.
- Хранилища всех типов: блочные тома хранения, объектные хранилища, моментальные снимки диска, а в некоторых случаях и операции ввода-вывода в различных постоянных хранилищах. Взимается цена за каждый двоичный гига- или терабайт в месяц.

Для вычислительных ресурсов наиболее дорогостоящей является модель "плати или иди", также известная как "ценообразование по требованию". У AWS и DigitalOcean минимальный интервал биллинга составляет один час, а на GCP — минута. Цены варьируются от долей процента в час (самый маленький тип дроплета DigitalOcean с 512 Мбайт и одним ядром процессора или экземплярами AWS `t2.nano`) до нескольких долларов в час (экземпляр `i2.8xlarge` в AWS с 32 ядрами, ОЗУ 104 ГиБ и 8 × 800 Гб локальных SSD).

Вы можете добиться существенной экономии на виртуальных серверах, заплатив авансом за более длительные сроки. В компании AWS это называется "ценой зарезервированных экземпляров". К сожалению, чтобы точно определить, что покупать, необходимо выполнить очень тяжелую и длительную работу.

Зарезервированные экземпляры EC2 привязаны к определенному семейству экземпляров. Если позже вы решите, что вам нужно что-то другое, ваши инвестиции будут потеряны. С другой стороны, если вы зарезервируете экземпляр, вам гарантируется, что он будет доступен для вашего использования. С экземплярами по запросу желаемый тип может быть недоступен даже при его установке, в зависимости от текущей емкости и спроса. AWS продолжает корректировать свою структуру ценообразования, поэтому, к счастью, нынешняя система может быть упрощена в будущем.

Для рабочих нагрузок, которые допускают перерывы, AWS предлагает точечное ценообразование. Спот-рынок — это аукцион. Если ваша ставка превышает текущую спот-цену, вам будет предоставлен тип запрашиваемого вами экземпляра, пока цена не превысит максимальную ставку, после чего работа вашего экземпляра будет прервана. Цены могут быть значительно снижены по сравнению с EC2 по требованию и зарезервированным ценам, но варианты использования ограничены.

Сравнить цены на Google Compute Engine достаточно просто. При длительном использовании автоматически применяются скидки, и вы никогда не будете платить авансом. Вы платите полную базовую цену за первую неделю месяца, а инкрементная цена падает каждую неделю на 20% от базовой ставки до максимальной скидки 60%. Чистая скидка на полный месяц использования составляет 30%. Это примерно сопоставимо с дисконтом на один год зарезервированного экземпляра EC2, но вы можете изменять экземпляры в любое время.[7]

Еще более сложно прогнозировать сетевой трафик. Факторы, которые, как правило, вызывают высокие затраты на передачу данных, включают в себя следующее.

- Веб-сайты, которые загружают и обслуживают большие мультимедийные файлы (видео, образы, PDF-файлы и другие крупные документы) непосредственно из облака, а не выгружают их в CDN.
- Межзонный или межрегиональный трафик для кластеров баз данных, которые реплицируются для отказоустойчивости; например, программное обеспечение, такое как Cassandra, MongoDB и Riak.
- MapReduce или кластер хранилищ данных, которые охватывают несколько зон.
- Образы дисков и снимки томов, передаваемые между зонами или регионами для резервного копирования (или другим автоматическим процессом).

В ситуациях, когда репликация между несколькими зонами важна для обеспечения доступности, вы сэкономите на трансферных расходах, ограничив кластеры до двух зон,

[7]Совет для бережливых людей: поскольку схема скидок связана с вашим биллинговым циклом, время переходов имеет значение. Вы можете переключать типы экземпляров в начале или в конце цикла без штрафа. Наихудший случай заключается в том, чтобы переключаться на полпути биллингового цикла. Это переключение штрафуется примерно на 20% от месячной базовой ставки экземпляра.

а не используя три или более. Некоторые программы предлагают такие настройки, как сжатие, которое может уменьшить количество реплицированных данных.

Существенным источником расходов на AWS является количество операций ввода-вывода (IOPS) для томов EBS (External BLOB Storage). Цены за использование EBS взимаются за количество двоичных гигабайтов в месяц и за количество операций IOPS в месяц. Цена за 200 Гбайт EBS объемом 5000 IOS составляет несколько сотен долларов в месяц. Их кластер может разорить компанию.

Лучшая защита от высоких счетов — это измерение, мониторинг и трезвый расчет. Используйте функции автомасштабирования для удаления емкости, когда это не требуется, снижая затраты в периоды низкого спроса. Используйте более мелкие экземпляры для более детального управления. Следите за шаблонами использования, прежде чем тратить пакет на зарезервированные экземпляры или объемы с высокой пропускной способностью. Облако является гибким, и вы можете вносить изменения в свою инфраструктуру по мере необходимости.

🕮 Дополнительную информацию об CDN см. в разделе 19.2.

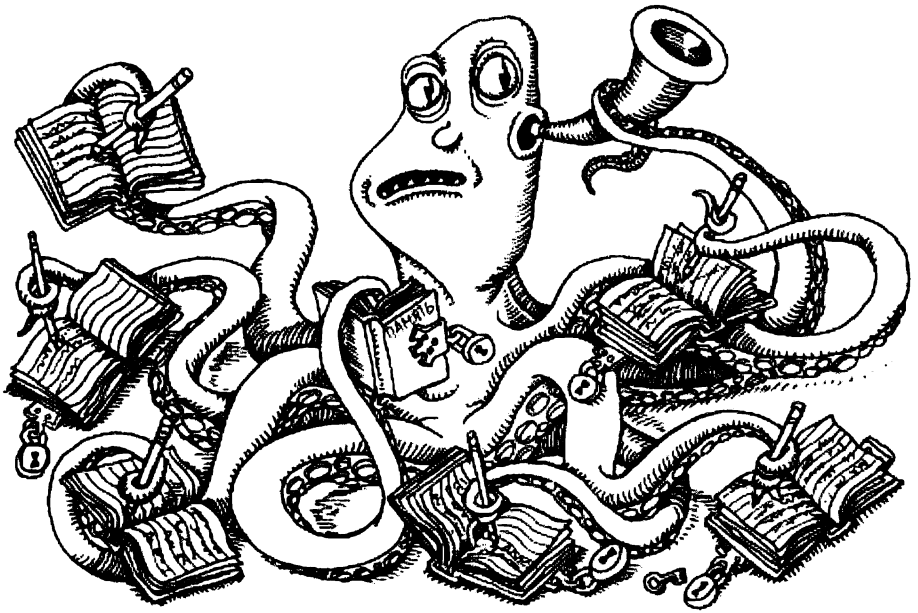
Окружающая среда ЦС растет, выявление того, где деньги расходуются, может стать проблемой. Большим облачным учетным записям могут быть полезны сторонние службы, которые анализируют использование и предлагают функции отслеживания и отчетности. Мы использовали Cloudability и CloudHealth. Оба подключаются к функциям выставления счетов AWS для разбивки отчетов по пользовательскому тегу, сервису или географическому местоположению.

9.6. Литература

- Wittig, Andreas, and Michael Wittig. *Amazon Web Services In Action*. Manning Publications, 2015.
- Google. `cloudplatform.googleblog.com`. Официальный блог Google Cloud Platform.
- Barr, Jeff, and others at Amazon Web Services. `aws.amazon.com/blogs/aws`. Официальный блог Amazon Web Services.
- DigitalOcean. `digitalocean.com/company/blog`. Технический и производственный блог DigitalOcean.
- Vogels, Werner. *All Things Distributed*. `allthingsdistributed.com`. Блог Вернера Фогельса (Werner Vogels), технического директора компании Amazon.
- Wardley, Simon. *Bits or pieces?* `blog.gardeviance.org`. Блог исследователя и законодателя мод в области облачных технологий Саймона Уордли (Simon Wardley), содержащий анализ тенденций в области облачных технологий, а иногда и резкую критику.
- Bias, Randy. `cloudscaling.com/blog`. Рэнди Байес — директор компании OpenStack, имеющий инсайдерскую информацию об индустрии частных облаков и его будущем.
- Cantrill, Bryan. *The Observation Deck*. `dtrace.org/blogs/bmc`. Интересные точки зрения и технические идеи о вычислениях технического директора компании Joyent, разработавшей специальную, но очень интересную платформу.
- Amazon. `youtube.com/AmazonWebServices`. Выступления на конференциях и другие видеоматериалы от компании AWS.

Глава 10

Журналирование



Системные демоны, ядро, утилиты и приложения — все они генерируют журнальные данные, которые регистрируются и попадают на далеко не безразмерные диски. Срок полезной службы большинства данных ограничен, поэтому их приходится группировать, сжимать, архивировать и, наконец, удалять. Доступом и журналами аудита необходимо управлять точно в соответствии с регулятивными правилами хранения информации или политикой безопасности, принятой для вашего хоста.

В большинстве случаев регистрируемое событие перехватывается в виде одной строки текста, которая включает временную метку, тип и степень серьезности события, а также имя и идентификатор процесса. Системные администраторы несут ответственность за извлечение полезной информации из этого потока сообщений.

Данная задача называется *управлением журналами*, и ее можно разделить на несколько основных подзадач.

- Сбор журналов из различных источников.
- Предоставление структурированного интерфейса для запросов, анализа, фильтрации и мониторинга сообщений.
- Управление хранением и истечением срока действия сообщений, чтобы информация сохранялась до тех пор, пока она потенциально полезна или юридически необходима, но не на неопределенный срок.

Система UNIX исторически управляет журналами через интегрированную, но несколько рудиментарную систему, известную как **syslog**, которая представляет приложения со стандартизованным интерфейсом для отправки сообщений журнала. Демон **syslog** сортирует сообщения и сохраняет их в файлы или пересылает их другому хосту по сети. К сожалению, **syslog** решает только первую из перечисленных выше задач регистрации (сбор сообщений), а ее конфигурация сильно варьируется среди операционных систем.

Возможно, из-за недостатков демона **syslog** многие приложения, сетевые демоны, сценарии запуска и другие средства ведения журнала полностью обходят **syslog** и записывают сообщения в свои собственные файлы. Это привело к тому, что в разных версиях UNIX и даже среди дистрибутивов Linux журналы значительно отличаются друг от друга.



Журнал **systemd** от Linux представляет собой вторую попытку привнести здравомыслие в этот беспорядок. Журнал собирает сообщения, сохраняет их в индексированном и сжатом двоичном формате и предоставляет интерфейс командной строки для просмотра и фильтрации журналов. Журнал может существовать в одиночестве или сосуществовать с демоном **syslog** с разной степенью интеграции в зависимости от конфигурации.

Разнообразные сторонние инструменты (как запатентованные, так и с открытым исходным кодом) затрагивают более сложную проблему отбора сообщений, которые поступают из большой сети систем. Эти инструменты включают такие вспомогательные средства, как графические интерфейсы, языки запросов, визуализация данных, оповещение и автоматическое обнаружение аномалий. Они могут масштабироваться для обработки томов сообщений порядка терабайт в день. Вы можете подписаться на эти продукты в виде облачной службы или разместить их самостоятельно в частной сети.

На рис. 10.1 изображена архитектура сайта, на котором используются все службы управления журналами, упомянутые выше. Администраторы и другие заинтересованные стороны могут запускать графический интерфейс централизованного журнального кластера для просмотра сообщений журнала, поступающих из систем по всей сети. Администраторы могут также регистрироваться на отдельных хостах и получать доступ к сообщениям через журнал **systemd** или файлы обычного текста, записанные демоном **syslog**. Если эта диаграмма вызывает у вас больше вопросов, чем ответов, вы читаете правильную главу.

При отладке проблем и устранении ошибок опытные администраторы в первую очередь обращаются к журналам. Файлы журналов часто содержат важные подсказки, указывающие на источник неприятных ошибок конфигурации, ошибок программного обеспечения и проблем безопасности. Журналы — это первое место, которое вы должны посмотреть, если демон дал сбой или произошел отказ от его запуска либо возникла хроническая ошибка при загрузке системы.

После принятия официальных стандартов, таких как PCI DSS, COBIT и ISO 27001, и достижения зрелости регуляторных норм в отдельных отраслях важность четко определенной журнальной стратегии возросла. В настоящее время эти внешние стандарты могут потребовать, чтобы для ведения журнала вы поддерживали централизованный репозитарий предприятия с временными метками, подтвержденными протоколом NTP (Network Time Protocol) и строго определенным графиком хранения.[1] Однако даже организации, не имеющие нормативных требований или требований соответствия, могут извлечь выгоду из централизованного ведения журналов.

[1]Конечно, точное системное время имеет значение даже без наличия регуляторных норм. Мы настоятельно рекомендуем включить протокол NTP для всех ваших систем.

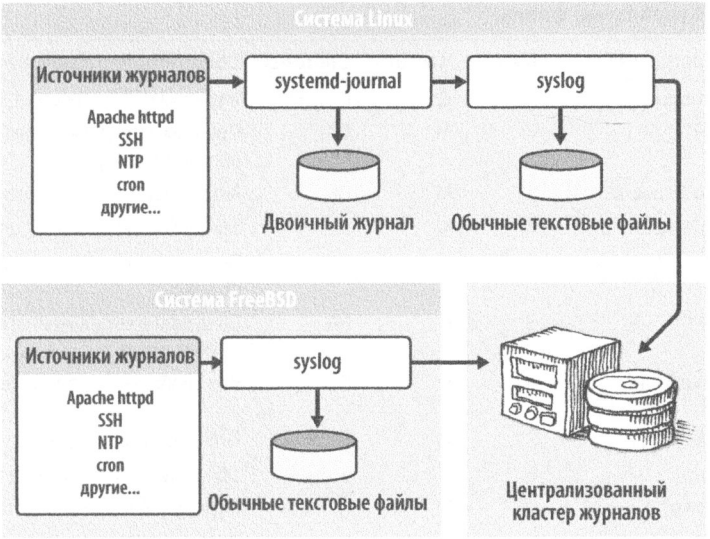


Рис. 10.1. Архитектура централизованной журнальной системы

В этой главе описывается собственное программное обеспечение для управления журналами, используемое в системах Linux и FreeBSD, включая **syslog**, журнал **systemd** и приложение **logrotate**. Мы также вводим некоторые дополнительные инструменты для централизации и анализа журналов по всей сети. Глава закрывается некоторыми общими советами для создания разумной политики управления журналом.

10.1. Местоположение файлов регистрации

Систему UNIX часто критикуют за несогласованность, и эта критика вполне справедлива. Посмотрите на содержимое каталога файлов регистрации — вы обязательно найдете файлы с такими именами, как **maillog**, **cron.log** и, возможно, еще нечто специфичное для демонов и дистрибутивов. По умолчанию большинство этих файлов хранится в каталоге **/var/adm** или **/var/log**, но некоторые приложения разбрасывают журнальные файлы по файловым системам.

В табл. 10.1 представлена информация о наиболее часто используемых журнальных файлах демонстрационных дистрибутивов. Указаны, в частности, следующие сведения:

- журнальные файлы, подлежащие архивированию или другой обработке;
- программа, создающая каждый из этих файлов;
- информация о том, где задается имя файла;
- периодичность удаления устаревшей информации, которая считается приемлемой;
- системы (из числа демонстрационных), в которых используется каждый из этих файлов;
- описание содержимого файлов.

Имена файлов в табл. 10.1 даны относительно каталогов, **/var/log**, если не указано иное. Многие журнальные файлы из числа перечисленных в табл. 10.1 контролируются системой Syslog, а остальные — приложениями.

Таблица 10.1. Журнальные файлы

Файл	Программа	Место[а]	Частота[а]	Системы[а]	Содержимое/назначение
`apache2/*`	`httpd`	Ф	Д	D	Журналы HTTP-сервера Apache (версия 2)
`apt*`	APT	Ф	М	D	Программы для инсталляции программных пакетов
`auth.log`	`sudo` и др.[б]	S	М	DF	Авторизационные сообщения
`boot.log`	Сценарии `rc`	Ф	М	R	Выходная информация сценариев запуска
`cloud-init.log`	`cloud-init`	Ф	-	-	Выходная информация сценариев запуска облаков
`cron`, `cron/log`	`cron`	S	Н	RA	Сведения о выполнении и об ошибках демона `cron`
`daemon.log`	Различные	S	Н	D*	Все сообщения средств демона
`debug*`	Различные	S	Д	F, D*	Отладочные сообщения
`dmesg`	Ядро	В	-	все	Образ буфера сообщений ядра
`dpkg.log`	`dpkg`	Ф	М	U	Журнал управления пакетом
`faillog`[в]	`login`	Н	Н	D*	Сообщения о неудачных попытках регистрации в системе
`httpd/*`	`httpd`	Ф	Д	R	Журналы HTTP-сервера Apache
`kern.log`	Ядро	S	Н	D	Все сообщения средств ядра
`lastlog`	`login`	В	-	R	Время последней регистрации в системе каждого пользователя (бинарный файл)
`mail*`	Связанные с электронной почтой	S	Н	RF	Все сообщения средств электронной почты
`messages`	Различные	S	Н	R	Основной системный журнальный файл
`samba/*`	`smbd` и др.	Ф	Н	-	Samba (совместное использование файлов в системах Windows/SMB)
`secure`	`sshd` и др.[б]	S	М	R	Конфиденциальные авторизационные сообщения
`sulog`	`su`	Ф	-	SAH	Учет удачных и неудачных попыток использования команды `su`
`syslog*`	Различные	S	Н	D	Основной системный журнальный файл
`wtmp`	`login`	В	М	RD	Сообщения о регистрации в системе (бинарный файл)
`xen/*`	Xen	Ф	1m	RD	Информация от монитора виртуальных машинах Xen
`Xorg.`*n*`.log`	`Xorg`	Ф	Н	R	Сообщения об ошибках сервера X Windows
`yum.log`	`yum`	Ф	М	R	Журнал управления пакетом

[а] Здесь используются следующие обозначения:
в столбце "Место" : Ф — конфигурационный файл, В — встроенный, S — Syslog,
в столбце "Частота": Д — ежедневно, Н — еженедельно, М — ежемесячно, *NNm* — зависит от размера (например, 1m — мегабайт),
в столбце "Системы": D — Debian и Ubuntu, D* — только Debian, R — Red Hat и CentOS, F — FreeBSD.

[б] Команды `passwd`, `sshd`, `login` и `shutdown` тоже записывают информацию в журнал авторизации.

[в] Двоичный файл, который должен быть прочитан с помощью утилиты `faillog`.

Журнальные файлы обычно принадлежат пользователю **root**, хотя соглашения о владельцах и режимах доступа к этим файлам не одинаковы в разных системах. В некоторых случаях менее привилегированный демон (такой как **httpd** или **mysqld**) может потребовать доступ для записи, а затем определить ее владельца и режим работы соответственно. Вам, возможно, для просмотра важных журнальных файлов, которые имеют строгие разрешения доступа, придется использовать команду **sudo**.

🕮 Дополнительную информацию о разделении диска см. в разделе 20.6.

Журнальные файлы могут очень быстро увеличиваться в размере, особенно это касается файлов для службы электронной почты, веб- и DNS-серверов. Неконтролируемый файл регистрации может заполнить весь диск и тем самым вывести систему из строя. Поэтому мы предпочитаем определять раздел **/var/log** как отдельный раздел диска или отдельную файловую систему. (Этот совет одинаково полезен как для облачных экземпляров, и частных виртуальных машин, так и для физических серверов.)

Специальные журнальные файлы

Большинство журнальных файлов — это обычные текстовые файлы, в которые записываются сообщения о "важных" событиях. Но некоторые файлы из числа перечисленных в табл. 10.1 имеют совершенно другое назначение.

Файл **wtmp** (иногда **wtmpx**) содержит записи о том, когда пользователи входили в систему и выходили из нее, а также когда система выключалась или перезагружалась. Это довольно простой файл (новые записи просто добавляются в конец), тем не менее он хранится в бинарном виде. Расшифровать содержимое файла можно с помощью команды **last**.

В файле **lastlog** регистрируется время последнего входа в систему каждого пользователя. Это разреженный бинарный файл, записи которого индексируются по идентификатору пользователя UID. Размер файла будет меньше, если назначать идентификаторы по порядку, хотя вряд ли об этом стоит сильно беспокоиться. Файл **lastlog** не должен участвовать в цикле ротации, поскольку его размер в большинстве случаев остается неизменным.

Кроме того, некоторые приложения (особенно базы данных) создают двоичные файлы транзакций. Не пытайтесь управлять этими файлами и даже просматривать их, иначе терминальное окно выйдет из строя.

Как просмотреть записи в журнале systemd

В системах Linux, в которых ведется журнал **systemd**, самым простым и легким способом просмотра записей является команда **journalctl**, которая выводит на экран сообщения из журнала **systemd**. С ее помощью можно просмотреть все записи в журнале или, указав флаг **-u**, — сообщения, касающиеся конкретного служебного модуля. Сообщения можно фильтровать и устанавливать другие ограничения, такие как интервал времени, идентификатор процесса и даже путь к конкретному выполняемому файлу.

Например, следующая выходная информация извлечена из записей журнала о демоне SSH:

```
$ journalctl -u ssh
-- Logs begin at Sat 2016-08-27 23:18:17 UTC, end at Sat 2016-08-27
   23:33:20 UTC. --
Aug 27 23:18:24 uxenial sshd[2230]: Server listening on 0.0.0.0 port 22.
Aug 27 23:18:24 uxenial sshd[2230]: Server listening on :: port 22.
Aug 27 23:18:24 uxenial systemd[1]: Starting Secure Shell server...
Aug 27 23:18:24 uxenial systemd[1]: Started OpenBSD Secure Shell server.
```

```
Aug 27 23:18:28 uxenial sshd[2326]: Accepted publickey for bwhaley from
    10.0.2.2 port 60341 ssh2: RSA SHA256:aaRfGdl0untn758+UCpxL7gkSwcs
    zkAYe/wukrdBATc
Aug 27 23:18:28 uxenial sshd[2326]: pam_unix(sshd:session): session
    opened for user bwhaley by (uid=0)
Aug 27 23:18:34 uxenial sshd[2480]: Did not receive identification string
    from 10.0.2.2
```

С помощью команды **journalctl -f** можно выводить на экран новые сообщения по мере их поступления. Это эквивалент любимой команды **tail -f**, позволяющей следить за добавляемыми текстовыми файлами по мере их поступления.

В следующем разделе мы рассмотрим демон **systemd-journald** и его конфигурацию.

10.2. Журнал systemd



Журнал **systemd** должен был заменить все остальные подсистемы Linux и поэтому он содержит демон регистрации с именем **systemd-journald**. Он выполняет большинство функций, связанных с регистрацией событий, в зависимости от конфигурации системы. Если вы сомневаетесь. стоит ли переходить на **systemd**, потому что **syslog** и так делает все, что вам нужно, посвятите немного времени изучению возможностей **systemd**. После этого вы убедитесь в его преимуществах.

В отличие от системы **syslog**, которая обычно сохраняет сообщения журнала в текстовых файлах, журнал **systemd** хранит сообщения в двоичном формате. Все атрибуты сообщений индексируются автоматически, что делает журнал проще и быстрее для поиска. Как обсуждалось выше, вы можете использовать команду **journalctl** для просмотра сообщений, хранящихся в журнале.

Журнал собирает и индексирует сообщения из нескольких источников.

- Сокет **/dev/log** для сбора сообщений от программного обеспечения, отправляющего сообщения согласно соглашениям системы Syslog.
- Файл устройства **/dev/kmsg** для сбора сообщений от ядра Linux. Журнальный демон **systemd** заменяет традиционный процесс **klogd**, который ранее прослушивал этот канал и перенаправлял сообщения от ядра в систему Syslog.
- Сокет UNIX **/run/systemd/journal/stdout** для обслуживания программного обеспечения, которое записывает сообщения журнала в стандартный вывод.
- Сокет UNIX **/run/systemd/journal** для сервисного программного обеспечения, отправляющего сообщения через API журнала **systemd**.
- Сообщения аудита от демона ядра **auditd**.

Смелые администраторы могут использовать утилиту **systemd-journal-remote** (и ее аналоги, **systemd-journal-gateway** и **systemd-journal-upload**) для потоковой передачи сериализованных сообщений журнала по сети в удаленный журнал. К сожалению, эта функция не поставляется предустановленной в обычных дистрибутивах. На момент написания данного текста эти пакеты были доступны для систем Debian и Ubuntu, но не для Red Hat или CentOS. Мы ожидаем, что в ближайшее время эта ситуация будет исправлена; тем временем мы рекомендуем придерживаться системы Syslog, если вам нужно пересылать сообщения журналов между системами.

Настройка журнала systemd

Конфигурационный файл журнала по умолчанию — **/etc/systemd/journald.conf**; однако этот файл не предназначен для прямого редактирования. Вместо этого добавьте свои настроенные конфигурации в каталог **/etc/systemd/journald.conf.d**. Любые файлы, размещенные там с расширением **.conf**, автоматически включаются в конфигурацию. Чтобы настроить собственные параметры, создайте в этом каталоге новый файл с расширением **.conf** и включите нужные параметры.

По умолчанию файл **journald.conf** содержит прокомментированную версию каждой возможной опции, а также значение по умолчанию каждого параметра, поэтому вы можете сразу увидеть, какие опции доступны. Они включают в себя максимальный размер журнала, период хранения сообщений и различные ограничения скорости.

Опция `Storage` определяет, следует ли сохранять журнал на диск. Возможные значения несколько сбивают с толку:

- `volatile` сохраняет журнал только в памяти.
- `persistent` сохраняет журнал в каталоге **/var/log/journal/**, создавая его при необходимости.
- `auto` сохраняет журнал в каталоге **/var/log/journal/**, но не создает каталог. Это значение по умолчанию.
- `none` удаляет все данные журнала.

Большинство дистрибутивов Linux (включая все наши примеры) по умолчанию используют значение `auto` и содержат каталог **/var/log/journal**. Следовательно, журнал не сохраняется между перезагрузками по умолчанию, что является неудачным.

Вы можете изменить это поведение либо путем создания каталога **/var/log/journal**, либо путем обновления журнала для использования постоянного хранилища и перезапуска **systemd-journald**:

```
# mkdir /etc/systemd/journald.conf.d/
# cat << END > /etc/systemd/journald.conf.d/storage.conf
[Journal]
Storage=persistent
END
# systemctl restart systemd-journald
```

Эта серия команд создает настраиваемый каталог конфигурации **journald.conf.d**, создает файл конфигурации, чтобы установить параметр `Storage` равным `persistent`, и перезапускает журнал, чтобы новые настройки вступили в силу. Демон **systemd-journald** теперь создаст каталог и сохранит журнал. Мы рекомендуем выполнить это изменение для всех систем; было бы очень обидно терять все данные журнала при каждой перезагрузке системы.

Одним из самых неожиданных параметров журнала является `Seal`, что позволяет системе Forward Secure Sealing (FSS) повысить целостность сообщений журнала. При включенной системе FSS сообщения, отправленные в журнал, не могут быть изменены без доступа к паре криптографических ключей. Вы сами генерируете пару ключей, выполнив команду **journalctl --setup-keys**. Обратитесь к справочным страницам для файла **journald.conf** и команды **journalctl**, чтобы увидеть полный обзор этой опции.

Добавление дополнительных параметров фильтрации для журнала

В конце разделе 10.1. мы привели короткий пример поиска журнала с помощью команды **journalctl**. В этом разделе мы покажем несколько дополнительных способов использования команды **journalctl** для фильтрации сообщений и сбора информации о журнале.

Для того чтобы нормальные пользователи могли читать из журнала без необходимых разрешений **sudo**, добавьте их в группу UNIX `systemd-journal`.

Опция **-disk-usage** показывает размер журнала на диске.

```
# journalctl --disk-usage
Journals take up 4.0M on disk.
```

Параметр **-list-boots** показывает последовательный список системных загрузок с числовыми идентификаторами. Самая последняя загрузка всегда имеет идентификатор, равный 0. Даты в конце строки показывают временные метки первого и последнего сообщений, сгенерированных во время этой загрузки.

```
# journalctl --list-boots
-1 ce0 ... Sun 2016-11-13 18:54:42 UTC-Mon 2016-11-14 00:09:31
 0 844 ... Mon 2016-11-14 00:09:38 UTC-Mon 2016-11-14 00:12:56
```

Вы можете использовать опцию **-b**, чтобы ограничить отображение журнала определенным сеансом загрузки. Например, для просмотра журналов, сгенерированных алгоритмом SSH во время текущего сеанса:

```
# journalctl -b 0 -u ssh
```

Для того чтобы показать все сообщения, начиная с прошлой полночи до сегодняшнего дня, выполните команду:

```
# journalctl --since=yesterday --until=now
```

Для того чтобы показать последние 100 записей журнала из определенного двоичного файла, выполните команду:

```
# journalctl -n 100 /usr/sbin/sshd
```

Для получения краткой справки об этих аргументах используйте команду **journalctl --help**.

Совместное использование с системой Syslog

Как система Syslog, так и журнал **systemd** по умолчанию активны для каждой из наших демонстрационных систем Linux. Оба пакета собирают и сохраняют сообщения журнала. Зачем нужно, чтобы оба они работали, и как это сделать?

К сожалению, в журнале отсутствуют многие функции, доступные в системе Syslog. Как будет показано в разделе 10.3, система rsyslog может получать сообщения от различных входных дополнительных модулей и перенаправлять их на разнообразный набор выходов в соответствии с фильтрами и правилами, которые невозможны, если используется журнал **systemd**. В среде **systemd** существует инструмент удаленного потока, **systemd-journal-remote**, но он относительно новый и не сравнивался с системой Syslog. Администраторам также может быть удобно хранить определенные файлы журналов в виде обычного текста, как это делает система Syslog, а не в двоичном формате журнала.

Мы ожидаем, что со временем новые функции в журнале узурпируют обязанности системы Syslog. Но на данный момент дистрибутивам Linux по-прежнему необходимо запустить обе системы для достижения полной функциональности.

Механика взаимодействия между журналом **systemd** и системой Syslog несколько запутанна. Начнем с того, что демон **systemd-journald** берет на себя ответственность за сбор журнальных сообщений из **/dev/log**, сокета протоколирования, который исторически контролировался системой Syslog.[2] Для того чтобы система Syslog могла выполнить регистрацию, она должна получить доступ к потоку сообщений через системный менеджер **systemd**. Система Syslog может извлекать сообщения из журнала двумя способами.

- Журнал **systemd** может пересылать сообщения другому сокету (обычно **/run/systemd/journal/syslog**), из которого демон системы Syslog может их читать. В этом режиме демон **systemd-journald** имитирует исходных отправителей сообщений в соответствии со стандартным API системы Syslog. Следовательно, перенаправляются только основные параметры сообщения; некоторые метаданные, специфичные для системы, теряются.
- В качестве альтернативы система Syslog может обрабатывать сообщения непосредственно из интерфейса прикладного программирования журнала таким же образом, как и команда **journalctl**. Этот метод требует явной поддержки сотрудничества со стороны **syslogd**, но это более полная форма интеграции, которая сохраняет метаданные для каждого сообщения.[3]

Системы Debian и Ubuntu по умолчанию используют прежний метод, но Red Hat и CentOS используют последний. Чтобы определить, какой тип интеграции был настроен в вашей системе, проверьте опцию ForwardToSyslog в файле **/etc/systemd/journald.conf**. Если его значение равно `yes`, используется переадресация сокетов.

10.3. Система Syslog

Syslog — это полнофункциональная система регистрации событий, написанная Эриком Оллманом (Eric Allman), и стандартный протокол регистрации событий, утвержденный IETF.[4] Она выполняет две важные функции: освобождает программистов от утомительной механической работы по ведению журнальных файлов и передает управление журнальной регистрацией в руки администраторов. До появления системы Syslog каждая программа сама выбирала схему регистрации событий, а у системных администраторов не было возможности контролировать, какая информация и где именно сохраняется.

Система Syslog отличается высокой гибкостью. Она позволяет сортировать сообщения по источникам (facility) и уровню важности (severity) и направлять их в различные пункты назначения: в журнальные файлы, на терминалы пользователей и даже на другие компьютеры. Одной из самых ценных особенностей этой системы является ее способность централизовать процедуру регистрации событий в сети.

---

[2]Точнее, ссылки журнала из **/dev/log** в **/run/systemd/journal/dev-log**.

[3]Краткое описание доступных метаданных см. на справочной странице **systemd.journal-fields**.

[4]Последняя по времени версия спецификации системы Syslog — RFC5424, но предыдущая версия, RFC3164, лучше отражает реальную инсталлированную систему.

В системах Linux исходный демон системы Syslog (**syslogd**) был заменен более новой реализацией, называемой Rsyslog (**rsyslogd**). Rsyslog — проект с открытым исходным кодом, который расширяет возможности исходной системы Syslog, но поддерживает обратную совместимость с интерфейсом прикладного программирования. Это самый разумный выбор для администраторов, работающих в современных системах UNIX и Linux, и это единственная версия Syslog, которую мы рассмотрим в этой главе.



Система Rsyslog доступна для FreeBSD, и мы рекомендуем вам принять ее, а не стандартный системный журнал FreeBSD, если у вас нет простых требований. Инструкции по преобразованию системы FreeBSD для использования демона **rsyslog** содержатся по адресу `wiki.rsyslog.com/index.php/FreeBSD`.

Если вы решите придерживаться традиционной системы `syslog` в операционной системе FreeBSD, перейдите в раздел "Синтаксис **sysklogd**" для получения информации о конфигурации.

Чтение сообщений системы Syslog

Простые сообщения системы Syslog можно читать с помощью инструментов систем UNIX и Linux для обработки текста, например команд **grep**, **less**, **cat** и **awk**. Приведенный ниже фрагмент кода демонстрирует сообщения о типичных событиях в журнале **/var/log/syslog**, поступившие от хоста системы Debian.

```
jessie# cat /var/log/syslog
Jul 16 19:43:01 jessie networking[244]: bound to 10.0.2.15 -- renewal in
    42093 seconds.
Jul 16 19:43:01 jessie rpcbind[397]: Starting rpcbind daemon....
Jul 16 19:43:01 jessie nfs-common[412]: Starting NFS common utilities:
    statd idmapd.
Jul 16 19:43:01 jessie cron[436]: (CRON) INFO (pidfile fd = 3)
Jul 16 19:43:01 jessie cron[436]: (CRON) INFO (Running @reboot jobs)
Jul 16 19:43:01 jessie acpid: starting up with netlink and the input layer
Jul 16 19:43:01 jessie docker[486]: time="2016-07-
    16T19:43:01.972678480Z" level=info msg="Daemon has completed
    initialization"
Jul 16 19:43:01 jessie docker[486]: time="2016-07-
    16T19:43:01.972896608Z" level=info msg="Docker daemon"
    commit=c3959b1 execdriver=native-0.2 graphdriver=aufs
    version=1.10.2
Jul 16 19:43:01 jessie docker[486]: time="2016-07-
    16T19:43:01.979505644Z" level=info msg="API listen on /var/run/
    docker.sock"
```

Этот пример содержит сообщения, поступившие от нескольких разных демонов и подсистем: сети, NFS, **cron**, Docker и демона управления питанием **acpid**. Каждое сообщение содержит следующие поля, разделенные пробелами.

- Временная метка.
- Системное имя хоста, в данном случае `jessie`.
- Имя процесса и его идентификатор в квадратных скобках.
- Полезная нагрузка сообщения.

Некоторые демоны кодируют полезную нагрузку сообщения, добавляя в них метаданные, которые сопровождают это сообщения. В приведенном выше примере процесс **docker** включает свою временную метку, уровень журнала и информацию о конфигурации демона. Эту дополнительную информацию генерирует и форматирует посылающий процесс.

Архитектура системы Rsyslog

Сообщения журнала можно представить как поток событий, а систему Rsyslog — как механизм обработки потока событий. Журнальные сообщения, которые интерпретируются как события, представляются в виде входов, обрабатываются фильтрами и пересылаются по адресу назначения. В системе Rsyslog каждый из этих этапов является конфигурируемым и модульным. По умолчанию система Rsyslog настроена в файле **/etc/rsyslog.conf**.

Процесс **rsyslogd** обычно запускается при загрузке и выполняется непрерывно. Программы, которые известны системе Syslog, заносят записи журнала в специальный файл **/dev/log**, сокет домена UNIX. В конфигурации систем без демона **systemd** демон **rsyslogd** напрямую считывает сообщения из этого сокета, консультируется с его конфигурационным файлом для руководства по их маршрутизации и отправляет каждое сообщение в соответствующее место назначения. Также можно (и желательно) настраивать демон **rsyslogd** для прослушивания сообщений в сетевом сокете.

Если вы изменяете файл **/etc/rsyslog.conf** или любой из его включенных файлов, то должны перезапустить демон **rsyslogd**, чтобы ваши изменения вступили в силу. Сигнал TERM прекращает работу демона. Сигнал HUP заставляет демон **rsyslogd** закрывать все открытые файлы журналов, что полезно для ротации (переименования и перезапуска) журналов.

По сложившейся традиции демон **rsyslogd** записывает свой идентификатор процесса в файл **/var/run/syslogd.pid**, поэтому послать сигнал из сценария в процесс **rsyslogd** не составляет труда.[5] Например, следующая команда посылает сигнал зависания.

```
$ sudo kill -HUP `/bin/cat /var/run/syslogd.pid`
```

Попытка сжать или выполнить ротацию журнального файла, который был открыт демоном **rsyslogd** для записи, — плохая идея, которая приводит к непредсказуемым результатам, поэтому перед этим обязательно отправьте сигнал HUP. Информацию о правильной ротации журнала с помощью утилиты **logrotate** см. в разделе 10.4.

Версии Rsyslog

Системы Red Hat и CentOS используют Rsyslog версии 7, а Debian и Ubuntu обновлены до версии 8. Пользователи FreeBSD, устанавливающие систему из портов, могут выбрать либо версию 7, либо 8. Как и следовало ожидать, в проекте Rsyslog рекомендуется использовать самую последнюю версию, но мы не следуем этим советам. Тем не менее, если ваша операционная система является самой современной, это не повлияет на ваш опыт ведения журнала.

Версия Rsyslog 8 является основным переработанным вариантом основного механизма, и, хотя в его устройстве многое изменилось с точки зрения разработчиков модулей, аспекты, обращенные к пользователю, остаются в основном неизменными. За некоторыми исключениями, конфигурации в следующих разделах действительны для обеих версий.

[5] В современных версиях системы Linux **/var/run** является символической ссылкой на **/run**.

Конфигурация Rsyslog

Поведение демона **rsyslogd** контролируется настройками в файле **/etc/rsyslog.conf**. Все наши примеры дистрибутивов Linux включают в себя простую конфигурацию с разумными значениями по умолчанию, которые подходят большинству организаций. Пустые строки и строки, начинающиеся с символа **#**, игнорируются. Строки в конфигурации системы Rsyslog обрабатываются в порядке от начала до конца, а порядок имеет значение.

В верхней части файла конфигурации находятся глобальные свойства, которые настраивают сам демон. В этих строках указаны загружаемые модули, формат сообщений по умолчанию, права собственности и разрешения файлов, рабочий каталог, в котором можно сохранить состояние системы Rsyslog, и другие параметры. Следующая примерная конфигурация адаптирована из стандартного файла **rsyslog.conf** в версии Debian Jessie.

```
# Поддержка локальной системы регистрации
$ModLoad imuxsock

# Поддержка регистрации ядра
$ModLoad imklog

# Выводить сообщения в традиционном формате с временными метками
$ActionFileDefaultTemplate RSYSLOG_TraditionalFileFormat

# Новым файлам регистрации назначается владелец root:adm
$FileOwner root
$FileGroup adm

# Стандартные права доступа для вновь созданных файлов и каталогов
$FileCreateMode 0640
$DirCreateMode 0755
$Umask 0022

# Каталог, в котором хранятся рабочие файлы rsyslog
$WorkDirectory /var/spool/rsyslog
```

Большинство дистрибутивов используют устаревшую директиву `$IncludeConfig` для включения дополнительных файлов из каталога конфигурации, обычно **/etc/rsyslog.d/*.conf**. Поскольку порядок важен, дистрибутивы организуют файлы с помощью предшествующих имен файлов с номерами. Например, конфигурация Ubuntu по умолчанию включает следующие файлы:

```
20-ufw.conf
21-cloudinit.conf
50-default.conf
```

Система Rsyslogd интерполирует эти файлы в файл **/etc/rsyslog.conf** в лексикографическом порядке, чтобы сформировать окончательную конфигурацию.

Фильтры, иногда называемые *селекторами*, составляют основную часть конфигурации системы Rsyslog. Они определяют, как система Rsyslog сортирует и обрабатывает сообщения. Фильтры формируются из выражений, выбираются конкретные критерии сообщений и действия, которые направляют выбранные сообщения в нужное место назначения.

Система Rsyslog понимает три синтаксиса конфигурации.

- Строки, использующие формат исходного файла конфигурации систем Syslog. После появления демона регистрации ядра **sysklogd** этот формат известен как "формат **sysklogd**". Он прост и эффективен, но имеет некоторые ограничения. Используйте его для создания простых фильтров.
- Устаревшие директивы системы Rsyslog, которые всегда начинаются с знака **$**. Их синтаксис уходит корнями в старые версии систем Rsyslog и действительно должен быть признан устаревшим. Однако не все опции были преобразованы в новый синтаксис, и поэтому этот синтаксис остается важным для некоторых функций.
- Синтаксис RainerScript, названный в честь Райнера Герхардса (Reiner Gerhards), ведущего автора системы Rsyslog. Это синтаксис сценариев, который поддерживает выражения и функции. Вы можете использовать его для настройки большинства, но не всех аспектов системы Rsyslog.

Многие конфигурации в реальном мире включают в себя сочетание всех трех форматов, иногда с запутанным эффектом. Хотя синтаксис RainerScript существует с 2008 г., он все еще используется немного реже других. К счастью, ни один из диалектов не является особенно сложным. Кроме того, на многих сайтах не придется делать крупную операцию по простым конфигурациям, включенным в их распределение ресурсов.

Для того чтобы перейти из традиционной конфигурации Syslog, просто начните с существующего файла **syslog.conf** и добавьте параметры для функций системы Rsyslog, которые вы хотите активировать.

Модули

Модули системы Rsyslog расширяют возможности основного механизма обработки. Все входы (источники) и выходы (адресаты) настраиваются через модули, а модули могут даже анализировать и изменять сообщения. Хотя большинство модулей были написаны Райнером Герхардсом, некоторые из них были внесены третьими лицами. Если вы программист на языке C, то можете написать свой собственный.

Имена модулей соответствуют предсказуемому префиксному шаблону. Те, которые начинаются с префикса `im`, являются модулями ввода; `om*` — модули вывода, `mm*` — модули модификации сообщений и т.д. Большинство модулей имеют дополнительные параметры конфигурации, которые настраивают их поведение. Документация о модулях системы Rsyslog является полным и исчерпывающим справочником.

В следующем списке кратко описаны некоторые из наиболее распространенных (или интересных) модулей ввода и вывода, а также несколько экзотических примеров.

- Модуль `imjournal` интегрируется с журналом **systemd**, как описано в разделе "Совместное использование с системой Syslog".
- Модуль `imuxsock` считывает сообщения из сокета домена UNIX. Если журнал **systemd** отсутствует, это значение устанавливается по умолчанию.
- Модуль `imklog` понимает, как читать сообщения ядра в системах Linux и BSD.
- Модуль `imfile` преобразует простой текстовый файл в формат сообщения системы Syslog. Это полезно для импорта файлов журналов, созданных с помощью программного обеспечения, не имеющего встроенной поддержки Syslog. Существуют два режима: режим опроса, который проверяет файл на наличие обновлений в настраиваемый интервал и режим уведомления (`inotify`), который использует интерфейс событий файловой системы Linux. Этот модуль допускает восстановление после отключения при перезапуске демона **rsyslogd**.

- Модули `imtcp` и `imudp` принимают сетевые сообщения через TCP и UDP соответственно. Они позволяют централизовать ведение журнала в сети. В сочетании с драйверами сетевого потока системы Rsyslog модуль TCP также может принимать взаимно аутентифицированные сообщения системы Syslog через TLS. Для сайтов Linux с чрезвычайно высоким объемом см. также модуль `imptcp`.
- Если модуль `immark` присутствует, система Rsyslog создает сообщения с отметкой времени через равные промежутки времени. Эти метки времени могут помочь вам понять, что ваша машина потерпела крах между 3:00 и 3:20 утра, а не просто ночью. Эта информация также является большой помощью, когда вы отлаживаете проблемы, которые происходят регулярно. Для настройки интервала метки используйте параметр `MarkMessagePeriod`.
- Модуль `omfile` записывает сообщения в файл. Это наиболее часто используемый модуль вывода и единственный, который настроен при установке по умолчанию.
- Модуль `omfwd` пересылает сообщения на удаленный сервер Syslog через TCP или UDP. Этот модуль особенно полезен, если ваша организация нуждается в централизованном протоколировании.
- Модуль `omkafka` — это реализация производителя для потокового движка Apache Kafka. Пользователи крупных сайтов могут извлечь выгоду из возможности обрабатывать сообщения, у которых есть много потенциальных потребителей.
- Аналогично модулю `omkafka` модуль `omelasticsearch` записывается непосредственно в кластер Elasticsearch. Дополнительную информацию о стеке управления журналом ELK, который включает Elasticsearch в качестве одного из своих компонентов, см. в разделе 10.5.
- Модуль `ommysql` отправляет сообщения в базу данных MySQL. Распределение источников Rsyslog содержит примерную схему. Для повышения надежности объедините этот модуль с устаревшей директивой `$MainMsgQueueSize`.

Модули можно загружать и настраивать либо с помощью устаревших форматов конфигурации, либо с помощью синтаксиса RainerScript. Ниже мы приводим некоторые примеры в разделах, посвященных конкретным форматам.

Синтаксис демона `sysklogd`

Синтаксис демона **`sysklogd`** — это традиционный формат системы Syslog. Если вы знаете, как работает стандартный демон **`syslogd`**, например его версия, инсталлированная в системе FreeBSD, то, вероятно, вы знаете все, что требуется. (Но учтите, что файл конфигурации традиционного демона **`syslogd`** называется **`/etc/syslog.conf`**, а не **`/etc/rsyslog.conf`**.)

Изначально этот формат был предназначен для пересылки сообщений определенного типа в заданный файл или по заданному сетевому адресу. Базовый синтаксис записей файла таков.

селектор *действие*

Селектор отделен от действия одним или несколькими пробелами или знаком табуляции. Например, строка

```
auth.*          /var/log/auth.log
```

обеспечивает сохранение сообщений, связанных с аутентификацией пользователей, в файле **`/var/log/auth.log`**.

Селектор указывает источник (facility), посылающий журнальное сообщение, и уровень важности (severity) этого сообщения. Формат селектора выглядит следующим образом.

источник.уровень_важности

Названия источников и уровней важности следует выбирать из стандартного списка значений; программы не могут самостоятельно составлять такие списки. Есть источники, определенные для ядра, для основных групп утилит и для локальных программ. Все остальное проходит под общим названием *user* (т.е. пользователь).

Селекторы могут содержать символы * и none, которые обозначают "все" и "ничего" соответственно. В селекторе разрешается через запятые перечислять группу источников. Допускается также разделение самих селекторов с помощью точки с запятой.

В общем случае селекторы объединяются операцией OR, т.е. указанное действие будет выполняться для сообщения, соответствующего любому селектору. Селектор уровня none означает исключение перечисленных в нем источников, независимо от того, что указано в остальных селекторах этой же строки.

Приведем несколько примеров селекторов.

```
# Применяем действие ко всем сообщениям от facility.level
facility.level                              action

# Применяем действие ко всем сообщениям от facility1.level и facility2.level
facility1, facility2.level                  action

# Применяем действие только к сообщениям от facility1.level1 и facility2.level2
facility1.level1; facility2.level2          action

# Применяем действие только источникам с заданным уровнем важности
*.level                                     action

# Применяем действие ко всем источникам, кроме badfacility
*.level;badfacility.none                    action
```

В табл. 10.2 перечислены допустимые названия источников. Они определены в стандартной библиотеке в файле **syslog.h**.

Таблица 10.2. Названия средств Syslog

Источник	Программы или сообщения, которые его используют
*	Все источники, кроме mark
auth	Команды, связанные с безопасностью и авторизацией
authpriv	Конфиденциальные авторизационные сообщения
cron	Демон **cron**
daemon	Системные демоны
ftp	Демон **ftpd** (устаревший)
kern	Ядро
local0-7	Восемь разновидностей локальных сообщений
lpr	Система спулинга построчной печати
mail	Система **sendmail**, **postfix** и другие почтовые программы
mark	Метки времени, генерируемые через одинаковые промежутки
news	Система телеконференций Usenet (устаревшая)
syslog	Внутренние сообщения демона **syslogd**
user	Пользовательские процессы (это установка по умолчанию, если не указано иное)

Не воспринимайте слишком серьезно различие между сообщениями системы Syslog типа `auth` и `authpriv`. В действительности все авторизационные сообщения являются конфиденциальными, и ни одно из них не должно быть открытым для всеобщего доступа. Журнал **sudo** использует `authpriv`.

В табл. 10.3 перечислены уровни важности, существующие в системе Syslog (в порядке убывания важности).

Таблица 10.3. Уровни важности сообщений Syslog

Уровень	Приблизительное значение
`emerg`	Экстренные сообщения; система вышла из строя
`alert`	Срочные сообщения; требуются срочные действия
`crit`	Критические состояния
`err`	Другие ошибочные состояния
`warning`	Предупреждения
`notice`	Необычные события, которые заслуживают внимания
`info`	Информационные сообщения
`debug`	Отладочные сообщения

Уровень сообщения определяет его важность. Границы между различными уровнями несколько размыты. Существует четкое различие между событиями, заслуживающими внимания, и предупреждениями (а также между предупреждениями и сообщениями об ошибках), но трудно однозначно разграничить значения уровней `alert` и `crit`.

Уровни обозначают минимальную важность, которую сообщение должно иметь для того, чтобы быть зарегистрированным. Например, сообщение от SSH имеющее уровень важности `warning`, будет соответствовать селектору `auth.warning`, а также селекторам `auth.info`, `auth.notice`, `auth.debug`, `*.warning`, `*.notice`, `*.info` и `*.debug`. Если в конфигурации указано, что сообщения `auth.info` должны регистрироваться в файле, то сообщения `auth.warning` тоже будут направляться в этот файл.

Этот формат также допускает символы = и ! перед названиями уровней, означающие "только этот уровень" и "за исключением этого и более высоких уровней" соответственно (табл. 10.4).

Таблица 10.4. Примеры использования квалификаторов уровня

Селектор	Назначение
`auth.info`	Выбор почтовых сообщений уровня `info` и выше
`auth.=info`	Выбор почтовых сообщений только уровня `info`
`auth.info;auth.!err`	Выбор почтовых сообщений уровней `info`, `notice` и `warning`
`auth.debug;auth.!=warning`	Выбор почтовых сообщений любого уровня, кроме `warning`

Поле *действие* определяет способ обработки сообщения. Возможные варианты перечислены в табл. 10.5.

Таблица 10.5. Типичные действия

Действие	Назначение
имя_файла	Дописать сообщение в указанный файл на локальном компьютере
@имя_хоста	Переслать сообщение демону **rsyslogd**, выполняющемуся на компьютере с указанным именем

Окончание табл. 10.5

Действие	Назначение
@IP_адрес	Переслать сообщение на *IP_адрес* по протоколу UDP, порт 514
@@IP_адрес	Переслать сообщение на *IP_адрес* по протоколу TCP, порт 514
|имя_FIFO	Записать сообщение в указанный именованный канал[a]
пользователь1, пользователь2,...	Вывести сообщение на экраны терминалов указанных пользователей, если они зарегистрированы в системе
*	Вывести сообщение на экраны терминалов всех пользователей, зарегистрированных в системе
~	Игнорировать сообщение
^программа;шаблон	Форматировать сообщение в соответствии со спецификацией *шаблон* и послать его *программе* как первый аргумент[6]

[a] Для получения дополнительной информации о каналах FIFO введите команду **man mkfifo**.

[6] Для получения дополнительной информации о шаблонах введите команду **man 5 rsyslog.conf**.

Если в качестве действия задано имя файла (или имя канала FIFO), то это должно быть полное имя. При отсутствии указанного файла демон **rsyslogd** создает его в процессе записи первого сообщения. Права владения и разрешения для файла указаны в глобальных директивах конфигурации (см. выше).

Приведем для примера несколько конфигураций, использующих традиционный синтаксис:

```
# Отправить сообщения ядра в файл kern.log
kern.*                                  -/var/log/kern.log
# Отправить сообщения Cron в файл cron.log
cron.*                                  /var/log/cron.log
# Отправить сообщения авторизации в файл auth.log
auth,authpriv.*                         /var/log/auth.log
# Отправить все остальные сообщения в файл syslog
*.*;auth,authpriv,cron,kern.none        -/var/log/syslog
```

Перед действием *имя_файла* можно ставить дефис, чтобы указать, что файловая система не должна синхронизироваться после внесения каждой журнальной записи. Синхронизация помогает сохранить как можно больше журнальной информации в случае сбоя, но для перегруженных журнальных файлов она может оказаться разрушительной с точки зрения производительности ввода-вывода. Мы рекомендуем ставить дефис (и, следовательно, запрещать синхронизацию) по умолчанию. Удаляйте тире только на время, когда исследуете проблему, которая вызывает панику ядра.

Устаревшие директивы

Несмотря на то что в документации по системе Rsyslog эти директивы называются устаревшими, они все еще широко используются во многих конфигурациях Rsyslog configurations. Устаревшие директивы могут конфигурировать все аспекты системы Rsyslog, включая глобальные параметры демонов, модули, фильтрацию и правила.

Однако на практике эти директивы чаще всего используются для настройки модулей и самого демона **rsyslogd**. Документация по системе Rsyslog не рекомендует использовать устаревшие директивы для форматирования правил обработки сообщений, утверждая, что это "крайне трудно сделать правильно". Для фильтрации и обработки сообщений следует использовать демон **sysklogd** или форматы RainerScript.

Параметры демонов и модулей очевидны. Например, приведенные ниже параметры позволяют регистрацию по протоколам UDP и TCP на стандартном порту системы

Syslog (514). Они также позволяют посылать клиентам пакеты активации, чтобы поддерживать соединения TCP открытыми; это позволяет уменьшить стоимость восстановления соединений после завершения сеанса.

```
$ModLoad imudp
$UDPServerRun 514
$ModLoad imtcp
$InputTCPServerRun 514
$InputTCPServerKeepAlive on
```

Для того чтобы ввести данные параметры в действие, необходимо добавить эти строки в новый файл, который включается в основную конфигурацию, например **`/etc/rsyslog.d/10-network-inputs.conf`**. Затем следует заново запустить демон **`rsyslogd`**. Любые параметры, модифицирующие поведение модулей, должны устанавливаться после того, как модуль будет загружен.

Некоторые из устаревших директив перечислены в табл. 10.6.

Таблица 10.6. Устаревшие параметры конфигурации системы Rsyslog

Параметр	Описание
`$MainMsgQueueSize`	Размер буфера памяти между полученным и отосланным сообщениями[a]
`$MaxMessageSize`	По умолчанию равен 8 Кбайт; должен быть установлен до загрузки любых модулей ввода
`$LocalHostName`	Замещает локальное имя хоста
`$WorkDirectory`	Указывает, где сохранить рабочие файлы системы Rsyslog
`$ModLoad`	Загружает модуль
`$MaxOpenFiles`	Модифицирует системный параметр `nofile`, заданный по умолчанию для системы Rsyslogd
`$IncludeConfig`	Включает дополнительные файлы конфигурации
`$UMASK`	Устанавливает флаг `umask` для новых файлов, созданных системой Rsyslogd

[a] Этот параметр полезен для медленно выполняемых действий, например для вставок в базу данных.

Синтаксис RainerScript

Синтаксис RainerScript — это язык обработки потока событий с возможностями фильтрации и управления потоком. Теоретически с помощью синтаксиса RainerScript можно также установить основные параметры демона **`rsyslogd`**. Однако, поскольку некоторые устаревшие параметры не имеют эквивалентов в синтаксисе RainerScript, смешивать два формата совершенно нецелесообразно.

Язык RainerScript более выразительный и понятный для человека, чем устаревшие директивы демона **`rsyslogd`**, но его синтаксис принципиально отличается от всех известных нам систем конфигурации. На практике, работа с этим синтаксисом немного утомляет. Тем не менее мы рекомендуем его для фильтрации и разработки правил обработки сообщений. В этом разделе мы обсудим лишь часть его функциональных возможностей.



Среди наших иллюстративных систем только Ubuntu по умолчанию использует синтаксис RainerScript в файлах конфигурации. Однако формат RainerScript можно использовать в любых системах, в которых есть Rsyslog версии 7 и выше.

Глобальные параметры демона можно установить с помощью объекта конфигурации `global()`. Например:

```
global(
   workDirectory="/var/spool/rsyslog"
   maxMessageSize="8192"
)
```

У большинства устаревших директив есть эквиваленты в языке RainerScript, например `workDirectory` и `maxMessageSize`, приведенные выше. Эквивалентный устаревший синтаксис для этой конфигурации выглядел бы следующим образом.

```
$WorkDirectory /var/spool/rsyslog
$MaxMessageSize 8192
```

Кроме того, с помощью языка RainerScript можно загрузить модули и установить их параметры. Например, чтобы загрузить модули UDP и TCP и применить к ним конфигурацию, показанную выше, можно выполнить следующий сценарий RainerScript:

```
module(load="imudp")
input(type="imudp" port="514")
module(load="imtcp" KeepAlive="on")
input(type="imtcp" port="514")
```

В языке RainerScript модули имеют параметры модуля и параметры ввода. Модуль загружается только один раз, а параметр модуля (например, параметр `KeepAlive` в модуле `imtcp`, упомянутом выше) применяется к модулю глобально. В отличие от этого, параметры ввода можно применять к одному и тому же модулю несколько раз. Например, мы могли бы приказать системе Rsyslog прослушивать порты TCP 514 и 1514:

```
module(load="imtcp" KeepAlive="on")
input(type="imtcp" port="514")
input(type="imtcp" port="1514")
```

Большая часть преимуществ системы RainerScript связана с возможностями фильтрации. С помощью его выражений можно выбирать сообщения. соответствующие определенным критериям, а затем применять к ним определенное действие. Например, следующие строки направляют сообщения аутентификации в файл **`/var/log/auth.log`**:

```
if $syslogfacility-text == 'auth' then {
    action(type="omfile" file="/var/log/auth.log")
}
```

В данном примере `$syslogfacility-text` — это свойство сообщения, т.е. часть метаданных сообщения. Знак доллара, стоящий перед именем свойства, сообщает системе Rsyslog, что это переменная. В данном случае действие заключается в том, что для записи соответствующих сообщений в файл **`auth.log`** используется модуль вывода `omfile`.

Наиболее часто используемые свойства перечислены в табл. 10.7.

Таблица 10.7. Наиболее часто используемые свойства системы Rsyslog

Свойство	Описание
`$msg`	Текст сообщения без метаданных
`$rawmsg`	Полное сообщение, включая метаданные
`$hostname`	Имя хоста, пославшего сообщение
`$syslogfacility`	Источник Syslog в числовой форме; см. стандарт RFC3164
`$syslogfacility-text`	Источник Syslog в текстовой форме
`$syslogseverity`	Уровень важности Syslog в числовой форме; см. стандарт RFC3164
`$syslogseverity-text`	Уровень важности Syslog в текстовой форме
`$timegenerated`	Время получения сообщения демоном **`rsyslogd`**
`$timereported`	Временная метка сообщения

Фильтр может содержать несколько других фильтров и действий. Следующий фрагмент кода направляет сообщения ядра с разными уровнями важности. Он записывает эти сообщения в файл и посылает по электронной почте, чтобы предупредить администратора о проблеме.

```
module(load="ommail")

if $syslogseverity-text == 'crit' and $syslogfacility-text == 'kern' then {
    action(type="omfile" file="/var/log/kern-crit.log")
    action(type="ommail"
        server="smtp.admin.com"
        port="25"
        mailfrom="rsyslog@admin.com"
        mailto="ben@admin.com"
        subject.text="Critical kernel error"
        action.execonlyonceeveryinterval="3600"
    )
}
```

Здесь указано, что мы не желаем генерировать больше одного сообщения электронной почти в час (3600 с).

Фильтрующие выражения допускают использование регулярных выражений, функций и других сложных механизмов. Более подробные сведения об этом можно найти в документации по формату RainerScript.

Примеры конфигурационных файлов

В этом разделе показаны три примера конфигурации систем Rsyslog. Первая — это базовая, но полная конфигурация, записывающая сообщения журналов в файлы. Второй пример — клиент журналирования, пересылающий сообщения системы Syslog и **httpd**-доступа, а также журналы ошибок на центральный сервер журналирования. Последний пример — это соответствующий сервер журналирования, принимающий сообщения от различных клиентов.

В этих примерах в основном используется формат RainerScript, потому что именно он был предложен для новейшей версии системы Rsyslog. Некоторые параметры являются корректными только в версии Rsyslog 8 и связаны с Linux-специфичными настройками, такими как `inotify`.

Базовая конфигурация системы Rsyslog

Следующий файл можно использовать как универсальный файл конфигурации **rsyslog.conf** в формате RainerScript в любой системе семейства Linux.

```
module(load="imuxsock")                    # Локальная система журналов
module(load="imklog")                      # Журнал ядра
module(load="immark" interval="3600")      # Часовая метка сообщений

# Настройка глобальных параметров rsyslogd
global(
    workDirectory = "/var/spool/rsyslog"
    maxMessageSize = "8192"
)

# Модуль выходного файла не обязательно загружать явно,
# но мы можем это сделать, чтобы заменить параметры, заданные по умолчанию.
```

```
module(load="builtin:omfile"
    # Используем традиционный формат временной метки
    template="RSYSLOG_TraditionalFileFormat"

    # Устанавливаем разрешения по умолчанию для всех журнальных файлов.
    fileOwner="root"
    fileGroup="adm"
    dirOwner="root"
    dirGroup="adm"
    fileCreateMode="0640"
    dirCreateMode="0755"
)

# Включаем файлы из /etc/rsyslog.d; они не эквивалентны RainerScript
$IncludeConfig /etc/rsyslog.d/*.conf
```

Этот пример начинается с установки нескольких параметров по умолчанию для демона **rsyslogd**. Код разрешения для новых журнальных файлов, заданный по умолчанию и равный 0640, устанавливает более строгие ограничения, чем параметр **omfile**, заданный по умолчанию и равный 0644.

Сетевой клиент журналирования

Этот клиент журналирования пересылает системные журналы, параметры доступа Apache и журналы ошибок на удаленный сервер с помощью протокола TCP.

```
# Посылаем все сообщения системы Syslog на сервер,
# используя синтаксис sysklogd
*.*            @@logs.admin.com

# imfile считывает сообщения из файла
# inotify эффективнее, чем опрашивание
# Это значение задано по умолчанию и приведено лишь для иллюстрации
module(load="imfile" mode="inotify")
# Импортируем журналы Apache с помощью модуля imfile
input(type="imfile"
    Tag="apache-access"
    File="/var/log/apache2/access.log"
    Severity="info"
)

input(type="imfile"
    Tag="apache-error"
    File="/var/log/apache2/error.log"
    Severity="info"
)

# Посылаем журналы Apache logs на центральный сервер журналирования
if $programname contains 'apache' then {
      action(type="omfwd"
      Target="logs.admin.com"
      Port="514"
      Protocol="tcp"
   )
}
```

По умолчанию демон Apache **httpd** не записывает сообщения в систему Syslog, поэтому журналы доступа и ошибок считываются из текстового файла с помощью модуля **imfile**.[6] Эти сообщения помечаются для дальнейшего использования в выражениях фильтрации.

Инструкция `if` представляет собой выражение фильтрации, которое ищет сообщения сервера Apache и пересылает их на центральный сервер журналирования `logs.admin.com`. Журналы пересылаются с помощью протокола TCP, который, хотя и является более надежным, чем протокол UDP, все еще может терять сообщения. Для гарантированной доставки можно использовать нестандартный модуль вывода RELP (Reliable Event Logging Protocol).

В реальных сценариях часть этой конфигурации, связанную с сервером Apache, можно записать в файл **/etc/rsyslog.d/55-apache.conf** в виде фрагмента конфигурации этого сервера.

Центральный сервер журналирования

Конфигурация соответствующего центрального сервера журналирования совершенно проста: прослушиваем входящие сообщения через порт TCP 514, фильтруем их по типу журнала и записываем файлы в каталог журналов сайта.

```
# Загружаем модуль ввода TCP и прослушиваем порт 514
# Количество клиентов не должно превышать 500
module(load="imtcp" MaxSessions="500")
input(type="imtcp" port="514")

# Сохраняем разные файлы в зависимости от типа сообщения
if $programname == 'apache-access' then {
   action(type="omfile" file="/var/log/site/apache/access.log")
} else if $programname == 'apache-error' then {
   action(type="omfile" file="/var/log/site/apache/error.log")
} else {
   # Все остальное направляется в общий журнальный файл системы Syslog
   action(type="omfile" file="/var/log/site/syslog")
}
```

🕮 Дополнительную информацию о конфигурациях см. в главе 23.

Центральный сервер журналирования генерирует временную метку для каждого сообщения во время его записи. Сообщения сервера Apache содержат отдельную временную метку, которая генерируется, когда демон **httpd** регистрирует сообщение. Обе эти метки заносятся в общие журнальные файлы.

Сообщения защиты системы Syslog

Система Rsyslog может отправлять и получать сообщения журнала через TLS, уровень шифрования и аутентификации, работающий по протоколу TCP. Общую информацию об уровне TLS см. в разделе 27.6.

В приведенном ниже примере предполагается, что центр сертификации, общедоступные сертификаты и ключи уже созданы. Для получения подробной информации об инфраструктуре открытого ключа и создании сертификатов см. раздел 27.6.

Эта конфигурация представляет новую опцию: драйвер сетевого потока, модуль, работающий на уровне между сетью и системой Rsyslog. Он обычно реализует функции,

[6] Демон **httpd** может вести журнал непосредственно в системе Syslog с помощью модуля **mod_syslog**, но мы используем модуль **imfile** для иллюстрации.

которые улучшают основные сетевые возможности. Уровень TLS подключается драйвером потока `gtls`.

Следующий пример позволяет использовать драйвер `gtls` для журнального сервера. Для драйвера `gtls` требуется сертификат CA, открытый сертификат и закрытый ключ сервера. Затем модуль `imtcp` включает драйвер потока `gtls`.

```
global(
    defaultNetstreamDriver="gtls"
    defaultNetstreamDriverCAFile="/etc/ssl/ca/admin.com.pem"
    defaultNetstreamDriverCertFile="/etc/ssl/certs/server.com.pem"
    defaultNetstreamDriverKeyFile="/etc/ssl/private/server.com.key"
)
module(
    load="imtcp"
    streamDriver.name="gtls"
    streamDriver.mode="1"
    streamDriver.authMode="x509/name"
)
input(type="imtcp" port="6514")
```

Сервер журнала прослушивает TLS-версию стандартного порта системы Syslog, 6514. Параметр `authMode` сообщает системе Syslog, какой тип проверки должен выполняться. Параметр `x509/name`, заданный по умолчанию, подразумевает проверку того факта, что сертификат подписан доверенным органом, а также проверяет имя субъекта, которое связывает сертификат с конкретным клиентом через систему DNS.

Конфигурация для клиентской стороны TLS-соединения аналогична. Используйте сертификат клиента, закрытый ключ и драйвер потока **`gtls`** для пересылки журналов с помощью модуля вывода.

```
global(
    defaultNetstreamDriver="gtls"
    defaultNetstreamDriverCAFile="/etc/ssl/ca/admin.com.pem"
    defaultNetstreamDriverCertFile="/etc/ssl/certs/client.com.pem"
    defaultNetstreamDriverKeyFile="/etc/ssl/private/client.com.key"
)
*.*     action(type="omfwd"
            Protocol="tcp"
            Target="logs.admin.com"
            Port="6514"
            StreamDriverMode="1"
            StreamDriver="gtls"
            StreamDriverAuthMode="x509/name"
        )
```

В данном случае мы пересылаем все журнальные сообщения с помощью довольно искусственного синтаксиса демона **`sysklogd`**: компонент действия имеет форму синтаксиса RainerScript, а не стандартного синтаксиса демона **`sysklogd`**. Если вы хотите выбирать, какие сообщения пересылать (или вам нужно отправлять разные классы сообщений в разные адреса), вы можете использовать выражения фильтра RainerScript, как показано в нескольких примерах ранее в этой главе.

Отладка системы Syslog

Команда **`logger`** позволяет выдавать журнальные сообщения в сценариях интерпретатора команд. Кроме того, эту команду можно использовать для проверки изменений

в конфигурационном файле системы Rsysog. Например, если в файл была добавлена строка

```
local5.warning                                  /tmp/evi.log
```

и нужно убедиться в том, что она работает, введите такую команду.

```
$ logger -p local5.warning "test message"
```

Строка, содержащая фразу `test message` (тестовое сообщение), должна быть записана в файл **`/tmp/evi.log`**. Если этого не произошло, то, возможно, вы забыли перезапустить демон **`rsyslog`**?

10.4. Журнальная регистрация на уровне ядра и на этапе начальной загрузки

Ядро и сценарии запуска системы представляют собой немалую проблему с точки зрения журнальной регистрации. В первом случае проблема заключается в том, чтобы средства регистрации событий, происходящих на этапе начальной загрузки и в ядре, не зависели от типа файловой системы и ее организации. Во втором случае трудность состоит в последовательной регистрации событий, происходящих в сценариях, а не в системных демонах или других программах, — их сообщения должны регистрироваться на другом уровне. Необходимо также избежать ненужного дублирования записей в сценариях и переадресации их вывода.

Первая из описанных проблем решается путем создания внутреннего буфера ограниченного размера, куда ядро заносит свои журнальные сообщения. Этот буфер достаточно велик для того, чтобы вместить сообщения обо всех действиях, предпринимаемых ядром на этапе начальной загрузки. Когда система полностью загрузится, пользовательский процесс сможет получить доступ к журнальному буферу ядра и скопировать его содержимое в нужное место.



В системах Linux демон **`systemd-journald`** извлекает сообщения ядра из буфера ядра, считывая файл устройства **`/dev/kmsg`**. Эти сообщения можно увидеть, выполнив команду **`journalctl -k`** или ее псевдоним, **`journalctl --dmesg`**. Кроме того, можно использовать традиционную команду **`dmesg`**.



В системе FreeBSD и старых версиях Linux лучшим средством для просмотра буфера ядра является команда **`dmesg`**; ее вывод содержит даже сообщения, которые были сгенерированы до начала выполнения процесса **`init`**.

Другой важный момент, связанный с журнальной регистрацией на уровне ядра, касается правильного управления системной консолью. Важно, чтобы по ходу загрузки системы вся выводимая информация попадала на консоль. С другой стороны, когда система загрузилась и начала выполняться, консольные сообщения больше раздражают, нежели помогают, особенно если консоль используется для регистрации в системе.

В системе Linux команда **`dmesg`** позволяет указывать в командной строке уровень важности консольных сообщений ядра.

```
ubuntu$ sudo dmesg -n 2
```

Сообщения уровня 7 наиболее информативны и включают отладочную информацию. К первому уровню относятся только сообщения о фатальных ошибках системы (чем ниже уровень, тем выше важность сообщений). Все сообщения, выдаваемые ядром,

продолжают поступать в журнальный буфер (а также в систему Syslog) независимо от того, направляются они на консоль или нет.

10.5. Управление журнальными файлами и их ротация

Утилита **logrotate**, разработанная Эриком Троаном (Erik Troan), реализует различные схемы управления журнальными файлами и является частью всех рассматриваемых нами дистрибутивов Linux. Она также выполняется в системе FreeBSD, но для этого вам придется инсталлировать ее на нескольких портах. По умолчанию система FreeBSD использует другой пакет ротации журналов под названием **newsyslog**; его подробное описание можно найти немного ниже.

Утилита logrotate: кросс-платформенное управление журналами

Конфигурация утилиты **logrotate** состоит из набора спецификаций для групп журнальных файлов, которыми следует управлять. Параметры, определенные вне какой-либо спецификации (например, `errors`, `rotate` и `weekly` в показанном ниже примере), являются глобальными. Их можно переопределять для конкретного журнального файла, а также указывать несколько раз в различных частях файла, меняя стандартные установки.

Ниже представлен несколько искусственный пример такого файла.

```
# Глобальные параметры
errors errors@book.admin.com
rotate 5
weekly

# Определения и параметры ротации журнальных файлов
/var/log/messages {
   postrotate
       /bin/kill -HUP `cat /var/run/syslogd.pid`
   endscript
}

/var/log/samba/*.log {
   notifempty
   copytruncate
   sharedscripts
   postrotate
       /bin/kill -HUP `cat /var/lock/samba/*.pid`
   endscript
}
```

В этой конфигурации раз в неделю осуществляется ротация файла **/var/log/messages**. Сохраняются пять последовательных версий файла, а демон **syslogd** уведомляется о каждом шаге ротации. Журнальные файлы системы Samba (их может быть несколько) также подвергаются ротации каждую неделю, но вместо того чтобы последовательно перемещать файлы, а затем создавать новый начальный файл, сценарий последовательно копирует журнальные файлы, усекая первый. Сигнал `HUP` посылается демонам Samba только после того, как произойдет ротация всех журнальных файлов.

Наиболее полезные параметры файла **logrotate.conf** перечислены в табл. 10.8.

Таблица 10.8. Параметры утилиты `logrotate`

Опция	Назначение
`compress`	Сжимать все версии журнального файла, кроме текущей
`daily, weekly, monthly`	Осуществлять ротацию журнальных файлов ежедневно, еженедельно или ежемесячно
`delaycompress`	Сжимать все версии журнального файла, кроме текущей и предыдущей
`endscript`	Этот параметр помечает конец блоков **`prerotate`** и **`postrotate`**
`errors` *почтовый_адрес*	Направлять сообщения об ошибках по указанному адресу
`missingok`	Не выдавать предупреждений об отсутствии журнального файла
`notifempty`	Не осуществлять ротацию журнального файла, если он пуст
`olddir` *каталог*	Помещать старые версии журнального файла в указанный каталог
`postrotate`	Этот параметр помечает начало блока команд, выполняемых после ротации журнального файла
`prerotate`	Этот параметр помечает начало блока команд, выполняемых перед ротацией журнального файла
`rotate` *n*	Включать в цикл ротации указанное число версий журнального файла
`sharedscripts`	Запускать сценарии один раз для всей группы журнальных файлов
`size` *порог*	Выполнять ротацию, если размер журнального файла превышает порог (например, 100 Кбайт, 4 Мбайт)

Утилита **`logrotate`** обычно запускается демоном **`cron`** раз в день. Ее стандартный конфигурационный файл называется **`/etc/logrotate.conf`**, но в командной строке можно указывать сразу несколько файлов (или каталогов, содержащих конфигурационные файлы).

Эта особенность широко используется в дистрибутивах Linux, в которых каталог **`/etc/logrotate.d`** определен как стандартное место хранения конфигурационных файлов утилиты **`logrotate`**. Программные пакеты, поддерживающие эту утилиту (а таковых немало), при инсталляции выбирают схему журнальной регистрации, что существенно упрощает их администрирование.

Параметр `delaycompress` заслуживает дополнительных разъяснений. Некоторые приложения даже после ротации некоторое время продолжают записывать сообщения в предыдущий журнальный файл. Параметр `delaycompress` предназначен для того, чтобы отложить сжатие на один цикл ротации. Этот параметр приводит к возникновению трех типов журнальных файлов: активный журнальный файл, файл с предыдущей ротации, но еще не сжатый, и сжатый файл после ротации.



Кроме утилиты **`logrotate`**, дистрибутив Ubuntu включает также более простую программу **`savelog`**, которая управляет ротацией отдельных файлов. Она проще утилиты **`logrotate`** и не использует конфигурационный файл (да и не нуждается в нем). В некоторых пакетах предпочтение отдается собственным конфигурациям, создаваемым с помощью утилиты **`savelog`**, а не определенным утилитой **`logrotate`**.

Утилита `newsyslog`: управление журналами в системе FreeBSD

Имя **`newsyslog`** может ввести в заблуждение. Утилита была названа так потому, что изначально была предназначена для ротации файлов, управляемых системой Syslog. В системе FreeBSD она является эквивалентом утилиты **`logrotate`**. Ее синтаксис и ре-

ализация принципиально отличаются от синтаксиса и реализации утилиты **logrotate**, но, если не учитывать несколько непривычный формат даты, синтаксис утилиты **newsyslog** в чем-то даже проще.

Основным файлом конфигурации является файл **/etc/newsyslog.conf**. Формат и синтаксис утилиты **newsyslog** описаны на справочной странице **man newsyslog**. По умолчанию файл **/etc/newsyslog.conf** содержит примеры стандартных журнальных файлов.

Как и **logrotate**, утилита **newsyslog** запускается демоном **cron**. В простейшей конфигурации системы FreeBSD файл **/etc/crontab** содержит строку, которая запускает утилиту **newsyslog** каждый час.

10.6. Управление журналами в крупном масштабе

Одно дело — записывать журнальные сообщения, хранить их на диске и пересылать на центральный сервер. Другое дело — обрабатывать данные регистрации сотен или тысяч серверов. Объемы сообщений просто слишком велики для эффективного управления без инструментов, предназначенных для работы в этом масштабе. К счастью, для решения этой проблемы доступны несколько коммерческих и открытых инструментов.

Стек ELK

Явным лидером в области программного обеспечения с открытым исходным кодом — и действительно одним из лучших пакетов программного обеспечения, с которыми мы с удовольствием работаем, — является мощный стек ELK, состоящий из инструментов Elasticsearch, Logstash и Kibana. Эта комбинация инструментов позволяет сортировать, искать, анализировать и визуализировать большие объемы журнальных данных, создаваемых глобальной сетью клиентов регистрации. Стек ELK создан компанией Elastic (`elastic.co`), которая также предлагает обучение, поддержку и корпоративные надстройки для ELK.

Elasticsearch — это масштабируемая база данных и поисковая система с интерфейсом прикладного программирования RESTful для запроса данных. Она написана на языке Java. Установки Elasticsearch могут варьироваться от одного хоста, который обрабатывает низкий объем данных, до нескольких десятков хостов в кластере, который индексирует много тысяч событий каждую секунду. Поиск и анализ журнальных данных является одним из самых популярных приложений для Elasticsearch.

Если поисковая система Elasticsearch является основным инструментом стека ELK, то конвейер обработки данных Logstash является его союзником и доверенным партнером. Конвейер Logstash принимает данные из многих источников, включая системы массового обслуживания, такие как RabbitMQ и AWS SQS. Он также может считывать данные непосредственно из сокетов TCP или UDP и традиционного протокола системы Syslog. Конвейер Logstash может анализировать сообщения для добавления дополнительных структурированных полей и отфильтровывать нежелательные или несоответствующие данные. Когда сообщения попадают в сеть, Logstash может записывать их в самые разные пункты назначения, включая, конечно, Elasticsearch.

Вы можете отправлять записи журнала в конвейер Logstash различными способами. Вы можете настроить вход конвейера Logstash на систему Syslog и использовать модуль вывода **omfwd** системы Rsyslog, как описано в конфигурации Rsyslog в разделе 10.3. Вы также можете использовать выделенный отправитель журналов. Собственная версия

Elastic называется Filebeat и может отправлять журналы либо в Logstash, либо непосредственно в Elasticsearch.

Последний компонент ELK, Kibana, является графическим интерфейсом для Elasticsearch. Он предоставляет интерфейс поиска, с помощью которого можно найти записи, которые вам нужны, среди всех данных, которые были проиндексированы поисковой системой Elasticsearch. Интерфейс Kibana может создавать графики и визуализации, которые помогают генерировать новые сведения о ваших приложениях. Можно, например, отобразить события журнала на карте, чтобы увидеть географически то, что происходит с вашими системами. Другие дополнительные модули добавляют оповещения и интерфейсы мониторинга системы.

Разумеется, ELK не обходится без эксплуатационной нагрузки. Построение крупномасштабного ELK-стека с настраиваемой конфигурацией — непростая задача, и управление ею требует времени и опыта. Большинству администраторов, которых мы знаем (включая нас!), приходилось случайно терять данные из-за ошибок в программном обеспечении или при его эксплуатации. Если вы решите развернуть ELK, имейте в виду, что вы соглашаетесь на значительные административные издержки.

Нам известна по крайней мере одна служба, `logz.io`, предлагающая услугу ELK на уровне производства. Вы можете отправлять сообщения журнала из своей сети по зашифрованным каналам в конечную точку, которую предоставляет `logz.io`. Там сообщения регистрируются, индексируются и становятся доступными через интерфейс Kibana. Это довольно дорогое решение, но его стоит оценить. Как и во многих облачных службах, вы можете обнаружить, что в конечном итоге реплицировать службу локально обойдется дороже.

Graylog

Graylog — аутсайдер по сравнению с пакетом ELK. Он похож на стек ELK: он тоже хранит данные в базе данных Elasticsearch и может принимать сообщения журнала напрямую или через конвейер Logstash, как и в стеке ELK. Реальным отличием является пользовательский интерфейс Graylog, который многие пользователи считают более удачным и простым в использовании.

Некоторые из платных продуктов ELK реализованы в Graylog как программы с открытым исходным кодом, включая поддержку ролевого контроля доступа и интеграцию с LDAP.

Система Graylog, безусловно, заслуживает внимания, когда вы выбираете новую инфраструктуру журналирования.

🕮 Дополнительную информацию о RBAC и LDAP см. в разделах 3.4 и 17.2.

Журналирование как услуга

Существует несколько коммерческих предложений по управлению журналами. Самым зрелым и надежным инструментом является Splunk; имеющий как версии, развернутые на виртуальном хосте, так и локальные версии. Некоторые из крупнейших корпоративных сетей используют Splunk не только как менеджер журналов, но и как систему бизнес-аналитики. Но если вы выберете Splunk, будьте готовы заплатить за эту привилегию.

Альтернативные опции SaaS включают Sumo Logic, Loggly и Papertrail, имеющие встроенную интеграцию системы Syslog и разумный интерфейс поиска. Если вы используете AWS, то служба Amazon CloudWatch Logs сможет собирать данные журналов как из служб AWS, так и из ваших собственных приложений.

10.7. Принципы обработки журнальных файлов

За прошедшие годы регистрационные события вышли из области системного администрирования и превратились в сложно решаемые проблемы в сфере управления событиями на уровне предприятий. Обработка событий, связанных с безопасностью, стандарты информационных технологий и законодательные акты могут потребовать целостного и систематического подхода к управлению данными регистрации.

Данные журналов регистрации от одной системы создают относительно небольшую нагрузку на устройства хранения, но ведение централизованного журнала регистрации событий от сотен серверов и десятков приложений — совсем другая история. В основном благодаря критически важной природе веб-служб журналы регистрации, создаваемые приложениями и демонами, по значимости вышли на уровень журналов регистрации, генерируемых операционными системами.

Разрабатывая стратегию обработки журнальных файлов, ответьте на следующие вопросы.

- Сколько систем и приложений вы предполагаете использовать?
- Инфраструктура хранения информации какого типа вам доступна?
- Как долго вы собираетесь хранить журналы регистрации?
- Какого типа события являются для вас значимыми?

Ответы на эти вопросы зависят от требований, предъявляемых к бизнесу, и от применяемых вами стандартов или нормативных документов. Например, один стандарт от Совета по стандартам безопасности индустрии платежных карт (Payment Card Industry Security Standards Council — PCI SSC) требует, чтобы журналы регистрации хранились на носителе с простым доступом к данным (например, это может быть локально монтируемый жесткий диск) в течение трех месяцев и архивировались для долговременного хранения в течение по крайней мере одного года. Кроме того, этот же стандарт включает конкретные требования к типам используемых данных.

Конечно, как заметил один из наших рецензентов, нельзя затребовать в суде данные журнала, которых у вас нет. По этой причине некоторые организации не собирают (или намеренно уничтожают) конфиденциальные данные журнала. Вы можете использовать этот подход или нет в зависимости от юридических норм, которые к вам применяются.

Как бы вы ни ответили на поставленные выше вопросы, обязательно соберите входную информацию из отделов по соблюдению правил и информационной безопасности, если они есть в вашей организации.

Для большинства приложений обычно рассматривают сбор как минимум следующей информации:

- имя пользователя или его идентификатор (ID);
- результат обработки события (успешная или нет);
- адрес источника для сетевых событий;
- дата и время (из такого официального источника, как NTP);
- оповещение о добавлении, изменении или удалении важных данных;
- подробные данные о событии.

🕮 Дополнительную информацию о RAID см. в разделе 20.8.

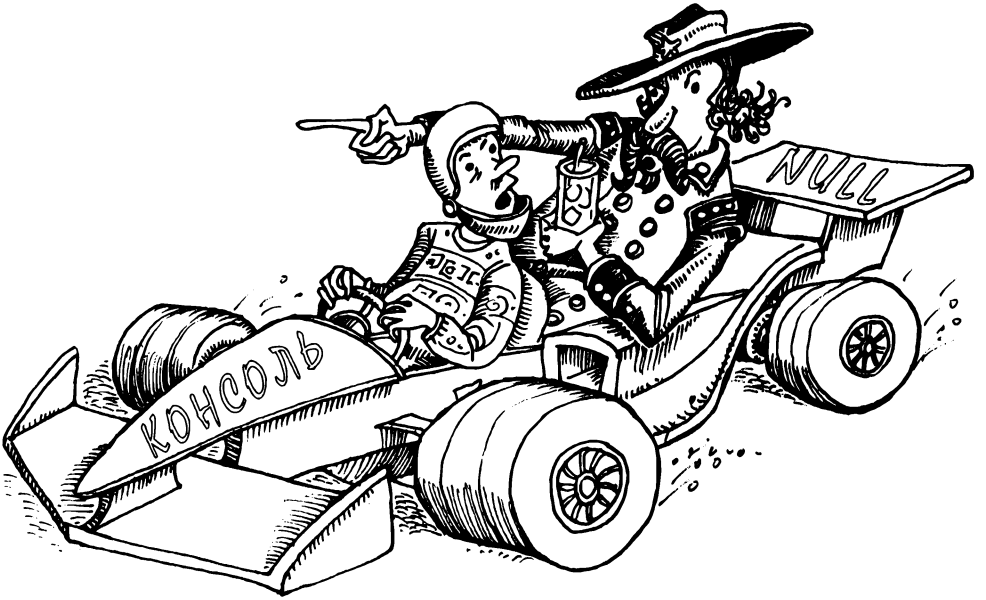
Сервер журналирования должен работать в соответствии с тщательно разработанной стратегией хранения информации. Например, облачные системы предлагают немедленный доступ к данным, собранным за последние 90 дней, службы объектных хранилищ — за год, наконец, архивные службы — за три года. Требования к хранению данных могут со временем меняться, и реализация может легко и успешно адаптироваться к изменению этих условий.

Доступ к централизированным серверам журналирования должен быть ограничен системными администраторами высокого уровня, программами и персоналом, задействованным в решении вопросов безопасности и адресной совместимости. В рамках организаций эти системы обычно не играют главную роль при принятии ежедневных решений за рамками удовлетворения требований к возможности проведения аудита, поэтому администраторы приложений, конечные пользователи и служба технической поддержки не имеют доступа к ним. Доступ к журнальным файлам на центральных серверах должен регистрироваться отдельно.

Организация централизованного журналирования требует определенных усилий, и в небольших системах она может не продемонстрировать преимущества от использования сети. Для обеспечения централизации мы предлагаем в качестве разумной пороговой величины использовать двадцать серверов. При меньшем количестве необходимо гарантировать, что журналы регистрации подвергаются надлежащей ротации и архивированию с частотой, достаточной для того, чтобы избежать заполнения диска. Включите журнальные файлы в систему мониторинга, чтобы получать извещения, если некоторый журнальный файл перестанет увеличиваться в размере.

Глава 11

Драйверы и ядро



Ядро — это главная часть операционных систем UNIX и Linux. Оно устанавливает правила, распределяет ресурсы и предоставляет пользователям основные услуги.

Функции ядра воспринимаются пользователями как нечто само собой разумеющееся. Это хорошо, потому что такие простые команды, как **`cat /etc/passwd`** на самом деле скрывают сложные последовательности базовых действий. Например, управляя самолетом, намного удобнее оперировать командами "подняться на высоту 35000 футов", чем тысячами мелких шагов, которые необходимо сделать, чтобы выполнить эту команду.

Ядро скрывает детали устройства аппаратного обеспечения системы за абстрактным высокоуровневым интерфейсом. Он очень похож на интерфейс прикладного программирования, обеспечивающий полезные функции взаимодействия с системой. В частности, этот интерфейс обеспечивает пять основных функций.

- Управление и абстрагирование аппаратного обеспечения.
- Процессы и потоки, а также способы взаимодействия между ними,
- Управление памятью (виртуальная память и защита областей памяти).
- Функции ввода-вывода (файловые системы, сетевые интерфейсы, последовательные интерфейсы и т.д.).
- Организационные функции (начало и завершение работы, таймеры, многозадачность и т.д.).

Только драйверы устройств знают о конкретных возможностях и протоколах связи аппаратного обеспечения системы. Пользовательские программы и остальная часть ядра в значительной степени не зависят от этих знаний. Например, файловая система на диске сильно отличается от сетевой файловой системы, но уровень VFS ядра делает их похожими на пользовательские процессы и на другие части ядра. Вам не нужно знать, направляются ли данные, которые вы записываете, в блок 3829 дискового устройства №8 или же он направляется на Ethernet-интерфейс e1000e, упакованный в TCP-пакет. Все, что вам нужно знать, это эти данные будут направлены в указанный вами файловый дескриптор.

Процессы (и потоки, их упрощенные аналоги) — это механизмы, благодаря которым ядро организовывает совместное использование центрального процессора и защиту памяти. Ядро плавно переключается между процессами системы, выделяя каждому выполняемому потоку небольшой отрезок времени, в котором необходимо выполнить работу. Ядро запрещает процессам читать и записывать данные в памяти друг друга, если у них нет явного разрешения на это.

Система управления памятью определяет адресное пространство для каждого процесса и создает иллюзию, что процессу принадлежит практически неограниченная область смежной памяти. На самом деле страницы памяти разных процессов смешаны в физической памяти системы. Их упорядочивают только схемы хранения и защиты памяти ядра.

Устройства ввода-вывода образуют уровень, расположенный выше драйверов устройств, но ниже большинства других частей ядра. Они состоят из служб файловой системы, сетевой подсистемы и различных других служб, которые используются для обмена данными с системой.

11.1. Ядра и системное администрирование

Почти все многоуровневые функции ядра написаны на языке C с небольшими вставками кода на языке ассемблера для обеспечения доступа к функциям процессора, которые недоступны с помощью директив компилятора языка C (например, атомарные инструкции чтения, модификации и записи, определенные многими центральными процессорами). К счастью, вы можете быть совершенно эффективным системным администратором, не будучи программистом на языке C и не касаясь кода ядра.

Тем не менее неизбежно, что в какой-то момент вам нужно будет немного поработать. Эта работа может принимать несколько форм.

Многие из задач ядра (например, переадресация сетевых пакетов) зависят от параметров настройки, доступных из пользовательского пространства. Установка этих значений для вашей среды и рабочей нагрузки является общей административной задачей.

Другой общей задачей, связанной с ядром, является установка новых драйверов устройств. На рынке постоянно появляются новые модели и типы оборудования (видеокарты, беспроводные устройства, специализированные звуковые карты и т.д.), а ядра, поставляемые поставщиками, не всегда способны их использовать.

В некоторых случаях вам может понадобиться также создать новую версию ядра из исходного кода. Системным администраторам не нужно строить ядра так часто, как раньше, но в некоторых ситуациях это имеет смысл. Это проще, чем кажется.

Ядра сложны. На удивление легко дестабилизировать ядро с помощью даже незначительных изменений. Даже если ядро просто загружается, оно может работать не так, как должно. Что еще хуже, вы не поймете, что производительность снизилась, если у вас нет структурированного плана оценки результатов вашей работы. Будьте консервативны, изменяя ядра, особенно в производственных системах, и всегда имейте план резервного копирования для возврата к хорошо известной конфигурации.

11.2. Нумерация версий ядра

Прежде чем мы погрузимся в глубины ядер, стоит потратить несколько слов, чтобы обсудить версии ядра и их связь с дистрибутивами.

Активная разработка ядер Linux и FreeBSD не прекращается ни на день. Со временем исправляются дефекты, добавляются новые функции и удаляются устаревшие функции.

Некоторые старые ядра продолжают поддерживаться в течение длительного периода времени. Аналогично некоторые дистрибутивы предпочитают подчеркивать стабильность и, следовательно, запускают более старые, более проверенные ядра. Другие дистрибутивы пытаются предложить самую последнюю поддержку и функции устройства, но в результате могут быть немного менее стабильными. Администратор должен выбрать один из этих вариантов таким образом, чтобы он соответствовал потребностям ваших пользователей. Ни одно решение не подходит для любого контекста.

Версии ядер для системы Linux



Ядро Linux и дистрибутивы, основанные на нем, разрабатываются отдельно от другого, поэтому у ядра есть своя схема управления версиями. Некоторые выпуски ядра достигают известной популярности, поэтому нет ничего необычного в том, что несколько независимых дистрибутивов используют одно и то же ядро. Чтобы узнать, в каком ядре работает данная система, выполните команду `uname -r`.

Ядра Linux называются в соответствии с правилами так называемого семантического управления версиями, т.е. они включают в себя три компонента: основную версию, второстепенную версию и уровень заплатки. В настоящее время не существует предсказуемой связи между номером версии и ее предполагаемым статусом как стабильным или экспериментальным; ядра называются стабильными, когда разработчики решают, что они стабильны. Кроме того, основной номер версии ядра исторически устанавливался по несколько причудливым правилам.

Многие стабильные версии ядра Linux могут одновременно находиться в долгосрочной эксплуатации. Ядра, поставляемые вместе с крупными дистрибутивами Linux, часто значительно отстают от последних выпусков. В некоторых дистрибутивах даже есть ядра, которые официально устарели.

Вы можете установить новые ядра путем компиляции и установки из исходного дерева ядра. Однако мы не рекомендуем вам это делать. Различные дистрибутивы имеют разные цели, и они выбирают версии ядра, соответствующие этим целям. Вы никогда не знаете, что дистрибутив не выбирает новое ядро из-за какой-то конкретной проблемы. Если вам нужно более новое ядро, установите дистрибутив, созданный вокруг этого ядра, вместо того, чтобы пытаться внедрить новое ядро в существующую систему.

🕮 Дополнительную информацию о семантическом управлении версиями см. на сайте `semver.org`.

Версии ядер FreeBSD



Система FreeBSD использует довольно простой подход к версиям и выпускам. Проект поддерживает две основные производственные версии, которые на момент выхода русского издания книги являются версиями 12 и 13. Ядро не имеет отдельной схемы управления версиями; оно выпускается как часть полной операционной системы и имеет тот же номер версии.

Более старый из двух основных выпусков (в данном случае FreeBSD 12) можно рассматривать как текущую версию. Он не содержит новых функций и делает упор на стабильности и безопасности.

Более поздняя версия (FreeBSD 13) прямо сейчас находится на стадии активной разработки. Она также имеет стабильные версии, предназначенные для общего использования, но код этого ядра всегда будет более новым и несколько менее проверенным по сравнению с предыдущей основной версией.

Как правило, подверсии (dot version) появляются примерно каждые четыре месяца. Основные выпуски явно поддерживаются в течение пяти лет, а подверсии внутри них поддерживаются в течение трех месяцев после выхода следующей подверсии. Полдверсии быстро устаревают; система FreeBSD предполагает, что пользователи будут обновлять ее с помощью заплаток.

11.3. Устройства и их драйверы

Драйвер устройства — это уровень абстракции, который управляет взаимодействием системы с конкретным типом аппаратного обеспечения, так что остальная часть ядра не должна знать ее специфику. Драйвер выполняет перевод аппаратных команд, понимаемых устройством, и стилизованным программным интерфейсом, определенным (и используемым) ядром. Уровень драйвера помогает сохранить большую часть независимого от ядра устройства.

Учитывая быстрые темпы разработки нового оборудования, практически невозможно постоянно обновлять дистрибутивы основной операционной системы, чтобы иметь возможность использовать новейшее оборудование. Следовательно, вам иногда потребуется добавить драйвер устройства в свою систему для поддержки нового оборудования.

В каждой системе можно инсталлировать только "свои" драйверы, причем зачастую только те, которые разработаны специально для конкретной версии ядра. Драйверы для других операционных систем (например, Windows) работать не будут. Об этом следует помнить, покупая новое оборудование. Кроме того, различные устройства в различной степени совместимы с дистрибутивами Linux и при работе в ее среде могут предоставлять различные функциональные возможности. Поэтому, выбирая оборудование, следует учитывать опыт его применения в других организациях.

Изготовители оборудования обращают все больше внимания на рынки FreeBSD и Linux и часто выпускают драйверы на своей продукции. В оптимальном варианте к устройству прилагаются и драйвер, и инструкции по его инсталляции. Однако с наибольшей вероятностью нужный драйвер можно найти на каком-нибудь неофициальном веб-сайте. Будьте осмотрительными.

Файлы и номера устройств

Во многих случаях драйверы устройств являются частью ядра, а не пользовательским процессом. Однако получить доступ к драйверу можно как из ядра, так и из пространства пользователя. Как правило, для этого используются файлы устройств, записанные в каталоге **/dev**. Ядро отображает операции над этими файлами в вызовы кода драйвера.

Для многих несетевых устройств в каталоге **/dev** имеются один или несколько соответствующих файлов. Сложные серверы могут поддерживать сотни различных устройств. С каждым файлом устройства в каталоге **/dev** связаны старший и младший номера устройства. Посредством этих номеров ядро преобразует обращения к файлу устройства в вызовы нужного драйвера.

Старший номер устройства обозначает драйвер, за которым закреплен данный файл (иначе говоря, он определяет тип устройства). Младший номер устройства обычно указывает на то, к какому конкретно устройству данного типа следует обращаться. Младший номер иногда называют номером модуля.

Узнать номера устройств позволяет команда **ls -l**.

```
linux$ ls -l /dev/sda
brw-rw----  1 root disk 8, 0 Jul 13 01:38 /dev/sda
```

В этом примере выдана информация о первом диске SCSI/SATA/SAS в системе Linux. Его старший номер равен 8, а младший — 0.

Младший номер иногда используется драйвером для выбора или активации определенных функций (характеристик) устройства. Например, накопителю на магнитной ленте могут соответствовать два файла в каталоге **/dev**, один из которых при закрытии будет автоматически обеспечивать обратную перемотку, а второй — нет. Драйвер волен интерпретировать младший номер устройства по своему усмотрению. Особенности работы драйвера можно узнать на соответствующей справочной странице.

Файлы устройств бывают двух типов: блочные и символьные. Чтение из блочного устройства и запись в него осуществляются по одному блоку за раз (блок — это группа байтов, размер которой обычно кратен 512), тогда как чтение из символьного устройства и запись в него происходят в побайтовом режиме. Символ `b` в выводе команды **ls** означает, что **/dev/sda** является блочным устройством. Если бы это устройство было символьным, то первым символом был бы символ `c`.

В прошлом определенные устройства могли действовать и как блочные, и как символьные, и существовали отдельные файлы устройств, чтобы сделать их доступными в любом режиме. Диски и ленты "жили двойной жизнью", но большинство других устройств этого не делали. Однако эта система параллельного доступа больше не используется. Все устройства, которые ранее работали в двух режимах, система FreeBSD теперь представляет в качестве символьных устройств, а Linux представляет их как блочные устройства.

Определенные системные функции удобно реализовывать, используя драйвер устройства, даже если связанное с ним устройство не существует. Такие фантомные устройства называют *псевдоустройствами*. Например, пользователю, зарегистрировавшемуся в системе по сети, назначается псевдотерминал (PTY), который с точки зрения высокоуровневого программного обеспечения ничем не отличается от обычного последовательного порта. Благодаря такому подходу программы, написанные в эпоху алфавитно-цифровых терминалов, могут функционировать в мире окон и сетей. Вот еще примеры псевдоустройств: **/dev/zero**, **/dev/null** и **/dev/urandom**.

Когда программа выполняет операцию над файлом устройства, ядро перехватывает обращение к файлу, ищет в таблице переходов соответствующую функцию и передает управление соответствующей части драйвера. Для выполнения необычных операций, не имеющих прямых аналогов в модели файловой системы (например, выталкивание дискеты из дисковода), используется системный вызов **ioctl**, который передает пользовательскую команду непосредственно драйверу. Типы запросов стандартной команды **ioctl** регистрируются в органе центральной авторизации аналогично номерам сетевых протоколов, используемых функцией управления пространствами IP-адресов IANA.

Система FreeBSD продолжает использовать традиционную систему **ioctl**. Традиционные устройства в системе Linux также применяют команду **ioctl**, но в современном коде сетевых устройств используется более гибкая система сокетов Netlink, описанная в спецификации RFC3549. Эти сокеты обеспечивают более гибкую систему сообщений, чем команда **ioctl**, потому что для них не требуется орган центральной авторизации.

Проблемы управления файлами устройств

Файлы устройств были сложной проблемой в течение многих лет. Когда системы поддерживали только несколько типов устройств, ручное обслуживание файлов устройств было управляемым. Однако, когда число доступных устройств выросло, файловая система **/dev** стала запутанной и часто содержала файлы, не имеющие отношения к текущей системе. В системе Red Hat Enterprise Linux версии 3 было включено более 18000 файлов устройств, по одному на все возможные устройства, которые можно было подключить к системе! Создание статических файлов устройств быстро стало огромной проблемой и эволюционным тупиком.

USB, FireWire, Thunderbolt и другие интерфейсы устройств вносят дополнительные сложности. В идеале диск, который первоначально распознается как **/dev/sda**, останется доступным как **/dev/sda**, несмотря на периодические отключения и независимо от активности других устройств и шин. Наличие других подключаемых устройств, таких как камеры, принтеры и сканеры (не говоря уже о других типах съемных носителей), усложняет ситуацию и еще больше ухудшает проблему надежной идентификации.

Сетевые интерфейсы имеют такую же проблему; они являются устройствами, но не имеют файлов устройств для их представления в каталоге **/dev**. Для этих устройств современным подходом является использование относительно простой системы прогнозируемых сетевых интерфейсов (Predictable Network Interface Names), которая назначает имена интерфейсов, сохраняемых при перезагрузках, модернизации оборудования и изменении драйверов. Современные системы теперь имеют аналогичные методы для работы с именами других устройств.

Создание файлов устройств

Файлы устройств можно создавать вручную. Однако в настоящее время сложно встретить людей, создающих файлы устройств с помощью команды **mknod**, которая имеет следующий синтаксис.

mknod *имя_файла тип старший младший*

Здесь *имя_файла* — это создаваемый файл устройства, *тип* — тип устройства (**c** — в случае символьного устройства и **b** — в случае блочного), а *старший* и *младший* — старший и младший номера устройства соответственно. Если вы вручную создаете файл устройства, драйвер которого уже имеется в ядре, просмотрите соответствующую этому драйверу man-страницу и найдите указанные в ней старший и младший номера.

Управление современными файловыми системами

Системы Linux и FreeBSD автоматизируют управление файлами устройств. В классическом стиле UNIX системы более или менее одинаковы по своей концепции, но полностью разделены по своим реализациям и форматам файлов конфигурации. Пусть расцветает тысяча цветов!

Если обнаружено новое устройство, обе системы автоматически создают соответствующие файлы устройства. Если устройство отключается (например, USB-накопитель вынимается из порта USB), его файлы устройства удаляются. По архитектурным соображениям системы Linux и FreeBSD изолируют часть "создания файлов устройств" из этого уравнения.

В системе FreeBSD устройства создаются ядром в специальной файловой системе (**devfs**), которая монтируется в каталоге **/dev**. В системе Linux за это действие отвечает

демон **udev**, работающий в пользовательском пространстве. Обе системы прослушивают базовый поток событий, генерируемых ядром, которые сообщают о подключении и отключении устройств.

Тем не менее с недавно обнаруженным устройством нам часто хочется выполнить намного больше операций, чем просто создать для него файл устройства. Например, если он представляет собой часть съемных носителей, мы могли бы автоматически смонтировать его как файловую систему. Если это концентратор или коммуникационное устройство, то скорее всего мы должны будем настроить его с помощью соответствующей подсистемы ядра.

Как Linux, так и FreeBSD оставляют эти расширенные процедуры демонам пользовательского пространства: **udevd** в случае Linux и **devd** в случае FreeBSD. Основное концептуальное различие между двумя платформами заключается в том, что Linux концентрирует большую часть логики в демоне **udevd**, тогда как файловая система **devfs** в системе FreeBSD сама по себе немного настраивается.

В табл. 11.1 представлены компоненты систем управления файлами устройств на обеих платформах.

Таблица 11.1. Схема автоматического управления устройствами

Component	Linux	FreeBSD
Файловая система `/dev`	`udev/devtmpfs`	`devfs`
Файлы конфигурации файловой системы `/dev`	—	`/etc/devfs.conf` `/etc/devfs.rules`
Управляющий демон устройства	`udevd`	`devd`

Файлы конфигурации демона	`/etc/udev/udev.conf`
	`/etc/devfs.rules`
	`/lib/udev/rules.d`
		`/etc/udev/rules.d`

Автоматическое монтирование файловой системы	`udevd`	`autofs`

Управление устройствами в Linux

Администраторы Linux должны помнить, как работает система правил **udevd**, и знать, как использовать команду **udevadm**. Однако прежде чем вникать в эти детали, давайте сначала рассмотрим базовую технологию **sysfs**, хранилища информации об устройстве, из которого **udevd** получает свои необработанные данные.

Файловая система sysfs: доступ к "сердцу" устройств

Виртуальная файловая система **sysfs** была добавлена в ядро Linux в версии 2.6 и предоставляет хорошо организованную и подробную информацию обо всех доступных устройствах, их конфигурациях и состоянии. Доступ к драйверу возможен как из ядра, так и со стороны команд пользовательского уровня.

Для выяснения всей информации об устройстве (используемом им прерывании IRQ, количестве блоков, поставленных в очередь на запись контроллером диска, и т.п.) можно исследовать каталог **/sys**, в котором обычно монтируется система **sysfs**. Один из основополагающих принципов формирования этой файловой системы состоит в том, что каждый файл в каталоге **/sys** должен представлять только один атрибут связанного с ним устройства. Это соглашение обеспечивает определенное структурирование в остальном хаотичного набора данных.

В табл. 11.2 перечислены каталоги, включенные в корневой каталог **/sys**, каждый из которых является подсистемой, регистрируемой с помощью **sysfs**. Состав этих каталогов зависит от дистрибутива.

Таблица 11.2. Подкаталоги каталога /sys

Каталог	Описание
`block`	Информация о блочных устройствах, например о жестких дисках
`bus`	Шины, известные ядру: PCI-E, SCSI, USB и др.
`class`	Дерево, организованное функциональными типами устройств, например звуковые и графические карты, устройства ввода и сетевые интерфейсы
`dev`	Информация об устройствах, разделенная между символьными и блочными устройствами
`devices`	Корректное представление (с точки зрения наследственности) всех обнаруженных устройств[a]
`firmware`	Интерфейсы с такими зависимыми от платформы подсистемами, как ACPI
`fs`	Каталог для хранения некоторых (но не всех) файловых систем, известных ядру
`kernel`	Значение таких внутренних характеристик ядра, как кеш и виртуальная память
`module`	Динамические модули, загруженные ядром
`power`	Некоторые сведения о состоянии производительности системы (обычно не используется)

[a] Например, звуковые и графические карты, устройства ввода и сетевые интерфейсы.

Первоначально информация о конфигурации устройств, если таковая существовала, хранилась в файловой системе **/proc**, унаследованной от операционной системы System V UNIX и развивавшейся несколько хаотически. В результате, она стала накапливать много лишней информации, не относящейся к конфигурации устройств. Хотя информация о процессах и ядре, связанная со временем выполнения, по-прежнему хранится в каталоге **/proc** для обеспечения обратной совместимости, предполагается, что со временем вся информация об устройствах будет перемещена в каталог **/sys**.

Команда udevadm: исследование устройств

Команда **udevadm** запрашивает информацию об устройствах, инициализирует события, управляет демоном **udevd** и отслеживает события менеджера **udev** и ядра. Системные администраторы используют ее, в основном, для создания и тестирования правил, которые описываются в следующем разделе.

Команда **udevadm** в качестве своего первого аргумента ожидает одну из шести команд: **info**, **trigger**, **settle**, **control**, **monitor** или **test**. Особый интерес для системных администраторов представляют две команды: **info**, которая выводит информацию, зависящую от конкретного устройства, и **control**, которая запускает и останавливает работу менеджера устройств **udev** или заставляет его перезагрузить свои файлы правил. Команда **monitor** отображает события по мере того, как они происходят в системе.

Следующая команда отображает все атрибуты демона **udev** для устройства **sdb**. Здесь результат выполнения команды представлен в усеченном виде, но в действительности он содержит список всех родительских устройств (например, шины USB), которые в дереве устройств являются предками устройства **sdb**.

```
linux$ udevadm info -a -n sdb
...
looking at device '/devices/pci0000:00/0000:00:11.0/0000:02:03.0/
     usb1/1-1/1-1:1.0/host6/target6:0:0/6:0:0:0/block/sdb':
     KERNEL=="sdb"
     SUBSYSTEM=="block"
     DRIVER==""
     ATTR{range}=="16"
```

```
    ATTR{ext_range}=="256"
    ATTR{removable}=="1"
    ATTR{ro}=="0"
    ATTR{size}=="1974271"
    ATTR{capability}=="53"
    ATTR{stat}==" 71 986 1561 860 1 0 1 12 0 592 872"
...
```

Все пути в результате выполнения команды **udevadm** (например, **/devices/pci0000:00/**...) приведены относительно каталога **/sys**.

Этот результат отформатирован так, чтобы вы могли при построении правил обращаться к менеджеру **udev**. Например, если бы запись **ATTR{size}=="1974271"** была уникальной для этого устройства, вы могли бы скопировать этот фрагмент в правило в качестве идентифицирующего критерия.

Узнать дополнительные возможности и синтаксис команды **udevadm** вы сможете, обратившись к ее справочной странице.

Создание правил и постоянных имен

Менеджер устройств **udev** опирается на набор правил, определяющих возможности управления устройствами и присваивания им имен. Стандартные правила хранятся в каталоге **/lib/udev/rules.d**, а локальные — в каталоге **/etc/udev/rules.d**. Вам не нужно редактировать или удалять стандартные правила — их можно игнорировать или переопределить путем создания нового файла стандартных правил с прежним именем в пользовательском каталоге правил.

Главным файлом конфигурации для менеджера **udev** является файл **/etc/udev/udev.conf**. Файлы **udev.conf** в наших примерах дистрибутивов содержат только комментарии, за исключением одной строки, которая позволяет выполнять регистрацию ошибок.

К сожалению, из-за политических разногласий между дистрибьюторами и разработчиками дистрибутивы плохо согласованы друг с другом. Многие имена файлов в каталоге стандартных правил не меняются при переходе от дистрибутива к дистрибутиву, в то время как содержимое этих файлов претерпевает существенные изменения.

Файлы правил именуются в соответствии с шаблоном *nn-описание*.**rules**, где *nn* — обычно двузначное число. Файлы обрабатываются в лексическом порядке, поэтому меньшие числа обрабатываются первыми. Файлы из двух каталогов правил объединяются до того, как демон менеджера udev, **udevd**, проанализирует их. Суффикс **.rules** обязателен — без него файлы игнорируются.

Правила задаются в таком формате.

условие, [*условие*, ...] *назначение* [,*назначение* ...]

Здесь элемент *условие* определяет ситуации, в которых должно применяться данное правило. Каждое выражение условия состоит из ключа, знака операции и значения. Например, в качестве потенциального компонента правила может использоваться условие **ATTR{size}=="1974271"** (см. пример выше), отбирающее все устройства, атрибут **size** которых в точности равен значению **1 974 271**.

Большинство ключей, участвующих в задании условия, связаны со свойствами устройств (которые демон **udevd** получает из файловой системы **/sys**), но есть и такие ключи, которые относятся к другим контекстно-зависимым атрибутам, таким как обрабатываемая операция (например, добавление устройства или его удаление из системы). Все элементы *условие* должны сопоставляться в порядке активизации правила.

Ключи, воспринимаемые менеджером устройств **udev**, представлены в табл. 11.3.

Таблица 11.3. Ключи, воспринимаемые менеджером устройств udev

Ключ условия	Действие
ACTION	Сопоставляется с типом события, например добавление или удаление
ATTR{*имя_файла*}	Сопоставляется со значениями файловой системы **sysfs** устройства[a]
DEVPATH	Сопоставляется с конкретным путем устройства
DRIVER	Сопоставляется с драйвером, используемым устройством
ENV{*ключ*}	Сопоставляется со значением переменной среды
KERNEL	Сопоставляется с именем ядра для устройства
PROGRAM	Выполняет внешнюю команду; условие считается выполненным, если код завершения команды равен 0
RESULT	Сопоставляется с результатом последнего вызова посредством ключа **PROGRAM**
SUBSYSTEM	Сопоставляется с конкретной подсистемой
TEST{*omask*}	Проверяет факт существования файла; элемент *omask* необязателен

[a] *Имя_файла* — это лист на дереве файловой системы sysfs, соответствующий конкретному атрибуту.

В правилах сопоставления элементы *назначение* задают действия, которые должен предпринять демон **udevd**, чтобы обработать событие. Формат этих элементов подобен формату элементов *условие*.

Самым важным ключом элемента *назначение* является ключ NAME, который означает, каким именем менеджер udev должен назвать устройство. Необязательный ключ SYMLINK создает символическую ссылку на устройство через путь, заданный в каталоге **/dev**.

Рассмотрим совместную работу этих компонентов на примере флеш-памяти USB. Предположим, что мы хотим сделать имя этого устройства постоянным и обеспечить автоматическое его монтирование и размонтирование.

Вставим нашу флеш-память и посмотрим, как ядро может идентифицировать ее. Возможны два пути. Выполнив команду **lsusb**, мы можем исследовать шину USB напрямую.

```
ubuntu$ lsusb
Bus 001 Device 007: ID 1307:0163 Transcend, Inc. USB Flash Drive
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
```

В качестве альтернативного варианта можем проверить записи в журнале ядра с помощью команд **dmesg** или **journalctl**. В нашем случае устройство оставляет пространную контрольную запись.

```
Aug  9 19:50:03 ubuntu kernel: [42689.253554] scsi 8:0:0:0:
      Direct-Access  Ut163    USB2FlashStorage 0.00 PQ: 0 ANSI: 2
 Aug  9 19:50:03 ubuntu kernel: [42689.292226] sd 8:0:0:0: [sdb] 1974271
     512-byte hardware sectors: (1.01 GB/963 MiB)
...
Aug  9 19:50:03 ubuntu kernel: [42689.304749] sd 8:0:0:0: [sdb] 1974271
     512-byte hardware sectors: (1.01 GB/963 MiB)
Aug  9 19:50:03 ubuntu kernel: [42689.307182]  sdb: sdb1
Aug  9 19:50:03 ubuntu kernel: [42689.427785] sd 8:0:0:0: [sdb] Attached
     SCSI removable disk
Aug  9 19:50:03 ubuntu kernel: [42689.428405]  sd 8:0:0:0: Attached scsi
     generic sg3 type 0
```

Представленные выше журнальные сообщения означают, что устройство было распознано как **sdb**, и это позволяет нам просто идентифицировать устройство в файловой системе **/sys**. Теперь мы можем просмотреть файловую систему **/sys** с помощью команды **udevadm** и поискать фрагменты правил, которые служат характеристикой устройства и могут использоваться в правилах менеджера **udev**.

```
ububtu$ udevadm info -a -p /block/sdb/sdb1
looking at device '/devices/pci0000:00/0000:00:11.0/0000:02:03.0/
    usb1/1-1/1-1:1.0/host30/target30:0:0/30:0:0:0/block/sdb/sdb1':
    KERNEL=="sdb1"
    SUBSYSTEM=="block"
    DRIVER==""
    ATTR{partition}=="1"
    ATTR{start}=="63"
    ATTR{size}=="1974208"
    ATTR{stat}=="  71   792 1857 808 0   0   0   0   0   512   808"

looking at parent device '/devices/pci0000:00/0000:00:11.0/0000:02:03.0/
    usb1/1-1/1-1:1.0/host30/target30: 0:0/30:0:0:0/block/sdb':
    KERNELS=="sdb"
    SUBSYSTEMS=="block"
    DRIVERS==""
    ...
    ATTRS{scsi_level}=="3"
    ATTRS{vendor}=="Ut163    "
    ATTRS{model}=="USB2FlashStorage"
...
```

Результат выполнения команды **udevadm** отображает возможности совпадений с заданными значениями. Одна из них — поле **size**, которое, вероятно, является уникальным для данного устройства. Но если бы размер раздела изменился, устройство не было бы распознано. Мы можем также использовать сочетание двух значений: соглашение ядра о назначении имен в виде **sd** с дополнительной буквой и содержимое атрибута модели `USB2FlashStorage`. При создании правил для данного конкретного устройства флеш-памяти еще одним вариантом мог бы послужить серийный номер устройства (здесь он не отображен).

Поместим наши правила для этого устройства в файл **/etc/udev/rules.d/10-local.rules**. Поскольку мы собираемся достичь сразу нескольких целей, нужно создать ряд правил.

Прежде всего, мы должны позаботиться о создании символических ссылок на устройство в каталоге **/dev**. Следующее правило для идентификации устройства использует наши знания о ключах условий `ATTRS` и `KERNEL`, почерпнутых из команды **udevadm**.

```
ATTRS{model}=="USB2FlashStorage", KERNEL=="sd[a-z]1",
   SYMLINK+="ate-flash%n"
```

(Это правило представлено в виде двух строк, чтобы оно поместилось на странице книги; в оригинальном файле оно занимает одну строку.)

Когда наше правило срабатывает, демон **udevd** устанавливает значение **/dev/ate-flash***N* в качестве символической ссылки на устройство. В действительности мы не ожидаем в системе появления более одного такого устройства. Если обнаружится несколько копий устройства, они получат уникальные имена в каталоге **/dev**, но точные имена зависят от порядка вставки устройств в систему.

Затем мы используем ключ ACTION для выполнения ряда команд всякий раз, когда устройство будет появляться на шине USB. Ключ элемента *назначение* RUN позволяет создавать соответствующий каталог точки монтирования и монтировать там наше устройство.

```
ACTION=="add", ATTRS{model}=="USB2FlashStorage", KERNEL=="sd[a-z]1",
   RUN+="/bin/mkdir -p /mnt/ate-flash%n"
ACTION=="add", ATTRS{model}=="USB2FlashStorage", KERNEL=="sd[a-z]1",
   PROGRAM=="/lib/udev/vol_id -t %N", RESULT=="vfat", RUN+="/bin/mount
   -t vfat /dev/%k /mnt/ate-flash%n"
```

Ключи PROGRAM и RUN выглядят похожими, но напомним, что PROGRAM — это ключ условия, который активен во время фазы выбора правила, тогда как RUN — это ключ назначения, который является частью действий, инициированных правилом. Второе правило с помощью опции **-t vfat** в команде **mount** еще до операции монтирования проверяет, что флеш-память содержит файловую систему Windows.

Аналогичные правила после удаления устройства из системы позволяют выполнить очистительно-восстановительные операции.

```
ACTION=="remove", ATTRS{model}=="USB2FlashStorage",
   KERNEL=="sd[a-z]1", RUN+="/bin/umount -l /mnt/ate-flash%n"
ACTION=="remove", ATTRS{model}=="USB2FlashStorage",
   KERNEL=="sd[a-z]1", RUN+="/bin/rmdir /mnt/ate-flash%n"
```

Теперь, когда правила подготовлены, мы должны уведомить демон **udevd** о наших изменениях. Команда **control**, используемая в качестве аргумента команды **udevadm**, — одна из немногих, которые требуют полномочий суперпользователя.

```
ubuntu$ sudo udevadm control --reload-rules
```

Опечатки молча игнорируются после перезагрузки, даже с использованием флага **--debug**, поэтому не забудьте дважды проверить синтаксис своих правил.

Вот и все! Теперь, когда флеш-память вставляется в разъем USB-порта, демон **udevd** создает символическую ссылку **/dev/ate-flash1** и монтирует устройство **/mnt/ate-flash1**.

```
ubuntu$ ls -l /dev/ate*
lrwxrwxrwx 1 root root 4 2009-08-09 21:22 /dev/ate-flash1 -> sdb1

ubuntu$ mount | grep ate
/dev/sdb1 on /mnt/ate-flash1 type vfat (rw)
```

Управление устройствами в системе FreeBSD



Как мы видели в кратком обзоре, приведенном выше, реализация автономной файловой системы **/dev** в системе FreeBSD называется **devfs**, а ее демон управления устройствами на пользовательском уровне — **devd**.

Файловая система devfs: автоматическая конфигурация файла устройства

В отличие от файловой системы **udev** от Linux, **devfs** может настраиваться самостоятельно. Однако эта система настройки является и своеобразной, и ограниченной. Она делится на загрузочную (**/etc/devfs.conf**) и динамическую части (**/etc/devfs.rules**). Эти два файла конфигурации имеют разные синтаксисы и немного разные возможности.

Файловая система **devfs** для статических (несъемных) устройств настраивается в файле **/etc/devfs.conf**. Каждая строка — это правило, начинающееся с действия. Возможными действиями являются `link`, `own` и `perm`. Действие `link` устанавливает символические ссылки для определенных устройств. Действия `own` и `perm` изменяют владельцев и разрешения файлов устройств соответственно.

Каждое действие принимает два параметра, интерпретация которых зависит от конкретного действия. Например, предположим, что мы хотим, чтобы наш привод DVD-ROM **/dev/cd0** также был доступен по имени **/dev/dvd**. Следующая строка сделала бы такой трюк:

```
link cd0 dvd
```

Мы могли бы установить владельца и разрешения для устройства с помощью следующих строк.

```
own cd0 root:sysadmin
perm cd0 0660
```

Так же как файл **/etc/devfs.conf** содержит действия, которые необходимо предпринять для встроенных устройств, файл **/etc/devfs.rules** содержит правила для съемных устройств. Правила в файле **devfs.rules** также позволяют сделать устройства скрытыми или недоступными, что может быть полезно для окружений **jail(8)**.

Демон devd: управление устройствами более высокого уровня

Демон **devd** работает в фоновом режиме, наблюдая за событиями ядра, связанными с устройствами, и действуя на правила, определенные в файле **/etc/devd.conf**. Конфигурация **devd** подробно описана на справочной странице **devd.conf**, но файл **devd.conf** по умолчанию содержит множество полезных примеров и разъясняющих комментариев.

Формат файла **/etc/devd.conf** концептуально прост и состоит из инструкций, содержащих группы подстановок. Инструкции — это, по сути, правила, а вложенные инструкции содержат сведения о правиле. Доступные типы операторов перечислены в табл. 11.4.

Таблица 11.4. Типы инструкций в файле /etc/devd.conf

Инструкция	Описание
`attach`	Что делать, когда устройство вставлено
`detach`	Что делать, когда устройство удалено
`nomatch`	Что делать, если никакие другие инструкции не соответствуют устройству
`notify`	Как реагировать на события ядра, касающиеся устройства
`options`	Параметры конфигурации для самого демона **devd**

Несмотря на концептуальную простоту, язык конфигурации для вложенных инструкций богат и сложен. По этой причине многие из общих инструкций конфигурации уже включены в файл конфигурации стандартного дистрибутива. Во многих случаях вам никогда не придется изменять значение по умолчанию в файле **/etc/devd.conf**.

Автоматическая установка устройств сменных носителей, таких как жесткие диски USB и флеш-накопители, теперь обрабатывается в системе FreeBSD с помощью файловой системы **autofs**, а не с помощью демона **devd**. Общую информацию о файловой системе **autofs** см. в разделе 21.7. Хотя файловая система **autofs** встречается в большинстве UNIX-подобных операционных систем, в системе FreeBSD на нее возлагаются дополнительные задачи.

11.4. Конфигурирование ядра Linux



Существует три основных метода конфигурирования ядра Linux:

- модификация настраиваемых (динамических) параметров ядра;
- повторное создание ядра (компиляция исходных файлов ядра с возможными модификациями и дополнениями);
- динамическая загрузка новых драйверов и модулей в существующее ядро.

Каждый из перечисленных методов имеет свою область применения. Проще всего настраивать параметры ядра. Именно так обычно и поступают. Компиляция ядра — наиболее сложный метод, и к нему прибегают реже всего. Но любой метод вполне можно освоить, это лишь вопрос практики.

Конфигурирование параметров ядра linux

Многие модули и драйверы ядра разрабатывались с учетом того, что все в системе подвержено изменению. Для повышения гибкости были созданы специальные информационные каналы, которые позволяют системному администратору динамически корректировать различные параметры, определяющие размеры внутренних таблиц ядра и его поведение в конкретных ситуациях. Все эти каналы расположены в файловой системе **/proc** (также называемой **procfs**), которая представляет собой интерфейс между ядром и приложениями пользовательского уровня. Часто большие приложения пользовательского уровня (особенно такие приложения "инфраструктуры", как базы данных) требуют настройки параметров ядра.

Специальные файлы в каталоге **/proc/sys** позволяют просматривать и динамически изменять значения параметров ядра. Они напоминают стандартные файлы Linux, но на самом деле являются "черным входом" в ядро. Если в одном из файлов есть значение, которое требуется изменить, можно попытаться осуществить запись в этот файл. К сожалению, не все файлы доступны для записи (независимо от прав доступа), а объем имеющейся документации оставляет желать лучшего. Информацию о некоторых значениях и их назначении можно получить в подкаталоге **Documentation/sysctl** каталога с исходным кодом ядра (или онлайн по адресу `kernel.org/doc`).

Например, чтобы изменить максимальное число файлов, которые одновременно могут быть открыты в системе, выполните такие действия.

```
linux# cat /proc/sys/fs/file-max
34916
linux# echo 32768 > /proc/sys/fs/file-max
```

Со временем вырабатывается привычка к столь нетрадиционному способу взаимодействия с ядром, тем более что он очень удобен. Но сразу предупредим: изменения при перезагрузке не сохраняются.

Для того чтобы эти параметры при перезагрузке сохранялись, необходимо использовать команду **sysctl**. Она позволяет задавать отдельные переменные либо в командной строке, либо с помощью списка пар *переменная=значение* в файле конфигурации. По умолчанию файл **/etc/sysctl.conf** считывается при загрузке системы, а его содержимое используется для установки начальных значений ее параметров.

Например, команда

```
linux# sysctl net.ipv4.ip_forward=0
```

отключает IP-пересылку. (Это можно сделать также, вручную отредактировав файл `/etc/sysctl.conf`.) Имена переменных задаются с помощью команды `sysctl`, которая заменяет обратные косые черты в структуре каталога `/proc/sys` точками.

В табл. 11.5 перечислены некоторые обычно настраиваемые параметры ядре Linux версии 3.10.0. В разных дистрибутивах значения, действующие по умолчанию, варьируются в широких пределах.

Таблица 11.5. Файлы каталога `/proc/sys`, содержащие некоторые настраиваемые параметры ядра

Файл	Что он делает
`cdrom/autoclose`	Автоматически закрывает диск CD-ROM при монтировании
`cdrom/autoeject`	Автоматически извлекает диск CD-ROM при размонтировании
`fs/file-max`	Устанавливает максимальное количество открытых файлов
`kernel/ctrl-alt-del`	Перезагрузка при нажатии клавиш <Ctrl+Al+Del>; может увеличить безопасность на незащищенных консолях
`kernel/panic`	Устанавливает секунды ожидания перед перезагрузкой после сообщения о неисправимой ошибке ядра: 0 — цикл или бесконечное зависание
`kernel/panic_on_oops`	Определяет поведение ядра после обнаружения сбоя или ошибки: 1 — всегда выдавать сообщение о неисправимой ошибке
`kernel/printk_ratelimit`	Устанавливает минимальный интервал в секундах между сообщениями ядра
`kernel/printk_ratelimit_burst`	Устанавливает количество сообщений подряд до достижения частоты `printk`
`kernel/shmmax`	Устанавливает максимальный объем совместно используемой памяти
`net/ip*/conf/default/rp_filter`	Включает проверку маршрутизации по IP-адресу отправителя[а]
`net/ip*/icmp_echo_ignore_all`	Включает игнорирование эхо-запросов ICMP, если равен единице[б]
`net/ip*/ip_forward`	Позволяет IP-переадресацию, если равен единице[в]
`net/ip*/ip_local_port_range`	Устанавливает диапазон локального порта, используемый при настройке соединения[г]
`net/ip*/tcp_syncookies`	Защищает от синхронных атак (SYN flood attack); включите, если подозреваете атаку "отказ в обслуживании" (DoS)
`tcp_fin_timeout`	Устанавливает секунды для ожидания окончательного пакета TCP FIN[д]
`vm/overcommit_memory`	Контролирует переполнение памяти, т. е. определяет, как ядро реагирует на нехватку физической памяти для обработки запроса на размещение виртуальной машины
`vm/overcommit_ratio`	Определяет, сколько физической памяти (в процентах) будет использоваться при недостатке памяти

[а]Этот механизм защиты от дезинформирующих помех заставляет ядро отбрасывать пакеты, полученные из невозможных адресов.

[б]Связанная переменная `icmp_echo_ignore_broadcasts` игнорирует широковещательные ICMP-сообщения. Почти всегда целесообразно установить это значение равным единице.

[в]Устанавливайте это значение равным единице, только если вы явно намереваетесь использовать свой Linux-ящик в качестве сетевого маршрутизатора.

[г]Увеличьте этот диапазон до 1024–65000 на серверах, которые инициируют множество исходящих подключений.

[д]На серверах с высоким трафиком для повышения производительности попробуйте установить это значение на более низком уровне (~ 20).

Обратите внимание на то, что есть два подкаталога IP-сети **/proc/sys/net:ipv4** и **ipv6**. Раньше администраторам приходилось беспокоиться только о поведении протокола IPv4, потому что он был единственным. Но на момент написания этой книги (2017) все блоки адресов IPv4 уже были распределены, а протокол IPv6 развернут и используется почти повсеместно, даже в небольших организациях.

В общем случае, когда вы изменяете параметр для IPv4, вы также должны изменить этот параметр для IPv6, если вы поддерживаете оба протокола. Слишком легко изменить одну версию IP, а не другую, а затем столкнуться с проблемами несколько месяцев или лет спустя, когда пользователь сообщит о странном сетевом поведении.

Сборка ядра

Операционная система Linux развивается столь быстрыми темпами, что рано или поздно ее ядро приходится компилировать заново. Постоянно появляются исправления ядра, драйверы устройств и новые функции. Все это вызывает противоречивые чувства. С одной стороны, удобно иметь под рукой всегда самые новые инструменты, а с другой — стремление идти в ногу со временем требует постоянных усилий и значительных затрат времени.

Не чините то, что еще не поломано

Хороший системный администратор тщательно взвешивает все за и против, планируя обновление ядра. Конечно, самая последняя версия — это всегда модно, но является ли она столь же стабильной, что и текущая версия? Нельзя ли подождать до конца месяца, когда будет инсталлироваться целая группа исправлений? Важно, чтобы желание быть на переднем крае технологического прогресса не шло вразрез с интересами пользователей.

Возьмите за правило устанавливать обновления и изменения, только когда планируемая выгода (обычно определяемая на основе критериев надежности и производительности) окупит усилия и время, затрачиваемые на инсталляцию. Если выгоду оценить затруднительно, это верный признак того, что с обновлением можно подождать. (Безусловно, изменения, связанные с безопасностью, следует инсталлировать без промедления.)

Настройка для сборки ядра Linux

Менее вероятно, что вам придется самостоятельно собирать ядро, если вы используете дистрибутив, в котором используется стабильное ядро. Раньше считалось, что вторая часть номера версии указывает, было ли ядро стабильным (четные числа) или находилось в разработке (нечетные числа). Но в наши дни разработчики ядра больше не следуют этой системе. Официальная информация о любой конкретной версии ядра находится на сайте `kernel.org`. Этот сайт также является лучшим источником исходного кода ядра Linux, если вы не связаны с конкретным поставщиком или дистрибутивом.

Каждый дистрибутив имеет особый способ настройки и сборки пользовательских ядер. Однако дистрибутивы также поддерживают традиционный способ делать что-то, что мы здесь описываем. В целом безопаснее использовать рекомендованную вами дистрибьюторскую процедуру.

Конфигурирование параметров ядра Linux

Большинство дистрибутивов инсталлируют исходные файлы ядра в каталог **/usr/src/kernels**. В любом случае, чтобы можно было создавать ядро системы, должен быть инсталлирован пакет его исходных кодов. Об инсталляции пакетов рассказывается в главе 6.

Весь процесс настройки ядра сосредоточен вокруг файла **.config**, находящегося в каталоге исходных кодов ядра. В данном файле приведена вся информация, касающаяся конфигурации ядра, но его формат очень труден для понимания. Для выяснения смысла различных параметров воспользуйтесь пособием по параметрам конфигурации ядра, приведенным в файле

путь_к_исходным_текстам_ядра/**Documentation/Configure.help**.

Для того чтобы не редактировать файл **.config** напрямую, в системе Linux предусмотрено несколько целевых модулей утилиты **make**, реализующих различные интерфейсы конфигурирования ядра. Если в системе инсталлирована оболочка KDE, то удобнее всего воспользоваться командой **make xconfig**. Аналогично, если вы используете оболочку GNOME, вероятно, лучше всего применить команду **make gconfig**. Эти команды отображают окно конфигурации, в котором можно выбрать драйверы устройств, добавляемые к ядру (или компилируемые в качестве загружаемых модулей).

В отсутствие графической среды на помощь придет команда **make menuconfig**. Наконец, есть старая команда **make config**, которая выводит запрос на изменение каждого конфигурационного параметра и не позволяет изменять ранее установленные значения. Мы рекомендуем пользоваться командой **make xconfig** или **make gconfig**, если используемая среда их поддерживает, и командой **make menuconfig** — в противном случае. Применения команды **make config** лучше избегать (как самой негибкой и трудоемкой).

При переносе конфигурации существующего ядра в новую версию ядра (или файловую систему) можно использовать команду **make oldconfig** для считывания ранее использовавшегося файла **.config** и определения только тех параметров, которые изменились или являются новыми.

Все эти команды довольно просты, так как их действие сводится к установке нужных опций. Тем не менее они мало подходят для ситуации, когда нужно поддерживать несколько версий ядра для различных платформ или аппаратных конфигураций.

В любом случае генерируется файл **.config**, имеющий примерно такой вид.

```
# Automatically generated make config: don't edit
# Code maturity level options

CONFIG_EXPERIMENTAL=y

# Processor type and features
# CONFIG_M386 is not set
# CONFIG_M486 is not set
# CONFIG_M586 is not set
# CONFIG_M586TSC is not set
CONFIG_M686=y
CONFIG_X86_WP_WORKS_OK=y
CONFIG_X86_INVLPG=y
CONFIG_X86_BSWAP=y
CONFIG_X86_POPAD_OK=y
CONFIG_X86_TSC=y
CONFIG_X86_GOOD_APIC=y
```

Как видите, содержимое файла не слишком понятно. Не дается никаких пояснений относительно того, что означают параметры `CONFIG`. По сути, каждая строка указывает на активизацию того или иного параметра ядра. Значение `y` говорит о том, что соответствующий компонент компилируется вместе с ядром; значение `m` означает, что компонент станет загружаемым модулем.

Не все компоненты можно сконфигурировать в виде модулей. Сведений об этом в файле **.config** нет, поэтому нужно самостоятельно покопаться в документации. Не так-то легко узнать и назначение параметров **CONFIG_***.

Компиляция ядра

Формирование файла **.config** — самый важный этап конфигурирования ядра Linux, но нужно выполнить еще ряд действий, прежде чем будет получено готовое ядро.

Схема всего процесса такова:

- перейти в каталог верхнего уровня (**cd**), содержащий исходные коды ядра;
- выполнить команду **make xconfig**, **make gconfig** или **make menuconfig**;
- выполнить команду **make clean**;
- выполнить команду **make**;
- выполнить команду **make modules_install**;
- выполнить команду **make install**.

Выполнять команду **make clean** не всегда обязательно, но удаление всего лишнего позволит избежать множества проблем.

Добавление драйвера устройства в Linux

В системе Linux драйверы устройств распространяются в одной из трех форм:

- исправления конкретной версии ядра;
- загружаемого модуля;
- инсталляционного сценария или пакета, устанавливающего соответствующие исправления.

Наиболее распространенная форма — инсталляционный сценарий или пакет. Если вам повезло и вы располагаете таким сценарием или пакетом для нового устройства, достаточно выполнить стандартную процедуру установки новой программы.

В большинстве случае установку исправления конкретной версии ядра можно выполнить с помощью следующей команды.

```
linux# cd путь_к_исходным_кодам_ядра ; patch -p1 < файл_исправления
```

11.5. Конфигурация ядра системы FreeBSD



Система FreeBSD поддерживает те же три метода изменения параметров ядра, что и Linux: динамическая настройка работающего ядра, построение нового ядра из исходного кода и загрузка динамических модулей.

Настройка параметров ядра FreeBSD

Многие параметры ядра FreeBSD можно динамически изменять с помощью команды **sysctl**, как это делается в системе Linux. Вы можете установить значения автоматически во время загрузки, добавив их в файл **/etc/sysctl.conf**. Таким образом могут быть изменены очень многие параметры; чтобы увидеть их все, наберите команду **sysctl -a**. Не все, что отображается в выводе этой команды, может быть изменено; многие параметры доступны только для чтения.

В следующих абзацах описываются некоторые из наиболее часто изменяемых или интересных параметров, которые вы можете настроить.

Параметры `net.inet.ip.forwarding` и `net.inet6.ip6.forwarding` управляют пересылкой IP-пакетов для протоколов IPv4 и IPv6.

Параметр `kern.maxfiles` устанавливает максимальное количество дескрипторов файлов, которые система может открыть. Возможно, вам придется увеличить этот параметр в таких системах, как база данных или веб-серверы.

Параметр `net.inet.tcp.mssdflt` устанавливает максимальный размер сегмента TCP по умолчанию, который является размером полезной нагрузки TCP-пакета, переносимой по протоколу IPv4. Определенные размеры полезной нагрузки слишком велики для сетевых линий дальней связи и, следовательно, могут быть сброшены их маршрутизаторами. Изменение этого параметра может быть полезно при отладке проблем с удаленной связью.

Параметр `net.inet.udp.blackhole` контролирует, будет ли возвращен пакет ICMP с меткой "`port unreachable`", когда пакет поступает на закрытый UDP-порт. Включение этой опции (т.е. блокирование пакетов с меткой "`port unreachable`") может создать помехи для сканирования портов и потенциальных злоумышленников.

Параметр `net.inet.tcp.blackhole` аналогичен по определению параметру `udp.blackhole`. Обычно протокол TCP отправляет ответ RST (сброс соединения), когда пакеты поступают на закрытые порты. Установка этого параметра равным единице запрещает любому SYN-пакету (настройка соединения), поступающему на закрытый порт, генерировать RST-пакет. Установка его равным двум предотвращает генерирование ответного RST-пакета для любого сегмента, который поступает на закрытый порт.

Параметр `kern.ipc.nmbclusters` контролирует количество кластеров буферов `mbuf`, доступных для системы. Буферы `mbuf` образуют внутреннюю структуру хранилища для сетевых пакетов, а кластеры таких структур можно рассматривать как полезную нагрузку. Для серверов, которые испытывают тяжелые сетевые нагрузки, это значение, возможно, потребуется увеличить по сравнению с заданным по умолчанию (в настоящее время оно равно 507270 на FreeBSD 13).

Параметр `kern.maxvnodes` устанавливает максимальное количество виртуальных узлов, которые являются структурами данных ядра для отслеживания файлов. Увеличение количества доступных виртуальных узлов может повысить пропускную способность диска на занятом сервере. Изучите значение `vfs.numvnodes` на серверах, которые испытывают низкую производительность; если его значение близко к значению `kern.maxvnodes`, увеличьте его.

Сборка ядра FreeBSD

Исходный код ядра поступает с серверов FreeBSD в виде сжатого архива в формате `tarball`. Просто скачайте и распакуйте его для инсталляции. После того как исходное дерево ядра инсталлировано, процесс настройки и сборки ядра аналогичен процессу в системе Linux. Однако исходный код ядра всегда находится в каталоге **`/usr/src/sys`**. В этом каталоге содержится совокупность подкаталогов, по одному для каждой поддерживаемой архитектуры. Внутри каждого каталога архитектуры подкаталог **`conf`** включает в себя файл конфигурации `GENERIC` для так называемого "универсального ядра", которое поддерживает все возможные устройства и опции.

Файл конфигурации аналогичен файлу **`.config`** в системе Linux. Первым шагом в создании настраиваемого ядра является копирование файла **`GENERIC`** в новый файл

с другим именем в том же каталоге, например **MYCUSTOM**. Второй шаг — отредактировать файл конфигурации и изменить его параметры, комментируя функции и устройства, которые вам не нужны. Последним шагом является сборка и инсталляция ядра. Этот последний шаг должен быть выполнен в каталоге верхнего уровня **/usr/src**.

Файлы конфигурации ядра FreeBSD должны быть отредактированы вручную. Для этой задачи нет специальных пользовательских интерфейсов, как в системе Linux. Информацию об общем формате можно найти в справочной странице **config(5)**, а информацию о том, как используется файл конфигурации, можно найти в справочной странице **config(8)**.

Файл конфигурации содержит некоторые внутренние комментарии, описывающие, что делает каждый вариант. Тем не менее вам по-прежнему нужны некоторые базовые знания по широкому спектру технологий, чтобы принимать обоснованные решения о том, что нужно оставить. В общем, вы захотите оставить все параметры из конфигурации **GENERIC** включенными и модифицировать только связанные с устройством строки ниже в файле конфигурации. Лучше оставить опции включенными, если вы не уверены, что они вам не нужны.

Для окончательного этапа сборки ядра в системе FreeBSD есть единая, высокоавтоматизированная команда **make buildkernel**, которая объединяет анализ файла конфигурации, создание каталогов сборки, копирование соответствующих исходных файлов и компиляцию этих файлов. Этой целевой задаче можно указать пользовательское имя файла конфигурации в виде переменной сборки, KERNCONF. Аналогичная целевая задача инсталляции, **make installkernel**, устанавливает ядро и загрузчик.

Вот краткое описание процесса.

1. Измените каталог (**cd**) на **/usr/src/sys/arch/conf** для вашей архитектуры.
2. Скопируйте файл с универсальной конфигурацией: **cp GENERIC MYCUSTOM**.
3. Отредактируйте конфигурационный файл **MYCUSTOM**.
4. Измените каталог на **/usr/src**.
5. Выполните команду **make buildkernel KERNCONF=MYCUSTOM**.
6. Выполните команду **make installkernel KERNCONF=MYCUSTOM**.

Обратите внимание на то, что эти шаги не допускают кросс-компиляции! Иначе говоря, если ваша машина сборки имеет архитектуру AMD64, вы не можете перейти в каталог **/usr/src/sys/sparc/conf**, выполнять обычные операции и получить ядро с поддержкой SPARC.

11.6. Загружаемые модули ядра

Загружаемые модули ядра в настоящее время являются общим фактором практически для всех версий UNIX. Каждый из наших примеров систем в той или иной форме реализует возможность динамической загрузки.

Благодаря наличию загружаемых модулей появляется возможность подключать и отключать драйверы устройств и другие компоненты ядра непосредственно в процессе работы системы. Это значительно облегчает инсталляцию драйверов, поскольку исполняемый код ядра не нужно менять. Кроме того, уменьшается размер ядра, так как ненужные драйверы просто не загружаются.

Загружаемые драйверы очень удобны, но они не обеспечивают стопроцентную безопасность. При каждой загрузке и выгрузке модуля существует риск нарушить работоспособность ядра. Поэтому во избежание нарушения работы системы не используйте непроверенные модули.

Подобно другим аспектам управления устройствами и драйверами, реализация загружаемых модулей зависит от операционной системы.

Загружаемые модули ядра в Linux



В системе Linux почти все программные компоненты можно сделать загружаемыми модулями. Исключение составляет драйвер устройства, на котором расположена корневая файловая система, а также драйвер мыши PS/2.

Загружаемые модули обычно хранятся в каталоге **/lib/modules/***версия*, где последняя часть имени — это версия ядра Linux, сообщаемая командой **uname -r**. Получить список загруженных в данный момент модулей можно с помощью команды **lsmod**.

```
redhat$ lsmod
Module                  Size    Used by
ipmi_devintf           13064    2
ipmi_si                36648    1
ipmi_msghandler        31848    2 ipmi_devintf,ipmi_si
iptable_filter          6721    0
ip_tables              21441    1 iptable_filter
...
```

Как видно из списка, в системе установлены модули интеллектуального интерфейса управления платформой (Intelligent Platform Management Interface — IPMI) и брандмауэр **iptables**.

Приведем пример загрузки вручную модуля ядра, который был установлен в предыдущем разделе.

```
redhat$ sudo modprobe snd-usb-audio
```

Загружаемым модулям ядра можно также передавать конфигурационные параметры.

```
redhat$ sudo modprobe snd-usb-audio nrpack=8 asynk_ulink=1
```

Модуль **modprobe** — это полуавтоматическая оболочка вокруг более примитивной команды, **insmod**. Этот модуль распознает зависимости, параметры, а также процедуры инсталляции и удаления. Он также проверяет номер версии на текущем ядре и выбирает подходящую версию модуля из каталога **/lib/modules**. Он просматривает файл **/etc/modprobe.conf**, чтобы выяснить, как обрабатывать каждый конкретный модуль.

После того как модуль вручную добавлен к ядру, удалить его можно только по явному запросу или при перезагрузке системы. Для удаления нашего аудиомодуля, загруженного выше, подойдет команда **modprobe -r snd-usb-audio**. Удаление происходит только при условии, что количество текущих ссылок на этот модуль равно нулю (см. столбец `Used by` в выводе команды **lsmod**).

Можно динамически создать файл **/etc/modprobe.conf**, соответствующий текущей конфигурации ядра, выполнив команду **modprobe -c**. Эта команда генерирует длинный файл, который выглядит примерно так.

```
#This file was generated by: modprobe -c
path[pcmcia]=/lib/modules/preferred
path[pcmcia]=/lib/modules/default
path[pcmcia]=/lib/modules/2.6.6
```

```
path[misc]=/lib/modules/2.6.6
...
# Aliases
alias block-major-1 rd
alias block-major-2 floppy
...
alias char-major-4 serial
alias char-major-5 serial
alias char-major-6 lp
...
alias dos msdos
alias plip0 plip
alias ppp0 ppp
options ne io=x0340 irq=9
```

Инструкции `path` указывают на то, где находится конкретный модуль. Можно модифицировать или добавлять записи данного типа, если нужно хранить модули в нестандартных каталогах.

Инструкция `alias` обеспечивает привязку старших номеров блочных устройств, старших номеров символьных устройств, файловых систем, сетевых устройств и сетевых протоколов к соответствующим модулям.

Строки с ключевым словом `options` не генерируются динамически, но администратор должен добавить их вручную. Они задают параметры, передаваемые модулю при загрузке. Например, в следующей строке аудиомодулю USB сообщаются дополнительные параметры.

```
options snd-usb-audio nrpack=8 asynk_unlink ==1
```

Команда **modprobe** понимает также инструкции `install` и `remove`. С их помощью указываются команды, которые выполняются, когда соответствующий модуль подключается к работающему ядру или отключается от него.

Загружаемые модули ядра в системе FreeBSD



Модули ядра в системе FreeBSD находятся в каталоге в **/boot/kernel** (стандартные модули, входящие в дистрибутив) или **/boot/modules** (переносные, коммерческие и специальные модули). Каждый модуль ядра использует расширение имени файла **.ko**, но нет необходимости указывать это расширение при загрузке, разгрузке или просмотре состояния модуля.

Например, чтобы загрузить модуль с именем **foo.ko**, выполните команду **kldload foo** в соответствующем каталоге. Чтобы выгрузить модуль, выполните команду **kldunload foo** из любого места. Чтобы просмотреть статус модуля, выполните команду **kldstat -m foo** из любого места. Выполнение команды **kldstat** без каких-либо параметров отображает состояние всех загружаемых модулей.

Модули, перечисленные в любом из файлов **/boot/defaults/loader.conf** (системные значения по умолчанию) или **/boot/loader.conf**, загружаются автоматически во время загрузки. Чтобы добавить новую запись в файл **/boot/loader.conf**, используйте строку вида

```
zfs_load = "YES"
```

Соответствующее имя переменной — это просто базовое имя модуля с добавлением к нему суффикса **_load**. Вышеприведенная строка гарантирует загрузку модуля **/boot/kernel/zfs.ko** при загрузке системы; он реализует файловую систему ZFS.

11.7. Загрузка

Теперь, когда мы рассмотрели основы ядра, пришло время узнать, что на самом деле происходит, когда ядро загружается и инициализируется при загрузке системы. Вы, без сомнения, видели бесчисленные загрузочные сообщения, но знаете ли вы, что на самом деле означают все эти сообщения?

Следующие сообщения и аннотации приводятся на некоторых ключевых этапах процесса загрузки. Они почти наверняка не будут точно соответствовать тому, что вы видите в своих системах и ядрах. Тем не менее они должны дать вам представление о некоторых основных темах, возникающих при загрузке, и о том, как запускаются ядра Linux и FreeBSD.

Загрузочные сообщения системы Linux



Первый загрузочный журнал, который мы рассмотрим, получен с машины CentOS 7 с ядром 3.10.0.

```
Feb 14 17:18:57 localhost kernel: Initializing cgroup subsys cpuset
Feb 14 17:18:57 localhost kernel: Initializing cgroup subsys cpu
Feb 14 17:18:57 localhost kernel: Initializing cgroup subsys cpuacct
Feb 14 17:18:57 localhost kernel: Linux version 3.10.0-327.el7.x86_64
    (builder@kbuilder.dev.centos.org) (gcc version 4.8.3 20140911 (Red
    Hat 4.8.3-9) (GCC) ) #1 SMP Thu Nov 19 22:10:57 UTC 2015
Feb 14 17:18:57 localhost kernel: Command line: BOOT_IMAGE=/
    vmlinuz-3.10.0-327.el7.x86_64 root=/dev/mapper/centos-root ro
    crashkernel=auto rd.lvm.lv=centos/root rd.lvm.lv=centos/swap rhgb
    quiet LANG=en_US.UTF-8
```

Эти начальные сообщения говорят нам, что группы управления верхнего уровня (`cgroup`) запускаются в ядре Linux 3.10.0. Сообщения дают информацию о том, кто собрал ядро, где и какой компилятор использовался (`gcc`). Обратите внимание: хотя этот журнал поступает из системы CentOS, он является клоном Red Hat, и сообщения загрузки напоминают нам об этом факте.

Параметры, установленные в конфигурации загрузки GRUB и переданные оттуда в ядро, перечислены выше как "командная строка".

```
Feb 14 17:18:57 localhost kernel: e820: BIOS-provided physical RAM map:
Feb 14 17:18:57 localhost kernel: BIOS-e820: [mem 0x0000000000000000-
    0x000000000009fbff] usable
Feb 14 17:18:57 localhost kernel: BIOS-e820: [mem 0x000000000009fc00-
    0x000000000009ffff] reserved
...
Feb 14 17:18:57 localhost kernel: Hypervisor detected: KVM
Feb 14 17:18:57 localhost kernel: AGP: No AGP bridge found
Feb 14 17:18:57 localhost kernel: x86 PAT enabled: cpu 0, old
    0x7040600070406, new 0x7010600070106
Feb 14 17:18:57 localhost kernel: CPU MTRRs all blank - virtualized
    system.
Feb 14 17:18:57 localhost kernel: e820: last_pfn = 0xdfff0 max_arch_pfn
    = 0x400000000
Feb 14 17:18:57 localhost kernel: found SMP MP-table at [mem 0x0009fff0-
    0x0009ffff] mapped at [ffff88000009fff0]
Feb 14 17:18:57 localhost kernel: init_memory_mapping: [mem
    0x00000000-0x000fffff]
...
```

Эти сообщения описывают процессор, обнаруженный ядром, и показывают, как распределяется оперативная память. Обратите внимание на то, что ядру известно, что оно загружается в гипервизор и на самом деле не работает на физическом оборудовании.

```
Feb 14 17:18:57 localhost kernel: ACPI: bus type PCI registered
Feb 14 17:18:57 localhost kernel: acpiphp: ACPI Hot Plug PCI Controller
    Driver version: 0.5
...
Feb 14 17:18:57 localhost kernel: PCI host bridge to bus 0000:00
Feb 14 17:18:57 localhost kernel: pci_bus 0000:00: root bus resource [bus
    00-ff]
Feb 14 17:18:57 localhost kernel: pci_bus 0000:00: root bus resource [io
    0x0000-0xffff]
Feb 14 17:18:57 localhost kernel: pci_bus 0000:00: root bus resource
    [mem 0x00000000-0xffffffff]
...
Feb 14 17:18:57 localhost kernel: SCSI subsystem initialized
Feb 14 17:18:57 localhost kernel: ACPI: bus type USB registered
Feb 14 17:18:57 localhost kernel: usbcore: registered new interface driver
    usbfs
Feb 14 17:18:57 localhost kernel: PCI: Using ACPI for IRQ routing
```

Здесь ядро инициализирует различные шины данных, включая шину PCI и подсистему USB.

```
Feb 14 17:18:57 localhost kernel: Non-volatile memory driver v1.3
Feb 14 17:18:57 localhost kernel: Linux agpgart interface v0.103
Feb 14 17:18:57 localhost kernel: crash memory driver: version 1.1
Feb 14 17:18:57 localhost kernel: rdac: device handler registered
Feb 14 17:18:57 localhost kernel: hp_sw: device handler registered
Feb 14 17:18:57 localhost kernel: emc: device handler registered
Feb 14 17:18:57 localhost kernel: alua: device handler registered
Feb 14 17:18:57 localhost kernel: libphy: Fixed MDIO Bus: probed
...
Feb 14 17:18:57 localhost kernel: usbserial: USB Serial support
    registered for generic
Feb 14 17:18:57 localhost kernel: i8042: PNP: PS/2 Controller
    [PNP0303:PS2K,PNP0f03:PS2M] at 0x60,0x64 irq 1,12
Feb 14 17:18:57 localhost kernel: serio: i8042 KBD port 0x60,0x64 irq 1
Feb 14 17:18:57 localhost kernel: serio: i8042 AUX port 0x60,0x64 irq 12
Feb 14 17:18:57 localhost kernel: mousedev: PS/2 mouse device common for
    all mice
Feb 14 17:18:57 localhost kernel: input: AT Translated Set 2 keyboard as /
    devices/platform/i8042/serio0/input/input2
Feb 14 17:18:57 localhost kernel: rtc_cmos rtc_cmos: rtc core: registered
    rtc_cmos as rtc0
Feb 14 17:18:57 localhost kernel: rtc_cmos rtc_cmos: alarms up to one
    day, 114 bytes nvram
Feb 14 17:18:57 localhost kernel: cpuidle: using governor menu
Feb 14 17:18:57 localhost kernel: usbhid: USB HID core driver
```

Эти сообщения документируют обнаружение ядром различных устройств, включая кнопку питания, концентратор USB, мышь и микросхему часов реального времени (RTC). Некоторые из устройств являются метаданными, а не реальным оборудованием; эти конструкции управляют группами реальных, связанных аппаратных устройств. Например, драйвер `usbhid` (USB Human Interface Device) управляет клавиатурами, мышами, планшетами, игровыми контроллерами и другими типами устройств ввода, которые следуют стандартам отчетности USB.

```
Feb 14 17:18:57 localhost kernel: drop_monitor: Initializing network drop
    monitor service
Feb 14 17:18:57 localhost kernel: TCP: cubic registered
Feb 14 17:18:57 localhost kernel: Initializing XFRM netlink socket
Feb 14 17:18:57 localhost kernel: NET: Registered protocol family 10
Feb 14 17:18:57 localhost kernel: NET: Registered protocol family 17
```

На этом этапе ядро инициализирует множество сетевых драйверов и средств. Подсистема `drop_monitor` — это подсистема ядра Red Hat, которая осуществляет комплексный мониторинг потери сетевых пакетов. TCP cubic — это алгоритм управления перегрузкой, оптимизированный для соединений с высокой задержкой и высокой пропускной способностью, т.е. каналов с повышенной пропускной способностью (long fat pipes).

Как упоминалось в разделе 11.3, сокеты Netlink представляют собой современный подход к обмену данными между процессами ядра и уровня пользователя. Сокет XFRM Netlink является связующим звеном между процессом IPsec на уровне пользователя и подпрограммами IPsec ядра.

Последние две строки документируют регистрацию двух дополнительных семейств сетевых протоколов.

```
Feb 14 17:18:57 localhost kernel: Loading compiled-in X.509 certificates
Feb 14 17:18:57 localhost kernel: Loaded X.509 cert 'CentOS Linux kpatch
    signing key: ea0413152cde1d98ebdca3fe6f0230904c9ef717'
Feb 14 17:18:57 localhost kernel: Loaded X.509 cert 'CentOS Linux Driver
    update signing key: 7f421ee0ab69461574bb358861dbe77762a4201b'
Feb 14 17:18:57 localhost kernel: Loaded X.509 cert 'CentOS Linux kernel
    signing key: 79ad886a113ca0223526336c0f825b8a94296ab3'
Feb 14 17:18:57 localhost kernel: registered taskstats version 1
Feb 14 17:18:57 localhost kernel: Key type trusted registered
Feb 14 17:18:57 localhost kernel: Key type encrypted registered
```

Как и другие операционные системы, CentOS предоставляет возможность включать и проверять обновления. Часть проверки использует сертификаты X.509, установленные в ядре.

```
Feb 14 17:18:57 localhost kernel: IMA: No TPM chip found, activating
    TPM-bypass!
Feb 14 17:18:57 localhost kernel: rtc_cmos rtc_cmos: setting system clock
    to 2017-02-14 22:18:57 UTC (1487110737)
```

Здесь ядро сообщает, что он не может найти доверенный платформенный модуль (TPM) в системе. Микросхемы TPM — это криптографические аппаратные устройства, обеспечивающие безопасные операции подписи. При правильном использовании они могут помешать взломать систему.

Например, микросхемы TPM можно использовать для подписания кода ядра и для отказа системы выполнить любую часть кода, для которой текущая подпись не соответствует подписи TPM. Эта мера помогает избежать выполнения вредоносного кода. Администратор, который ожидает иметь работоспособную микросхему TPM, будет недоволен, увидев это сообщение!

В последнем сообщении показано, как ядро устанавливает часы реального времени с батарейным питанием в текущее время суток. Это тот же пакет RTC, который мы видели ранее, когда он был идентифицирован как устройство.

```
Feb 14 17:18:57 localhost kernel: e1000: Intel(R) PRO/1000 Network
    Driver - version 7.3.21-k8-NAPI
Feb 14 17:18:57 localhost kernel: e1000: Copyright (c) 1999-2006 Intel
    Corporation.
```

```
Feb 14 17:18:58 localhost kernel: e1000 0000:00:03.0 eth0:
    (PCI:33MHz:32-bit) 08:00:27:d0:ae:6f
Feb 14 17:18:58 localhost kernel: e1000 0000:00:03.0 eth0: Intel(R)
    PRO/1000 Network Connection
```

Теперь ядро нашло интерфейс гигабитного Ethernet и инициализировало его. MAC-адрес интерфейса (`08:00:27:d0:ae:6f`) может вас заинтересовать, если вы хотите, чтобы этот компьютер получал свой IP-адрес через DHCP. Конкретные IP-адреса часто блокируются для определенных MAC-адресов в конфигурации сервера DHCP, поэтому серверы могут поддерживать непрерывность IP-адресов.

```
Feb 14 17:18:58 localhost kernel: scsi host0: ata_piix
Feb 14 17:18:58 localhost kernel: ata1: PATA max UDMA/33 cmd 0x1f0 ctl
    0x3f6 bmdma 0xd000 irq 14
Feb 14 17:18:58 localhost kernel: ahci 0000:00:0d.0: flags: 64bit ncq
    stag only ccc
Feb 14 17:18:58 localhost kernel: scsi host2: ahci
Feb 14 17:18:58 localhost kernel: ata3: SATA max UDMA/133 abar
    m8192@0xf0806000 port 0xf0806100 irq 21
Feb 14 17:18:58 localhost kernel: ata2.00: ATAPI: VBOX CD-ROM, 1.0, max
    UDMA/133
Feb 14 17:18:58 localhost kernel: ata2.00: configured for UDMA/33
Feb 14 17:18:58 localhost kernel: scsi 1:0:0:0: CD-ROM VBOX
    CD-ROM           1.0 PQ: 0 ANSI: 5
Feb 14 17:18:58 localhost kernel: tsc: Refined TSC clocksource
    calibration: 3399.654 MHz
Feb 14 17:18:58 localhost kernel: ata3: SATA link up 3.0 Gbps (SStatus
    123 SControl 300)
Feb 14 17:18:58 localhost kernel: ata3.00: ATA-6: VBOX HARDDISK, 1.0,
    max UDMA/133
Feb 14 17:18:58 localhost kernel: ata3.00: 16777216 sectors, multi 128:
    LBA48 NCQ (depth 31/32)
Feb 14 17:18:58 localhost kernel: ata3.00: configured for UDMA/133
Feb 14 17:18:58 localhost kernel: scsi 2:0:0:0: Direct-Access ATA
    VBOX HARDDISK    1.0 PQ: 0 ANSI: 5
Feb 14 17:18:58 localhost kernel: sr 1:0:0:0: [sr0] scsi3-mmc drive:
    32x/32x xa/form2 tray
Feb 14 17:18:58 localhost kernel: cdrom: Uniform CD-ROM driver Revision:
    3.20
Feb 14 17:18:58 localhost kernel: sd 2:0:0:0: [sda] 16777216 512-byte
    logical blocks: (8.58 GB/8.00 GiB)
Feb 14 17:18:58 localhost kernel: sd 2:0:0:0: [sda] Attached SCSI disk
```

🕮 Дополнительную информацию о сервере DHCP см. в разделе 13.7.

```
Feb 14 17:18:58 localhost kernel: SGI XFS with ACLs, security attributes,
    no debug enabled
Feb 14 17:18:58 localhost kernel: XFS (dm-0): Mounting V4 Filesystem
Feb 14 17:18:59 localhost kernel: XFS (dm-0): Ending clean mount
```

Здесь ядро распознает и инициализирует различные диски и поддерживающие устройства (жесткие диски, виртуальный CD-ROM на основе SCSI и жесткий диск ATA). Он также монтирует файловую систему (XFS), которая является частью подсистемы device-mapper (файловая система dm-0).

Как вы можете видеть, сообщения загрузки Linux в Linux довольно подробны. Тем не менее вы можете быть уверены, что увидите все, что делает ядро, когда оно запускается. Это наиболее полезная функция при возникновении проблем.

Загрузочные сообщения системы FreeBSD



Ниже приведен журнал из системы FreeBSD 10.3-RELEASE, которая запускает ядро, поставляемое с дистрибутивом. Большая часть информации будет выглядеть странно знакомой; последовательность событий очень похожа на последовательность, обнаруженную в системе Linux. Одно из отличительных отличий заключается в том, что ядро FreeBSD производит гораздо меньше загрузочных сообщений, чем Linux. По сравнению с Linux, FreeBSD намного молчаливее.

```
Sep 25 12:48:36 bucephalus kernel: FreeBSD 10.3-RELEASE #0 r297264: Fri
    Mar 25 02:10:02 UTC 2016
Sep 25 12:48:36 bucephalus kernel: root@relengl.nyi.freebsd.org:/usr/obj/
    usr/src/sys/GENERIC amd64
Sep 25 12:48:36 bucephalus kernel: FreeBSD clang version 3.4.1 (tags/
    RELEASE_34/dotl-final 208032) 20140512
```

В начальных сообщениях, приведенных выше, говорится о выпуске операционной системы, времени, когда ядро было собрано из исходных файлов, имени сборщика, файла конфигурации, который использовался, и, наконец, компилятора, который сгенерировал код (Clang версия 3.4.1).

```
Sep 25 12:48:36 bucephalus kernel: real memory = 4831838208 (4608 MB)
Sep 25 12:48:36 bucephalus kernel: avail memory = 4116848640 (3926 MB)
```

Выше представлены общий объем памяти системы и ее объем, доступный для кода пользователя. Остальная часть памяти зарезервирована для самого ядра. Общая память в объеме 4608 Мбайт, вероятно, выглядит немного странно. Однако этот экземпляр FreeBSD работает под гипервизором. Объем реальной памяти — это произвольное значение, которое было установлено при настройке виртуальной машины. В системах с физическими устройствами объем общей памяти, вероятно, будет кратен степени двойки, поскольку именно так производятся микросхемы RAM RAM (например, 8192 Мбайт).

```
Sep 25 12:48:36 bucephalus kernel: vgapci0: <VGA-compatible display>
    mem 0xe0000000-0xe0ffffff irq 18 at device 2.0 on pci0
Sep 25 12:48:36 bucephalus kernel: vgapci0: Boot video device
```

Существует видеодисплей по умолчанию, который был найден на шине PCI. На выходе отображается диапазон памяти, в который был отображен буфер кадра.

```
Sep 25 12:48:36 bucephalus kernel: em0: <Intel(R) PRO/1000 Legacy
    Network Connection 1.1.0> port 0xd010-0xd017 mem 0xf0000000-
    0xf001ffff irq 19 at device 3.0 on pci0
Sep 25 12:48:36 bucephalus kernel: em0: Ethernet address:
    08:00:27:b5:49:fc
```

Выше приведены интерфейс Ethernet, а также его аппаратный (MAC) адрес.

```
Sep 25 12:48:36 bucephalus kernel: usbus0: 12Mbps Full Speed USB v1.0
Sep 25 12:48:36 bucephalus kernel: ugen0.1: <Apple> at usbus0
Sep 25 12:48:36 bucephalus kernel: uhub0: <Apple OHCI root HUB, class
    9/0, rev 1.00/1.00, addr 1> on usbus0
Sep 25 12:48:36 bucephalus kernel: ada0 at ata0 bus 0 scbus0 tgt 0 lun 0
Sep 25 12:48:36 bucephalus kernel: cd0 at ata1 bus 0 scbus1 tgt 0 lun 0
Sep 25 12:48:36 bucephalus kernel: cd0: <VBOX CD-ROM 1.0> Removable
    CD-ROM SCSI device

Sep 25 12:48:36 bucephalus kernel: cd0: Serial Number VB2-01700376
Sep 25 12:48:36 bucephalus kernel: cd0: 33.300MB/s transfers (UDMA2,
    ATAPI 12bytes, PIO 65534bytes)
```

```
Sep 25 12:48:36 bucephalus kernel: cd0: Attempt to query device size
    failed: NOT READY, Medium not present
Sep 25 12:48:36 bucephalus kernel: ada0: <VBOX HARDDISK 1.0> ATA-6
    device
Sep 25 12:48:36 bucephalus kernel: ada0: Serial Number
    VBcf309b40-154c5085
Sep 25 12:48:36 bucephalus kernel: ada0: 33.300MB/s transfers (UDMA2,
    PIO 65536bytes)
Sep 25 12:48:36 bucephalus kernel: ada0: 4108MB (8413280 512 byte
    sectors)
Sep 25 12:48:36 bucephalus kernel: ada0: Previously was known as ad0
```

Как показано выше, ядро инициализирует шину USB, концентратор USB, дисковод CD-ROM (на самом деле привод DVD-ROM, но виртуализованный, чтобы выглядеть как CD-ROM) и драйвер диска `ada`.

```
Sep 25 12:48:36 bucephalus kernel: random: unblocking device.
Sep 25 12:48:36 bucephalus kernel: Timecounter "TSC-low" frequency
    1700040409 Hz quality 1000
Sep 25 12:48:36 bucephalus kernel: Root mount waiting for: usbus0
Sep 25 12:48:36 bucephalus kernel: uhub0: 12 ports with 12 removable,
    self powered
Sep 25 12:48:36 bucephalus kernel: Trying to mount root from ufs:/dev/
    ada0p2 [rw]...
```

Последние сообщения в загрузочном журнале FreeBSD содержат множество вариантов и результатов. Псевдоустройство `random` оценивает энтропию системы и генерирует случайные числа. Ядро инициализирует генератор случайных чисел и устанавливает его в режим без блокировки. Появилось еще несколько устройств, и ядро смонтировало корневую файловую систему.

🕮 Дополнительную информацию о драйвере `random` см. в разделе 30.4.

На этом этапе загружаются сообщения загрузки ядра. После того как корневая файловая система смонтирована, ядро переходит в многопользовательский режим и выполняет сценарии запуска на уровне пользователя. Эти сценарии, в свою очередь, запускают системные службы и делают систему доступной для использования.

11.8. Загрузка альтернативных ядер в облаке

Облачные экземпляры загружаются иначе, чем традиционное оборудование. Большинство провайдеров облачных вычислений отказались от загрузчика GRUB и использовали либо модифицированный загрузчик с открытым исходным кодом, либо какую-то схему, которая вообще избегает использования загрузчика. Поэтому для загрузки альтернативного ядра в экземпляре облака обычно требуется, чтобы вы взаимодействовали с веб-консолью облачного провайдера или интерфейсом прикладного программирования.

В этом разделе кратко описаны некоторые особенности, связанные с загрузкой и выбором ядра на наших примерах облачных платформ. Более общее введение в облачные системы см. в главе 9.

На сервере AWS вам нужно начать с базового AMI (образа машины Amazon), который использует загрузчик PV-GRUB. PV-GRUB запускает исправленную версию устаревшего GRUB и позволяет указать ядро в файле `menu.lst` вашего AMI.

После компиляции нового ядра отредактируйте файл **/boot/grub/menu.lst**, чтобы добавить его в список загрузки.

```
default 0
fallback 1
timeout 0
hiddenmenu

title My Linux Kernel
root (hd0)
kernel /boot/my-vmlinuz-4.3 root=LABEL=/ console=hvc0
initrd /boot/my-initrd.img-4.3

title Amazon Linux
root (hd0)
kernel /boot/vmlinuz-4.1.10-17.31.amzn1.x86 root=LABEL=/ console=hvc0
initrd /boot/initramfs-4.1.10-17.31.amzn1.x86.img
```

Здесь пользовательское ядро задано по умолчанию, а параметр `fallback` указывает на стандартное ядро Amazon Linux. Наличие резервной копии помогает гарантировать, что ваша система может загрузиться, даже если настраиваемое ядро не может быть загружено или работает неправильно. Дополнительная информация об этом процессе содержится в руководстве пользователя Amazon EC2 для экземпляров Linux.

В прошлом провайдер DigitalOcean обходил загрузчик с помощью функции QEMU (сокращение от Quick Emulator), которая позволяла загружать ядро и RAM-диск непосредственно в дроплет. К счастью, теперь DigitalOcean позволяет дроплетам использовать свои собственные загрузчики. Ими поддерживаются большинство современных операционных систем, включая CoreOS, FreeBSD, Fedora, Ubuntu, Debian и CentOS. Изменения параметров загрузки, включая выбор ядра, обрабатываются соответствующими загрузчиками операционных систем (обычно GRUB).

Облачная платформа Google (GCP) является самой гибкой платформой, когда дело доходит до управления загрузкой. Google позволяет загружать полные изображения системных дисков в вашу учетную запись Compute Engine. Обратите внимание: для правильной загрузки образа GCP он должен использовать схему разбиения MBR и включать соответствующий (установленный) загрузчик. UEFI и GPT здесь не применяются!

Справочник `cloud.google.com/compute/docs/creating-custom-image` по созданию изображений невероятно подробный и описывает не только необходимые параметры ядра, но и рекомендованные параметры безопасности.

11.9. Ошибки ядра

Сбой ядра (например, паника ядра, т.е. сообщение о неисправимой ошибке) — это горькая реальность, которая может случиться даже в правильно настроенных системах. Для этого есть множество причин. К сбою системы могут привести неправильные команды, введенные привилегированными пользователями, но более распространенной причиной является неисправное оборудование. Ошибки физической памяти и жесткого диска (сбойные сектора на диске или устройстве) также печально известны тем, что вызывают панику ядра.

Также возможно, что к сбоям приводят ошибки в реализации ядра. Однако в ядрах, отмеченных как стабильные, такие аварии чрезвычайно редки. Драйверы устройств — это другое дело. Они поступают из разных источников и часто имеют более низкое качество кода.

Если аппаратное обеспечение является основной причиной сбоя, имейте в виду, что авария могла произойти задолго до сбоя устройства, вызвавшего его. Например, существует возможность "горячей" замены жесткого диска. Система может довольно долго работать бесперебойно, пока вы не попытаетесь перезагрузить или выполнить какую-либо другую операцию, зависящую от этого конкретного диска.

Несмотря на эмоциональные названия "паника" и "авария", паника ядра — это, как правило, относительно структурированное событие. Программы из пространства пользователя полагаются на то, что ядро само справится с многими видами неправильного поведения, но ядро должно выполнять мониторинг себя самого. По этой причине ядра содержат средства для проверки работоспособности, которые пытаются проверять важные структуры данных и инварианты. Ни одна из этих проверок не должна потерпеть неудачу; если это произойдет, это будет достаточным основанием для паники и остановки системы, и ядро делает это профилактически.

По крайней мере, это традиционный подход. Linux немного ослабила это правило с помощью функции `oops`; см. следующий раздел.

Ошибки ядра Linux



В системе Linux существуют четыре разновидности отказа ядра: мягкая блокировка, жесткая блокировка, паника и печально известная функция `oops`. Каждый из них обычно обеспечивает полную трассировку стека, за исключением определенных мягких блокировок, которые можно восстановить без паники.

Мягкая блокировка происходит, когда система находится в режиме ядра более нескольких секунд, тем самым предотвращая запуск пользовательских задач. Интервал настраивается, но обычно он составляет около 10 с, что является длительным периодом, когда процесс отказывается от циклов процессора! Во время мягкой блокировки ядро — единственный работающий компонент, но оно по-прежнему обслуживает прерывания, такие как сетевые интерфейсы и клавиатуры. Данные все еще текут и выходят из системы, хотя и потенциально искалечены.

Жесткая блокировка похожа на мягкую, но с дополнительным усложнением — она прекращает обработку большинства прерываний процессора. Жесткие блокировки — это откровенно патологические состояния, которые обнаруживаются относительно быстро, тогда как мягкие блокировки могут возникать даже в правильно настроенных системах, находящихся в экстремальном состоянии, например испытывающих высокую нагрузку на процессор.

В обоих случаях трассировка стека и отображение регистров процессора ("thumbstone") обычно сбрасываются на консоль. Трассировка показывает последовательность вызовов функций, которые привели к блокировке. В большинстве случаев эта трассировка достаточно подробно рассказывает о причине проблемы.

Мягкая или жесткая блокировка почти всегда является результатом аппаратного сбоя, а их наиболее распространенной причиной является сбойная память. Второй наиболее распространенной причиной мягкой блокировки является циклическая блокировка ядра (spinlock), которая удерживается слишком долго; однако эта ситуация обычно происходит только с нестандартными модулями ядра. Если вы используете какие-либо необычные модули, попробуйте разгрузить их и посмотреть, не возникнет ли проблема.

Когда происходит блокировка, обычное поведение заключается в том, чтобы система оставалась замороженной, а трассировка оставалась видимой на консоли, но в не-

которых средах предпочтительнее системная паника и, следовательно, перезагрузка. Например, в автоматизированной тестовой среде желательно избегать зависания, поэтому эти системы часто настраиваются для перезагрузки в безопасное ядро после обнаружения блокировки.

Команда **sysctl** может настроить как мягкие, так и жесткие блокировки на создание паники.

```
linux$ sudo sysctl kernel.softlockup_panic=1
linux$ sudo sysctl kernel.nmi_watchdog=1
```

Вы можете установить эти параметры при загрузке, указав их в файле **/etc/sysctl.conf**, как и любой другой параметр ядра.

Функция Linux `oops` является обобщением традиционного подхода UNIX к целостности ядра под лозунгом: "Паника при любой аномалии". Название `oops` ничего не значит; это просто английское междометие "Ой". Функция `oops` в ядре Linux может привести к панике, но не обязательно. Если ядро может устранить аномалию менее радикальным способом, например прекратить отдельный процесс, она может сделать это.

Когда вызывается функция `oops`, ядро генерирует результаты трассировки в буфере сообщений ядра, которые можно просмотреть с помощью команды **dmesg**. Причина вызова функции `oops` указана выше. Это может быть нечто вроде "невозможно обрабатывать запрос подкачки ядра на виртуальном адресе `0x000000000000`".

Вероятно, вы не будете отлаживать свое собственное ядро. Тем не менее ваши шансы привлечь интерес разработчика ядра или модуля значительно увеличиваются, если вы хорошо справляетесь с получением доступной контекстной и диагностической информации, включая результаты полной трассировки.

Самая ценная информация находится в начале результатов трассировки. Этот факт может стать проблемой после полномасштабной паники ядра. В физической системе вы можете просто перейти на консоль и просмотреть всю историю, чтобы увидеть полный дамп. Но на виртуальной машине консоль может быть окном, которое становится замороженным при панике в Linux; это зависит от гипервизора. Если результаты трассировки прокручиваются в окне и исчезают из поля зрения, вы не сможете увидеть причину сбоя.

Одним из способов минимизации вероятности потери информации является увеличение разрешения экрана консоли. Мы обнаружили, что разрешение 1280×1024 позволяет адекватно отображать полный текст большинства панических сообщений ядра.

Вы можете установить разрешение консоли, изменив файл **/etc/grub2/grub.cfg** и добавив в него инструкцию `vga=795` в качестве параметра загрузки ядра, которое вы хотите загрузить. Вы также можете установить разрешение, добавив это предложение в командную строку ядра на экране меню загрузки GRUB. Последний подход позволяет выполнять тестирование, не делая изменения постоянными.

Чтобы сделать изменение постоянным, найдите элемент меню с командой загрузки для ядра, которое вы хотите загрузить, и измените его. Например, если команда загрузки выглядит так:

```
linux16 /vmlinuz-3.10.0-229.el7.x86_64 root=/dev/mapper/centosroot
ro rd.lvm.lv=centos/root rd.lvm.lv=centos/swap crashkernel=auto
biosdevname=0 net.ifnames=0 LANG=en_US.UTF-8
```

то просто модифицируйте его, добавив в конец параметр `vga=795`:

```
linux16 /vmlinuz-3.10.0-229.el7.x86_64 root=/dev/mapper/centosroot
ro rd.lvm.lv=centos/root rd.lvm.lv=centos/swap crashkernel=auto
biosdevname=0 net.ifnames=0 LANG=en_US.UTF-8 vga=795
```

В табл. 11.6 перечислены другие параметры разрешения, которые можно установить с помощью параметра загрузки `vga`.

Таблица 11.6. Режимы экрана VGA

	Глубина цвета (биты)			
Геометрические размеры	**8**	**15**	**16**	**24**
640 x 480	769	784	785	786
800 x 600	771	787	788	789
1024 x 768	773	790	791	792
1280 x 1024	775	793	884	795
1400 x 1050	834	–	–	–
1600 x 1200	884	–	–	–

Паника ядра в системе FreeBSD



Система FreeBSD не разглашает много информации, когда ядро впадает в панику. Если вы используете универсальное ядро из производственной версии, то, сталкиваясь с регулярной паникой, лучше всего выполнить отладку ядра. Соберите заново универсальное ядро, включив в конфигурацию ядра параметр `makeoptions DEBUG=-g` и выполнив повторную загрузку с этим новым ядром. После повторной паники системы вы можете использовать команду **kgdb** для генерации трассировки стека из результирующего аварийного дампа, расположенного в каталоге **/var/crash**.

Конечно, если вы используете необычные модули ядра, и ядро не паникует, когда вы их не загружаете, это хороший показатель того, где скрывается проблема. Важное замечание: аварийные дампы имеют тот же размер, что и реальная (физическая) память, поэтому вы должны убедиться, что в каталоге **/var/crash** есть, по крайней мере, достаточно места. Однако есть способы обойти это: для получения дополнительной информации обратитесь к справочным страницам для команд **dumpon** и **savecore** и переменной `dumpdev` в файле **/etc/rc.conf**.

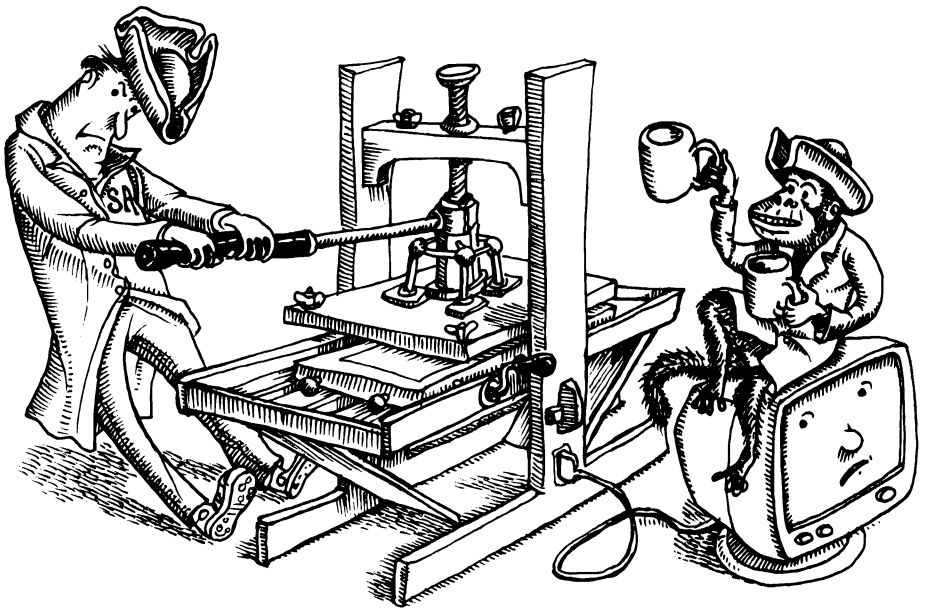
11.10. Литература

Самую свежую информацию о ядрах можно найти не сайте `lwn.net`. Кроме того, мы рекомендуем следующие книги.

- Bovet, Daniel P., and Marco Cesati. *Understanding the Linux Kernel (3rd Edition)*. Sebastopol, CA: O'Reilly Media, 2006.
- Love, Robert. *Linux Kernel Development (3rd Edition)*. Upper Saddle River, NJ: Addison-Wesley Professional, 2010. (Роберт Лав. *Ядро Linux: описание процесса разработки*, 3-е издание, пер. с англ., изд. "Диалектика", 2012 г.)
- McKusick, Marshall Kirk, et al. *The Design and Implementation of the FreeBSD Operating System (2nd Edition)*. Upper Saddle River, NJ: Addison-Wesley Professional, 2014.
- Rosen, Rami. *Linux Kernel Networking: Implementation and Theory*. Apress, 2014.

Глава 12

Печать



Печать — это вынужденная необходимость. Никто не хочет с ней связываться, но каждому пользователю когда-нибудь придется что-нибудь распечатать. К лучшему или худшему, для печати в системах UNIX и Linux обычно требуется, по крайней мере, некоторая конфигурация, а иногда и тщательный уход системного администратора.

С давних времен существовали три распространенные системы печати: BSD, System V и CUPS (Common UNIX Printing System). Сегодня Linux и FreeBSD используют CUPS, современную, сложную, сетевую и безопасную систему печати. CUPS включает в себя современный графический интерфейс на основе браузера, а также команды на уровне оболочки, которые позволяют управлять системой печати сценариями.

Прежде чем мы начнем, отметим общий момент: системные администраторы часто считают, что печать имеет более низкий приоритет, чем пользователи. Администраторы привыкли читать документы в режиме онлайн, но пользователям часто требуется печатная копия, и они хотят, чтобы система печати работала в 100% случаев. Удовлетворение этих желаний — один из самых простых способов, с помощью которых системные администраторы зарабатывают одобрение пользователей.

Печать зависит от нескольких компонентов.

- Диспетчер печати (`spool`), собирающий и планирующий задания. Слово `spool` возникло как акроним выражения Simultaneous Peripheral Operation On-Line (одновременная работа периферийных устройств в режиме "онлайн"). Теперь это всего лишь термин.

- Утилиты пользовательского уровня (интерфейсы командной строки или графические интерфейсы), передающие команды диспетчеру печати. Эти утилиты отправляют задания в диспетчер печати, запрашивают систему о заданиях (как ожидающих, так и выполненных), удаляют или перенаправляют задания и настраивают другие части системы.
- Серверные службы, управляющие самими печатающими устройствами. (Они обычно скрыты.)
- Сетевой протокол, позволяющий диспетчерам общаться и передавать задания.

В современных средах часто используются сетевые принтеры, которые минимизируют количество настроек и обработки, которые должны выполняться на стороне UNIX или Linux.

12.1. Система печати CUPS

Система CUPS была создана Майклом Свитом (Michael Sweet) и принята в качестве стандартной системы печати для Linux, FreeBSD и macOS. Майкл работает в компании Apple с 2007 г. и продолжает развивать CUPS и ее окружение.

Подобно тому, как новые системы доставки почты включают в себя команду **sendmail**, позволяя старым сценариям (и пожилым системным администраторам!) работать так, как они работали в дни былой славы **sendmail**, система CUPS поставляет традиционные команды, такие как **lp** и **lpr**, которые обратно совместимы с устаревшей печатью UNIX-системы.

Серверы CUPS — это веб-серверы, а клиенты CUPS — это веб-клиенты. Клиентами CUPS могут быть как команды (наподобие **lpr** и **lpq**), так и приложения с графическим пользовательским интерфейсом. По сути, все они являются веб-приложениями, даже если просто передают информацию демону CUPS в локальной системе. Серверы CUPS могут также функционировать как клиенты других серверов CUPS.

Сервер CUPS предоставляет полнофункциональный веб-интерфейс на порту 631. Для администраторов веб-браузер обычно является самым удобным способом для управления системой: достаточно просто перейти на адрес `http://printhost:631`. Если вам необходимо зашифрованное соединение с демоном (и ваша система позволяет это), используйте порт `https://printhost:433`. Для управления системой сценарии могут использовать дискретные команды, и пользователи могут обращаться к ним с помощью интерфейсов GNOME или KDE. Эти способы эквивалентны.

В основе всех взаимодействий между серверами CUPS и клиентами лежит протокол HTTP, точнее, протокол печати в Интернете IPP (Internet Printing Protocol), его улучшенная версия. Клиенты посылают задания с помощью операции `POST` протокола HTTP/IPP и запрашивают их статус с помощью команды `GET` из того же протокола. Файлы конфигурации CUPS также выглядят подозрительно похожими на файлы конфигурации веб-сервера Apache.

Интерфейсы для системы печати

Система CUPS является достаточно современной, так что большинство ее функций можно выполнить из графического пользовательского интерфейса, а администрирование часто осуществляется с помощью веб-браузера. Системный администратор (и, возможно, некоторые из консервативных пользователей) может также использовать команды обо-

лочки. В системе CUPS есть аналоги многих команд оболочки из классических систем печати BSD и System V. К сожалению, система CUPS не эмулирует все, без исключения, свойства этих систем. Иногда она эмулирует старые интерфейсы *слишком* хорошо; например, вместо того чтобы выдать пользователю краткую информативную справку, команды **lpr --help** и **lp --help** просто распечатывают сообщения об ошибках.

Вот как можно было бы распечатать файлы **foo.pdf** и **/tmp/testprint.pdf**.

```
$ lpr foo.pdf /tmp/testprint.ps
```

Команда **lpr** передает копии файлов серверу CUPS **cupsd**, который сохраняет их в очереди на печать (print queue). Когда принтер готов, CUPS начинает обрабатывать каждый файл по очереди.

Во время печати система CUPS проверяет как документ, так и PPD-файл принтера (PostScript Printer Description — описание принтеров в PostScript), чтобы узнать, что необходимо сделать для того, чтобы документ был распечатан должным образом. (Несмотря на свое название, PPD-файлы используются и для принтеров, не понимающих язык PostScript.)

Для того чтобы подготовить задание к печати на конкретном принтере, CUPS пропускает его через конвейер, состоящий из фильтров. Эти фильтры могут выполнять самые разные функции. Например, фильтр может изменять формат задания так, чтобы на каждой физической странице печаталось по два экземпляра уменьшенных в размере страниц, или преобразовывать задание из формата Postscript в PCL. Фильтр также может выполнять такую специфическую для принтера операцию обработки, как инициализация принтера. Фильтр даже может преобразовывать изображения в растровый формат, превращая абстрактные инструкции, такие как "проведите линию поперек страницы", в двоичное изображение. Такая функция полезна для принтеров, которые не могут самостоятельно выполнять растеризацию или не понимают язык, на котором сформулировано задание.

Последним этапом в конвейере печати является внутренний интерфейс, который отправляет задание с хоста на принтер через подходящий протокол, такой как Ethernet. Она также отправляет информацию о состоянии обратно на сервер CUPS. Передав задание на печать, демон CUPS возвращается снова к обработке своих очередей и запросов от клиентов. Принтер приступает к работе, пытаясь распечатать переданное ему задание.

Очередь на печать

Централизованное управление демона **cupsd** позволяет легко понять, как работают пользовательские команды. Например, команда **lpq** запрашивает с сервера CUPS информацию о состоянии задания и форматирует ее для отображения. Клиенты CUPS могут просить сервер откладывать задания, отменять их или изменять их приоритет. Они также могут переносить задания из одной очереди в другую.

Почти все изменения требуют указания номера задания, который сообщается в отчете, возвращаемом командой **lpq**. Например, чтобы удалить какое-нибудь задание, нужно просто выполнить такую команду: **lprm** *идентификатор_задания*.

Команда **lpstat -t** предоставляет хороший сводный отчет об общем состоянии сервера печати.

Множество принтеров

Система CUPS создает для каждого принтера отдельную очередь. Для клиентов командной строки существует аргумент (обычно **-P** *принтер* или **-p** *принтер*) для указа-

ния очереди принтера. Также можно самостоятельно задать принтер, который должен использоваться по умолчанию, просто установив переменную окружения **PRINTER**:

```
$ export PRINTER=имя_принтера
```

или указав системе CUPS использовать определенный параметр по умолчанию для данной учетной записи.

```
$ lpoptions -dимя_принтера
```

При выполнении от имени пользователя **root**, команда **lpoptions** устанавливает системные параметры по умолчанию, которые хранятся в каталоге **/etc/cups/lpoptions**. Эта команда позволяет каждому пользователю устанавливать свои параметры, которые хранятся в каталоге **~/.cups/lpoptions**. Команда **lpoptions -l** отображает список текущих опций.

Экземпляры принтеров

Если имеется только один принтер и его нужно использовать и для быстрой печати черновых вариантов, и для качественной печати производственных документов, система CUPS позволяет создавать разные "экземпляры принтера".

Например, если уже есть принтер с именем `Phaser_6120`, команда

```
$ lpoptions -p Phaser_6120/2up -o number-up=2 -o job-sheets=standard
```

создаст экземпляр с именем **Phaser_6120/2up**, печатающий по две странице на листе и добавляющий титульные страницы. После создания нового экземпляра команда

```
$ lpr -P Phaser_6120/2up biglisting.ps
```

затем распечатает файл PostScript **biglisting.ps** как задание с двумя страницами на каждом листе и соответствующими титульными страницами.

Сетевая печать

С точки зрения системы CUPS сеть со множеством машин не очень отличается от изолированной машины. На каждом компьютере выполняется демон **cupsd**, и все эти демоны CUPS взаимодействуют между собой.

Для того чтобы сконфигурировать демон CUPS так, чтобы он принимал задания на печать с удаленных систем, можно отредактировать файл **/etc/cups/cupsd.conf** (подробнее об этом будет рассказываться чуть позже в этой главе, в разделе "Настройка сервера сетевой печати"). Настроенные таким образом серверы CUPS по умолчанию каждые 30 с рассылают информацию об обслуживаемых ими принтерах. Благодаря этому имеющиеся в локальной сети компьютеры автоматически узнают о доступных им принтерах. Такую же конфигурацию можно создать, просто установив флажки в графическом пользовательском интерфейсе CUPS в вашем браузере.

Если кто-нибудь подключил новый принтер или включил в сеть ноутбук либо инсталлировал новую рабочую станцию, то с помощью команды **cupsd** можно найти и увидеть новые компоненты сети, щелкнув на кнопке Find New Printers вкладки Administration графического пользовательского интерфейса CUPS.

Сделать принтеры доступными для множества сетей или подсетей немного сложнее, потому что рассылаемые пакеты не пересекают границ подсетей. Наиболее популярным решением является выделить в каждой подсети подчиненный сервер, который будет запрашивать информацию с серверов в других подсетях и затем рассылать ее машинам в своей локальной подсети.

Предположим, необходимо, чтобы серверы `allie` (`192.168.1.5`) и `jj` (`192.168.2.14`), находящиеся в разных подсетях, были доступны пользователям в третьей подсети, `192.168.3`. Чтобы решить эту проблему, нужно просто выделить подчиненный сервер (например, `copeland`, `192.168.3.10`) и добавить в его файл **`cupsd.conf`** такие строки.

```
BrowsePoll allie
BrowsePoll jj
BrowseRelay 127.0.0.1 192.168.3.255
```

Первые две строки указывают демону **`cupsd`** подчиненного сервера запрашивать у находящихся на серверах `allie` и `jj` демонов **`cupsd`** информацию об обслуживаемых ими принтерах. Третья строка указывает серверу `copeland` рассылать получаемую информацию по своей собственной подсети. Все очень просто!

Фильтры

Вместо того чтобы использовать специализированную утилиту печати для каждого принтера, система CUPS применяет ряд фильтров, которые преобразовывают формат распечатываемого файла в формат, понятный для имеющегося принтера.

Схема фильтров в системе CUPS является довольно изящной. Когда система CUPS получает подлежащий распечатке файл, она определяет его тип MIME. Затем она проверяет файл PPD и выясняет, какие типы MIME может обрабатывать принтер. После этого с помощью файлов **`.convs`** она решает, какая цепочка фильтров может преобразовать один тип в другой и сколько это будет стоить. В заключение она выбирает цепочку и пропускает документ через фильтры. Последний фильтр в цепочке передает серверу пригодный для печати формат, а он, в свою очередь, передает данные на принтер либо через аппаратное устройство, либо через протокол, поддерживаемый принтером.

Для распознавания типа поступающих данных система CUPS использует правила, указанные в файле **`/etc/cups/mime.types`**. Например, правило

```
application/pdf       pdf string (0,%PDF)
```

означает следующее: "Если файл имеет расширение **`pdf`** или начинается со строки %PDF, тогда его типом MIME является **`application/pdf`**".

О том, как преобразовать один тип данных в другой, система CUPS узнает, просматривая правила в файле **`/etc/cups/mime.convs`**. Например, правило

```
application/pdf       application/postscript 33 pdftops
```

означает следующее: "Для того чтобы преобразовать файл **`application/pdf`** в файл **`application/postscript`**, нужно выполнить фильтр **`pdftops`**". Число 33 — это стоимость такой операции преобразования. Если выясняется, что одно и то же преобразование могут выполнить несколько цепочек фильтров, система CUPS выбирает цепочку с наименьшей общей стоимостью. (Стоимость определяется тем, кто создал файл, например производителем дистрибутивов. Мы не знаем, как. Если вы хотите уточнить этот механизм, потому что считаете, что можете сделать это лучше, значит, у вас слишком много свободного времени.)

Последним компонентом в конвейере CUPS является фильтр, который непосредственно взаимодействует с принтером. В PPD-файле принтера, не поддерживающего язык PostScript, можно увидеть строки типа

```
*cupsFilter: "application/vnd.cups-postscript 0 foomatic-rip"
```

или даже

```
*cupsFilter: "application/vnd.cups-postscript foomatic-rip"
```

Строка, заключенная в кавычки, имеет точно такой же формат, как и строка в файле **mime.convs**, но в ней указан только один тип MIME, а не два. Она сообщает о том, что фильтр **foomatic-rip** преобразовывает данные типа **application/vnd.cups-postscript** в родной формат данных принтера. Стоимость равна нулю (или вообще не указывается), потому что существует только один способ выполнить эту операцию, так что зачем притворяться, что есть еще какая-то стоимость? (PPD-файлы проекта Gutenprint для принтеров, не поддерживающих язык PostScript, немного отличаются.)

Узнать, какие фильтры доступны в системе, можно при помощи команды **locate pstops**. Фильтр **pstops** — это популярный фильтр, который выполняет с PostScript-заданиями различные манипуляции, например добавляет PostScript-команды для указания количества копий. Остальные фильтры обязательно будут где-нибудь поблизости.

Для того чтобы просмотреть список доступных серверных служб (back ends), выполните команду **lpinfo -v**. Если в используемой системе нет серверной службы для необходимого сетевого протокола, возможно, она доступна в Интернете или на сайте производителя системы Linux.

12.2. Управление сервером CUPS

Демон **cupsd** запускается во время загрузки и выполняется непрерывно. Все изученные нами разновидности системы Linux работают именно так по умолчанию.

🕮 Информацию о конфигурационном файле Apache см. в разделе 19.4.

Конфигурационный файл CUPS называется **cupsd.conf**; обычно его можно найти в каталоге **/etc/cups**. По формату он похож на конфигурационный файл Apache. Если вы умеете работать с одним из этих файлов, значит, сможете работать и с другим. Просматривать и редактировать файл **cupsd.conf** можно как с помощью текстового редактора, так и посредством графического пользовательского интерфейса CUPS.

Стандартный конфигурационный файл хорошо прокомментирован. Эти комментарии и справочная страница **cupsd.conf** настолько подробные, что мы не будем повторять ту же самую информацию в книге.

Система CUPS считывает файл конфигурации только в момент запуска. Если вы внесли изменения в конфигурационный файл **cupsd.conf**, то должны запустить систему печати снова, чтобы они вступили в силу. Если изменения вносились с помощью графического пользовательского интерфейса через веб-браузер, то они вступят в силу автоматически.

Настройка сетевого сервера печати

Если у вас возникли проблемы с сетевой печатью, просмотрите графический интерфейс CUPS на основе браузера и убедитесь, что все настройки установлены правильно. Возможные проблемные области включают неопубликованный принтер, сервер CUPS, который не передает свои принтеры в сеть, или сервер CUPS, который не принимает задания сетевой печати.

Для того чтобы заставить систему CUPS принимать задания на печать из сети, нужно внести два изменения в файл **cupsd.conf**. Сначала нужно изменить

```
<Location />
Order Deny,Allow
Deny From All
Allow From 127.0.0.1
</Location>
```

на

```
<Location />
Order Deny,Allow
Deny From All
Allow From 127.0.0.1
Allow From адрес_сети
</Location>
```

В качестве параметра *адрес_сети* следует указать IP-адрес сети, из которой должны приниматься задания на печать (например, `192.168.0.0`).

Далее нужно отыскать ключевое слово `BrowseAddress` и указать напротив него адрес рассылки в этой сети и порт CUPS.

```
BrowseAddress 192.168.0.255:631
```

Эти два действия заставят сервер принимать запросы с любой машины в сети и рассылать известную ему информацию об обслуживаемом им принтере всем имеющимся в сети демонам CUPS. Вот и все! После перезапуска демон **cupsd** станет сервером.

Автоматическое конфигурирование принтера

На самом деле система CUPS может использоваться и без принтера (например, для преобразования файлов в формат PDF или формат факса), но в основном она все-таки используется для управления принтерами. В этом разделе речь пойдет о самих принтерах.

В некоторых случаях добавление принтера является тривиальным. Система CUPS старается автоматически обнаружить USB-принтеры при их подключении и выяснить, что с ними нужно делать.

Даже если установку приходится выполнять самостоятельно, процесс добавления принтера часто состоит всего лишь из подключения оборудования, установки соединения с веб-интерфейсом CUPS по адресу `localhost:631/admin` и предоставления ответов на несколько вопросов. Среды KDE и GNOME поставляются с собственными утилитами для конфигурирования принтера, которые могут использоваться вместо интерфейса CUPS.

Если кто-то другой добавляет принтер и об этом становится известно одному или нескольким функционирующим в сети серверам CUPS, ваш сервер CUPS тоже уведомляется о появлении нового принтера. Ни добавлять этот принтер явно в локальный перечень устройств, ни копировать его PDD-файл на свою машину вам не нужно. Это магия.

Конфигурирование сетевых принтеров

Сетевым принтерам — основной частью интерфейса которых является сетевая плата Ethernet или оборудование Wi-Fi — необходима собственная конфигурация только для того, чтобы быть членами сети TCP/IP. В частности, им необходимо знать свой IP-адрес и сетевую маску. Такая информация обычно передается им одним из двух способов.

Почти все современные принтеры могут получать эту информацию по сети от сервера BOOTP или DHCP. Этот метод работает хорошо в средах со множеством гомогенных принтеров. Более подробную информацию о DHCP можно найти в разделе 13.6.

Второй способ заключается в установке статического IP-адреса при помощи консоли принтера, которая обычно состоит из ряда кнопок на передней панели и однострочной индикаторной панели. Все, что нужно сделать, — это "походить" по меню и отыскать раздел, где можно установить IP-адрес. (Если вдруг попадется команда, позволяющая распечатать меню, следует воспользоваться ею и сохранить печатную версию.)

После завершения конфигурирования сетевые принтеры обычно получают веб-консоль, доступную через браузер. Однако для того, чтобы с ними можно было работать через эту консоль, у них должен быть IP-адрес и они уже должны функционировать в сети, когда вы к ним обратитесь, так что этот интерфейс оказывается недоступным как раз тогда, когда он вам очень нужен.

Примеры конфигурирования принтеров

В качестве примеров добавим принтер с параллельным интерфейсом `groucho` и сетевой принтер `fezmo` из командной строки.

```
$ sudo lpadmin -p groucho -E -v parallel:/dev/lp0 -m pxlcolor.ppd
$ sudo lpadmin -p fezmo -E -v socket://192.168.0.12 -m laserjet.ppd
```

Как вы видите, интерфейс `groucho` предоставляется через порт **`/dev/lp0`**, а `femzo` — через IP-адрес `192.168.0.12`. Мы указываем каждое устройство в виде универсального идентификатора ресурса (URI) и выбираем PDD-файл из списка PDD-файлов, находящихся в каталоге **`/usr/share/cups/model`**.

Если локальный демон **cupsd** сконфигурирован как сетевой сервер, он сразу же сделает эти новые принтеры доступными для других клиентов в сети.

Поскольку сервер **cupsd** конфигурируется как сетевой, он немедленно делает новые принтеры доступными для других клиентов в сети. Перезапуск при этом не требуется.

CUPS позволяет использовать для принтеров самые разные URI-идентификаторы. Вот еще несколько примеров.

```
ipp://zoe.canary.com/ipp
lpd://riley.canary.com/ps
serial://dev/ttyS0?baud=9600+parity=even+bits=7
socket://gillian.canary.com:9100
usb://XEROX/Phaser%206120?serial=YGG210547
```

Одни URI-идентификаторы принимают параметры (например, `serial`), а другие — нет. Команда **`lpinfo -v`** отображает список устройств, которые система может видеть, и список типов URI-идентификаторов, которые понимает система CUPS.

Отключение принтера

Если вы хотите удалить принтер или класс, это можно легко сделать при помощи команды **`lpadmin -x`**.

```
$ lpadmin -x fezmo
```

Но как быть, если нужно только временно отключить принтер для прохождения технического осмотра, а не удалять его? В таком случае можно просто заблокировать очередь печати на входе или выходе. Если отключить "хвост" (т.е. выходную или принтерную сторону) очереди, пользователи по-прежнему смогут отправлять задания, но эти задания никогда не будут печататься. Если отключить "головную" (входную) часть очереди, те задания, которые уже находятся в очереди, будут распечатаны, а вот все попытки добавить в очередь новые задания будут отклоняться.

Команды **cupsdisable** и **cupsenable** контролируют выходную сторону очереди, а команды **reject** и **accept** — ее принимающую сторону.

```
$ sudo cupsdisable groucho
$ sudo reject corbet
```

Какую же из них лучше использовать? Принимать задания на печать, которые точно не будут распечатаны в ближайшем будущем, глупо, поэтому для длительных периодов простоя лучше использовать команду **reject**. Для коротких перерывов, которые даже не должны быть заметны пользователям (например, когда нужно просто удалить застрявшую в принтере бумагу), лучше использовать команду **cupsdisable**.

Администраторы иногда просят дать им какую-нибудь подсказку, которая помогла бы запомнить, какие команды контролируют каждый конец очереди. Предлагаем попробовать такой вариант: если система CUPS "отклоняет" (**reject**) задание, это означает, что его нельзя "предоставить". Еще один способ не путать команды — это запомнить, что приниматься (**accept**) и отклоняться (**reject**) могут только *задания на печать*, а отключаться (**disable**) и включаться (**enable**) — *только принтеры*.

Система CUPS иногда сама временно отключает принтер, с которым не может нормально работать (например, из-за того, что кто-то выдернул его кабель). Устранив проблему, следует снова активизировать очередь. Если забыть это сделать, команда **lpstat** напомнит. (Более подробную информацию по этой теме и альтернативному подходу можно найти по адресу `www.linuxprinting.org/beh.html`.)

Другие связанные с конфигурированием задачи

Современные принтеры имеют очень много опций, которые можно конфигурировать. Система CUPS позволяет настраивать самые разные функциональные возможности с помощью ее веб-интерфейса и команд **lpadmin** и **lpoptions**. Как правило, команду **lpadmin** используют для выполнения задач на уровне системы, а команду **lpoptions** — для выполнения задач на уровне пользователя.

Команда **lpadmin** позволяет более четко задать ограничения доступа, чем команды **disable** и **reject**. Например, с ее помощью можно создавать квоты на печать и указывать, на каких именно принтерах разрешается выполнять печать тем или иным пользователям.

В табл. 12.1 перечислены все команды, которые поставляются с системой CUPS, и их источники, в которых они использовались первоначально.

Таблица 12.1. Утилиты командной строки, поставляемые с системой CUPS, и системы, в которых они использовались первоначально

	Команда	Функция
CUPS	**cups-config**	Выводит информацию об API-интерфейсе, компиляторе, каталоге и канале связи системы CUPS
	cupsdisable[a]	Останавливает печать принтера или класса
	cupsenable[a]	Восстанавливает печать принтера или класса
	lpinfo	Показывает доступные устройства или драйверы
	lpoptions	Отображает или устанавливает опции и параметры по умолчанию принтера
	lppasswd	Добавляет, изменяет или удаляет пароли дайджеста

Окончание табл. 12.1

	Команда	Функция
System V	`accept, reject`	Принимает или отклоняет поступающие в очередь задания
	`cancel`	Отменяет задания
	`lp`	Печатает файлы
	`lpadmin`	Конфигурирует принтеры и классы CUPS
	`lpmove`	Перемещает задание в новое место
	`lpstat`	Печатает информацию о состоянии CUPS
BSD	`lpc`	Представляет собой общую программу для управления принтерами
	`lpq`	Отображает статус очереди принтера
	`lpr`	Печатает файлы
	`lprm`	Отменяет задания на печать

[a] На самом деле это просто переименованные команды **`disable`** и **`enable`** из System V.

12.3. Советы по выявлению проблем

Принтеры сочетают все слабости механического устройства со странностями внешней операционной системы. Они (и программное обеспечение, которое ими управляет) обожают создавать проблемы для вас и ваших пользователей. Рассмотрим несколько советов по борьбе с проблемными принтерами.

Повторный запуск демона печати

Никогда не следует забывать перезапускать демоны после внесения изменений в конфигурационный файл.

Перезапустить демон **`cupsd`** можно любым способом, который операционная система предусматривает для повторного запуска демонов, — выполнить команду **`systemctl restart org.cups.cupsd.service`** или аналогичную команду. Теоретически можно также отправить демону **`cupsd`** сигнал HUP или использовать графический пользовательский интерфейс системы CUPS.

Регистрационные журналы

Система CUPS поддерживает три регистрационных журнала: журнал страниц, журнал доступа и журнал ошибок. Журнал страниц представляет собой просто список напечатанных страниц. Журнал доступа и журнал ошибок похожи на аналогичные журналы в веб-сервере Apache. И в этом нет ничего удивительного, потому что сервер CUPS — это веб-сервер.

Уровень регистрации и путь к журнальным файлам задается в файле **`cupsd.conf`**. Обычно журнальные файлы сохраняются в каталоге **`/var/log`**.

Ниже приведен фрагмент из журнального файла, охватывающий одно задание на печать.

```
I [12/Jul/2017:18:59:08 -0600] Adding start banner page "none" to job 24.
I [12/Jul/2017:18:59:08 -0600] Adding end banner page "none" to job 24.
I [12/Jul/2017:18:59:08 -0600] Job 24 queued on 'Phaser_6120' by 'jsh'.
I [12/Jul/2006:18:59:08 -0600] Started filter usr/libexec/cups/filter/pstops
     (PID 19985) for job 24.
I [12/Jul/2017:18:59:08 -0600] Started backend /usr/libexec/cups/backend/usb
     (PID 19986) for job 24.
```

Проблемы с прямой печатью

Удостовериться в наличии физического соединения с локальным принтером можно, напрямую выполнив внутреннюю программу принтера. Например, вот что мы получим, если выполним внутреннюю программу принтера, который подключается через USB-кабель.

```
$ /usr/lib/cups/backend/usb
direct usb "Unknown" "USB Printer (usb)"
direct usb://XEROX/Phaser%206120?serial=YGG210547 "XEROX Phaser 6120"
    "Phaser 6120"
```

Если USB-кабель принтера Phaser 6120 отключится, строка для этого принтера исчезнет из вывода внутренней программы.

```
$ /usr/lib/cups/backend/usb
direct usb "Unknown" "USB Printer (usb)"
```

Проблемы с печатью в сети

Прежде чем начинать искать проблему печати в сети, попробуйте установить соединение с обслуживающим данный принтер демоном **cupsd** при помощи браузера (*имя_хоста*:631) или команды **telnet** (**telnet** *имя_хоста* **631**).

В случае возникновения проблем при настройке соединения с сетевым принтером помните о том, что на клиентском компьютере у вас должна быть очередь для заданий, способ для решения того, куда отправлять задание, и метод отправки задания на компьютер, который отвечает за обслуживание очереди на печать. На сервере печати у вас должно быть место для постановки задания в очередь, соответствующие права доступа для разрешения печати задания и способ для вывода данных на устройство.

Для того чтобы выяснить причину этих проблем, возможно, придется заглянуть в следующие файлы.

- Системные журнальные файлы на клиентском компьютере (в случае проблем с распознаванием имен и правами доступа).
- Системные журнальные файлы на сервере печати (в случае проблем с правами доступа).
- Журнальные файлы на клиентском компьютере (если отсутствуют какие-то фильтры или каталоги или если имеются неизвестные принтеры и т.д.).
- Журнальные файлы демона на сервере печати (в случае появления сообщений о неправильных именах устройств, неправильных форматах и т.д.).
- Журнальные файлы принтера на компьютере, получившем задание (в случае появления ошибок при передаче задания), как указано в переменной `lf` в файле **/etc/printcap/** в системе BSD.
- Журнальные файлы принтера на компьютере, отправившем задание (в случае появления сообщений о неправильной предварительной обработке или ошибочной постановке задания в очередь).

Путь к журнальным файлам CUPS указан в файле **/etc/cups/cupsd.conf**. Общую информацию об управлении журналами регистрации см. в главе 10.

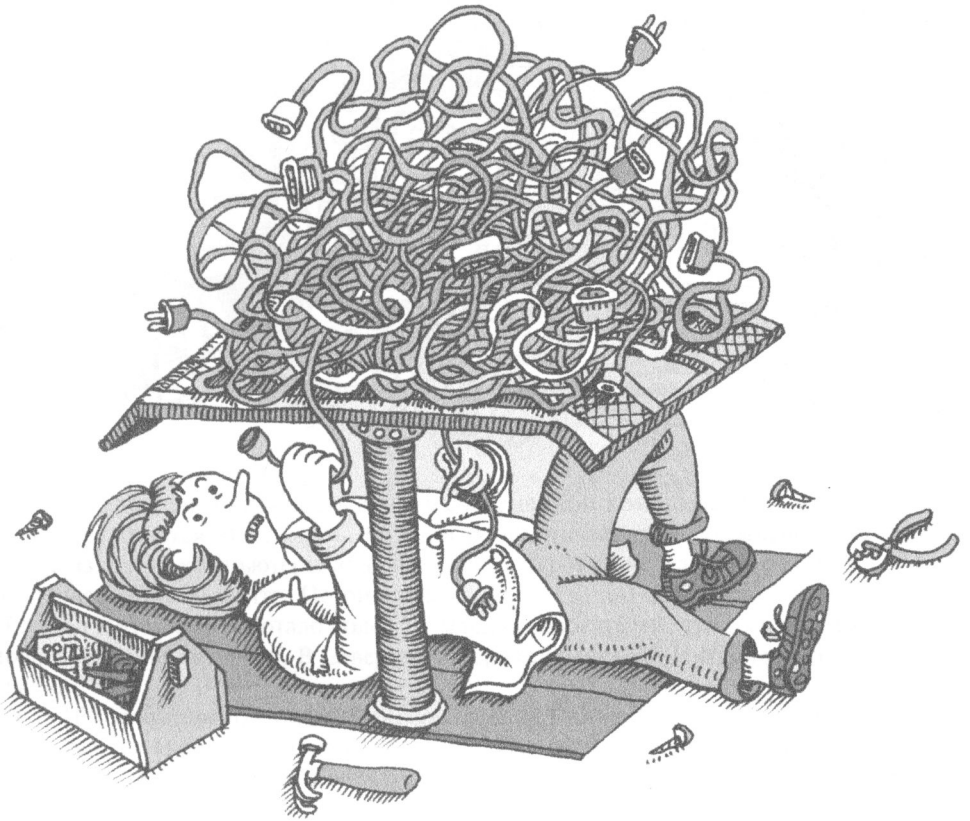
12.4. Литература

Система CUPS поставляется с обширной документацией в формате HTML. Для того чтобы получить к ней доступ, необходимо соединиться с сервером CUPS и щелкнуть на справочной ссылке. Разумеется это не поможет, ели у вас нет связи с этим сервером. Впрочем, на вашем компьютере эта документация инсталлируется в каталог **/usr/share/doc/cups** в форматах HTML и PDF. Если ее там нет, обратитесь к менеджеру, поставившему дистрибутивный пакет, или на сайт `cups.org`.

- Shah, Ankur. *CUPS Administrative Guide: A practical tutorial to installing, managing, and securing this powerful printing system*. Birmingham, UK: Packt Publishing, 2008.

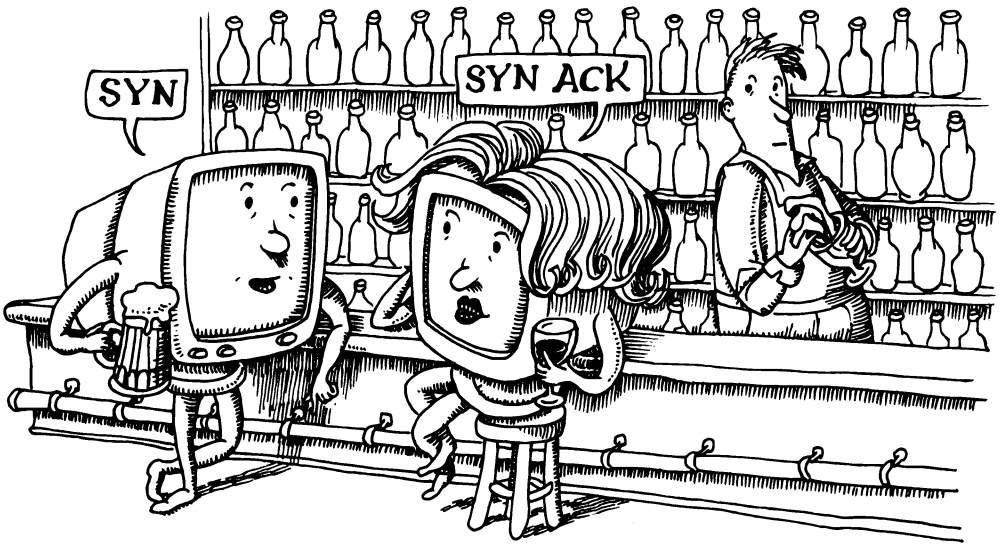
ЧАСТЬ II

Работа в сетях



Глава 13

Сети TCP/IP



Трудно переоценить важность сетей в современном компьютерном мире, хотя многие все же пытаются это сделать. Во многих организациях — возможно, даже в большинстве из них — компьютеры используются, в первую очередь, для работы в веб и доступа к электронной почте. По оценкам сайта `internetworldstat.com`, к средине 2017 года в Интернете работало более 3,7 млрд пользователей, что составляет чуть менее половины населения Земли. В Северной Америке внедрение Интернета уже приближается к 90%.

TCP/IP (Transmission Control Protocol/Internet Protocol) — это сетевая система, лежащая в основе Интернета. Она не зависит ни от аппаратного обеспечения, ни от операционной системы, поэтому все устройства, использующие протоколы TCP/IP, могут обмениваться данными ("взаимодействовать"), несмотря на различия.

Система TCP/IP работает в сетях любого размера и любой топологии, независимо от того, соединены они с внешним миром или нет. В этой главе протоколы TCP/IP рассматриваются в политическом и техническом аспектах, связанных с Интернетом, но изолированные сети на уровне протоколов TCP/IP очень похожи друг на друга.

13.1. Система TCP/IP и Интернет

История системы TCP/IP тесно связана с историей Интернета и уходит корнями на несколько десятилетий назад. Популярность Интернета во многом обусловлена элегантностью и гибкостью системы TCP/IP, а также тем, что это открытое и некоммерческое семейство протоколов. В свою очередь, широкое распространение системы TCP/IP

именно в Интернете позволило этому семейству одержать верх над несколькими конкурирующими семействами, популярными в свое время по политическим или коммерческим причинам.

Прародительницей Интернета была сеть ARPANET, организованная в 1969 году Министерством обороны США. К концу 1980-х годов эта сеть перестала быть научно-исследовательской и наступила эра коммерческого Интернета. В настоящее время Интернет представляет собой совокупность частных сетей, принадлежащих провайдерам интернет-услуг (internet service provider — ISP) и соединяющихся в так называемых "точках обмена трафиком" (peering points).

Кто управляет Интернетом

Проектирование Интернета и его протоколов всегда осуществлялось на кооперативных и открытых началах, но его точная структура изменялась по мере того, как Интернет превращался в инструмент открытого пользования и движущую силу мировой экономики. В настоящее время управление Интернетом разделено на административное, техническое и политическое, но границы между этими функциями часто размыты. Основными действующими лицами в управлении Интернетом являются следующие организации.

- ICANN (Internet Corporation for Assigned Names and Numbers — Корпорация по назначению имен и адресов в Интернете). Эта группа с наибольшим правом может быть названа ответственной за Интернет. Только она обладает всеми реальными полномочиями. Корпорация ICANN контролирует выделение адресов и доменных имен в Интернете, а также другие аспекты его деятельности, например номера протокольных портов. Эта организация, основанная как некоммерческая корпорация, расположена в Калифорнии (`www.icann.org`).
- ISOC (Internet Society — Сообщество пользователей Интернета) — открытая членская организация, представляющая интересы пользователей Интернета. Несмотря на то что эта организация преследует образовательные и политические цели, она широко известна как прикрытие для технического развития Интернета. В частности, организация ISOC является головной организацией по отношению к проблемной группе проектирования Интернета (`ietf.org`), контролирующей всю техническую работу. ISOC является международной некоммерческой организацией. Ее офисы расположены в Вашингтоне, округ Колумбия, и Женеве (`www.isoc.org`).
- IGF (Internet Governance Forum) — это организация, созданная Организацией Объединенных Наций относительно недавно, в 2005 году, чтобы предоставить возможность для проведения международных и политических дискуссий, посвященных Интернету. Она проводит ежегодные конференции, но ее роль со временем возрастает, поскольку правительства разных стран пытаются установить более жесткий контроль за Интернетом (`intgovforum.org`).

Среди перечисленных организаций наибольшая ответственность лежит на ICANN: она выступает в качестве руководящего органа Интернета, исправляя ошибки, допущенные в прошлом, и продумывая пути дальнейшего развития Интернета с учетом потребностей пользователей, правительств и бизнеса.

Сетевые стандарты и документация

Если вы не уснули сразу после того, как прочитали название этого раздела, значит, вы выпили несколько чашек кофе. Тем не менее умение разбираться в технической документации, выпускаемой руководящими органами Интернета, чрезвычайно важно для сетевых администраторов, и это не так уж и скучно, как кажется на первый взгляд.

Техническая деятельность сообщества пользователей Интернета находит отражение в серии документов, известных как RFC (Request For Comments — запрос комментариев). В формате документов RFC публикуются стандарты протоколов, предлагаемые нововведения, а также информационные бюллетени. Получая сначала статус черновых проектов (Internet Drafts), после бурных дебатов в телеконференциях и на форумах IETF они либо отвергаются, либо получают официальный статус. Свое мнение по поводу предлагаемого стандарта может высказать любой участник обсуждения. Иногда документ RFC представляет собой не стандартизацию протоколов Интернета, а всего лишь констатацию или объяснение некоторых аспектов современной практики.

Документы RFC имеют порядковые номера. На сегодняшний день их более 8200. У каждого документа есть описательное название (например, *Algorithms for Synchronizing Network Clocks* — алгоритмы сетевой синхронизации часов), но во избежание неоднозначности на документы чаще всего ссылаются по номерам. Будучи опубликованным, документ RFC никогда не меняется. Изменения и дополнения публикуются в виде новых документов с собственными номерами. Обновление может либо дополнять и разъяснять документ RFC, либо полностью его заменять.

Существует множество источников, распространяющих документы RFC, но сайт `tfc-editor.org` является центральным диспетчерским пунктом и всегда содержит самую свежую информацию. Прежде чем тратить время на чтение документа RFC, выясните его статус на сайте `tfc-editor.org`; возможно, этот документ уже не содержит наиболее актуальную информацию.

Процесс публикации стандартов Интернета подробно описан в документе RFC2026. Другой полезный метадокумент — RFC5540, *40 Years of RFCs* (40 лет существования документов RFC). В нем описаны культурные и технические аспекты публикации документов RFC.

Пусть читателей не пугает обилие технических подробностей в документах RFC. В большинстве из них содержатся вводные описания, резюме и толкования, которые будут полезны системным администраторам. Некоторые документы специально написаны в виде обзора или общего введения. Чтение документов RFC — это, возможно, не самый простой способ разобраться в той или иной теме, но это авторитетный, лаконичный и бесплатный источник информации.

Не все документы RFC написаны сухим техническим языком. Встречаются документы развлекательного содержания (часть из них опубликована первого апреля), среди них нам особенно нравятся следующие:

- RFC1149 — *A Standard for the Transmission of IP Datagrams on Avian Carriers* (стандарт передачи датаграмм с помощью птиц)[1];
- RFC1925 — *The Twelve Networking Truths* (12 сетевых истин);
- RFC3251 — *Electricity over IP* (передача электричества по протоколу IP);

[1]Группа энтузиастов системы Linux из города Берген в Норвегии, называющая себя BLUG (Bergen Linux User Group), действительно реализовала протокол CPIP (Carrier Pigeon Internet Protocol — межсетевой протокол голубиной почты) в соответствии с документом RFC1139. Детали см. на их веб-сайте по адресу: *blug.linux.no/rfc1149*.

- RFC4041 — *Requirements for Morality Section in Routing Area Drafts* (моральный кодекс маршрутизатора);
- RFC6214 — *Adaptation of RFC1149 for IPv6* (адаптация RFC1149 к протоколу IPv6);
- RFC6921 — *Design Considerations for Faster-Than-Light Communications* (вопросы проектирования систем связи, работающих со сверхсветовой скоростью);
- RFC7511 — *Scenic Routing for IPv6* (живописная маршрутизация для Iv6).

Помимо собственных порядковых номеров, документам RFC могут назначаться номера серий FYI (For Your Information — к вашему сведению), BCP (Best Current Practice — лучший существующий подход) и STD (Standard — стандарт). Перечисленные серии являются подмножествами серии RFC, группирующими документы по важности или назначению.

Документы FYI — это вводные или информационные материалы, предназначенные для широкой аудитории. Как правило, именно с них лучше всего начинать изучать незнакомую тему. К сожалению, эта серия в последнее время стала иссякать, и сейчас современных документов FYI очень немного.

Документы BCP описывают рекомендуемые процедуры для администраторов вебсайтов в Интернете. Они содержат административные предписания и представляют большую ценность для системных администраторов.

Документы STD содержат описания протоколов Интернета, прошедших процедуру проверки и тестирования в IETF и формально принятых в качестве стандартов.

Нумерация документов в рамках серий RFC, FYI, STD и BCP ведется раздельно, поэтому один документ может иметь несколько номеров. Например, документ RFC1713, *Tools for DNS Debugging* (инструменты для отладки системы DNS) известен также под номером FYI127.

13.2. Основы работы в сети

Завершив краткий обзор, давайте подробнее рассмотрим, что собой представляют протоколы TCP/IP. TCP/IP — это семейство сетевых протоколов, ориентированных на совместную работу. В состав семейства входит несколько компонентов:

- IP (Internet Protocol — межсетевой протокол) обеспечивает передачу пакетов данных с одного компьютера на другой (RFC791);
- ICMP (Internet Control Message Protocol — протокол управляющих сообщений в Интернете) отвечает за различные виды низкоуровневой поддержки протокола IP, включая сообщения об ошибках, вспомогательные маршрутизирующие запросы и отладочные сообщения (RFC792);
- ARP (Address Resolution Protocol — протокол преобразования адресов) обеспечивает трансляцию IP-адресов в аппаратные адреса (RFC826)[2];
- UDP (User Datagram Protocol — протокол передачи датаграмм пользователя) обеспечивает непроверяемую одностороннюю доставку данных (RFC768);
- TCP (Transmission Control Protocol — протокол управления передачей) обеспечивает надежный дуплексный канал связи между процессами на двух компьютерах с возможностью управления потоками и контроля ошибок (RFC793).

[2]Мы немного грешим против истины, утверждая, что протокол ARP входит в семейство TCP/IP. На самом деле он вполне может использоваться вместе с другими семействами протоколов. Просто этот протокол является неотъемлемой частью стека TCP/IP в большинстве локальных сетей.

Эти протоколы образуют иерархию (или стек), в которой протокол верхнего уровня использует протокол нижележащего уровня. Систему TCP/IP обычно описывают в виде пятиуровневой структуры (рис. 13.1), но реальная система TCP/IP содержит только три из этих уровней.

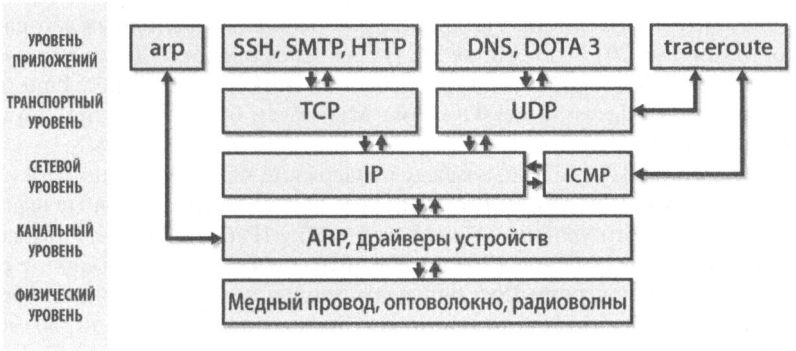


Рис. 13.1. Семейство протоколов TCP/IP

Версии IPv4 и IPv6

В течение последних трех десятилетий широкое распространение получила четвертая версия протокола TCP/IP, известная также под именем IPv4. В ней используются четырехбайтовые IP-адреса. Современная версия, IPv6, расширила адресное пространство до 16 байт, а также учла опыт использования версии IPv4. В нее не были включены возможности протокола IPv4, которые оказались не очень нужными. Это позволило ускорить работу протокола и облегчило его реализацию. Кроме того, версия IPv6 объединила системы безопасности и аутентификации в рамках одного базового протокола.

Все современные операционные системы и многие сетевые устройства уже поддерживают протокол IPv6. Компания Google публикует статистику использования протокола IPv6 своими клиентами на сайте `google.com/ipv6`. По состоянию на март 2017 года доля пользователей протокола IPv6, обращавшихся к серверам компании Google, составила около 14%. В США их доля составляет 30%.

Хотя эти числа и выглядят правдоподобными, на самом деле они могут вводить в заблуждение, потому что большинство мобильных устройств по умолчанию используют протокол IPv6 при передаче данных по сети. В то же время домашние и корпоративные сети преимущественно используют протокол IPv4.

Разработка и развертывание протокола IPv6 были в большой степени мотивированы беспокойством о том, что четырехбайтное адресное пространство протокола IPv4 практически исчерпано. Это действительно так: в настоящее время свободные IPv4-адреса остались только в Африке (см. сайт `ipv4.potaroo.net`). Первым свободные IPv4 адреса исчерпал Азиатско-Тихоокеанский регион еще 19 апреля 2011 года.

Возникает вопрос: если все IPv4-адреса в мире исчерпаны, то почему эта версия протокола остается доминирующей?

По большей части это объясняется тем, что мы научились более эффективно использовать адреса IPv4, которые у нас есть. Протокол NAT (Network Address Translation — трансляция сетевых адресов, см. раздел 13.4) позволяет целым сетям машин скрываться за одним адресом IPv4. Технология CIDR (Classless Inter-Domain Routing — бесклассовая междоменная маршрутизация, см. раздел 13.4) гибко разделяет сети и способствует эф-

фективной магистральной маршрутизации. Конфликт адресов IPv4 по-прежнему существует, но, как и диапазон вещательных частот, в наши дни они, как правило, перераспределяются по экономическим, а не технологическим критериям.

Основная проблема, которая ограничивает принятие версии IPv6, заключается в том, что поддержка версии IPv4 остается обязательной для того, чтобы устройство было полноценным элементом Интернета. Например, вот несколько основных веб-сайтов, которые по состоянию на 2017 год еще не доступны через протокол IPv6: Amazon, Reddit, eBay, IMDB, Hotmail, Tumblr, MSN, Apple, The New York Times, Twitter, Pinterest, Bing[3], WordPress, Dropbox, Craigslist, Stack Overflow. Мы могли бы продолжить, но полагаем. что вы поняли намек.[4]

Ваш выбор не между IPv4 и IPv6, а между поддержкой исключительно IPv4 и одновременной поддержкой как IPv4, так и IPv6. Когда все службы, перечисленные выше, и многие другие службы второго уровня, добавят поддержку IPv6, вы сможете трезво рассмотреть возможность использования IPv6 вместо IPv4. До тех пор будет вполне разумным попросить разработчиков версии IPv6 оправдать усилия по его внедрению, обеспечивая лучшую производительность, безопасность или функциональность. Или, возможно, открыть дверь в мир услуг IPv6, которые просто невозможно получить через IPv4.

К сожалению, этих услуг не существует, и IPv6 фактически не предлагает каких-либо из этих преимуществ. Да, это элегантный и хорошо продуманный протокол, который улучшает IPv4. И да, в некотором смысле им проще управлять, чем IPv4, и он требует меньше усилий (например, меньше зависит от технологии NAT). Но, в конце концов, это просто улучшенная версия IPv4 с большим адресным пространством. Тот факт, что вы должны управлять им вместе с IPv4, исключает любое потенциальное повышение эффективности. Причиной существования протокола IPv6 остается опасение исчерпания IPv4-адресов, и на сегодняшний день последствия этого исчерпания просто не были достаточно болезненными, чтобы стимулировать широкомасштабный переход на IPv6.

Мы издаем эту книгу в течение длительного времени, и в последних нескольких изданиях мы писали, что протокол IPv6 находится всего в одном шаге от того, чтобы стать основной технологией. В 2017 году мы испытываем сверхъестественное чувство дежавю, связанное с протоколом IPv6, который все ярче сияет на горизонте, но все еще не решает никаких проблем и предлагает слишком мало стимулов для преобразования. IPv6 — это будущее сетей, и, очевидно, так будет всегда.

Аргументы в пользу фактического развертывания IPv6 внутри вашей сети остаются прежними: в какой-то момент это придется сделать. Протокол IPv6 превосходен с инженерной точки зрения. Вам необходимо получить опыт работы с IPv6, чтобы вас не застал врасплох его приход. Все крутые парни должны это уметь.

Мы говорим: "Конечно, вперед, поддерживайте IPv6, если вам так хочется. Это ответственный и перспективный путь. Это ваш гражданский долг — осваивая IPv6, вы приближаете тот день, когда протокол IPv6 станет единственным, с чем вам придется иметь дело. Но если вам не нравится разбираться в его деталях, ничего страшного. У вас впереди годы, пока возникнет реальная необходимость перехода".

Разумеется, ни один из этих комментариев не имеет силы, если ваша организация предлагает публичные услуги в Интернете. В этом случае внедрение IPv6 — ваша свя-

[3]Присутствие сайта Bing компании Microsoft в этом списке особенно интересно, учитывая, что это один из нескольких крупных сайтов, представленных в материалах Всемирной маркетинговой кампании IPv6 Launch 2012 (под девизом «На этот раз это реально»). Мы не знаем полной истории этой ситуации, но сайт Bing, очевидно, поддерживал IPv6 в какой-то момент, а затем компания решила, что это не стоит возникающих проблем. См. `Worldipv6launch.org`.

[4]Это сайты, чьи первичные веб-адреса не связаны ни с одним адресом IPv6 (записью **AAAA**) в DNS.

тая обязанность. Не слушайте тех, кто продолжает отговаривать вас от перехода на IPv6. Кем вы хотите быть: Google или Microsoft Bing?

Кроме того, есть аргументы в пользу внедрения протокола IPv6 в центрах обработки данных, где прямая связь с внешним миром через IPv4 не требуется. В этих ограниченных средах у вас действительно есть возможность перейти на IPv6 и оставить IPv4 позади, тем самым упростив свою инфраструктуру. Серверы, ориентированные на Интернет, могут общаться с помощью протокола IPv4, даже если они маршрутизируют весь внутренний и внутренний трафик через IPv6.

В заключение приведем несколько дополнительных соображений.

- Протокол IPv6 уже давно готов к производству. Ошибки реализации не являются серьезной проблемой. Ожидается, что он будет работать так же надежно, как и IPv4.
- С точки зрения аппаратного обеспечения поддержка протокола IPv6 должна считаться обязательной для всех новых приобретаемых устройств. Сомнительно, что в наши дни вы можете найти какой-либо комплект сетевого оборудования корпоративного уровня, который не поддерживает IPv6, но многие потребительские устройства по-прежнему используют IPv4.

В книге мы сосредоточились на протоколе IPv4, поскольку именно он является основной версией протокола TCP/IP. Все, что касается версии IPv6, мы отговариваем отдельно. К счастью для системных администраторов, версии IPv4 и IPv6 очень похожи. Если вы знаете протокол IPv4, то вы знаете бо́льшую часть того, что вам следует знать о протоколе IPv6. Основное различие между этими версиями заключается в том, что они используют разные схемы адресации. В версии IPv6 использовано несколько новых концепций адресации и несколько новых обозначений. Вот и все.

Пакеты и их инкапсуляция

Система TCP/IP располагает средствами поддержки целого ряда физических сетей и транспортных систем, включая технологии Ethernet, Token Ring, MPLS (Multiprotocol Label Switching), беспроводную технологию Ethernet, а также системы с последовательными соединениями. Управление аппаратными устройствами осуществляется на канальном уровне архитектуры TCP/IP, а протоколам более высоких уровней неизвестно, как именно используются аппаратные средства.

Данные передаются по сети в виде пакетов, которые имеют максимальный размер, определяемый ограничениями канального уровня. Каждый пакет состоит из заголовка и полезного содержимого. Заголовок содержит сведения о том, откуда прибыл пакет и куда он направляется. В него могут также включаться контрольные суммы, информация, характерная для конкретного протокола, и другие инструкции, касающиеся обработки пакета. Полезное содержимое — это данные, подлежащие пересылке.

Название базового блока передачи данных зависит от уровня протокола. На канальном уровне это кадр, или фрейм, в протоколе IP — пакет, а в протоколе TCP — сегмент. Мы будем придерживаться универсального термина "пакет".

Готовящийся к отправке пакет передается вниз по стеку протоколов, и каждый протокол добавляет в него собственный заголовок. Сформированный пакет одного протокола становится полезным содержимым пакета, генерируемого следующим протоколом. Эта операция называется инкапсуляцией. На принимающей стороне инкапсулированные пакеты восстанавливаются в обратном порядке при прохождении вверх по стеку.

Например, пакет UDP, передаваемый по сети Ethernet, упакован в трех различных "конвертах". В среде Ethernet он "вкладывается" в простой физический фрейм, заго-

ловок которого содержит сведения об аппаратных адресах отправителя и ближайшего получателя, длине фрейма и его контрольной сумме (CRC). Полезным содержимым Ethernet-фрейма является IP-пакет. Полезное содержимое IP-пакета — это UDP-пакет, и, наконец, полезное содержимое UDP-пакета состоит из собственно данных, подлежащих передаче. Компоненты такого пакета изображены на рис. 13.2.

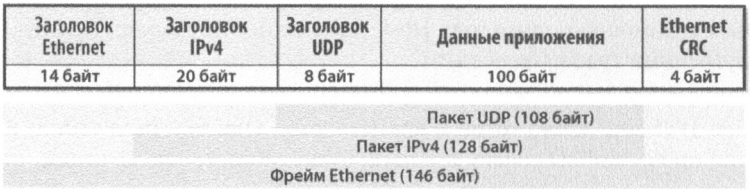


Рис. 13.2. Стандартный сетевой пакет

Стандарты формирования фреймов Ethernet

На канальном уровне к пакетам добавляются заголовки и между ними вставляются разделители. Заголовки содержат информацию об адресах канального уровня и контрольные суммы, а разделители позволяют принимающей стороне понять, где заканчивается один пакет и начинается другой. Процесс добавления вспомогательных битов называется формированием фреймов или кадровой разбивкой.

На самом деле канальный уровень разделен на две части: MAC (подуровень Media Access Control) и LLC (подуровень Link Layer Control). Подуровень MAC работает с аудиовизуальной информацией и передает пакеты по проводам. Подуровень LLC формирует фреймы.

В настоящее время существует единственный стандарт формирования фреймов по технологии Ethernet: DIX Ethernet II. Кроме того, по историческим причинам используется еще несколько стандартов, основанных на стандарте IEEE 802.2, особенно в сетях Novell.

Максимальный размер передаваемого блока

Размер сетевых пакетов ограничивается как характеристиками аппаратных средств, так и требованиями протоколов. Например, объем полезного содержимого стандартного Ethernet-фрейма не может превышать 1500 байт. Предельный размер пакета устанавливается на канальном уровне и называется максимальной единицей передачи (Maximum Transfer Unit — MTU). Стандартные значения параметра MTU для разных видов сетей приведены в табл. 13.1.

Таблица 13.1. Максимальные размеры передаваемых блоков в сетях различных типов

Тип сетевого соединения	Максимальная единица передачи
Ethernet	1500 байт (1492 в спецификации 802.2) [a]
IPv6 (на любом аппаратном обеспечении)	Не менее 1280 байт на уровне IP
Token Ring	Конфигурируется [б]
Двухточечные WAN-соединения	Конфигурируется, обычно 1500 или 45006 байт
Двухточечные каналы ГВС (T1, T3)	Конфигурируется, обычно 1500 или 4500 байт

[a] Дополнительную информацию об Ethernet-пакетах "jumbo" см. в разделе 14.1.

[б] Распространенные значения таковы: 552, 1064, 2088, 4508 и 8232. Иногда выбирается значение 1500 для совместимости с Ethernet.

Протокол IPv4 разделяет пакеты на такие фрагменты, чтобы их размер соответствовал требованиям к MTU конкретного сетевого соединения. Если пакет проходит через несколько сетей, в одной из них параметр MTU может оказаться меньшим, чем в исходной сети. В этом случае маршрутизатор, пересылающий пакет в сеть с меньшим значением MTU, подвергнет пакет дальнейшей фрагментации.

Фрагментация "на лету" нежелательна в условиях сильной загруженности маршрутизаторов. В протоколе IPv6 эта возможность устранена. Пакеты по-прежнему могут фрагментироваться, но эту задачу должен выполнять вызывающий хост.

В рамках протокола IPv4 отправитель может обнаружить канал, способный пропустить пакет с наименьшим показателем MTU, установив на пакете флаг "не фрагментировать". Если пакет достигнет промежуточного маршрутизатора, который не способен его пропустить, этот маршрутизатор вернет отправителю сообщение об ошибке ICMP. Пакет ICMP содержит показатель MTU сети, требующей пакеты меньшего размера, и этот показатель затем становится ориентировочным размером для пакетов, отправляемых по данному адресу.

Обнаружение MTU-пути в протоколе IPv6 осуществляется аналогично, но поскольку промежуточным маршрутизаторам никогда не разрешается фрагментировать пакеты IPv6, все пакеты IPv6 действуют так, как будто у них включен флаг "не фрагментировать". Любой пакет IPv6, который слишком велик для размещения в нисходящем канале, вызывает отправку ICMP-сообщения отправителю.

В протоколе TCP путь MTU определяется автоматически, даже в версии IPv4. Протокол UDP такой возможности не имеет и перекладывает дополнительную работу на уровень IP.

Проблема фрагментации в протоколе Iv4 может оказаться достаточно коварной. Несмотря на то что выявление пути MTU должно автоматически разрешить конфликты, иногда администратору приходится вмешиваться. Например, в виртуальной частной сети с туннельной структурой необходимо проверять размер пакетов, проходящих через туннель. Обычно их начальный размер — 1500 байт, но когда к ним добавляется туннельный заголовок, размер пакетов становится равным примерно 1540 байт и уже требуется фрагментация. Уменьшение максимального размера передаваемого блока позволяет избежать фрагментации и повысить производительность туннельной сети. Просмотрите **справочные** страницы команд `ifconfig` или `ip-link`, чтобы узнать, как настроить параметр MTU сетевой платы.

13.3. Адресация пакетов

Подобно письмам и сообщениям электронной почты, сетевые пакеты могут достичь пункта назначения только при наличии правильного адреса. В системе TCP/IP используется сочетание нескольких схем адресации.

- Адреса MAC (media access control) для использования в сетевом оборудовании.
- Сетевые адреса протоколов IPv4 и IPv6 для использования в программном обеспечении.
- Имена компьютеров для использования пользователями.

Аппаратная адресация (MAC)

Каждый сетевой интерфейс компьютера имеет один MAC-адрес канального уровня, который отличает его от других компьютеров в физической сети, а также один или

несколько IP-адресов, идентифицирующих интерфейс в глобальной сети Интернета. Последнее утверждение стоит повторить: IP-адрес идентифицирует *сетевые интерфейсы, а не машины*. (Для пользователей это различие не имеет значения, но администраторы должны об этом знать.)

Самый нижний уровень адресации задается сетевыми аппаратными средствами. Например, Ethernet-устройствам в процессе изготовления назначаются уникальные шестибайтовые аппаратные адреса. Эти адреса традиционно записываются в виде ряда двухцифровых шестнадцатеричных байтов, разделенных двоеточиями, например `00:50:8d:9a:3b:df`.

Сетевые платы Token Ring также имеют шестибайтовые адреса. В некоторых сетях с двухточечным соединением (например, в PPP-сетях) аппаратные адреса вообще не нужны: адрес пункта назначения указывается непосредственно при установке соединения.

Шестибайтовый Ethernet-адрес разбивается на две части: первые три байта определяют изготовителя устройства, а последние три — выступают в качестве уникального серийного номера, назначаемого изготовителем. Системные администраторы могут выяснить марку устройства, вызывающего проблемы в сети, поискав трехбайтовый идентификатор соответствующих пакетов в таблице идентификаторов изготовителей. Трехбайтовые коды на самом деле представляют собой идентификаторы OUI (Organizationally Unique Identifier — уникальный идентификатор организации), присваиваемые организацией IEEE, поэтому их можно найти непосредственно в базе данных IEEE по адресу `standards.ieee.org/regauth/oui`.

Разумеется, отношения между производителями микросхем, компонентов и системы носят сложный характер, поэтому идентификатор изготовителя, закодированный в MAC-адресе, может ввести пользователя в заблуждение.

Теоретически аппаратные адреса Ethernet должны назначаться на постоянной основе и оставаться неизменными. К сожалению, некоторые сетевые платы допускают программное задание аппаратных адресов. Это удобно при замене испорченных компьютеров или сетевых карт, MAC-адрес которых менять по тем или иным причинам нежелательно (например, если его фильтруют все ваши коммутаторы, если ваш DHCP-сервер выдает адреса на основе MAC-адресов или MAC-адрес был использован как лицензионный ключ для программного обеспечения). Фальсифицируемые MAC-адреса могут также оказаться полезными, если вам необходимо проникнуть в беспроводную сеть, использующую механизм управления доступом на основе MAC-адресов. Однако, чтобы не усложнять ситуацию, мы рекомендуем сохранять уникальность MAC-адресов.

IP-адресация

🕮 Дополнительную информацию о технологии NAT и пространствах частных адресов см. в разделе 13.4.

На следующем, более высоком, уровне используется интернет-адресация (чаще называемая IP-адресацией). IP-адреса аппаратно независимы. В любом конкретном сетевом контексте IP-адрес идентифицирует конкретное и уникальное место назначения. Тем не менее не совсем точно говорить, что IP-адреса глобально уникальны, потому что существует несколько исключений: NAT использует один IP-адрес интерфейса для обработки трафика для нескольких машин; пространства частных адресов IP — это адреса, которые могут использовать сразу в нескольких локальных сетях,

если эти адреса не видны в Интернете; альтернативная адресация назначает один IP-адрес нескольким машинам одновременно.

Соответствие между IP-адресами и аппаратными адресами устанавливается на канальном уровне модели TCP/IP. В сетях, поддерживающих широковещательный режим (т.е. в сетях, позволяющих адресовать пакеты "всем компьютерам данного физического сегмента"), протокол ARP обеспечивает автоматическую привязку адресов без вмешательства системного администратора. В протоколе IPv6 MAC-адреса интерфейсов можно использовать как часть IP-адресов, благодаря чему преобразование IP-адресов в аппаратные адреса становится практически автоматическим.

🕮 Подробнее о протоколе ARP рассказывается в разделе 13.5.

"Адресация" имен машин

🕮 Дополнительную информацию о системе DNS см. в главе 16.

Поскольку IP-адреса представляют собой длинные числа, запоминать их трудно. Операционные системы позволяют закреплять за IP-адресом одно или несколько текстовых имен, чтобы вместо **4.31.198.49** пользователь мог ввести **rfc-editor.org**. В системах UNIX и Linux это отображение можно осуществить с помощью статического файла (**/etc/hosts**), базы данных LDAP и, наконец, DNS (Domain Name System) — глобальной системы доменных имен. Следует помнить о том, что имя компьютера — это лишь сокращенный способ записи IP-адреса, и он относится к сетевому интерфейсу, а не компьютеру.

Порты

IP-адреса идентифицируют сетевые интерфейсы компьютера, но они недостаточно конкретны для идентификации отдельных процессов и служб, многие из которых могут активно использоваться в сети одновременно. Протоколы TCP и UDP расширяют концепцию IP-адресов, вводя понятие *порта*. Порт представляет собой 16-разрядное число, добавляемое к IP-адресу и указывающее конкретный канал взаимодействия. Его значение варьируется в диапазоне от 1 до 65535.

Стандартные службы, такие как SMTP, SSH и HTTP, ассоциируются с хорошо известными портами, определенными в файле **/etc/services**. Вот некоторые типичные записи из файла **services**.

```
...
smtp        25/udp                    # Simple Mail Transfer
smtp        25/tcp                    # Simple Mail Transfer
...
domain      53/udp                    # Domain Name Server
domain      53/tcp                    # Domain Name Server
...
http        80/udp    www www-http    # World Wide Web HTTP
http        80/tcp    www www-http    # World Wide Web HTTP
...
kerberos    88/udp                    # Kerberos
kerberos    88/tcp                    # Kerberos
...
```

Файл **services** является частью инфраструктуры. Вам не нужно изменять его, хотя вы можете сделать это, если хотите добавить нестандартную службу. Полный список назначенных портов вы можете найти по адресу **iana.org/assignments/port-numbers**.

Хотя в протоколах TCP и UDP предусмотрены порты, и эти порты имеют одинаковые наборы потенциальных значений, их пространства портов полностью разделены и не связаны друг с другом. Брандмауэры должны быть настроены отдельно для каждого из этих протоколов.

Чтобы предотвратить подмену системных служб, системы UNIX ограничивают программы привязкой к номерам портов до 1024, если они не выполняются с правами root или не имеют соответствующих возможностей Linux. Любой может общаться с сервером, работающим на низком номере порта; ограничение распространяется только на программу, прослушивающую порт.

Сегодня привилегированная портовая система в равной степени неудобна и ненадежна. Во многих случаях более безопасно запускать стандартные службы на непривилегированных портах в качестве непривилегированных пользователей и перенаправлять сетевой трафик на эти высокопроизводительные порты через балансировщик нагрузки или какой-либо другой тип сетевого устройства. Эта практика ограничивает распространение ненужных прав root и добавляет дополнительный уровень абстракции в вашу инфраструктуру.

Типы адресов

В протоколе IP поддерживается несколько типов адресов, некоторые из них имеют эквиваленты на канальном уровне.

- Направленные (unicast) — адреса, которые обозначают отдельный сетевой интерфейс.
- Групповые (multicast) — адреса, идентифицирующие группу хостов.
- Широковещательные (broadcast) — адреса, обозначающие все хосты локальной сети.
- Альтернативные (anycast) — адреса, обозначающие любой из группы хостов.

Режим группового вещания используется, к примеру, в видеоконференциях, где одна и та же последовательность пакетов посылается всем участникам конференции. Протокол IGMP (Internet Group Management Protocol — протокол управления группами хостов Интернета) отвечает за управление совокупностями хостов, идентифицируемыми как один обобщенный адресат.

Групповые адреса в настоящее время в Интернете практически не используются. Тем не менее они были использованы в протоколе IPv6, в котором широковещательные адреса, по существу, представляют собой специализированную форму групповой адресации.

Альтернативные адреса обеспечивают балансирование нагрузки на канальный уровень сети, разрешая доставлять пакеты в ближайший из нескольких пунктов назначения в смысле сетевой маршрутизации. Можно было ожидать, что эти адреса будут напоминать групповые, но фактически они больше похожи на направленные адреса.

Большинство деталей механизма альтернативных адресов скрыто на уровне маршрутизации, а не на уровне протокола IP. Благодаря альтернативной адресации произошло реальное ослабление традиционных требований, чтобы IP-адреса однозначно идентифицировали пункт назначения. С формальной точки зрения альтернативная реализация предназначена для протокола IPv6, но аналогичный трюк можно осуществить и в протоколе IPv4, например, так, как это сделано для корневых серверов имен DNS.

13.4. IP-адреса

За исключением групповых адресов, адреса Интернета состоят из двух частей: сетевой и машинной. Сетевая часть идентифицирует логическую сеть, к которой относится адрес, а машинная — хост этой сети. В протоколе IPv4 адреса состоят из четырех байтов, а граница между сетевой и машинной частями устанавливается административно. В протоколе IPv6 адреса состоят из 16 байт, а сетевая и машинная части всегда состоят из 8 байт.

В протоколе IPv4 адреса записываются в виде группы десятичных чисел (по одному на каждый байт), разделенных точками, например 209.85.171.147. Крайний слева байт — старший; он всегда относится к сетевой части адреса.

Если первым байтом адреса является число 127, то оно обозначает *интерфейс обратной связи* ("loopback network") — фиктивную сеть, не имеющую реального аппаратного интерфейса и состоящую из одного компьютера. Адрес 127.0.0.1 всегда ссылается на текущий компьютер. Ему соответствует символическое имя `localhost`. (Это еще одно небольшое нарушение требования уникальности IP-адресов, поскольку каждый компьютер интерпретирует адрес 127.0.0.1 как адрес другого компьютера, хотя этим компьютером является он сам.)

Адреса в протоколе IPv6 и их текстовые эквиваленты являются немного более сложными. Они будут рассмотрены далее в подразделе "Адресация в протоколе IPv6".

IP-адрес и другие параметры сетевого интерфейса задаются командой `ifconfig`. Она описывается в разделе 13.9.

Классы адресов в протоколе IPv4

Исторически IP-адреса группировались в *классы*, которые определялись на основании первых битов крайнего слева байта. Классы отличались распределением байтов адреса между сетевой и машинной частями. Современные маршрутизаторы используют явные маски для задания сетевой части адреса, причем компоненты адреса могут разделяться не обязательно по границе байтов. Тем не менее традиционные классы все еще используются по умолчанию, если не предоставлена явная маска.

Классы A, B и C обозначают обычные IP-адреса, а классы D и E применяются при групповой адресации и в исследовательских целях. В табл. 13.2 представлены характеристики каждого класса адресов. Сетевая часть адреса помечена буквой C, а машинная — буквой M.

Таблица 13.2. Классы IP-адресов

Класс	Первый байт[a]	Формат	Комментарии
A	1–127	C.M.M.M	Самые первые сети или адреса, зарезервированные для Министерства обороны США
B	128–191	C.C.M.M	Крупные организации, обычно с подсетями; адреса данного класса почти полностью заняты
C	192–223	C.C.C.M	Небольшие организации; адреса данного класса получить легко, они выделяются целыми блоками
D	224–239	—	Групповые адреса; не назначаются на постоянной основе
E	240–255	—	Экспериментальные адреса

[a] Значение 0 не используется в качестве первого байта обычных IP-адресов. Значение 127 зарезервировано для адресов обратной связи.

В редких случаях в состав локальной сети входит более ста компьютеров. По этой причине полезность адресов класса A или B (которые допускают наличие в одной сети соответственно 16 777 214 и 65 534 хостов) весьма сомнительна. К примеру, 127 адресов сетей класса A занимают половину доступного адресного пространства. Кто же знал, что адресное пространство протокола IPv4 станет таким ценным!

Подсети IPv4

Для того чтобы эффективнее использовать адреса, определенную долю машинной части адреса можно "одолжить" для расширения сетевой части, явно указав четырехбайтовую маску подсети ("subnet mask"), или сетевую маску ("netmask"), в которой единицы соответствуют сетевой части, а нули — машинной. Единицы должны занимать левую часть маски и следовать одна за другой без разрывов. Сетевая часть должна занимать не менее восьми битов, а машинная — не менее двух битов. Следовательно, маска подсети в соответствии с протоколом IPv4 допускает только 22 возможных значения.

Например, четыре байта адреса класса B обычно интерпретируются как C.C.M.M. Следовательно, неявная маска подсети класса B в десятичной системе выглядит как 255.255.0.0. Однако адрес с маской 255.255.255.0 можно интерпретировать как C.C.C.M. Использование такой маски превращает одну сеть класса B в 256 разных подсетей, подобных сетям класса C, т.е. в каждую из них может входить 254 компьютера.

Маски подсети, как и любой сетевой интерфейс, задаются с помощью команд `ip` или `ifconfig`. По умолчанию эти команды используют класс адреса для того, чтобы выяснить, какие биты относятся к сетевой части. Если задается явная маска, то эта функция просто отменяется.

🕮 Дополнительную информацию о командах `ip` и `ifconfig` см. в разделе 13.10.

Сетевые маски, не оканчивающиеся на границе байта, труднее декодировать. Они обычно записываются в виде суффикса */XX*, где *XX* — число битов в сетевой части адреса (длина маски). Такую запись иногда называют *нотацией* CIDR (Classless Inter-Domain Routing — протокол бесклассовой междоменной маршрутизации). Например, адрес 128.138.243.0/26 обозначает первую из четырех сетей с общим компонентом адреса 128.138.243. В трех других сетях последний байт адреса равен 64, 128 и 192. Сетевая маска, связанная с этими сетями, имеет вид 255.255.255.192 или `0xFFFFFFC0`. В двоичном представлении это 26 единиц с последующими шестью нулями (рис. 13.3).

IP-адрес	128	.	138	.	243	.	0
Десятичная сетевая маска	255	.	255	.	255	.	192
Шестнадцатиричная сетевая маска	f f	.	f f	.	f f	.	c 0
Двоичная сетевая маска	1111 1111	.	1111 1111	.	1111 1111	.	1100 0000

Рис. 13.3. Структура маски подсети в разных системах счисления

В сети с маской /26 для нумерации хостов отводится шесть битов (32 – 26 = 6). Таким образом, появляется возможность задать 64 адреса (2^6 = 64). В действительности допускается использовать лишь 62 адреса, поскольку адреса, полностью состоящие из нулей или единиц, зарезервированы (для обозначения самой сети и широковещательного режима соответственно).

В нашем примере старшие два бита последнего байта адреса могут принимать значения 00, 01, 10 и 11. Таким образом, сеть 128.138.243.0/24 может быть разделена на четыре сети /26:

- 128.138.243.0/26 (0 в десятичной системе — **00**000000 в двоичной);
- 128.138.243.64/26 (64 в десятичной системе — **01**000000 в двоичной);
- 128.138.243.128/26 (128 в десятичной системе — **10**000000 в двоичной);
- 128.138.243.192/26 (192 в десятичной системе — **11**000000 в двоичной).

Выделенные полужирным шрифтом биты последнего байта каждого адреса относятся к его сетевой части.

Трюки и инструменты для арифметических вычислений, связанных с подсетями

Манипулировать всеми этими битами в уме трудно, но есть ряд приемов, позволяющих упростить вычисления. Число хостов в сети и значение последнего байта сетевой маски в сумме всегда дают 256.

последний байт сетевой маски = 256 – *размер сети*

Так, в рассмотренном выше случае формула даст результат 256 – 64 = 192, где 192 — последний байт маски. Другое правило гласит о том, что значение последнего байта фактического адреса сети (не сетевой маски) должно нацело делиться на число хостов сети. В рассматриваемом примере последние байты равны 0, 64, 128 и 192 — каждое из этих чисел делится нацело на 64.[5]

Имея только IP-адрес (допустим, 128.138.243.100), невозможно сказать, каким будет адрес сети и широковещательный адрес. В табл. 13.3 представлены возможные варианты для масок /16 (выбирается по умолчанию для адресов класса B), /24 и /26 (наиболее приемлемая длина, если имеющееся адресное пространство невелико).

Таблица 13.3. Пример расшифровки IP-адреса

IP-адрес	Сетевая маска	Адрес сети	Широковещательный адрес
128.138.243.100/16	255.255.0.0	128.138.0.0	128.138.255.255
128.138.243.100/24	255.255.255.0	128.138.243.0	128.138.243.255
128.138.243.100/26	255.255.255.192	128.138.243.64	128.138.243.127

Сетевой (все нули в машинной части) и широковещательный (все единицы в машинной части) адреса сокращают число хостов в каждой локальной сети на 2, поэтому в самой маленькой сети формально должно быть 4 хоста: два реальных, соединенных напрямую, и два фиктивных (сетевой и широковещательный). Для того чтобы создать такую сеть, необходимо отвести два бита под машинную часть адреса, т.е. суффикс адреса будет /30, а сетевая маска — 255.255.255.252 или `0xFFFFFFFC`. Но есть еще сеть с маской /31, которая трактуется как особый случай (RFC3021): у нее нет сетевого и широковещательного адресов, а оба оставшихся адреса используются для идентификации хостов. Маска такой сети равна 255.255.255.254.

Кришан Джодис (Krischan Jodies) написал полезную программу под названием IP Calculator (она доступна по адресу `jodies.de/ipcalc`), позволяющую выполнять арифметические операции над двоичными, шестнадцатеричными числами и масками.

[5] Разумеется, нуль делится на любое число.

Программа IP Calculator делает все, что требуется пользователю при работе с сетевыми адресами и масками подсети, а также широковещательными адресами и т.п.

Кроме того, существует версия этого калькулятора **ipcalc**, которая запускается из командной строки. Она находится в стандартных репозиториях систем Debian, Ubuntu и FreeBSD.



В операционных системах Red Hat и CentOS существует совершенно независимая программа, которая тоже называется **ipcalc**. Однако она относительно бесполезна, потому что распознает только стандартные классы IP-адресов.

Ниже приведен образец работы программы **ipcalc** (формат вывода для наглядности немного изменен).

```
$ ipcalc 24.8.175.69/28
Address:    24.8.175.69              00110000.00001000.10101111.0100  0101
Netmask:    255.255.255.240 = 28     11111111.11111111.11111111.1111  0000
Wildcard:   0.0.0.255                00000000.00000000.00000000.1111  1111
=>
Network:    24.8.175.64/28           00011000.00010000.10101111.0100  0000
Broadcast:  24.8.175.65              00011000.00010000.10101111.0100  0001
HostMin:    24.8.175.78              00011000.00010000.10101111.0100  1110
HostMax:    24.8.175.79              00011000.00010000.10101111.0100  1111
Host/Net      14                      Class A
```

Эти результаты позволяют не только просто и понятно представить адреса, но и "копировать и вставлять" версии. Очень полезная программа.

Если этот калькулятор вам по каким-то причинам не подойдет, воспользуйтесь стандартной утилитой **bc**, поскольку она может выполнять арифметические операции в любой системе счисления. Установите основания систем счисления для ввода и вывода, используя директивы **ibase** и **obase**. Сначала выполните директиву **obase**, в противном случае ее результат будет интерпретироваться в зависимости от нового основания системы счисления, установленной директивой **ibase**.

CIDR: протокол бесклассовой междоменной маршрутизации

Как и подсети, прямым расширением которых он является, протокол CIDR основан на использовании явной маски подсети, определяющей границу между сетевой и машинной частями адреса. Но, в отличие от подсетей, протокол CIDR допускает, чтобы сетевая часть была *меньше*, чем того требует класс адреса. Благодаря укороченной маске возникает эффект объединения нескольких сетей для облегчения маршрутизации. По этой причине протокол CIDR иногда называют *протоколом формирования суперсетей*.

🕮 Протокол CIDR определен в документе RFC4632.

Протокол CIDR упрощает информацию о маршрутизации и устанавливает иерархию в этом процессе. Несмотря на то что протокол CIDR задумывался как временная мера для облегчения маршрутизации в рамках протокола IPv6, он оказался достаточно мощным средством для решения проблемы роста Интернета, возникшей в последнее десятилетие.

Предположим, организации предоставлен блок из восьми адресов класса C, пронумерованных последовательно от 192.144.0.0 до 192.144.7.0 (в нотации CIDR — 192.144.0.0/21). Внутри самой организации адреса могут распределяться так:

- 1 сеть с длиной маски /21 — 2046 хостов, сетевая маска 255.255.248.0;
- 8 сетей с длиной маски /24 — 254 хоста в каждой, сетевая маска 255.255.255.0;

- 16 сетей с длиной маски /25 — 126 хостов в каждой, сетевая маска 255.255.255.128;
- 32 сети с длиной маски /26 — 62 хоста в каждой, сетевая маска 255.255.255.192 и т.д.

Для перечисленных адресов не обязательно иметь 32, 16 или даже 8 записей в таблице маршрутизации. Ведь все они ссылаются на хосты одной и той же организации, поэтому пакеты предварительно нужно доставлять в общий приемный пункт. В таблицу маршрутизации достаточно внести запись 192.144.0.0/21. Протокол CIDR позволяет даже выделять фрагменты адресных пространств классов A и B, благодаря чему увеличивается число доступных адресов.

Внутри сети допускается смешение подсетей с разной длиной маски, если только они не перекрывают друг друга. Это называется формированием подсетей переменного размера. Например, интернет-провайдер, которому выделена адресная область 192.144.0.0/21, может создать группу сетей с маской /30 для коммутируемых PPP-клиентов, несколько сетей с маской /24 — для крупных клиентов и ряд сетей с маской /27 — для более мелких компаний.

Все хосты конкретной сети должны конфигурироваться с помощью одной сетевой маски. Нельзя для одного хоста задать длину маски /24, а для другого — /25.

Выделение адресов

Формально назначаются лишь адреса сетей. Организации должны самостоятельно определять номера компьютеров и закреплять за ними IP-адреса. Деление на подсети также осуществляется произвольно.

Организация ICANN в административном порядке делегировала полномочия по распределению адресов трем региональным организациям, которые предоставляют блоки адресов интернет-провайдерам в рамках своих регионов (табл. 13.4). Провайдеры, в свою очередь, выделяют адреса отдельным клиентам. Только крупные провайдеры могут напрямую посылать запросы в один из регистров, спонсируемых организацией ICAAN.

Таблица 13.4. Региональные реестры IP-адресов

Организация	Веб-адрес	Охватываемый регион
ARIN	`arin.net`	Северная часть Карибских островов
APNIC	`apnic.net`	Азия и Океания, включая Австралию и Новую Зеландию
AfriNIC	`adrinic.net`	Африка
LACNIC	`lacnic.net`	Центральная и Южная Америка, часть Карибских островов
RIPE NCC	`ripe.net`	Европа и прилегающие регионы

Делегирование полномочий организации ICANN региональным регистрам обеспечивает должную агрегацию адресов в таблицах маршрутизации магистральных сетей. Клиенты, получившие адреса от своего провайдера, не обязаны иметь маршрутные записи в магистральных серверах. Достаточно одной записи, ссылающейся на провайдера.

Частные адреса и система NAT

Другое временное решение проблемы сокращающегося адресного пространства протокола IPv4 заключается в использовании частных областей IP-адресов, описанных в документе RFC1918. Частные адреса используются только во внутренней сети предприятия и никогда не маршрутизируются в Интернет (во всяком случае, преднамеренно). Преобразование частных адресов в адреса, выделенные интернет-провайдером, осуществляется пограничным маршрутизатором.

Документ RFC1918 определяет, что одна сеть класса A, 16 сетей класса B и 256 сетей класса C резервируются для частного использования и никогда не выделяются глобально. В табл. 13.5 показаны диапазоны частных адресов (каждый диапазон представлен в более короткой нотации протокола CIDR).

Таблица 13.5. IP-адреса, зарезервированные для частного использования

Класс	Начало	Конец	Диапазон CIDR
A	10.0.0.0	10.255.255.255	10.0.0.0/8
B	172.16.0.0	172.31.255.255	172.16.0.0/12
C	192.168.0.0	192.168.255.255	192.168.0.0/16

Изначально идея состояла в том, чтобы хосты могли самостоятельно выбирать класс адресов из указанных возможностей и правильно определять свой размер. Однако в настоящее время протокол CIDR и подсети стали универсальным инструментом, поэтому для всех новых частных сетей целесообразнее всего использовать адреса класса A (разумеется, с подсетями).

Для того чтобы хосты, использующие частные адреса, могли получать доступ в Интернет, на пограничном маршрутизаторе организации должна выполняться система NAT (Network Address Translation — трансляция сетевых адресов). Эта система перехватывает пакеты и заменяет в них адрес отправителя реальным внешним IP-адресом. Может также происходить замена номера исходного порта. Система хранит таблицу преобразований между внутренними и внешними адресами/портами, чтобы ответные пакеты доставлялись нужному адресату.

Многие варианты системы NAT на самом деле выполняют преобразование адресов портов (Port Address Translation — PAT): они используют один внешний IP-адрес и соединяют его с несколькими внутренними хостами. Например, эта конфигурация по умолчанию установлена в большинстве популярных маршрутизаторов, использующих кабель и модемы. На практике реализации систем NAT и PAT очень похожи, поэтому они обе называются NAT.

Хост, использующий систему NAT, запрашивает небольшой сегмент адресного пространства у провайдера, но большинство адресов теперь используется для внутрисистемных привязок и не назначается отдельным компьютерам. Если хост позднее пожелает сменить провайдера, потребуется лишь изменить конфигурацию пограничного маршрутизатора и его системы NAT, но не самих компьютеров.

Крупные организации, использующие адреса NAT и RFC1918, должны иметь некоторую форму центральной координации, чтобы все хосты, независимо от их отдела или административной группы, имели уникальные IP-адреса. Ситуация может осложниться, когда одна компания, использующая адресное пространство RFC1918, приобретает другую компанию, которая делает то же самое, или объединяется с ней. Части объединенной организации необходимо часто перенумеровывать.

Существует возможность заставить операционные системы UNIX или Linux выполнять функции NAT, хотя во многих организациях предпочитают делегировать эти обязанности маршрутизаторам или устройствам подключения к сети.[6] Вопросы, связанные с конкретными поставщиками, мы обсудим в этой главе позднее.

Неправильная конфигурация системы NAT может привести к тому, что пакеты с частными адресами начнут проникать в Интернет. Они могут достичь хоста назначения, но от-

[6] Разумеется, многие маршрутизаторы в настоящее время могут запускать встроенные ядра системы Linux. Но даже в этом случае их изначальные системы остаются более эффективными и безопасными, чем универсальные компьютеры, которые также пересылают пакеты.

ветные пакеты не будут получены. Организация CAIDA[7], замеряющая трафик магистральных сетей, сообщает о том, что 0,1–0,2% пакетов, проходящих по магистрали, имеют либо частные адреса, либо неправильные контрольные суммы. На первый взгляд показатель кажется незначительным, но на самом деле это приводит к тому, что каждую минуту в сети циркулируют тысячи пакетов. Информацию по статистике Интернета и средствам измерения производительности глобальной сети можно получить на сайте `caida.org`.

Одной из особенностей системы NAT является то, что хост Интернета не может напрямую подключиться к внутренним машинам организации. Для того чтобы преодолеть это ограничение, в некоторых реализациях системы NAT разрешается создавать туннели, которые поддерживают прямые соединения с выбранными хостами.[8]

Еще одна проблема заключается в том, что некоторые приложения встраивают IP-адреса в информационную часть пакетов. Такие приложения не могут нормально работать совместно с NAT. К ним относятся определенные протоколы маршрутизации, программы потоковой доставки данных и ряд FTP-команд. Система NAT иногда также отключает виртуальные частные сети (virtual private network — VPN).

Система NAT скрывает внутреннюю структуру сети. Это может показаться преимуществом с точки зрения безопасности, но специалисты считают, что на самом деле система NAT не обеспечивает должную безопасность и уж во всяком случае не устраняет потребность в брандмауэре. Кроме того, она препятствует попыткам оценить размеры и топологию Интернета (см. документ RFC4864, *Local Network Protection for IPv6*, содержащий полезную информацию о реальных и мнимых преимуществах системы NAT и протокола IPv4).

Адресация в стандарте IPv6

Адреса IPv6 имеют длину 128 бит. Эти длинные адреса изначально были предназначены для решения проблемы исчерпания IP-адресов. Однако теперь они, помимо прочего, позволяют справиться с проблемами маршрутизации, мобильности и локальности ссылок.

Граница между сетевой частью и машинной частью IPv6-адреса фиксирована на /64, поэтому не может возникнуть никаких разногласий или путаницы в отношении того, насколько длинной является сетевая часть адреса на самом деле. Иными словами, истинной подсети в мире IPv6 больше не существует, хотя термин подсеть сохраняется как синоним локальной сети. Несмотря на то что номера сетей всегда имеют длину 64 бит, маршрутизаторы не должны учитывать все 64 бита при принятии решений о маршрутизации. Они могут маршрутизировать пакеты по префиксу, как и в протоколе CIDR.

Нотация адресов IPv6

Стандартная нотация адресов IPv6 делит 128 бит адреса на 8 групп по 16 бит каждый, разделенных двоеточиями. Например,

```
2607:f8b0:000a:0806:0000:0000:0000:200e
```[9]

[7]CAIDA (Cooperative Association for Internet Data Analysis — совместная ассоциация по анализу данных в сети Интернет) находится в Центре суперкомпьютеров, который расположен в здании Калифорнийского университета в Сан-Диего (`www.caida.org`).

[8]Многие маршрутизаторы поддерживают также стандарты Universal Plug and Play (UPnP), продвигаемые компанией Microsoft. Эти стандарты позволяют внутренним машинам устанавливать собственные динамические туннели NAT. В зависимости от точки зрения пользователя, это может оказаться как удачным решением, так и угрозой безопасности. При желании эту функциональную возможность маршрутизатора легко отключить.

[9]Это настоящий IPv6-адрес, поэтому не используйте его в своих системах даже для экспериментов. В стандарте RFC3849 предполагается, что в документации и примерах указаны адреса IPv6 в блоке с префиксом `2001:db8::/32`. Но мы хотели показать реальный пример, который маршрутизируется в сети Интернет.

Каждая 16-битная группа представлена четырьмя шестнадцатеричными цифрами. Это отличается от нотации IPv4, в которой каждый байт адреса представлен десятичным номером.

Несколько условных упрощений помогают ограничить объем ввода, необходимого для представления адресов IPv6. Во-первых, вам не нужно указывать ведущие нули внутри группы. Группа `000a` в третьей группе выше может быть записана просто как `a`, а четвертая группа `0806` — как `806`. Группы со значением `0000` должны быть представлены как `0`. Применение этого правила уменьшает адрес, приведенный выше, до следующей строки:

```
2607:f8b0:a:806:0:0:0:200e
```

Во-вторых, вы можете заменить любое количество смежных нулевых 16-разрядных групп двойным двоеточием:

```
2607:f8b0:a:806::200e
```

Символы `::` в адресе можно использовать только один раз. Они могут отображаться как первый или последний компонент. Например, адрес обратной связи IPv6 (аналогичный 127.0.0.1 в IPv4) равен `::1`, что эквивалентно `0:0:0:0:0:0:0:1`.

Исходная спецификация адресов IPv6, RFC4921, описывала эти упрощения нотации, но не требовала их использования. В результате для записи заданного IPv6-адреса может существовать несколько способов, совместимых со спецификацией RFC491, что иллюстрируется несколькими версиями приведенного выше примера.

Эта полиморфность делает поиск и сопоставление трудной задачей, потому что адреса перед сравнением необходимо нормализовать. Это проблема: мы не можем ожидать, что стандартное программное обеспечение для обработки данных, такое как электронные таблицы, языки сценариев и базы данных, знает подробности нотации IPv6.

Документ RFC5952 обновил спецификацию RFC4921, чтобы сделать упрощенные обозначения обязательными. Он также добавил еще несколько правил, гарантирующих, что каждый адрес имеет только одно текстовое представление.

- Шестнадцатеричные цифры `a-f` должны быть представлены строчными буквами.
- Элемент `::` не может заменять одну 16-битную группу. (Просто используйте `:0:`.)
- Если есть несколько групп, которые можно заменить элементом `::`, то выбирать следует максимально длинную последовательность нулей.

На практике все еще существуют адреса, несовместимые со спецификацией RFC5952, и почти все сетевое программное обеспечение их тоже принимает. Однако мы настоятельно рекомендуем следовать правилам RFC5952 в ваших конфигурациях, ведении документации и программном обеспечении.

Префиксы IPv6

Адреса IPv4 не были предназначены для географической кластеризации наподобие телефонных номеров или почтовых индексов, но кластеризация была добавлена постфактум в виде соглашений CIDR. (Разумеется, соответствующая "география" на самом деле представляет собой пространство маршрутизации, а не физическое местоположение.) Протокол CIDR был настолько технически успешным, что иерархическое переназначение сетевых адресов теперь принимается во всем протоколе IPv6.

Ваш интернет-провайдер IPv6 получает блоки префиксов IPv6 от одного из региональных реестров, перечисленных в табл. 13.4. Интернет-провайдер, в свою очередь, назначает вам префикс, который вы добавляете к локальным частям адресов, обычно

на пограничном маршрутизаторе. Организации могут свободно устанавливать границы делегирования в назначенных им адресных пространствах.

Всякий раз, когда префикс адреса представляется в текстовой форме, протокол IPv6 использует обозначение CIDR для представления длины префикса. Общий шаблон имеет следующий вид:

IPv6-адрес/префикс-длина-в-десятичном-виде

Часть IPv6-адреса приведена в предыдущем разделе. Это должен быть полноразмерный 128-разрядный адрес. В большинстве случаев биты адресов за префиксом устанавливаются равными нулю. Однако иногда бывает необходимо указать полный адрес хоста вместе с длиной префикса; намерение и смысл обычно ясны из контекста.

IPv6-адрес, указанный в предыдущем разделе, приводит к серверу Google. 32-разрядный префикс, который маршрутизируется на североамериканской интернет-магистрали, имеет вид:

```
2607: f8b0::/32
```

В этом случае адресный префикс был присвоен региональным интернет-регистратором ARIN непосредственно компании Google. Это можно проверить, просмотрев префикс на сайте `arin.net`.[10] Промежуточный интернет-провайдер отсутствует. Компания Google отвечает за структурирование оставшихся 32 бит сетевого номера по своему усмотрению. Скорее всего, в инфраструктуре Google используются несколько дополнительных уровней префиксов.

Автоматическая нумерация машин

Идентификатор 64-битного интерфейса машины (машинная часть адреса IPv6) может быть автоматически получен из 48-разрядного MAC-адреса интерфейса (аппаратного) интерфейса с помощью алгоритма, известного как "измененный EUI-64", документированного в спецификации RFC4291.

В частности, идентификатор интерфейса представляет собой только MAC-адрес с двумя байтами `0xFFFE`, вставленными в середине, и одним инвертированным битом. Инвертированный бит — это седьмой самый старший бит первого байта; другими словами, вы применяете операцию XOR к первому байту и числу `0x02`. Например, на интерфейсе с MAC-адресом `00:1b:21:30:e9:c7` идентификатор автоматически сгенерированного интерфейса будет равен `02`**`1`**`b:21ff:fe30:e9c7`. Подчеркнутая цифра равна 2, а не 0, из-за инвертированного бита.

Эта схема позволяет автоматически нумеровать машины, что удобно для системных администраторов, поскольку им достаточно управлять только сетевой частью адресов.

То, что MAC-адрес можно увидеть на уровне протокола IP, имеет как хорошие, так и плохие последствия. Хорошая часть состоит в том, что нумерация машин может быть полностью автоматической. Плохая часть заключается в том, что изготовитель интерфейсной платы кодируется в первой половине MAC-адреса (см. раздел 13.3), поэтому вы неизбежно частично отказываетесь от конфиденциальности. Это раскрывает хакерам часть информации об архитектуре сети. Стандарты IPv6 указывают на то, что серверы не обязаны использовать MAC-адреса для получения идентификаторов компьютеров; они могут использовать любую систему нумерации.

---

[10] В этом случае длина префикса блока адреса, назначенного регистратором ARIN, и записи таблицы маршрутизации одинаковы, но это не всегда верно. Префикс распределения определяет административную границу, тогда как префикс маршрутизации относится к местоположению маршрутного пространства.

Виртуальные серверы имеют виртуальные сетевые интерфейсы. MAC-адреса, связанные с этими интерфейсами, обычно рандомизированы, что гарантирует уникальность только в конкретном локальном контексте.

Автоматическая конфигурация локальных адресов

Автоматически сгенерированные номера компьютеров, описанные в предыдущем разделе, объединяются с несколькими другими простыми функциями IPv6, чтобы обеспечить автоматическую настройку сети для интерфейсов IPv6. Общая схема называется SLAAC (StateLess Address AutoConfiguration). Конфигурация SLAAC для интерфейса начинается с назначения адреса локальной сети, которая имеет фиксированный сетевой адрес `fe80::/64`. Часть адреса, определяющая машину, устанавливается на основе MAC-адреса интерфейса, как описано выше.

Протокол IPv6 не имеет широковещательных адресов в стиле IPv4 как таковых, но термин *локальная сеть* играет примерно ту же роль и означает "данную физическую сеть". Маршрутизаторы никогда не пересылают пакеты, отправленные по адресам в этой сети.

После того как был установлен локальный адрес связи для интерфейса, стек протокола IPv6 отправляет пакет ICMP Router Solicitation (запрос к маршрутизатору по протоколу ICMP) на адрес многоадресатной рассылки, включающей все маршрутизаторы.

Маршрутизаторы отвечают пакетами сообщений ICMP Router, в которых перечислены сетевые номера IPv6 (на самом деле, префиксы), используемые в сети.

Если в одной из этих сетей установлен флаг автоконфигурации ("autoconfiguration OK"), запрашивающий хост назначает дополнительный адрес своему интерфейсу, который объединяет часть сети, объявляемую маршрутизатором, с частью автоматически генерируемого адреса хоста, созданного с использованием модифицированного алгоритма EUI-64. Другие поля в анонсе маршрутизатора позволяют ему идентифицировать себя как соответствующий шлюз по умолчанию и передавать сетевой параметр MTU.

В конечном результате новый хост становится полноправным элементом сети IPv6 без необходимости запуска какого-либо сервера (кроме маршрутизатора) и без какой-либо локальной конфигурации. К сожалению, эта система не распространяется на конфигурацию программного обеспечения более высокого уровня, такого как DNS, поэтому вы все равно должны запускать традиционный сервер DHCPv6.

🕮 Дополнительную информацию по протоколу DCHP см. в разделе 13.7.

Иногда можно встретить сетевую автоматическую конфигурацию IPv6, связанную с именем, полученному по протоколу обнаружения соседей (Neighbor Discovery Protocol, NDP). Хотя протокол NDP описывается в документе RFC4861, этот термин на самом деле довольно расплывчатый. Он охватывает использование и интерпретацию различных типов пакетов ICMPv6, некоторые из которых связаны только с периферийным доступом к обнаружению сетевых соседей. С технической точки зрения взаимосвязь заключается в том, что описанная выше процедура SLAAC использует некоторые, но не все, типы ICMPv6, определенные в документе RFC4861. Проще назвать это SLAAC или "автоконфигурацией IPv6" и зарезервировать термин *обнаружение соседа* для описанного процесса сопоставления IP-MAC, описанного в разделе 13.5

Туннелирование IPv6

Для облегчения перехода от IPv4 к IPv6 были предложены различные схемы, в основном ориентированные на способы туннелирования трафика IPv6 через сеть IPv4 для компенсации пробелов в поддержке IPv6. Две широко используемые системы тун-

нелирования называются 6to4 и Teredo; последний, названный в честь семьи древесных червей, сверлящих деревья, может использоваться на системах, расположенных за устройством NAT.

Источники информации о протоколе IPv6

Ниже перечислены некоторые полезные источники информации о протоколе IPv6.

- `worldipv6launch.com` — множество разнообразной информации о протоколе IPv6.
- RFC3587 — спецификация *IPv6 Global Unicast Address Format.*
- RFC4291 — спецификация *Version 6 Addressing Architecture.*

13.5. Маршрутизация

Маршрутизация — это процесс направления пакета по лабиринту сетей, находящихся между отправителем и получателем. В системе TCP/IP маршрутизация происходит примерно так, как путешественник, первый раз посетивший страну, находит нужный ему дом, задавая вопросы местным жителям. Первый человек, с которым он заговорит, возможно, укажет ему нужный город. Войдя в город, путешественник спросит другого человека, и тот расскажет, как попасть на нужную улицу. В конце концов наш путешественник подойдет достаточно близко к конечному пункту своих странствий, чтобы кто-нибудь указал ему дом, который он ищет.

Маршрутная информация в системе TCP/IP имеет форму правил (*маршрутов*), например: "Для того чтобы достичь сети A, посылайте пакеты через компьютер C". Часто существует и стандартный маршрут. В нем объясняется, что нужно делать с пакетами, предназначенными для отправки в сеть, маршрут к которой не указан явным образом.

Данные маршрутизации хранятся в одной из таблиц ядра. Каждый элемент этой таблицы содержит несколько параметров, включая сетевую маску. Для направления пакета по заданному адресу ядро подбирает наиболее конкретный маршрут (т.е. тот, где самая длинная маска). Если ни один из маршрутов (в том числе стандартный) не подходит, то отправителю возвращается ICMP-сообщение вида "network unreachable" (сеть недоступна).

Слово "маршрутизация" широко употребляется в двух различных смыслах:

- процедура поиска сетевого адреса в специальной таблице для передачи пакета в пункт его назначения;
- процесс построения этой таблицы.

Ниже мы рассмотрим, как осуществляется переадресация пакетов и как вручную добавить или удалить маршрут. Более сложные вопросы, связанные с работой протоколов маршрутизации, создающих и обслуживающих маршрутные таблицы, освещаются в главе 15.

Таблицы маршрутизации

Таблицу маршрутизации можно просмотреть с помощью команды **`ip route show`** в системе Linux или **`netstat -r`** в системе FreeBSD. Команда **`netstat`** в системе Linux также существует и продолжает применяться. Мы используем ее в примерах просто, чтобы не приводить две разные версии одного и того же вывода. Версия **`ip`** выдает такое же содержимое, но в другом формате.

Команда **netstat -rn** запрещает поиск доменных имен в системе DNS и выводит всю информацию в числовом виде, что в принципе полезнее. Ниже приводится таблица маршрутизации IPv4, которая должна дать читателям более ясное представление о том, что такое маршрут.

```
redhat$ netstat -rn
Destination     Genmask          Gateway          Fl  MSS   Iface
132.236.227.0   255.255.255.0    132.236.227.93   U   1500  eth0
default         0.0.0.0          132.236.227.1    UG  1500  eth0
132.236.212.0   255.255.255.192  132.236.212.1    U   1500  eth1
132.236.220.64  255.255.255.192  132.236.212.6    UG  1500  eth1
127.0.0.1       255.255.255.255  127.0.0.1        U   3584  lo
```

В рассматриваемой системе есть два сетевых интерфейса: 132.236.227.93 (eth0) — в сети 132.236.227.0/24 и 132.236.212.1 (eth1) — в сети 132.236.212.0/26.

Поле destination обычно содержит адрес сети. В поле gateway отображается адрес локального сетевого интерфейса или соседнего хоста; в ядре системы Linux для вызова шлюза, установленного по умолчанию, в поле gateway может быть записан адрес 0.0.0.0.

Например, четвертый маршрут указывает, что для достижения сети 132.236.220.64/26 пакеты следует посылать в шлюз 132.236.212.6 через интерфейс eth1. Вторая запись содержит описание стандартного маршрута. Пакеты, не адресованные явно ни в одну из трех указанных сетей (или самому компьютеру), будут направлены в стандартный шлюз 132.236.227.1.

Компьютеры могут посылать пакеты только тем шлюзам, которые физически подключены к той же сети. Локальные компьютеры могут перемещать пакеты только на один шаг в направлении пункта назначения, поэтому в него бессмысленно включать информацию о шлюзах, не являющихся смежными в таблице локальной маршрутизации. Каждый шлюз, через который проходит пакет, принимает следующее решение о его перемещении, анализируя собственную таблицу локальной маршрутизации.[11]

Вести таблицы маршрутизации можно статически, динамически или комбинированным способом. Статический маршрут задается в явном виде с помощью команды **route** (FreeBSD). Он должен оставаться в таблице маршрутизации в течение всего времени работы системы. Во многих случаях такие маршруты задаются на этапе начальной загрузки с помощью одного из сценариев запуска системы. Например, команды в системе Linux

```
ip route add 132.236.220.64/26 via 132.236.212.6 dev eth1
ip route add default via 132.236.227.1 dev eth0
```

добавляют, соответственно, четвертый и второй маршруты из числа тех, что отображаются выше командой **netstat -rn** (первый и третий маршруты добавляются командой **ifconfig** при конфигурировании интерфейсов eth0 и eth1). Эквивалентная команда в системе FreeBSD имеет вид

```
route add -net 132.236.220.64/26 gw 132.236.212.6 eth1
route add default gw 132.236.227.1 eth0
```

Последний маршрут также добавляется на этапе начальной загрузки. Он определяет *интерфейс обратной связи* (loopback interface), препятствующий пакетам, которые компьютер посылает сам себе, выходить в сеть. Вместо этого они напрямую переносятся из выходной сетевой очереди ядра во входную очередь.

[11] Маршрутизация по IP-адресу отправителя является исключением из этого правила; см. раздел 13.8.

В относительно стабильной локальной сети статическая маршрутизация является достаточно эффективным решением. Эта система проста в управлении и надежна в эксплуатации, но требует знания системным администратором топологии сети на момент начальной загрузки, а также того, чтобы эта топология в периоды между загрузками не изменялась.

Большинство компьютеров локальной сети имеет единственный выход во внешний мир, поэтому маршрутизация осуществляется очень просто: достаточно на этапе начальной загрузки добавить стандартный маршрут. Компьютер может получить стандартный маршрут вместе со своим IP-адресом у сервера DHCP (этот протокол описан в разделе 13.7).

В сетях с более сложной топологией требуется динамическая маршрутизация. Она осуществляется процессом-демоном, который ведет и модифицирует таблицу маршрутизации. Демоны маршрутизации, "обитающие" на различных компьютерах, взаимодействуют друг с другом с целью определения топологии сети и решения вопроса о том, как добраться до удаленных адресатов. Существует несколько таких демонов. Подробно они будут рассмотрены в главе 15.

Директивы переадресации протокола ICMP

Несмотря на то что протокол IP не предусматривает средств управления маршрутной информацией, в нем имеется механизм контроля нарушений: директивы переадресации протокола ICMP. Если маршрутизатор направляет пакет компьютеру, находящемуся в той же сети, из которой этот пакет был получен, то что-то работает неправильно. Поскольку отправитель, маршрутизатор и маршрутизатор следующего перехода находятся в одной сети, то пакет можно послать не через два перехода, а через один. Маршрутизатор делает вывод о том, что таблицы маршрутизации отправителя являются неточными или неполными.

В этой ситуации маршрутизатор может уведомить отправителя о проблеме с помощью переадресующего ICMP-пакета. В таком пакете сообщается следующее: "Не нужно посылать мне пакеты для хоста *xxx*; их следует адресовать хосту *yyy*".

Теоретически, получив директиву переадресации, отправитель может обновить свою таблицу маршрутизации, чтобы следующие пакеты, предназначенные данному адресату, шли по более прямому пути. На практике переадресация не подразумевает этапа аутентификации и поэтому не может считаться доверенной. Маршрутизатор может проигнорировать полученную команду перенаправить трафик в другое место, но чаще всего системы UNIX и Linux пытаются это сделать по умолчанию. В таких ситуациях необходимо уточнить возможные источники команд переадресации в своей сети и отключить их, если они могут создать проблемы.



В Linux допустимость ICMP-директив определяется элементом **accept_redirects** файловой системы **/proc**. О том, как проверять и устанавливать этот и подобные ему параметры, рассказывается в разделе 13.10.



В системе FreeBSD допустимость переадресации ICMP управляется параметрами `net.inet.icmp.drop_redirect` и `net.inet6.icmp6.rediraccept` соответственно. Для того чтобы игнорировать переадресацию установите их равными единице и нулю соответственно в файле **/etc/sysctl.conf**. (Для активации новых установок необходимо выполнить перезагрузку или команду **sudo /etc/rc.d/sysctl reload**).

13.6. ARP: протокол преобразования адресов в IPv4 и IPv6

Несмотря на то что IP-адреса являются аппаратно-независимыми, для фактической передачи данных на канальном уровне должны применяться аппаратные адреса[12]. В протоколах IPv4 и Iv6 используются разные протоколы, которые практически одинаковым образом определяют, какой аппаратный адрес связан с тем или иным IP-адресом.

В протоколе IPv4 используется протокол ARP (Address Resolution Protocol — протокол преобразования адресов), определенный спецификацией RFC826. В протоколе IPv6 используется протокол обнаружения соседей (Neighbor Discovery Protocol, ND), определенный спецификацией RFC4861. Эти протоколы можно применять в сетях любых типов, которые поддерживают широковещательный режим или многоадресатную рассылку по всем хостам, но чаще всего их рассматривают в контексте сетей Ethernet.

Когда компьютер А хочет послать пакет компьютеру Б, находящемуся в той же сети Ethernet, он использует протоколы ARP или ND для нахождения аппаратного адреса Б. Если же компьютер Б расположен в другой сети, то компьютер А с помощью подсистемы маршрутизации определяет IP-адрес маршрутизатора следующего перехода, а затем с помощью протоколов ARP или ND выясняет аппаратный адрес этого маршрутизатора. Эти протоколы позволяют находить только адреса устройств, непосредственно подключенных к той же сети.

Каждый компьютер хранит в памяти специальную таблицу, называемую кешем ARP или ND, которая содержит результаты последних ARP-запросов. В нормальных условиях многие адреса, необходимые компьютеру, выявляются вскоре после начальной загрузки, поэтому таблицы ARP и ND мало влияют на загруженность сети.

Протоколы ARP и ND функционируют путем широковещательной рассылки пакетов примерно следующего содержания: "Знает ли кто-нибудь аппаратный адрес для IP-адреса X?" Устройство, которое разыскивают, узнает свой IP-адрес и посылает ответ: "Да, это IP-адрес одного из моих сетевых интерфейсов, соответствующий Ethernet-адрес — `08:00:20:00:fb:6a`".

Исходный запрос включает IP- и MAC-адрес запрашивающей стороны, благодаря чему разыскиваемое устройство может ответить, не посылая собственный запрос. Это позволяет обоим компьютерам узнать адреса друг друга за один сеанс обмена пакетами. Другие компьютеры, "слышавшие" исходное широковещательное сообщение, посланное инициатором запроса, тоже могут записать информацию о его адресах.



В системе Linux команда **`ip neigh`** изучает и обрабатывает содержимое кеша ARP или ND. Обычно она используется для добавления и удаления записей, но может также очищать всю таблицу и отображать ее. В частности, команда **`ip neigh show`** отображает содержимое кеша.



В системе FreeBSD команда **`arp`** управляет кешем ARP, а команда **`ndp`** предоставляет доступ к кешу ND.

Эти команды, как правило, применяются для отладки и при работе со специальным оборудованием. Например, если два хоста сети имеют одинаковый IP-адрес, то на одном из них запись в ARP-таблице будет правильной, а на другом — нет. С помощью команды **`arp`** можно найти хост-нарушитель.

[12]Это не касается двухточечных соединений, где получатель иногда подразумевается неявно.

Неправильные записи в кеше могут быть признаком того, что некто, имеющий доступ в вашу локальную сеть осуществляет подмену сетевого трафика. Этот вид атаки известен как ARP-фишинг (ARP spoofing) или отравление кеша (cache poisoning).

13.7. DHCP: протокол динамического конфигурирования хостов

📖 Протокол DHCP определен в документах RFC2131, 2132 и 3115.

Когда вы добавляете устройство или компьютер в сеть, то обычно получаете свой IP-адрес в локальной сети, настраиваете соответствующий маршрутизатор, заданный по умолчанию, и присоединяетесь к локальному серверу DNS. Все это за вас может сделать протокол DHCP (Dynamic Host Configuration Protocol — протокол динамического конфигурирования хостов).

Протокол DHCP дает возможность клиенту взять сетевые и административные параметры "в аренду" у центрального сервера, отвечающего за их распространение. Принцип аренды особенно удобен для персональных компьютеров, которые выключены, когда на них никто не работает, и интернет-провайдеров, чьи клиенты подключаются по коммутируемым линиям.

К "арендуемым" параметрам относятся следующие:

- IP-адреса и сетевые маски;
- адреса шлюзов (стандартные маршруты);
- адреса DNS-серверов;
- имена компьютеров, на которых выполняется система Syslog;
- адреса серверов WINS, X-серверов шрифтов, прокси-серверов и NTP-серверов;
- адреса серверов TFTP (для получения файла начальной загрузки).

Существуют десятки других параметров (см. RFC2132 для IPv4 и RFC3315 для Iv6). Впрочем, на практике экзотические параметры используются редко.

Периодически клиенты должны повторно обращаться к DHCP-серверу с целью продления срока аренды. Если этого не делать, аренда рано или поздно закончится. DHCP-сервер будет тогда волен предоставить адрес (или иной арендованный параметр) другому клиенту. Срок аренды конфигурируется, но обычно он достаточно велик (до нескольких дней).

Даже если вы хотите, чтобы каждый хост имел собственный постоянный IP-адрес, протокол DHCP может значительно сэкономить ваши время и усилия. Когда сервер DHCP сконфигурирован и запущен, клиенты почти автоматически определяют параметры сетевой конфигурации на этапе начальной загрузки, и никакой путаницы не возникает.

Программное обеспечение DHCP

Организация ISC (Internet Software Consortium — консорциум разработчиков программного обеспечения для Интернета) поддерживает прекрасный открытый источник информации о протоколе DHCP. В настоящее время широко используются версии пакета IVS 2, 3 и 4, которые прекрасно выполняют основные функции. Версия 3 поддерживает резервное копирование серверов DHCP, а версия 4 — протокол IPv6. Сервер, клиент и агентов ретрансляции (relay agents) можно загрузить с веб-сайта `ics.org`.

Все основные дистрибутивы системы Linux используют ту или иную версию пакета ISC, хотя его серверную часть, возможно, придется устанавливать явно. В системах Red Hat и CentOS эта серверная часть называется **dhcp**, в системах Debian и Ubuntu — **ics-dhcp-server**, а в системе FreeBSD — **ics-dhcp43-server**. Убедитесь, что вы установили именно то программное обеспечение, которое хотели, поскольку многие системы имеют собственные реализации серверной и клиентской частей протокола DHCP.

Мы рекомендуем не вмешиваться в клиентскую часть протокола DHCP, поскольку эта часть кода относительно проста и поставляется заранее сконфигурированной и готовой к использованию. Изменение клиентской части протокола DHCP — непростая задача.

Однако, если вам необходимо запустить DHCP-*сервер*, мы рекомендуем использовать пакет ISC, а не пакеты конкретных поставщиков. В обычной неоднородной среде процедура администрирования сильно упростится, если будет применяться единая стандартная реализация протокола DHCP. Пакет ISC — это надежное, открытое решение, без проблем инсталлируемое в большинстве систем.

Другим вариантом является Dnsmasq, сервер, который реализует службу DHCP в сочетании с DNS-пересылкой. Это аккуратный пакет, который работает практически в любой системе. Главная страница проекта — `thekelleys.org.uk/dnsmasq/doc.html`.

Программное обеспечение DHCP-сервера также встроено в большинство маршрутизаторов. Конфигурация обычно более болезненна, чем на UNIX- или Linux-сервере, но надежность и доступность могут быть выше.

В следующих подразделах мы вкратце рассмотрим особенности протокола DHCP, инсталляцию сервера ISC, реализующего этот протокол, а также конфигурирование DHCP-клиентов.

Схема работы DHCP

Протокол DHCP — это расширение протокола BOOTP, который был придуман для того, чтобы бездисковые UNIX-станции могли загружаться по сети. Протокол DHCP не ограничивается этими параметрами, вводя понятие "аренды".

DHCP-клиент начинает диалог с DHCP-сервером, посылая широковещательное сообщение вида "помогите мне узнать, кто я"[13]. Если в локальной сети есть DHCP-сервер, он договаривается с клиентом об аренде IP-адреса и других сетевых параметров (сетевая маска, адреса сервера имен и стандартного шлюза). Если же такого сервера нет, то серверы других подсетей могут получить первоначальное широковещательное сообщение через особую часть программного обеспечения DHCP, которая называется *агентом ретрансляции*.

По истечении половины срока аренды клиент должен ее продлить. Сервер обязан отслеживать адреса, предоставленные в аренду, и сохранять эту информацию при перезагрузке. Предполагается, что клиенты делают то же самое, хотя это не обязательно. Благодаря этому достигается максимальная стабилизация сетевой конфигурации. Теоретически все программное обеспечение должно быть готово к мгновенному изменению сетевой конфигурации, но большая часть программного обеспечения по-прежнему необоснованно допускает, что сеть остается неизменной.

[13]Клиенты IPv4 инициируют обмен информацией с DHCP-сервером, используя обобщенный широковещательный адрес. В этот момент клиенты еще не знают маски своих подсетей и, следовательно, не могут использовать широковещательный адрес подсети.

Программное обеспечение DHCP, созданное организацией ISC

Демон сервера ISC называется **dhcpd**, а его файл конфигурации — **dhcpd.conf**. Обычно этот файл находится в каталоге **/etc** или **/etc/dhcp3**. Формат файла конфигурации довольно хрупкий; стоит только пропустить двоеточие, как вы получите запутанное и бесполезное сообщение об ошибке.

Для настройки нового DHCP-сервера необходимо также создать пустой файл базы данных. Проверьте резюме в конце справочной страницы о демоне **dhcpd**, чтобы определить правильное место для лицензионного файла вашей системы. Обычно оно находится где-то под каталогом **/var**.

Для заполнения файла **dhcpd.conf** потребуется следующая информация:

- адреса подсетей, для которых демон **dhcpd** должен управлять IP-адресами, и диапазоны выделяемых адресов;
- список статических адресов, которые вы хотите создать, а также аппаратные MAC-адреса получателей;
- начальный и максимальный сроки аренды в секундах;
- остальные параметры, которые сервер должен передавать DHCP-клиентам: сетевая маска, стандартный маршрут, домен DNS, адреса серверов имен и т.д.

На справочной странице (man page), посвященной демону **dhcpd**, дан обзор процесса конфигурации. Точный синтаксис конфигурационного файла описан на справочной странице файла **dhcpd.conf**. Кроме того, следует убедиться, что демон **dhcpd** автоматически запускается на этапе начальной загрузки системы (см. главу 3). Если система не выполняет эту операцию автоматически, полезно сделать запуск демона условным, осуществляемым при наличии файла **dhcpd.conf**.

Ниже показан пример файла **dhcpd.conf**, взятого из Linux-системы с двумя сетевыми интерфейсами: внутренним и внешним, подключенным к Интернету. На компьютере выполняется система NAT для трансляции адресов внутренней сети, которой предоставляются десять IP-адресов. Файл содержит пустую запись для внешнего интерфейса (обязательна) и запись `host` для одного компьютера, которому нужен фиксированный адрес.

```
# global options
option domain-name "synack.net";
option domain-name-servers gw.synack.net;
option subnet-mask 255.255.255.0;
default-lease-time 600;
max-lease-time 7200;

subnet 192.168.1.0 netmask 255.255.255.0 {
    range 192.168.1.51 192.168.1.60;
    option broadcast-address 192.168.1.255;
    option routers gw.synack.net;
}

subnet 209.180.251.0 netmask 255.255.255.0 {
}

host gandalf {
    hardware ethernet 08:00:07:12:34:56;
    fixed-address gandalf.synack.net;
}
```

Более подробная информация о системе DNS приведена в главе 16.

Если вы не назначаете статические адреса, как это сделано выше, необходимо проанализировать, как ваша DHCP-конфигурация будет взаимодействовать с системой DNS. Проще всего присвоить каждому динамически выделяемому адресу общее имя (например, `dhcp1.synack.net`) и разрешить, чтобы имена отдельных компьютеров изменялись вместе с IP-адресами. В качестве альтернативы можно сконфигурировать демон **dhcpd** так, чтобы он обновлял базу данных DNS при выделении очередного адреса. Динамические обновления — сложное решение, но оно позволяет сохранять имена компьютеров неизменными.

Агентом ретрансляции в протоколе DHCP, реализованном организацией ISC, является отдельный демон **dhcrelay**. Это простая программа, не имеющая собственного файла конфигурации, хотя дистрибутивы системы Linux часто добавляют средства инсталляции, передающие соответствующие аргументы командной строки для вашего сервера. Агент **dhcrelay** прослушивает запросы DHCP в локальной сети и передает их на указанные вами удаленные серверы DHCP. Это удобно и для централизации управления службами DHCP, и для облегчения резервного копирования серверов DHCP.

Клиент протокола DHCP по версии организации ISC также не имеет конфигурации. Он хранит файл состояний для каждого соединения в каталогах **/var/lib/dhcp** или **/var/lib/dhclient**. Имена файлов совпадают с именами интерфейсов, которые они описывают. Например, файл **dhclient-eth0.leases** может содержать все сетевые параметры, которые клиент **dhclient** установил для интерфейса **eth0**.

13.8. Вопросы безопасности

Теме безопасности посвящена глава 27, но ряд вопросов, касающихся IP-сетей, заслуживает отдельного упоминания. В этом разделе рассмотрим сетевые механизмы, которые традиционно вызывают проблемы безопасности, и опишем пути решения этих проблем. Детали функционирования систем, приводимых в качестве примера, очень отличаются друг о друга (как и методы их изменения), поэтому они описываются в отдельных подразделах.

Перенаправление IP-пакетов

Если в UNIX- или Linux-системе разрешено перенаправление IP-пакетов, то компьютер может выступать в качестве маршрутизатора. Иначе говоря, он может принимать пакеты от третьей стороны, поступающие на его сетевой интерфейс и пересылать их шлюзу или хосту через другой интерфейс.

Если ваша система не имеет несколько сетевых интерфейсов и не предназначена для функционирования в роли маршрутизатора, эту функцию рекомендуется отключить. Хосты, перенаправляющие пакеты, часто оказываются вовлеченными в атаки на системы безопасности, будучи вынужденными выдавать внешние пакеты за свои собственные. Эта уловка позволяет пакетам злоумышленников обходить сетевые сканеры и фильтры.

Лучше всего, чтобы хост использовал несколько сетевых интерфейсов для своего собственного трафика и не пересылал посторонние пакеты.

Директивы переадресации протокола ICMP

С помощью переадресующих ICMP-пакетов можно злонамеренно менять направление трафика и редактировать таблицы маршрутизации. Большинство операционных систем по умолчанию принимает эти пакеты и следует содержащимся в них инструкциям.

Но, согласитесь, вряд ли можно назвать приемлемой ситуацию, когда на несколько часов весь трафик организации перенаправляется конкуренту, особенно если в это время выполняется резервное копирование. Мы рекомендуем так конфигурировать маршрутизаторы (или системы, играющие роль маршрутизаторов), чтобы переадресующие ICMP-пакеты игнорировались и, возможно, фиксировались в журнальном файле.

Маршрутизация по адресу отправителя

Механизм маршрутизации по адресу отправителя в протоколе IPv4 позволяет отправителю явно указать последовательность шлюзов, через которые должен пройти пакет на пути к получателю. При этом отключается алгоритм поиска следующего перехода, выполняемый на каждом шлюзе для определения того, куда нужно послать пакет.

Маршрутизация по адресу отправителя была частью исходной спецификации протокола IP и служила для целей тестирования. Но она создает проблему с точки зрения безопасности, ведь пакеты часто фильтруются в зависимости от того, откуда они прибыли. Злоумышленник может так подобрать маршрут, что пакет будет казаться прибывшим из внутренней сети, а не из Интернета, поэтому брандмауэр его пропустит. Мы рекомендуем не принимать и не перенаправлять доставленные подобным образом пакеты.

Несмотря на критику маршрутизации по протоколу IPv4, развернутую в Интернете, она каким-то образом сумела проникнуть в протокол IPv6. Однако эта функция IPv6 была отвергнута спецификацией RFC5095 в 2007 г. Соответствующие реализации протокола IPv6 теперь должны отклонять пакеты с маршрутизацией и возвращать сообщение об ошибке отправителю.[14] Системы Linux и FreeBSD следуют стандарту RFC5095, как и коммерческие маршрутизаторы.

Широковещательные пакеты эхо-запросов и другие виды направленных широковещательных сообщений

Пакеты эхо-запросов, несущие в себе широковещательный адрес сети (а не конкретного хоста), обычно доставляются всем хостам сети. Такие пакеты применяются в атаках типа "отказ от обслуживания", например в так называемых атаках Smurf (по названию программы, в которой они впервые были применены).

Широковещательные пакеты эхо-запросов являются "направленными" в том смысле, что они посылаются по широковещательному адресу конкретной удаленной сети. Стандартные подходы к обработке таких пакетов постепенно меняются. Например, в операционной системе Cisco IOS 11.x и более ранних версий по умолчанию осуществлялась переадресация направленных широковещательных пакетов, но в версиях 12.0 и выше этого уже нет. Обычно можно настроить стек TCP/IP так, чтобы широковещательные пакеты, приходящие из других сетей, игнорировались, но, поскольку это нужно сделать для каждого сетевого интерфейса, подобная задача является весьма нетривиальной в крупной организации.

Подмена IP-адресов

Исходный адрес IP-пакета обычно заполняется реализациями протокола TCP/IP в ядрах и представляет собой адрес хоста, с которого был отправлен пакет. Но если про-

[14]Тем не менее маршрутизация по адресу отправителя в протоколе IPv6 может быть в некоторой степени реабилитирована в виде "сегментной маршрутизации", которая теперь интегрирована в ядро Linux (см. `Lwn.net/Articles/722804`, где обсуждается эта технология).

грамма, создающая пакет, работает с низкоуровневым IP-сокетом, она может подставить любой исходный адрес, какой пожелает. Это называется подменой IP-адресов и обычно ассоциируется с попытками взлома сети. В этой схеме жертвой часто становится компьютер, идентифицируемый по поддельному IP-адресу (если он действительно существует). Сообщения об ошибках и ответные пакеты могут переполнить и вывести из строя сеть жертвы. Подмена пакетов от большого множества внешних машин называется "распределенной атакой на основе отказа в обслуживании" (distributed denial-of-service attack).

Поддельные IP-адреса следует запрещать на пограничном маршрутизаторе, блокируя отправляемые пакеты, исходные адреса которых не находятся в подконтрольном диапазоне. Это особенно важно в университетских сетях, где студенты любят экспериментировать и часто подобным образом срывают свою злость.

В то же время, если в локальной сети используются адреса из частного диапазона, то фильтрации могут подвергаться пакеты с частными адресами, пытающиеся "проскочить" в Интернет. Ответ на такие пакеты никогда не будет получен из-за отсутствия соответствующих маршрутов в магистральной сети. Их появление свидетельствует о том, что в системе имеется внутренняя ошибка конфигурации.

Необходимо также защищаться от хакеров, подделывающих исходные адреса внешних пакетов, вследствие чего брандмауэр начинает считать, будто они поступили из локальной сети. Помочь этому может эвристический метод, "одноадресная трансляция по обратному пути " (unicast reverse path forwarding — uRPF). В этом случае IP-шлюзы будут отвергать все пакеты, удовлетворяющие следующему условию: интерфейс, через который пришел пакет, не совпадает с тем интерфейсом, через который пакет уйдет, если его исходный адрес будет равен целевому адресу. Для проверки источника сетевых пакетов этот метод использует обычную таблицу IP-маршрутизации. Метод uRPF применяется не только в специализированных маршрутизаторах, но и в ядре системы Linux, в которой этот режим включен по умолчанию.

Если ваш сервер имеет несколько выходов в Интернет, целесообразно разделить внешние маршруты на исходящие и входящие. В этом случае следует отключить режим uRPF, чтобы протокол маршрутизации работал правильно. Если же выход в Интернет только один, безопаснее включить режим uRF.

Встроенные брандмауэры

Как правило, соединение вашей локальной сети с внешним миром и управление трафиком в соответствии с правилами, принятыми в организации, осуществляют сетевые фильтры пакетов или брандмауэры (firewalls). К сожалению, компания Microsoft извратила представление о том, как должен работать брандмауэр, на примере своих систем Windows, печально известных своей уязвимостью. Несколько последних выпусков системы Windows содержали свои собственные брандмауэры и "громко возмущались", когда пользователь пытался их отключить.

Все системы, которые мы используем в качестве примеров, содержат программное обеспечение для фильтрации пакетов, но отсюда не следует, что каждой UNIX- или Linux-машине нужен отдельный брандмауэр. Это не так. Механизмы фильтрации пакетов, встроенные в эти системы, позволяют этим машинам выполнять функции сетевых шлюзов.

Однако мы не рекомендуем использовать рабочую станцию как брандмауэр. Даже самая совершенная операционная система слишком сложна, чтобы быть надежной. Специальное программное обеспечение более предсказуемое и более надежное, даже если оно тайно использует систему Linux.

Даже сложное программное обеспечение, предлагаемое, например, компанией Check Point (чьи программы выполняются на хостах под управлением операционных систем UNIX, Linux и Windows), уступает в надежности межсетевым экранам серии Adaptive Security Appliance компании Cisco, хотя имеет почти такую же цену!

Более подробное обсуждение брандмауэров содержится в разделе 13.14.

Виртуальные частные сети

Многим организациям, имеющим офисы в различных частях света, хотелось бы, чтобы все эти офисы были соединены одной большой частной сетью. К сожалению, стоимость аренды трансконтинентальных и даже транснациональных линий связи делает это нереальным. Таким организациям приходится использовать Интернет в качестве "частного" канала, организуя серию защищенных, зашифрованных "туннелей" между офисами. Сетевые конгломераты подобного рода называются *виртуальными частными сетями* (virtual private network — VPN).

Возможности виртуальных частных сетей необходимы также сотрудникам, которые должны соединяться с офисом из дома или находятся в поездке. Система VPN не решает всех вопросов, связанных с безопасностью и данным специальным соединением, но во многих отношениях она вполне безопасна.

🕮 О протоколе IPsec рассказывается в разделе 27.9.

В ряде частных сетей используется протокол IPsec, который в 1998 г. был стандартизован организацией IETF в качестве низкоуровневого приложения к протоколу IP. В других сетях, таких как OpenVPN, система безопасности VPN реализуется на основе протокола TCP с помощью криптографического протокола Transport Layer Security (TSL), который ранее был известен под названием Secure Sockets Layer (SSL). Протокол TSL ожидает своей очереди на стандартизацию в организации IETF, хотя он все еще не принят окончательно (по состоянию на 2017 г.).

Существует масса запатентованных реализаций систем VPN. Эти системы не взаимодействуют ни друг с другом, ни со стандартизованными системами VPN, но это нельзя считать недостатком, если все конечные точки сети находятся под контролем.

С этой точки зрения системы VPN, основанные на протоколе TLS, являются лидерами. Они также безопасны, как и протокол IPsec, но намного проще. Свободная реализация в виде OpenVN также не наносит никакого вреда. (К сожалению, эта система пока не работает под управлением операционных систем HP-UX и AIX.)

Для поддержки пользователей, работающих дома или в поездке, стало общепринятым использование небольшого компонента Java или ActiveX, загружаемого через их веб-браузеры. Эти компоненты устанавливают соединение VPN с сетью предприятия. Этот механизм удобен для пользователей, но следует учесть, что браузеры сильно отличаются друг от друга: некоторые из них реализуют службу VPN с помощью псевдосетевого интерфейса, а другие используют только специальные порты. Впрочем, веб-браузеры последней категории намного малочисленнее, чем знаменитые веб-прокси.

Пожалуйста, проверьте, что вы правильно понимаете технологию, на которой основаны применяемые решения, и не ожидайте невозможного. Истинная служба VPN (т.е. полноценное IP-соединение через сетевой интерфейс) требует наличия административных привилегий и инсталляции программного обеспечения на клиентской машине, независимо от того, какая операционная система на ней установлена: Windows или UNIX. Кроме того, следует проверить совместимость браузера, поскольку механизм, применяемый при реализации систем VPN с помощью одного браузера, часто не поддерживается другим.

13.9. Основы конфигурирования сети

Процесс подключения нового компьютера к существующей локальной сети состоит всего из нескольких этапов, но каждая система делает это по-своему. Системы обычно предоставляют графический пользовательский интерфейс, позволяющий настроить базовую сетевую конфигурацию, но в более сложных (или автоматизированных) сценариях инсталляции может понадобиться отредактировать файл конфигурации вручную.

Прежде чем подключать компьютер к сети, где есть выход в Интернет, его нужно оснастить средствами защиты (см. главу 27), чтобы ненароком не привлечь в локальную сеть хакеров.

Основные этапы подключения компьютера.

1. Назначение компьютеру уникального IP-адреса и сетевого имени.
2. Настройка компьютера на конфигурирование своих сетевых интерфейсов в процессе начальной загрузки.
3. Задание стандартного маршрута и, возможно, других параметров маршрутизации.
4. Настройка DNS-сервера, чтобы к компьютеру можно было получать доступ через Интернет.

Если в сети используется сервер DHCP (Dynamic Host Configuration Protocol — протокол динамического конфигурирования хостов), он возьмет на себя перечисленные выше обязанности. Инсталляция новой операционной системы обычно настраивает свою конфигурацию через протокол DHCP, поэтому новые машины могут вообще не потребовать определения сетевой конфигурации. Общая информация о протоколе DHCP приведена в разделе 13.7.

После внесения любого изменения, которое может повлиять на загрузку операционной системы, следует выполнить ее перезагрузку, чтобы проверить, что машина была подключена правильно. Через полгода, когда произойдет сбой питания и машина откажется загрузиться, будет сложно вспомнить, какие изменения вызвали эти проблемы.

Процесс проектирования и инсталляции физической сети описан в главе 14. Если вы имеете дело с существующей сетью и в общих чертах знаете, как она организована, то, наверное, читать о физических аспектах сетей имеет смысл только в том случае, когда планируется расширение действующей сети.

В этом разделе мы рассмотрим разные команды и проблемы, связанные с ручной настройкой конфигурации сети. Этого вполне достаточно для применения к любой версии систем UNIX или Linux. В разделах, посвященных конкретным версиям операционных систем, мы обсудим особенности, которые отличают эти системы от систем UNIX и Linux и друг от друга.

Просматривая базовую конфигурацию сети любой машины, полезно протестировать соединение с помощью базовых инструментов, таких как утилиты `ping` и `tUbuntuoute`. Эти инструменты описаны в разделе 13.12.

Присвоение сетевых имен и IP-адресов

Существует множество теорий о том, как лучше всего определить соответствие между именами компьютеров и IP-адресами в локальной сети: с помощью файла `hosts`, протокола LDAP, системы DNS или комбинации этих средств. С одной стороны, возникают конфликтующие цели, такие как возможность масштабирования, согласованность и удобство эксплуатации, а с другой — системы, достаточно гибкие, чтобы позволить

загрузку и функционирование компьютеров, когда не все службы доступны. Управление этими факторами описывается в разделе 19.5.

🕮 Система DNS описывается в главе 16.

Еще один важный аспект, который следует учитывать, — возможность изменения адресов в будущем. Если вы не используете частные адреса RFC1918 (см. раздел 13.4), то ваши IP-адреса могут быть изменены при переключении на нового интернет-провайдера. Это довольно неприятная процедура, если администратор должен лично посетить каждый компьютер и вручную сконфигурировать его. Чтобы не усложнять себе жизнь, указывайте в конфигурационных файлах сетевые имена, а привязки к IP-адресам храните только в базе данных DNS и файлах конфигурации DHCP.

Использование файла **/etc/hosts** — старейший и простейший способ преобразования сетевых имен в IP-адреса. Каждая строка файла начинается с IP-адреса и содержит различные символьные имена, под которыми известен адрес.

Ниже показан пример содержимого файла **/etc/hosts** для хоста **lollipop**.

```
127.0.0.1       localhost
::1             localhost ip6-localhost
ff02::1         ip6-allnodes
ff02::2         ip6-allrouters
192.108.21.48   lollipop.atrust.com lollipop loghost
192.108.21.254 chimchim-gw.atrust.com chimchim-gw
192.108.21.1    ns.atrust.com ns
192.225.33.5    licenses.atrust.com license-server
```

Минимальный вариант этого файла должен содержать первые две строки. Чаще всего в первой записи файла определяется хост **localhost**. Кроме того, в этом файле могут быть записаны IPv4- или IPv6-адреса.

Поскольку файл **/etc/hosts** содержит лишь локальные адреса привязки и должен находиться на каждой клиентской системе, лучше всего зарезервировать его для отображений, требующихся при загрузке (т.е. для адресов самого хоста и маршрутизатора, заданного по умолчанию, а также для имен серверов). Для поиска остальных локальных и глобальных адресов лучше использовать систему DNS или протокол LDAP. Иногда в файле **/etc/hosts** хранятся записи, о которых не следует "знать" другим компьютерам и которых нет в DNS.[15]

Команда **hostname** назначает компьютеру сетевое имя. Она обычно вызывается на этапе начальной загрузки из сценариев запуска системы, которые запрашивают назначаемое имя из конфигурационного файла. (Естественно, все поставщики систем называют этот файл по-разному. Информация о конкретных системах приведена в разделе 13.10.) В большинстве современных систем компьютеру назначается полностью определенное доменное имя, т.е. оно включает как имя хоста, так и имя домена DNS, например `anchor.cs.colorado.edu`.

В небольшой организации вполне можно выделять IP-адреса и сетевые имена вручную. Когда же в организации множество сетей и разнородных административных групп, лучше придерживаться принципа централизации. Об уникальности динамически назначаемых сетевых параметров заботится сервер DHCP.

[15]Для этой цели можно также использовать разделенную конфигурацию системы DNS (см. раздел 16.7).

Настройка сетевых интерфейсов и протокола IP

Сетевой интерфейс — это часть оборудования, которое потенциально может быть подключено к сети. Фактическое оборудование сильно варьируется. Это может быть разъем RJ-45 с соответствующим оборудованием сигнализации для проводного Ethernet, беспроводным радиоприемником или даже виртуальным аппаратным обеспечением, которое подключается к виртуальной сети.

Каждая система имеет по крайней мере два сетевых интерфейса: виртуальный интерфейс обратной связи и по крайней мере одну настоящую сетевую карту или порт. На персональном компьютере с несколькими разъемами Ethernet отдельный интерфейс сети обычно управляет каждым разъемом. (Эти интерфейсы довольно часто имеют оборудование, отличающееся от остальных.)

В большинстве систем вы можете увидеть все сетевые интерфейсы с помощью команд **ip link show** (Linux) или **ifconfig -a** (FreeBSD), независимо от того, были ли настроены или запущены интерфейсы. Вот пример из системы Ubuntu:

```
$ ip link show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
    mode DEFAULT group default qlen 1
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp0s5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
    state UP mode DEFAULT group default qlen 1000
   link/ether 00:1c:42:b4:fa:54 brd ff:ff:ff:ff:ff:ff
```

Соглашения об именовании носителей различаются. Текущие версии Linux пытаются гарантировать, что имена интерфейсов не меняются со временем, поэтому имена несколько произвольны (например, `enp0s5`). В системе FreeBSD и более ранних ядрах Linux используется традиционная схема нумерации "драйвер и экземпляр", в результате чего имена имеют вид `em0` или `eth0`.

Сетевое оборудование часто имеет настраиваемые параметры, характерные для его типа носителя и мало связанные с TCP/IP как таковым. Одним из распространенных примеров этого является современный Ethernet, в котором интерфейсная плата может поддерживать 10, 100, 1000 или даже 10000 Мбит/с в полудуплексном или полнодуплексном режиме. В большинстве устройств по умолчанию используется режим автосогласования, в котором как карта, так и ее восходящее соединение (обычно это порт коммутатора) пытаются угадать, что другие хотят использовать.

Исторически автоматическая настройка напоминала ковбоя, который пытается с завязанными глазами поймать теленка. Современные сетевые устройства взаимодействуют намного лучше, но автоматическая настройка по-прежнему является возможным источником сбоя. Высокие потери пакетов (особенно для больших пакетов) являются общим артефактом неудачной автоматической настройки.

Если вы вручную настроите сетевое соединение, то отключите автоматическую настройку с обеих сторон. Это вызывает интуитивное предположение, что можно вручную настроить одну сторону соединения, а затем позволить другой стороне автоматически адаптироваться к этим настройкам. Но, увы, на самом деле автоматическая настройка Ethernet работает не так. Все участники должны согласиться с тем, что сеть настроена либо автоматически, либо вручную.

Точный метод автоматической настройки аппаратных параметров варьируется в широких пределах. Мы обсудим этот вопрос подробнее в разделе 13.10.

Все сетевые протоколы, которые расположены выше уровня интерфейсного оборудования, имеют собственную конфигурацию. Протоколы IPv4 и IPv6 — это единствен-

ные протоколы, которые обычно требуется настроить, но важно понимать, что конфигурации определены для пар интерфейс-протокол. В частности, протоколы IPv4 и IPv6 являются полностью отдельными мирами, каждый из которых имеет свою собственную конфигурацию.

IP-конфигурация в основном связана с настройкой IP-адреса для интерфейса. Протокол IPv4 также должен знать маску подсети (сетевую маску) для подключенной сети, чтобы она могла различать сетевые и хостовые части адресов. На уровне сетевого трафика внутри и вне интерфейса IPv6 не использует сетевые маски; сетевая и хостовая части адреса IPv6 имеют фиксированный размер.

В протоколе IPv4 можно назначить широковещательный адрес на любой IP-адрес, действительный для сети, к которой подключен компьютер. Некоторые организации выбрали странные значения для широковещательного адреса в надежде избежать определенных типов атак типа "отказ в обслуживании", которые используют широковещательные эхо-запросы, но это рискованно и, вероятно, излишне. Неправильная настройка широковещательного адреса каждой машины может привести к широковещательным штормам, в которых пакеты перемещаются от машины к машине до истечения срока их TTL.[16]

Лучший способ избежать проблем с широковещательными сообщениями — не допустить, чтобы ваши пограничные маршрутизаторы пересылали их, и сделать так, чтобы отдельные хосты не реагировали на них. В протоколе IPv6 больше нет широковещательной передачи; она заменена различными формами многоадресатной рассылки.

Интерфейсу можно назначить несколько IP-адресов. В прошлом иногда было полезно сделать так, чтобы один сервер мог обслуживать несколько веб-сайтов; однако потребность в этой функции была заменена заголовком HTTP `Host` и функцией SNI для TLS (см. раздел 19.1).

Настройка маршрутизации

Маршрутизация рассматривается нами в этой и 15-й главе. И хотя информация по основам маршрутизации и командах **`ip route`** (Linux) и **`route`** (FreeBSD приведена здесь, полезно прочитать также несколько первых разделов главы 15.

Маршрутизация осуществляется на уровне протокола IP. Когда приходит пакет, предназначенный для другого хоста, его IP-адрес получателя сравнивается с записями в таблице маршрутизации ядра. При совпадении (хотя бы частичном) с каким-нибудь маршрутом в таблице пакет направляется по IP-адресу следующего шлюза, связанного с данным маршрутом.

Но есть два особых случая. Во-первых, пакет может быть адресован компьютеру, включенному в ту же сеть, что и хост-отправитель. В этом случае адрес следующего шлюза соответствует одному из интерфейсов локального компьютера, и пакет посылается прямо получателю. Маршруты такого типа добавляются командами **`ifconfig`** и **`ip address`** при конфигурировании интерфейса.

[16]Широковещательные штормы возникают из-за того, что один и тот же широковещательный адрес канального уровня должен использоваться для передачи пакетов независимо от того, какой для этой цели назначен широковещательный IP-адрес. Например, предположим, что машина X считает, что широковещательный адрес равен A1, а машина Y — A2. Если X отправит пакет по адресу A1, то Y получит пакет (поскольку адрес получателя канального уровня является широковещательным адресом), увидит, что пакет не предназначен для него и не отправлен по широковещательному адресу (потому что Y считает, что широковещательный адрес равен A2), поэтому может перенаправить пакет обратно в сеть. Если две машины находятся в состоянии Y, пакет будет циркулировать до истечения срока его TTL. Широковещательные штормы могут подорвать пропускную способность, особенно в большой коммутируемой сети.

Во-вторых, может вообще не оказаться маршрута, совпадающего с адресом получателя. В этом случае применяется стандартный маршрут (установленный по умолчанию), если таковой имеется, иначе отправителю посылается ICMP-сообщение "network unreachable" (сеть недоступна) или "host unreachable" (хост недоступен).

Многие локальные сети имеют единственный выход во внешний мир, поэтому им требуется только один маршрут, указывающий на этот выход. В магистральной сети Интернета нет стандартных маршрутов. Если для некоторого получателя в таблице маршрутизации не установлен маршрут, пакет не может быть доставлен.

Каждая команда **ip route** (Linux) или **route** (FreeBSD) добавляет или удаляет один маршрут. Прототипы этих команд выглядят следующим образом.

```
linux# ip route add 192.168.45.128/25 via zulu-gw.atrust.net
freebsd# route add -net 192.168.45.128/25 zulu-gw.atrust.net
```

Эта команда добавляет маршрут в сеть 192.168.45.128/25 через шлюз маршрутизатора `zulu-gw.atrust.net`, который должен быть либо соседним хостом, либо одним из локальных интерфейсов хоста. Естественно, маршрутизатор должен уметь преобразовывать имя `zulu-gw.atrust.net` в IP-адрес. Если ваш DNS-сервер находится на другой стороне шлюза, используйте числовой IP-адрес!

Сети назначения традиционно задаются отдельными IP-адресами и сетевыми масками, но в настоящее время все версии команды **route**, за исключением ее версии в системе HP-UX, понимают обозначения CIDR (например, 128.138.176.0/20). Система обозначений CIDR яснее, и пользователю остается лишь побеспокоиться о некоторых вопросах синтаксиса, зависящего от системы.

Перечислим еще несколько приемов.

- Если вы хотите увидеть имена, а не числа, то для того, чтобы проверить существующие маршруты, используйте команды **netstat -nr** или **netstat -r**. Числа часто удобнее при отладке, поскольку преобразование их в имена не всегда работает должным образом. Пример работы команды **netstat** приведен в разделе 13.5. В системе Linux команда **ip route show** считается стандартом де-факто для просмотра маршрутов. Однако, по нашему мнению, ее вывод менее понятен, чем вывод команды **netstat**.
- Для указания стандартного маршрута системы используйте ключевое слово **default** вместо адреса или сетевого имени. Мнемонически оно эквивалентно значению 0.0.0.0/0, что соответствует любому адресу и менее конкретно, чем любой реальный адрес пункта назначения маршрута.
- Для того чтобы удалить записи из таблицы маршрутизации, используйте команды **ip route delete** (Linux) или **route del** (FreeBSD).
- Для инициализации таблицы маршрутизации и начала работы используйте команды **ip route flush** (Linux) или **route flush** (FreeBSD).
- Маршруты в протоколе IPv6 задаются так же, как и маршруты в протоколе IPv4. Для того чтобы указать команде **ip route**, что используется адресное пространство протокола IPv6, необходимо использовать опцию **-6**. Как правило, команда **ip route** без проблем распознает маршруты IPv6 (проверяя формат адреса), но аргумент **-6** также допускается.
- Файл **/etc/networks** задает преобразование имен в сетевые номера аналогично тому, как файл **hosts** отображает имена хостов в IP-адреса. Такие команды, как **route**, ожидающие сетевой номер, могут понимать и имена, если они указаны в файле **networks**. Сетевые имена можно также получить из баз данных NIS или DNS (см. документ RFC1101).

Конфигурирование DNS

Для того чтобы сконфигурировать компьютер в качестве DNS-клиента, достаточно отредактировать файл **/etc/resolv.conf**. DNS-служба при этом, строго говоря, не нужна, но трудно представить себе ситуацию, в которой без нее можно было бы вообще обойтись.

Файл **resolv.conf** содержит список DNS-доменов, просматриваемых при анализе неполных имен (например, `anchor` вместо `anchor.cs.colorado.edu`), и список IP-адресов серверов имен, в которых осуществляется поиск имен. Ниже показан пример этого файла; подробнее о нем рассказывается в разделе 16.1.

```
search cs.colorado.edu colorado.edu
nameserver 128.138.242.1
nameserver 128.138.243.151
nameserver 192.108.21.1
```

Первым в файле **/etc/resolv.conf** должен быть указан ближайший стабильный сервер имен. Серверы опрашиваются по очереди и перерыв между последовательными опросами может оказаться довольно долгим. Всего можно задать три записи `nameserver`. Желательно указывать более одного сервера.

Если локальный хост получает адреса своих DNS-служб через протокол DHCP, то клиентское программное обеспечение протокола DHCP записывает адреса, полученные в аренду, в файл **resolv.conf**. Поскольку конфигурация DHCP в большинстве систем задана по умолчанию, обычно нет необходимости редактировать файл **resolv.conf** самостоятельно, если DHCP-сервер настроен корректно.

Многие серверы используют реализацию DNS-сервера Active Directory, созданную компанией Miscosoft. Она прекрасно работает с файлом **resolv.conf**, используемым в системах UNIX и Linux, и нет никакой необходимости придумывать что-либо другое.

Сетевое конфигурирование в различных системах

В ранних версиях систем UNIX настройка сетевой конфигурации осуществлялась путем редактирования сценариев системной загрузки и непосредственного изменения содержащихся в них команд. Современные системы содержат сценарии, доступные только для чтения; они охватывают множество сценариев конфигурирования и осуществляют выбор, основываясь на повторном использовании информации, содержащейся в других системных файлах, или на данных, содержащихся в файлах конфигурации.

Несмотря на то что разделение конфигурации и реализации — хорошая идея, каждая система осуществляет ее немного иначе, чем другая. Формат и способы использования файлов **/etc/hosts** и **/etc/resolv.conf** в системах UNIX и Linux довольно хорошо согласованы, но этого нельзя со всей определенностью сказать обо всех системах.

Многие системы обеспечивают графический пользовательский интерфейс для выполнения основных задач, связанных с конфигурированием, однако соответствие между визуальным интерфейсом и файлами конфигурации часто остается неясным. Кроме того, графический пользовательский интерфейс часто игнорирует сложные варианты конфигурации и остается довольно неудобным для удаленного и автоматизированного администрирования.

В следующих разделах мы раскритикуем некоторые варианты конфигурирования, опишем то, что происходит за кулисами, и раскроем детали сетевого конфигурирования для каждой из популярных операционных систем. В частности, мы рассмотрим следующие вопросы.

- Основная конфигурация.
- Конфигурация DHC-клиента.
- Динамическое конфигурирование и настройка.
- Безопасность, брандмауэры, фильтрация и конфигурация NAT.

Следует иметь в виду, что в большинстве случаев сетевое конфигурирование осуществляется в момент загрузки, поэтому эта тема частично перекрывается с информацией, представленной в главе 2.

13.10. Сетевое конфигурирование в системе Linux



Разработчики системы Linux любят чинить ее на скорую руку и часто реализуют функциональные возможности и алгоритмы, пока не имеющие принятых стандартов. Примером этого является включение сменных алгоритмов контроля за перегрузкой в ядро системы Linux версии 2.6.13. Некоторые функциональные возможности предусматривают варианты для сетей с потерями (lossy networks), высокоскоростных глобальных вычислительных сетей (wide area network — WAN) с большими потерями пакетов, спутниковыми каналами связи и т.д. Стандартный механизм "reno", реализованный в протоколе TCP (медленный старт, избегание перегрузок, быстрая повторная пересылка и быстрое восстановление), по-прежнему используется по умолчанию, но в конкретной среде может оказаться более приемлемым другой вариант.[17]

После внесения любых изменений в файл, управляющий сетевой конфигурацией в момент загрузки, необходимо либо перезагрузить систему, либо отключить сетевой интерфейс, а затем снова включить его, чтобы изменения вступили в силу. В большинстве систем Linux для этой цели можно использовать команды **ifdown** *интерфейс* или **ifup** *интерфейс*, хотя их реализации не являются идентичными.

Программа NetworkManager

Поддержка работы в мобильной сети, осуществляемая в системе Linux, некоторое время была довольно неорганизованной, пока в 2004 г. не появилась программа NetworkManager. Она состоит из службы, предназначенной для непрерывного функционирования, а также приложения для работы с областью уведомлений, позволяющего конфигурировать индивидуальные сетевые интерфейсы. Кроме разнообразных проводных сетей, программа NetworkManager поддерживает также работу беспроводных сетей, систем беспроводного широкополосного доступа и частных виртуальных сетей (virtual private network — VPN). Она постоянно осуществляет оценку доступных сетей и переключает службу на "предпочтительные" сети, как только те становятся доступными. Наиболее предпочтительными являются проводные сети, за ними следуют обычные беспроводные сети.

Все это довольно сильно изменило сетевую конфигурацию системы Linux. Помимо того что она стала более изменчивой по сравнению с традиционной статической конфигурацией, теперь она запускается и управляется пользователем, а не системным администратором. Программа NetworkManager широко внедрена в дистрибутивы систем Linux, включая все экземпляры, рассматриваемые в книге, но для того, чтобы избежать нарушения существующих сценариев и настроек, она обычно предусматривается как

[17] Дополнительную информацию об использовании альтернативных алгоритмов управления см. на сайте `lwn.net/Articles/701165`.

"параллельная вселенная" в дополнение к традиционной сетевой конфигурации, которая использовалась в прошлом.

Система Debian запускает программу NetworkManager по умолчанию, сохраняя при этом статически конфигурируемые сетевые интерфейсы за пределами ее досягаемости. Системы Red Hat и CentCisco вообще не запускают программу NetworkManager по умолчанию.

Программа NetworkManager в основном используется в ноутбуках, поскольку их сетевое окружение может часто изменяться. Для серверов и настольных систем программа NetworkManager не обязательна и фактически может даже усложнять администрирование. В этом окружении она должна игнорироваться или отключаться.

Команда ip: ручное конфигурирование сети

В прошлом системы Linux и UNIX использовали одни и те же основные команды для настройки сети и проверки состояния: **ifconfig**, **route** и **netstat**. Они попрежнему доступны в большинстве дистрибутивов, но активная разработка перешла к пакету **iproute2**, который содержит команды **ip** (для большинства повседневных сетевых конфигураций, включая маршрутизацию) и **ss** (приблизительный аналог команды **netstat** для изучения состояния сетевых сокетов).

Если вы привыкли к традиционным командам, стоит попробовать перестроиться на команду **ip**. Унаследованные команды не будут выполняться вечно, и хотя они охватывают общие сценарии конфигурации, они не предоставляют доступ к полному набору функций сетевого стека Linux. Команда **ip** работает понятнее и точнее.

Второй аргумент команды **ip** указывает, какой объект вы хотите настроить или изучить. Существует много вариантов, но главные — это **ip link** для настройки сетевых интерфейсов, **ip address** — для привязки сетевых адресов к интерфейсам и **ip route** — для изменения или печати таблицы маршрутизации[18]. Большинство объектов понимают команды **list** или **show**, предназначенные для вывода их текущего состояния, поэтому команда **ip link show** выводит список сетевых интерфейсов, **ip route show** выводит текущую таблицу маршрутизации, а **ip address show** отображает список назначенных IP-адресов.

Справочные man-страницы для команды **ip** разделяются в соответствии с подкомандами. Например, чтобы просмотреть подробную информацию о конфигурации интерфейса, выполните команду **man ip-link**. Вы также можете выполнить команду **ip link help**, чтобы увидеть короткую шпаргалку.

Команда **ifconfig** в системе UNIX объединяет концепцию конфигурации интерфейса с концепцией настройки параметров для определенного сетевого протокола. Фактически несколько протоколов могут выполняться на определенном сетевом интерфейсе (первый пример — это одновременная поддержка протоколов IPv4 и IPv6), и каждый из этих протоколов может поддерживать несколько адресов, поэтому разделение между командами **ip link** и **ip address** является действительно разумным решением. Большинство системных администраторов, которые традиционно думают о "конфигурации интерфейса", на самом деле занимаются настройкой протоколов IPv4 и IPv6.

Команда **ip** принимает аргумент **-4** или **-6** для явной настройки протокола IPv4 или IPv6, но необходимость указывать эти параметры возникает редко. Команда **ip** определяет правильный режим, просто просматривая формат адресов, которые вы предоставляете.

Основная конфигурация интерфейса выглядит так:

```
# ip address add 192.168.1.13/26 broadcast 192.168.1.63 dev enp0s5
```

[18]Аргументы команды **ip** можно заменять сокращениями, поэтому **ip ad** — это эквивалент **ip address**. Для ясности мы приводим полные имена.

В данном случае раздел **broadcast** является излишним, потому что в любом случае это значение будет установлено по умолчанию с учетом сетевой маски. Но если бы вам потребовалось задать этот параметр явным образом, это можно сделать именно так, как показано выше.

Конечно, в повседневной жизни вы обычно не настраиваете сетевые адреса вручную. В следующих разделах описывается, как наши демонстрационные дистрибутивы обрабатывают статическую конфигурацию сети с точки зрения файлов конфигурации.

Сетевое конфигурирование в системе Ubuntu



Как показано в табл. 13.6, система Ubuntu задает конфигурацию сети в файлах **/etc/hostname/** и **/etc/network/interfaces**, а справочная информация записана в файле **/etc/network/options**.

Таблица 13.6. Сетевая конфигурация системы Ubuntu, записанная в каталоге /etc

Файл	Содержимое
hostname	Имя компьютера
network/interfaces	IP-адрес, сетевая маска, стандартный маршрут

Имя компьютера задается в файле **/etc/hostname/**. Имя, записанное в этом файле, должно быть полностью квалифицированным; оно используется в самых разных контекстах, некоторые из которых требуют полной квалификации.

IP-адрес, сетевая маска и стандартный шлюз задаются в файле **/etc/network/interfaces**. Каждый интерфейс задается строкой, начинающейся с ключевого слова `iface`. За этой строкой могут следовать строки, задающие дополнительные параметры. Рассмотрим пример.

```
auto lo enp0s5
iface lo inet loopback
iface enp0s5 inet static
     address 192.168.1.102
     netmask 255.255.255.0
     gateway 192.168.1.254
```

Команды **ifup** и **ifdown** читают этот файл и, соответственно, подключают или отключают интерфейсы, вызывая низкоуровневые команды (например, **ifconfig**) с соответствующими параметрами. Строка `auto` задает интерфейсы, которые должны включаться при загрузке или при выполнении команды **ifup -a**.

Ключевое слово `inet` в строке `iface` определяет семейство адресов протокола IPv4. Для настройки протокола IPv6 включите конфигурацию `inet6`.

Ключевое слово `static` обозначает способ описания интерфейса и указывает на то, что IP-адрес и сетевая маска интерфейса `enp0s5` будут заданы непосредственно. Для статических интерфейсов строки `address` и `netmask` являются обязательными. В строке `gateway` определяется адрес стандартного шлюза.

```
iface enp0s5 inet dhcp
```

Сетевое конфигурирование в системе Red Hat и CentOS

RHEL 

Сетевая конфигурация систем Red Hat и CentOS записана в каталоге **/etc/sysconfig**. Основные файлы конфигурации перечислены в табл. 13.7.

Таблица 13.7. Сетевые конфигурационные файлы SuSE из каталога /etc/sysconfig/network

Файл	Содержимое
network	Имя компьютера, стандартный маршрут
network-scripts/ifcfg-*имя_инт*	Параметры интерфейсов: IP-адреса, сетевые маски и т.д.
network-scripts/route-*имя_инт*	Параметры маршрута: аргументы команды **ip route**

Имя компьютера записывается в файл **/etc/sysconfig/network**, который также содержит имя домена DNS и информацию о стандартном шлюзе. (По существу, именно в этом файле указаны все сетевые настройки, не зависящие от интерфейса.)

Например, ниже приведено содержимое файла **network** для компьютера с единственным интерфейсом Ethernet.

```
NETWORKING=yes
NETWORKING_IPV6=no
HOSTNAME=redhat.toadranch.com
DOMAINNAME=toadranch.com        ### необязательно
GATEWAY=192.168.1.254
```

Данные, связанные с интерфейсом, хранятся в файле **/etc/sysconfig/network-scripts/ifcfg-***имя_инт*, где параметр *имя_инт* — имя сетевого интерфейса. Эти файлы конфигурации задают IP-адрес, сетевую маску, сеть и широковещательный адрес для каждого интерфейса. Они также содержат строку, указывающую, должен ли интерфейс включаться в момент загрузки.

Типичный компьютер имеет файлы для интерфейса Ethernet (`eth0`) и интерфейс обратной связи (`lo`). Вот каким будет содержимое файлов **ifcfg-ifcfg-eth0** и **ifcfg-lo** для хоста `redhat.toadranch.com`, описанного выше в файле **network**.

```
DEVICE=eth0
IPADDR=192.168.1.13
NETMASK=255.255.255.0
NETWORK=192.168.1.0
BROADCAST=192.168.1.255
ONBOOT=yes
```

и

```
DEVICE=lo
IPADDR=127.0.0.1
NETMASK=255.0.0.0
NETWORK=127.0.0.0
BROADCAST=127.255.255.255
ONBOOT=yes
NAME=loopback
```

Настройки интерфейса `eth0` на основе протокола DHCP выглядят еще проще.

```
DEVICE=eth0
BOOTROTO=dhcp
ONBOOT=yes
```

После изменения информации о конфигурации в каталоге **/etc/sysconfig** следует выполнить команду **ifdown** *имя_инт*, а затем **ifup** *имя_инт*, где *имя_инт* — имя соответствующего интерфейса. Для одновременной конфигурации нескольких интерфейсов можно использовать команду **service network restart**, которая перезагружает сеть.

Строки в файле **network-scripts/route-***имя_инт* передаются как аргументы команды **ip route** при конфигурировании соответствующего интерфейса. Например, строка

```
default via 192.168.0.1
```

задает стандартный маршрут. На самом деле эта конфигурация не зависит от интерфейса, но для ясности ее следует указать в файле, соответствующем конкретному интерфейсу, которому будут передаваться пакеты, поступающие по маршруту, установленному по умолчанию.

Настройка сетевого оборудования в системе Linux

Команда **ethtool** запрашивает и устанавливает параметры сетевого интерфейса, характерные для среды, например скорость связи и дуплекс. Она представляет собой замену старой команды **mii-tool**, но в некоторых системах используются обе команды. Если утилита **ethtool** не инсталлирована по умолчанию, она обычно включается в дополнительный пакет (под тем же названием).

Запросить состояние интерфейса можно, просто назвав его имя. Например, интерфейс `eth0` (типичная сетевая карта на материнской плате персонального компьютера), описанный ниже, допускает автонастройку и в настоящий момент работает на предельной скорости.

```
# ethtool eth0
Settings for eth0:
        Supported ports: [ TP MII ]
        Supported link modes:     10baseT/Half     10baseT/Full
                                  100baseT/Half    100baseT/Full
                                  1000baseT/Half   1000baseT/Full
        Supports auto-negotiation: Yes
        Advertises link modes:    10baseT/Half     10baseT/Full
                                  100baseT/Half    100baseT/Full
                                  1000baseT/Half   1000baseT/Full
        Advertises auto-negotiation: Yes
        Speed: 1000Mb/s
        Duplex: Full
        Port: MII
        PHYAD: 0
        Transceiver: internal
        Auto-negotiation: on
        Supports Wake-on: pumbg
        Wake-on: g
        Current message level: 0x00000033 (51)
        Link detected: yes
```

Для того чтобы перевести этот интерфейс в полнодуплексный режим на скорости 100 Мб/с, используется следующая команда.

```
# ethtool -s eth0 speed 100 duplex full
```

Если вы пытаетесь определить, надежна ли автонастройка в вашей среде, то может оказаться полезной команда **ethtool -r**. Она вызывает мгновенную перенастройку параметров связи.

Еще одним полезным вариантом является опция **-k**, которая показывает, какая из задач, связанных с протоколом, была назначена сетевому интерфейсу, а не выполняется ядром. Большинство интерфейсов может вычислять контрольные суммы, а также вы-

полнять сегментацию. Если вы сомневаетесь, что сетевой интерфейс надежно выполняет эти задания, то лучше их отключить. Для включения и отключения этих функций используется команда **ethtool -K** в сочетании с другими опциями. (Опция **-k** демонстрирует текущее состояние, а опция **-K** устанавливает его.)

Любые изменения, сделанные с помощью команды **ethtool**, являются временными. Если вы хотите, чтобы они стали постоянными, то следует убедиться, что команда **ethtool** выполняется как часть конфигурации сетевого интерфейса. Для этого лучше всего сделать ее частью конфигурации каждого интерфейса; если просто выполнять какие-то команды **ethtool** во время загрузки, то ваша конфигурация не будет правильно учитывать случаи, когда интерфейсы запускаются вновь без перезагрузки системы.



Работая с системами Red Hat и CentOS, можно включить в файл конфигурации интерфейса, расположенный в каталоге **/etc/sysconfig/network-scripts**, строку `ETHTOOL_OPTS=line`. В качестве аргумента команда **ifup** передает команде **ethtool** целую строку.



В системах Debian и Ubuntu команду **ethtool** можно выполнить в рамках сценария `post-up`, заданного в конфигурации интерфейса в каталоге **/etc/network/interfaces**.

Опции протокола Linux TCP/IP

Система Linux помещает представление каждой настраиваемой переменной ядра в виртуальную файловую систему **/proc**. Информацию о файловой системе **/proc** см. в разделе 11.4.

Сетевые переменные находятся в каталогах **/proc/sys/net/ipv4** и **/proc/sys/net/ipv6**.

Каталог **ipv4** содержит гораздо больше параметров, чем каталог **ipv6**, но в основном это объясняется тем, что протоколы, не зависящие от IP-версии, такие как TCP и UDP, хранят свои параметры в каталоге **ipv4**. Префикс, такой как **tcp_** или **udp_**, сообщает вам, к какому протоколу относится тот или иной параметр.

Подкаталоги **conf** в каталогах **ipv4** и **ipv6** содержат параметры, заданные для каждого интерфейса. Они включают подкаталоги **all** и **default**, а также подкаталог для каждого интерфейса (включая обратную связь). Каждый подкаталог имеет одинаковый набор файлов.

```
ubuntu$ ls -F /proc/sys/net/ipv4/conf/default
accept_local        drop_gratuitous_arp                 proxy_arp
accept_redirects    drop_unicast_in_l2_multicast        proxy_arp_pvlan
accept_source_route force_igmp_version                  route_localnet
arp_accept          forwarding                          rp_filter
arp_announce        igmpv2_unsolicited_report_interval  secure_redirects
arp_filter          igmpv3_unsolicited_report_interval  send_redirects
arp_ignore          ignore_routes_with_linkdown         shared_media
arp_notify          log_martians                        src_valid_mark
bootp_relay         mc_forwarding                       tag
disable_policy      medium_id
disable_xfrm        promote_secondaries
```

Например, если изменить переменную в подкаталоге **conf/enp0s5**, то это изменение будет относиться только к данному интерфейсу. Если изменить значение в каталоге **conf/all**, то можно ожидать, что соответствующее изменение произойдет во всех существующих интерфейсах, но не факт, что это произойдет на самом деле. Для каждой

переменной существуют собственные правила принятия изменений с помощью каталога **all**. Некоторые изменения применяются к существующим значениям по правилу OR, некоторые — по правилу AND, а остальные — по правилам MAX и MIN. Кроме исходного кода ядра, этот процесс больше не описан ни в одном документе, поэтому таких неопределенных ситуаций лучше всего избегать, ограничивая изменения отдельными интерфейсами.

Если изменить переменную в каталоге **conf/default**, то новое значение будет относиться ко всем интерфейсам, которые будут конфигурироваться позднее. С другой стороны, лучше всего использовать значения, принятые по умолчанию, лишь для справки; это позволит восстановить нормальное функционирование системы в случае неудачного конфигурирования.

Каталоги **/proc/sys/net/ipv4/neigh** и **/proc/sys/net/ipv6/neigh** также содержат подкаталоги для каждого интерфейса. Файлы в каждом из этих подкаталогов управляют ARP-таблицей и выявлением "соседей" данного интерфейса по протоколу IPv6. Приведем список этих переменных; переменные, имена которых начинаются с букв gc (garbage collection — сборка мусора), определяют, каким образом устаревают и аннулируются записи в ARP-таблице.

```
ubuntu$ ls -F /proc/sys/net/ipv4/neigh/default
anycast_delay             gc_interval      locktime          retrans_time
app_solicit               gc_stale_time    mcast_resolicit   retrans_time_ms
base_reachable_time       gc_thresh1       mcast_solicit     ucast_solicit
base_reachable_time_ms    gc_thresh2       proxy_delay       unres_qlen
delay_first_probe_time    gc_thresh3       proxy_qlen        unres_qlen_bytes
```

Для того чтобы увидеть значение переменной, следует выполнить команду **cat**. Для того чтобы задать значение переменной, следует применить команду **echo** к соответствующему имени файла, хотя на практике это проще сделать с помощью команды **sysctl**, которая представляет собой интерфейс командной строки для тех же значений.

Например, команда

```
ubuntu# cat /proc/sys/net/ipv4/icmp_echo_ignore_broadcast
0
```

показывает, что значение этой переменной равно нулю, т.е. широковещательная проверка связи не игнорируется. Для того чтобы установить это значение равным единице (и тем самым предотвратить атаку DoS типа Smurf), следует выполнить команду

```
ubuntu# sudo sh -c "echo 1 > icmp_echo_ignore_broadcasts"[19]
```

в каталоге **/proc/sys/net** или

```
ubuntu$ sysctl net.ipv4.icmp_echo_ignore_broadcasts=1
```

Имена переменных в команде **sysctl** представляют собой имена путей относительно каталога **/proc/sys**. Традиционным разделителем являются точки, но команда **sysctl** также допускает использование обратной косой черты.

[19]Если попытаться выполнить эту команду в форме **sudo echo 1 > icmp_echo_ignore_broadcasts**, то будет сгенерировано сообщение "в разрешении отказано" ("permission denied"), потому что ваша оболочка пытается открыть выходной файл до выполнения команды **sudo**. Вы же хотите применить команду **sudo** как к команде **echo**, так и к перенаправлению. Следовательно, вы должны создать корневую подоболочку (root subshell), в которой необходимо выполнять всю команду целиком.

Обычно вы зарегистрированы в той же сети, которую пытаетесь настроить, поэтому будьте осторожны! Вы можете так запутать настройки, что придется перегружать систему с консоли, а это может оказаться неудобным, если система, например, находится в Пойнт-Барроу, штат Аляска, и за окном январь. Прежде чем даже просто подумать о настройке головного компьютера, проведите тестирование на своей настольной системе.

Для того чтобы указанные изменения стали постоянными (или, говоря точнее, чтобы они восстанавливались при каждой перезагрузке системы), добавьте соответствующие переменные в каталог **/etc/sysctl.conf**, который считывается командой **sysctl** во время загрузки. Например, строка

```
net.ipv4.ip_forward=0
```

в файле **/etc/sysctl.conf** отключает пересылку IP-пакетов на заданный хост.

Некоторые подкаталоги каталога **/proc** документированы лучше, чем другие. Рекомендуем обратиться к разделу 7 справочной системы протокола. Например, команда **man 7 icmp** документирует четыре из шести возможных вариантов. (При этом необходимо, чтобы справочные страницы о ядре системы Linux были инсталлированы заранее.)

Кроме того, полезные комментарии можно найти в файле **ip-sysctl.txt**, находящемся в дистрибутивах исходного кода ядра. Если исходный код ядра не инсталлирован, то отправьте в поисковую систему запрос `ip-sysctl-txt`.

Переменные ядра, связанные с безопасностью

В табл. 13.8 описано стандартное поведение систем Linux в отношении различных сетевых методик обеспечения безопасности. Все они кратко рассмотрены выше. Мы рекомендуем установить соответствующие параметры так, чтобы система не отвечала на широковещательные ICMP-запросы, не подчинялась директивам переадресации и не принимала пакеты, маршрутизируемые по адресу отправителя. Все эти параметры должны быть установлены по умолчанию, за исключением **accept_redirects**.

Таблица 13.8. Режимы безопасности в системе Linux, заданные по умолчанию

Функциональная возможность	Реализация на простом хосте	Реализация на шлюзе	Управляющий файл (в каталоге /proc/sys/net/ipv4)
Пересылка IP-пакетов	Отключена	Включена	**ip_forward** для всей системы **conf/***интерфейс***/forwarding** для каждого интерфейса [a]
Переадресующие ICMP-пакеты	Принимаются	Игнорируются	**conf/***интерфейс***/accept_redirects**
Маршрутизация по адресу отправителя"	Отключена	Отключена	**conf/***интерфейс***/accept_source_route**
Широковещательное тестирование	Игнорируется	Игнорируется	**icmp_echo_ignore_broadcasts**

[a] В качестве параметра *интерфейс* может быть задано имя конкретного интерфейса или ключевое слово **all**.

13.11. Сеть FreeBSD



Будучи прямым потомком линии BSD, операционная система FreeBSD остается чем-то вроде эталонной реализации протокола TCP/IP. В нем не хватает многих разработок, которые усложняют сетевой стек Linux. С точки зрения системных администраторов, конфигурация сети FreeBSD проста и ясная.

Команда `ifconfig`: настройка сетевых интерфейсов

Команда **ifconfig** включает или отключает сетевой интерфейс, устанавливает его IP-адрес и маску подсети и задает различные другие опции и параметры. Обычно она выполняется во время загрузки с параметрами командной строки, взятыми из файлов конфигурации, но ее также можно запускать вручную для внесения изменений. Будьте осторожны, внося изменения с помощью команды **ifconfig**, если вы зарегистрировались удаленно, — многие системные администраторы были заблокированы в таких ситуациях и вынуждены были приезжать лично, чтобы исправить ситуацию.

Команда **ifconfig** чаще всего имеет форму

ifconfig *интерфейс* [*семейство*] *адрес параметры* ...

Например, команда

```
ifconfig em0 192.168.1.13/26 up
```

устанавливает адрес IPv4 и сетевую маску, связанную с интерфейсом `em0`, и готовит интерфейс для использования.

Параметр *интерфейс* идентифицирует аппаратный интерфейс, к которому применяется команда. Интерфейс обратной связи называется `lo0`. Имена реальных интерфейсов различаются в зависимости от их аппаратных драйверов. Команда **ifconfig -a** выводит список сетевых интерфейсов системы и их текущие настройки.

Параметр *семейство* указывает команде **ifconfig**, какой сетевой протокол ("семейство адресов") вы хотите настроить. Вы можете настроить несколько протоколов на одном интерфейсе и использовать их все одновременно, но настраивать их необходимо по отдельности. Основные параметры здесь — **inet** для IPv4 и **inet6** для IPv6; **inet** предполагается, если вы опустите параметр.

Параметр *адрес* указывает IP-адрес интерфейса. Здесь также допустимо имя хоста, но оно должно определено для IP-адреса во время загрузки. Для основного интерфейса машины это означает, что имя хоста должно быть указано в файле локальных хостов, поскольку другие методы определения имен зависят от инициализации сети.

Ключевое слово **up** включает интерфейс; **down** отключает его. Когда команда **ifconfig** назначает интерфейсу IP-адрес, как в приведенном выше примере, параметр **up** является неявным и не должен упоминаться по имени.

Для сетей, имеющих подсети, можно указать сетевую маску в стиле CIDR, как показано в примере выше, или включить отдельный аргумент **netmask**. Маска может быть указана в десятичной системе с точками или в виде 4-байтового шестнадцатеричного числа, начинающегося с **0x**.

Параметр **broadcast** указывает IP-широковещательный адрес интерфейса, выраженный в шестнадцатеричной или точечной четырехзначной нотации. Широковещательный адрес по умолчанию — это номер, в котором машинная часть состоит из одних единиц. В приведенном выше примере использования команды **ifconfig** автоматически настроенный широковещательный адрес равен 192.168.1.61.

Конфигурация сетевого оборудования в системе FreeBSD

В системе FreeBSD нет специальной команды, аналогичной **ethertool** в системе Linux. Вместо этого команда **ifconfig** передает информацию о конфигурации вплоть до драйвера сетевого интерфейса через разделы **media** и **mediaopt**. Допустимые значения этих параметров зависят от аппаратного обеспечения. Чтобы найти их список, прочитайте справочную страницу для конкретного драйвера.

Например, интерфейс с именем em0 использует драйвер "em". На справочной странице **man 4 em** показано, что этот драйвер предназначен для определенных типов проводного Ethernet-оборудования на базе Intel. Чтобы заставить этот интерфейс работать в гигабитном режиме с использованием четырехпарной кабельной системы (типичная конфигурация), команда должна иметь вид:

```
# ifconfig em0 media 1000baseT mediaopt full-duplex
```

Вы можете включить эти параметры среды передачи данных вместе с другими параметрами конфигурации для интерфейса.

Конфигурирование сети во время загрузки системы FreeBSD

Статическая система конфигурации системы FreeBSD довольно простая. Все параметры сети находятся в файле **/etc/rc.conf**, а также в других системных настройках. Вот типичная конфигурация:

```
hostname="freebeer"
ifconfig_em0="inet 192.168.0.48 netmask 255.255.255.0"
defaultrouter="192.168.0.1"
```

Каждый сетевой интерфейс имеет собственную переменную **ifconfig_***. Значение переменной просто передается команде **ifconfig** как ряд аргументов командной строки. Раздел defaultrouter определяет сетевой шлюз по умолчанию.

Чтобы получить сетевую конфигурацию системы с DHCP-сервера, используйте следующий параметр:

```
ifconfig_em0="DHCP"
```

Эта форма является "черным ящиком" и не передается команде **ifconfig**, которая не знает, как интерпретировать аргумент **DHCP**. Вместо этого она активирует сценарии запуска команды **dhclient em0**. Чтобы изменить рабочие параметры системы DHCP (время ожидания и т.д.), установите их в файле **/etc/dhclient.conf**. Версия этого файла по умолчанию пуста, за исключением комментариев, и обычно ее не нужно изменять.

Если вы измените конфигурацию сети, то можете выполнить команду **service netif restart**, чтобы повторить начальную процедуру настройки. Если вы изменили параметр defaultrouter, также выполните команду **service routing restart**.

Конфигурирование протокола TCP/IP в системе FreeBSD

Параметры сети на уровне ядра FreeBSD контролируются аналогично настройкам Linux (см. раздел 13.10), за исключением того, что в ней нет иерархии **/proc**, в которой можно использовать права пользователя root. Вместо этого выполоните команду **sysctl -ad**, чтобы просмотреть доступные параметры и их однострочные описания. Их много (5495 на FreeBSD 11), поэтому вам нужно отобрать с помощью утилиты grep вероятных кандидатов, таких как "redirect" или "^net".

В табл. 13.9 приведен список параметров, связанных с безопасностью.

Таблица 13.9. Параметры сетевой безопасности в системе FreeBSD

Параметр	Ст. зн.	Что он делает, когда установлено значение 1
net.inet.ip.forwarding	0	Действует как маршрутизатор для пакетов IPv4
net.inet6.ip6.forwarding	0	Действует как маршрутизатор для пакетов IPv6
net.inet.tcp.blackhole	0	Отключает сообщения о недостижимости пакетов для закрытых портов
net.inet.udp.blackhole	0	Не отправляет RST-пакеты для закрытых портов TCP
net.inet.icmp.drop_redirect	0	Игнорирует перенаправления IPv4 ICMP
net.inet6.icmp6.rediraccept	1	Принимает (подчиняется) ICMP-перенаправлениям по протоколу IPv6
net.inet.ip.accept_sourceroute	0	Разрешает маршрутизацию пакетов IPv4 по адресу отправителя

Параметры `blackhole` потенциально полезны для систем, которые вы хотите защитить от сканеров портов, но они изменяют стандартное поведение протоколов UDP и TCP. Вы также можете отключить прием переадресаций ICMP для протоколов IPv4 и IPv6.

Вы можете установить параметры в работающем ядре с помощью команды **sysctl**. Например,

```
$ sudo sysctl net.inet.icmp.drop_redirect=1
```

Для установки параметров во время загрузки поместите их в файл **/etc/sysctl.conf**.

```
net.inet.icmp.drop_redirect=1
```

13.12. Сетевые проблемы

Для отладки сети на уровне протокола TCP/IP доступно несколько хороших инструментов. Большинство из них предоставляют информацию на низком уровне, поэтому для их использования вы должны понимать основные идеи протокола TCP/IP и маршрутизации.

В этом разделе мы начнем с некоторой общей стратегии устранения неполадок. Затем рассмотрим несколько важных инструментов, включая **ping**, **traceroute**, **tcpdump** и Wireshark. В этой главе не обсуждаются команды **arp**, **ndp**, **ss** или **netstat**, хотя они также являются полезными инструментами отладки.

Прежде чем атаковать свою сеть, учтите следующие принципы.

- Делайте по одному изменению за раз. Проверьте каждое изменение, чтобы убедиться, что оно возымело эффект, на который вы рассчитывали. Откажитесь от любых изменений, которые имеют нежелательный эффект.
- Зафиксируйте в документе ситуацию, которая сложилась до вашего вмешательства, и записывайте все вносимые вами изменения.
- Начните с одного конца системы или сети и пройдитесь по критическим компонентам системы, пока не обнаружите проблему. Например, вы можете начать с просмотра конфигурации сети на клиентской стороне, пройти весь путь вплоть до физических подключений, исследовать сетевое оборудование и, наконец, проверить физические подключения и конфигурацию программного обеспечения сервера.
- Или используйте слои сети для устранения проблемы. Начните сверху вниз или снизу вверх и пройдитесь по стеку протокола.

Этот последний момент заслуживает более подробного обсуждения. Как описано в разделе 13.2, архитектура TCP/IP определяет несколько уровней абстракции, на которых могут функционировать компоненты сети. Например, HTTP зависит от TCP, TCP зависит от IP, IP зависит от протокола Ethernet, а протокол Ethernet зависит от целостности сетевого кабеля. Вы можете значительно сократить время, затрачиваемое на отладку проблемы, если сначала выясните, какой слой работает плохо.

Проходя по стеку сверху вниз или снизу вверх, задавайте себе такие вопросы.

- Есть ли у вас физическая связь и горят ли светодиоды соединения?
- Правильно ли настроен ваш интерфейс?
- Показывают ли ваши таблицы ARP другие хосты?
- Существует ли межсетевой экран (брандмауэр) на вашем локальном компьютере?
- Существует ли межсетевой экран где-то между вами и пунктом назначения?
- Если задействованы брандмауэры, передают ли они пакеты и ответы ICMP эхо-запросов?
- Можете ли вы выполнить эхо-запрос по адресу локального хоста (127.0.0.1)?
- Можете ли вы выполнить эхо-запросы к другим локальным хостам по их IP-адресам?
- Правильно ли работает DNS?[20]
- Можете ли вы посылать эхо-запросы другим локальным хостам по имени?
- Можете ли вы посылать эхо-запросы хостам в другой сети?
- Работают ли службы высокого уровня, такие как веб-серверы и SSH-серверы?
- Вы действительно проверяли брандмауэры?

После того как вы определили, где проблема, и нашли решение, изучите эффект, который вызовут ваши последующие проверки и предполагаемые исправления по отношению к другим службам и хостам.

Команда `ping`: проверьте, работает ли хост

Команда `ping` и ее IPv6-аналог `ping6` крайне просты, но во многих ситуациях они являются единственными командами, которые необходимы для сетевой отладки. Они отправляют ICMP-пакет `ECHO_REQUEST` на целевой хост и ждут, вернет ли хост ответ.

Команду `ping` можно использовать для проверки состояния отдельных хостов и тестирования сегментов сети. В обработку эхо-запроса вовлечены все таблицы маршрутизации, физические сети и шлюзы, поэтому сеть должна работать более или менее устойчиво, чтобы команда `ping` завершилась успешно. Если команда `ping` не работает, вы можете быть уверены, что ничего более сложного работать тоже не будет.

Однако это правило не применяется к сетям или хостам, которые блокируют эхо-запросы ICMP с помощью брандмауэра.[21] Прежде чем сделать вывод о том, что целевой

[20] Если машина подвисает во время загрузки, загружается очень медленно или зависает от входящих соединений SSH, основной подозреваемой должна быть система DNS. В большинстве систем используется механизм преобразования имен, который настраивается в файле `/etc/nsswitch.conf`. Если в системе запущен `nscd`, демон кеширования службы имен, этот компонент заслуживает некоторого подозрения. Если демон `nscd` выходит из строя или неправильно сконфигурирован, это влияет на поиск имен. Используйте команду `getent`, чтобы проверить правильность работы ваших механизмов преобразования имен и серверов имен (например, `getent hosts google.com`).

[21] Обратите внимание на то, что в последних версиях операционной системы Windows эхо-запросы по умолчанию блокируются.

хост игнорирует эхо-запрос, убедитесь, что брандмауэр не мешает вашей отладке. Чтобы облегчить отладку, можно ненадолго отключить промежуточный межсетевой экран.

Если ваша сеть находится в плохом состоянии, возможно, что не работает система DNS. Упростите ситуацию, используя числовые IP-адреса при выполнении эхо-запросов, и примените параметр **-n** команды **ping** для предотвращения попыток выполнить обратный поиск по IP-адресам — этот поиск также инициирует запросы DNS.

Помните о проблеме брандмауэра, если вы используете команду **ping** для проверки возможности подключения к Интернету. Одни известные серверы отвечают на пакеты эхо-запросов, а другие нет. Мы обнаружили, что сервер **google.com** всегда отвечает на эхо-запросы.

Если не задать аргумент, устанавливающий количество пакетов, то большинство версий команды **ping** будут выполняться в бесконечном цикле. После того как вы выполнили эхо-запрос, введите символ прерывания (обычно <Ctrl+C>), чтобы выйти.

Вот пример:

```
linux$ ping beast
PING beast (10.1.1.46): 56 bytes of data.
64 bytes from beast (10.1.1.46): icmp_seq=0 ttl=54 time=48.3ms
64 bytes from beast (10.1.1.46): icmp_seq=1 ttl=54 time=46.4ms
64 bytes from beast (10.1.1.46): icmp_seq=2 ttl=54 time=88.7ms
^C
--- beast ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2026ms
rtt min/avg/max/mdev = 46.490/61.202/88.731/19.481 ms
```

Вывод для сервера `beast` показывает IP-адрес хоста, порядковый номер ICMP каждого ответного пакета и время прохождения в оба конца. Вышеприведенный вывод говорит о том, что сервер `beast` работает и подключен к сети.

Номер последовательности ICMP является особенно ценной информацией. Разрывы в последовательности свидетельствуют об отброшенных пакетах. Обычно они сопровождаются сообщением для каждого недостающего пакета.

Несмотря на то что протокол IP не гарантирует доставку пакетов, работоспособная сеть должна терять очень мало пакетов. Проблемы с потерянным пакетом важны для отслеживания, поскольку они, как правило, маскируются протоколами более высокого уровня. Возможно, сеть функционирует правильно, но она работает медленнее, чем должна, не только из-за повторных пакетов, но также из-за накладных расходов протокола, необходимых для их обнаружения и управления ими.

Чтобы отследить причину исчезновения пакетов, сначала запустите команду **traceroute** (см. следующий раздел), чтобы обнаружить маршрут, по которому пакеты отправляются на целевой хост. Затем выполните поочередную проверку промежуточных шлюзов, чтобы узнать, какое соединение удаляет пакеты. Чтобы выявить проблему, необходимо отправить достаточно большое количество пакетов. Ошибка, как правило, кроется в соединении между последним шлюзом, которому вы можете послать эхо-запросы без потери пакетов, и следующим за ним шлюзом.

Время, затраченное на передачу пакетов в оба конца, сообщаемое командой **ping**, может дать представление об общей скорости пути через сеть. Умеренные вариации времени в оба конца обычно не указывают на проблемы. Иногда пакеты могут задерживаться на десятки или сотни миллисекунд без видимой причины; так работает протокол IP. Вы должны увидеть довольно согласованное время прохождения в оба конца для большинства пакетов, с периодическими провалами. Многие из сегодняшних маршрутизаторов реализуют ограниченные по скорости или низкоприоритетные ответы

на ICMP-пакеты, это означает, что маршрутизатор может отложить ответ на ваш эхо-запрос, если он в текущий момент обрабатывает большой объем другого трафика.

Программа **ping** может отправлять пакеты эхо-запросов любого размера, поэтому, используя пакет, превышающий MTU сети (1500 байт для Ethernet), вы можете принудительно его фрагментировать. Эта практика помогает идентифицировать ошибки среды передачи данных или другие проблемы низкого уровня, такие как проблемы с перегруженной сетью или VPN. Размер требуемого пакета задается в байтах после флага **-s**:

```
$ ping -s 1500 cuinfo.cornell.edu
```

Обратите внимание на то, что даже такая простая команда, как **ping**, может вызвать драматические последствия. В 1998 г. так называемая атака "Ping of Death" вывела из строя большое количество систем UNIX и Windows. Эта атака была выполнена просто путем передачи слишком большого пакета эхо-запросов. Когда фрагментированный пакет был повторно собран, он заполнил буфер памяти получателя и вывел машину из строя. Хотя проблема "Ping of Death" уже давно исправлена, следует иметь в виду несколько других предостережений относительно команды **ping**.

Во-первых, только с помощью команды **ping** трудно отличить отказ сети от отказа сервера. В среде, где обычно выполняются эхо-запросы, неудачное выполнение команды **ping** просто говорит вам о том, что что-то идет не так, как надо. Также стоит отметить, что успешный эхо-запрос не дает подробной информации о состоянии целевой машины. Пакеты эхо-запросов обрабатываются в стеке протокола IP и не требуют, чтобы серверный процесс выполнялся на зондируемом хосте. Ответ гарантирует только, что машина включена и не испытала панику ядра. Для проверки доступности отдельных служб, таких как HTTP и DNS, вам понадобятся методы более высокого уровня.

Команда traceroute: трассировка IP-пакетов

Команда **traceroute**, первоначально написанная Ваном Джейкобсоном (Van Jacobson), раскрывает последовательность шлюзов, через которые IP-пакет перемещается, чтобы добраться до места назначения. Все современные операционные системы поставляются с некоторой версией команды **traceroute**.[22] Синтаксис этой команды простой:

traceroute *имя_хоста*

Большинство параметров этой команды не так важны при ежедневном использовании. Как обычно, параметр *имя_хоста* может быть указан как в виде DNS-имени или IP-адреса. Вывод — это просто список хостов, начиная с первого шлюза и заканчивая пунктом назначения. Например, применение команды **traceroute** к маршруту, проходящему от нашего хоста `jaguar` до хоста `nubark`, производит следующий вывод.

```
$ traceroute nubark
traceroute to nubark (192.168.2.10), 30 hops max, 38 byte packets
 1 lab-gw (172.16.8.254) 0.840 ms 0.693 ms 0.671 ms
 2 dmz-gw (192.168.1.254) 4.642 ms 4.582 ms 4.674 ms
 3 nubark (192.168.2.10) 7.959 ms 5.949 ms 5.908 ms
```

Анализируя этот вывод, мы можем сказать, что хост `jaguar` находится в трех переходах от `nubark`, и мы можем видеть, какие шлюзы задействованы в соединении. Также показано время прохождения в оба конца для каждого шлюза — в выводе измеряются и отображаются три отсчета для каждого перехода. Типичный маршрут между интернет-хостами часто включает в себя более 15 переходов, даже если оба сервера находятся в одном городе.

[22] Эта команда есть даже в системе Windows, но под именем **tracert** (попробуйте догадаться, почему).

Команда **traceroute** начинает работу с установки искусственно заниженного значения в поле времени жизни (TTL, или "подсчет переходов") исходящего пакета. По мере поступления пакетов на шлюз их значение TTL уменьшается. Когда шлюз уменьшает TTL до 0, он отбрасывает пакет и отправляет сообщение ICMP "time exceeded" ("время жизни") обратно на исходный хост.

🕮 Дополнительную информацию об обратном поиске DNS см в разделе 16.5.

Первый из трех пакетов **traceroute** в приведенном выше примере имеет значение TTL, равное единице. Первый шлюз, через который проходит такой пакет (в данном случае `lab-gw`), определяет, что TTL был превышен, и уведомляет хост **jaguar** об отброшенном пакете, отправив сообщение ICMP. IP-адрес отправителя в заголовке пакета ошибок идентифицирует шлюз, а команда **traceroute** выполняет поиск этого адреса в DNS, чтобы найти имя хоста, на котором находится шлюз.

Чтобы определить шлюз второго перехода, команда **traceroute** отправляет второй пакет с TTL-полем, равным двум. Первый шлюз маршрутизирует пакет и уменьшает его TTL на единицу. На втором шлюзе пакет затем отбрасывается, и сообщения об ошибках ICMP генерируются по-прежнему. Этот процесс продолжается до тех пор, пока значение TTL не будет равным количеству переходов на целевой хост, и пакеты не достигнут получателя.

Большинство маршрутизаторов отправляют свои сообщения ICMP через интерфейс, расположенный ближе всего к получателю. Если вы выполните **traceroute** в обратном порядке с целевого хоста, то сможете увидеть разные IP-адреса, используемые для идентификации одного и того же набора маршрутизаторов. Вы также можете обнаружить, что пакеты, проходящие в обратном направлении, имеют совершенно другой путь — конфигурацию, известную как асимметричная маршрутизация.

Поскольку команда **traceroute** отправляет по три пакета для каждого значения поля TTL, иногда вы можете наблюдать интересный эффект. Если промежуточный шлюз мультиплексирует трафик по нескольким маршрутам, пакеты могут быть возвращены разными хостами; в этом случае команда **traceroute** просто выводит их все.

Рассмотрим более интересный маршрут от хоста из Швейцарии до сервера `caida.org` компьютерного центра San Diego Supercomputer Center.[23]

```
linux$ traceroute caida.org
traceroute to caida.org (192.172.226.78), 30 hops max, 46 byte packets
1   gw-oetiker.init7.net (213.144.138.193) 1.122 ms 0.182 ms 0.170 ms
2   r1zur1.core.init7.net (77.109.128.209) 0.527 ms 0.204 ms 0.202 ms
3   r1fra1.core.init7.net (77.109.128.250) 18.27 ms 6.99 ms 16.59 ms
4   r1ams1.core.init7.net (77.109.128.154) 19.54 ms 21.85 ms 13.51 ms
5   r1lon1.core.init7.net (77.109.128.150) 19.16 ms 21.15 ms 24.86 ms
6   r1lax1.ce.init7.net (82.197.168.69) 158.23 ms 158.22 ms 158.27 ms
7   cenic.laap.net (198.32.146.32) 158.34 ms 158.30 ms 158.24 ms
8   dc-lax-core2-ge.cenic.net (137.164.46.119) 158.60 ms * 158.71 ms
9   dc-tus-agg1-core2-10ge.cenic.net (137.164.46.7) 159 ms 159 ms 159 ms
10  dc-sdsc2-tus-dc-ge.cenic.net (137.164.24.174) 161 ms 161 ms 161 ms
11  pinot.sdsc.edu (198.17.46.56) 161.559 ms 161.381 ms 161.439 ms
12  rommie.caida.org (192.172.226.78) 161.442 ms 161.445 ms 161.532 ms
```

Этот вывод показывает, что пакеты в течение длительного времени перемещаются внутри сети Init Seven. Иногда мы можем угадать местоположение шлюзов по их име-

[23] Мы удалили несколько долей миллисекунд из более длинных строк, чтобы они уместились на странице.

нам. Ядро Init Seven простирается от Цюриха (zur) до Франкфурта (fra), Амстердама (ам), Лондона (lon) и, наконец, Лос-Анджелеса (lax). Здесь трафик переносится в сеть cenic.net, которая доставляет пакеты хосту caida.org в сети компьютерного центра San Diego Supercomputer Center (sdsc) в Ла-Холле, Калифорния.

На переходе 8 мы видим звездочку вместо одного из показателей. Это означает, что не был получен ответ на зондирующий пакет (т.е. пакет с ошибкой). В данном случае вероятной причиной является затор, но это не единственная возможность. Команда **traceroute** использует низкоприоритетные ICMP-пакеты, которые многие маршрутизаторы распознают и отбрасывают в пользу реального трафика. Несколько звездочек не должны приводить вас в состояние паники.

Если вы видите звездочки во всех полях времени для данного шлюза, значит, из этой машины не возвращаются сообщения об истечении времени. Возможно, шлюз просто не работает. Иногда шлюз или брандмауэр настроены на безмолвное удаление пакетов с просроченными TTL. В этом случае вы все равно можете видеть, что находится за "немым" шлюзом хоста. Вполне возможно, что ошибочные пакеты шлюза возвращаются слишком медленно, поэтому команда **traceroute** перестает их ждать.

Некоторые брандмауэры полностью блокируют сообщения ICMP об истечении времени. Если такой брандмауэр расположен вдоль пути, вы не получите информацию о каком-либо из шлюзов, расположенных за ним. Тем не менее вы все равно можете определить общее количество переходов в пункт назначения, потому что зондирующие пакеты в конечном итоге проходят всегда.

Кроме того, некоторые брандмауэры могут блокировать исходящие UDP-датаграммы, отправленные командой **traceroute**, чтобы инициировать ответы ICMP. Эта проблема приводит к тому, что команда **traceroute** не сообщает какую-либо полезную информацию вообще. Если вы обнаружите, что ваш собственный брандмауэр препятствует выполнению команды **traceroute**, убедитесь, что брандмауэр пропускает трафик через UDP-порты 33434–33534, а также пакеты ICMP ECHO (тип 8).

Медленное соединение не обязательно указывает на неисправность. Некоторые физические сети имеют естественную высокую задержку. Хорошим примером являются беспроводные сети UMTS/EDGE/GPRS. Медлительность также может быть признаком высокой нагрузки на принимающую сеть. Нелогично высокое время передачи в оба конца может подтвердить такую гипотезу.

Вместо звездочки или отметки времени вы можете иногда видеть обозначение !N. Это обозначение указывает, что текущий шлюз отправил сообщение об ошибке "network unreachable" ("недоступность сети"), что означает, что он не знает, как перенаправить ваш пакет. Существуют и другие варианты: !H для ошибки "host unreachable" ("хост недоступен") и !P —"protocol unreachable" ("протокол недоступен"). Шлюз, который возвращает любое из этих сообщений об ошибках, обычно является последним переходом, до которого вы можете добраться. У этого хоста часто возникает проблема маршрутизации (возможно, вызванная вышедшим из строя сетевым соединением): либо его статические маршруты являются неправильными, либо динамические протоколы не могут определить маршрут до места назначения.

Если вам кажется, что команда **traceroute** не работает или работает медленно, это может объясняться временем ожидания при попытке определить имена хостов с помощью системы DNS. Если DNS-сервер на хосте, с которого вы выполняете команду **traceroute**, не работает, используйте команду **traceroute -n** для запроса числового вывода. Этот параметр отключает поиск имен хостов; это может быть единственный способ заставить команду **traceroute** функционировать в поврежденной сети.

Для использования команды **traceroute** необходимы привилегии пользователя root. Чтобы эта команда стала доступной для обычных пользователей, необходимо установить для нее флаг setuid для root. В некоторых дистрибутивах Linux для команды **traceroute** флаг setuid сброшен. В зависимости от вашей среды и потребностей вы можете либо установить бит setuid, либо дать заинтересованным пользователям доступ к команде через утилиту **sudo**.

В последние годы появилось несколько новых утилит, похожих на **traceroute**, которые могут обходить блокировки межсетевых экранов ICMP (см. Wiki PERTKB для обзора этих инструментов на сайте goo.gl/fXpMeu). Нам особенно нравится утилита **mtr**, которая имеет интерфейс, похожий на интерфейс команды **top**, и демонстрирует своего рода усовершенствованную команду **traceroute**. Отлично!

Решая проблемы маршрутизации, посмотрите на свою сеть с точки зрения внешнего мира. Несколько веб-служб трассировки маршрутов позволяют увидеть результат выполнения команды **traceroute** в обратном порядке прямо в окне браузера. Томас Кернен (Thomas Kernen) ведет список этих служб на сайте **traceroute.org**.

Пакетные анализаторы трафика

Утилита **tcpdump** и программа Wireshark относятся к классу инструментов, известных как пакетные анализаторы трафика (sniffers). Они прослушивают сетевой трафик и записывают или распечатывают пакеты, соответствующие критериям по вашему выбору. Например, вы можете проверить все пакеты, отправленные на конкретный хост или с него, или пакеты TCP, связанные с одним конкретным сетевым соединением.

Пакетные анализаторы трафика полезны как для решения проблем, о которых вы знаете, так и для обнаружения совершенно новых проблем. Иногда полезно обследовать сеть, чтобы убедиться, что трафик в порядке.

Пакетные анализаторы трафика должны иметь возможность перехватывать трафик, который локальная машина обычно не получает (или, по крайней мере, не обращает на него внимание), поэтому базовое сетевое оборудование должно разрешать доступ к каждому пакету. Широковещательные технологии, такие как Ethernet, работают прекрасно, как и большинство других современных локальных сетевых технологий.

🕮 Дополнительную информацию о сетевых коммутаторах см. в разделе 14.1.

Пакетные анализаторы должны видеть как можно больше необработанного сетевого трафика, но им могут мешать сетевые коммутаторы, которые в соответствии со своим предназначением пытаются ограничить распространение "ненужных" пакетов. Тем не менее работа анализатора трафика в коммутируемой сети может оказаться информативной. Вы можете обнаружить проблемы, связанные с широковещательными или многоадресатными пакетами. В зависимости от производителя коммутатора вас может удивить большой объем трафика. Даже если вы не видите сетевой трафик других систем, пакетный анализатор может быть полезен для отслеживания проблем, связанных с локальным хостом.

Помимо доступа ко всем сетевым пакетам, аппаратное обеспечение интерфейса должно передавать эти пакеты на уровень программного обеспечения. Адреса пакетов обычно проверяются на аппаратном уровне, и только пакеты широковещательной и многоадресатной передачи, адресованные локальному хосту, передаются в ядро. В режиме прослушивания ("promiscuous mode") интерфейс позволяет ядру читать все пакеты в сети, даже те, которые предназначены для других хостов.

Пакетные анализаторы трафика понимают многие форматы пакетов, используемые стандартными сетевыми службами, и могут отображать эти пакеты в удобочитаемой

форме. Эта возможность облегчает отслеживание потока обмена информацией между двумя программами. Некоторые сетевые анализаторы выводят текстовое содержимое пакета в дополнение к заголовку пакета и поэтому полезны для исследования протоколов высокого уровня.

Поскольку некоторые протоколы отправляют информацию (и даже пароли) по сети в виде открытого текста, старайтесь не вторгаться в конфиденциальность ваших пользователей. С другой стороны, ничто так ярко не свидетельствует о необходимости криптографической защиты, как обнаружение пароля в виде открытого текста, в захваченном в сетевом пакете.

Анализаторы трафика считывают данные прямо с сетевого устройства, поэтому они должны запускаться с правами `root`. Хотя это ограничение должно уменьшить вероятность того, что обычные пользователи будут прослушивать ваш сетевой трафик, на самом деле это не слишком большое препятствие. В некоторых организациях предпочитают удалять программы анализаторов трафика с большинства хостов, чтобы уменьшить вероятность злоупотреблений. Кроме того, вы должны проверить интерфейсы своих систем, чтобы убедиться, что они не работают в режиме прослушивания без вашего ведома или согласия.

Утилита `tcpdump`: пакетный анализатор трафика из командной строки

Утилита **`tcpdump`** — еще один удивительный сетевой инструмент, разработанный Ваном Джейкобсоном и работающий в большинстве систем. Утилита **`tcpdump`** уже давно является стандартным анализатором трафика, а большинство других инструментов анализа сети считывают и записывают файлы трассировки в формате **`tcpdump`**, также известном как формат **`libpcap`**.

По умолчанию утилита **`tcpdump`** настраивается на первый сетевой интерфейс, с которым она сталкивается. Если она выбирает неправильный интерфейс, вы можете заставить ее настроиться на правильный интерфейс с помощью флага **`-i`**. Если система DNS не работает или вы просто не хотите, чтобы утилита **`tcpdump`** выполняла поиск по имени, используйте параметр **`-n`**. Этот параметр важен, потому что медленно работающая служба DNS может заставить фильтр отбрасывать пакеты до того, как их можно будет обработать с помощью утилиты **`tcpdump`**.

Флаг **`-v`** увеличивает информацию о пакетах, а **`-vv`** дает вам еще больше данных. Наконец, утилита **`tcpdump`** может записывать пакеты в файлы с помощью флага **`-w`** и считывать их обратно с помощью флага **`-r`**.

Обратите внимание на то, что команда **`tcpdump -w`** сохраняет по умолчанию только заголовки пакетов. Это значение по умолчанию создает небольшие дампы, но самая полезная и релевантная информация в них может отсутствовать. Таким образом, если вы не уверены, что вам нужны только заголовки, используйте параметр **`-s`** со значением порядка 1560 (фактические значения зависят от параметра MTU) для захвата целых пакетов для последующего их анализа.

В качестве примера рассмотрим следующий усеченный вывод, поступающий от машины с именем `nubark`. Спецификации фильтра **`host bull`** ограничивают отображение пакетов теми, которые непосредственно связаны с машиной `bull`, либо в качестве отправителя, либо в качестве получателя.

```
$ sudo tcpdump host bull
12:35:23.519339 bull.41537 > nubark.domain: A? atrust.com. (28) (DF)
12:35:23.519961 nubark.domain > bull.41537: A 66.77.122.161 (112) (DF)
```

Первый пакет показывает, что машина `bull` отправила DNS-запрос на поиск IP-адреса, соответствующему имени `atrust.com`, машине `nubark`. Ответ — это IP-адрес машины, связанной с этим именем, который равен 66.77.122.161. Обратите внимание на метку времени слева и распознавание утилитой **tcpdump** протокола уровня приложений (в данном случае DNS). Номер порта на машине `bull` произволен и поэтому выводится в виде числа (41537), но поскольку номер порта DNS-сервера (53) хорошо известен, утилита **tcpdump** выводит его символическое имя с суффиксом `domain`.

Пакетные анализаторы трафика могут обеспечить огромное количество информации, которая может перегрузить не только вас, но и операционную систему. Чтобы избежать этой проблемы в загруженных сетях, утилита **tcpdump** позволяет создавать сложные фильтры. Например, следующий фильтр собирает только входящий веб-трафик из одной подсети:

```
$ sudo tcpdump src net 192.168.1.0/24 and dst port 80
```

Страница **man tcpdump** содержит несколько хороших примеров расширенной фильтрации, а также полный список примитивов.

Wireshark и TShark: улучшенные варианты tcpdump

Утилита **tcpdump** существует уже давно, но в последнее время быстро завоевывает популярность новая программа с открытым исходным кодом под названием Wireshark (ранее известная как Ethereal). Программа Wireshark находится в активной разработке и обладает большей функциональностью, чем многие коммерческие продукты для анализа трафика. Это невероятно мощный инструмент анализа, который должен быть включен в набор инструментов сетевого эксперта. Кроме того, это бесценное учебное пособие.

Программа Wireshark включает в себя графический пользовательский интерфейс **wirehark** и интерфейс командной строки **tshark**. Она доступна в качестве базового пакета для большинства операционных систем. Если она не находится в основном хранилище вашей системы, проверьте сайт `wirehark.org`, где размещен исходный код и множество предварительно скомпилированных двоичных файлов.

Программа Wireshark может читать и записывать файлы трассировки в форматах, используемых многими другими анализаторами пакетов. Еще одна удобная функция заключается в том, что вы можете щелкнуть по любому пакету в TCP-цепочке и попросить программу Wireshark собрать (объединить вместе) данные полезной нагрузки всех пакетов в потоке. Эта функция полезна, если вы хотите изучить данные, переданные во время полного сеанса обмена по протоколу TCP, например через соединение, по которому передается электронное сообщение по сети.

Фильтры захвата Wireshark функционально идентичны фильтрам утилиты **tcpdump**, поскольку Wireshark использует ту же базовую библиотеку **libpcap**. Однако обратите внимание на одну важную проблему, связанную с программой Wireshark — добавленную функцию "фильтров отображения", которые влияют на то, что вы видите, а не на то, что действительно захватывает анализатор пакетов. Синтаксис фильтра отображения более мощный, чем синтаксис **libpcap**, поддерживаемый во время захвата. Фильтры отображения выглядят несколько похожими, но они не совпадают.

Программа Wireshark имеет встроенные модули разбора (dissectors) для широкого спектра сетевых протоколов, включая многие используемые для реализации сетевых хранилищ SAN. Она преобразует пакеты в структурированное дерево информации, в котором каждый бит пакета описывается обычным английским языком.

🕮 Дополнительную информацию о хранилищах SAN см. в разделе 21.1.

Следует сделать одно замечание относительно программы Wireshark: хотя у нее много удобных функций, на протяжении многих лет ей также требовалось много обновлений для системы безопасности. Запустите текущую копию и не оставляйте ее на неопределенный срок на машинах, содержащих конфиденциальную информацию; они могут стать потенциальным маршрутом для атаки.

🕮 Дополнительную информацию о программе Wireshark можно почерпнуть из третьего издания книги Криса Сандерса *Анализ пакетов: практическое руководство по использованию Wireshark и tcpdump для решения реальных проблем в локальных сетях* (пер. с англ., изд. "Диалектика", 2019 г.).

13.13. Мониторинг сети

В главе 28 описывается несколько универсальных платформ, которые могут помочь структурировать постоянный контроль над вашими системами и сетями. Эти системы принимают данные из различных источников, суммируют их таким образом, чтобы освещать текущие тенденции и предупреждают администраторов о проблемах, требующих немедленного внимания.

Сеть является ключевым компонентом любой вычислительной среды, поэтому часто это одна из первых частей инфраструктуры, которая подвергается систематическому мониторингу. Если вы не чувствуете себя готовым присоединиться к единой платформе мониторинга для всех ваших административных потребностей, используйте пакеты, описанные в этом разделе, для мелкомасштабного мониторинга, ориентированного на сеть.

Программа SmokePing: постепенный сбор статистики об эхо-запросах

Даже вполне благополучные сети иногда теряют пакеты. С другой стороны, сети не должны терять пакеты регулярно, даже с низкой скоростью, потому что воздействие на пользователей может быть непропорционально серьезным. Поскольку протоколы высокого уровня часто продолжают работать даже при наличии потери пакетов, вы никогда не заметите удаленные пакеты, если не будете активно контролировать их.

SmokePing, инструмент с открытым исходным кодом от Тобиаса Ойтикера (Tobias Oetiker), поможет вам разработать более полную картину поведения ваших сетей. SmokePing отправляет несколько пакетов эхо-запросов на целевой хост через равные промежутки времени. Он показывает историю каждой контролируемой связи через веб-интерфейс и может отправлять сигналы тревоги, когда что-то идет не так. Вы можете получить копию этой программы по адресу `oss.oetiker.ch/smokeping`.

На рис. 13.4 показан график SmokePing. Вертикальная ось представляет собой время прохождения эхо-запросов, а горизонтальная ось — время (недели). Черная линия, из которой торчат серые всплески, указывает среднюю продолжительность прохождения эхо-запросов в оба конца. Сами пики — это время прохождения отдельных пакетов. Поскольку серые линии в этом графе расположены только над срединной линией, подавляющее большинство пакетов должно перемещаться со скоростью, близкой к средней, причем лишь несколько из них задерживаются. Это типичная ситуация.

Ступенчатый характер срединной линии указывает на то, что среднее время прохождения пакетов до этого пункта назначения в течение периода мониторинга менялось несколько раз. Наиболее вероятные гипотезы, объясняющие это наблюдение, заключа-

ются в том, что доступ к хосту осуществляется по нескольким путям или что он фактически представляет собой набор из нескольких хостов, имеющих одно и то же имя DNS, но несколько IP-адресов.

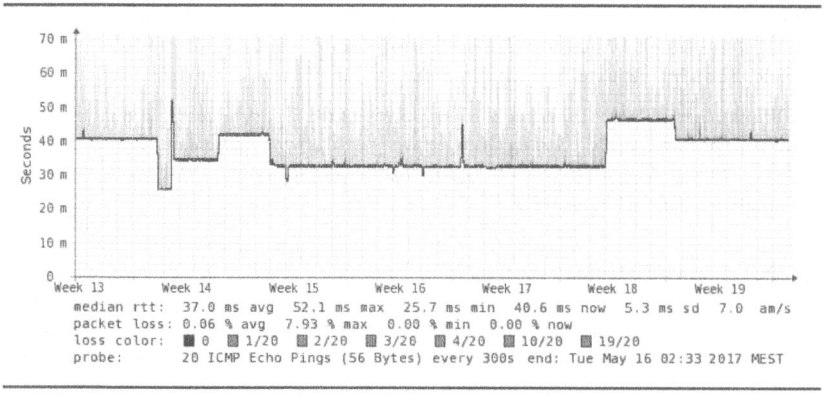


Рис. 13.4. Пример графика SmokePing

Программа iPerf: отслеживание производительности сети

Инструменты на основе эхо-запросов полезны для проверки доступности, но недостаточно эффективны для анализа и отслеживания производительности сети. Для решения этой задачи предназначена программа iPerf. В последней версии iPerf3 имеется широкий набор функций, которые администраторы могут использовать для точной настройки сетевых параметров, чтобы достичь максимальной производительности.

Рассмотрим мониторинг пропускной способности с помощью программы iPerf. Фактически программа iPerf всего лишь открывает соединение (TCP или UDP) между двумя серверами, передает данные между ними и записывает, сколько времени занял этот процесс.

После установки программы **iperf** на обеих машинах запустите сервер.

```
$ iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
```

Затем с машины, которую вы хотите протестировать, передайте некоторые данные, как показано ниже.

```
$ iperf -c 10.211.55.11
------------------------------------------------------------
Client connecting to 10.211.55.11, TCP port 5001
TCP window size: 22.5 KByte (default)
------------------------------------------------------------
[ 3] local 10.211.55.10 port 53862 connected with 10.211.55.11 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 3] 0.0-10.0 sec  4.13 GBytes  3.55 Gbits/sec
```

Программа iPerf возвращает большие объемы потоковых данных для отслеживания полосы пропускания. Это особенно полезно для оценки влияния изменений параметров ядра, которые управляют сетевым стеком, таких как изменения в максимальном блоке передачи (MTU); более подробную информацию см. в разделе 13.2.

Программа Cacti: сбор и отображение данных

Программа Cacti, доступная на сайте `cacti.net`, предлагает несколько привлекательных функций. Она использует отдельный пакет, RRDtool, в качестве вспомогательного средства, в котором хранятся данные мониторинга в виде баз данных статического размера, не требующих технического обслуживания.

Кактусы хранят столько данных, сколько необходимо для создания графиков, которые вы хотите построить. Например, программа Cacti может сохранять одну выборку каждую минуту в течение дня, одну выборку каждый час в неделю и одну выборку каждую неделю в течение года. Эта схема консолидации позволяет поддерживать важный исторический контекст без необходимости хранить неважные данные или тратить время на администрирование базы данных.

Программа Cacti может записывать и отображать любую SNMP-переменную (см. раздел 28.9), а также многие другие показатели производительности. Вы можете собирать любые данные, которые захотите. В сочетании с агентом NET-SNMP программа Cacti генерирует историческую перспективу практически для любого системного или сетевого ресурса.

На рис. 13.5 приведены примеры графиков, созданных программой Cacti. Эти графики показывают среднее значение нагрузки на устройстве в течение нескольких недель вместе с дневным трафиком на сетевом интерфейсе.

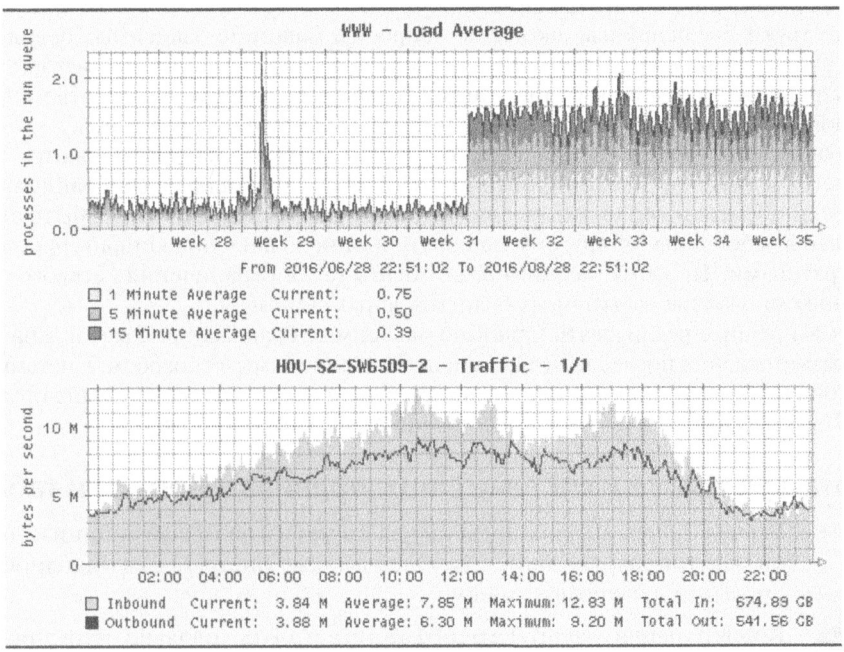


Рис. 13.5. Пример графика Cacti

Программа Cacti легко управляется с помощью веб-интерфейса, а также предоставляет другие преимущества встроенного инструмента RRDtool, такие как низкая стоимость обслуживания и красивое графическое представление. Ссылки на текущие версии программ RRDtool и Cacti, а также десятки других инструментов мониторинга можно найти на домашней странице RRDtool на сайте `rrdtool.org`.

13.14. Брандмауэры и система NAT

Мы *не* рекомендуем использовать системы Linux, UNIX или Windows в качестве брандмауэров из-за отсутствия безопасности, присущей полноценным операционным системам общего назначения.[24] Однако все операционные системы имеют функции брандмауэра и могут стать вполне работоспособной заменой для организаций, у которых нет денег для покупки дорогого брандмауэра. Аналогично брандмауэр Linux или UNIX является прекрасным вариантом для домашнего пользователя, обладающего знаниями в области безопасности и предпочитающего самостоятельно латать ее дыры.

Если вы используете компьютер общего назначения в качестве брандмауэра, убедитесь, что на нем установлены параметры безопасности и соответствующие заплатки. Брандмауэр — отличное место для реализации всех рекомендаций, изложенных в главе 27. (Брандмауэры с фильтрацией пакетов обсуждаются в разделе 27.7. Если вы не знакомы с базовой концепцией брандмауэра, вероятно, было бы разумно прочитать этот раздел, прежде чем продолжить чтение.)

Компании Microsoft в значительной степени удалось убедить мир в том, что каждый компьютер нуждается в собственном встроенном брандмауэре. Однако это не так. Фактически машинно-зависимые брандмауэры могут привести к некорректной работе систем и появлению таинственных сетевых проблем, если они не управляются в соответствии с общепринятыми стандартами.

Существуют две основные школы по вопросу о машинно-зависимых брандмауэрах. Первая школа считает их излишними. Согласно этой точке зрения брандмауэры принадлежат маршрутизаторам шлюза, где они могут защитить всю сеть посредством применения одного последовательного (и постоянно применяемого) набора правил.

Вторая школа считает, что машинно-зависимые брандмауэры являются важным компонентом глубоко эшелонированной обороны. Хотя шлюзовые брандмауэры теоретически достаточны для управления сетевым трафиком, они могут быть скомпрометированы, обойдены по другому маршруту или неправильно сконфигурированы администраторами. Поэтому разумно реализовать те же ограничения сетевого трафика с помощью множества избыточных брандмауэрных систем.

Если вы решите реализовать машинно-зависимые брандмауэры, вам нужна система для их развертывания последовательным и легко обновляемым способом. Системы управления конфигурацией, описанные в главе 23, являются прекрасными кандидатами на эту задачу. Не полагайтесь на ручную конфигурацию; она слишком уязвима.

Утилита `iptables` в системе Linux: правила, цепочки и таблицы



Версия 2.4 ядра Linux представила совершенно новый механизм обработки пакетов под названием Netfilter, а также инструмент командной строки `iptables` для управления им.[25]



Конфигурация утилиты **`iptables`** может быть довольно затруднительной. Системы Debian и Ubuntu имеют простой интерфейс **`ufw`**, который упрощает общие операции и настройку конфигурации. Стоит проверить, насколько далеки ваши потребности от типичных.

[24]Тем не менее многие бюджетные потребительские сетевые устройства, такие как маршрутизаторы Linksys, используют Linux и **`iptables`** в своем ядре.

[25]Еще более новая система, **`nftables`**, была доступна в версии ядра 3.13 с 2014 г. Это разработка системы Netfilter, которая конфигурируется командой **`nft`**, а не командой **`iptables`**. Мы не обсуждаем ее в этой книге, но ее стоит внедрить в организациях, использующих новые ядра.

Утилита **iptables** применяет к сетевым пакетам упорядоченные цепочки правил. Наборы цепочек образуют таблицы и используются для обработки определенных видов трафика.

Например, таблица **iptables** по умолчанию называется *фильтром*. Цепочки правил в этой таблице используются для пакетной фильтрации сетевого трафика. Таблица фильтров содержит три стандартные цепочки: FORWARD, INPUT и OUTPUT. Каждый пакет, обработанный ядром, проходит через одну из этих цепочек.

Правила в цепочке FORWARD применяются ко всем пакетам, которые поступают на один сетевой интерфейс и должны быть перенаправлены на другой. Правила в цепочках INPUT и OUTPUT применяются к трафику, адресованному локальному хосту или исходящему из него соответственно.

Как правило, эти три стандартные цепочки, — все, что вам нужно для создания межсетевого экрана между двумя сетевыми интерфейсами. При необходимости вы можете определить настраиваемую конфигурацию для поддержки более сложных сценариев учета или маршрутизации.

В дополнение к таблице фильтров **iptables** включает таблицы nat и mangle. В таблице nat содержатся цепочки правил, которые управляют преобразованием сетевых адресов (здесь nat — это имя таблицы **iptables**, а NAT — это название общей схемы перевода адресов). Система NAT обсуждается в разделе 13.4, а пример таблицы nat в действии показан ниже. Позже в этом разделе мы используем цепочку PREROUTING сети nat для фильтрации пакетов, препятствующей анализу трафика.

Таблица mangle содержит цепочки, которые модифицируют или заменяют содержимое сетевых пакетов вне контекста NAT и фильтрации пакетов. Хотя таблица mangle удобна для специальной обработки пакетов, например для сброса значений времени ожидания IP, она обычно не используется в большинстве производственных сред. Мы обсуждаем только таблицы фильтров и nat в этом разделе, оставляя таблицу mangle вне поля зрения.

Цели правил iptables

Каждое правило, составляющее цепочку, имеет раздел target, который определяет, что делать с соответствующими пакетами. Если пакет соответствует правилу, в большинстве случаев он сразу обрабатывается; никакие дополнительные правила не проверяются. Хотя многие цели определены внутри программы **iptables**, существует возможность задать другую цель цепочки.

Целями, доступными для правил в таблице фильтров, являются ACCEPT, DROP, REJECT, LOG, ULOG, REDIRECT, RETURN, MIRROR и QUEUE. Если правило приводит к цели ACCEPT, то соответствующим ему пакетам разрешается продолжать свой путь. Цели DROP и REJECT отбрасывают свои пакеты; цель DROP не выдает никаких сообщений, а цель REJECT возвращает сообщение об ошибке ICMP. Цель LOG предоставляет простой способ отслеживания пакетов, если они соответствуют правилам, а ULOG расширяет протоколирование.

Цель REDIRECT отправляет пакеты на прокси-сервер вместо того, чтобы позволить им идти своим путем. Например, вы можете использовать эту функцию, чтобы заставить весь веб-трафик вашего сайта проходить через кеш-сервер, например Squid. Цель RETURN завершает определяемые пользователем цепочки и действует аналогично оператору return в вызове подпрограммы. Цель MIRROR меняет местами IP-адрес отправителя и адреса получателя перед отправкой пакета. Наконец, цель QUEUE передает пакеты локальным пользовательским программам через модуль ядра.

Настройка брандмауэра iptables

Прежде чем вы сможете использовать программу **iptables** в качестве брандмауэра, необходимо включить переадресацию IP и убедиться, что в ядро загружены различные модули **iptables**. Дополнительные сведения о включении IP-переадресации см. в разделе 13.10. Пакеты, в которых устанавливается программа **iptables**, обычно включают сценарии запуска для обеспечения ее подключения и загрузки.

Брандмауэр Linux обычно реализуется как серия команд **iptables**, содержащихся в сценарии запуска **rc**. Команды **iptables** обычно принимают одну из следующих форм:

```
iptables -F имя-цепочки
iptables -P имя-цепочки цель
iptables -A имя-цепочки -i интерфейс -j цель
```

Первая форма (**-F**) сбрасывает все предыдущие правила из цепочки. Вторая форма (**-P**) устанавливает политику по умолчанию (т.е. цель) для цепочки. Мы рекомендуем использовать DROP для целевой цепочки по умолчанию. Третья форма (**-A**) добавляет текущую спецификацию в цепочку. Если вы не укажете таблицу с аргументом **-t**, ваши команды применяются к цепочкам в таблице фильтров. Параметр **-i** применяет правило к именованному интерфейсу, а **-j** идентифицирует цель. Программа **iptables** принимает множество других параметров, часть из которых приведена в табл. 13.10.

Таблица 13.10. Флаги командной строки для фильтров iptables

Параметр	Описание или возможные значения
-p *протокол*	Ограничение протокола: **tcp**, **udp** или **icmp**
-s *ip-отправителя*	Ограничение отправителя по имени или адресу (подходит система обозначений CIDR)
-d *ip-получателя*	Ограничение получателя по имени или адресу
--sport *порт#*	Ограничение порта отправителя (обратите внимание на двойное тире)
--dport *порт#*	Ограничение порта назначения (обратите внимание на двойное тире)
--icmp-type *тип*	Ограничение типа ICMP-пакета (обратите внимание на двойное тире)
!	Отмена параметра
-t *таблица*	Указание таблицы, к которой применяется команда (по умолчанию — фильтр)

Полный пример

Ниже мы разберем полный пример. Мы предполагаем, что интерфейс eth1 имеет выход в Интернет и что интерфейс eth0 имеет выход во внутреннюю сеть. IP-адрес eth1 — 128.138.101.4, IP-адрес eth0 — 10.1.1.1, причем оба интерфейса имеют сетевую маску 255.255.255.0. В этом примере используется фильтрация пакетов без сохранения состояния для защиты веб-сервера с IP-адресом 10.1.1.2, что является стандартным методом защиты интернет-серверов. Позже в этом примере мы покажем, как использовать фильтрацию с сохранением состояния для защиты пользователей настольных компьютеров.

Наш первый набор правил инициализирует таблицу фильтров. Во-первых, все цепочки в таблице пусты, а для цели цепочек INPUT и FORWARD по умолчанию установлено значение DROP. Как и в случае с любым другим сетевым брандмауэром, наиболее безопасной стратегией является удаление любых пакетов, которые вы явно не разрешали.

```
iptables -F
iptables -P INPUT DROP
iptables -P FORWARD DROP
```

Поскольку правила оцениваются по порядку, мы ставим наши самые загружаемые правила впереди.[26] Первое правило разрешает все подключения через брандмауэр, которые происходят из доверенной сети. Следующие три правила в цепочке FORWARD позволяют подключение через брандмауэр к сетевым службам 10.1.1.2. В частности, мы разрешаем подключения SSH (порт 22), HTTP (порт 80) и HTTPS (порт 443) к нашему веб-серверу.

```
iptables -A FORWARD -i eth0 -p ANY -j ACCEPT
iptables -A FORWARD -d 10.1.1.2 -p tcp --dport 22 -j ACCEPT
iptables -A FORWARD -d 10.1.1.2 -p tcp --dport 80 -j ACCEPT
iptables -A FORWARD -d 10.1.1.2 -p tcp --dport 443 -j ACCEPT
```

Единственным TCP-трафиком, который мы разрешаем к нашему хосту брандмауэра (10.1.1.1), является SSH, что полезно для управления самим брандмауэром. Второе правило, указанное ниже, позволяет осуществлять циклический трафик, который остается локальным для хоста. Администраторы нервничают, когда не могут выполнить эхо-запрос по маршруту, заданному по умолчанию, поэтому третье правило разрешает пакеты ICMP ECHO_REQUEST с внутренних IP-адресов.

```
iptables -A INPUT -i eth0 -d 10.1.1.1 -p tcp --dport 22 -j ACCEPT
iptables -A INPUT -i lo -d 127.0.0.1 -p ANY -j ACCEPT
iptables -A INPUT -i eth0 -d 10.1.1.1 -p icmp --icmp-type 8 -j ACCEPT
```

Чтобы любой IP-хост в Интернете работал правильно, необходимо разрешить передачу через брандмауэр ICMP-пакетов определенных типов. Следующие восемь правил допускают передачу минимального набора ICMP-пакетов для хоста брандмауэра, а также для сети, расположенной за ним.

```
iptables -A INPUT -p icmp --icmp-type 0 -j ACCEPT
iptables -A INPUT -p icmp --icmp-type 3 -j ACCEPT
iptables -A INPUT -p icmp --icmp-type 5 -j ACCEPT
iptables -A INPUT -p icmp --icmp-type 11 -j ACCEPT
iptables -A FORWARD -d 10.1.1.2 -p icmp --icmp-type 0 -j ACCEPT
iptables -A FORWARD -d 10.1.1.2 -p icmp --icmp-type 3 -j ACCEPT
iptables -A FORWARD -d 10.1.1.2 -p icmp --icmp-type 5 -j ACCEPT
iptables -A FORWARD -d 10.1.1.2 -p icmp --icmp-type 11 -j ACCEPT
```

🕮 Дополнительную информацию об анализе интернет-трафика см. в разделе 13.8.

Затем добавляем правила в цепочку PREROUTING в таблице nat. Хотя таблица nat не предназначена для фильтрации пакетов, ее цепочка PREROUTING особенно полезна для фильтрации, предотвращающей анализа трафика. Если мы помещаем записи DROP в цепочку PREROUTING, они не должны присутствовать в цепочках INPUT и FORWARD, так как цепочка PREROUTING применяется ко всем пакетам, которые входят в хост брандмауэра. Лучше помещать записи в одном месте, а не дублировать их.

```
iptables -t nat -A PREROUTING -i eth1 -s 10.0.0.0/8 -j DROP
iptables -t nat -A PREROUTING -i eth1 -s 172.16.0.0/12 -j DROP
iptables -t nat -A PREROUTING -i eth1 -s 192.168.0.0/16 -j DROP
iptables -t nat -A PREROUTING -i eth1 -s 127.0.0.0/8 -j DROP
iptables -t nat -A PREROUTING -i eth1 -s 224.0.0.0/4 -j DROP
```

Наконец, мы завершаем как цепочки INPUT, так и FORWARD с правилом, запрещающим пакеты, которые не были разрешены явным образом. Хотя мы уже применяли это поведение с помощью команд **iptables -P**, цель LOG позволяет нам видеть, кто обращается к нам из Интернета.

[26] Однако вы должны быть осторожными, чтобы изменение порядка правил для повышения производительности не влияло на функциональность.

```
iptables -A INPUT -i eth1 -j LOG
iptables -A FORWARD -i eth1 -j LOG
```

По желанию мы могли бы настроить IP NAT для маскировки частного адресного пространства, используемого во внутренней сети. Для получения дополнительной информации об NAT см. раздел 13.4.

Одной из самых мощных функций, которой межсетевой экран Netfilter обеспечивает брандмауэр Linux, является фильтрация пакетов с учетом состояния. Вместо того чтобы разрешать определенные входящие службы, брандмауэр для клиентов, подключающихся к Интернету, должен разрешать входящие ответы на запросы клиентов.

Простая цепочка состояния FORWARD ниже позволяет всему трафику покидать нашу сеть, но разрешает только входящий трафик, связанный с подключениями, инициированными нашими хостами.

```
iptables -A FORWARD -i eth0 -p ANY -j ACCEPT
iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
```

Чтобы позволить программе **iptables** отслеживать сложные сетевые сеансы, такие как FTP и IRC, необходимо загрузить некоторые модули ядра. Если эти модули не загружены, программа **iptables** просто запрещает эти соединения. Несмотря на то что фильтры с установленными состояниями могут повысить безопасность вашего сайта, они также увеличивают сложность сети и могут снизить производительность. Перед реализацией в брандмауэре убедитесь, что вам нужна функциональность с сохранением состояния.

Возможно, лучший способ отладки ваших наборов правил **iptables** — использовать команду **iptables -L -v**. Эти параметры позволяют получить информацию о том, сколько раз каждое правило в ваших цепочках соответствовало пакету. Мы часто добавляем временные правила **iptables** с целью LOG, когда нам нужна дополнительная информация о пакетах, которые совпадают. Более сложные проблемы часто можно решать, используя пакетный анализатор трафика, такой как **tcpdump**.

Linux NAT и фильтрация пакетов

Система Linux традиционно реализует только ограниченную форму преобразования сетевых адресов (NAT), которая более правильно называется Port Address Translation (PAT). Вместо использования диапазона IP-адресов в качестве истинной реализации NAT PAT мультиплексирует все соединения на один адрес. Детали и различия не имеют большого практического значения.

Программа **iptables** реализует технологию NAT, а также фильтрацию пакетов. В более ранних версиях Linux эта функциональность была немного беспорядочной, но программа **iptables** проводит намного более четкое разделение между технологией NAT и фильтрацией. Конечно, если вы используете NAT, чтобы локальные хосты могли пользоваться Интернетом, вы должны также использовать полный набор фильтров брандмауэра.

Чтобы заставить технолгию NAT работать, включите переадресацию IP в ядре, установив переменную ядра **/proc/sys/net/ipv4/ip_forward** равной единице. Кроме того, вставьте соответствующие модули ядра:

```
$ sudo modprobe iptable_nat
$ sudo modprobe ip_conntrack
$ sudo modprobe ip_conntrack_ftp
```

Многие другие модули отслеживают соединения; см. подкаталог **net/netfilter** в каталоге **/lib/modules** для получения более полного списка и включения тех, которые вам нужны.

Команда **iptables** для маршрутизации пакетов с использованием NAT имеет форму

```
iptables -t nat -A POSTROUTING -o eth0 -j SNAT --to 128.138.101.4
```

Этот пример относится к тому же хосту, что и пример фильтрации в предыдущем разделе, поэтому `eth1` — это интерфейс, подключенный к Интернету. Интерфейс `eth1` не отображается непосредственно в командной строке выше, но его IP-адрес является тем, который появляется в качестве аргумента для `-to`. Интерфейс `eth0` — это тот интерфейс, который подключен к внутренней сети.

Для интернет-хостов, кажется, что все пакеты от хостов во внутренней сети имеют IP-адрес `eth1`. Хост, который реализует NAT, принимает входящие пакеты, просматривает их истинные адресаты, переписывает их с соответствующим внутренним IP-адресом сети и отправляет по маршруту.

IPFilter для UNIX-систем



IPFilter, пакет с открытым исходным кодом, разработанный Дарреном Ридом (Darren Reed), предоставляет технологию NAT и услуги брандмауэра с поддержкой состояния в различных системах, включая Linux и FreeBSD. Вы можете использовать пакет IPFilter в качестве загружаемого модуля ядра (который рекомендуется разработчиками) или статически включить его в ядро.

Пакет IPFilter является зрелым и полнофункциональным. Он имеет активное сообщество пользователей и историю непрерывного развития. Он способен отслеживать состояние даже для протоколов без учета состояния, таких как UDP и ICMP.

Пакет IPFilter считывает правила фильтрации из файла конфигурации (обычно **/etc/ipf/ipf.conf** или **/etc/ipf.conf**), вместо того, чтобы предлагать вам выполнять серию команд так же, как и **iptables**.

Примером простого правила, которое может отображаться в файле **ipf.conf**, является

```
block in all
```

Это правило блокирует весь входящий трафик (т.е. сетевую активность, получаемую системой) на всех сетевых интерфейсах. Это, конечно, безопасно, но не особенно полезно!

В табл. 13.11 показаны некоторые из возможных условий, которые могут появиться в правиле **ipf**.

Таблица 13.11. Типичные условия, используемые пакетом Ipf

Условие	Описание или возможные значения
`on` *интерфейс*	Применяет правило к указанному интерфейсу
`proto` *протокол*	Выбирает пакет в соответствии с протоколом: `tcp`, `udp` или `icmp`
`from` *ip-отправителя*	Фильтры по отправителю: хост, сеть или `any`
`to` *ip-получателя*	Фильтры по получателю: хост, сеть или `any`
`port = порт #`	Фильтры по имени порта (из файла **/etc/services**) или номеру[a]
`flags` *флаг-спец*	Фильтры в соответствии с битами флагов TCP-заголовка
`icmp-type` *число*	Фильтры по типу и коду ICMP
`keep state`	Сохраняет сведения о потоке сеанса; см. ниже

[a] Можно использовать любой оператор сравнения: =, <, >, <=, > = и т.д.

Пакет IPFilter оценивает правила в порядке, в котором они представлены в файле конфигурации. При этом срабатывает *самое последнее* совпадающее правило. Например, входящие пакеты, проходящие по следующему фильтру, всегда будут проходить:

```
block in all
pass in all
```

Правило `block` относится ко всем пакетам, как и правило `pass`, но правило `pass` — это последнее совпадающее правило. Чтобы принудительно применить правило соответствия и заставить пакет IPFilter пропустить последующие правила, используйте ключевое слово `quick`:

```
block in quick all
pass in all
```

Промышленный брандмауэр обычно содержит множество правил, поэтому для поддержания его высокой производительности важно использовать ключевое слово `quick`. Без этого каждый пакет оценивается по каждому правилу, и эта расточительность является слишком дорогостоящей.

Возможно, наиболее распространенным использованием брандмауэра является контроль доступа к определенной сети или хосту и от него, часто по отношению к определенному порту. Пакет IPFilter имеет мощный синтаксис для управления трафиком на этом уровне детализации. В следующих правилах входящий трафик разрешен для сети 10.0.0.0/24 на портах TCP 80 и 443 и на UDP-порту 53.

```
block out quick all
pass in quick proto tcp from any to 10.0.0.0/24 port = 80 keep state
pass in quick proto tcp from any to 10.0.0.0/24 port = 443 keep state
pass in quick proto udp from any to 10.0.0.0/24 port = 53 keep state
block in all
```

Ключевые слова `keep state` заслуживают особого внимания. Программа IPFilter может отслеживать соединения, отмечая первый пакет новых сеансов. Например, когда новый пакет поступает на порт 80 хоста 10.0.0.10, программа IPFilter делает запись в таблице состояний и пропускает пакет. Эти ключевые слова позволяют также веб-серверу отправить ответ, даже если первое правило явно блокирует весь исходящий трафик.

Ключевые слова `keep state` также полезны для устройств, которые не предлагают никаких услуг, но которые должны инициировать соединения. Следующий набор правил разрешает все обмены информацией, инициированные хостом 192.168.10.10. Он блокирует все входящие пакеты, кроме тех, которые связаны с уже установленными соединениями.

```
block in quick all
pass out quick from 192.168.10.10/32 to any keep state
```

Ключевые слова `keep state` работают также для UDP- и ICMP-пакетов, но поскольку эти протоколы не имеют состояния, их работа немного сложнее: IPFilter разрешает ответы на UDP- или ICMP-пакет в течение 60 с после того, как входящий пакет просматривается фильтром. Например, если UDP-пакет поступил от хоста `10.0.0.10` с порта `32000` и адресован хосту `192.168.10.10` на порт `53`, то ответ UDP от хоста `192.168.10.10` будет разрешен в течение 60 с. Аналогично эхо-ответ ICMP (ping-ответ) разрешен после ввода эхо-запроса в таблицу состояний.

🕮 Дополнительную информацию о технологии NAT см. в разделе 13.4.

Для предоставления услуг NAT программа IPFilter использует ключевое слово `map` (вместо `pass` и `block`). В следующем правиле трафик из сети `10.0.0.0/24` отображается на текущий маршрутизируемый адрес интерфейса `em0`.

```
map em0 10.0.0.0/24 -> 0/32
```

Если адрес `em0` изменится, то фильтр должен быть перезагружен. Это может случиться, если `em0` получает IP-адрес динамически через DHCP. По этой причине NAT-функции IPFilter лучше всего использовать на серверах со статическим IP-адресом в интерфейсе, обращенном к Интернету.

В табл. 13.12 перечислены инструменты командной строки, которые поставляются с пакетом IPFilter.

Таблица 13.12. Команды IPFilter

Команда	Функция
ipf	Управляет правилами и списками фильтров
ipfstat	Получает статистику фильтрации пакетов
ipmon	Отслеживает зарегистрированную информацию о фильтрах
ipnat	Управляет правилами NAT

Из команд, приведенных в табл. 13.12, наиболее часто используется **ipf**. Эта команда принимает файл правил в качестве входных данных и добавляет правильно разобранные правила в список фильтров ядра. Если вы не используете аргумент **-Fa**, который сбрасывает все существующие правила, команда **ipf** добавляет правила в конец фильтра. Например, чтобы очистить существующий набор фильтров ядра и загрузить правила из файла **ipf.conf**, используйте следующий синтаксис:

```
$ sudo ipf -Fa -f /etc/ipf/ipf.conf
```

Программа IPFilter использует файлы псевдоустройств из каталога **/dev** для управления доступом, и по умолчанию только пользователь `root` может редактировать список фильтров. Мы рекомендуем оставить разрешения по умолчанию на месте и использовать для поддержки фильтра программу **sudo**.

Используйте флаг **-v** команды **ipf** при загрузке файла правил для выявления синтаксических ошибок и других проблем в конфигурации.

13.15. Облачные сети

Одна из интересных особенностей облака заключается в том, что вы можете определить сетевую среду, в которой действуют ваши виртуальные серверы. Разумеется, в конечном счете облачные серверы работают на физических компьютерах, которые подключены к реальному сетевому оборудованию. Однако это не обязательно означает, что виртуальные серверы, работающие на одном и том же хосте, объединены в сеть. Комбинация технологии виртуализации и программируемого сетевого коммутационного оборудования дает платформенным поставщикам большую гибкость для определения сетевой модели, которую они экспортируют клиентам.

Виртуальное частное облако AWS (VPC)

VPC, программная сетевая технология для Amazon Web Services, создает частные сети в более широкой сети AWS. Технология VPC впервые была представлена в 2009 г.

как мост между локальным центром обработки данных и облаком, открывая множество гибридных вариантов использования для корпоративных организаций. Сегодня VPC является центральной особенностью AWS и по умолчанию включена для всех учетных записей. Экземпляры EC2 для новых учетных записей AWS должны быть созданы в соответствии с технологией VPC, а большинство новых служб AWS запускаются с поддержкой родной VPC.[27]

Перечислим основные функциональные особенности VPC.

- Диапазон адресов IPv4, выбранный из частного адресного пространства RFC1918, выраженный в нотации CIDR (например, 10.110.0.0/16 для адресов 10.110.0.0-10.110.255.255).[28]
- Сегментация адресного пространства VPC на более мелкие подсети.
- Таблицы маршрутизации, определяющие, куда отправлять трафик.
- Группы безопасности, которые действуют как брандмауэры для экземпляров EC2.
- Списки контроля доступа к сети (Network Access Control List — NACL) для изоляции подсетей друг от друга.

Вы можете создать столько частных сетей VPC, сколько вам нужно, и ни один другой пользователь AWS не имеет доступа к их сетевому трафику.[29] Сети VPC в одном регионе могут взаимодействовать, создавая частные маршруты между отдельными сетями. Сети VPC в разных регионах могут быть связаны с программными VPN-туннелями через Интернет или с дорогостоящими, настраиваемыми, прямыми подключениями к центрам обработки данных AWS через частные сети, которые вы должны арендовать у телекоммуникационной компании.

Сети VPC могут быть такими же маленькими, как сеть /28, или большими как /16. Это очень важно планировать заранее, потому что после создания сети VPC ее размер не может быть скорректирован. Выберите адресное пространство, которое достаточно велико для обеспечения будущего роста, но также убедитесь, что оно не конфликтует с другими сетями, которые вы можете подключить.

Подсети и таблицы маршрутизации

🕮 Дополнительную информацию о сегменте DMZ см. в разделе 27.8.

Как и традиционные сети, сети VPC разделены на подсети. Публичные подсети предназначены для серверов, которые должны напрямую общаться с клиентами в Интернете. Они сродни традиционным сегментам DMZ. Частные подсети недоступны из Интернета и предназначены для надежных или конфиденциальных систем.

Маршрутизация VPC проще, чем маршрутизация традиционной аппаратной сети, поскольку облако не моделирует физическую топологию. Каждый получатель достигается за один логический переход.

В мире физических сетей каждое устройство имеет таблицу маршрутизации, которая сообщает ему, как маршрутизировать исходящие сетевые пакеты. Но в сети VPC таблицы маршрутизации также являются абстрактным объектом, который определяется через веб-консоль AWS или ее эквивалент командной строки. Каждая подсеть VPC имеет свя-

[27] Долгое время пользователи жаловались на то, что службы AWS являются неполными, если они не поддерживают технологию VPC.

[28] Недавно в технологию VPC также добавили поддержку протокола IPv6

[29] В зависимости от состояния вашей учетной записи служба AWS может сначала ограничить вас пятью сетями VPC. Однако, если вам это нужно, вы можете запросить более высокий лимит.

занную таблицу маршрутизации VPC. Когда экземпляры создаются в подсети, их таблицы маршрутизации инициализируются из шаблона VPC.

Простейшая таблица маршрутизации содержит только статический стандартный маршрут для доступа к другим экземплярам в пределах одного и того же VPC. Вы можете добавить дополнительные маршруты для доступа в Интернет, локальные сети (через VPN-соединения) или другие VPC (через пиринговые подключения).

Компонент, называемый *интернет-шлюзом*, соединяет сеть VPC с Интернетом. Этот объект прозрачен для администратора и управляется службой AWS. Однако, если экземпляры должны иметь подключение к Интернету, необходимо создать этот объект самостоятельно и добавить его к сети VPC. Хосты в публичных подсетях могут напрямую обращаться к интернет-шлюзу.

Экземпляры в частных подсетях недоступны из Интернета, даже если им назначены публичные IP-адреса, что приводит к большой путанице для новых пользователей.

Для исходящего доступа они должны переходить через NAT-шлюз в общую подсеть. Технология VPC предлагает управляемую функцию NAT, которая экономит ваши накладные расходы на запуск собственного шлюза, но это требует дополнительных почасовых затрат. Шлюз NAT является потенциальным узким местом для приложений с высокими требованиями к пропускной способности, поэтому лучше найти серверы для таких приложений в общественных подсетях, избегая NAT.

Реализация службы AWS в протоколе IPv6 не имеет механизма NAT, и все экземпляры, настроенные для IPv6, получают общедоступные (т.е. маршрутизируемые) адреса IPv6. Вы делаете частные подсети IPv6 конфиденциальными, подключая их через интернет-шлюз, предназначенный только для исходящих соединений (объект с префиксом `egw`), который блокирует входящие соединения. Шлюз запоминает состояние, поэтому внешние хосты могут общаться с серверами в частной сети IPv6, пока сервер AWS инициирует соединение.

Чтобы понять сетевую маршрутизацию для экземпляра, целесообразнее обратиться к таблице маршрутизации VPC для своей подсети, чем просматривать фактическую таблицу маршрутизации экземпляра (которую, например, может вывести на экран с помощью команды **`netstat -r`** или **`ip route show`** при регистрации в экземпляре). Версия VPC идентифицирует шлюзы ("цели") своими идентификаторами AWS, что упрощает анализ таблицы.

В частности, просмотрев таблицу маршрутизации VPC, можно легко отличить публичные подсети от частных. Если стандартный шлюз (т.е. цель с адресом `0.0.0.0/0`) является интернет-шлюзом (объект с префиксом `igw`), то эта подсеть является общедоступной. Если стандартный шлюз является устройством NAT (целью маршрута с префиксом идентификатора экземпляра `i` или `nat`), то подсеть является частной.

В табл. 13.13 показана примерная таблица маршрутизации для частной подсети.

Таблица 13.13. Пример таблицы маршрутизации VPC для частной подсети

Пункт назначения	Цель	Тип цели
`10.110.0.0/16`	`local`	Встроенный маршрут для локальной сети VPC
`0.0.0.0/0`	`nat-a31ed812`	Доступ в Интернет через шлюз VPC NAT
`10.120.0.0/16`	`pcx-38c3e8b2`	Пиринговое соединение с другой VPC
`192.168.0.0/16`	`vgw-1e513d90`	VPN-шлюз во внешнюю сеть

Сети VPC являются региональными, но подсети ограничены одной зоной доступности. Чтобы создать высокодоступные системы, используйте по меньшей мере одну

подсеть для каждой зоны и равномерно распределите экземпляры всех подсетей. В типичной схеме балансировщики нагрузки или другие прокси-серверы размещаются в публичных подсетях и ограничивается доступ к веб-серверам, приложения и серверам баз данных только из частных подсетей.

Группы безопасности и NACL

Группы безопасности — это брандмауэры для экземпляров EC2. Правила группы безопасности определяют, какие исходные адреса разрешены для трафика ICMP, UDP и TCP (входные правила), а какие порты в других системах могут быть доступны экземплярам (выходные правила). Группы безопасности по умолчанию запрещают все подключения, поэтому любые добавленные вами правила создают дополнительный трафик.

Все экземпляры EC2 принадлежат хотя бы одной группе безопасности, но могут быть частью даже пяти групп.[30] Чем больше групп безопасности принадлежит экземпляру, тем более запутанным может быть определение того, какой трафик разрешен, а какой нет. Мы предпочитаем, чтобы каждый экземпляр находился только в одной группе безопасности, даже если эта конфигурация приводит к некоторым повторяющимся правилам среди групп.

При добавлении правил в группы безопасности всегда учитывайте принцип наименьших привилегий. Открытие излишне большого количества портов создает угрозу для безопасности, особенно для систем с общедоступными маршрутизируемыми IP-адресами. Например, веб-серверу могут потребоваться только порты 22 (SSH, используется для управления и администрирования системой), 80 (HTTP) и 443 (HTTPS).

Кроме того, все хосты должны принимать ICMP-пакеты, используемые для реализации обнаружения MTU-пути. Неспособность принять эти пакеты может значительно снизить пропускную способность сети, поэтому мы с недоумением восприняли решение AWS блокировать их по умолчанию (см. страницу `goo.gl/WrETNq` (глубокая ссылка на сайте `docs.aws.amazon.com`), где описаны действия по включению этих пакетов).

Большинство групп безопасности имеют детальные правила, регламентирующие входящий трафик, но разрешают весь исходящий трафик, как показано в табл. 13.14. Эта конфигурация удобна, так как вам не нужно думать о том, какую внешнюю связь имеет ваша система. Тем не менее злоумышленникам легче достичь своей цели, если они могут подобрать средства и общаться с системами, находящимися под их внешним управлением. Самые безопасные сети имеют как входящие, так и исходящие ограничения.

Таблица 13.14. Типичные правила группы безопасности

Направление	Протокол	Порты	CIDR	Примечание
Вход	TCP	22	10.110.0.0/16	SSH из внутренней сети
Вход	TCP	80	0.0.0.0/0	HTTP из любого места
Вход	TCP	443	0.0.0.0/0	HTTPS из любого места
Вход	ICMP	n/a[a]	0.0.0.0/0	Разрешить обнаружение MTU пути
Выход	ALL	ALL	0.0.0.0/0	Исходящий трафик (все OK)

[a] Подробные инструкции см. на странице `goo.gl/WrETNq`; эта запись немного сложна для настройки.

[30] Группы безопасности фактически связаны с сетевыми интерфейсами, и экземпляр может иметь более одного сетевого интерфейса. Поэтому, чтобы быть совершенно правильно понятыми, мы должны сказать, что максимальное количество групп безопасности — это количество сетевых интерфейсов, умноженное на пять.

Подобно спискам контроля доступа на устройстве брандмауэра, списки NACL управляют трафиком между подсетями. В отличие от групп безопасности, списки NACL не имеют состояния: они не различают новые и существующие соединения. Они чем-то похожи на списки NACL на аппаратном брандмауэре. Списки NACL разрешают весь трафик по умолчанию. На практике группы безопасности используются гораздо чаще, чем NACL.

Пример архитектуры VPC

На рис. 13.6 изображены две сети VPC, каждая из которых содержит публичные и частные подсети. Сеть 2 размещает балансировщик эластичной нагрузки в своих публичных подсетях. ELB действует как прокси-сервер для некоторых экземпляров EC2 с автоматическим масштабированием, которые находятся в частной подсети и защищают эти экземпляры из Интернета. Службе 2 в сети 2 может потребоваться доступ к службе 1, размещенной в сети 1, и они могут общаться конфиденциально посредством пиринга VPC.

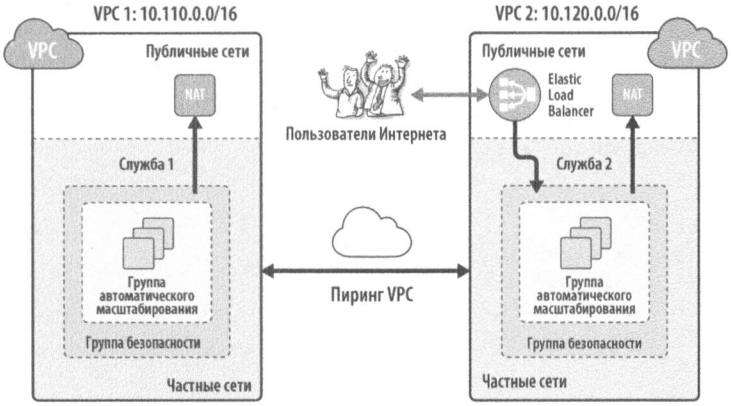


Рис. 13.6. Пиринг сетей VPC с публичными и частными подсетями

Архитектурные диаграммы, такие как на рис. 13.6, обеспечивают более четкие технические детали, чем письменные описания. Мы создаем такие диаграммы для каждого используемого приложения.

Создание VPC с помощью программы Terraform

Сети VPC состоят из множества ресурсов, каждый из которых имеет свои собственные настройки и параметры. Взаимозависимости между этими объектами сложны. Вы можете создавать и управлять практически всем, используя интерфейс командной строки (CLI) или веб-консоль, но этот подход требует, чтобы вы сохраняли все мелочи в своей голове. Даже если вы можете держать все детали прямо во время начальной настройки, отслеживать вашу работу с течением времени становится все сложнее.

Terraform, инструмент от HashiCorp, создает облачные ресурсы и управляет ими. Например, Terraform может создавать VPC, запускать экземпляры, а затем инициализировать эти экземпляры, запуская скрипты или другие инструменты управления конфигурацией. Конфигурация Terraform выражается на языке конфигурации HashiCorp (HCL), декларативном формате, который похож на JSON, но добавляет переменную интерполяцию и комментарии. Файл можно отслеживать в системе контроля версий, поэтому его легко обновлять и адаптировать.

В приведенном ниже примере показана конфигурация Terraform для простой сети VPC с одной открытой подсетью. Мы считаем, что она довольно понятно документирована.

```
# Диапазон адресов VPC в виде переменной
variable "vpc_cidr" {
   default = "10.110.0.0/16"
}

# Диапазон адресов для публичной подсети
variable "public_subnet_cidr" {
   default = "10.110.0.0/24"
}

# Сеть VPC
resource "aws_vpc" "default" {
cidr_block = "${var.vpc_cidr}"
enable_dns_hostnames = true
}

# Интернет-шлюз для соединения VPC с Интернетом
resource "aws_internet_gateway" "default" {
   vpc_id = "${aws_vpc.default.id}"
}

# Публичная подсеть
resource "aws_subnet" "public-us-west-2a" {
   vpc_id = "${aws_vpc.default.id}"
   cidr_block = "${var.public_subnet_cidr}"
   availability_zone = "us-west-2a"
}

# Таблица маршрутизации для публичной подсети
resource "aws_route_table" "public-us-west-2a" {
   vpc_id = "${aws_vpc.default.id}"
   route {
      cidr_block = "0.0.0.0/0"
      gateway_id = "${aws_internet_gateway.default.id}"
   }
}

# Связывание таблицы маршрутизации с публичной подсетью
resource "aws_route_table_association" "public-us-west-2-a" {
   subnet_id = "${aws_subnet.public-us-west-2a.id}"
   route_table_id = "${aws_route_table.public-us-west-2a.id}"
}
```

Документация Terraform является авторитетным источником информации. Множество конфигураций можно найти в репозитории Terraform и в Интернете.

Для того чтобы программа Terraform создала сеть VPC, выполняется команда **`terraform apply`**. Она анализирует текущий каталог (по умолчанию), находит **`.tf`**-файлы, обрабатывает каждый из них, создает план выполнения, а затем делает вызовы API в соответствующем порядке. Вы можете установить учетные данные AWS API в файле конфигурации или с помощью переменных среды `AWS_ACCESS_KEY_ID` и `AWS_SECRET_ACCESS_KEY`, как мы это сделали ниже.

```
$ AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
$ AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENGbPxRfiCYEXAMPLEKEY
$ time terraform apply
aws_vpc.default: Creating...
   cidr_block:                 "" => "10.110.0.0/16"
   default_network_acl_id:     "" => "<computed>"
   default_security_group_id:  "" => "<computed>"
   dhcp_options_id:            "" => "<computed>"
   enable_dns_hostnames:       "" => "1"
   enable_dns_support:         "" => "<computed>"
   main_route_table_id:        "" => "<computed>"
aws_vpc.default: Creation complete
aws_internet_gateway.default: Creating...
   vpc_id: "" => "vpc-a9ebe3cc"
aws_subnet.public-us-west-2a: Creating...
   availability_zone: "" => "us-west-2a"
   cidr_block: "" => "10.110.0.0/24"
   map_public_ip_on_launch: "" => "0"
   vpc_id: "" => "vpc-a9ebe3cc"
aws_subnet.public-us-west-2a: Creation complete
aws_route_table.public-us-west-2a: Creation complete
[snip]
Apply complete! Resources: 5 added, 0 changed, 0 destroyed.
real 0m4.530s
user 0m0.221s
sys 0m0.172s
```

Команда **time** измеряет, сколько времени потребуется для создания всех ресурсов в конфигурации (около 4,5 с). Значения <computed> указывают, что программа Terraform выбрала значения по умолчанию, потому что мы не указали эти настройки явно.

Состояние всех ресурсов, созданных Terraform, сохраняется в файле **terraform.tfstate**. Этот файл должен быть сохранен, чтобы программа Terraform знала, какие ресурсы находятся под ее контролем. В будущем Terraform самостоятельно откроет управляемые ресурсы.

Мы также легко можем стереть VPC.

```
$ terraform destroy -force
aws_vpc.default: Refreshing state... (ID: vpc-87ebe3e2)
aws_subnet.public-us-west-2a: Refreshing state... (ID: subnet-7c596a0b)
aws_internet_gateway.default: Refreshing state... (ID: igw-dc95edb9)
aws_route_table.public-us-west-2a: Refreshing state... (ID: rtb-2fc7214b)
aws_route_table_association.public-us-west-2-a: Refreshing state... (ID:
   rtbassoc-da479bbe)
aws_route_table_association.public-us-west-2-a: Destroying...
aws_route_table_association.public-us-west-2-a: Destruction complete
aws_subnet.public-us-west-2a: Destroying...
aws_route_table.public-us-west-2a: Destroying...
aws_route_table.public-us-west-2a: Destruction complete
aws_internet_gateway.default: Destroying...
aws_subnet.public-us-west-2a: Destruction complete
aws_internet_gateway.default: Destruction complete
aws_vpc.default: Destroying...
aws_vpc.default: Destruction complete
Apply complete! Resources: 0 added, 0 changed, 5 destroyed.
```

Программа Terraform не зависит от конкретного облака, поэтому она может управлять ресурсами для AWS, GCP, DigitalOcean, Azure, Docker и других поставщиков.

Когда использовать Terraform, а когда — CLI? Если вы создаете инфраструктуру для команды или проекта, или вам нужно будет внести изменения и повторить сборку позже, используйте Terraform. Если вам нужно запустить быстрый экземпляр в качестве теста, просмотреть сведения о ресурсе или получить доступ к API из сценария оболочки, используйте CLI.

Сеть на платформе Google Cloud Platform

На платформе Google Cloud Platform сетевое взаимодействие является функциональной частью платформы, а не представляет собой отдельный сервис. Частные сети GCP являются глобальными: экземпляр в регионе `us-east1` может связываться с другим экземпляром в регионе `europe-west1` через частную сеть, что упрощает создание глобальных сетевых служб. Сетевой трафик между экземплярами в одной и той же зоне является бесплатным, но есть плата за трафик между зонами или регионами.

Новые проекты имеют сеть по умолчанию с диапазоном адресов `10.240.0.0/16`. Вы можете создать до пяти отдельных сетей для каждого проекта, а экземпляры являются членами одной сети. Многие сайты используют эту сетевую архитектуру для изоляции тестов и разработки от производственных систем.

Сети могут подразделяться по регионам с подсетями. Это относительно недавнее дополнение к платформе GCP, которое отличается от подсетей на базе AWS. Глобальная сеть не должна быть частью одного диапазона префиксов IPv4, и в каждом регионе может быть несколько префиксов. Платформа GCP настраивает всю маршрутизацию, поэтому экземпляры на разных CIDR-блоках в одной и той же сети могут по-прежнему взаимодействовать друг с другом. Эта топология показана на рис. 13.7.



Рис. 13.7. Частная сеть GCP с подсетями и несколькими регионами

Подсети не классифицируются как публичные или частные; вместо этого экземпляры, которые не должны принимать входящий трафик из Интернета, могут просто не иметь общедоступного адреса, обращенного к Интернету. Компания Google предлагает статические внешние IP-адреса, которые вы можете использовать в записях DNS, не

опасаясь, что они будут назначены другому клиенту. Когда экземпляр имеет внешний адрес, вы все равно не увидите его, выполнив команду **ip addr show**; компания Google выполняет перевод адресов для вас.

По умолчанию правила брандмауэра в сети GCP применяются ко всем экземплярам. Чтобы ограничить правила меньшим набором экземпляров, вы можете пометить экземпляры и фильтровать правила в соответствии с тегами. Глобальные правила брандмауэра по умолчанию запрещают все, кроме следующих:

- ICMP-трафик для 0/0;
- RDP (удаленный настольный компьютер для Windows, TCP-порт 3389) для 0/0;
- SSH (порт TCP 22) для 0/0;
- все порты и протоколы для внутренней сети (по умолчанию 10.240.0.0/16).

Когда речь идет о решениях, влияющих на безопасность, мы всегда возвращаемся к принципу наименьших привилегий. В этом случае мы рекомендуем сузить эти правила по умолчанию, чтобы полностью блокировать RDP, разрешать SSH только из ваших собственных исходных IP-адресов и дополнительно ограничивать трафик в сети GCP. Вы также можете заблокировать ICMP, но имейте в виду, что вам нужно разрешить ICMP-пакеты типа 3, код 4, чтобы включить обнаружение MTU-пути.

Сеть DigitalOcean

Сеть DigitalOcean не имеет частной сети, по крайней мере такой, как GCP и AWS. Дроплеты могут иметь частные интерфейсы, которые обмениваются данными по внутренней сети в одном регионе. Однако эта сеть используется совместно со всеми другими клиентами DigitalOcean в том же регионе. Это небольшое улучшение по сравнению с использованием Интернета, но брандмауэры и шифрование в пути становятся жесткими требованиями.

Мы можем исследовать загруженный дроплет DigitalOcean с помощью интерфейса CLI **tugboat**:

```
$ tugboat info ulsah
Droplet fuzzy name provided. Finding droplet ID...done, 8857202
   (ulsah-ubuntu-15-10)
Name:            ulsah-ubuntu-15-10
ID:              8857202
Status:          active
IP4:             45.55.1.165
IP6:             2604:A880:0001:0020:0000:0000:01EF:D001
Private IP:      10.134.131.213
Region:          San Francisco 1 - sfo1
Image:           14169855 - ubuntu-15-10-x64
Size:            512MB
Backups Active:  false
```

Вывод включает в себя адрес IPv6 в дополнение к общедоступным и частным адресам IPv4. На примере мы можем продолжить изучение, просмотрев адреса на локальных интерфейсах.

```
# tugboat ssh ulsah-ubuntu-15-10
# ip address show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
     state UP group default qlen 1000
     link/ether 04:01:87:26:d6:01 brd ff:ff:ff:ff:ff:ff
```

```
    inet 45.55.1.165/19 brd 45.55.31.255 scope global eth0
      valid_lft forever preferred_lft forever
    inet 10.12.0.8/16 scope global eth0
      valid_lft forever preferred_lft forever
    inet6 fe80::601:87ff:fe26:d601/64 scope link
      valid_lft forever preferred_lft forever
# ip address show eth1
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
    state UP group default qlen 1000
    link/ether 04:01:87:26:d6:02 brd ff:ff:ff:ff:ff:ff
    inet 10.134.131.213/16 brd 10.134.255.255 scope global eth1
      valid_lft forever preferred_lft forever
    inet6 fe80::601:87ff:fe26:d602/64 scope link
      valid_lft forever preferred_lft forever
```

Общий адрес присваивается непосредственно интерфейсу `eth0`, а не транслируется провайдером, как на других облачных платформах. Каждый интерфейс также имеет IPv6-адрес, поэтому можно одновременно обслуживать трафик через IPv4 и IPv6.

13.16. Литература

История

- Comer, Douglas E. *Internetworking with TCP/IP Volume 1: Principles, Protocols, and Architectures (6th Edition)*. Upper Saddle River, NJ: Prentice Hall, 2013. Эта книга долгое время являлась стандартным справочником по протоколам TCP/IP. Она написана как учебник для студентов и является хорошим введением в тему.
- Salus, Peter H. *Casting the Net, From ARPANET to INTERNET and Beyond*. Reading, MA: Addison-Wesley Professional, 1995. Это прекрасная история ARPANET, предтечи Интернета, написанная ученым, который так долго был близко знаком с разработчиками системы UNIX, что стал восприниматься как один из них. Эта книга представляет собой отличный сборник документов об истории Интернета и его различных технологиях (см. сайт `isoc.org/internet/history`).

Классика

- Stevens, W. Richard. *UNIX Network Programming*. Upper Saddle River, NJ: Prentice Hall, 1990.
- Stevens, W. Richard, Bill Fenner, and Andrew M. Rudoff. *UNIX Network Programming, Volume 1, The Sockets Networking API (3rd Edition)*. Upper Saddle River, NJ: Addison-Wesley, 2003.
- Stevens, W. Richard. *UNIX Network Programming, Volume 2: Interprocess Communications (2nd Edition)*. Upper Saddle River, NJ: Addison-Wesley, 1999.

  Эти книги — канонические учебники о сетевых классах, в том числе о сетевом программировании в UNIX. Если вам нужен только интерфейс сокетов Berkeley, оригинальное издание по-прежнему является прекрасным источником знаний. Если вам нужен интерфейс STREAMS, то третья версия, включающая IPv6, — хороший выбор. Все три книги четко и ясно написаны в типичном стиле Ричарда Стивенса.

- Tanenbaum, Andrew S., and David J. Wetherall. *Computer Networks (5th Edition)*. Upper Saddle River, NJ: Prentice Hall PTR, 2011.

  Это первая книга о сетях, которая стала классикой. Она содержит подробное описание всех деталей физических и канальных уровней стека протоколов. Последнее издание охватывает беспроводные сети, гигабитную сеть Ethernet, одноранговые сети, голосовую IP-телефонию, сотовые сети и т.д.

Протоколы

- Fall, Kevin R., and W. Richard Stevens. *TCP/IP Illustrated, Volume One: The Protocols (2nd Edition)*. Reading, MA: Addison-Wesley, 2011.
- Wright, Gary R., and W. Richard Stevens. *TCP/IP Illustrated, Volume Two: The Implementation*. Reading, MA: Addison-Wesley, 1995.

  Книги в серии TCP/IP Illustrated — отличное и подробное руководство по стеку протоколов TCP/IP.
- Hunt, Craig. *TCP/IP Network Administration (3rd Edition)*. Sebastopol, CA: O'Reilly Media, 2002. Как и другие книги данной серии, эта книга предназначена для администраторов систем UNIX. Половина книги посвящена протоколу TCP/IP, а остальная часть описывает высокоуровневые функции UNIX, такие как электронная почта и удаленная регистрация.
- Farrel, Adrian. *The Internet and Its Protocols: A Comparative Approach*. San Francisco, CA: Morgan Kaufmann Publishers, 2004.
- Kozierak, Charles M. *The TCP/IP Guide: A Comprehensive, Illustrated Internet Protocols Reference*. San Francisco, CA: No Starch Press, 2005.
- Donahue, Gary A. *Network Warrior: Everything You Need to Know That Wasn't on the CCNA Exam*. Sebastopol, CA: O'Reilly Media, 2015.

Глава 14

Сетевые аппаратные средства



Независимо от того, работают ваши системы в центре обработки данных, в облаке или пусковой ракетной шахте, у них есть нечто общее — необходимость обмена информацией по сети. Возможность быстрой и надежной передачи данных необходима в любой среде. Если есть область, в которой технология UNIX затронула человеческие жизни и повлияла на другие операционные системы, то она связана с практической реализацией крупномасштабного пакетированного транспорта данных.

Сети проходят такую же эволюцию, как и серверы, поскольку физические и логические представления сети все больше разделяются уровнем виртуализации, имеющим собственную конфигурацию. В облаке такие конфигурации являются стандартными, но даже физические центры обработки данных в настоящее время часто включают в себя слой программно конфигурируемых сетей (software-defined networking — SDN).

Администраторы взаимодействуют с сетевым оборудованием реального мира менее часто, чем когда-то, но знакомство с традиционными сетями остается решающим навыком. Виртуализированные сети тесно имитируют физические сети в своих функциях, терминологии, архитектуре и топологии.

Многие сетевые технологии продвигались в течение долгих лет, но в результате появился очевидный победитель — Ethernet. Сегодня технологию Ethernet можно встретить всюду: от игровых приставок до холодильников. Глубокое понимание принципов работы этой системы чрезвычайно важно для успешной работы системного администратора.

Совершенно очевидно, что быстродействие и надежность сетей непосредственно влияют на результаты деятельности компаний. Однако в настоящее время сетевые технологии настолько вездесущи, что состояние сети может повлиять на возможность вза-

имодействия между людьми, например возможность делать телефонные звонки. Плохая организация сети — это личная и профессиональная неудача, которая может иметь катастрофические социальные последствия. Кроме того, устранение этих недостатков порой обходится очень дорого.

Успешное создание сети зависит от по крайней мере четырех важнейших факторов:

- разработки разумной структуры сети;
- выбора высококачественного оборудования;
- правильной инсталляции и документирования;
- компетентной эксплуатации и сопровождения.

В этой главе рассматриваются принципы, инсталляция и функционирование сетей Ethernet. Мы также кратко опишем такие устаревшие технологии, как DSL (Digital Subscriber Line), которые обычно предстают перед конечными пользователями в облике — сюрприз! — технологии Ethernet.

14.1. Технология Ethernet: сетевая панацея

Захватив более 95% мирового рынка локальных сетей (Local Area Network — LAN), технология Ethernet в самых разных формах проявляется почти всюду. Разработку стандарта Ethernet начал Боб Меткалф (Bob Metcalfe) из Массачусетского технологического института в рамках своей кандидатской диссертации, но в настоящее время она описана во многих стандартах IEEE.

В первоначальной спецификации Ethernet была определена скорость передачи данных 3 Мбит/с (мега*бит* в секунду), но почти сразу же она выросла до 10 Мбит/с. Как только в 1994 году была закончена работа над стандартом, предусматривавшим скорость 100 Мбит/с, стало ясно, что технология Ethernet будет лишь эволюционировать, а не вытесняться новой технологией. Это вызвало гонку технологий, в ходе которой производители старались создать все более быстродействующую версию Ethernet, и это соревнование еще не закончено. Основные этапы эволюции различных стандартов Ethernet приведены в табл. 14.1[1].

Таблица 14.1. Эволюция Ethernet

Год	Скорость	Название стандарта	Номер IEEE	Расстояние	Средство передачи[a]
1973	3 Мбит/с	Xerox Ethernet	–	?	Коаксиальный кабель
1976	10 Мбит/с	Ethernet 1	–	500 м	Коаксиальный кабель RG-11
1989	10 Мбит/с	10BASE-T	802.3	100 м	Медный кабель НВП категории 3
1994	100 Мбит/с	100Base-TX	802.3u	100 м	Медный кабель НВП категории 5
1999	1 Гбит/с	1000BASE-T ("gigabit")	802.3ab	100 м	Медный кабель НВП категорий 5e и 6
2006	10 Гбит/с	10GBASE-T ("10 Gig")	802.3an	100 м	ВП категории 6a, 7, НВП категории 7a
2009	40 Гбит/с	40GBASE-CR4	P802.3ba	10 м	Медный кабель НВП
		40GBASE-SR4		100 м	ММ-оптоволокно

[1] Мы не упомянули несколько менее популярных стандартов.

Окончание табл. 14.1

Год	Скорость	Название стандарта	Номер IEEE	Расстояние	Средство передачи[а]
2009	100 Гбит/с	100GBASE-CR10	P802.3ba	10 м	Медный кабель НВП
		100GBASE-SR10		100 м	ММ-оптоволокно
2018[б]	200 Гбит/с	200GBASE-FR4	P802.3bs[в]	2 км	CWDM-оптоволокно
		200Gbase-LR4		10 км	CWDM-оптоволокно
2018[б]	400 Гбит/с	400GBASE-SR16	P802.3bs	100 м	ММ-оптоволокно (16 жил)
		400Gbase-DR4		500 м	ММ-оптоволокно (4 жилы)
		400GBASE-FR8		2 км	CWDM-оптоволокно
		400Gbase-LR8		10 км	CWDM-оптоволокно
2020[б]	1Тбит/с	TbE	TBD	TBD	TBD

[а] ММ — многомодовое, НВП — неэкранированная витая пара, ВП — витая пара, CWDM — разреженное спектральное мультиплексирование.

[б] Промышленный проект.

[в] Мы немного сомневаемся и предполагаем, что этот вариант кодировки был неудачным совпадением.

Как работает Ethernet

Технологию Ethernet можно представить в виде великосветского раута, на котором гости (компьютеры) не перебивают друг друга, а ждут паузы в разговоре (отсутствия трафика в сетевом кабеле), чтобы заговорить. Если два гостя начинают говорить одновременно (т.е. возникает конфликт), оба они останавливаются, извиняются друг перед другом, ждут немного, а затем один из них начинает говорить снова.

В технической терминологии такая схема называется CSMA/CD (Carrier Sense Multiple Access with Collision Detection — множественный доступ с контролем несущей и обнаружением конфликтов). Смысл этого названия заключается в следующем:

- контроль несущей (CS) — вы можете говорить одновременно с другими;
- множественный доступ (MA) — говорить могут все;
- обнаружение конфликтов (CD) — вы знаете, когда перебиваете кого-то.

Фактическая задержка при обнаружении конфликтов является случайной. Это позволяет избежать такого развития событий, при котором два компьютера одновременно передают сообщения в сеть, обнаруживают коллизию, ждут некоторое время, а затем синхронно возобновляют передачу, переполняя, таким образом, сеть конфликтами.

В настоящее время важность соглашений CSMA/CD осознали даже приверженцы коммутаторов, которые обычно ограничивают количество хостов в домене, в котором происходят коллизии, до двух. (Если продолжить аналогию с великосветским раутом, можно описать этот вариант как ситуацию, в которой два собеседника, как в старом кино, чопорно сидят на противоположных концах длинного обеденного стола.)

Топология Ethernet

С точки зрения топологии сеть Ethernet представляет собой разветвляющуюся шину, но без петель. У пакета есть только один путь следования между любыми двумя хостами, расположенными в одной сети. В сети Ethernet могут передаваться пакеты трех типов: однонаправленные (unicast), групповые (multicast) и широковещательные (broadcast). Пакеты первого типа адресованы одному хосту, второго — группе хостов, третьего — всем хостам сегмента.

Широковещательный домен — это совокупность хостов, которые принимают пакеты, направляемые по аппаратному широковещательному адресу. В каждом логическом сегменте сети Ethernet существует только один широковещательный домен. В ранних стандартах Ethernet и средствах передачи (например, 10Base5) понятия физического и логического сегментов были тождественными, поскольку все пакеты передавались по одному большому кабелю, в который втыкались сетевые интерфейсы компьютеров[2].

С появлением современных коммутаторов логические сегменты стали включать в себя множество (десятки и даже сотни) физических сегментов, к которым подключено всего два устройства: порт коммутатора и компьютер. Коммутаторы отвечают за доставку групповых и однонаправленных пакетов в физический сегмент, где расположен нужный адресат (адресаты); широковещательные пакеты направляются во все сетевые порты логического сегмента.

С появлением коммутаторов сегодняшние логические сегменты обычно состоят из многих физических сегментов (возможно, десятков или сотен), к которым подключены только два устройства: порт коммутатора и хост.[3] Коммутаторы несут ответственность за сопровождение многоадресатных и одноадресатных пакетов к физическим (или беспроводным) сегментам, на которых находятся предполагаемые получатели. Широковещательный трафик пересылается всем портам в логическом сегменте.

Логический сегмент может состоять из физических сегментов, имеющих разную скорость передачи данных. Следовательно, коммутаторы должны иметь средства буферизации и синхронизации для предотвращения возможных конфликтов.

Неэкранированная витая пара

Неэкранированная витая пара (НВП) — самая популярная среда передачи данных в сетях Ethernet. Общая схема сети на основе НВП изображена на рис. 14.1.

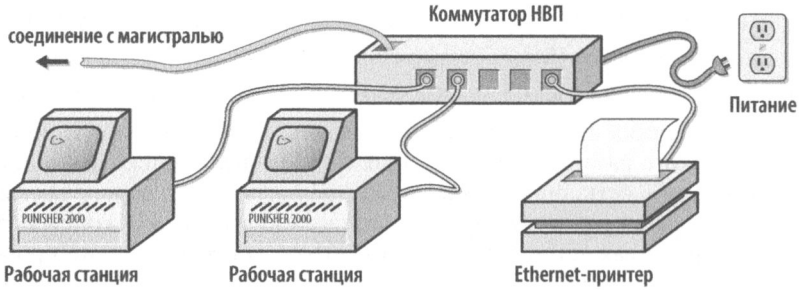


Рис. 14.1. Схема сети на основе НВП

[2]Мы не шутим! Подключение нового компьютера к сети предполагало прокалывание отверстия в изоляции кабеля с помощью специального соединителя, называемого «зуб вампира», который позволял добраться до центрального проводника. Этот соединитель затем зажимался винтами.

[3]Беспроводные сети — еще один распространенный тип логического сегмента Ethernet. Они ведут себя, скорее, как традиционные формы Ethernet, которые используют один кабель для соединения многих хостов сети (см. раздел 14.2).

Провода, используемые в современных локальных вычислительных сетях на основе неэкранированной витой пары, обычно подразделяют на восемь категорий. Эта система оценки параметров была впервые введена компанией Anixter, крупным поставщиком кабельной продукции, и впоследствии стандартизирована организацией TIA (Telecommunications Industry Association — ассоциация телекоммуникационной промышленности). Сегодня выделяются категории 1–7 и промежуточные категории 5e и 6a.

Организация ISO (International Organization for Standartization — Международная организация по стандартизации) тоже подключилась к процессу стандартизации кабелей и предложила собственную классификацию, которая почти в точности повторяет классификацию TIA. Например, кабель категории 5 в системе TIA эквивалентен кабелю класса D в системе ISO. Для наглядности в табл. 14.2 подытожены ключевые различия между основными современными стандартами кабелей. Эта таблица поможет вам произвести впечатление на своих друзей во время вечеринки.

Таблица 14.2. Характеристики кабелей НВП

Параметр[a]	Единица измерения	Категории						
		5 D[б]	5e	6 E	6a EA	7 F	7a FA	8 I
Ширина полосы	МГц	100	100	250	500	600	1000	2000
Затухание	дБ	24	24	21,7	18,4	20,8	60	50
NEXT	дБ	27,1	30,1	39,9	59	62,1	60,4	35,6
ELFEXT	дБ	17	17,4	23,2	43,1	46,0	35,1	-
Затухание отраженного сигнала (обратная потеря)	дБ	8	10	12	32	14,1	61,93	8
Задержка распространения сигнала	нс	548	548	548	548	504	534	548

[a]NEXT (Near-end crosstalk) — ослабление перекрестной наводки на ближнем конце. ELFEXT (Equal level far-end crosstalk) — ослабление равноуровневой перекрестной наводки на дальнем конце.

[б]Включая дополнительные спецификации TIA TSB 95 и ISO FDAM 2.

Кабель категории 5 поддерживает работу на скорости 100 Мбит/с. Кабели категорий 5e, 6 и 6a поддерживают скорость передачи 1 Гбит/с и в настоящее время используются в качестве стандарта. Кабель категории 6a лучше всего подходит для организации новых сетей, поскольку он особенно устойчив к помехам, возникающим из-за использования старых стандартов передачи сигналов (например, 10BASE-T). При прокладке кабелей категорий 5 и 5e были зафиксированы определенные проблемы. Кабели категорий 7 и 7a предназначены для передачи данных со скоростью 10 Гбит/с, а кабели категорий 8 — 40 Гбит/с.

Более быстродействующие стандарты требуют применения нескольких пар НВП. Это ускоряет передачу данных по линии связи по сравнению с использованием единственной пары. Для соединений 10BASE-T требуются две пары проводов категории 5. В соединениях 100BASE-TX предельная длина та же, но используются две пары проводов категории 5. Соединение 1000BASE-TX требует четырех пар проводов категорий 5e или 6/6a. Аналогично соединение 10GBASE-TX требует четырех пар проводов категорий 6a, 7 или 7a. Длина кабеля во всех стандартах ограничена 100 м.

Существуют провода с поливинилхлоридной и тефлоновой изоляцией. Выбор изоляции диктуется средой, в которой будут проложены кабели. В замкнутых помещениях, связанных с вентиляционной системой здания, обычно требуется тефлоновая изоляция[4]. Поливинилхлоридная изоляция дешевле и проще в эксплуатации.

[4]Конкретную информацию можно получить у пожарного инспектора или ответственного за пожарную безопасность.

Подключая четырехпарный НВП-кабель к коммутационным панелям и настенным розеткам RJ-45, придерживайтесь стандарта разводки TIA/EIA-568A. Этот стандарт, совместимый с другими вариантами RJ-45 (например, RS-232), позволяет избежать ошибок при разводке концов кабеля, независимо от того, есть ли свободный доступ к парам. Требования стандарта отражены в табл. 14.3.

Таблица 14.3. Стандарт TIA/EIA-568A для подключения четырехпарного НВП-кабеля к розетке RJ-45

Пара	Цвета	Контакты разъема
1	Белый/синий	5/4
2	Белый/оранжевый	3/6
3	Белый/зеленый	1/2
4	Белый/коричневый	7/8

Имеющаяся в здании проводка может не подходить для прокладки сетей, в зависимости от того, как и когда она прокладывалась.

Оптическое волокно

Оптическое волокно используется в тех ситуациях, когда применение медного кабеля по тем или иным причинам неприемлемо. Оптическое волокно передает сигнал быстрее, чем медный провод. Кроме того, оно является более устойчивым к электрическим помехам, что в некоторых приложениях очень важно. Там, где оптическое волокно не является абсолютно необходимым, обычно выбирают медный кабель, поскольку он дешевле и с ним легче работать.

Оптическое волокно бывает "многомодовым" и "одномодовым". Многомодовое оптическое волокно обычно используется в зданиях или комплексах зданий. Оно толще, чем одномодовое, и может проводить несколько лучей света; это свойство позволяет использовать менее дорогую электронику (например, в качестве источника света можно использовать светодиоды).

Одномодовое оптическое волокно часто используется в магистральных приложениях, например для прокладки линий связи между городами и регионами. Оно может проводить только один световой луч и требует дорогой прецизионной электроники в конечных точках.

Стандарт TIA-598C рекомендует цветовую кодировку оптического волокна, представленную в табл. 14.4. Следует помнить основное правило: все элементы должны соответствовать друг другу. Оптическое волокно, соединяющее конечные точки, оптические кабели перекрестной коммутации и электронные приборы, установленные в конечных точках, должны иметь один и тот же тип и размер. Обратите внимание на то, что кабели OM1 и OM2 не являются взаимозаменяемыми, хотя и окрашены в один и тот же оранжевый цвет — проверьте размеры, указанные на кабелях, чтобы убедиться, что они соответствуют друг другу. Если вы нарушите это правило, то вам будет сложно обеспечить изоляцию в конечных точках.

На концах оптических волокон используются разъемы более чем 30 типов, и нет ни четких правил, ни принципов, регламентирующих их выбор. В каждой конкретной ситуации на выбор того или иного типа разъема влияют поставщики оборудования или параметры оптического волокна, уже проложенного внутри здания.

Таблица 14.4. Атрибуты стандартных оптических волокон

Количество мод	Название ISO*	Диаметр сердечника, мкм	Диаметр оптической оболочки, мкм	Цвет
Много	OM1	62,5	125	Оранжевый
Много	OM2	50	125	Оранжевый
Много	OM3	50 *	125	Голубой
Одна	OS1	8–10	125	Желтый

[a] В соответсвии со стандартатом ISO 11801.

[б] OM3 оптимизирован под лазерный луч.

Соединение и расширение сетей Ethernet

Сети Ethernet можно соединять с помощью устройств нескольких типов. На выбор устройств, описанных ниже, влияет их стоимость, причем более дешевые устройства описаны в первую очередь. Чем сложнее логические правила, по которым устройства перемещают биты из одной сети в другую, тем больше аппаратного и встроенного программного обеспечения необходимо и тем более дорогой становится сеть.

Концентраторы

Концентраторы (hub) иногда еще называют *повторителями* (repeators). Это активные устройства, используемые для соединения сегментов сетей Ethernet на физическом уровне. Им требуется внешний источник питания.

Выступая в качестве повторителя, концентратор ретранслирует Ethernet-фреймы, но никак не интерпретирует их. Он "не имеет представления" ни о том, куда направляются пакеты, ни о том, какой протокол они используют. За исключением экзотических ситуаций, *концентраторы больше не должны использоваться в промышленных сетях*, и мы не советуем их использовать даже в домашних сетях. (Почему? Потому что коммутаторы (switches) значительно эффективнее используют полосу пропускания частот в сети и в настоящее время стоят недорого.)

Коммутаторы

Коммутатор соединяет сети Ethernet на канальном уровне. Его назначение — объединить две физические сети так, чтобы они выглядели как одна большая физическая сеть. В настоящее время коммутаторы являются промышленным стандартом для соединения устройств Ethernet.

Коммутаторы принимают, регенерируют и ретранслируют пакеты на аппаратном уровне. Они используют алгоритм динамического обучения. Коммутаторы запоминают, какие исходные адреса поступают с одного порта, а какие — с другого. Пакет переходит из одного порта в другой только при необходимости. Первоначально пересылаются все пакеты, но через несколько секунд, когда коммутатор изучит расположение большинства хостов сети, запускается механизм фильтрации.

Поскольку между сетями пересылаются не все пакеты, каждый сегмент кабеля менее загружен, чем в случае, когда все компьютеры подключены к одному кабелю. А если учесть, что основной трафик имеет тенденцию к локализации, то увеличение реальной пропускной способности может оказаться заметным. Кроме того, коммутатор не влияет на логическую модель сети, поэтому его установка требует лишь незначительного вмешательства со стороны администратора.

Если сеть имеет петли, коммутатор может безнадежно запутаться, потому что пакеты, посылаемые одним компьютером, окажутся сразу на двух (или более) портах коммутатора. В одной сети Ethernet петлей не бывает, но после объединения нескольких таких сетей с помощью маршрутизаторов и коммутаторов топология изменится, вследствие чего может образоваться несколько путей к одному хосту. Некоторые коммутаторы решают эту проблему путем резервирования альтернативных маршрутов на тот случай, если основной маршрут станет недоступным. Они упрощают топологию видимой ими сети, отсекая дублирующиеся пути до тех пор, пока в оставшихся сегментах не окажется только по одному маршруту к каждому хосту сети. Другие коммутаторы создают между сетями двойные каналы и переключают трафик по циклическому принципу.

Коммутаторы должны просматривать каждый пакет, определяя, нужно ли его пересылать в другой сегмент. Производительность этих устройств обычно измеряют как скоростью просмотра пакетов, так и скоростью их пересылки. Многие поставщики не указывают в диаграммах производительности коммутаторов размеры протестированных пакетов, поэтому реальная производительность может быть ниже объявленной.

Несмотря на то что быстродействие коммутаторов Ethernet все время растет, эффективно использовать их можно при объединении в один логический сегмент не более сотни компьютеров. В крупных коммутируемых сетях часто возникают проблемы наподобие "широковещательных штормов", поскольку широковещательный трафик должен проходить через все порты. Для решения этой проблемы нужно изолировать широковещательный трафик между коммутируемыми сегментами посредством маршрутизатора (создавая тем самым более одного логического Ethernet-сегмента).

Выбор коммутатора может представлять определенную трудность. В этом сегменте рынка очень высокая конкуренция, следствием которой являются многочисленные рекламные заявления, не всегда подтверждаемые на практике. Поэтому не стоит особо доверять данным, которые приводятся поставщиками; лучше прислушаться к советам независимых экспертов (просмотрите тесты, приводимые в журналах). В последние годы нередко случалось так, что чей-то продукт оказывался "лучшим" в течение нескольких месяцев, а затем, после попыток внесения улучшений, его производительность или надежность падала ниже критической отметки.

В любом случае убедитесь, что скорость объединительной панели коммутатора является достаточной. У хорошо спроектированного коммутатора эта скорость должна превышать сумму скоростей всех его портов.

Коммутаторы, позволяющие создавать виртуальные локальные сети

В крупных организациях можно использовать коммутаторы, позволяющие разбивать их порты (программным путем) на группы, называемые виртуальными локальными сетями (Virtual Local Area Network — VLAN). Виртуальная локальная сеть — это группа портов, принадлежащая к одному логическому сегменту, как если бы порты были соединены со своим собственным выделенным коммутатором. Подобное секционирование позволяет повысить степень изоляции трафика, что полезно с точки зрения как безопасности, так и производительности.

Трафиком между виртуальными локальными сетями управляет маршрутизатор или, в некоторых случаях, модуль маршрутизации или уровень программной маршрутизации самого коммутатора. Расширение этой системы, называемое *транкингом виртуальной локальной сети* (один из примеров реализации — протокол IEEE 802.1Q), позволяет разным коммутаторам обслуживать порты одной логической виртуальной локальной сети.

Важно помнить, что сами сети VLAN почти не обеспечивают дополнительной защиты. Для того чтобы обеспечить защиту, необходимо фильтровать трафик VLAN.

Маршрутизаторы

Маршрутизаторы (известные также как "коммутаторы третьего уровня") направляют трафик на третьем сетевом уровне модели OSI. Маршрутизаторы доставляют пакеты адресатам на основании информации, хранящейся в TCP/IP-заголовках. Помимо простого перемещения пакетов, маршрутизаторы могут также выполнять ряд особых функций, например фильтрацию пакетов (в соответствии с правилами безопасности), разделение трафика по приоритетам (в соответствии с заданным качеством обслуживания) и обнаружение общей сетевой топологии.

Конфигурация маршрутизаторов бывает фиксированной или модульной.

- Устройства первого типа содержат сетевые интерфейсы, установленные в заводских условиях. Они обычно подходят для специализированных применений. Например, маршрутизатор с интерфейсами T1 и Ethernet может оказаться удобным, когда нужно подключить небольшую компанию к Интернету.
- Модульные маршрутизаторы имеют слотовую или шинную архитектуру, а интерфейсы к ним добавляются пользователями. Как правило, это более дорогие устройства, но зато они гибче в эксплуатации.

В зависимости от необходимой надежности и ожидаемого трафика, специализированный маршрутизатор может оказаться как дороже, так и дешевле системы UNIX или Linux, сконфигурированной в качестве маршрутизатора. Однако специализированное устройство, как правило, демонстрирует более высокую производительность и надежность. Это та область сетевого проектирования, где лучше заранее вложить чуть больше денег, чем потом иметь головную боль.

Автосогласование

С появлением разных стандартов Ethernet возникла необходимость, чтобы устройства могли идентифицировать конфигурацию своих соседей и согласовывать с ними свои настройки. Например, сеть не будет работать, если на одной стороне соединения она работает со скоростью 1 Гбит/с, а на другой — со скоростью 10 Гбит/с. Для выявления и решения этой проблемы организацией IEEE был разработан стандарт автосогласования Ethernet. В одних случаях он работает, а в других применяется неправильно и лишь усугубляет проблему.

Следует запомнить два золотых правила автосогласования.

- Вы *обязаны* использовать автосогласование всех интерфейсов, работающих на скорости 1 Гбит/с и выше. Этого требует стандарт.
- Если интерфейсы ограничены скоростями 100 Мбит/с и ниже, необходимо либо конфигурировать *оба конца* соединения, либо вручную настроить скорость и дуплекс (половинный или полный) *обеих* сторон. Если в режиме автосогласования настроить только одну сторону соединения, то в большинстве случаев она не сможет выяснить, какую конфигурацию имеет другая сторона. В результате конфигурация станет несогласованной и производительность упадет.

Для того чтобы выяснить, как задать стратегию автосогласования интерфейсов, прочитайте специальный раздел 13.10.

Передача электропитания по сетям Ethernet

Технология передачи питания по сетям Ethernet (Power on Ethernet — PoE) основана на передаче электропитания по той же неэкранированной витой паре (UTP Ethernet), по которой передается сигнал Ethernet. Данная технология регламентируется стандартом IEEE 802.3af. Это особенно удобно для систем связи, обеспечивающих передачу речевого сигнала по сети Интернет (Voice over IP — VoIP), или пунктов доступа к системе беспроводной связи (мы указали только два примера, но список можно продолжить), в которых требуется как маломощный источник питания, так и сетевое соединение.

По мощности питания системы PoE разделяются на четыре класса в диапазоне от 3,84 до 25,5 Вт. Промышленность, которая никогда не останавливается на достигнутом, уже работает над новым стандартом (802.3bt), предусматривающим более высокую мощность (более 60 Вт). Будет ли этого достаточно, чтобы подключить духовку Easy-Bake к сетевому порту в конференц-зале?[5]

Технология PoE порождает два обстоятельства, о которых должен знать системный администратор.

- Вы должны знать о существовании устройств PoE в вашей инфраструктуре, чтобы правильно спланировать доступ к портам коммутаторов, поддерживающих технологию PoE. Эти порты дороже, чем порты, не поддерживающие технологию PoE.
- Вычисляя расход электроэнергии на обслуживание коммуникационных шкафов, содержащих коммутаторы PoE, следует учитывать мощность устройств PoE. Обратите внимание на то, что вы не должны учитывать дополнительный расход электроэнергии на охлаждение коммуникационных шкафов, поскольку большая часть тепла, выделяемого из-за потребления мощности PoE, рассеивается за пределами шкафа (обычно по офису).

Гигантские пакеты

Технология Ethernet стандартизована для типичного пакета размером 1 500 байт (вместе с фреймом — 1 518 байт). Это значение было выбрано давно, когда сети были медленными и память для буферов была дефицитной. В настоящее время пакеты размером 1 500 байт выглядят крохотными в контексте гигабитных сетей Ethernet. Поскольку с каждым пакетом связаны накладные расходы и определенное время задержки, производительность сети можно повысить, если допустить более крупные размеры пакетов.

К сожалению, стандарты IEEE для разных типов сетей Ethernet запрещают использование крупных пакетов по соображениям совместимости сетей. Однако, поскольку скорость магистрального трафика часто во много раз превышает установленный предел, нестандартные большие пакеты Ethernet в современных сетях перестали быть редкостью. Подстрекаемые нетерпеливыми потребителями, производители сетевого оборудования негласно бойкотируют стандарт IEEE и обеспечивают поддержку крупных фреймов в своей гигабитной продукции.

Для использования так называемых *гигантских пакетов* (jumbo frames) необходимо лишь повысить максимально возможный размер пакета (maximal transmission unit — MTU) в интерфейсах сети. Повышение производительности зависит от вида трафика, но наибольший выигрыш достигается для крупномасштабных перемещений по протоколу TCP (например, в файловых службах NFSv4 или CIFS). Ожидается, что умеренное, но заметное повышение производительности должно составить примерно 10%.

---

[5]Для интересующихся этим вопросом: да, существует возможность загрузить небольшую систему Linux через порт сети PoE. Возможно, проще всего это сделать с помощью Raspberry Pi и коммутатора Pi PoE Switch HAT.

Тем не менее следует отметить следующее.

- Поддерживать и использовать гигантские пакеты должно все сетевое оборудование в подсетях, включая коммутаторы и маршрутизаторы. Их нельзя смешивать и подгонять.
- Поскольку гигантские пакеты являются нестандартными, обычно их необходимо разрешать явным образом. Устройства могут принимать гигантские пакеты по умолчанию, но, вероятнее всего, они не будут их генерировать.
- Поскольку гигантские пакеты представляют собой незаконное явление, не существует соглашения, насколько большими они могут или должны быть. Типичной величиной является 9000 байт или 9018 вместе с фреймом. Необходимо проверить, какой максимальный размер пакета может принять ваше устройство. Пакеты размером больше 9 Кбайт иногда называют сверхгигантскими, но это экзотическое название вас пугать не должно. Чем больше размер, тем лучше, по крайней мере в диапазоне до 64 Кбайт.

Мы одобряем использование гигантских пакетов в гигабитных сетях Ethernet, но будьте готовы к дополнительной отладке, если что-то пойдет не так, как надо. Лучше всего развернуть новую сеть, задав максимально возможный размер пакета по умолчанию, а позднее, когда надежность сети будет проверена, изменить эти настройки и разрешить гигантские пакеты.

14.2. Беспроводные сети: локальная сеть для кочевников

Беспроводные сети состоят из беспроводных точек доступа (Wireless Access Points — WAP) и клиентов беспроводной сети. Точки WAP могут соединяться традиционными проводными сетями (обычная конфигурация) или с другими точками WAP без использования проводов (конфигурация известна под названием "беспроводная сеть").

Стандарты беспроводных сетей

Распространенными стандартами беспроводных сетей в настоящее время являются IEEE 802.11g, 802.11n и .802.11ac Стандарт 802.11g работает на частоте 2,4 ГГц и обеспечивает доступ к локальной сети со скоростью, достигающей 54 Мбит/с. Радиус действия одной точки доступа колеблется от 100 м до 40 км, в зависимости от оборудования и физических особенностей местности.

Стандарт 802.11n обеспечивает скорость до 600 Мбит/с [6] и может использовать частоты как 5 ГГц, так и 2,4 ГГц (при этом рекомендуется использовать диапазон 5 ГГц). Радиус действия точки доступа в стандарте IEEE 802.11n в два раза больше, чем в стандарте IEEE 802.11g. Преемником стандарта IEEE 802.11n является стандарт IEEE 802.11ac, поддерживающий производительность многостанционной сети на уровне до 1 Гбит/с.

Все эти стандарты обозначаются одним общим термином Wi-Fi. Формально говоря, метка Wi-Fi ограничена семейством стандартов IEEE 802.11. Однако это лишь один из многих видов аппаратного обеспечения Ethernet, доступных на рынке, поэтому все беспроводные сети Ethernet называются Wi-Fi.

[6] Скорость 600 Мбит/с в стандарте 802.11n является, скорее, теоретической. На практике полоса пропускания в окрестности точки WAP при оптимальной конфигурации может обеспечить скорость передачи данных не более 400 Мбит/с. Это объясняется различием между теоретическими и практическими возможностями оборудования и среды. В беспроводных сетях всякое бывает!

В настоящее время стандарты 802.11g и 802.11n стали общепринятыми. Трансиверы недороги и встроены в большинство ноутбуков. Кроме того, платы расширения также стоят недорого и доступны для любых персональных компьютеров.

Доступ клиентов к беспроводной сети

Вы можете настроить системы UNIX или Linux для подключения к беспроводной сети в качестве клиента, если у вас есть правильное оборудование и драйвер. Поскольку большинство беспроводных плат на базе персональных компьютеров все еще предназначены для системы Microsoft Windows, они могут не поставляться с завода с драйверами FreeBSD или Linux.

При попытке добавить беспроводное подключение к системе FreeBSD или Linux вам, скорее всего, понадобятся следующие команды:

- **`ifconfig`** — для конфигурирования интерфейса беспроводной сети;
- **`iwlist`** — для получения списка доступных точек доступа к беспроводной сети;
- **`iwconfig`** — для настройки параметров беспроводного соединения;
- **`wpa_supplicant`** — для аутентификации в беспроводной сети (или проводной сети 802.1x).

К сожалению, гонка продаж дешевого оборудования часто означает, что для настройки правильной работы беспроводного адаптера в системе UNIX или Linux может потребоваться много часов проб и ошибок. Планируйте все заранее или выясните в Интернете, какой адаптер лучше всего подходит для вашей операционной системы.

Беспроводные коммутаторы и точки беспроводного доступа

Все хотят иметь доступ к беспроводной сети в любом месте, и для обеспечения этой услуги доступно множество продуктов. Но, как и во многих других областях, вы получаете то, за что платите. Недорогие устройства часто удовлетворяют потребности домашних пользователей, но не могут хорошо масштабироваться в корпоративной среде.

Топология беспроводных сетей

Точки беспроводного доступа (Wireless Access Point — WAP) обычно представляют собой специализированные устройства, состоящие из одной или нескольких радиостанций и некоторой формы встроенной сетевой операционной системы, часто урезанной версии Linux. Одна точка WAP может обеспечить подключение нескольких клиентов, но их число ограничено. Хорошее эмпирическое правило состоит в том, чтобы одновременно обслуживать не более сорока клиентов с помощью одной корпоративной точки WAP. В качестве клиента может действовать любое устройство, которое обменивается данными по беспроводному стандарту, поддерживаемому вашими точками WAP.

Точки WAP имеют один или несколько "служебных идентификаторов сети", а также идентификатор SSID, который служит именем беспроводной локальной сети и должен быть уникальным в определенной окрестности. Когда клиент хочет подключиться к беспроводной локальной сети, он выясняет, какие идентификаторы SSID доступны, и выбирает одну из этих сетей.

Вы можете сделать имя своего идентификатора SSID осмысленным и легко запоминающимся, например `Third Floor Public` (Третий этаж, открытый доступ), или изобрести что-нибудь необычное. Некоторые из наших любимых имен SSID:

- `FBI Surveillance Van` (Служба наблюдения ФБР);
- `The Promised LAN` (Обещанная локальная сеть);
- `IP Freely` (Свободный IP);
- `Get Off My LAN` (Убирайся из моей локальной сети);
- `Virus Distribution Center` (Центр распространения вирусов);
- `Access Denied` (Доступ запрещен).

Нет ничего лучше, чем изобретательные чудаки... В простейших сценариях точка WAP объявляет единственный SSID, ваш клиент подключается к этому SSID и всё — вы в сети!

Тем не менее несколько аспектов беспроводной сети действительно просты. Что делать, если ваш дом или здание слишком большие, чтобы обслуживаться одной точкой WAP? Или что если вам нужно предоставлять разные сети различным группам пользователей (например, сотрудникам или гостям)? Для этих случаев вам необходимо стратегически структурировать свою беспроводную сеть.

Вы можете использовать несколько SSID для разбивки групп пользователей или функций. Как правило, вы сопоставляете их с отдельными виртуальными локальными сетями, которые затем можно маршрутизировать или фильтровать по желанию, как и проводные сети.

Частотный спектр, выделенный для беспроводной сети 802.11, разбивается на полосы, обычно называемые *каналами*. Точка WAP самостоятельно выбирает свободный радиоканал для объявления SSID. Клиенты и точка WAP используют этот канал для связи, формируя единый широковещательный домен. Ближайшие точки WAP, скорее всего, будут выбирать другие каналы, чтобы максимизировать доступную полосу пропускания и минимизировать помехи.

Теория состоит в том, что по мере того, как клиенты перемещаются по окружающей среде, они будут отделяться от одной точки WAP, когда ее сигнал становится слабым, и соединяться с ближней точкой WAP с более сильным сигналом. Однако теория и реальность часто не согласуются друг с другом. Многие клиенты поддерживают связь с точкой WAP, излучающей слабый сигнал, и игнорируют лучшие варианты.

В большинстве ситуаций вы должны разрешить WAP автоматически выбирать свои предпочтительные каналы. Если вы должны вручную вмешаться в этот процесс, используя стандарты 802.11b/g/n, рассмотрите выбор между каналами 1, 6 или 11. Спектр, выделенный этим каналам, не перекрывается, поэтому комбинации этих каналов обеспечивают наибольшую вероятность широкого распространения — открытую беспроводную магистраль. Каналы по умолчанию для 802.11a/ac не перекрываются вообще, поэтому просто выберите свой любимый номер.

Некоторые точки WAP имеют несколько антенн и используют технологию множественного ввода и множественного вывода (multiple-input, multiple-output — MIMO). Эта практика может увеличить доступную полосу пропускания, используя несколько передатчиков и приемников, чтобы использовать преимущества смещения сигнала в результате задержки распространения. В некоторых ситуациях эта технология может обеспечить небольшое улучшение производительности, хотя, вероятно, не такое значительное улучшение, как широкая сеть антенн.

Если вам нужна физически большая зона покрытия, разверните несколько точек WAP. Если область полностью открыта, вы можете развернуть их в структуре решетки. Если существуют физические препятствия вроде стен, проведите исследование для определения наилучших вариантов размещения точек WAP с учетом физических атрибутов вашего пространства.

Дешевая беспроводная связь

Нам нравятся продукты Ubiquiti (`ubnt.com`) для недорогих, высокопроизводительных домашних сетей. Google Wifi — замечательное облачное решение, если вы поддерживаете связь с удаленными членами семьи. Другим вариантом является запуск урезанной версии Linux (например, OpenWrt или LEDE) на коммерческой точке WAP (см. сайт `Openwrt.org` для получения дополнительной информации и списка совместимого оборудования).

Буквально десятки продавцов сейчас поставляют оборудование для точек беспроводного доступа. Вы можете купить их в Home Depot и даже в продуктовом магазине. Дешевые точки доступа (в диапазоне 30 долл.), вероятно, будут плохо работать при обработке больших файлов или наличии нескольких активных клиентов.

Дорогая беспроводная связь

Большая беспроводная связь означает большие деньги. Предоставление надежной беспроводной сети высокой плотности (в крупных больницах, спортивных учреждениях, школах, городах) представляет собой сложную задачу, связанную с ограничениями физических установок, плотностью пользователей и законами физики. В таких ситуациях вам нужны беспроводные устройства корпоративного класса, которые знают местоположение и состояние каждой точки WAP и активно настраивают каналы WAP, силу сигналов и группы клиентов, чтобы обеспечить наилучшие результаты. Эти системы обычно поддерживают прозрачный роуминг, который позволяет группе клиентов с определенной виртуальной локальной сетью и сессией беспрепятственно перемещаться между точками WAP.

Наши любимые крупные беспроводные платформы — это Aerohive и Meraki (последняя принадлежит компании Cisco). Эти платформы следующего поколения управляются из облака, что позволяет вам пить мартини на пляже, контролируя свою сеть через браузер. Вы даже можете выбросить отдельных пользователей из беспроводной сети, не вставая с шезлонга. Уйди, противный!

Если вы развертываете беспроводную сеть в больших масштабах, вам, вероятно, придется приобрести анализатор беспроводной сети. Мы настоятельно рекомендуем аналитические продукты, разработанные компанией AirMagnet.

Безопасность беспроводных сетей

Традиционно безопасность беспроводных сетей очень низкая. Существует протокол WEP (Wired Equivalent Privacy), применяемый в сетях 802.11b и для шифрования пакетов, передаваемых с помощью радиоволн. К сожалению, в современной версии стандарта была обнаружена фатальная проектная недоработка, которая делает его практически бесполезным. Посторонний человек, находящийся за пределами здания, может получить прямой доступ к сети и остаться незамеченным.

Тем не менее недавно появившиеся стандарты Wi-Fi Protected Access (WPA) возродили доверие к безопасности беспроводных сетей. В настоящее время во всех новых инсталляциях должны использоваться стандарты WPA (в частности, стандарт WPA2), а не WEP. Без применения стандарта WPA2 беспроводные сети должны считаться полностью незащищенными и не должны использоваться за пределами предприятия. Даже дома не используйте стандарт WEP!

Для того чтобы запомнить, что стандарт WEP является незащищенным, а стандарт WPA — безопасным, просто расшифруйте аббревиатуру WAP (Wired Equivalent Privacy — конфиденциальность на уровне проводных сетей). Это название точно отражает суть дела; протокол WEP обеспечивает такую защиту, как проводная сеть, допускающая

непосредственное подключение посторонних лиц. (Иначе говоря, никакой защиты — по крайне мере на уровне IP.)

14.3. SDN: программно-коммутируемые сети

Как и при виртуализации серверов, разделение физического сетевого оборудования с функциональной архитектурой сети может значительно повысить гибкость и управляемость. Лучшим средством для достижения этих целей являются программно-коммутируемые сети (software-defined networking — SDN).

Основная идея SDN заключается в том, что компоненты, управляющие сетью (плоскость управления), физически отделены от компонентов, которые пересылают пакеты (плоскость данных). Плоскость данных программируется через плоскость управления, поэтому вы можете настраивать или динамически изменять маршруты передачи данных для достижения целей производительности, безопасности и доступности.

Как и многое в нашей отрасли, SDN для корпоративных сетей превратилась в маркетинговый трюк. Первоначальная цель состояла в том, чтобы стандартизировать независимые от поставщика способы перенастройки сетевых компонентов. Несмотря на то что некоторые из этих планов были реализованы, многие поставщики теперь предлагают собственные продукты SDN для предприятий, которые в какой-то мере степени противоречат первоначальной цели SDN. Если вы изучаете пространство предприятия SDN, выбирайте продукты, соответствующие открытым стандартам и совместимые с продуктами других поставщиков.

Для крупных поставщиков облачных вычислений SDN добавляет уровень гибкости, который уменьшает вашу потребность знать (или заботиться) о том, где определенный ресурс находится физически. Хотя эти решения могут быть коммерческими, они тесно интегрированы в платформы облачных провайдеров и могут упростить настройку вашей виртуальной инфраструктуры.

Сеть SDN и ее система управления, основанная на интерфейсах API, предлагают системным администраторам соблазнительную возможность интегрировать управление топологией сети с другими инструментами стиля DevOps для непрерывной интеграции и развертывания. Возможно, в каком-то идеальном мире у вас всегда есть производственная среда, поставленная и готовая к активации одним щелчком мыши. По мере того как новая среда продвигается к производству, сетевая инфраструктура магическим образом преобразуется, устраняя простои, заметные для пользователя, и необходимость планировать окна обслуживания.

14.4. Тестирование и отладка сетей

Ключ к отладке сети — ее разбивка на сегменты и тестирование каждого из них до тех пор, пока не будет обнаружена неисправность. Загадочные лампочки на коммутаторах и концентраторах (обозначающие, к примеру, состояние канала и наличие трафика пакетов) помогают быстро выявить источник проблемы. Для того чтобы эти индикаторы работали так, как вы хотите, следует руководствоваться первоклассной документацией.

Как всегда, важно иметь под рукой нужные инструменты, чтобы выполнить работу правильно и без проволочек. На рынке предлагаются средства сетевой отладки двух типов (правда, наблюдается тенденция к их объединению).

Устройство первого типа — ручной кабельный тестер. Он измеряет электрические характеристики кабеля, включая его длину (для этого применяется особая технология,

называемая рефлектометрией во временной области). Такие устройства способны выявлять простейшие проблемы, например разрыв или неправильную разводку кабеля.

Нашим любимым инструментом тестирования локальных сетей является устройство Fluke LanMeter. Это универсальный анализатор, способный даже посылать эхо-пакеты протокола ICMP. Профессиональные варианты этого оборудования описаны на специальном веб-сайте. Для телекоммуникационных сетей WAN лучше всего подходит тестер T-BERD, выпускаемый компанией Viavi (`viavisolutions.com`).

Средства отладки второго типа — это анализаторы сетевых пакетов. Они просматривают сетевые пакеты на предмет наличия ошибок протоколов, неправильной конфигурации и прочего беспорядка. Эти анализаторы работают на уровне каналов, а не на электрическом уровне, поэтому они не могут распознавать проблемы, связанные с физическими повреждениями кабелей или электропитанием.

Существуют профессиональные анализаторы сетевых пакетов, но мы нашли свободно распространяемую программу Wireshark[7] (`wireshark.org`), которая может выполняться на полнофункциональном ноутбуке. Именно ее можно считать наилучшим выбором. Более подробная информация об анализаторах сетевых пакетов приведена в разделе 13.12.

14.5. Прокладка кабелей

Если вы занялись прокладкой кабелей в здании, то самый ценный совет, который мы можем вам дать, звучит так: "Делайте все правильно с первого раза". Это не та область, в которой можно скупиться или халтурить. Покупая качественные материалы, выбирая компетентного подрядчика для прокладки кабелей и устанавливая дополнительные разъемы (отводы), вы тем самым избежите многолетних мучений.

Неэкранированная витая пара

Кабель категории 6a имеет наилучшее соотношение цены и производительности на современном рынке. Его стандартный вариант — четыре пары проводов под одной оболочкой, что подходит для большинства соединений, включая RS-232 и гигабитные линии.

Спецификации кабеля категории 6a требуют, чтобы скрутка провода заканчивалась в точке контакта. Для того чтобы обеспечить это требование, необходимы специальное обучение и оконечное оборудование. При этом необходимо использовать настенные розетки и коммутационные панели категории 6a. Самые хорошие отзывы заслужила продукция компании Siemon.

Офисные точки подключения

Многие годы идут споры, сколько точек подключения требуется для офиса. Одной точки подключения на офис явно недостаточно. Сколько же нужно — две или четыре? Мы рекомендуем четыре, обосновывая это следующими причинами.

- Их можно использовать просто для подключения телефонов и других специализированных устройств.
- Большинство пользователей предпочитают подключаться с помощью беспроводных сетей, а не проводов.
- Гораздо дешевле проложить весь кабель сразу, чем делать это поэтапно.

[7] Как и многие популярные программы, программа Wireshark часто подвергается атакам хакеров. Убедитесь, что вы используете самую последнюю ее версию.

- Средства, выделенные на приобретение проводов, лучше потратить на основную инфраструктуру, а не на оборудование отдельных офисов.

При прокладке кабеля в здании можно установить дополнительные розетки в коридорах, конференц-залах, столовых, туалетных комнатах и на потолках (для точек беспроводного доступа). Однако не забывайте о безопасности и размещайте открыто предоставляемые порты на "гостевой" виртуальной локальной сети, не допуская посторонних к своим внутренним сетевым ресурсам. Публикуя защищенные публичные порты, используйте стандарт аутентификации 802.1x.

Стандарты кабельных систем

Необходимость обеспечения всех видов деятельности внутри современных зданий обусловливает потребность в крупной и сложной кабельной инфраструктуре. Заглянув в обычный коммутационный шкаф, вы будете потрясены, увидев его стенки, сплошь покрытые непомеченными проводами одного цвета.

С целью улучшения оперативного контроля и стандартизации кабельных систем зданий в феврале 1993 г. организация TIA опубликовала административный стандарт на телекоммуникационную инфраструктуру коммерческих зданий (TIA/EIA-606). В 2012 г. появилась его обновленная версия TIA/EIA-606-B.

Этот стандарт устанавливает требования и принципы идентификации и документирования телекоммуникационной инфраструктуры. Он касается следующих аспектов:

- оконечной аппаратуры;
- кабелей;
- прокладки кабелей;
- расстояний между элементами оборудования;
- цветовой маркировки;
- символических обозначений стандартных компонентов.

В частности, определены стандартные цвета маркировки проводов (табл. 14.5).

Таблица 14.5. Таблица цветовой маркировки по стандарту TIA/EIA-606

Тип оконечного устройства	Цвет	Код[а]	Комментарии
Граничное	Оранжевый	150C	Центральная телефонная станция
Сетевые соединения	Зеленый	353C	Также применяется для вспомогательных электросетей
Общее оборудование[б]	Фиолетовый	264C	Основное оборудование коммутации и передачи данных
Магистраль первого уровня	Белый	—	Кабели
Магистраль второго уровня	Серый	422C	Кабели
Станция	Синий	291C	Горизонтальные кабели
Магистраль между зданиями	Коричневый	465C	Кампусные кабели
Разное	Желтый	101C	Служебные и сигнальные линии
Ключевые телефонные системы	Красный	184C	—

[а]В соответствии с цветовой моделью Pantone.

[б]Офисные АТС, компьютеры, локальные сети, мультиплексоры и т.д.

14.6. Проектирование сетей

В этом разделе рассматриваются вопросы, связанные с логическим и физическим проектированием сетей среднего размера. Представленные здесь идеи подходят для нескольких сотен хостов, но неприменимы ни для трех, ни для нескольких тысяч компьютеров, включенных в одну сеть. Также предполагается, что работа будет начата с нуля.

Основной объем работ по проектированию сети состоит из определения:

- типов сред передачи;
- топологии и способов прокладки кабелей;
- системы концентраторов, коммутаторов и маршрутизаторов.

Еще один ключевой вопрос проектирования сети связан с управлением перегрузкой. Например, файловые протоколы NFS и SMB очень сильно загружают сеть, поэтому такие файловые системы нежелательно подключать по магистральному кабелю.

Ниже анализируются аспекты, которые необходимо учитывать при проектировании сети.

Структура сети и архитектура здания

Структуру сети проще изменить, чем архитектуру здания, но обе они должны нормально сосуществовать. Если вам крупно повезло, т.е. представилась возможность проектировать сеть до постройки здания, будьте щедрым. К сожалению, в большинстве случаев здание и отдел технического обслуживания компании на момент проектирования сети уже существуют и налагают жесткие ограничения на структуру сети.

В уже построенных зданиях сеть должна адаптироваться к архитектуре, а не противостоять ей. В современных зданиях, помимо высоковольтной электропроводки, водо- и газопроводов, иногда имеются каналы для прокладки кабелей. Часто монтируются подвесные потолки — настоящий подарок для тех, кто прокладывает сеть. Во многих университетских городках существуют туннели, которые облегчают создание сетей.

Необходимо следить за целостностью брандмауэров.[8] При прокладке кабеля через брандмауэр отверстие должно соответствовать диаметру кабеля и заполняться негорючим веществом. Выбирая кабель, учитывайте наличие приточной вентиляции. Если узнают, что вы нарушили правила пожарной безопасности, вас могут оштрафовать и заставить устранить недостатки, даже если для этого придется проложить заново всю сеть.

Логическая структура сети должна соответствовать физическим ограничениям зданий, в которых она будет функционировать. Приступая к проектированию, помните, что можно найти логически красивое решение, а затем вдруг обнаружить, что реализовать его физически сложно или вообще невозможно.

Расширение сетей

Прогнозировать потребности на десять лет вперед очень сложно, особенно в области вычислительной техники и сетей. Поэтому, проектируя сеть, всегда следует учитывать перспективы ее расширения и увеличения пропускной способности. Прокладывая кабель, особенно в труднодоступных местах, протягивайте в три-четыре раза больше пар,

[8] Речь идет о брандмауэрах в виде бетонных, кирпичных или огнеупорных стен, которые препятствуют распространению огня по всему зданию. Значительно отличаясь от сетевых брандмауэров, они не менее важны.

чем нужно. Помните: основная часть стоимости прокладки сети приходится на оплату труда, а не на материалы.

Даже если волоконно-оптические линии не планируется использовать немедленно, разумно будет все же проложить немного оптического волокна, особенно если известно, что впоследствии протянуть его будет гораздо труднее. Прокладывайте и многомодовый, и одномодовый кабели. Как правило, нужным оказывается как раз тот кабель, который не проложен.

Перегрузка

Сеть — как цепь: ее качество определяется самым слабым или самым медленным звеном. Производительность Ethernet, как и многих других сетевых технологий, при увеличении нагрузки падает.

Активно эксплуатируемые коммутаторы, нестыкующиеся интерфейсы, низкоскоростные каналы связи — все это может привести к перегрузке. Эффективный способ борьбы с ней заключается в локализации трафика путем создания подсетей и установки маршрутизаторов. Подсети можно использовать и для изоляции компьютеров, задействованных в отдельных экспериментах. Трудно проводить эксперимент на нескольких компьютерах, если нет надежного способа изолировать их физически и логически от остальной части сети.

Обслуживание и документирование

Опыт показывает, что удобство обслуживания сети напрямую зависит от качества документации на нее. Точная, полная, своевременно корректируемая документация абсолютно необходима.

Кабели следует маркировать во всех точках подключения. Рекомендуем вкладывать копии местных монтажных схем в коммутационные шкафы, чтобы при всех изменениях эти экземпляры можно было скорректировать на месте. Каждые несколько недель необходимо переносить все корректировки в электронную базу данных.

Стыки между крупными системами в виде коммутаторов или маршрутизаторов могут упростить отладку, поскольку позволяют изолировать части сети и отлаживать их по отдельности. Полезно также разграничивать административные области.

14.7. Управление сетью

Если необходимо обеспечить нормальную работу сети, одни функции управления следует централизовать, другие — распределить, а третьи — оставить на локальном уровне. Требуется сформулировать и согласовать обоснованные "правила поведения добропорядочных граждан".

Типичная крупномасштабная среда включает в себя:

- магистральную сеть, соединяющую здания;
- сети подразделений, подключенные к магистрали;
- подсети рабочих групп в рамках подразделения;
- соединения с внешним миром (например, с Интернетом или периферийными филиалами).

При проектировании и реализации сетей следует предусматривать централизованные контроль, ответственность, сопровождение и финансирование. Поскольку подразделения, как правило, стремятся свести к минимуму собственные расходы, быстро растет число сетей с централизованной оплатой каждого соединения. Вот основные объекты централизованного управления:

- структура сети, в том числе принципы использования подсетей, маршрутизаторов, коммутаторов и т.д.;
- магистральный кабель, в том числе подключения к нему;
- IP-адреса и имена компьютеров, доменные имена;
- используемые протоколы (требуется обеспечить их взаимодействие);
- правила доступа в Интернет.

Имена доменов, IP-адреса и сетевые имена компьютеров в определенном смысле уже находятся под централизованным контролем таких организаций, как ARIN (American Registry for Internet Numbers) и ICANN, но координация использования этих элементов на локальном уровне также необходима.

Центральный орган управления имеет общее представление о сети, ее структуре, производительности и перспективах роста. Он может позволить себе иметь собственное контрольное оборудование (и обслуживающий его персонал) и следить за нормальной работой магистральной сети. Центральный орган может настоять на правильном выборе структуры сети, даже если для этого придется заставить подразделение купить маршрутизатор и создать подсеть для подключения к магистрали. Такое решение иногда необходимо для того, чтобы новое соединение не навредило работе существующей сети.

Если в сети работают разнородные компьютеры, операционные системы и протоколы, обязательно нужно иметь "высокоинтеллектуальный" маршрутизатор (например, компании Cisco), который будет служить шлюзом между сетями.

14.8. Рекомендуемые поставщики

Занимаясь более 30 лет инсталляцией сетей по всему миру, мы не раз обжигались на продуктах, которые не соответствовали спецификациям, имели завышенную цену, неправильно указанные характеристики или как-то иначе не оправдывали ожидания. Ниже приведен список поставщиков, которым мы доверяем и услугами которых рекомендуем пользоваться.

Кабели и разъемные соединения

AMP (подразделение Tyco) (800) 522-6752 amp.com	Anixter (800) 264-9837 anixter.com	Black Box Corporation (724)746-5500 blackbox.com
Belden Cable (800) 235-3361 (765) 983-5200 belden.com	Siemon Company (860) 945-4395 siemon.com	Newark Electronics (800) 463-9275 newark.com

Тестовые приборы

Fluke	Siemon	Viavi
(800) 443-5853	(800) 945-4395	(844) 468-4284
fluke.com	siemon.com	vivasolutions.com

Маршрутизаторы/коммутаторы

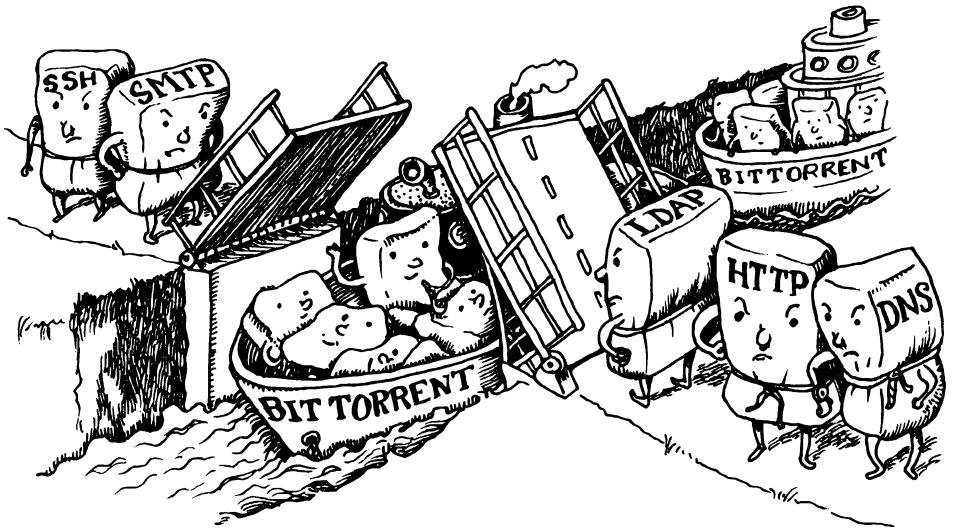
Cisco Systems	Juniper Network
(415) 326-1941	(408) 745-2000
www.cisco.com	juniper.com

14.9. Литература

- ANSI/TIA/EIA-568-A. *Commercial Building Telecommunications Cabling Standard* и ANSI/TIA/EIA-606, *Administration Standard for the Telecommunications Infrastructure of Commercial Buildings*. Это стандарты телекоммуникационной промышленности для построения кабельных систем зданий. К сожалению, они не бесплатны. Посетите веб-сайт www.tiaonline.org.
- Barnett David, Groth David and Jim McBee. *Cabling: The Complete Guide to Network Wiring (3rd* edition). San Francisco: Sybex, 2004.
- Goransson, Paul, and Chuck Black. *Software Defined Networks, A Comprehensive Approach* (2nd Edition). Burlington, MA: Morgan Kaufman, 2016.
- Spurgeon, Charles, and Joann Zimmerman. *Ethernet: The Definitive Guide: Designing and Managing Local Area Networks (2nd Edition)*. Sebastopol, CA: O'Reilly, 2014.

Глава 15

IP-маршрутизация



Во всем мире доступны более 4,3 миллиарда IP-адресов, поэтому получение пакетов в нужном месте в Интернете — непростая задача.[1] Маршрутизация IP-пакетов уже была кратко представлена в главе 13. В этой главе мы рассмотрим процесс маршрутизации более подробно и исследуем несколько сетевых протоколов, которые позволяют маршрутизаторам автоматически обнаруживать эффективные маршруты. Протоколы маршрутизации не только уменьшают административную нагрузку, связанную с обработкой информации о маршрутизации, но также позволяют быстро перенаправить сетевой трафик, если маршрутизатор, соединение или сеть не вышли из строя.

Важно отличать реальное перенаправление IP-пакетов от управления таблицей маршрутизации, хотя в обоих случаях употребляют один и тот же термин — *маршрутизация*. Перенаправление пакетов — простая процедура, тогда как вычисление маршрутов происходит по довольно запутанному алгоритму. Мы познакомимся только с одноадресной маршрутизацией, так как при многоадресной (когда пакеты направляются группе подписчиков) возникает ряд новых проблем, рассмотреть которые, учитывая объем книги, не представляется возможным.

Для того чтобы составить правильное представление о маршрутизации, в большинстве случаев достаточно информации, изложенной в главе 13. Если сетевая инфраструктура уже налажена, можно просто задать единственный статический маршрут (как описано в разделе "Маршрутизация") и все готово — у вас есть информация обо всем Интернете. Если же вы стремитесь справиться с сетью, имеющей сложную топологию, или используете операционные системы UNIX либо Linux как часть своей сетевой ин-

[1] См. `wapo.st/world-ip`.

фраструктуры, тогда эта глава о протоколах и инструментах динамической маршрутизации окажется полезной для вас.[2]

IP-маршрутизация (как для IPv4, так и для IPv6) — маршрутизация "следующего перехода". В любой момент времени системе, обрабатывающей пакет, необходимо определить только следующий хост или маршрутизатор в пути пакета до конечного пункта назначения. Это отличается от подхода многих устаревших протоколов, которые определяют точный путь, по которому пакет будет перемещаться, прежде чем покинет свой исходный хост, схему, известную как маршрутизация от источника.[3]

15.1. Подробнее о маршрутизации пакетов

Прежде чем приступать к изучению алгоритмов маршрутизации, рассмотрим подробнее, как используются маршрутные таблицы. Предположим, сеть имеет топологию, изображенную на рис. 15.1.

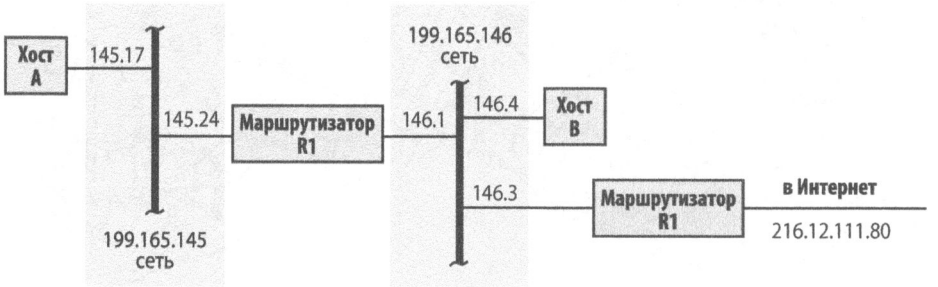


Рис. 15.1. Демонстрационная сеть

Для простоты мы начинаем этот пример с протокола IPv4; таблица маршрутизации IPv6 приведена ниже.

Маршрутизатор R1 соединяет две сети, а маршрутизатор R2 — одну из этих сетей с внешним миром. Посмотрим, как выглядят таблицы маршрутизации и некоторые конкретные сценарии перенаправления пакетов. Таблица хоста A выглядит следующим образом.

```
A$ netstat -rn
Destination     Gateway           Genmask         Flags  MSS Window irtt Iface
127.0.0.0       0.0.0.0           255.0.0.0       U      0   0      0    lo
199.165.145.0   0.0.0.0           255.255.255.0   U      0   0      0    eth0
0.0.0.0         199.165.145.24    0.0.0.0         UG     0   0      0    eth0
```

В приведенном выше примере для запроса таблицы маршрутизации используется хорошо известный инструмент **netstat**. Этот инструмент распространяется с операционной системой FreeBSD и доступен для Linux как часть пакета **net-tools**. Этот пакет больше не имеет активной поддержки и, как результат, считается устаревшим. Официально рекомендованным способом получить эту информацию в Linux является менее функциональная команда **ip route**:

[2]Мы не рекомендуем использовать системы UNIX или Linux в качестве сетевых маршрутизаторов в производственной инфраструктуре — лучше купите выделенный маршрутизатор.

[3]IP-пакеты также могут маршрутизироваться на основе адреса отправителя — по крайней мере теоретически, но это почти никогда не выполняется. Эта функция не получила широкого распространения из соображений безопасности.

```
A$ ip route
default via 199.165.145.24 dev eth0 onlink
199.165.145.0/24 dev eth0 proto kernel scope link src 199.165.145.17
```

Результат работы команды **netstat -rn** немного легче читать, поэтому мы используем его для последующих примеров и исследования рис. 15.1.

Хост A имеет самую простую конфигурацию среди всех четырех компьютеров. Первые два маршрута описывают собственные сетевые интерфейсы хоста. Они необходимы, чтобы пакеты, направляемые непосредственно в подключенные сети, не маршрутизировались особым образом. Устройство `eth0` — это Ethernet-интерфейс хоста A, а `lo` — интерфейс обратной связи (виртуальный сетевой интерфейс, эмулируемый программным обеспечением). Обычно такие записи автоматически добавляются при конфигурировании сетевого интерфейса.

🕮 См. обсуждение сетевых масок в разделе 13.4.

Стандартный маршрут хоста A задает перенаправление всех пакетов, не адресованных интерфейсу обратной связи или сети 195.165.145, на маршрутизатор R1, адрес которого в данной сети — 199.165.145.24. Шлюзы должны находиться на расстоянии одного перехода.

🕮 Дополнительную информацию об адресах см. в разделе 13.3.

Предположим теперь, что хост A посылает пакет хосту B с адресом 199.165.146.4. IP-подсистема ищет маршрут к сети 199.165.146 и, не найдя такового, отправляет пакет по стандартному маршруту, т.е. маршрутизатору R1. На рис. 15.2 приведена структура пакета, проходящего по сети Ethernet (в заголовке Ethernet указаны MAC-адреса соответствующих интерфейсов компьютеров A и R1 в сети 145).

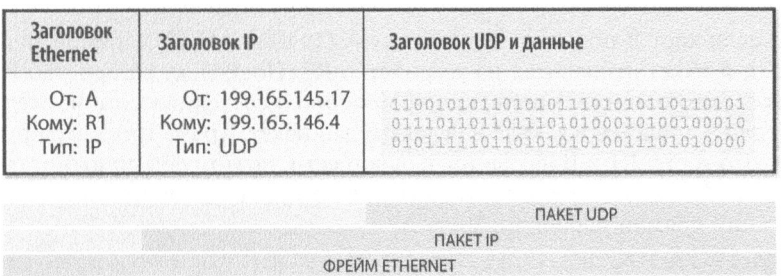


Рис. 15.2. Ethernet-пакет

Аппаратный Ethernet-адрес получателя соответствует маршрутизатору R1, но IP-пакет, скрытый в Ethernet-фрейме, не содержит никаких упоминаний о маршрутизаторе. Когда маршрутизатор просматривает поступивший пакет, он обнаруживает, что не является его получателем. Тогда он обращается к собственной таблице маршрутизации, чтобы узнать, как переслать пакет хосту B, не переписывая его IP-заголовок (необходимо, чтобы отправителем пакета оставался хост A).

Таблица маршрутизации устройства R1 выглядит следующим образом.

```
R1$ netstat -rn
Destination     Gateway         Genmask         Flags  MSS  Window irtt Iface
127.0.0.0       0.0.0.0         255.0.0.0       U      0    0      0    lo
199.165.145.0   0.0.0.0         255.255.255.0   U      0    0      0    eth0
199.165.146.0   0.0.0.0         255.255.255.0   U      0    0      0    eth1
0.0.0.0         199.165.146.3   0.0.0.0         UG     0    0      0    eth1
```

Она почти аналогична таблице хоста A, за исключением того, что здесь присутствует два физических сетевых интерфейса. Стандартный маршрут в данном случае ведет к устройству R2, поскольку через него осуществляется выход в Интернет. Пакеты, адресованные любой из сетей 199.165, доставляются напрямую.

Подобно хосту A, хост B имеет один реальный сетевой интерфейс. Но для корректного функционирования ему необходим дополнительный маршрут, поскольку хост имеет прямое соединение сразу с двумя маршрутизаторами. Трафик сети 199.165.145 должен проходить через устройство R1, а весь остальной трафик — направляться в Интернет через устройство R2.

```
B$ netstat -rn
Destination     Gateway         Genmask         Flags  MSS  Window irtt Iface
127.0.0.0       0.0.0.0         255.0.0.0       U      0    0      0    lo
199.165.145.0   199.165.146.1   255.255.255.0   U      0    0      0    eth0
199.165.146.0   0.0.0.0         255.255.255.0   U      0    0      0    eth0
0.0.0.0         199.165.146.3   0.0.0.0         UG     0    0      0    eth0
```

Описание переадресации ICMP см. в разделе 13.5.

Теоретически, как и хост A, хост B можно сконфигурировать так, чтобы изначально он "знал" только об одном шлюзе, полагаясь на помощь директив переадресации протокола ICMP для устранения избыточных переходов. Ниже показан пример начальной конфигурации.

```
B$ netstat -rn
Destination     Gateway         Genmask         Flags  MSS Window irtt Iface
127.0.0.0       0.0.0.0         255.0.0.0       U      0   0      0    lo
199.165.146.0   0.0.0.0         255.255.255.0   U      0   0      0    eth0
0.0.0.0         199.165.146.3   0.0.0.0         UG     0   0      0    eth0
```

Теперь, если хост B посылает пакет хосту A (199.165.145.17), прямой маршрут найден не будет, и пакет отправится на устройство R2. Поскольку устройство R2 является маршрутизатором, оно имеет полную информацию о состоянии сети и, следовательно, "знает" о роли устройства R1, куда и будет послан пакет. Кроме того, маршрутизатор R2 обнаружит, что хосты R1 и B находятся в одной сети, поэтому он дополнительно направит хосту B ICMP-сообщение, в соответствии с которым хост B добавит в свою таблицу маршрутизации прямой маршрут к хосту A.

```
199.165.145.17 199.165.146.1 255.255.255.255 UGHD    0   0      0    eth0
```

Благодаря этому маршруту весь трафик, адресованный хосту A, начнет идти непосредственно через маршрутизатор R1. Однако это изменение не затрагивает трафик к другим хостам в сети 145. Для них нужно получить отдельные директивы от маршрутизатора R2.

Некоторые администраторы выбирают для своих систем директивы переадресации протокола ICMP в качестве основного "протокола" маршрутизации, думая, что подобный подход обеспечит большую динамичность. К сожалению, системы и маршрутизаторы осуществляют перенаправление по-разному. Некоторые хранят эти директивы постоянно. Другие удаляют их из таблицы маршрутизации через относительно короткое время (5–15 минут). Третьи вообще игнорируют их, что, вероятно, правильно с точки зрения безопасности.

Перенаправления имеют несколько других потенциальных недостатков: например, увеличение нагрузки на сеть, увеличение нагрузки на R2, беспорядок в таблице маршрутизации и зависимость от дополнительных серверов, поэтому мы не рекомендуем их

использовать. В правильно настроенной сети перенаправления никогда не должны появляться в таблице маршрутизации.

Если вы используете адреса IPv6, применяется та же модель. Вот как выглядит таблица маршрутизации с хоста, работающего под управлением операционной системы FreeBSD, на котором запущен протокол IPv6:

```
$ netstat -rn
Destination           Gateway              Flags  Netif  Expire
default               2001:886b:4452::1    UGS    re0
2001:886b:4452::/64   link#1               U      re0
fe80::/10             ::1                  UGRS   lo0
fe80::%re0/64         link#1               U      re0
```

Как и в протоколе IPv4, первый маршрут является значением по умолчанию, которое используется, когда нет более конкретных записей. Следующая строка содержит маршрут к глобальной сети IPv6, где находится хост, `2001:886b:4452::/64`. Последние две строки являются особенными; они представляют маршрут к зарезервированной сети IPv6 `fe80`, известной как локальная сеть одноадресатной передачи. Эта сеть используется для трафика, который привязан к локальному широковещательному домену (как правило, к одному и тому же сегменту физической сети). Он чаще всего используется сетевыми службами, которые должны найти друг друга в одноадресатной сети, например OSPF. Не используйте подобные локальные адреса для обычных сетевых целей.

15.2. Демоны и протоколы маршрутизации

В простых сетях, подобно той, которая представлена на рис. 15.1, маршрутизацию целесообразно настраивать вручную. Однако с определенного момента сети становятся слишком сложными для подобного администрирования. Вместо того чтобы явно сообщать каждому компьютеру, как находить другие компьютеры и сети, лучше заставить сами компьютеры искать эту информацию. Данная задача возлагается на протоколы маршрутизации и демоны, которые их реализуют.

Основное преимущество протоколов маршрутизации над системами статической маршрутизации заключается в том, что они позволяют быстро адаптироваться к изменениям в топологии сети. Когда пропадает канал связи, демоны быстро находят альтернативные маршруты, если они существуют, и сообщают о них в сети, связанные с этим каналом.

Демоны маршрутизации собирают информацию из трех источников: конфигурационных файлов, существующих таблиц маршрутизации и "родственных" демонов других систем. Собранные данные объединяются, и вычисляется оптимальный набор маршрутов, после чего новые маршруты записываются в системную таблицу (и при необходимости посылаются другим системам посредством протоколов маршрутизации). Состояние сети время от времени меняется, поэтому демоны должны периодически опрашивать друг друга, чтобы убедиться в актуальности имеющейся у них информации.

Конкретный алгоритм вычисления маршрутов зависит от протоколов. Последние бывают двух типов: дистанционно-векторные и топологические.

Дистанционно-векторные протоколы

В основе дистанционно-векторных протоколов лежит следующая идея: если маршрутизатор X находится в пяти переходах от сети Y и является моим соседом, то я нахожусь в шести переходах от данной сети. Демон, работающий по такому протоколу, объявляет о том,

как далеко, по его расчетам, расположены известные ему сети. Если соседние демоны не "знают" более коротких маршрутов к этим сетям, они помечают данный компьютер как оптимальный шлюз. В противном случае они просто игнорируют этот анонс. Предполагается, что со временем таблицы маршрутизации придут в стабильное состояние.

Это действительно элегантная идея, и, если бы все работало так, как задумано, маршрутизация существенно упростилась бы. К сожалению, описанный алгоритм не лучшим образом справляется с изменениями топологии.[4] Иногда стабилизация таблиц вообще не наступает вследствие возникновения бесконечных циклов (например, маршрутизатор X получает информацию от маршрутизатора Y и посылает ее маршрутизатору Z, который возвращает ее маршрутизатору Y). На практике приходится вводить сложные эвристические правила или задавать ограничения. К примеру, в протоколе RIP (Routing Information Protocol — протокол маршрутной информации) считается, что любая сеть, находящаяся на расстоянии более пятнадцати переходов, недоступна.

Даже в обычных ситуациях может потребоваться слишком много циклов обновлений, прежде чем все маршрутизаторы перейдут в стабильное состояние. Следовательно, чтобы не превысить допустимые пределы, необходимо сделать периоды обновлений короткими, а это, в свою очередь, ведет к тому, что дистанционно-векторные протоколы оказываются слишком "словоохотливыми". Например, протокол RIP требует, чтобы маршрутизаторы осуществляли широковещательную рассылку всей имеющейся у них информации каждые 30 с. В протоколе EIGRP обновления анонсируются каждые 90 с.

В противоположность этому в протоколе BGP (Border Gateway Protocol — протокол граничного шлюза) вся таблица посылается один раз, после чего изменения распространяются по мере возникновения. Такая оптимизация позволяет существенно снизить "переговорный" трафик (большей частью ненужный).

В табл. 15.1 перечислены дистанционно-векторные протоколы, широко используемые в настоящее время.

Таблица 15.1. Распространенные дистанционно-векторные протоколы маршрутизации

Аббревиатура	Полное название	Область применения
RIP	Routing Information Protocol (протокол маршрутной информации)	Локальные сети
RIPng	Routing Information Protocol (протокол маршрутной информации), следующее поколение	Локальные сети с протоколом IPv6
EIGRP [a]	Enhanced Interior Gateway Routing Protocol (улучшенный протокол маршрутизации внутреннего шлюза)	Глобальные сети, корпоративные локальные сети
BGP	Border Gateway Protocol (протокол граничного шлюза)	Магистральные сети Интернета

[a] Протокол EIGRP является собственностью компании Cisco.

Топологические протоколы

В топологических протоколах, или *протоколах состояния канала*, информация рассылается в относительно необработанном виде. Записи выглядят примерно так: "Маршрутизатор X является смежным по отношению к маршрутизатору Y, и канал функционирует". Полный набор таких записей образует карту сетевых связей, на ос-

[4] Проблема заключается в том, что изменения топологии могут приводить к удлинению существующих маршрутов. Некоторые дистанционно-векторные протоколы, например EIGRP, хранят информацию о всех возможных маршрутах, чтобы на крайний случай был "план отступления". Впрочем, конкретные детали не так уж важны.

новании которой каждый маршрутизатор может сформировать собственную таблицу. Основное преимущество топологических протоколов, по сравнению с дистанционно-векторными, заключается в способности быстро стабилизировать таблицы маршрутизации в случае непредвиденного сбоя. К издержкам относится необходимость хранения полной карты соединений на каждом хосте, для чего требуются дополнительные процессорные мощности и память.

Поскольку процесс общения между маршрутизаторами не является частью собственно алгоритма вычисления маршрутов, появляется возможность реализовать топологические протоколы так, чтобы не возникало циклов. Изменения в топологической базе данных распространяются по сети очень быстро, не перегружая ни канал, ни центральный процессор.

Топологические протоколы сложнее дистанционно-векторных, зато они позволяют реализовать такие технологии, как маршрутизация на основании запрашиваемого типа обслуживания (поле TOS IP-пакета) и поддержка нескольких маршрутов к одному адресату.

Единственным топологическим протоколом, получившим широкое распространение, является протокол OSPF.

Метрика стоимости

Для того чтобы протокол маршрутизации мог определить, какой путь к заданной сети является кратчайшим, необходимо вначале выяснить, что значит "кратчайший". Это путь с наименьшим числом переходов? С наименьшей задержкой? С наименьшими финансовыми затратами?

Для целей маршрутизации качество канала связи определяется числом, называемым *метрикой стоимости*. Путем сложения метрик отдельных отрезков пути вычисляется общая стоимость маршрута. В простейших системах каждому каналу назначается стоимость 1, и в результате метрикой маршрута становится число переходов. Но любой из перечисленных выше критериев может являться метрикой стоимости.

Эксперты в области сетей долго и упорно трудились над тем, чтобы определение такого понятия, как метрика стоимости, было максимально гибким, а некоторые современные протоколы даже позволяют использовать разные метрики для разных видов сетевого трафика. Тем не менее в 99% случаев на все это можно не обращать внимания. Для большинства систем вполне подойдут стандартные метрики.

Бывают ситуации, когда кратчайший физический маршрут к адресату не должен быть выбран по умолчанию из административных соображений. В таких случаях можно искусственно завысить метрики критических каналов. Всю остальную работу предоставьте демонам.

Внутренние и внешние протоколы

Автономная система — это группа сетей, находящихся под административным или политическим контролем одного юридического лица. Такое определение довольно расплывчато. Реальные автономные системы могут представлять собой как глобальные корпоративные сети, так и сети университетов или даже отдельных факультетов. Все зависит от того, как осуществляется маршрутизация. Сейчас наблюдается тенденция к укрупнению автономных систем. Это упрощает администрирование и повышает эффективность маршрутизации.

Маршрутизация внутри автономной системы отличается от маршрутизации между такими системами. Протоколы второго типа (внешние, или протоколы внешних шлюзов) должны управлять множеством маршрутов к различным сетям и учитывать тот

факт, что соседние маршрутизаторы находятся под контролем других людей. Внешние протоколы не раскрывают топологию автономной системы, поэтому в определенном смысле их можно рассматривать как второй уровень маршрутизации, на котором соединяются группы сетей, а не отдельные компьютеры или кабели.

На практике небольшим и среднего размера организациям редко требуется внешний протокол, если только они не подключены к нескольким провайдерам одновременно. При наличии нескольких провайдеров традиционное разделение на локальный домен и домен Интернета нарушается, поскольку маршрутизаторам приходится определять, какой маршрут в Интернете лучше всего подходит для конкретного адреса. (Это не означает, что каждый маршрутизатор должен знать всю необходимую информацию. Большинство хостов может направлять свои пакеты внутреннему шлюзу, хранящему необходимые сведения.)

Поскольку внешние протоколы не сильно отличаются от своих внутренних аналогов, в этой главе мы сосредоточим внимание на внутренних протоколах и демонах, которые их поддерживают. Если в вашей сети поддерживается внешний протокол, обратитесь к литературным источниками, ссылки на которые приведены в конце главы.

15.3. Основные протоколы маршрутизации

В этом разделе мы познакомимся с основными внутренними протоколами маршрутизации, узнаем их преимущества и недостатки.

Протоколы RIP и RIPng

RIP (Routing Information Protocol — протокол маршрутной информации) — это старый протокол компании Xerox, адаптированный для IP-сетей. Его IP-версия была описана примерно в 1988 году в документе RFC1058. Существует три версии этого протокола: RIPv1, RIPv2 и RIPng только для протокола IPv6 (ng (**n**ext **g**eneration) означает "следующее поколение").

Все версии этого протокола представляют собой простые дистанционно-векторные протоколы, метрикой стоимости в которых является количество переходов. Поскольку протокол RIP разрабатывался в те времена, когда отдельные компьютеры были дорогими, а сети маленькими, в версии RIPv1 предполагается, что все хосты, находящиеся на расстоянии пятнадцати и более переходов, недоступны. В более поздних версиях это ограничение не было снято, чтобы стимулировать администраторов сложных сетей переходить на более сложные протоколы маршрутизации.

🕮 Информация о бесклассовой адресации, известной под именем CIDR, приведена в разделе 13.4.

Протокол RIPv2 — это улучшенная версия протокола RIP, в которой вместе с адресом следующего перехода передается сетевая маска. Это упрощает управление сетями, где есть подсети и применяется протокол CIDR, по сравнению с протоколом RIPv1. В нем была также предпринята невнятная попытка усилить безопасность протокола RIP.

Протокол RIPv2 можно выполнять в режиме совместимости. Это позволяет сохранить большинство его новых функциональных возможностей, не отказываясь от получателей, использующих простой протокол RIP. Во многих аспектах протокол RIPv2 идентичен исходному протоколу, и отдавать предпочтение следует именно ему.

🕮 Детали протокола IPv6 приведены в разделе 13.2.

Протокол RIPng представляет собой переформулирование протокола RIP в терминах протокола IPv6. Он может использоваться только в рамках протокола IPv6, в то время как протокол RIPv2 — только в рамках протокола IPv4. Если вы хотите использовать как протокол IPv4, так и протокол IPv6 вместе с протоколом RIP, то RIP и RIPng необходимо выполнять как отдельные протоколы.

Несмотря на то что протокол RIP известен своим расточительным использованием широковещательного режима, он весьма эффективен при частых изменениях сети, а также в тех случаях, когда топология удаленных сетей неизвестна. Однако после сбоя канала он может замедлить стабилизацию системы.

Сначала исследователи были уверены, что появление более сложных протоколов маршрутизации, таких как OSPF, сделает протокол RIP устаревшим. Тем не менее протокол RIP продолжает использоваться, потому что он простой, легкий в реализации и не требует сложной настройки. Таким образом, слухи о смерти протокола RIP оказались слишком преувеличенными.

Протокол RIP широко используется на платформах, не использующих операционную систему UNIX. Многие устройства, включая сетевые принтеры и сетевые управляемые SNMP-компоненты, способны принимать RIP-сообщения, узнавая о возможных сетевых шлюзах. Кроме того, почти во всех версиях систем UNIX и Linux в той или иной форме существует клиент протокола RIP. Таким образом, протокол RIP считается "наименьшим общим знаменателем" протоколов маршрутизации. Как правило, он применяется для маршрутизации в пределах локальной сети, тогда как глобальную маршрутизацию осуществляют более мощные протоколы.

На некоторых серверах запускаются пассивные демоны протокола RIP (обычно демон `routed` или `ripd` из пакета Quagga), которые ожидают сообщений об изменениях в сети, не осуществляя широковещательной рассылки собственной информации. Реальные вычисления маршрутов выполняются с помощью более производительных протоколов, таких как OSPF (см. следующий раздел). Протокол RIP используется только как механизм распространения.

Протокол OSPF

OSPF (Open Shortest Path First — открытый протокол обнаружения первого кратчайшего маршрута) является самым популярным топологическим протоколом. Термин *первый кратчайший маршрут* (shortest path first) означает специальный математический алгоритм, по которому вычисляются маршруты; термин *открытый* (open) — синоним слова "непатентованный". Основная версия протокола OSPF (версия 2) определена в документе RFC2328, а расширенная версия протокола OSPF, поддерживающая протокол IPv6 (версия 3), — в документе RFC5340. Первая версия протокола OSPF устарела и сейчас не используется.

OSPF — протокол промышленного уровня, который эффективно функционирует в крупных сетях со сложной топологией. По сравнению с протоколом RIP, он имеет ряд преимуществ, включая возможность управления несколькими маршрутами, ведущими к одному адресату, и возможность разделения сети на сегменты ("области"), которые будут делиться друг с другом только высокоуровневыми данными маршрутизации. Сам протокол очень сложный, поэтому имеет смысл использовать его только в крупных системах, где важна эффективность маршрутизации. Для эффективного использования протокола OSPF необходимо, чтобы ваша схема адресации имела иерархический характер.

В спецификации протокола OSPF не навязывается конкретная метрика стоимости. По умолчанию в реализации этого протокола компанией Cisco в качестве метрики используется пропускная способность сети.

Протокол EIGRP

EIGRP (Enhanced Interior Gateway Routing Protocol — улучшенный протокол маршрутизации внутреннего шлюза) — это патентованный протокол маршрутизации, используемый только маршрутизаторами компании Cisco. Протокол IGRP был разработан для устранения некоторых недостатков протокола RIP еще в те времена, когда не было такого надежного стандарта, как протокол OSPF. Протокол IGRP был отклонен в пользу протокола EIGRP, который допускает произвольные сетевые маски CIDR. Протоколы IGRP и EIGPR конфигурируются одинаково, несмотря на различия в их организации.

Протокол EIGRP поддерживает протокол IPv6, но в нем, как и во всех других протоколах маршрутизации, адресные пространства IPv6 и IPv4 конфигурируются отдельно и существуют как параллельные домены маршрутизации.

Протокол EIGRP является дистанционно-векторным, но он спроектирован так, чтобы избежать проблем зацикливания и медленной стабилизации, свойственных другим протоколам данного класса. В этом смысле протокол EIGRP считается образцом. Для большинства применений протоколы EIGRP и OSPF обеспечивают равные функциональные возможности.

BGP: протокол граничного шлюза

Протокол BGP (Border Gateway Protocol — протокол граничного шлюза) является протоколом внешней маршрутизации, т.е. он управляет трафиком между автономными системами, а не между отдельными сетями. Существовало несколько популярных протоколов внешней маршрутизации, но протокол BGP пережил их всех.

В настоящее время BGP является стандартным протоколом, используемым для магистральной маршрутизации в Интернете. В средине 2017 года таблица маршрутизации Интернета содержала около 660 тысяч префиксов. Совершенно очевидно, что это масштаб, при котором магистральная маршрутизация существенно отличается от локальной.

15.4. Многоадресатная координация протокола маршрутизации

Маршрутизаторы должны обмениваться информацией, чтобы узнать, как добраться до мест в сети, но, чтобы добраться до мест в сети, они должны поговорить с маршрутизатором. Эта проблема с курицей и яйцом чаще всего решается посредством многоадресатной связи. Это сетевой эквивалент согласия на встречу с вашим другом на определенном углу улицы, если вы разделитесь. Этот процесс обычно невидим для системных администраторов, но иногда вы можете видеть этот многоадресатный трафик в трассировке пакета или при других видах сетевой отладки.

Согласованные групповые адреса для различных протоколов маршрутизации перечислены в табл. 15.2.

Таблица 15.2. Групповые адреса для протоколов маршрутизации

Описание	IPv6	IPv4
Все системы в подсети	`ff02::1`	`224.0.0.1`
Все маршрутизаторы в подсети	`ff02::2`	`224.0.0.2`
Неназначенные	`ff02::3`	`224.0.0.3`
Маршрутизаторы DVMRP	`ff02::4`	`224.0.0.4`
Маршрутизаторы OSPF	`ff02::5`	`224.0.0.5`
Маршрутизаторы OSPF DR	`ff02::6`	`224.0.0.6`
Маршрутизаторы RIP	`ff02::9`	`224.0.0.9`
Маршрутизаторы EIGRP	`ff02::10`	`224.0.0.10`

15.5. Выбор критериев стратегии маршрутизации

Существует четыре уровня сложности, характеризующих процесс управления маршрутизацией в сети:

- отсутствие маршрутизации как таковой;
- только статическая маршрутизация;
- преимущественно статическая маршрутизация, но клиенты принимают RIP-обновления;
- динамическая маршрутизация.

Общая топология сети существенно влияет на маршрутизацию каждого конкретного сегмента. В различных сетях могут требоваться совершенно разные уровни поддержки маршрутизации. При выборе стратегии следует руководствоваться перечисленными ниже эмпирическими правилами.

- Автономная сеть не нуждается в маршрутизации.
- Если из сети есть только один выход во внешний мир, клиенты сети (нешлюзовые хосты) должны иметь стандартный статический маршрут к этому шлюзу. Никакой другой конфигурации не требуется, кроме как на самом шлюзе.
- Можно задать явный статический маршрут к шлюзу, ведущему в небольшую группу сетей, и стандартный маршрут к шлюзу, ведущему во внешний мир. Однако динамическая маршрутизация предпочтительнее, если до нужной сети можно добраться разными маршрутами.
- Если сети пересекают государственные или административные границы, следует использовать динамическую маршрутизацию, даже если сложность сетей этого не требует.
- Протокол RIP отлично работает и широко используется. Не отказывайтесь от него просто потому, что он имеет репутацию устаревшего протокола.
- Проблема протокола RIP заключается в том, что его невозможно масштабировать до бесконечности. Расширение сети рано или поздно приведет к отказу от него. Из-за этого протокол RIP считается промежуточным протоколом с узкой областью применения. Эта область ограничена с одной стороны сетями, которые являются слишком простыми, чтобы применять в них какой-либо протокол маршрутизации, а с другой стороны — сетями, которые являются слишком сложными

для протокола RIP. Если вы планируете расширять свою сеть, то было бы целесообразно игнорировать протокол RIP вообще.

- Даже если протокол RIP не соответствует вашей стратегии глобальной маршрутизации, он остается хорошим способом распределения маршрутов к концевым хостам. Однако его не следует применять без особой надобности: системы в сети, имеющей только один шлюз, никогда не требуют динамического обновления.
- Протоколы EIGRP и OSPF имеют одинаковые функциональные возможности, но протокол EIGRP является собственностью компании Cisco. Компания Cisco делает прекрасные и оптимальные по соотношению "цена — качество" маршрутизаторы; тем не менее стандартизация протокола EIGRP ограничивает ваши возможности будущего расширения.
- Маршрутизаторы, подключенные к магистрали Интернета, должны использовать протокол BGP. Обычно большинство маршрутизаторов имеет только один вход и, следовательно, для них достаточно задать простой статический стандартный маршрут.
- EIGRP и OSPF примерно одинаково функциональны, но EIGRP является собственностью Cisco. Cisco делает превосходные и конкурентоспособные по стоимости маршрутизаторы; тем не менее стандартизация EIGRP ограничивает ваш выбор для будущего расширения.
- Маршрутизаторы, подключенные к Интернету через несколько провайдеров восходящего потока, должны использовать протокол BGP. Однако большинство маршрутизаторов имеют только один восходящий путь и поэтому могут использовать простой статический маршрут по умолчанию.

Хорошей стратегией по умолчанию для организации среднего размера с относительно стабильной локальной структурой и подключением к чужой сети является использование комбинации статической и динамической маршрутизации. Маршрутизаторы в локальной структуре, которые не приводят к внешним сетям, могут использовать статическую маршрутизацию, пересылку всех неизвестных пакетов на машину по умолчанию, которая понимает внешний мир и выполняет динамическую маршрутизацию.

Сеть, которая слишком сложна для управления с помощью этой схемы, должна опираться на динамическую маршрутизацию. Статические маршруты по умолчанию все еще можно использовать в сетях, состоящих из листьев, но машины в сетях с несколькими маршрутизаторами должны запускаться маршрутизированным или другим RIP-приемником в пассивном режиме.

15.6. Демоны маршрутизации

Мы не рекомендуем использовать системы UNIX и Linux в качестве маршрутизаторов в производственных сетях. Специализированные маршрутизаторы проще, надежнее, безопаснее и более быстродействующие (даже если они скрытно запускают ядро системы Linux). Иначе говоря, очень хорошо иметь возможность организовать новую подсеть, используя всего лишь сетевую карту стоимостью 15 долл. и коммутатор за 40 долл. Это вполне разумный подход к организации разреженных, тестовых и вспомогательных сетей.

Системы, действующие как шлюзы в таких подсетях, не требуют дополнительных инструментов для управления их собственными таблицами маршрутизации. Статические маршруты вполне адекватны как для машин, используемых в качестве шлюзов, так и для машин, представляющих собой хосты самой подсети. Однако если

вы хотите, чтобы ваша подсеть была доступной для других сетей в вашей организации, вам необходимо сообщить о ее существовании и идентифицировать маршрутизатор, к которому будут привязаны пакеты, посылаемые данной подсетью. Обычно для этого на шлюзе запускается демон маршрутизации.

Системы UNIX и Linux могут участвовать в большинстве протоколов маршрутизации с помощью различных демонов маршрутизации. Важным исключением из этого правила является протокол EIGRP, который, насколько нам известно, не имеет широко доступной реализации в системах UNIX и Linux.

Поскольку демоны маршрутизации редко реализуются в производственных системах, мы не будем подробно описывать их использование и конфигурацию. Тем не менее в следующих разделах мы приведем краткое описание типичных программ и укажем, где искать детали конфигурации.

Демон routed: устаревшая реализация в протоколе RIP

Долгое время демон **routed** был стандартным демоном маршрутизации, и его до сих пор включают в дистрибутивы некоторых систем. Демон **routed** понимает только протокол RIP и при этом плохо: даже поддержка версии RIPv2 не поправила ситуацию. Демон **routed** не понимает протокол RIPng, реализация которого основана на современных демонах, таких как Quagga и **ramd** в системе HP-UX.

Демон **routed** целесообразно использовать в пассивном режиме (**-q**), в котором он принимает сообщения об обновлениях маршрутов, но не осуществляет широковещательную рассылку собственных сообщений. Кроме указания опций в командной строке, демон **routed** не требует конфигурирования. Он представляет собой дешевый и простой способ получения сообщений об обновлениях маршрутов без сложного конфигурирования.

Демон заносит обнаруженные маршруты в таблицу ядра. Все маршруты должны анонсироваться, как минимум каждые четыре минуты, иначе они будут удалены. Правда, демон **route** помнит, какие маршруты он добавлял, и не удаляет статические маршруты, заданные с помощью команды **route**.

🕮 См. способы ручного управления таблицами маршрутизации в разделе 13.5.

Пакет Quagga: основной демон маршрутизации

Quagga (`quagga.net`) — это опытно-конструкторское подразделение проекта Zebra, выполняемого в рамках проекта GNU. Проект Zebra был запущен Кунихиро Ишигуро (Kunihiro Ishiguro) и Йошинари Йошикава (Yoshinari Yoshikawa) для реализации многопротокольной маршрутизации с помощью коллекции независимых демонов, а не одного монолитного приложения. В реальной жизни квагга (quagga) — это исчезнувший подвид зебры, которая была последний раз сфотографирована в 1870 году. Но его цифровая реинкарнация Quagga выжила, а проект Zebra закрылся.

В настоящее время пакет Quagga реализует все протоколы RIP (все версии), OSPF (версии 2 и 3) и BGP. Он выполняется в системах Linux, FreeBSD и разных вариантах системы BSD. В системе Solaris и версиях системы Linux, имеющихся в нашем распоряжении, пакет Quagga либо инсталлирован по умолчанию, либо доступен в качестве вспомогательного пакета, который можно загрузить из стандартного системного репозитория программного обеспечения.

В пакете Quagga демон **zebra** играет роль информационного центра для маршрутизации. Он управляет взаимодействием между таблицей маршрутизации ядра и демона-

ми, соответствующими отдельным протоколам маршрутизации (**ripd**, **ripngd**, **ospfd**, **ospf6d** и **bgpd**). Кроме того, он управляет потоками информации о маршрутах, которыми обмениваются протоколы. Каждый демон имеет свой собственный конфигурационный файл в каталоге **/etc/quagga**.

Для того чтобы послать запрос или изменить конфигурацию, можно соединиться с любым из демонов Quagga с помощью интерфейса командной строки (**vtysh** в системе Linux и **quaggaadm** в системе Solaris). Командный язык разработан так, чтобы он был понятен пользователям операционной системы IOS компании Cisco; дополнительные детали можно найти в разделе, посвященном маршрутизаторам Cisco. Как и в системе IOS, для того чтобы войти в режим "суперпользователя", необходимо выполнить команду **enable**, для того чтобы ввести команды конфигурирования, необходимо выполнить команду **config term**, а для того чтобы сохранить изменения конфигурации в файле конфигурации демона, необходимо выполнить команду **write**.

Официальная документация доступна на сайте `quagga.net` в виде файлов в форматах HTML и PDF. Несмотря на свою полноту, эта документация содержит, в основном, удобный каталог опций, а не общий обзор системы. Более практичную документацию можно найти на сайте `wiki.quagga.net`. Здесь находятся подробно прокомментированные примеры файлов конфигурации, ответы на часто задаваемые вопросы и советы.

Несмотря на то что файлы конфигурации имеют простой формат, необходимо знать протокол, конфигурацию которого вы настраиваете, а также понимать, что означают те или иные опции. Список рекомендованной литературы по этому вопросу приведен в конце главы.

Маршрутизатор XORP

Проект XORP (eXtensible Open Router Platform — расширяемая платформа маршрутизации с открытым кодом) был открыт примерно в то же время, что и проект Zebra, но его цели были более универсальными. Вместо маршрутизации проект XORP был нацелен на эмулирование всех функций заданного маршрутизатора, включая фильтрацию пакетов и управление трафиком. Информация об этом проекте хранится на сайте `xorp.org`.

Интересный аспект платформы XORP заключается в том, что она не только функционирует в разных операционных системах (Linux, разных версиях BSD, Mac OS X и Windows Server 2003), но и может запускаться непосредственно с "живого" компакт-диска (т.е. непосредственно с компакт-диска, без инсталляции на жесткий диск. — *Примеч. ред.*) на персональном компьютере. Этот "живой" компакт-диск (live CD) скрытно основан на системе Linux, но он прошел долгую эволюцию и позволяет превращать типичный персональный компьютер в специализированное устройство для маршрутизации.

15.7. Маршрутизаторы Cisco

Маршрутизаторы, выпускаемые компанией Cisco Systems, Inc., сегодня являются стандартом де-факто на рынке интернет-маршрутизаторов. Захватив более 60% рынка, компания Cisco добилась широкой известности своих продуктов, к тому же есть много специалистов, умеющих работать с этими устройствами. Раньше в качестве маршрутизаторов часто приходилось использовать UNIX-системы с несколькими сетевыми интерфейсами. Сегодня специализированные маршрутизаторы размещены в коммутационных шкафах и над подвесными потолками, где сходятся сетевые кабели.

Большинство маршрутизаторов Cisco работает под управлением операционной системы Cisco IOS, которая является собственностью компании и не имеет отношения к системе UNIX. У нее достаточно большой набор команд: полная бумажная документация занимает на полке почти полтора метра. Мы никогда не смогли бы полностью описать здесь эту операционную систему, но все же постараемся дать о ней основные сведения.

По умолчанию в системе IOS определены два уровня доступа (пользовательский и привилегированный), каждый из которых включает механизм парольной защиты.

Чтобы войти в пользовательский режим можно воспользоваться **ssh**. Вы получите приглашение на ввод пароля.

```
$ ssh acme-gw.acme.com
Password: <password>
```

После ввода правильного пароля появится приглашение интерпретатора EXEC.

```
acme-gw.acme.com>
```

С этого момента можно начинать вводить команды, например **show interfaces** для отображения списка сетевых интерфейсов маршрутизатора или **show ?** для получения справки по возможным параметрам.

Для того чтобы перейти в привилегированный режим, введите команду **enable**, а при последующем запросе — привилегированный пароль. Признаком привилегированного режима является наличие символа "**#**" в конце строки приглашения.

```
acme-gw.acme.com#
```

Будьте осторожны! В данном режиме можно делать все что угодно, включая удаление конфигурационной информации и даже самой операционной системы. Если сомневаетесь, обратитесь к справочным руководствам по системе Cisco или одной из исчерпывающих книг, выпускаемых издательством Cisco Press.

Команда **show running** позволяет узнать текущие динамические параметры маршрутизатора, а по команде **show config** отображаются текущие неизменяемые параметры. В большинстве случаев оба набора данных идентичны. Типичная конфигурация выглядит следующим образом.

```
acme-gw.acme.com# show running
Current configuration:
version 12.4
hostname acme-gw
enable secret xxxxxxxx
ip subnet-zero

interface Ethernet0
    description Acme internal network
        ip address 192.108.21.254 255.255.255.0
        no ip directed-broadcast
interface Ethernet1
    description Acme backbone network
        ip address 192.225.33.254 255.255.255.0
        no ip directed-broadcast
ip classless
line con 0
transport input none
line aux 0
    transport input telnet
line vty 0 4
    password xxxxxxxx
```

```
    login
end
```

Модифицировать конфигурацию маршрутизатора можно разными способами. Компания Cisco предлагает графические утилиты, работающие в некоторых версиях UNIX/Linux и Windows, но опытные сетевые администраторы никогда ими не пользуются. Их самый мощный "инструмент" — командная строка. Кроме того, с помощью команды **scp** можно загрузить с маршрутизатора его конфигурационный файл и отредактировать в любимом текстовом редакторе, после чего снова записать файл в память маршрутизатора.

Для того чтобы отредактировать конфигурационный файл в режиме командной строки, введите команду **config term**.

```
acme-gw.acme.com# config term
Enter configuration commands, one per line. End with CNTL/Z.
acme-gw(config)#
```

Теперь можно вводить конфигурационные команды в том виде, в котором их будет отображать команда **show running**. Например, если требуется изменить IP-адрес интерфейса `Ethernet0`, введите следующее.

```
interface Ethernet0
ip address 192.225.40.253 255.255.255.0
```

По окончании ввода конфигурационных команд нажмите <Ctrl+Z>, чтобы вернуться к обычной командной строке. Если все прошло успешно, сохраните новую конфигурацию в постоянной памяти посредством команды **write mem**.

Приведем несколько советов, касающихся эффективной работы с маршрутизаторами Cisco.

- Присвойте маршрутизатору имя с помощью команды **hostname**. Это позволит избежать несчастных случаев, связанных с изменением конфигурации не того маршрутизатора. Заданное имя всегда будет отображаться в командной строке.
- Всегда храните резервную копию конфигурационного файла маршрутизатора. С помощью команд **scp** или **tftp** можно каждую ночь пересылать текущую конфигурацию в другую систему на хранение.
- Зачастую существует возможность хранить копию конфигурации в энергонезависимой памяти NVRAM или на переносном флеш-накопителе. Не пренебрегайте этим!
- Сконфигурировав маршрутизатор для SSH-доступа, отключите протокол Telnet совсем.
- Контролируйте доступ к командной строке маршрутизатора, создавая списки доступа для каждого порта VTY (аналогичен порту PTY в UNIX-системе). Это не позволит посторонним пользователям "взломать" маршрутизатор.
- Управляйте трафиком между сетями (по возможности, и во внешний мир тоже), создавая списки доступа для каждого интерфейса. Более подробная информация приведена в разделе 22.11.
- Физически ограничивайте доступ к маршрутизаторам. Ведь если у злоумышленника будет физический доступ к устройству, он легко сможет изменить пароль привилегированного режима.

Если у вас несколько маршрутизаторов и несколько сотрудников, отвечающих за их работу, воспользуйтесь утилитой RANCID, которую можно загрузить с сайта `shruberry.net`. Название RANCID (игра слов: *rancid — негодный. — Примеч. ред.*) го-

ворит само за себя, но у этой программы есть одно преимущество: она регистрируется на ваших маршрутизаторах каждую ночь и получает их файлы конфигурации. Затем она сравнивает эти файлы и сообщает вам об их изменениях. Кроме того, она автоматически передает файлы конфигурации системе контроля версий (см. раздел 7.8).

15.8. Литература

- Perlman Radia. *Interconnections: Bridges, Routers, Switches, and Interworking Protocols (2nd Edition).* Reading, MA: Addison-Wiley, 2000. Это выдающаяся книга в данной области. Если вы решили купить только одну книгу об основах работы в сетях, покупайте ее. Кроме того, не упускайте шанса "зависнуть" с Радией — она очень веселая и обладает удивительно обширными знаниями.
- Edgeworth, Brad, Aaron Foss, and Ramiro Garza Rios. *IP Routing on Cisco IOS, IOS XE, and IOS XR: An Essential Guide to Understanding and Implementing IP Routing Protocols.* Indianapolis, IN: Cisco Press, 2014.
- Huitema Christian. *Routing in the Internet, Second Edition.* Prentice Hall, 2000. Эта книга представляет собой отличное введение в маршрутизацию для начинающих. В ней описано большинство используемых сегодня протоколов маршрутизации, а также рассматривается ряд сложных тем, например групповое вещание.

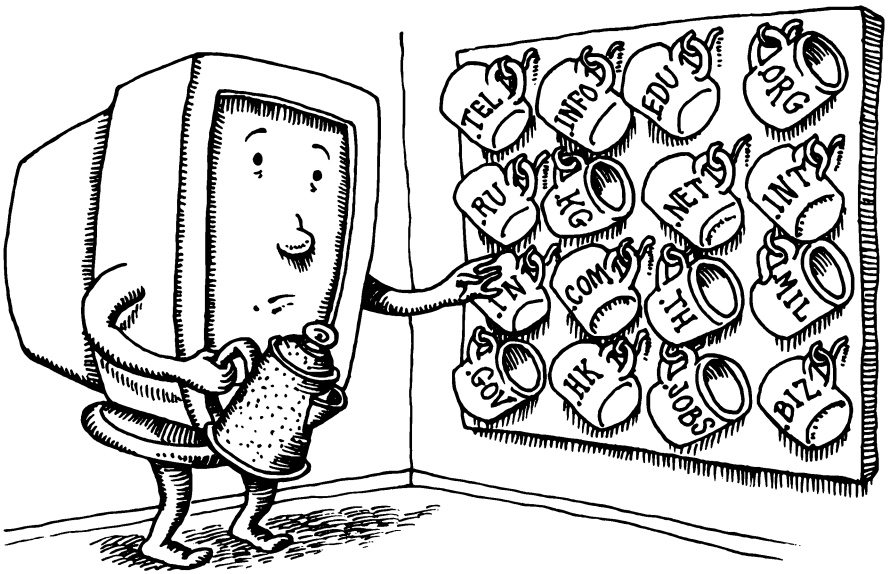
Существует много документов RFC, посвященных маршрутизации. Основные из них перечислены в табл. 15.3.

Таблица 15.3. Документы RFC, посвященные маршрутизации

RFC	Название	Авторы
1256	ICMP Router Discovery Messages	Deering
1724	RIP Version 2 MIB Extension	Malkin, Baker
2080	RIng for IPv6	Malkin, Minnear
2328	OSPF Version 2	Moy
2453	RIP Version 2	Malkin
4271	A Border Gateway Protocol 4 (BGP-4)	Rekhter, Li, et al.
4552	Authentication/Confidentiality for OSPFv3	Gupta, Melam
4822	RIPv2 Cryprographic Authentication	Arkinson, Fanto
4861	Neighbor Discovery for IPv6	Narten et al.
5175	IPv6 Router Advertisement Flags Option	Haberman, Hinden
5308	Routing IPv6 with IS-IS	Hopps
5340	OSPF for IPv6	Coltun et al.
5643	Management Information Base for OSFv3	Joyal, NManral, et al.

Глава 16

DNS: система доменных имен



Интернет обеспечивает мгновенный доступ к ресурсам по всему миру, и каждый из этих компьютеров или сайтов имеет уникальное имя (например, `google.com`). Тем не менее любой, кто пытался найти друга или потерянного ребенка на переполненном стадионе, знает, что мало просто знать имя и его выкрикивать. Существенным фактором для поиска чего-либо (или кого-либо) является организованная система передачи, обновления и распространения имен и их местоположений.

Пользователи и программы на уровне пользователя любят ссылаться на ресурсы по имени (например, `amazon.com`), но сетевое программное обеспечение низкого уровня понимает только IP-адреса (например, 54.239.17.6). Сопоставление имен и адресов является самой известной и, возможно, самой важной функцией DNS, системы доменных имен. DNS включает в себя другие элементы и функции, но почти без исключения все они существуют для поддержки этой основной цели.

На протяжении всей истории Интернета система DNS вызывала как восхищение, так и критику. Ее первоначальная элегантность и простота способствовали успеху в первые годы и позволили Интернету быстро расти под небольшим централизованным управлением. По мере роста потребностей в дополнительных функциональных возможностях система DNS стала нуждаться в усовершенствовании. Иногда эти функции добавлялись способом, который сегодня выглядит уродливым. Скептики указывают на недостатки инфраструктуры DNS в качестве доказательства того, что Интернет находится на грани краха.

Однако, что ни говори, основные концепции и протоколы DNS до сих пор выдерживали рост от нескольких сотен хостов в одной стране до всемирной сети, которая под-

держивает более трех миллиардов пользователей на более чем одном миллиарде компьютеров.[1] Примеров информационной системы, которая выросла до этого масштаба с таким количеством вопросов, больше нет. Без DNS Интернет давно бы провалился.

16.1. Архитектура DNS

Система DNS — это распределенная база данных. В рамках этой модели один сервер хранит данные для компьютеров, о которых он знает, а другой сервер хранит данные для своего собственного набора компьютеров, при этом серверы взаимодействуют и обмениваются данными, когда одному серверу необходимо найти данные на другом сервере. С административной точки зрения DNS-серверы, которые вы настроили для своего домена, отвечают на внешние запросы об именах в вашем домене и в свою очередь запрашивают серверы других доменов от имени ваших пользователей.

Запросы и ответы

DNS-запрос состоит из имени и типа записи. Возвращаемый ответ — это набор "записей о ресурсах" (resource records — RR), которые реагируют на запрос (или, альтернативно, на ответ, указывающий, что имя и тип записи, которые вы запрашивали, не существуют).

"Реагирует" не обязательно означает "знает". DNS-серверы образуют иерархию, и для ответа на конкретный запрос может потребоваться связь с серверами на нескольких уровнях (см. раздел 16.4).[2] Серверы, не знающие ответа на запрос, возвращают записи ресурсов, которые могут помочь клиенту найти сервер, знающий ответ.

Наиболее распространенный запрос — это запрос записи **A**, которая возвращает IP-адрес, связанный с именем. На рис. 16.1 показан типичный сценарий.



Рис. 16.1. Простой поиск имени

Во-первых, человек вводит имя желаемого сайта в веб-браузер. Затем браузер вызывает библиотеку DNS Resolver для поиска соответствующего адреса. Библиотека Resolver создает запрос на запись **A** и отправляет ее на сервер имен, который возвращает запись **A** в своем ответе. Наконец, браузер открывает TCP-соединение с целевым хостом через IP-адрес, возвращаемый сервером имен.

[1] Статистика относительно пользователей взята с сайта `internetlivestats.com/internet-users`, а статистика по хостам — на сайте `statista.com`.

[2] Серверы имен обычно получают запросы на порт 53 UDP.

Поставщики услуг DNS

Несколько лет назад одной из основных задач каждого системного администратора было создание и обслуживание DNS-сервера. Сегодня ситуация изменилась. Если организация вообще поддерживает DNS-сервер, он часто используется только для внутреннего использования.[3]

Каждой организации по-прежнему нужен внешний DNS-сервер, но теперь для использования этой функции используется один из многих коммерческих "управляемых" поставщиков DNS. Эти службы предлагают интерфейс управления графическим интерфейсом и высокодоступную, защищенную DNS-инфраструктуру за очень небольшую плату в день. Amazon Route 53, CloudFlare, GoDaddy, DNS Made Easy и Rackspace — это лишь некоторые из основных поставщиков.

Конечно, вы можете настроить и сохранить свой собственный DNS-сервер (внутренний или внешний), если хотите. У вас есть десятки реализаций DNS на выбор, но система интернет-имен Berkeley Internet Name Domain (BIND) по-прежнему доминирует в Интернете. Более 75% DNS-серверов являются ее разновидностью.[4]

Независимо от того, какой путь вы выберете, в качестве системного администратора вам необходимо понять основные концепции и архитектуру DNS. Первые несколько разделов данной главы посвящены этим важным фундаментальным знаниям. Начиная с раздела 16.8, мы показываем некоторые конкретные конфигурации для BIND.

16.2. DNS для поиска

Независимо от того, запускаете ли вы свой собственный сервер имен, пользуйтесь управляемой службой DNS или кто-то предоставляет службу DNS для вас, вы обязательно захотите настроить все свои системы на поиск имен в DNS.

Для этого необходимо пройти два этапа. Во-первых, следует настроить свои системы как DNS-клиенты. Во-вторых, нужно сообщить системам, когда использовать DNS, а не другие методы поиска имен, такие как статический файл **`/etc/hosts`**.

`resolv.conf`: конфигурация клиентского модуля распознавания

Каждый компьютер, подключенный к сети, должен быть клиентом системы DNS. Конфигурация клиентской части системы DNS задается в файле **`/etc/resolv.conf`**, содержащем список DNS-серверов, которым можно посылать запросы.

🕮 Дополнительную информацию о протоколе см. в разделе 13.6.

Если ваш компьютер получает свой IP-адрес и сетевые параметры от DHCP-сервера, то файл **`/etc/resolv.conf`** должен заполняться автоматически. В противном случае его нужно редактировать вручную. Формат записей файла имеет следующий вид.

```
search имя_домена ...
nameserver IP-адрес
```

[3]Система Active Directory Microsoft включает интегрированный DNS-сервер, который прекрасно сочетается с другими службами, поддерживаемыми Microsoft в корпоративной среде. Однако Active Directory подходит только для внутреннего использования. Эта система никогда не должна использоваться как внешний DNS-сервер, связанный с Интернетом, из-за потенциальных проблем с безопасностью.

[4]См. обзор *ISC Internet Domain Survey*, опубликованный в июле 2015 г.

Может быть указано от одного до трех серверов имен. Рассмотрим пример.

```
search atrust.com booklab.atrust.com
nameserver 63.173.189.1       ; ns1
nameserver 174.129.219.225    ; ns2
```

В строке `search` приведен список доменов, которые следует опрашивать, если имя компьютера определено не полностью. Например, когда пользователь вводит команду **`ssh coraline`**, то распознаватель дополняет имя первым доменом в списке (в рассмотренном выше примере — `atrust.com`), после чего ищет хост `coraline.atrust.com`. Если это имя не найдено, делается попытка найти хост `coraline.booklab.atrust.com`. Количество доменов, которые можно задать в директиве `search`, зависит от специфики распознавателя; большинство из них допускает от шести до восьми доменов, при этом предельное количество символов равно 256.

Серверы имен, указанные в файле **`resolv.conf`**, должны разрешать вашему компьютеру посылать запросы и обязаны давать на них полные ответы (т.е. быть рекурсивными), а не отсылать вас на другие серверы имен (см. раздел 16.4).

Серверы имен опрашиваются по порядку. Если первый сервер отвечает на запрос, другие серверы игнорируются. Если возникает проблема, то по истечении времени, отведенного для ответа на запрос, он переадресуется следующему серверу. Все серверы опрашиваются по очереди, до четырех раз каждый. С каждой неудачей тайм-аут увеличивается. По умолчанию интервал тайм-аута задается равным пяти секундам, что для нетерпеливых пользователей кажется вечностью.

`nsswitch.conf`: кого я запрашиваю по имени?

Операционные системы FreeBSD и Linux используют файл переключения **`/etc/nsswitch.conf`** для того, чтобы указать, как выполняется отображение IP-адреса в имя компьютера и в каком порядке следует использовать систему DNS — первой, последней или вообще не использовать. Если файла переключения нет, то по умолчанию устанавливается следующее поведение.

```
hosts: dns [!UNAVAIL=return] files
```

Раздел `!UNAVAIL` означает, что если служба DNS доступна, но имя в ней не найдено, следует прекратить попытки поиска и не продолжать просмотр следующей записи (в данном случае файла **`/etc/hosts`**). Если сервер имен не был запущен (это бывает во время загрузки системы), то процесс поиска *рекомендуется* согласовывать с файлом **`hosts`**.

Во всех рассматриваемых нами дистрибутивах систем в файле **`nsswitch.conf`** содержится следующая запись, определяющую поведение по умолчанию.

```
hosts: files dns
```

Эта конфигурация дает приоритет файлу **`/etc/hosts`**, который всегда проверяется. Обращение к системе DNS происходит только для поиска имен, которые невозможно распознать с помощью файла **`/etc/hosts`**.

Не существует "наилучшего" способа конфигурации поиска — все зависит от того, как управляется ваша сеть. В общем, мы предпочитаем хранить как можно больше информации об хосте в системе DNS, но мы также пытаемся сохранить возможность вернуться при необходимости назад к статическим файлам **`hosts`** в процессе загрузки системы.

Если служба имени предоставляется вам внешней организацией, вы можете выполнить настройку DNS после настройки файлов **`resolv.conf`** и **`nsswitch.conf`**. В этом случае вы можете пропустить оставшуюся часть этой главы или продолжать читать дальше, чтобы узнать больше.

16.3. Пространство имен DNS

Пространство имен DNS представляет собой дерево с двумя основными ветвями, соответствующими прямым и обратным преобразованиям. Прямые преобразования ставят в соответствие именам компьютеров их IP-адреса (и другие записи), а обратные преобразования отображают IP-адреса в имена компьютеров. Каждое полностью определенное имя компьютера (например, `nubark.atrust.com`) — это узел дерева на ветви прямых преобразований, а каждый IP-адрес — это узел дерева на ветви обратных преобразований. Уровни дерева разделяются точками; корнем дерева является точка.

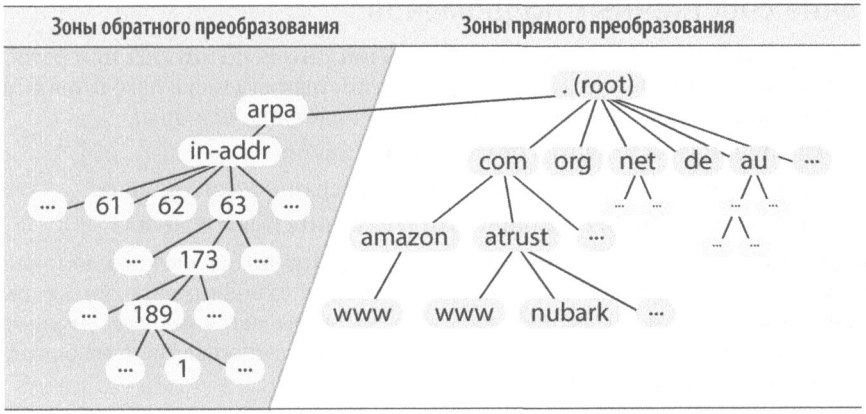


Рис. 16.2. Дерево DNS-зон

Для того чтобы система DNS могла работать с данными обоих видов, ветвь IP-адресов в пространстве имен инвертируется, т.е. октеты IP-адресов записываются в обратном порядке. Например, если хост "`nubark.atrust.com.`" имеет адрес 63.173.189.1, то соответствующий узел на ветви прямых преобразований в дереве имеет имя "`nubark.atrust.com.`", а узел на ветви обратных преобразований имеет имя "`1.189.173.63.in-addr.arpa.`"[5]

Оба эти имени заканчиваются точкой, так же как полные пути файлов всегда начинаются с косой черты. Это делает их "полностью квалифицированными доменными именами", или FQDN для краткости.

Вне контекста системы DNS имена, такие как `nubark.atrust.com` (без конечной точки), иногда называются "полностью квалифицированными именами хостов", но это жаргонизм. В самой системе DNS наличие или отсутствие конечной точки имеет решающее значение.

Существует два типа доменов верхнего уровня: национальные домены верхнего уровня (country code top-level domain — ccTLD) и общие домены верхнего уровня (generic top level domain — gTLD). Организация ICANN (International Corporation for Assigned Names and Numbers) управляет проектом регистрации имен в доменах gTLD, таких как `com`, `net` и `org`, с помощью аккредитованных агентств. Для регистрации имени в домене ccTLD зайдите на веб-страницу организации IANA (Internet Assigned Numbers Authority) `iana.org/cctld` и найдите реестр соответствующей страны.

[5] Часть имени `in-addr.arpa` представляет собой фиксированный суффикс.

Регистрация доменного имени

Для того чтобы зарегистрировать домен второго уровня, необходимо подать заявку в администрацию соответствующего домена верхнего уровня. Для того чтобы заполнить бланки регистрации, необходимо назначить ответственного технического специалиста и ответственного администратора, а также выбрать хотя бы два компьютера, которые будут серверами домена. Кроме того, необходимо выбрать доменное имя, никем не занятое. Стоимость регистрации зависит от агентства, но в настоящее время она относительно небольшая.

Создание собственных поддоменов

Процедура создания поддомена аналогична той, что используется при регистрации домена второго уровня, только центральная администрация теперь находится в пределах самой организации. Этот процесс предусматривает следующие этапы.

- Выбор имени, уникального в пределах организации.
- Назначение двух или более компьютеров серверами нового домена.[6]
- Согласование действий с администратором родительского домена.

Прежде чем передавать полномочия, администратор родительского домена должен убедиться в том, что серверы имен дочернего домена сконфигурированы и работают правильно. В противном случае произойдет то, что называется *некорректным делегированием*, и вы получите по электронной почте неприятное сообщение с требованием исправить ошибку (см. раздел 17.15).

16.4. Как работает система DNS

Серверы имен по всему миру работают вместе, чтобы отвечать на запросы. Как правило, они распространяют информацию, поддерживаемую любым администратором, ближайшим к цели запроса. Понимание ролей и взаимоотношений серверов имен важно как для повседневных операций, так и для отладки.

Серверы имен

Сервер имен выполняет несколько рутинных операций.

- Отвечает на запросы об именах и IP-адресах компьютеров.
- Посылает запросы локальным и удаленным компьютерам от имени своих пользователей.
- Кеширует ответы на запросы, для того чтобы ускорить ответы на них в следующий раз.
- Пересылает данные между серверами имен, чтобы обеспечить их синхронизацию.

Серверы имен работают с зонами, которые представляют собой домен без поддоменов. Часто термин *домен* употребляется как синоним термина *зона*, даже в этой книге.

Серверы имен работают в нескольких разных режимах, поэтому дать их точное определение нелегко. Ситуация усложняется еще и тем, что один и тот же сервер может выполнять неодинаковые функции в разных зонах. В табл. 16.1 перечислены прилагательные, употребляемые при описании серверов имен.

[6] С технической точки зрения, поскольку вы сами устанавливаете правила для своего поддомена, может быть один сервер или несколько.

Таблица 16.1. Классификация серверов имен

Тип сервера	Описание
Авторитетный (authoritative)	Официальный представитель зоны
главный (master)	Основное хранилище данных зоны; берет информацию из дискового файла
первичный (primary)	Синоним термина *главный*
подчиненный (slave)	Копирует данные с главного сервера
вторичный (secondary)	Синоним термина *подчиненный*
тупиковый (stub)	Похож на подчиненный сервер, но копирует данные только о серверах имен (записи NS)
внутренний (distribution)	Сервер, доступный только в пределах домена (также называется *невидимым сервером*)
Неавторитетный [a] (nonauthoritative)	Отвечает на запросы, пользуясь данными из кеша; не "знает", являются ли эти данные корректными
кеширующий (caching)	Кеширует данные, полученные от предыдущих запросов; обычно не имеет локальных зон
переадресующий (forwarder)	Выполняет запросы от имени многих клиентов; формирует большой кеш
Рекурсивный (recursive)	Осуществляет запросы от имени клиента до тех пор, пока не будет найден ответ
Нерекурсивный (nonrecursive)	Отсылает клиента к другому серверу, если не может получить ответ на запрос

[a] Строго говоря, атрибут "неавторитетный" относится к ответу на DNS-запрос, а не к самому серверу.

Серверы имен систематизируются на основании источника данных (авторитетный, кеширующий, главный, подчиненный), типа хранимых данных (усеченный), пути распространения запроса (переадресующий), типа выдаваемого ответа (рекурсивный, нерекурсивный) и, наконец, доступности сервера (внутренний). В следующих подразделах конкретизируются наиболее важные из этих различий; об остальных различиях пойдет речь в других разделах главы.

Авторитетные и кеширующие серверы

Главный, подчиненный и кеширующий серверы различаются только двумя характеристиками: откуда поступают данные и авторитетен ли сервер для домена. В каждой зоне есть один главный сервер имен.[7] На нем хранится официальная копия данных зоны (в файле на диске). Системный администратор модифицирует информацию, касающуюся зоны, редактируя файлы главного сервера.

Подчиненный сервер копирует свои данные с главного сервера посредством операции, называемой *передачей зоны* (zone transfer). В зоне должен быть как минимум один подчиненный сервер. Тупиковый сервер — особый вид подчиненного сервера, загружающий с главного сервера только записи NS (описания серверов имен). Один и тот же компьютер может быть главным сервером для одних зон и подчиненным — для других.

🕮 Дополнительную информацию о передаче зоны см. в разделе в разделе 16.9.

Кеширующий сервер имен загружает адреса серверов корневого домена из конфигурационного файла и накапливает остальные данные, кешируя ответы на выдаваемые

[7] Некоторые организации используют несколько главных серверов или вообще обходятся без них; мы описываем вариант, в котором существует один главный сервер.

им запросы. Собственных данных у кеширующего сервера нет, и он не является авторитетным ни для одной зоны (за исключением, возможно, зоны локального компьютера).

Гарантируется, что авторитетный ответ сервера имен является точным; неавторитетный ответ может быть устаревшим. Тем не менее очень многие неавторитетные ответы оказываются совершенно корректными. Главные и подчиненные серверы авторитетны для своих зон, но не для кешированной информации о других доменах. По правде говоря, даже авторитетные ответы могут быть неточными, если системный администратор изменил данные главного сервера (например, обновил порядковый номер зоны) и забыл передать их подчиненным серверам.

В каждой зоне должен быть хотя бы один подчиненный сервер. В идеале подчиненных серверов имен должно быть минимум два, причем один из них — вне организации. Подчиненные серверы на местах должны быть включены в разные сети и в разные цепи электроснабжения. Если служба имен вдруг перестанет функционировать, пользователи не смогут нормально работать в сети.

Рекурсивные и нерекурсивные серверы

Серверы имен бывают рекурсивными и нерекурсивными. Если у нерекурсивного сервера есть адрес, оставшийся в кеше от одного из предыдущих запросов, или он является авторитетным для домена, к которому относится запрашиваемое имя, то он даст соответствующий ответ. В противном случае он вернет отсылку на авторитетные серверы другого домена, которые с большей вероятностью ответят на запрос. Клиенты нерекурсивного сервера должны быть готовы принимать отсылки и обрабатывать их.

У нерекурсивных серверов обычно есть веские причины не выполнять дополнительную работу. Например, все корневые серверы и серверы доменов верхнего уровня являются нерекурсивными, поскольку им приходится обрабатывать десятки тысяч запросов в секунду.

Рекурсивный сервер возвращает только реальные ответы или сообщения об ошибках. Он сам отслеживает отсылки, освобождая от этой обязанности клиента. Базовая процедура анализа запроса, по сути, остается неизменной.

Из соображений безопасности, серверы имен организации, доступные извне, всегда должны быть нерекурсивными. Рекурсивные серверы имен, которые видны извне, могут стать объектом для атаки.

Учтите, что библиотечные функции распознавания имен *не понимают* отсылок. Любой локальный сервер имен, указанный в файле **resolv.conf** клиента, должен быть рекурсивным.

Записи о ресурсах

Каждая организация поддерживает один или несколько фрагментов распределенной базы данных, лежащей в основе всемирной системы DNS. Ваш фрагмент данных состоит из текстовых файлов, содержащих записи о каждом компьютере вашей сети; эти записи называются "записями о ресурсах". Каждая запись представляет собой отдельную строку, состоящую из имени (обычно имени компьютера), типа записи и некоторых значений. Поле имени можно не указывать, если его значение совпадает с именем в предыдущей строке.

Например, строки

```
nubark      IN   A    63.173.189.1
            IN   MX   10 mailserver.atrust.com.
```

в файле "прямого преобразования" (с именем **atrust.com**) и строка

```
1           IN   PTR nubark.atrust.com.
```

в файле "обратного преобразования" (с именем **63.173.189.rev**) связывают сайт nubark.atrust.com с IP-адресом 63.173.189.1. Запись MX перенаправляет сообщение электронной почты, адресованное на эту машину, на компьютер mailserver.atrust.com.

Поля IN обозначают классы записей. На практике, эти поля означают Интернет.

Записи о ресурсах — это универсальный язык системы DNS. Они не зависят от файлов конфигурации, управляющих операциями, которые выполняются на любой реализации данного сервера DNS. Все они являются фрагментами данных, циркулирующих внутри системы DNS, и кешируются в разных местах.

Делегирование

Все серверы имен считывают имена корневых серверов из локальных файлов конфигурации или содержат их в своем коде. Корневые серверы "знают" о доменах com, org, edu, fi, de и других доменах верхнего уровня. Эта цепочка продолжается дальше: сервер домена edu "знает" о домене colorado.edu, berkely.edu, сервер домена com "знает" о домене admin.com и т.д. Каждая зона может делегировать полномочия по управлению своими поддоменами другим серверам.

Рассмотрим реальный пример. Предположим, требуется узнать адрес хоста vangogh.cs.berkeley.edu, находясь на хосте lair.cs.colorado.edu. Компьютер lair просит локальный сервер имен, ns.cs.colorado.edu, найти ответ на этот вопрос. Последующие события представлены на рис. 16.3.

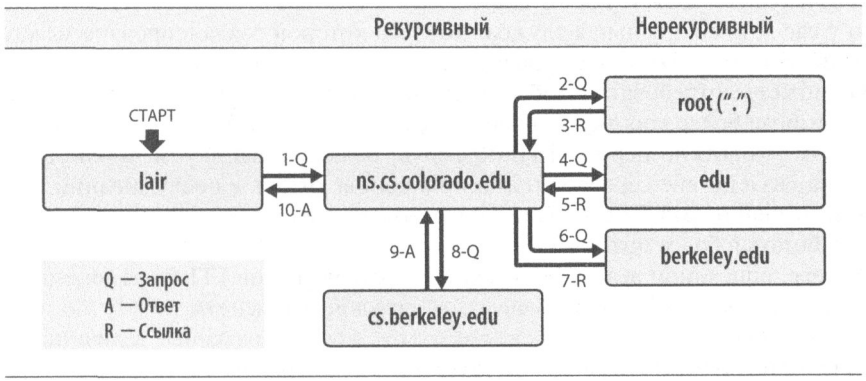


Рис. 16.3. Обработка запроса в DNS

Числа возле стрелок определяют порядок событий, а буквы — тип транзакции (запрос, ссылка или ответ). Предполагается, что никакие из требуемых данных предварительно не кешировались, за исключением имен и IP-адресов серверов корневого домена.

Локальный сервер имен не "знает" адреса компьютера vangogh. Более того, ему ничего не известно о доменах cs.berkeley.edu, berkeley.edu и даже edu. Он "знает" лишь некоторые серверы корневого домена, запрашивает корневой домен об хосте vangogh.cs.berkeley.edu и получает ссылку на серверы в домене edu.

Локальный сервер имен является рекурсивным. Если ответ на запрос содержит ссылку на другой сервер, то локальный сервер повторно направляет запрос новому серверу. Он продолжает этот процесс, пока не найдет требуемый сервер.

В нашем примере локальный сервер имен посылает запрос серверу домена edu (как всегда, запрашивая компьютер vangogh.cs.berkeley.edu) и получает ссылку на неза-

висимые серверы домена berkeley.edu. Затем локальный сервер повторяет запрос, направляя его в домен berkeley.edu. Если сервер университета Беркли не содержит ответ в кеше, он вернет ссылку на домен cs.berkeley.edu. Сервер этого домена компетентен в отношении запрашиваемой информации и возвращает адрес компьютера vangogh.

По окончании процедуры в кеше сервера ns.cs.colorado.edu окажется адрес компьютера vangogh. В кеше будут также находиться списки серверов доменов edu, berkeley.edu и cs.berkeley.edu.

Детали процедуры запроса можно выяснить с помощью утилит **dig +trace** или **drill -T**.[8]

Кеширование и эффективность

Кеширование повышает эффективность поиска: кешированный ответ выдается почти мгновенно и обычно точен, так как адресно-именные соответствия меняются редко. Ответ хранится в течение периода времени, называемого TTL (time to live), продолжительность которого задается владельцем искомой записи. Большинство запросов касается локальных компьютеров и обслуживается быстро. Повышению эффективности невольно содействуют и сами пользователи, так как многие запросы повторяются.

Обычно записи о ресурсах вашей организации должны использовать период TTL, продолжительность которого, как правило, лежит в диапазоне от одного часа до одного дня. Чем дольше период TTL, тем меньше трафик сети, потребляемый интернет-клиентами, получающими свежие копии записи.

Если у вас есть специальная служба, загрузка которой сбалансирована между логическими подсетями (этот процесс называется *балансированием нагрузки глобального сервера*), вы можете потребовать, чтобы поставщик услуг, выполняющий балансирование загрузки, выбрал более короткую продолжительность TTL, например десять секунд или одну минуту. (Короткий период TTL позволяет балансировщику загрузки быстро реагировать на бездействие серверов и атаки на основе отказа в обслуживании.) Система с короткими периодами TTL продолжает работать корректно, но ваши серверы имен должны работать в более интенсивном режиме.

В примере, описанном выше, продолжительность периодов TTL была установлена так: на корневых серверах — 42 дня, в домене edu — два дня, в домене berkeley.edu — два дня и один день — для сайта vangough.cs.berkeley.edu. Это разумные величины. Если вы планируете крупную перенумерацию, то можете сделать периоды TTL короче, чем раньше.

Серверы DNS также реализуют негативное кеширование. Иначе говоря, они помнят, в каких ситуациях запрос остался без ответа, и не повторяют его, пока не истечет период TTL для негативного кеширования. Ответы при негативном кешировании сохраняются в следующих ситуациях.

- Нет хоста или домена, соответствующего запрашиваемому имени.
- Данные запрашиваемого типа для данного хоста не существуют.
- Запрашиваемый сервер не отвечает.
- Сервер недоступен из-за проблем в сети.

Сервер BIND кеширует данные в двух первых ситуациях, а сервер Unbound — во всех четырех. Количество повторений негативного кеширования можно задавать в любой реализации.

[8] Утилиты **dig** и **drill** — это инструменты системы DNS; утилита **dig** входит в дистрибутивный набор сервера BIND, а утилита **drill** разработана группой NLnet Labs.

Неоднозначные ответы и балансировка загрузки DNS

Сервер имен в ответ на запрос часто получает несколько записей. Например, при попытке узнать адрес сервера имен корневого домена можно получить список, содержащий все 13 корневых серверов. Большинство серверов имен возвращает ответы в случайном порядке, выполняя примитивный вид балансирования загрузки.

Можно достичь балансирования загрузки своих серверов, закрепив одно имя за несколькими IP-адресами (которые в реальности соответствуют разным компьютерам).

```
www             IN   A    192.168.0.1
                IN   A    192.168.0.2
                IN   A    192.168.0.3
```

Большинство серверов имен возвращают многорекордные множества в другом порядке каждый раз, когда они получают запрос, вращая их в циклическом режиме. Когда клиент получает ответ с несколькими записями, наиболее распространенное поведение заключается в том, чтобы попробовать адреса в порядке, возвращаемом DNS-сервером.[9]

Эта схема обычно называется *балансированием нагрузки* на основе циклического распределения нагрузки. Однако в лучшем случае это грубое решение. На больших сайтах используется программное обеспечение балансирования нагрузки (например, HAProxy, см. раздел 19.6) или специализированные устройства балансирования нагрузки.

Отладка с помощью инструментов запросов

🕮 Дополнительную информацию о спецификациях DNSSEC см. в разделе 16.10.

Пять инструментов командной строки, которые запрашивают базу данных DNS, распространяются с помощью дистибутивов BIND: **nslookup**, **dig**, **host**, **drill** и **delv**. Утилиты **nslookup** и **host** просты и имеют неплохую производительность, но, чтобы получить все детали, нужны утилиты **dig** или **drill**. Утилита **drill** лучше работает с цепочками подписей DNSSEC. Имя команды **drill** обыгрывает имя **dig** (Domain Information Groper — средство сбора информации о доменах), подсказывая, что благодаря этой утилите можно получить еще больше информации от системы DNS, чем с помощью утилиты **dig**. Утилита **delv** впервые появилась в системе BIND 9.10 и в конечном итоге заменит утилиту **drill** для отладки DNSSEC.

🕮 Дополнительную информацию о расщеплении DNS см. в разделе 16.7.

По умолчанию утилиты **dig** и **drill** посылают запросы серверам имен, сконфигурированным в файле **/etc/resolv.conf**. Аргумент *@сервер_имен* адресует команду конкретному серверу имен. Способность посылать запросы конкретному серверу позволяет гарантировать, что любые изменения, вносимые в зону, будут распространены среди подчиненных серверов и во внешнем мире. Это свойство особенно полезно, если вы используете представления (расщепляете DNS) и должны проверять, правильно ли они сконфигурированы.

Если указать тип записи, то команды **dig** и **drill** будут выполнять запрос только к этим записям. Псевдотип **any** действует немного неожиданно: вместо того чтобы вернуть все данные, ассоциированные с указанным именем, он возвращает все *кешированные* данные, связанные с ним. Итак, чтобы получить все записи, необходимо выполнить команду **dig** *домен* **NS**, за которой следует выражение **dig @nsl.***домен.домен* **any**. (Авторитетные данные в этом контексте рассматриваются как кешированные.)

[9]Однако это поведение не является обязательным. Разные клиенты могут вести себя по-разному.

Утилита **dig** имеет более 50 опций, а утилита **drill** — около половины этого количества. Получив флаг **-h**, эти утилиты выдают список всех своих опций. (Для того чтобы упорядочить вывод, можно передать его утилите **less**.) Для обеих утилит опция **-x** изменяет порядок следования байтов в IP-адресе на противоположный и выполняет обратный запрос. Флаги **+trace** утилиты **dig** и **-T** утилиты **drill** показывают итеративные шаги в процессе распознавания, начиная от корня и вниз по дереву иерархии.

Если ответ является авторитетным (т.е. приходит непосредственно от главного или подчиненного сервера данной зоны), то команды **dig** и **drill** во флагах вывода используют символы `aa`. Код `ad` означает, что ответ является аутентичным в смысле протокола DNSSEC. Тестируя новую конфигурацию, следует убедиться, что вам доступны данные как локальных, так и удаленных хостов. Если вы имеете доступ к хосту только по его IP-адресу, а не по имени, то в этом, вероятно, виновата система DNS.

Чаще всего утилита **dig** используется для выяснения того, какие записи в настоящее время возвращаются для определенного имени. Если возвращается только ответ `AUTHORITY`, значит, вы перенаправлены на другой сервер имен. Если возвращается ответ `ANSWER`, значит, на ваш вопрос был дан ответ (и, может быть, была включена другая информация).

Часто бывает полезно следить за цепочкой делегирования вручную с корневых серверов, чтобы убедиться, что все находится в правильных местах. Ниже мы рассмотрим пример этого процесса для названия `www.viawest.com`. Во-первых, мы запрашиваем корневой сервер, чтобы узнать, кто является авторитетным для сайта `viawest.com`, запросив начальную запись (start-of-authority — `SOA`).

```
$ dig @a.root-servers.net viawest.com soa
; <<>> DiG 9.8.3-P1 <<>> @a.root-servers.net viawest.com soa
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 7824
;; flags: qr rd; QUERY: 1, ANSWER: 0, AUTHORITY: 13, ADDITIONAL: 14
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;viawest.com. IN SOA

;; AUTHORITY SECTION:
com.                172800 IN NS    c.gtld-servers.net.
com.                172800 IN NS    b.gtld-servers.net.
com.                172800 IN NS    a.gtld-servers.net.
...
;; ADDITIONAL SECTION:
c.gtld-servers.net. 172800 IN A    192.26.92.30
b.gtld-servers.net. 172800 IN A    192.33.14.30
b.gtld-servers.net. 172800 IN AAAA 2001:503:231d::2:30
a.gtld-servers.net. 172800 IN A    192.5.6.30
...

;; Query time: 62 msec
;; SERVER: 198.41.0.4#53(198.41.0.4)
;; WHEN: Wed Feb 3 18:37:37 2016
;; MSG SIZE rcvd: 489
```

Обратите внимание на то, что возвращенное состояние имеет значение `NOERROR`. Это говорит о том, что запрос возвратил ответ без заметных ошибок. Другими распростра-

ненными значениями состояния являются NXDOMAIN, которое указывает, что запрошенное имя не существует (или не зарегистрировано) и SERVFAIL, которое обычно указывает на ошибку конфигурации на самом сервере имен.

Этот раздел AUTHORITY SECTION указывает, что глобальные серверы домена верхнего уровня (gTLD) являются следующим звеном в цепочке полномочий для этого домена. Итак, мы выбираем один случайным образом и повторяем один и тот же запрос.

```
$ dig @c.gtld-servers.net viawest.com soa
; <<>> DiG 9.8.3-P1 <<>> @c.gtld-servers.net viawest.com soa
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 9760
;; flags: qr rd; QUERY: 1, ANSWER: 0, AUTHORITY: 2, ADDITIONAL: 2
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;viawest.com.                IN SOA

;; AUTHORITY SECTION:
viawest.com.         172800 IN NS      ns1.viawest.net.
viawest.com.         172800 IN NS      ns2.viawest.net.

;; ADDITIONAL SECTION:
ns1.viawest.net.     172800 IN A       216.87.64.12
ns2.viawest.net.     172800 IN A       209.170.216.2
;; Query time: 52 msec

;; SERVER: 192.26.92.30#53(192.26.92.30)
;; WHEN: Wed Feb 3 18:40:48 2016
;; MSG SIZE rcvd: 108
```

Этот ответ намного более краткий, и теперь мы знаем, что следующий сервер для запроса — ns1.viawest.com (или ns2.viawest.com).

```
$ dig @ns1.viawest.net viawest.com soa
; <<>> DiG 9.8.3-P1 <<>> @ns2.viawest.net viawest.com soa
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 61543
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1
;; WARNING: recursion requested but not available
;; QUESTION SECTION:
;viawest.com.                IN SOA

;; ANSWER SECTION:
viawest.com.         3600   IN SOA    mvec.viawest.net. hostmaster.
   viawest.net. 2007112567 3600 1800 1209600 3600

;; AUTHORITY SECTION:
viawest.com.        86400   IN  NS    ns2.viawest.net.

;; ADDITIONAL SECTION:
ns2.viawest.net.     3600   IN   A    209.170.216.2

;; Query time: 5 msec
;; SERVER: 216.87.64.12#53(216.87.64.12)
```

```
;; WHEN: Wed Feb 3 18:42:20 2016
;; MSG SIZE rcvd: 126
```

Этот запрос возвращает ANSWER для домена viawest.com. Теперь мы знаем авторитетный сервер имен и можем запросить имя, которое мы действительно хотим, www.viawest.com.

```
$ dig @ns1.viawest.net www.viawest.com any
; <<>> DiG 9.8.3-P1 <<>> @ns1.viawest.net www.viawest.com any
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 29968
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;www.viawest.com.          IN ANY

;; ANSWER SECTION:
www.viawest.com.       60   IN CNAME  hm-d8ebfa-via1.threatx.io.

;; AUTHORITY SECTION:
viawest.com.        86400   IN NS     ns2.viawest.net.

;; ADDITIONAL SECTION:
ns2.viawest.net.     3600   IN  A     209.170.216.2

;; Query time: 6 msec
;; SERVER: 216.87.64.12#53(216.87.64.12)
;; WHEN: Wed Feb 3 18:46:38 2016
;; MSG SIZE rcvd: 117
```

Этот окончательный запрос показывает нам, что сайт www.viawest.com имеет запись CNAME, указывающую на hm-d8ebfa-via1.threatx.io, что означает другое имя для хоста threatx (хоста, управляемого облачным распределенным поставщиком услуг). Конечно, если вы запросите рекурсивный сервер имен, он проследует за всей цепочкой делегирования от вашего имени. Однако при отладке обычно более полезно исследовать цепочку ссылка за ссылкой.

16.5. База данных DNS

Базы данных зон DNS — это множество текстовых файлов, поддерживаемых системным администратором на главном сервере имен зоны. Эти текстовые файлы часто называются *файлами зон*. Они содержат записи двух типов: команды синтаксического анализатора (например, $ORIGIN и $TTL) и записи о ресурсах. Базе данных принадлежат только записи о ресурсах, а команды синтаксического анализатора предназначены для того, чтобы упростить ввод записей.

📖 Команды файла зоны стандартизованы в документах RFC 1035 и 2308.

Команды синтаксического анализатора в файлах зон

Команды могут быть встроены в файлы зон, чтобы сделать их более читабельными и облегчить их сопровождение. Эти команды либо влияют на способ, с помощью кото-

рого синтаксический анализатор интерпретирует последующие записи, либо сами являются сокращением нескольких записей DNS. После того как файл зоны будет прочитан и интерпретирован, ни одна из этих команд не становится частью данных о зоне (по крайней мере в их исходном виде).

Три команды ($ORIGIN, $INCLUDE и $TTL) являются стандартными в системе DNS, а четвертая, $GENERATE, относится только к серверу BIND. Пример использования команды $GENERATE приведен ниже. Стандартные команды выглядят следующим образом. Команды должны начинаться в первой колонке и занимать отдельную строку.

Файлы зон читаются и анализируются сверху вниз за один проход. Когда сервер имен читает файл зоны, он добавляет стандартное имя домена ("источник") к любому имени, которое не полностью определено. По умолчанию источником служит домен, указанный в файле конфигурации сервера имен. Однако посредством директивы $ORIGIN можно задать или изменить источник в файле зоны.

```
$ORIGIN имя_домена
```

Использование относительных имен вместо полностью определенных позволяет сэкономить много времени на вводе данных и делает файлы зон гораздо более удобными для восприятия.

Во многих организациях в файлы зон включаются директивы $INCLUDE, позволяющие разделять базы данных зон на логические блоки или хранить ключи шифрования в отдельном файле с ограниченными правами доступа. Синтаксис директивы $INCLUDE таков.

```
$INCLUDE имя_файла [источник]
```

Указанный файл включается в базу данных в том месте, где стоит эта директива. Если имя файла не является полностью определенным, оно интерпретируется относительно каталога, из которого был запущен сервер имен.

Если задано значение параметра *источник*, то синтаксический анализатор действует так, будто чтению файла предшествовала директива $ORIGIN. Обратите внимание на то, что значение *источник* не возвращается к своему предыдущему значению после выполнения директивы $INCLUDE. Возможно, вы захотите вернуться к предыдущему значению либо в конце включенного файла, либо в строке, следующей за директивой $INCLUDE.

Директива $TTL задает стандартное время существования последующих записей. Она должна стоять в первой строке файла зоны. По умолчанию время жизни измеряется в секундах, но можно задать и другие единицы измерения: часы (h), минуты (m), дни (d) или недели (w). Например, все перечисленные ниже директивы

```
$TTL 86400
$TTL 24h
$TTL 1d
```

устанавливают значение $TTL равным одному дню.

Записи о ресурсах

С каждой зоной в иерархии DNS связан набор записей о ресурсах. Базовый формат этой записи имеет следующий вид.

```
[имя] [ttl] [класс] тип данные
```

Поля разделяются знаками табуляции или пробелами и могут содержать специальные символы (табл. 16.2).

Таблица 16.2. Специальные символы, используемые в записях о ресурсах

Символ	Назначение
;	Начало комментария
@	Имя текущей зоны
()	Разбивка данных на несколько строк
*	Метасимвол[a] (только в поле *имя*)

[a] Предупреждения, связанные с этим символом, приведены ниже.

Поле *имя* идентифицирует объект (обычно хост или домен), к которому относится запись. Если несколько последовательно расположенных записей ссылаются на один и тот же объект, то после первой записи поле *имя* можно опустить. Поле должно начинаться в первой колонке, если она присутствует.

Имя может быть относительным либо абсолютным. Абсолютные имена заканчиваются точкой и полностью определены. На внутреннем уровне программное обеспечение работает с полными именами, добавляя имя текущего домена и точку ко всем именам, которые не заканчиваются точкой. Это позволяет укорачивать имена, но часто сопряжено с ошибками.

Например, в домене `cs.colorado.edu` имя **anchor** интерпретируется как "`anchor.cs.colorado.edu.`". Если же ввести имя `anchor.cs.colorado.edu`, то отсутствие точки в конце также будет подразумевать сокращенное имя, к которому следует добавить стандартный домен, что в итоге даст имя "`anchor.cs.colorado.edu.cs.colorado.edu.`". Это очень распространенная ошибка.

В поле *ttl* (time-to-live — время существования) задается время (в секундах), в течение которого элемент данных может оставаться в кеше и при этом считаться достоверным. Это поле часто опускают, но оно обязательно на корневом сервере в файле подсказок. Стандартное значение поля задается директивой `$TTL`, указываемой в первой строке файла зоны.

Если время существования записей задать равным примерно неделе, то это приведет к значительному снижению сетевого трафика и нагрузки на DNS. Однако следует помнить о том, что после сохранения записи в кеше за пределами локальной сети ее уже нельзя принудительно сделать недостоверной. Поэтому, планируя серьезную реструктуризацию сети, сделайте значение `$TTL` достаточно низким (например, один час), чтобы записи, находящиеся во внешних кешах, быстро устарели, а новые параметры в течение часа вступили в действие. После завершения работы восстановите исходное значение этого параметра.

Некоторые организации устанавливают небольшое время существования своих записей на серверах, имеющих выход в Интернет, для того чтобы в случае возникновения проблем на этих серверах (сбоя сети, отказа аппаратного обеспечения или атаки на основе отказа в обслуживании) администраторы могли отреагировать, изменив параметры отображения "имя-адрес". Поскольку исходное время существования записей было незначительным, новые значения будут быстро распространены по сети. Например, имя `google.com` имеет параметр TTL, равный пяти минутам, а параметр TTL на серверах имен компании Google равен четырем дням (345 600 с).

```
google.com          300     IN    A   209.85.171.100
google.com       345600     IN    NS  ns1.google.com
ns1.google.com   345600     IN    A   216.239.32.10
```

Для выяснения этих данных мы использовали команду `dig`; для простоты вывод был сокращен.

В поле *класс* задается тип сети. По умолчанию запись `IN` всегда означает Интернет.

Существуют различные типы DNS-записей, из которых широко используются менее десяти. В стандарте IPv6 было добавлено еще несколько типов. Записи о ресурсах разбиваются на четыре группы.

- Записи зон определяют домены и их серверы имен.
- Базовые записи связывают имена с адресами и обеспечивают маршрутизацию электронной почты.[10]
- Аутентификационные записи предоставляют информацию, касающуюся аутентификации и сигнатур.
- Вспомогательные записи содержат дополнительную информацию о компьютерах и доменах.

Содержимое поля *данные* зависит от типа записи. Запрос DNS о конкретном домене и типе записи получает в ответ все соответствующие ему записи о ресурсах, извлеченные из файла зоны (табл. 16.3).

Таблица 16.3. Записи базы данных DNS

	Тип	Имя	Назначение
Зонные	`SOA`	Start Of Authority	Определение DNS-зоны
	`NS`	Name Server	Определение серверов имен зоны, делегирование полномочий поддоменам
Базовые	`A`	IPv4 Address	Преобразование имени в адрес IPv4
	`AAAA`	IPv6 Address	Преобразование имени в адрес IPv6; ранее считалась устаревшей, но в настоящее время возрождается
	`PTR`	Pointer	Преобразование адреса в имя
	`MX`	Mail Exchanger	Управляет маршрутизацией почты
Безопасность	`DS`	Delegation Signer	Хеширует подписанный ключ дочерней зоны
	`DNSKEY`	Public Key	Открытый ключ для имени DNS
	`NSEC`	Next Secure	Используется вместе со спецификацией DNSSEC для генерации отказов
	`NSEC3`	Next Secure v3	Используется вместе со спецификацией DNSSEC для генерации отказов
	`RRSIG`	Signature	Множество подписанных аутентифицированных записей о ресурсах
Вспомогательные	`CNAME`	Canonical Name	Дополнительные имена (псевдонимы) хоста
	`SRV`	Services	Местонахождение известных служб в пределах домена
	`TXT`	Text	Комментарии или нестандартная информация

Некоторые типы записей устарели, либо являются экспериментальными, либо не нашли широкого применения. Полный их список приведен в документации. Большинство записей поддерживается вручную (путем редактирования текстовых фай-

[10] Записи о маршрутизации почты MX относятся как к категории записей зон, так и к категории базовых записей, поскольку они могут относиться как ко всей зоне, так и к отдельным хостам.

лов), но защищенные записи о ресурсах требуют криптографической обработки и поэтому управляются с помощью программного обеспечения. Эти записи описаны в подразделе, посвященном спецификации DNSSEC, в разделе 16.10.

Порядок записей о ресурсах почти произволен, но по традиции запись SOA должна идти первой. Последующие записи могут располагаться в любом порядке, но обычно сразу же после записи SOA следуют записи NS. Записи по каждому хосту, как правило, группируются. Вообще, принято упорядочивать записи по полю *имя*, хотя некоторые организации упорядочивают их по IP-адресам, чтобы было легче идентифицировать неиспользуемые адреса. Файлы зон на подчиненных серверах не поддерживаются вручную. Они записываются программным обеспечением сервера имен; порядок записей скрыт.

Подробно описывая все типы записей о ресурсах в следующих разделах, мы в качестве примера рассмотрим записи из базы данных домена `atrust.com`. Поскольку домен по умолчанию — `atrust.com.`, то имя хоста **`bark`** в действительности означает `bark.atrust.com`.

🕮 Более подробная информация о документах RFC приведена в разделе 13.1.

Формат и интерпретация каждого типа записей о ресурсах описаны организацией IETF в ряде документов RFC. В следующих разделах укажем конкретные документы RFC, относящиеся к каждой записи (вместе с годом их выпуска), в специальных примечаниях.

Запись SOA

Запись SOA (Start of Authority — начало полномочий) обозначает начало *зоны* — группы записей о ресурсах, расположенных в одной точке пространства имен DNS. Этот узел дерева DNS называют также точкой передачи полномочий. Как мы увидим ниже, домен DNS обычно охватывает минимум две зоны: для прямого преобразования имен компьютеров в IP-адреса и обратного преобразования. Часть дерева DNS, которая служит целям прямого преобразования, упорядочена по именам, а ветвь обратного преобразования — по IP-адресам.

🕮 Записи SOA определены в документе RFC1035 (1987).

Для каждой зоны создается только одна запись SOA. Запись SOA содержит имя зоны, контактный адрес администрации зоны, порядковый номер и различные параметры обновления данных. Комментарии начинаются с точки с запятой. Рассмотрим следующий пример.

```
; Начало зоны для домена atrust.com

atrust.com    IN  SOA  ns1.atrust.com. hostmaster.atrust.com. (
      2009070200  ; Порядковый номер
      10800       ; Refresh  (3 часа)
      1200        ; Retry    (20 минут)
      360000      ; Expire   (больше 40 дней)
      3600 )       ; Minimum  (1 час)
```

Поле *имя* записи SOA (в нашем примере `atrust.com.`) часто содержит символ `@`, обозначающий сокращенное имя текущей зоны. Текущее имя задается в инструкции `zone` файла **`named.conf`** или в элементе `name` файла зоны **`nsd.conf`**. Оно может быть изменено в файле зоны посредством директивы синтаксического анализатора `$ORIGIN`.

В показанном фрагменте нет поля *ttl*. Класс зоны — `IN` (Интернет), тип записи — SOA, а остальные элементы образуют поле *данные*. Числовые параметры в скобках пред-

ставляют собой продолжительность временных интервалов и часто записываются в одной строке с комментариями.

Имя `ns1.atrust.com.` указывает на главный сервер имен этой зоны.[11]

Имя `hostmaster.cs.colorado.edu.` определяет адрес электронной почты для контактов с администратором домена. Адрес дан в формате "*пользователь.хост*." (а не *пользователь@хост*). К сожалению, из-за спама и по другим причинам большинство организаций не спешат обновлять свою контактную информацию.

Круглые скобки позволяют разбить запись SOA на несколько строк.

Первый числовой параметр — это порядковый номер конфигурации зоны. С его помощью подчиненные серверы определяют, когда следует загружать новые данные. Порядковым номером может быть любое 32-разрядное целое число, причем оно должно увеличиваться при каждом изменении файла зоны. Во многих организациях в этом номере зашифровывается дата последней модификации файла. Например, значение `2017110200` указывает на первое изменение в файле зоны, сделанное 2 ноября 2017 года.

Порядковые номера могут не быть последовательными, но должны монотонно возрастать. Если случайно задать на главном сервере очень большое число и передать его подчиненным серверам, то исправить порядковый номер на главном сервере не удастся. Подчиненные серверы запрашивают новые данные только в том случае, когда порядковый номер записи SOA главного сервера больше, чем у них.

Существует два способа решения этой проблемы.

- Можно воспользоваться особенностями последовательного пространства, из которого выбираются порядковые номера. Суть в том, что к имеющемуся номеру добавляется "магическое" число (2^{31}), заставляющее подчиненные серверы обновить свои данные, после чего устанавливается требуемый номер. Этот прием описывается в книге *DNS and BIND* издательства O'Reilly и в документе RFC1982.
- Более хитроумный и утомительный способ — изменить порядковый номер на главном сервере, удалить процессы на подчиненных серверах вместе с их резервными базами данных, а затем перезапустить серверы. При отсутствии кеша подчиненные серверы повторно загрузят базы данных с главного сервера. Этот метод трудно реализовать, если, следуя полезным советам, вы распределили подчиненные серверы по географическому признаку, особенно если на этих подчиненных серверах нет системных администраторов.

Часто пользователи делают одну и ту же ошибку: меняют файлы данных, забывая при этом исправить порядковый номер. "В наказание" сервер имен откажется распространить внесенные изменения на подчиненные серверы.

Следующие четыре элемента записи SOA — значения интервалов времени (по умолчанию в секундах), определяющих, как долго данные могут находиться в кеше в различных точках глобальной базы данных DNS. Можно поменять единицы измерения, воспользовавшись суффиксами `m` (минуты), `h` (часы), `d` (дни) и `w` (недели). Например, выражение `1h30m` означает "1 час 30 минут". Выбор интервала времени требует компромисса между эффективностью (использование старого значения дешевле выборки нового) и точностью (новые значения более точные). Эти четыре поля называются *refresh*, *update*, *expire* и *minimum*.

Первый элемент задает периодичность обновления данных. Он показывает, как часто подчиненные серверы должны связываться с главным сервером и проверять, не поме-

[11] На самом деле в записи SOA может быть указан любой сервер имен зоны, если вы не используете динамическую систему DNS. В этом случае запись SOA должна содержать имя главного сервера.

нялся ли порядковый номер конфигурации зоны. Если база данных зоны изменилась (порядковый номер главного сервера стал *больше*, чем у подчиненного сервера), подчиненные серверы должны обновить свои копии базы данных. Общепринятые значения для этого интервала — от одного до шести часов (3600–21600 секунд).

Вместо того чтобы пассивно ждать истечения периода обновления, современные серверы BIND (всегда) и NSD (иногда) самостоятельно уведомляют свои подчиненные серверы об изменениях зон. Уведомление об обновлении может быть потеряно из-за перегруженности сети, поэтому значение параметра *refresh* должно быть разумным.

Если подчиненный сервер пытается узнать порядковый номер у главного сервера, а тот не отвечает, то через некоторое время будет сделана повторная попытка. Как показывает опыт, нормальное значение для второго элемента — от 20 до 60 минут (1200–3600 секунд).

Если главный сервер длительное время отключен, то подчиненные серверы будут безуспешно пытаться обновить свои данные. По истечении определенного времени все подчиненные серверы должны "решить", что главный сервер не включится никогда и его данные наверняка устарели. Параметр *expire* определяет, как долго подчиненные серверы будут обслуживать домен в отсутствие главного сервера. Система должна сохранить работоспособность, даже если главный сервер не работает неделю, поэтому третьему элементу следует присваивать большое значение. Мы рекомендуем выбирать интервал от одного до двух месяцев.

Четвертый параметр *minimum* в записи SOA определяет время существования в кеше отрицательных ответов. Время существования положительных ответов (т.е. собственно записей) устанавливается директивой $TTL в начале файла зоны. Как свидетельствует практика, значение $TTL должно быть от нескольких часов до нескольких дней, а минимальное время существования отрицательных ответов — один-два часа (но не более трех).

Значения параметров $TTL, *expire* и *minimum* в конечном итоге вынуждают всех пользователей DNS удалять старые данные. Изначально система доменных имен основывалась на том, что информация об хостах относительно стабильна и меняется нечасто. Однако с появлением протокола DHCP и портативных компьютеров все изменилось. Теперь разработчики серверов имен отчаянно пытаются угнаться за временем, внедряя механизмы инкрементных передач зоны и динамических обновлений, которые будут описаны позднее. Концепция синхронизации была описана в этой главе ранее.

Записи NS

Записи NS (name server — сервер имен) идентифицируют серверы имен, которые авторитетны для зоны (т.е. все главные и подчиненные серверы), а также делегируют полномочия по управлению поддоменами другим организациям. Обычно эти записи следуют сразу после записи SOA.

🕮 Записи NS определены в документе RFC1035 (1987).

Эти записи имеют следующий формат.

```
зона [ttl] [IN] NS имя_хоста
```

Рассмотрим пример.

```
        NS    ns1.atrust.com.
        NS    ns2.atrust.com.
booklab NS    ubuntu.booklab.atrust.com.
        NS    ns1.atrust.com.
```

Первые две строки определяют серверы имен для домена atrust.com. Здесь параметр *name* не указан, потому что он совпадает с полем *name* записи SOA, которая предшествует этим записям; поэтому поле *name* оставлено пустым. Параметр *class* также не указан, потому что по умолчанию он равен IN и его необязательно задавать явно.

Третья и четвертая строки делегируют поддомен с именем booklab.atrust.com. серверам имен ubuntu.booklab и ns1. Эти записи на самом деле являются частью поддомена booklab, но они должны также появляться в родительской зоне atrust.com, чтобы делегирование было возможным. Аналогично записи NS для домена atrust.com хранятся в файле зоны .com, чтобы определить поддомен atrust.com и идентифицировать его серверы. Серверы домена .com отсылают запросы об хостах в домене atrust.com серверам, перечисленным в записях NS для домена atrust.com внутри домена .com.

🕮 Дополнительную информацию о делегировании см. в разделе 16.4.

Список серверов имен, расположенных в родительской зоне, должен содержать актуальную информацию, если это возможно. Несуществующие серверы, перечисленные в родительской зоне, замедляют работу службы имен, хотя клиенты в конце концов все равно отправляются на один из функционирующих серверов. Если ни один из серверов, перечисленных в родительской зоне, не существует в дочерней зоне, возникает так называемое некорректное делегирование (см. раздел 16.11).

Дополнительные серверы в дочерней зоне допускаются, поскольку хотя бы один из дочерних серверов имеет запись NS в родительской зоне. Проверьте делегирование с помощью команд **dig** или **drill**, чтобы убедиться, что оно определяет корректный набор серверов (см. раздел 16.4).

Записи A

Записи A (address — адрес) являются основной частью базы данных DNS. Они обеспечивают перевод имен компьютеров в IP-адреса. Для каждого из сетевых интерфейсов компьютера должна существовать одна запись A. Она имеет следующий формат.

имя_хоста [ttl] [IN] A *IP-адрес*

Рассмотрим пример.

```
ns1         IN  A   63.173.189.1
```

В этом примере поле *имя_хоста* не завершается точкой, поэтому сервер имен добавляет домен, заданный по умолчанию, и образует полностью определенное имя "ns1.atrust.com.". Эта запись связывает данное имя с IP-адресом 63.173.189.1.

Записи AAAA

Записи AAAA (address — адрес) представляют собой эквивалент записей A в протоколе IPv6. Эти записи не зависят от транспортного протокола, с помощью которого они доставляются. Публикация записей IPv6 в ваших DNS-зонах не означает, что вы должны отвечать на DNS-запросы с помощью протокола Iv6.

Запись AAAA имеет следующий формат.

имя_хоста [ttl] [IN] AAAA *IP-адрес*

Рассмотрим пример.

```
f.root-servers.net.         IN  AAAA   2001:500:2f::f
```

Каждая часть адреса, завершающаяся двоеточием, состоит из четырех шестнадцатеричных цифр с пропущенными ведущими нулями. Два соседних двоеточия означают "достаточное количество нулей для заполнения полного адреса IPv6, состоящего из 128 бит".

Записи PTR

Записи PTR обеспечивают обратный перевод IP-адресов в доменные имена. Как указывалось ранее, записи обратного отображения существуют в домене in-addr.arpa и именуются байтами IP-адреса, следующими в обратном порядке. Например, зона для подсети 189 в данном примере имеет имя 189.174.63.in-addr.arpa.

Общий формат записи PTR таков.

```
адрес [ttl] IN PTR имя_хоста
```

Например, запись PTR в зоне 189.173.63.in-addr.arpa, соответствующая приведенной выше записи A для хоста ns1, будет иметь следующий вид.

```
1          IN  PTR   ns1.atrust.com.
```

Имя 1 не заканчивается точкой и потому является относительным. Вопрос: относительно чего? Не относительно домена atrust.com, поскольку для того, чтобы эта запись была точной, домен по умолчанию должен называться 189.173.63.in-addr.arpa.

Этого можно добиться, поместив записи PTR для каждой подсети в отдельный файл. Домен, связанный по умолчанию с этим файлом, задан в файле конфигурации сервера имен. Другой способ выполнить обратное преобразование — включить в файл зоны записи вида

```
1.189      IN  PTR   ns1.atrust.com.
```

с доменом по умолчанию 173.63.in-addr.arpa. В некоторых организациях все записи обратного преобразования помещаются в один файл, а подсеть задается посредством директивы $ORIGIN. Обратите внимание на то, что имя хоста ns1.atrust.com должно заканчиваться точкой, иначе к нему будет добавлена строка 173.63.in-addr.arpa.

Поскольку atrust.com и 189.173.63.in-addr.arpa — разные области пространства имен DNS, они составляют две отдельные зоны. У каждой зоны должна быть своя запись SOA и свои записи о ресурсах. Помимо определения зоны in-addr.arpa, для каждой реальной сети нужно также задать зону, которая охватывала бы интерфейс обратной связи (127.0.0.0), по крайней мере при работе с сервером BIND. Пример показан в разделе 16.8.

🕮 Дополнительную информацию о подсетях см. в разделе 13.4.

Описанные привязки прекрасно работают, когда маски подсетей проходят по границе байтов. Но как выполнять обратные преобразования для подсети вида 63.173.189.0/26, где последний байт может находиться в одной из четырех подсетей: 0-63, 64-127, 128-191 и 192-255? В документе RFC2317 описан остроумный прием, основанный на применении записей CNAME.

Важно, чтобы записи A соответствовали записям PTR. Несовпадение и отсутствие последних приводит к ошибкам аутентификации, в результате чего система замедляет работу. Это само по себе неприятно, но еще хуже то, что появляется почва для атак типа на основе отказа от обслуживания, которые направлены на любое приложение, требующее соответствия записям A при обратном преобразовании.

Для протокола IPv6 информация обратного отображения, соответствующая адресу AAAA, представляет собой запись PTR в домене верхнего уровня ip6.arpa.

Формат "nibble" меняет адресную запись AAAA на обратную, расширяя каждый фрагмент адреса, разделенный двоеточием, до четырех шестнадцатеричных цифр, а затем меняя порядок следования этих цифр и приписывая ip6.arpa в конце. Например, запись PTR, которая соответствует нашей записи AAAA из предыдущего примера, имеет вид

```
f.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.f.2.0.0.0.0.5.0.1. 0.0.2.ip6.arpa.
    PTR f.root-servers.net.
```

Эта строка была разделена на две, чтобы информация поместилась на странице. К сожалению, это не очень удобно для системного администратора, который должен вводить, отлаживать или даже анализировать такие данные. Конечно, в ваших реальных файлах зоны DNS оператор $ORIGIN может скрыть некоторые сложности.

Записи MX

Записи MX (mail exchanger — обмен почтой) используются системой электронной почты для более эффективной маршрутизации почтовых сообщений. Запись MX подменяет адресата сообщения, в большинстве случаев направляя сообщение концентратору электронной почты на сервере получателя, а не прямо на его рабочую станцию.

🕮 Записи MX определены в документе RFC1035 (1987).

Запись MX имеет следующий формат.

```
имя [ttl] [IN] MX приоритет хост ...
```

Записи, приведенные ниже, направляют почту, предназначенную для получателя user@server.atrust.com, на машину mail.server.atrust.com, если она включена и доступна.

```
somehost  IN  MX  10 piper mailserver.atrust.com.
          IN  MX  20 mail-relay3.atrust.com.
```

Сначала опрашиваются хосты с более низким числовым значением приоритета: число 0 имеет самый высокий приоритет , а 65535 — самый низкий.

Записи MX полезны во многих ситуациях:

- если в системе есть центральный концентратор почты;
- если вы хотите фильтровать почту от спама и вирусов перед ее доставкой;
- если хост-адресат выключен;
- если адресат не подключен к Интернету;
- если администратор локальной сети лучше знает, куда следует рассылать сообщения (т.е. всегда).

Компьютер, принимающий почту для другого хоста, должен настроить свою программу пересылки соответствующим образом. Как задать такую конфигурацию для программы **sendmail** и почтовых серверов Postfix, показано в разделах 20.10 и 20.15 соответственно.

В базе данных DNS иногда можно встретить метазаписи MX.

```
*           IN  MX   10 mailserver.atrust.com.
```

На первый взгляд кажется, что такая запись позволяет избежать многократного ввода данных и является стандартной для всех хостов. Однако метасимвол интерпретируется совсем не так, как можно ожидать. Он соответствует полю имени, которое еще *не* было указано в явном виде в других записях о ресурсах.

Следовательно, нельзя использовать звездочку с целью задания стандартного значения для всех своих компьютеров. С ее помощью можно задать стандартные имена

для "чужих" компьютеров. В результате на концентратор будет поступать масса почтовых сообщений, тут же отвергаемых по той причине, что имя хоста, обозначенное звездочкой, не принадлежит домену. Поэтому необходимо избегать применения метасимвола в записях MX.

Записи CNAME

Записи CNAME (canonical name — каноническое имя) позволяют назначать хосту дополнительные мнемонические имена. Псевдонимы широко применяются для закрепления за компьютером какой-либо функции либо просто для сокращения его имени. Реальное имя иногда называют *каноническим*. Приведем несколько примеров.

```
ftp          IN  CNAME   anchor
kb           IN  CNAME   kibblesnbits
```

🕮 Записи CNAME определены в документе RFC1035 (1987).

Запись CNAME имеет следующий формат.

псевдоним [*ttl*] [IN] CNAME *имя_хоста*

Обнаружив запись CNAME, программное обеспечение системы DNS перестает посылать запросы по мнемоническому имени и переключается на реальное имя. Если у компьютера есть запись CNAME, то другие записи для хоста (A, MX, NS и др.) должны ссылаться на его реальное, а не мнемоническое имя.[12]

Длина цепочки записей CNAME не должна составлять больше восьми элементов. Цепочка — это когда одна запись CNAME ссылается на другую, а та — на третью и т.д. Последним, восьмым, элементом должно быть реальное имя хоста. При использовании записей CNAME запись PTR должна ссылаться на реальное имя, а не псевдоним.

Для того чтобы избежать использования записей CNAME, можно опубликовать записи A как для реальных имен, так и для псевдонимов. Эта конфигурация работает немного быстрее, поскольку в ней отсутствует дополнительный уровень косвенной адресации.

Документ RFC1033 требует, чтобы "вершина" зоны (иногда называемая "корневым" или "голым" доменом) распознавала одну или несколько записей A (и/или AAAA). Запрещается использование записи CNAME. Другими словами, вы можете сделать так:

```
www.yourdomain.com.          CNAME some-name.somecloud.com.
```

но не так:

```
yourdomain.com.              CNAME some-name.somecloud.com.
```

Это ограничение потенциально опасно, особенно если вы хотите, чтобы вершина ссылалась куда-то в сеть облачного провайдера и при этом IP-адрес сервера мог изменяться. В этой ситуации статическая запись A не является надежным вариантом.

Чтобы устранить проблему, необходимо использовать управляемого поставщика услуг DNS (например, AWS Route 53 или CloudFlare), использующего какую-то систему для обхода требований RFC1033. Как правило, эти системы позволяют указывать свои записи вершины способом, подобным CNAME, но фактически они обслуживают записи A во внешнем мире. При этом работу по синхронизации записей A с фактической целью выполняет поставщик услуг DNS.

[12]Это правило, касающееся записей CNAME, явным образом ослаблено в спецификациях DNSSEC, которые добавили цифровые подписи к каждому набору записей о ресурсах DNS.

Записи SRV

Эти записи определяют местонахождение служб в пределах домена. Например, благодаря записи SRV можно запросить удаленный домен и узнать имя его FTP-сервера. Раньше в подобной ситуации приходилось действовать наугад в надежде на то, что администратор удаленного домена, следуя традиционной практике, добавил запись CNAME для имени "ftp" в базу данных DNS.

🕮 Записи SRV определены в документе RFC2782 (2000).

Гораздо разумнее применять для этих целей записи SRV. С их помощью администраторам намного удобнее менять адреса служб и контролировать их использование. Однако требуется, чтобы сами клиенты знали, как найти и проанализировать записи SRV, поэтому эффект от их применения пока не столь ощутим.

Записи SRV напоминают обобщенные записи MX с дополнительными полями, которые позволяют администратору DNS управлять внешними соединениями и распределять нагрузку на сервер. Формат записей выглядит следующим образом.

служба.протокол.имя [*ttl*] [IN] SRV *приоритет вес порт сервер*

Аргумент *служба* представляет собой имя службы, определенное в базе данных IANA (Internet Assigned Numbers Authority — Агентство по выделению имен и уникальных параметров протоколов Интернета). Получить доступ к этой базе данных можно по адресу iana.org/numbers.htm. Аргумент *протокол* должен быть равен tcp либо udp. Аргумент *имя* — это домен, на который ссылается запись SRV. Аргумент *приоритет* имеет тот же смысл, что и в записи MX. Аргумент *вес* используется для распределения нагрузки между несколькими серверами, *порт* — это номер порта, на котором выполняется служба, а *сервер* — имя сервера, предоставляющего данную услугу. Для того чтобы избежать повторного обращения, в ответ на запрос к записи SRV обычно возвращается запись A сервера.

Если вес равен 0, то специального распределения нагрузки не требуется. Имя сервера, равное '.', означает, что служба на данном хосте недоступна.

Ниже показан пример из документа RFC2782, адаптированный для домена atrust.com.

```
_ftp.tcp        SRV  0  0  21   ftp-server.atrust.com.

; одна четверть соединений обслуживается старым компьютером,
; а три четверти - новым
_ssh.tcp        SRV  0  1  22   old-slow-box.atrust.com.
                SRV  0  3  22   new-fast-box.atrust.com.

; основной сервер доступен через порт 80,
; а резервный -- через порт 8000 на новом компьютере
_http.tcp       SRV  0  0  80   www-server.atrust.com.
                SRV  10 0  8000 new-fast-box.atrust.com.

; в адресной строке можно указывать как http://www.atrust.com,
; так и http://atrust.com
_http.tcp.www   SRV  0  0  80   www-server.atrust.com.
                SRV  10 0  8000 new-fast-box.atrust.com.

; все остальные службы блокированы
*._tcp          SRV  0  0  0    .
*._udp          SRV  0  0  0    .
```

В этом примере демонстрируется использование аргументов *вес* (служба SSH) и *приоритет* (служба HTTP). Задействованы оба сервера SSH, причем нагрузка между ними распределяется в соответствии с производительностью серверов. Все остальные службы заблокированы, включая службы для протоколов TCP и UDP. Однако тот факт, что эти службы не доступны в системе DNS, не означает, что они на самом деле не выполняются, просто вы не можете найти их с помощью системы DNS.

Записи TXT

Эти записи добавляют в базу данных DNS произвольный текст. Например, в нашей базе данных имеется следующая запись, идентифицирующая организацию.

```
IN          TXT  "Applied Trust Engineering, Boulder, CO, USA"
```

Эта запись стоит непосредственно после записей SOA и NS зоны `atrust.com`, наследуя от них поле *имя*.

🕮 Записи TXT определены в документе RFC1035 (1987).

Запись TXT имеет такой формат.

имя [*ttl*] [IN] TXT *информация* ...

Все информационные элементы должны быть заключены в кавычки. Это может быть как одна, так и несколько строк, каждая из которых взята в кавычки. Будьте внимательны: одна пропущенная закрывающая кавычка может привести к разрушению базы данных DNS, поскольку все последующие записи, вплоть до очередной кавычки, загадочным образом исчезнут.

Как и у других записей о ресурсах, у записей TXT нет внутреннего порядка, поэтому серверы возвращают их в произвольном порядке. Для того чтобы закодировать длинные элементы, такие как адреса, следует использовать длинные строки, а не набор нескольких записей TXT.

Поскольку записи TXT не имеют конкретного формата, их иногда используют для тестирования новых типов записей в системе DNS без изменения самой системы DNS.

Записи SPF, DKIM и DMARC

Записи SPF (Sender Policy Framework), DKIM (DomainKeys Identified Mail) и DMARC (Domain-based Message Authentication, Reporting, and Conformance) представляют собой стандарты, которые пытаются остановить постоянно растущий в Интернете поток несанкционированной коммерческой электронной почты (т.е. спама). Каждая из этих систем распределяет информацию о борьбе со спамом через DNS в виде записей TXT, поэтому они не являются истинными типами записей DNS.[13] По этой причине мы рассматриваем данные системы в разделе 18.4.

Записи о ресурсах DNSSEC

В настоящее время со спецификациями DNSSEC ассоциируются шесть типов записей.

Записи типов DS и DNSKEY предназначены для хранения разнообразных ключей и их отпечатков. Записи RRSIG содержат подписи других записей в зоне (по существу, наборов записей). И наконец, записи типов NSEC и NSEC3 позволяют серверам DNS подпи-

[13] Это немного не так. Для SPF существует определенный тип записи DNS; однако предпочтительной является версия записи TXT.

сывать несуществующие записи, обеспечивая криптографическую безопасность при отрицательных ответах на запросы. Эти шесть записей отличаются от большинства других записей тем, что они генерируются программами, а не набираются вручную.

Спецификации DNSSEC представляют собой большую тему, заслуживающую отдельного изучения, поэтому мы обсудим записи DNSSEC и их использование в разделе 16.10.

16.6. Программное обеспечение BIND

BIND, система Berkeley Internet Name Domain, представляет собой пакет программного обеспечения с открытым исходным кодом от консорциума Internet Systems Consortium (ISC), который реализует системы DNS для Linux, UNIX, MacOS и Windows. Четыре основных варианта BIND — BIND 4, BIND 8, BIND 9 и BIND 10 — в настоящее время разрабатываются консорциумом ISC. В этой книге мы рассматриваем только версию BIND 9.

Компоненты системы BIND

В пакет BIND входит четыре основных компонента:

- демон сервера имен **named**, отвечающий на запросы;
- библиотечные функции распознавания, контактирующие с серверами распределенной базы данных DNS от имени пользователей;
- утилиты **nslookup**, **dig** и **host**, позволяющие выполнять DNS-запросы из командной строки;
- программа **rndc** для дистанционного управления демоном **named**.

Первоочередной задачей системного администратора, работающего с пакетом BIND, является упорядочение многочисленных опций и функциональных возможностей, предоставляемых пакетом BIND, а также выбор наиболее подходящих для текущей ситуации.

Файлы конфигурации

Полная конфигурация демона **named** включает его конфигурационный файл (**named.conf**), файлы данных зоны, содержащие адресные привязки для каждого хоста, и файл подсказок для корневого сервера имен. Авторитетные серверы должны иметь файл конфигурации и файлы данных зоны для каждой зоны, относительно которых они являются главными серверами. Кеширующие серверы должны иметь файл конфигурации и файл подсказок для корневого сервера имен.

Файл конфигурации демона **named** имеет специфический формат; все остальные файлы представляют собой коллекции отдельных записей данных DNS, рассмотренных в разделе 16.4.

В конфигурационном файле **named.conf** задается роль хоста (главный, подчиненный, тупиковый или только кеширующий) и определяется способ, которым он должен получать копию записей о данных каждой зоны, которую он обслуживает. Здесь же приводятся всевозможные параметры — как глобальные, связанные с работой самого демона **named**, так и локальные, связанные с серверами и зонами и относящиеся только к части трафика DNS.

Файл конфигурации состоит из набора инструкций, каждая из которых оканчивается точкой с запятой и описывается в последующих разделах книги. К сожалению, формат

файла довольно "хрупкий": достаточно одной пропущенной точки с запятой, чтобы все перестало работать.

Комментарии допускаются везде, где могут стоять пробелы. Поддерживаются комментарии в стиле языков C, C++ и командного интерпретатора, но лучше выбрать один стиль и придерживаться только его.

```
/* Это комментарий, который может занимать несколько строк. */
// Весь текст до конца строки является комментарием.
# Весь текст до конца строки является комментарием.
```

Каждая инструкция начинается с ключевого слова, определяющего ее тип. Может присутствовать несколько инструкций одного типа, за исключением `options` и `logging`. Отдельные инструкции, а также их части могут отсутствовать; в этом случае будут приняты установки по умолчанию. Список поддерживаемых инструкций приведен в табл. 16.4.

Таблица 16.4. Инструкции, используемые в файле `named.conf`

Инструкция	Назначение
`include`	Подключает внешний файл
`options`	Задает глобальные параметры конфигурации сервера имен, а также установки по умолчанию
`acl`	Формирует списки управления доступом
`key`	Определяет параметры аутентификации
`trusted-keys`	Задает заранее установленные ключи шифрования
`server`	Задает параметры сервера
`masters`	Определяет список главных серверов для тупиковых и подчиненных зон
`logging`	Устанавливает категории журнальных сообщений и каналы их распространения
`statistics-channels`	Выводит текущую статистику в виде XML-файла
`zone`	Определяет зону записей о ресурсах
`controls`	Определяет, как утилита **`ndc`** будет управлять сервером имен **`named`**
`view`	Определяет представление данных зоны
`lwres`	Конфигурирует сервер имен **`named`** в качестве упрощенного распознавателя

Прежде чем приступать к описанию этих инструкций и их использования для конфигурирования демона имен **`named`**, необходимо рассказать о специальной структуре данных, используемой во многих инструкциях, — *список соответствия адресов* (address match list). Этот список является обобщением понятия IP-адреса и может включать:

- IP-адрес, либо IPv4, либо IPv6 (например, `199.165.145.4` или `fe80::202:b3ff:fe1e:8329`);
- адрес сети с маской CIDR[14] (например, `199.165/16`);
- имя ранее определенного списка управления доступом (задается с помощью инструкции `acl`);
- криптографический ключ аутентификации;
- оператор отрицания `!`.

[14]Сетевые маски CIDR описываются в разделе 13.4.

Списки соответствия адресов используются в качестве аргументов инструкций и опций. Приведем ряд примеров.

```
{ ! 1.2.3.13; 1.2.3/24; };
{ 128.138/16; 198.11.16/24; 204.228.69/24; 127.0.0.1; };
```

В первом списке из числа адресов исключается хост 1.2.3.13, но разрешаются другие адреса в сети 1.2.3.0/24. Во втором списке указаны сети, закрепленные за университетом штата Колорадо. Фигурные скобки и завершающая точка с запятой не являются частью списка; они относятся к инструкции, в которой содержится список.

Когда IP-адрес или адрес сети проверяется на соответствие списку, содержимое списка просматривается по очереди до тех пор, пока не будет найдено совпадение. Порядок элементов списка важен, так как ищется первое совпадение. Например, если в первом из показанных выше списков поменять элементы местами, результат не будет достигнут, поскольку адрес 1.2.3.13 удовлетворит первому условию (сетевому адресу 1.2.3.0/24) и второе условие никогда не будет проверяться.

Перейдем к изучению инструкций! Некоторые из них лаконичны и ясны, а для описания других может потребоваться отдельная глава.

Инструкция include

Когда конфигурационный файл становится слишком большим, можно разбить его на части, поместив их в отдельные файлы. Дополнительные компоненты подключаются к файлу **named.conf** посредством инструкции include.

```
include "путь";
```

Если указан относительный *путь*, то он добавляется к имени каталога, заданному в параметре directory. Очень часто инструкцию include применяют для подключения файлов, содержащих криптографические ключи, которые должны быть недоступными для посторонних. Для того чтобы не запрещать доступ ко всему файлу **named.conf**, ключи хранят в отдельных файлах, которые может читать лишь демон **named**. Эти файлы затем включаются в файл **named.conf**.

Многие организации размещают инструкции zone в отдельных файлах, а затем с помощью инструкции include объединяют их. Это позволяет отделить части конфигурации, являющиеся относительно статичными, от частей, подвергающихся частым изменениям.

Инструкция options

Эта инструкция задает глобальные параметры конфигурации, часть из которых впоследствии может быть переопределена для конкретных зон или серверов. Общий формат инструкции имеет следующий вид.

```
options {
  параметр;
  параметр;
    ...
};
```

Если в файле **named.conf** нет инструкции options, принимаются значения по умолчанию.

В пакете BIND существует много параметров, даже слишком много. В версии 9.9 их более 170, что очень затрудняет их изучение. Полный список параметров можно найти в документации. К сожалению, ходят слухи, что разработчики пакета BIND подумывают

о том, чтобы удалить некоторые параметры, которые оказались неудачными или больше не нужны. Это будет ударом для организаций, которые все же используют такие параметры и нуждаются в них. Ниже будут описаны лишь те параметры, которые рекомендуется задавать. (Мы опросили разработчиков пакета BIND и учли их мнения и советы.)

Для более полного изучения параметров рекомендуем обратиться к одной из посвященных системе DNS и пакету BIND книг, упомянутых в конце главы. Кроме того, советуем обратиться к документации, сопровождающей пакет BIND. Все параметры, включая их синтаксис и значения по умолчанию, описаны в документе **ARM**, находящемся в каталоге **doc** и входящем в дистрибутивный набор пакета. Полный список параметров содержится в файле **doc/misc/options**.

По мере изложения будем сопровождать описание параметров краткими комментариями. Значения по умолчанию указываются в квадратных скобках после самого параметра. В большинстве случаев значения по умолчанию вполне приемлемы. Сами параметры перечисляются без определенного порядка.

🕮 Местоположение файлов

```
directory "путь";                    [каталог запуска сервера]
key-directory "путь";                [совпадает с параметром directory]
```

Инструкция `directory` задает каталог, в который должен перейти демон **named**. Когда в конфигурационном файле встречаются относительные пути к файлам, они интерпретируются относительно этого каталога. Параметр *путь* должен быть абсолютным. В этот каталог помещаются все выходные файлы (отладочные, статистические данные и т.д.). Параметр `key-directory` определяет место хранения криптографического ключа. Он не должен быть доступен для посторонних.

Мы рекомендуем хранить все конфигурационные файлы пакета BIND (кроме **named.conf** и **resolv.conf**) в подкаталоге каталога **/var** (или любого другого каталога, где содержатся конфигурационные файлы других программ). Мы предпочитаем каталоги **/var/named** или **/var/domain**.

🕮 Идентификация сервера имен

```
version "строка"             [реальный номер версии сервера]
hostname "строка"            [реальное имя сервера]
server-id "строка"           [реальный номер версии сервера]
```

Строка `version` идентифицирует версию программного обеспечения сервера имен, а строка `hostname` — сам сервер, как и строка `server-id`. Эти параметры позволяют скрывать истинные значения. Каждый из них позволяет записывать данные записи `TXT` в класс CHAOS, где любопытные хакеры могут найти их с помощью команды **dig**.

Параметры `hostname` и `server-id` были включены относительно недавно и предназначены для дублирования экземпляров корневого и общих доменов верхнего уровня при альтернативной маршрутизации.

🕮 Синхронизация зон

```
notify yes | master-only | explicit | no;                    [yes]
also-notify id_адреса_серверов;                              [пусто]
allow-notify список_соответствующих_адресов;                 [пусто]
```

Параметры `notify` и `also_notify` применяются только к главным серверам, а параметр `allow-notify` — только к подчиненным.

В ранних версиях сервер BIND синхронизировал файлы зон между главным и подчиненным серверами только по истечении тайм-аута для обновления, заданного в записях зоны SOA. В настоящее время главный сервер **named** автоматически уведомляет подчиненные серверы при перезагрузке соответствующей базы данных зоны, если параметр `notify` равен `yes`. В ответ на полученное уведомление подчиненные серверы связываются с главным сервером, чтобы проверить, изменились ли файлы, и обновляют свои копии данных.

Параметр `notify` может устанавливаться как на глобальном уровне, так и на уровне отдельных зон. Это позволяет быстрее обновлять файлы зон при внесении изменений. По умолчанию каждый авторитетный сервер посылает обновления всем другим авторитетным серверам (такой способ Роль Викси (Paul Vixie) назвал "разбрызгиванием"). Если параметр `notify` установлен равным `master-only`, то "разговорчивость" сервера ограничена и уведомления посылаются только подчиненным серверам зон, по отношению к которым данный сервер является главным. Если параметр `notify` установлен равным `explicit`, то демон **named** уведомляет только серверы, указанные в разделе `also-notify`.

🕮 Более подробно тупиковые зоны обсуждаются в подразделе "Инструкция **zone**".

Демон **named** выясняет, какие компьютеры являются подчиненными серверами зоны, просматривая записи NS этой зоны. При наличии параметра `also-notify` будут также уведомляться дополнительные серверы, которые не зарегистрированы посредством записей NS. Подобный прием используется, если в организации имеются внутренние серверы. В списке `also-notify` не нужно указывать тупиковые серверы: их интересуют только записи NS, поэтому они могут участвовать в обычном цикле обновления.

В списке `also-notify` должны присутствовать только IP-адреса и, возможно, порты. Раздел `also-notify` следует использовать в файле конфигурации вторичного сервера только в том случае, если вы хотите, чтобы их уведомлял другой сервер имен, отличный от главного.

Для серверов с несколькими интерфейсами дополнительные параметры задают IP-адреса и порты, предназначенные для исходящих уведомлений.

🕮 Рекурсия запросов

```
recursion yes | no;                                    [yes]
allow-recursion { список_соответствия_адресов }; [все хосты]
```

Параметр `recursion` указывает на то, должен ли демон **named** обрабатывать запросы пользователей рекурсивно. Данный параметр можно задать для авторитетного сервера своей зоны, но мы не рекомендуем это делать. Лучше всего разделить авторитетные и кеширующие серверы.

Если сервер имен должен быть рекурсивным, то установите параметр `recursive` равным `yes` и включите раздел `allow-recursion`, чтобы демон **named** мог отличать запросы, исходящие из вашей организации, от удаленных запросов. Демон **named** будет действовать рекурсивно для пользователей вашей организации и нерекурсивно — для всех остальных. Если ваш сервер имен рекурсивно обрабатывает все запросы, то он называется *открытым распознавателем* (open resolver) и может стать рефлектором для некоторого вида атак (см. документ RFC5358).

🕮 Использование кеширующей памяти

```
recursive-clients число; [1000]
max-cache-size число;    [неограничено]
```

Если ваш сервер имеет слишком большой объем трафика, то иногда необходимо использовать параметры `recursive-clients` и `max-cache-size`. Параметр `recursive-clients` управляет многочисленными рекурсивными процессами поиска, которые сервер выполняет одновременно. Каждый из них требует около 20 двоичных килобайт памяти. Параметр `max-cache-size` ограничивает объем памяти сервера, который будет использоваться для кеширования ответов на запросы. Если объем кеша увеличивается слишком сильно, то демон **named** удаляет записи до истечения их периода TTL, чтобы предотвратить превышение лимита.

🕮 Использование IP-порта

```
use-v4-udp-ports [ range начало конец];          [range 1024 65535]
use-v6-udp-ports [ range начало конец];          [range 1024 65535]

avoid-v4-udp-ports [ список_портов ];            [пусто]
avoid-v6-udp-ports [ список_портов ];            [пусто]

query-source-v4 v4-адрес [ порт ];  [любой] #ВНИМАНИЕ, не используйте порт
query-source-v6 v6-адрес [ порт ];  [любой] #ВНИМАНИЕ, не используйте порт
```

Порты отправителя стали играть важную роль в системе DNS после обнаружения ошибки протокола DNS, выявленной Дэном Камински (Dan Kaminsky). Этот недостаток допускает атаку кеша DNS, когда серверы имен используют предсказуемые порты источника и идентификаторы запросов. Параметры `use-` и `avoid-` для портов UDP в сочетании с изменениями программного обеспечения демона **named** смягчили последствия таких атак.

В прошлом некоторые системные администраторы задавали конкретный номер порта отправителя и поэтому были вынуждены настраивать свои брандмауэры так, чтобы они распознавали его и принимали UDP-пакеты только для него. Однако после сообщения Каминского этот прием стал небезопасным. Не используйте параметр `query-address` для того, чтобы задать фиксированный порт отправителя для запросов DNS, иначе вы не сможете воспользоваться "защитой Каминского", которая предусматривает использование большого количества случайных портов.

Значения, задаваемые по умолчанию для параметров `use-v*`, не требуют изменений. Если ваш брандмауэр блокирует некоторые порты из заданного диапазона (например, порт 2049 для системы RPC), то может возникнуть небольшая проблема. Когда ваш сервер имен посылает запрос, используя в качестве источника один из заблокированных портов, брандмауэр блокирует ответ и сервер имен в конце концов прекращает ожидание и посылает запрос снова. Это не фатально, но раздражает.

Во избежание этого следует использовать параметр `avoid-v*`, заставляющий систему BIND избегать заблокированных портов. Любые порты UDP с крупными номерами, заблокированные вашим брандмауэром, должны быть включены в этот список.[15] Если вы обновляете свой брандмауэр в ответ на опасную атаку, то не забудьте обновить и список портов.

Параметры `query-source` позволяют указывать IP-адреса, предназначенные для исходящих запросов. Например, может возникнуть необходимость использовать конкрет-

[15] Некоторые брандмауэры запоминают состояние и могут распознавать ответ системы DNS на запрос, посланный секунду назад. Таким серверам не требуется помощь, обеспечиваемая параметрами `avoid-v*-udp-ports`.

ный IP-адрес, чтобы пройти через брандмауэр или отличить внутреннее представление от внешнего.

🕮 Использование переадресации

```
forwarders { ip_адрес; ip_адрес; ... };    [пустой список]
forward only | first;                      [first]
```

Вместо того чтобы каждый сервер имен выполнял собственные внешние запросы, можно сделать один или несколько серверов *переадресующими*. В такой конфигурации обычный сервер просматривает в своем кеше записи, для которых он авторитетен, и, если не находит ответ, посылает запрос переадресующему серверу. Последний создает кеш, которым пользуются все остальные серверы. Назначение происходит неявно — в файле конфигурации сервера нет никакой информации о том, что он является переадресующим.

В параметре `forwarders` приводится список IP-адресов компьютеров, являющихся переадресующими серверами. Они опрашиваются по очереди. При использовании переадресующего сервера традиционная процедура поиска домена, которая начинается с корневого сервера, а затем продолжается по цепочке отсылок, нарушается, поэтому нужно внимательно следить за тем, чтобы не возникало циклов переадресации.

Сервер, для которого установлен атрибут `forward only`, опрашивает переадресующие серверы и кеширует их ответы самостоятельно. Если переадресующий сервер не ответит, запрос потерпит неудачу. Сервер с атрибутом `forward first` связывается с переадресующими серверами, но при необходимости может обработать запрос самостоятельно.

Поскольку у параметра `forwarders` нет значения по умолчанию, переадресация происходит только в том случае, когда она задана явно. Включать переадресацию можно либо глобально, либо в пределах отдельных зон.

🕮 Разрешения

```
allow-query { список_соответствия_адресов };          [все хосты]
allow-query-cache { список_соответствия_адресов };    [все хосты]
allow-transfer { список_соответствия_адресов };       [все хосты]
allow-update { список_соответствия_адресов };         [нет]
blackhole { список_соответствия_адресов };            [нет]
```

Эти параметры позволяют указать, какие хосты (или сети) могут обращаться к серверу имен и запрашивать пересылку баз данных зон, а также динамически обновлять зоны. Списки соответствия адресов представляют собой слабый способ обеспечения безопасности и уязвимы для подделки IP-адресов, поэтому полагаться на них рискованно. Вероятно, не составит никакого труда заставить ваш сервер ответить на DNS-запрос, но следует избегать использования параметров `allow_update` и `allow_transfer`; вместо них следует применять криптографические ключи.

В списке `blackhole` перечислены серверы, которые никогда не "удостоятся внимания" демона **named**: он не будет принимать от них запросы и обращаться к ним за ответом.

🕮 Размеры пакетов

```
edns-udp-size номер;                [4096]
max-udp-size номер;                 [4096]
```

Все компьютеры в Интернете должны иметь возможность заново собирать фрагментированные UDP-пакеты, размер которых составляет 512 байт и меньше. Эти консервативные требования имели смысл в 1980-х годах, но в настоящее время такой размер

смехотворно мал. Современные маршрутизаторы и брандмауэры могут обрабатывать намного более крупные пакеты, но достаточно всего одного неправильного звена в цепочке IP-адресов, чтобы прервать весь путь.

Поскольку система DNS использует для запросов пакеты UDP, а ответы часто превышают 512 байт, ее администраторы иногда опасаются утери крупных UDP-пакетов. Если большой по размеру пакет был фрагментирован, а ваш брандмауэр пропускает только первый фрагмент, то адресат получит усеченный ответ и повторно пошлет запрос по протоколу TCP. Протокол TCP связан с намного более крупными затратами ресурсов, в то же время загруженные серверы в корневом узле и серверы TLD не должны увеличивать трафик по протоколу TCP из-за того, что у кого-то вышел из строя брандмауэр.

Параметр `edns-udp-size` задает размер буфера для повторной сборки, который сервер имен сообщает посредством протокола EDNS0, представляющего собой расширенный вариант протокола DNS. Параметр `max-udp-size` задает максимальный размер пакета, который сервер может реально посылать. Оба размера измеряются в байтах. Разумным диапазоном является 512–4096 байт.

🕮 Управление спецификациями DNSSEC

```
dnssec-enable yes | no;                          [yes]
dnssec-validation yes | no;                      [yes]
dnssec-must-be-secure домен yes | no;            [нет]
```

Эти параметры конфигурируют поддержку спецификаций DNSSEC. Обсуждение спецификаций DNSSEC и подробное описание их настроек приведены в разделе 16.10.

Параметр `dnssec-enable` на авторитетном сервере должен быть включен. Для рекурсивных серверов должны быть включены параметры `dnssec-enable` и `dnssec-validation`, а также указана точка доверия с помощью инструкции `trusted-key`.

По умолчанию параметры `dnssec-enable` и `dnssec-validation` включены. Это приводит к таким последствиям.

- Авторитетный сервер подписанной зоны с включенным режимом DNSSEC, отвечая на запрос, возвращает ответ, содержащий требуемые записи о ресурсах и их подписи.
- Авторитетный сервер подписанной зоны с выключенным режимом DNSSEC, отвечая на запрос, возвращает ответ, содержащий только требуемые записи о ресурсах, как это делалось до появления спецификаций DNSSEC.
- Авторитетный сервер неподписанной зоны возвращает ответ, содержащий только требуемые записи о ресурсах; подписи в него не включаются.
- Рекурсивный сервер посылает запросы от имени пользователей с включенным режимом DNSSEC.
- Рекурсивный сервер проверяет подписи, включенные в подписанные ответы, прежде чем возвращать данные пользователю.

Элемент конфигурации `dnssec-must-be-secure` позволяет указать, что вы будете принимать только безопасные ответы от конкретных доменов или, наоборот, вам все равно и вы согласны принимать небезопасные ответы. Например, вы можете задать значения `yes` для домена `important-stuff.mybank.com` и `no` — для домена `marketing.mybank.com`.

🕮 Статистика

```
zone-statistics yes | no       [no]
```

Этот элемент конфигурации заставляет демон **named** накапливать статистические данные как о зонах, так и в целом. Для вывода статистики на экран выполните команду **rndc stats**.

🕮 Настройка производительности

```
clients-per-query целое_число;       [10]  #Количество клиентов,
                                           #ожидающих один и тот же запрос
max-clients-per-query целое_число;  [100] #Максимально допустимое количество
                                           #клиентов до отказа сервера
datasize целое_число  [неограничено]       #Максимальное количество памяти,
                                           #которое может использовать сервер
files целое_число     [неограничено]       #Максимальное количество
                                           #одновременно открытых файлов
lame-ttl целое_число; [10min]              #Период времени до кеширования
                                           #сбойного сервера
max-acache-size целое_число  []            #Размер кеша для дополнительных
                                           #данных
max-cache-size целое_число   []            #Максимальный размер памяти
                                           #для кешированных ответов
max-cache-ttl целое_число    [1week]       #Максимальный период TTL для
                                           #кеширования позитивных данных
max-journal-size целое_число []            #Максимальный размер файла
                                           #для журнала транзакций
max-ncache-ttl целое_число   [3hrs]        #Максимальный период TTL для
                                           #кеширования негативных данных
tcp-clients целое_число;     [100]         #Максимально допустимое количество
                                           #TCP-клиентов
```

Этот длинный список элементов конфигурации можно использовать для настройки демона **named**, чтобы он хорошо работал на вашем аппаратном обеспечении. Мы не описываем их подробно, но если у вас возникнут проблемы с производительностью, сначала следует попробовать устранить их с помощью этих настроек.

Итак, мы завершаем обзор параметров! Перейдем к остальным инструкциям.

Инструкция acl

Список управления доступом — это просто именованный список соответствия адресов.

```
acl имя_списка {
    список_соответствия_адресов
};
```

Заданное имя можно указывать везде, где требуется список соответствия адресов.

Инструкция `acl` должна быть самой первой в файле **named.conf**, так что не пытайтесь ставить ее среди других объявлений. Файл **named.conf** читается за один проход, поэтому списки управления доступом должны быть определены до того, как на них встретится ссылка. Существует четыре стандартных списка соответствий:

- `any` — всем хостам;
- `localnets` — всем хостам локальной сети;
- `localhost` — самому компьютеру;
- `none` — ни одному хосту.

Сети, входящие в группу `localnets`, определяются адресами интерфейсов компьютера с учетом сетевых масок.

Инструкция key (TSIG)

Рассматриваемая инструкция определяет именованный ключ шифрования (т.е. пароль), используемый для аутентификации соединения между двумя серверами, например между главным и подчиненным серверами при передаче зоны или между сервером и процессом **rndc**, который им управляет. Подробнее о механизмах аутентификации, используемых в пакете BIND, речь пойдет в разделе 16.10. Здесь же мы лишь вкратце опишем суть процесса.

Для создания ключа нужно указать как алгоритм шифрования, так и общий секретный ключ, который представлен в виде строки, закодированной в формате Base64 (см. раздел 16.10).

```
key идентификатор_ключа {
    algorithm строка;
    secret строка;
};
```

Как и в случае списков управления доступом, идентификатор ключа должен быть определен в файле **named.conf** с помощью инструкции key до того, как встретится ссылка на ключ. Для того чтобы связать ключ с конкретным сервером, включите идентификатор ключа в список keys соответствующей инструкции server. Ключ используется как для проверки запросов, поступающих от сервера, так и для подписи ответов на эти запросы.

Секретная информация, принадлежащая группе лиц, не должна храниться в файле, доступном для внешнего мира. Для включения его в файл **named.conf** следует использовать инструкцию include.

Инструкция server

Демон **named** способен общаться с серверами, которые не используют последнюю версию пакета BIND или просто неправильно сконфигурированы. Инструкция server сообщает демону характеристики удаленных серверов. Она может заменять стандартные значения параметров конкретного сервера, хотя это не обязательно, пока вы не захотите сконфигурировать ключи для передачи зоны.

```
server ip-адрес {
    bogus yes | no;                                          [нет]
    provide-ixfr yes | no;                                   [да]
    request-ixfr yes | no;                                   [да]
    keys {идентификатор_ключа; идентификатор_ключа; ...}; [нет]
    transfer-source ip-адрес [порт];       [ближайший интерфейс]
    transfer-source-v6 ipv6-адрес [порт]; [ближайший интерфейс]
};
```

Посредством инструкции server можно переопределять значения глобальных конфигурационных параметров, относящихся к серверам. Просто укажите нужные параметры и их новые значения. Мы показали не все параметры инструкции server, а только самые необходимые. Полный список параметров можно найти в документации пакета BIND.

Если для сервера задан атрибут bogus, демон **named** не посылает ему запросы. Этот атрибут следует устанавливать для серверов, которые работают неправильно. Параметр bogus отличается от глобального атрибута blackhole тем, что подавляет только внешние запросы. В отличие от него, атрибут blackhole полностью исключает все формы взаимодействия с перечисленными серверами.

Сервер имен BIND, являющийся главным сервером динамически обновляемой зоны, выполняет инкрементные передачи зон, если атрибут `provide-ixfr` равен `yes`. Точно так же подчиненный сервер запрашивает инкрементные передачи зон от главного сервера, если атрибут `request-ixfr` равен `yes`. Динамическая система DNS обсуждается в разделе 16.9.

В списке `keys` задаются идентификаторы ключей, которые ранее были определены в инструкции `key` для использования в сигнатурах транзакций TSIG (см. раздел 16.10). К любому запросу, посылаемому удаленному серверу, будет добавлена сигнатура, зашифрованная посредством этого ключа. Запросы, поступающие от удаленного сервера, могут не иметь цифровой подписи, но если она есть, то будет проверена.

Атрибут `transfer-source` определяет IPv4- и IPv6-адреса интерфейса (и, возможно, порта), которые следует использовать в качестве адреса отправителя (порта) для запросов на передачу зоны. Эти атрибуты необходимы только тогда, когда система имеет несколько интерфейсов и IP-адрес удаленного сервера указан в списке `allow-transfer`.

Инструкция `masters`

Эта инструкция позволяет задавать имена одного или нескольких главных серверов, указывая их IP-адреса и криптографические ключи. Затем это имя можно использовать в разделе `masters` инструкций `zone` вместо повторения IP-адресов и ключей.

🕮 В каких случаях главных серверов бывает несколько? Ответ см. через несколько страниц.

Инструкция `masters` удобна, когда несколько подчиненных или тупиковых зон получают данные от одного и того же удаленного сервера. Если адреса и криптографические ключи удаленного сервера изменились, вы можете обновить инструкцию `masters`, которая их задает, а не изменять многочисленные инструкции `zone`.

Синтаксис этой инструкции выглядит следующим образом.

```
masters имя [ ip-адрес [port ip-порт] [key ключ]; ...]
```

Инструкция `logging`

Демон **named** в настоящее время заслуживает награды "За наиболее конфигурируемую подсистему журнальной регистрации на Земле". Система Syslog предоставляет программистам контроль над приоритетами сообщений, а системным администраторам — контроль над местом хранения этих сообщений. Но в рамках заданного приоритета администратор не может указать: "Это сообщение меня интересует, а это — нет". В систему BIND были добавлены категории, позволяющие классифицировать сообщения по типам, и каналы, расширяющие возможности в плане хранения сообщений. Категории назначаются программистом, а каналы — администратором.

Поскольку вопросы журнальной регистрации лежат несколько в стороне от нашего повествования (особенно если учесть объем материала), мы рассмотрим их позднее.

Инструкция `statistics-channels`

Эта инструкция `statistics-channels` позволяет соединиться с выполняемым демоном **named** с помощью браузера и просмотреть статистические показатели, которые на нем собраны. Поскольку статистические показатели сервера имен могут быть конфиденциальными, следует ограничить доступ к этим данным, открыв их только для дове-

ренных хостов вашей собственной организации. Синтаксис этой инструкции выглядит следующим образом.

```
statistics-channels {
    inet(ip-адрес | *) port порт# allow { список_соответствия_адресов } ;
    ...
}
```

В файл конфигурации можно включить несколько последовательностей `inet-port-allow`. По умолчанию IP-адреса заданы значением `any`, а порт — значением 80 (обычное значение для порта HTTP). Для использования каналов сбора статистики демон **named** следует компилировать с поддержкой библиотеки **libxml2**.

Инструкция zone

Это "сердце" файла **named.conf**, которое сообщает демону **named** о зонах, для которых он авторитетен, и задает параметры управления каждой зоной. Инструкция `zone` также используется для предварительной загрузки "подсказок" с корневого сервера ("подсказки" — это имена и адреса корневых серверов, участвующих в инициализации DNS-поиска).

Точный формат инструкции `zone` зависит от роли, которую демон **named** должен играть в отношении этой зоны. Возможными типами зоны являются `master`, `slave`, `hint`, `forward`, `stub` и `delegation-only`. Мы не будем рассматривать зоны типов `stub` (используется только в системе BIND) и `delegation-only` (применяется для прекращения использования записей с шаблонными символами в зонах верхнего уровня при извещении служб регистратора). Остальные типы зон описаны в последующих разделах.

Многие из рассмотренных ранее глобальных параметров могут быть частью инструкции `zone`, т.е. переопределять глобальные установки. Мы не будем повторно упоминать о них, за исключением тех параметров, которые наиболее часто используются.

Конфигурирование главного сервера зоны

Ниже показан формат инструкции `zone` для зоны, в которой демон **named** является главным сервером имен.

```
zone "имя_домена" {
    type master;
    file "путь";
};
```

Доменное имя в спецификации зоны всегда дается в двойных кавычках.

База данных зоны хранится на диске в текстовом файле, доступном для редактирования. Поскольку нет соглашения об именовании этого файла, в объявлении зоны должна присутствовать директива `file`. Файл зоны представляет собой набор записей о DNS-ресурсах; их формат описан в разделе 16.4.

В инструкции `zone` часто задается также ряд других серверных атрибутов.

```
allow-query { список_соответствия_адресов };     [все хосты]
allow-transfer { список_соответствия_адресов }; [все хосты]
allow-update { список_соответствия_адресов };    [ни один]
zone-statistics yes | no;                        [нет]
```

Параметры, связанные с управлением доступом, не являются обязательными, но мы рекомендуем их использовать. Они содержат либо IP-адрес, либо криптографический ключ TSIG. Как правило, криптографический ключ является более безопасным. Если для зоны поддерживаются динамические обновления, в параметре `allow-update` дол-

жен содержаться список хостов, от которых разрешено принимать данные. Динамические обновления применимы только в отношении главных зон; параметр `allow-update` не может присутствовать в объявлении подчиненной зоны. Убедитесь, что в списке указаны лишь локальные компьютеры (например, DHCP-серверы), а не весь Интернет[16].

Параметр `zone-statistics` заставляет демон **named** отслеживать статистику запросов-ответов, например точное число и общий процент отсылок, рекурсивных и ошибочных запросов.

При наличии столь большого числа параметров зоны (есть еще порядка сорока, о которых мы не упомянули) конфигурация зоны кажется довольно сложной. Однако существует возможность объявить главную зону, состоящую только из пути к файлу зоны. Ниже показан слегка модифицированный пример, взятый из документации.

```
zone "example.com" {
    type master;
    file "forward/example.com";
    allow-query { any; };
    allow-transfer { my-slaves; };
};
```

Имя `my-slaves` относится к списку управления доступом, определенному ранее.

Конфигурирование подчиненного сервера зоны

Формат инструкции `zone` для подчиненного сервера будет почти таким же, как и в случае главного сервера.

```
zone "имя_домена" {
    type slave;
    file "путь";
    masters { ip_адрес [port ip_порт] [key имя_ключа]; ... };
    allow-query { список_соответствия_адресов };    [все хосты]
};
```

Подчиненные серверы обычно хранят полную копию базы данных зоны. Директива `file` задает имя локального файла, в котором сохраняется реплицированная база данных. Получив новую копию данных зоны, сервер сохраняет ее в этом файле. В случае сбоя и перезагрузки сервера файл можно просто прочитать с диска, а не пересылать по сети.

Редактировать файл реплицированной базы данных не нужно, так как он управляется демоном **named**. Тем не менее имеет смысл просмотреть его, если есть подозрение, что в базу данных главного сервера закралась ошибка. Файл подчиненной зоны создается уже после того, как демон **named** интерпретировал исходные данные зоны и раскрыл все относительные имена. Когда в файле присутствуют имена наподобие

```
128.138.243.151.cs.colorado.edu.
anchor.cs.colorado.edu.cs.colorado.edu.
```

можно быть уверенным в том, что где-то пропущена завершающая точка.

В списке `masters` перечислены IP-адреса компьютеров, с которых может быть загружена база данных зоны. Она также может содержать имя списка главных серверов, определенных предыдущей инструкцией `masters`.

Ранее мы говорили, что только один компьютер может быть главным сервером зоны, так почему же разрешается список, состоящий из нескольких адресов? Во-первых,

[16]Необходима также входная фильтрация на брандмауэре (см. раздел 13.14), а еще лучше использовать технологию аутентификации TSIG.

у главного компьютера может быть несколько сетевых интерфейсов и, соответственно, несколько IP-адресов. Когда один из интерфейсов становится недоступным (проблемы с сетью или маршрутизацией), остаются в резерве другие интерфейсы. Следовательно, рекомендуется указывать все топологически различные адреса главного сервера.

Во-вторых, демону **named** в действительности все равно, откуда поступают данные зоны. Он так же легко загружает базу данных с подчиненного сервера, как и с главного. Этой особенностью демона можно воспользоваться для того, чтобы сделать подчиненный сервер, с которым легко устанавливается связь, резервной копией главного сервера. В любом случае IP-адреса проверяются по очереди до тех пор, пока не будет найден работающий сервер. Теоретически можно даже создать иерархию серверов, в которой главный сервер обслуживает несколько серверов второго уровня, а те, в свою очередь, обслуживают множество серверов третьего уровня.

Задание "подсказок" корневых серверов

Инструкция `zone` типа `hint` сообщает демону **named** местонахождение файла, из которого он может загрузить имена и адреса корневых серверов имен, чтобы предварительно заполнить свой кеш.

```
zone "." {
    type hint;
    file "путь";
};
```

"Подсказки" — это набор DNS-записей, в которых перечислены серверы корневого домена ("."). Они нужны для того, чтобы демон **named** знал, откуда начинать поиск доменов. Не имея "подсказок", демон знал бы только о тех доменах, которые сам обслуживает, а также об их поддоменах.

Когда демон **named** начинает работу, он повторно загружает подсказки с одного из корневых серверов. Следовательно, все будет в порядке, если файл подсказок содержит хотя бы одну запись с правильным и доступным корневым сервером. Для целей резервирования подсказки о корневом сервере также компилируются в демон **named**.

Файл "подсказок" чаще всего называется **root.cache**. В нем содержатся результаты ответов, которые можно получить, запросив у корневого сервера список серверов имен в домене ".". На самом деле вы можете сгенерировать файл подсказок, выполнив утилиту **dig**. Рассмотрим пример.

```
$ dig @f.root-servers.net . ns > root.cache
```

Обратите внимание на точку. Если сервер `f.root-servers.net` не отвечает, вы можете выполнить запрос, не указывая конкретный сервер.

```
$ dig . ns > root.cache
```

Результат будет аналогичный. Однако вы получите список корневых серверов от кеша локального сервера имен, а не от авторитетного сервера. Это должно быть удобно — даже если вы не перезагружали или перезапускали сервер имен год или два назад, он периодически обновляет свои записи о корневых серверах по истечении их периода TTL.

Задание зоны переадресации

Зона типа `forward` переопределяет глобальные параметры переадресации демона **named** (сначала опрашиваем корневой сервер, а потом следуем по цепочке отсылок) для конкретного домена.

```
zone "имя_домена" {
    type forward
    forward only | first;
    forwarders { ip_адрес; ip_адрес; ...}
};
```

Создавать такую зону имеет смысл, если у организации имеется деловой партнер и нужно направлять трафик непосредственно его серверам имен в обход стандартного пути распространения запросов.

Инструкция `controls` для команды `rndc`

Инструкция `controls` определяет, каким образом команда **rndc** будет управлять работой демона **named**. Эта команда может запускать и останавливать демон, выводить отчет о его состоянии, переводить демон в режим отладки и т.д. Будучи сетевой утилитой, она требует должной конфигурации, чтобы злоумышленники не могли через Интернет влиять на работу сервера имен. Формат инструкции `controls` имеет следующий вид.

```
controls {
    inet ip_адрес port порт allow { список_соответствия_адресов }
       keys {список_ключей}

}
```

Если параметр `port` пропущен, по умолчанию принимается порт 953.

Удаленное управление сервером имен кажется одновременно и удобной, и опасной процедурой. Аутентификация необходима, причем ключи, указанные в списке соответствия адресов, игнорируются, а используются только те ключи, которые перечислены в списке `keys`.

В системе BIND существует команда **rndc-confgen**, позволяющая сгенерировать ключ аутентификации, используемый в диалоге между командой **rndc** и демоном **named**. Это можно сделать двумя способами. Первый способ — заставить обе стороны загрузить ключ из общего конфигурационного файла (**/etc/rndc.key**) или поместить ключ в два разных файла (**/etc/rndc.conf** для **rndc** и **/etc/named.conf** для **named**). Второй способ сложнее, но он необходим, если демон **named** и команда **rndc** запускаются на разных компьютерах. Команда **rndc-confgen -a** задает ключи для доступа к локальному хосту.

При отсутствии инструкции `controls` в список соответствия адресов заносится адрес интерфейса обратной связи, а ключ ищется в файле **/etc/rndc.conf**. Поскольку аутентификация в этой версии пакета обязательна, команда **rndc** не сможет управлять работой демона **named** без ключа. Это кажется чрезмерным требованием, но подумайте вот о чем: даже если команда **rndc** работает только на хосте 127.0.0.1 и его адрес заблокирован для внешнего мира на брандмауэре, вы все равно выдаете всем локальным пользователям определенный кредит доверия, подразумевая, что они не будут делать ничего противозаконного с сервером имен. В то же время любой из них может подключиться с помощью утилиты **telnet** к управляющему порту и ввести команду **stop**. Чем вам не атака по типу отказа в обслуживании?

Ниже показаны результаты работы команды **rndc-confgen**, генерирующей 256-разрядный ключ (размер ключа объясняется тем, что он позволяет уместить результаты на странице!). Обычно вывод команды направляется не на экран, а в файл **/etc/rndc.conf**. Комментарии в нижней части включают строки, которые необходимо добавить в файл **named.conf**, чтобы демон **named** и команда **rndc** могли взаимодействовать.

```
% ./rndc-confgen -b 256
# Start of rndc.conf
key "rndc-key" {
    algorithm hmac-md5;
    secret "orZuz5amkUnEp52zlHxD6cd5hACldOGsG/elP/dv2IY=";
};

options {
   default-key "rndc-key";
   default-server 127.0.0.1;
   default-port 953;
};
# End of rndc.conf

# Use with the following in named.conf, adjusting the allow list
# as needed:
# key "rndc-key" {
#    algorithm hmac-md5;
#    secret "orZuz5amkUnEp52zlHxD6cd5hACldOGsG/elP/dv2IY=";
# };
#
# controls {
#    inet 127.0.0.1 port 953
#       allow { 127.0.0.1; } keys { "rndc-key"; };
# };
# End of named.conf
```

16.7. Расщепление DNS и инструкция view

Во многих организациях требуется, чтобы внутреннее представление сети отличалось от представления этой сети с точки зрения пользователей Интернета. Например, внутренние пользователи могут видеть все компьютеры зоны и лишь несколько разрешенных внешних серверов. Или другой вариант: внутреннее и внешнее представления одинаковы по охвату хостов, однако внутренним пользователям предоставляются дополнительные (либо отличающиеся) записи. В частности, записи MX, предназначенные для маршрутизации электронной почты, за пределами домена ссылаются на почтовый концентратор, а внутри домена — на отдельные рабочие станции.

🕮 Дополнительную информацию о пространствах частных адресов см. в разделе 13.3.

Расщепленная конфигурация DNS особенно удобна организациям, которые во внутренних сетях применяют IP-адреса из частного диапазона (RFC1918). Например, запрос об имени хоста с IP-адресом 10.0.0.1 не может быть обработан глобальной системой доменных имен, но он вполне допустим в контексте локальной сети. Среди запросов, поступающих на корневые серверы имен, 4-5% либо имеют исходный IP-адрес из частного диапазона, либо ссылаются на такой адрес. Ни на один из этих запросов нельзя ответить. Запросы обоих типов являются следствием неправильной конфигурации либо пакета BIND, либо "доменов" Microsoft.

Инструкция `view` содержит список адресов, определяющий, кто из клиентов имеет доступ к данному представлению, а также ряд параметров, применимых ко всем зонам в представлении, и, наконец, определения самих зон. Синтаксис инструкции имеет следующий вид.

```
view имя_представления {
   match-clients { список_соответствия_адресов };          [все хосты]
   match-destinations { список_соответствия_адресов };     [все хосты]
   match-recursive-only yes | no;                          [нет]
   параметр_представления;
   инструкция_zone; ...
}
```

Инструкция `view` всегда содержит атрибут `match-clients`, который фильтрует IP-адреса источников, указанные в пакете запроса, и обычно используется для обслуживания внутренних и внешних представлений данных DNS, принадлежащих организации. Для более качественного контроля можно также фильтровать адреса пунктов назначения и требовать рекурсивные запросы.

Атрибут `match-destinations` задает просмотр адресов назначения в пакете запроса и является полезным в компьютерах с несколькими интерфейсами, когда вы хотите обрабатывать разные данные DNS в зависимости от интерфейса, из которого поступил запрос. Атрибут `match-recursive-only` требует, чтобы запросы были рекурсивными, а также высылались разрешенными клиентами. Итеративные запросы позволяют просмотреть кеш сайта; этот атрибут предотвращает такие ситуации.

Представления просматриваются по порядку, поэтому инструкции `view` с самыми большими ограничениями доступа должны идти впереди. Зоны в разных представлениях могут иметь одинаковые имена. Представления налагают ограничение на структуру файла **`named.conf`**: если они присутствуют, *все* инструкции `zone` должны находиться в контексте представлений.

Ниже приводится взятый из документации к BIND 9 пример, имитирующий расщепление пространства DNS-имен на внутреннее и внешнее. В обоих представлениях задается одна и та же зона, но с разной базой данных.

```
view "internal" {
    match-clients { наши_сети; }; // только внутренние сети
    recursion yes;                // только для внутренних клиентов
    zone "example.com" {          // полное представление зоны
        type master;
        file "example-internal.db";
    };
};

view "external" {
    match-clients { any; };       // допускаются любые запросы
    recursion no;                 // рекурсия запрещена
    zone "example.com" {          // только "общедоступные" хосты
        type master;
        file "example-external.db";
    };
};
```

Если поменять порядок представлений, никто не сможет получить доступ к внутреннему представлению. Адреса внутренних хостов пройдут проверку на соответствие условию `any` в предложении `match-clients` внешнего представления до того, как начнет проверяться внутреннее представление.

Второй пример представления DNS, который будет рассмотрен ниже, демонстрирует еще один вид представлений.

16.8. Примеры конфигурации системы BIND

После подробного знакомства с файлом **named.conf** настало время рассмотреть ряд готовых примеров. В следующих подразделах мы опишем две типичные конфигурации:

- зона локального хоста;
- сервер небольшой компании, занимающейся вопросами безопасности и применяющей расщепленную конфигурацию DNS.

Зона локального хоста

Адрес 127.0.0.1 протокола IPv4 относится к самому локальному хосту и должен быть преобразован в имя "`localhost.`" [17] Одни организации преобразовывают адрес в имя "`localhost.`*`локальный_домен`*", а другие делают и то, и другое. Соответствующий адрес в протоколе IPv6 равен `::1`.

Если вы забыли задать конфигурацию зоны локального хоста, то ваша организация может прекратить посылать запросы корневым серверам на получение информации о локальном хосте. Корневые серверы получают так много таких запросов, что операторы предлагают просто добавить обобщенное отображение между локальным хостом и адресом 127.0.0.1. на корневом уровне. Другими необычными именами в популярной категории "фиктивные домены TLD" являются `lan`, `home`, `localdomain` и `domain`.

Прямое преобразование имени локального хоста можно определить в файле зоны прямого преобразования (forward zone) для домена (с помощью соответствующей инструкции `$ORIGIN`) или в своем собственном файле. Каждый сервер, даже кеширующий, обычно является главным для своего собственного обратного домена локального хоста.

Ниже приведено несколько строк из файла **named.conf**, описывающих конфигурацию локального хоста.

```
zone "localhost" { // зона прямого преобразования локального хоста
    type master;
    file "localhost";
    allow-update { none; };
};

zone "0.0.127.in-addr.arpa" {    // зона обратного преобразования
                                 //  локального хоста
    type master;
    file "127.0.0";
    allow-update { none; };
};
```

Соответствующий файл прямой зоны `localhost` содержит следующие строки.

```
$TTL 30d
; localhost.
@               IN   SOA   localhost. postmaster.localhost. (
                             2015050801     ;регистрационный номер
                             3600           ;тайм-аут обновления
                             1800           ;тайм-аут повторения
                             604800         ;срок действия
```

[17] На самом деле к локальному хосту относится весь класс сетей A 127/8, но большинство использует просто адрес 127.0.0.1.

```
                        3600 )          ;минимум
                NS    localhost.
                A     127.0.0.1
```

Обратный файл **127.0.0** имеет следующий вид.

```
$TTL 30d
; 0.0.127.in-addr.arpa
@               IN    SOA   localhost. postmaster.localhost. (
                            2015050801          ;регистрационный номер
                            3600                ;тайм-аут обновления
                            1800                ;тайм-аут повторения
                            604800              ;срок действия
                            3600 )              ;минимальный тайм-аут
                NS    localhost.
1               PTR   localhost.
```

Отображение для адреса локального хоста (127.0.0.1) никогда не изменяется, поэтому тайм-ауты могут быть большими. Обратите внимание на регистрационный номер, который кодирует дату; данный файл последний раз изменялся в 2015 году. Кроме того, обратите внимание на то, что для домена локального хоста указан только главный сервер имен. Символ @ здесь означает "0.0.127.in-addr.arpa.".

Небольшая компания, предоставляющая консалтинговые услуги в области безопасности

Наш следующий пример посвящен небольшой компании, специализирующейся на предоставлении консультаций по компьютерной безопасности. На ее сервере Red Hat Enterprise Linux установлен пакет BIND 9 и применяется расщепленная конфигурация DNS, в которой внутренние и внешние пользователи получают разную информацию. В компании используются адреса из частного диапазона. Запросы, касающиеся этих адресов, не должны выходить в Интернет и засорять глобальную систему доменных имен. Ниже приведен соответствующий файл **named.conf**, слегка переформатированный и снабженный комментариями.

```
options {
     directory "/var/domain";
     version "root@atrust.com";
     allow-transfer {82.165.230.84; 71.33.249.193; 127.0.0.1;};
     listen-on { 192.168.2.10; 192.168.2.1; 127.0.0.1; 192.168.2.12;}
};

include "atrust.key";                // Файл с режимом доступа 600

controls {
    inet 127.0.0.1 allow { 127.0.0.1; } keys { atkey; };
};

view "internal" {                    // Внутреннее представление

    match-clients { 192.168.1.0/24; 192.168.2.0/24; };
    recursion yes;

    include "infrustructure.zones"; // Корневые подсказки, прямое и обратное
                                    // преобразование адреса локального хоста
```

```
    zone "atrust.com" {              // Внутренняя зона
                                     // прямого преобразования
                                     // хоста localhost
       type master;
       file internal/atrust.com";
    };

    zone "1.168.192.in-addr.arpa" { // Внутренняя зона
                                     // обратного преобразования
       type master;
       file "internal/192.168.1.rev";
       allow-update { none; };
       };

   ... // Тут пропущено много других зон

include "internal/trademark.zones"l  // atrust.net, atrust.org и другие

                                     // подчиненные серверы
};                                   // конец внутреннего представления

view "world" {                       // Внешнее представление

    match-clients { any; };
    recursion no;

    zone "atrust.com" {              // Внешняя зона прямого преобразования
        type master;
        file "world/atrust.com";
        a;;ow-update { none; };
    };

    zone "189.172.63.in-addr.arpa" {  // Внешняя зона
                                      // обратного преобразования
        type master;
        file "world/63.173.189.rev";
        allow-update { none; };
    };
    include "world/trademark.zones";  // atrust.net, atrust.org и другие
                                      // подчиненные серверы

    zone "admin.com" {                // Главные зоны только
                                      // во внешнем представлении
       type master;
       file "worls/admin.com";
       allow-update [ none; ] ;

   };
   ...// Тут пропущено много главных и подчиненных зон
};// конец внешнего представления
```

Файл **atrust.key** содержит определение ключа "atkey".

```
key "atkey" {
    algorithm hmac-md5;
    secret "общий секретный ключ";
};
```

Файл **trademark.zones** содержит варианты имени atrust.com как в разных доменах верхнего уровня (net, org, us, info и т.д.), так и с разным написанием (appliedtrust.com и т.д.), а файл **infrastructure.zones** содержит корневые подсказки и файлы локального хоста.

Файлы зон определяют два разных представления (внутреннее и внешнее) и тип сервера (главный или подчиненный). Это отображается в соглашении об именах файлов зон. Во внутреннем представлении данный сервер является рекурсивным. Это представление содержит все локальные хосты, многие из которых используют частные адреса. Во внешнем представлении этот сервер не является рекурсивным. Данное представление содержит только выбранные хосты домена atrust.com и внешние зоны, для которых они обеспечивают DNS-услуги главных или подчиненных серверов.

Ниже приведены фрагменты файлов **internal/atrust.com** и **world/atrust.com**. Сначала рассмотрим файл **internal**.

```
; файл atrust.com - внутренний файл

$TTL 86400
$ORIGIN atrust.com.
@     3600    SOA  ns1.atrust.com. trent.atrust.com. (
                       2001110500 10800 1200 3600000 3600 )
      3600    NS   NS1.atrust.com.
      3600    NS   NS2.artust.com.
      3600    MX   10 mailserver.atrust.com.
      3600    A    66.77.22.161

ns1           A    192.168.2.1
ns2           A    66.77.122.161
www           A    66.77.122.161
mailserver    A    192.168.2.11
exchange      A    192.168.1.100
secure        A    66.77.122.161
...
```

Здесь использованы частные адреса RFC1918. Кроме того, обратите внимание на то, что вместо записей CNAME для определения имен локального хоста эта организация использует несколько записей A.[18] Эта схема работает быстрее, поскольку она не учитывает результат проверки записи CNAME в дополнительном запросе. Этот подход неплох, но каждому IP-адресу должна соответствовать только одна запись PTR в зоне обратного преобразования.

Рассмотрим текст внешнего представления той же зоны atrust.com из файла **world/atrust.com**.

```
; файл atrust.com - внешний файл

$TTL 57600
$ORIGIN atrust.com.
@                   SOA  ns1.atrust.com. trent.atrust.com. (
                             2010030400 10800 1200 3600000 3600 )
                    NS   NS1.atrust.com.
                    NS   NS2.atrusr.com.
```

[18]Одно время распознавание записей было потенциально быстрее, чем распознавание CNAME, потому что они освобождали клиентов от необходимости выполнять второй DNS-запрос для получения адреса цели CNAME. В наши дни DNS-серверы стали изощреннее и автоматически включают запись A для цели в исходном ответе на запрос (если они ее знают).

```
                    MX    10 mailserver.atrust.com.
                    A     66.77.122.161
ns1.atrust.com.     A     206.168.198.209
ns2.atrust.com.     A     66.77.122.161
www                 A     69.77.122.161
mailserver          A     206.168.198.209
secure              A     66.77.122.161

; обратные преобразования
exterior1           A      206.168.198.209
209.198.168.206     PTR    exterior1.atrust.com.
exterior2           A      206.168.198.213
213.198.168.206     PTR    exterior2.atrust.com.
```

Как и во внутреннем преобразовании, имена реализованы с помощью записей A. Очень немногие хосты на самом деле видны во внешнем представлении, хотя из данных фрагментов это не следует. Обратите внимание на то, что компьютеры, которые видны в обоих представлениях (например, ns1.atrust.com), во внутреннем представлении имеют частные адреса RFC1918, а во внешнем — реальные IP-адреса.

В этих файлах зоны параметр TTL по умолчанию установлен равным 16 часам (5700 секунд), а для внутренних зон — одному дню (86400 секунд).

16.9. Обновление файла зоны

Когда в домен вносится изменение (например, добавляется или удаляется компьютер), следует обновить файлы данных на главном сервере. Кроме того, необходимо увеличить порядковый номер в записи SOA для этой зоны. В заключение вы должны заставить программное обеспечение принять изменения.

Последний этап зависит от программного обеспечения. Если вы используете систему BIND, выполните команду **rndc reload**, чтобы демон **named** принял изменения. Вы можете также остановить работу демона **named** и запустить его снова, но если ваш сервер является одновременно авторитетным по отношению к вашей зоне и рекурсивным по отношению к вашим пользователям, то эта операция аннулирует кешированные данные, полученные от других доменов.

Обновленные данные немедленно передаются подчиненным серверам главных серверов BIND, поскольку параметр notify включен по умолчанию. Если же по какой-то причине он отключен, подчиненным серверам придется дождаться, пока истечет период обновления, установленный в записи SOA зоны (обычно один час).

Если параметр notify отключен, можно выполнить на каждом подчиненном сервере BIND команду **rndc reload**, которая заставит демон связаться с главным сервером, убедиться в том, что база данных зоны изменена, и запросить ее пересылку.

При изменении имени или IP-адреса компьютера не забывайте модифицировать зоны как прямого, так и обратного преобразований. Игнорирование последних может привести к появлению скрытых ошибок: часть команд будет работать, а часть — нет.

Если изменить файлы данных, но не обновить порядковый номер в записи SOA, то изменения вступят в силу только на главном сервере (после перезагрузки), но не на подчиненных.

Не следует редактировать файлы данных, относящиеся к подчиненным серверам. Эти файлы ведет сам демон **named**, и системные администраторы не должны вмешиваться в его работу. Лучше просматривать файлы данных сервера BIND, не делая в них никаких изменений.

Передача зоны

Серверы DNS синхронизируются посредством механизма передачи зоны. Передача зоны может включать в себя всю зону (такая передача называется AXFR) или только последние изменения (такая передача именуется IXFR). По умолчанию передача зоны осуществляется по протоколу TCP через порт 53. Сервер BIND регистрирует соответствующую информацию в системном журнале с пометкой "xfer-in" или "xfer-out".

В процессе передачи зоны подчиненный сервер запрашивает информацию от главного сервера и создает резервную копию данных о зоне на диске. Если данные на главном сервере не изменились, что определяется по порядковым номерам (не по реальным данным), то обновления не выполняются и резервные файлы просто фиксируются. (Иначе говоря, их время изменения устанавливается равным текущему времени.)

И отправляющий, и получающий серверы способны отвечать на запросы при передаче зоны. Подчиненный сервер начинает использовать новые данные только после завершения передачи.

Если зона очень большая (например, com) или обновляется динамически (о чем пойдет речь в следующем подразделе), то объем изменений обычно мал в сравнении с размером зоны. При передаче IXFR пересылаются только изменения (когда размер изменений начинает превышать размер зоны, включается режим обычной передачи AXFR). Механизм IXFR напоминает программу **patch**: старая база данных сравнивается и синхронизируется с новой.

В системе BIND передача IXFR задается по умолчанию, а демон **named**, предназначенный для ведения журнала транзакций, — именем *имя_зоны*.**jnl**. В инструкции server любого сервера, которому нужны такие передачи, можно задать параметры provide-ixfr и request-ixfr. Параметр provide-ixfr включает и отключает службу IXFR для зон, по отношению к которым сервер является главным. Параметр request-ixfr включает и отключает службу IXFR для зон, по отношению к которым сервер является подчиненным.

```
provide-ixfr yes ;          # В инструкции сервера BIND
request-ixfr yes ;          # В инструкции сервера BIND
```

Механизм IXFR может работать и с зонами, редактируемыми вручную. Для того чтобы включить этот режим, следует включить параметр ixfr-from-difference. Механизм IXFR требует, чтобы файлы зон были упорядочены в каноническом порядке. Если запрос IXFR передается серверу, который его не поддерживает, автоматически включается режим обычной передачи зоны AXFR.

Динамические обновления в системе BIND

Система DNS основана на предположении, что соответствие между именами и адресами относительно стабильно и меняется нечасто. Но это правило постоянно нарушается, если в организации используется протокол DHCP и при подключении к сети компьютеру динамически назначается IP-адрес. Существует два классических решения этой проблемы: добавить обобщенные записи в базу данных DNS или непрерывно редактировать DNS-файлы. В большинстве случаев ни одно из решений не является удовлетворительным.

🕮 Подробнее о протоколе DCHP см. в разделе 13.7.

Первое решение должно быть знакомо каждому, кто имеет коммутируемый выход в Интернет. Конфигурация DNS в этом случае выглядит примерно следующим образом.

```
dhcp-host1.domain.    IN  A   192.168.0.1
dhcp-host2.domain.    IN  A   192.168.0.2
...
```

Это простое решение, но оно означает, что указанные имена постоянно связаны с IP-адресами и, следовательно, компьютер, получающий новый адрес, меняет имя. В такой среде трудно соблюдать требования безопасности и вести журнальную регистрацию.

Механизм динамических обновлений, появившийся в последних версиях BIND, предлагает альтернативное решение. Он позволяет демону DHCP уведомлять сервер BIND о сделанных адресных назначениях, обновляя, таким образом, содержимое базы данных DNS "на лету". Динамические изменения могут добавлять, удалять и модифицировать запросы о ресурсах. Если динамические изменения дозволены, демон **named** ведет журнал динамических изменений (*имя_зоны*.**jnl**), который может оказаться полезным при сбое сервера. Демон **named** восстанавливает состояние зоны, сохраненное в памяти, считывая исходные файлы зон и воспроизводя изменения на основе журнала.

После того как зона была динамически обновлена, ее уже нельзя редактировать вручную. Необходимо сначала остановить сервер BIND с помощью команд **rndc freeze** *зона* или **rndc freeze** *зона класс представление*. Эти команды синхронизируют журнальный файл с файлом главной зоны на диске, а затем удаляют журнал. После этого можно отредактировать файл зоны вручную. К сожалению, исходное форматирование файла пропадет (он будет иметь вид файла, который ведется демоном **named** на подчиненных серверах).

При "заморозке" зоны попытки динамического обновления отменяются. Для того чтобы загрузить файл зоны с диска и все-таки выполнить динамическое обновление, следует использовать команду **rndc thaw** с теми же аргументами, которые были указаны при "заморозке" зоны.

Утилита **nsupdate**, входящая в дистрибутив BIND 9, позволяет осуществлять динамические обновления из командной строки. Утилита работает в пакетном режиме, принимая команды, вводимые с клавиатуры, или читая их из файла. Пустая строка или команда **send** являются признаком завершения обновления и пересылают изменения на сервер. Две пустые строки означают конец входного потока. В командном языке предусмотрена примитивная инструкция `if`, позволяющая формулировать условия вида "если имя неизвестно в DNS, добавить его". Искомое имя (или набор записей о ресурсах) может существовать либо отсутствовать. В качестве предиката утилиты **nsupdate** можно потребовать, чтобы имя или запись о ресурсе существовали или не существовали.

Ниже показан простейший сценарий **nsupdate**, который заносит в базу данных информацию о новом хосте, а также псевдоним существующего хоста при условии, что этот псевдоним нигде не используется. Угловые скобки создаются программой **nsupdate** и не являются частью командного сценария.

```
$ nsupdate
> update add newhost.cs.colorado.edu 86400 A 128.138.243.16
>
> prereq nxdomain gypsy.cs.colorado.edu
> update add gypsy.cs.colorado.edu CNAME evi-laptop.cs.colorado.edu
```

Динамические обновления — очень опасная возможность. Они потенциально способны предоставить право неконтролируемой записи важных системных данных. Не пытайтесь контролировать доступ на основании IP-адресов: их легко подделать. Лучше воспользоваться системой аутентификации TSIG с общим секретным ключом. Например, в системе BIND 9 поддерживается команда

```
$ nsupdate -k каталог_ключа:файл_ключа
```

или

```
% nsupdate -k каталог_ключа:секретный_ключ
```

Поскольку пароль задается в командной строке после ключа **-y**, его может увидеть тот, кто запустит команду **w** или **ps** в правильный момент. По этой причине более предпочтительной является форма **-k**. Подробнее технология TSIG описана в разделе 16.10.

Динамические обновления зоны разрешаются в файле **named.conf** посредством параметра `allow-update` или `update-policy`. Параметр `allow-update` предоставляет право обновления любых записей клиентам, чьи IP-адреса и ключи шифрования указаны в списке. Параметр `update-policy` появился в BIND 9 и позволяет точнее управлять обновлениями на основании имен хостов и типов записей. Он требует использовать механизмы аутентификации. Оба параметра являются зонными.

С помощью параметра `update-policy` можно разрешить клиентам обновлять свои записи `A` и `PTR`, но не `SOA`, `NS` или `KEY`. Можно также позволить хосту обновлять только свои записи. Имена допускается задавать явно, в виде поддоменов, с метасимволами или с помощью ключевого слова `self`, которое задает правила доступа компьютера к собственным записям. Записи о ресурсах идентифицируются по классу или типу.

Синтаксис параметра `update-policy` выглядит следующим образом.

```
(grant|deny) сущность имя_типа имя [типы] ;
```

Сущность — это имя криптографического ключа, необходимого для авторизации обновлений. *Имя_типа* имеет четыре значения: `name`, `subdomain`, `wildcard` или `self`. Опция `self` является особенно полезной, потому что позволяет хосту обновлять только свои записи. По возможности, следует использовать именно ее.

Имя — это обновляемая зона, а *типы* — типы записей о ресурсах, которые можно обновить. Если типы не указаны, то могут обновляться записи всех типов, кроме `SOA`, `NS`, `RRSIG` и `NSEC` или `NSEC3`. Рассмотрим пример.

```
update-policy { grant dhcp-key subdomain dhcp.cs.colorado.edu A } ;
```

В такой конфигурации любому, у кого есть ключ `dhcp-key`, разрешается обновлять адресные записи в поддомене `dhcp.cs.colorado.edu`. Эта директива должна появиться в файле **named.conf** в инструкции `zone` домена `dhcp.cs.colorado.edu`. Потребуется также инструкция `key`, содержащая определение ключа `dhcp-key`.

Приведенный ниже фрагмент файла **named.conf** факультета компьютерных наук университета Колорадо (Computer Science Department at the University of Colorado) использует инструкцию `update-policy` для того, чтобы позволить студентам, находящимся в классе системного администратора, обновлять свои поддомены, не касаясь при этом остального окружения системы DNS.

```
// saclass.net
zone "saclass.net" {
      type master;
      file "saclass/saclass.net";
      update-policy {
          grant feanor_mroe. subdomain saclass.net.;
          grant mojo_mroe. subdomain saclass.net.;
          grant dawdle_mroe. subdomain saclass.net.;
          grant pirate_mroe. subdomain saclass.net.;
          ...
      };
...
```

16.10. Вопросы безопасности DNS

DNS появилась как совершенно открытая система, но в процессе развития она становилась все более защищенной, приобретая необходимые средства защиты. По умолчанию каждый, у кого есть доступ в Интернет, может исследовать чужой домен отдельными запросами с помощью таких команд, как **dig**, **host**, **nslookup** или **drill**. В некоторых случаях можно получить образ всей базы данных DNS.

Для устранения этих недостатков в последние версии пакета BIND были введены различные средства управления доступом, основанные на проверке адресов или криптографической аутентификации. В табл. 16.5 перечислены элементы подсистемы безопасности, которые настраиваются в файлах **named.conf**.

Таблица 16.5. Средства защиты сервера BIND

Параметр	Инструкции	Что определяет
acl	Разные	Списки управления доступом
allow-query	options, zone	Кто может посылать запросы зоне или серверу
allow-recursion	options	Кто может посылать рекурсивные запросы
allow-transfer	options, zone	Кто может запрашивать передачи зон
allow-update	zone	Кто может выполнять динамические обновления
blackhole	options	Какие серверы нужно полностью игнорировать
bogus	server	Какие серверы никогда нельзя опрашивать
update-policy	zone	Какие обновления разрешены

Еще раз о списках управления доступом на сервере BIND

Список управления доступом — это именованный список соответствия адресов, который может служить аргументом различных директив, в частности `allow-query`, `allow-transfer` и `blackhole`. Синтаксис списков был рассмотрен ранее. Кроме того, списки управления доступом могут способствовать укреплению безопасности DNS-серверов.

В каждой организации должен существовать хотя бы один список для недоступных адресов и один — для локальных. Рассмотрим пример.

```
acl bogusnets {         // список недоступных и фиктивных сетей
    0.0.0.0/8;          // адреса по умолчанию
    1.0.0.0/8;          // зарезервированные адреса
    2.0.0.0/8;          // зарезервированные адреса
    169.254.0.0/16;     // канально-локальные делегируемые адреса
    192.0.2.0/24;       // тестовые адреса, наподобие example.com
    224.0.0.0/3;        // пространство групповых адресов
    10.0.0.0/8;         // частное адресное пространство (RFC1918)[19]
    172.16.0.0/8;       // частное адресное пространство (RFC1918)
    192.168.0.0/16;     // частное адресное пространство (RFC1918)
};

acl cunets {            // список сетей университета штата Колорадо
   128.138.0.0/16;      // основная кампусная сеть
```

[19]Не делайте частные адреса недоступными, если они используются для конфигурирования внутренних DNS-серверов!

```
    198.11.16.0/24;
    204.228.69.0/24;
};
```

Далее нужно в глобальной секции `options` конфигурационного файла разместить следующие директивы.

```
allow-recursion { cunets; };
blackhole { bogusnets; };
```

Желательно также ограничить передачи зон только легитимными подчиненными серверами. Это достигается с помощью следующих списков.

```
acl ourslaves {
    128.138.242.1;      // сервер anchor
    ...
};

acl measurements {
    198.32.4.0/24;      // проект Билла Маннинга, адрес v4
    2001:478:6:::/48;   // проект Билла Маннинга, адрес v6
};
```

Собственно ограничение реализуется такой строкой.

```
allow-transfer { ourslaves; measurements; };
```

Передачи разрешены только нашим подчиненным серверам, а также компьютерам глобального исследовательского проекта, который посвящен определению размеров сети Интернет и процента неправильно сконфигурированных серверов. Подобное ограничение лишает остальных пользователей возможности получать дамп всей базы данных с помощью команды **dig** (см. раздел 16.4).

Разумеется, нужно по-прежнему защищать сеть на более низком уровне с помощью списков управления доступом на маршрутизаторе и стандартных средств защиты на каждом хосте. Если такой возможности нет, ограничьте трафик DNS-пакетов шлюзовым компьютером, который находится под постоянным административным контролем.

Открытые распознаватели

Открытый распознаватель — это рекурсивный кеширующий сервер имен, получающий запросы от любого пользователя Интернета и отвечающий на них. Открытые распознаватели опасны. Внешние пользователи могут потреблять ваши ресурсы без вашего разрешения или ведома, и, если они делают это со злым умыслом, кеш вашего распознавателя может быть "заражен".

Что еще хуже, открытые распознаватели иногда используются злоумышленниками для усиления распределенной атаки на основе отказа в обслуживании. Организаторы атаки посылают запрос на ваш распознаватель, указывая фальшивый адрес отправителя, который является объектом атаки. Ваш распознаватель послушно отвечает на эти запросы и посылает огромный пакет жертве. Жертва не посылала запросы, но она обязана выполнить маршрутизацию и обработать сетевой трафик. Умножьте объем пакета на количество распознавателей, и вы осознаете реальную опасность, грозящую жертве.

Статистика свидетельствует о том, что от 70 до 75% кеширующих серверов имен в настоящее время являются открытыми распознавателями! Сайт `dns.measurement-factory.com/tools` может помочь вам проверить вашу организацию. Зайдите на него, щелкните на ссылке Open resolver test и введите IP-адрес вашего сервера имен. Вы мо-

жете также проверить все серверы имен вашей сети или все серверы вашей организации, используя идентификаторы `whois`.

Для того чтобы ваши кеширующие серверы имен отвечали на запросы только локальных пользователей, используйте список управления доступом в файлах **named.conf**.

Работа в виртуальном окружении chroot

Если хакеры взломают ваш сервер, то они смогут получить доступ к системе под видом законного пользователя. Для того чтобы уменьшить опасность, возникающую в этой ситуации, вы можете запустить сервер в виртуальном окружении **chroot** под видом непривилегированного пользователя либо сделать и то, и другое одновременно.

Для демона **named** флаг команды **-t** задает каталог виртуального окружения **chroot**, а флаг **-u** указывает идентификатор пользователя, под которым работает демон **named**. Рассмотрим пример.

```
$ sudo named -u 53
```

Эта команда запускает демон **named** как корневой процесс, но после того как демон **named** закончит рутинные операции в роли корневого процесса, он потеряет привилегии и запустится под идентификатором UID 53.

Многие организации не используют флаги **-u** и **-t**, но если объявлена тревога, они должны реагировать быстрее, чем хакеры.

Виртуальное окружение **chroot** не может быть пустым каталогом, поскольку оно должно содержать все файлы, необходимые серверу имен для нормальной работы, — **/dev/null**, **/dev/random**, файлы зон, файлы конфигурации, ключи, файлы системного журнала и доменного сокета UNIX, **/var** и др. Для того чтобы настроить их, требуется выполнить довольно большую работу. Вызов системы **chroot** осуществляется после загрузки библиотек, поэтому копировать общие библиотеки в виртуальное окружение не обязательно.

Безопасные межсерверные взаимодействия посредством технологий TSIG и TKEY

Пока спецификация DNSSEC (см. следующий подраздел) находилась на стадии принятия, группа IETF разработала более простой механизм, названный TSIG (RFC2845). Он позволял организовать безопасное взаимодействие серверов благодаря использованию сигнатур транзакций. Контроль доступа, основанный на таких сигнатурах, надежнее контроля на основе исходных IP-адресов. Технология TSIG позволяет защитить передачу зоны между главным и подчиненным серверами, а также защитить динамические изменения.

В технологии TSIG применяется симметричная схема шифрования, т.е. ключ шифрования совпадает с ключом дешифрования. Такой ключ называется *общим секретным ключом* (shared secret). Спецификация TSIG допускает использование нескольких методов шифрования. В пакете BIND реализованы некоторые из таких методов. Для каждой пары серверов, между которыми организуется защищенный канал связи, должен создаваться свой ключ.

Спецификация TSIG гораздо менее затратная в вычислительном плане, чем шифрование с открытым ключом, но она подходит только для локальной сети, где число пар взаимодействующих серверов невелико. На глобальную сеть эта спецификация не распространяется.

Настройка технологии TSIG для сервера BIND

Утилита **dnssec-keygen**, являющаяся частью пакета BIND, генерирует ключ для пары серверов. Рассмотрим, например, следующую команду.

```
$ dnssec-keygen -a HMAC-SHA256 -b 128 -n HOST master-slave1
Kmaster-slave1+163+15496
```

Флаг **-b 128** означает, что утилита **dnssec-keygen** должна создать 128-разрядный ключ. В данном случае мы используем 128-разрядный ключ только лишь для того, чтобы он поместился на странице. В реальной жизни следует использовать более длинный ключ; максимально допустимая длина ключа равна 512.

Данная команда создает два файла:

```
Kmaster-slave1.+163+15496.private
Kmaster-slave1.+163+15496.key
```

Здесь **163** — код алгоритма HMAC-SHA256, а **15496** — случайное число, используемое в качестве идентификатора ключа на случай, если у одной пары серверов есть несколько ключей.[20] Оба файла содержат один и тот же ключ, но в разных форматах.

Файл **.private** выглядит примерно так.

```
Private-key-format: v1.3
Algorithm: 163 (HMAC_SHA256)
Key: owKt6ZWOlu0gaVFkwOqGxA==
Bits: AAA=
Created: 20160218012956
Publish: 20160218012956
Activate: 20160218012956
```

Содержимое файла **.key** будет таким.

```
master-slave1. IN KEY 512 3 163 owKt6ZWOlu0gaVFkwOqGxA==
```

Обратите внимание на то, что утилита **dnssec-keygen** добавила точки в конец имени каждого ключа в обоих именах файлов и внутри файла **.key**. Это объясняется тем, что когда утилита **dnssec-keygen** использует ключи DNSSEC, добавляемые в файлы зон, имена этих ключей должны быть полностью определенными именами доменов и, следовательно, должны заканчиваться точкой. Таким образом, нам нужны два инструмента: один для общих секретных ключей, а второй — для пар открытых ключей.

Вообще-то, файл **.key** на самом деле не нужен: он создается потому, что утилита **dnssec-keygen** имеет двойное назначение. Просто удалите его. Число 512 в записи KEY означает не длину ключа, а флаг, идентифицирующий запись как запись о главном ключе DNS.

После всех этих сложностей вы могли быть разочарованы тем, что сгенерированный ключ представляет собой просто длинное случайное число. Этот ключ можно было бы сгенерировать вручную, записав строку в кодировке ASCII, длина которой делится на четыре, и считать, что вы применили 64-разрядное кодирование, или использовать утилиту **mmencode** для того, чтобы зашифровать случайную строку. Способ, которым вы кодируете ключ, не имеет значения; он просто должен существовать на обеих машинах.

Скопируйте ключ из файла **.private** на оба сервера с помощью команды **scp** или просто скопируйте и вставьте его. *Не используйте* утилиты **telnet** и **ftp** для копирования ключа: даже внутренние сети могут быть небезопасными.

[20] Это число выглядит случайным, но на самом деле оно представляет собой хешированное значение ключа TSIG.

🕮 Команда **scp** является частью пакета SSH. Дополнительную информацию см. в разделе 27.7.

Ключ должен быть записан в файлах **named.conf** на обоих компьютерах. В связи с тем, что эти файлы общедоступны, а ключ — нет, поместите ключ в отдельный файл, который будет включаться в файл **named.conf**. Файл ключа должен иметь уровень доступа, равный 600, и принадлежать пользователю демона **named**.

Например, можно создать файл **master-slave1.tsig** и включить в него следующий фрагмент.

```
key master-slave1 {
    algorithm hmac-md5 ;
secret "сгенерированный_ключ" ;
};
```

В файл **named.conf** ближе к началу нужно добавить такую строку.

```
include "master-slave1.tsig";
```

На данном этапе лишь определен ключ. Для того чтобы его можно было использовать для подписи и проверки обновлений, нужно посредством инструкции `server` и директивы `keys` заставить каждый сервер идентифицировать другую сторону. Например, в инструкцию `zone` на главном сервере можно включить строку

```
allow-transfer { key master-slave1. ; };
```

а в файл **named.conf** подчиненного сервера — строку

```
server IP-адрес-главного_сервера { keys { master-slave1 ; } ; } ;
```

Имя ключа в нашем примере носит общий характер. Если вы используете ключи TSIG для многих зон, то, возможно, захотите включить в имя ключа имя зоны, чтобы все стало ясно.

Если вы впервые используете подписи транзакций, запускайте демон **named** на первом уровне отладки (режимы отладки будут описаны позднее), пока сообщения об ошибках не исчезнут.

При использовании ключей TSIG и подписей транзакций между главным и подчиненным серверами необходимо синхронизировать часы на обоих серверах по протоколу NTP. Если часы слишком сильно расходятся (более чем на пять минут), то верификация подписи не будет выполнена. Эту проблему иногда очень сложно распознать.

TSIG — это механизм сервера BIND, позволяющий двум хостам автоматически генерировать общий секретный ключ, не прибегая к телефонным звонкам или секретному копированию для распространения ключа. Он использует алгоритм обмена ключами Диффи–Хеллмана (Diffie-Hellman key exchange algorithm), в котором на каждой стороне генерируется случайное число, над ним выполняются определенные математические операции, а результат посылается другой стороне. Затем каждая сторона объединяет свое и полученное числа по определенному математическому правилу и получает один и тот же ключ. Злоумышленник может перехватить сообщение, но он не сможет выполнить над ним требуемые математические операции.[21]

Серверы компании Microsoft используют нестандартный вариант механизма TSIG, который называется GSS-TSIG. В нем для обмена ключами используется технология TKEY. Если вам нужно, чтобы сервер компании Microsoft взаимодействовал с сервером BIND, используйте опции `tkey-domain` и `tkey-gssapi-credential`.

[21] Математическую основу этого алгоритма образует задача дискретного логарифмирования, особенность которой состоит в том, что в модулярной арифметике возвести число в степень довольно просто, а вычислить логарифм по этой степени практически невозможно.

Другой механизм для подписи транзакций между серверами или службами динамического обновления и главным сервером называется SIG(0). Он использует криптографию, основанную на открытых ключах. Детали этой технологии описаны в документах RFC2535 и RFC2931.

Технология DNSSEC

DNSSEC — это набор расширений DNS, позволяющих аутентифицировать источник данных зоны и проверить их целостность, используя шифрование с открытым ключом. Таким образом, DNSSEC позволяет DNS-клиентам задавать вопросы вида "Действительно ли данные зоны поступили от владельца зоны?" и "Те ли это данные, которые послал владелец?"

Технология DNSSEC основана на цепочке доверия: корневые серверы предоставляют подтверждающую информацию для доменов верхнего уровня, те — для доменов второго уровня и т.д.

В системах шифрования с открытым ключом используются два ключа: один шифрует (подписывает) сообщение, а другой дешифрует (проверяет) его. Публикуемые данные подписываются секретным "личным" ключом. Любой может проверить правильность цифровой подписи с помощью соответствующего "открытого" ключа, который свободно распространяется. Если открытый ключ позволил правильно расшифровать файл зоны, значит, зона была зашифрована искомым личным ключом. Суть в том, чтобы убедиться в подлинности открытого ключа, используемого для проверки. Такие системы шифрования позволяют одной из сторон поставить свою "подпись" на открытом ключе, передаваемом другой стороне, гарантируя легитимность ключа. Отсюда термин *цепочка доверия*.

Данные, составляющие зону, слишком велики для шифрования с открытым ключом; процесс шифрования получится слишком медленным. Вместе с тем данные сами по себе не секретны, поэтому они просто пропускаются через функцию хеширования (к примеру, вычисляется контрольная сумма MD5), а результат подписывается (шифруется) секретным ключом зоны. Полученный зашифрованный хеш-код называется *цифровой подписью* или *сигнатурой* зоны. Цифровые подписи обычно добавляются к данным, подлинность которых они удостоверяют в виде записей RRSIG в подписанном файле зоны.

Для проверки сигнатуры ее нужно дешифровать открытым ключом подписавшей стороны, "прогнать" данные через тот же алгоритм хеширования и сравнить вычисленный хеш-код с расшифрованным. В случае совпадения подписавшее лицо считается аутентифицированным, а данные — целостными.

В технологии DNSSEC каждая зона имеет открытые и секретные ключи. Фактически она имеет два набора ключей: пара ключей для подписи зоны и пара ключей для подписи ключа. Секретным ключом подписи зоны подписывается каждый набор ресурсов (т.е. каждый набор записей одного типа, относящихся к одному хосту). Открытый ключ подписи зоны используется для проверки сигнатур и включается в базу данных зоны в виде записи `DNSKEY`.

Родительские зоны содержат записи DS, представляющие собой хешированный вариант записей `DNSKEY` для самоподписывающихся ключей подписи ключей (self-signed key-signing keys). Сервер имен проверяет аутентичность записи `DNSKEY` дочерней зоны, сравнивая ее с подписью родительской зоны. Для проверки аутентичности ключа родительской зоны сервер имен проверяет родительскую зону родительской зоны и так далее, пока не вернется в корневую зону. Открытый ключ корневой зоны должен быть открыто опубликован и включен в файлы "подсказок" корневой зоны.

В соответствии со спецификациями DNSSEC, если зона имеет несколько ключей, каждый из них должен применяться при проверке данных. Это требование объясняется

тем, что ключи могут быть изменены без прерывания работы службы DNS. Если рекурсивный сервер имен, использующий технологию DNSSEC, посылает запрос неподписанной зоне, то поступающий неподписанный ответ считается корректным. Проблема возникает лишь тогда, когда срок действия подписи истек или родительская и дочерняя зоны не согласовали текущий ключ DNSKEY дочерней зоны.

Правила протокола DNSSEC

Прежде чем приступать к развертыванию протокола DNSSEC, следует усвоить несколько правил и процедур или хотя бы подумать о них. Рассмотрим примеры.

- Ключи какого размера вы будете использовать? Более длинные ключи являются более надежными, но они увеличивают размеры пакетов.
- Как часто будете изменять ключи, если никаких нарушений правил безопасности не происходит?

Мы предлагаем завести специальный журнал, в который вы будете записывать данные о каждом сгенерированном ключе; об аппаратном и программном обеспечении, используемом для этого; о дескрипторе, присвоенном ключу; о версии программного генератора ключей; использованном алгоритме; длине ключа и сроке действия подписи. Если впоследствии криптографический алгоритм окажется скомпрометированным, вы сможете проверить свой журнал и выяснить, не подвергаетесь ли вы опасности.

Записи о ресурсах DNSSEC

Протокол DNSSEC использует шесть типов записей о ресурсах, кратко описанных в разделе 16.4, — DS, DLV, DNSKEY, RRSIG, NSEC и NSEC3. Сначала мы опишем их в целом, а затем очертим их использование в процессе подписания зоны. Каждая из этих записей создается инструментами протокола DNSSEC, а не вводится в файл зоны с помощью текстового редактора.

Запись DS (Designated Signer) возникает только в дочерней зоне и означает, что подзона является безопасной (подписанной). Она также идентифицирует ключ, используемый дочерней зоной для подписания своего собственного набора записей о ресурсах KEY. Запись DS содержит идентификатор ключа (пятизначное число), криптографический алгоритм, тип дайджеста (digest type) и дайджест записи об открытом ключе, разрешенном (или использованном) для подписи записи о ключе дочерней зоны. Соответствующий пример приведен ниже.[22]

```
example.com. IN  DS   682  5  1  12898DCF9F7AD20DBCE159E7...
```

Изменение существующих ключей в родительской и дочерней зонах является сложной проблемой и требует сотрудничества и взаимодействия между родительской и дочерней зонами. Для решения этой проблемы создаются записи DS, используются отдельные ключи для подписи ключей и зон, а также применяются пары многократных ключей.

Ключи, содержащиеся в записи о ресурсах DNSKEY, могут быть либо ключами для подписания ключа (key-signing key — KSK), либо ключами для подписания зоны (zone-signing key — ZSK). Для различия между ними используется новый флаг под названием SEP ("secure entry point" — "точка безопасного входа"). Пятнадцатый бит поля флагов устанавливается равным единице для ключа KSK и 0 — для ключа ZSK. Это соглашение делает поле флагов нечетным числом для ключа KSK и четным для ключей

[22] В этом разделе 64-разрядные хеши и ключи были усечены, чтобы сэкономить пространство и лучше проиллюстрировать структуру записей.

ZSK, если интерпретировать его как десятичное число. В настоящее время эти значения равны 257 и 256 соответственно.

Для обеспечения удобного перехода от одного ключа к другому можно генерировать многократные ключи. Дочерняя зона может изменить свои ключи подписания зоны, не сообщая об этом родительской зоне. Она должна согласовывать с родительской зоной только изменение ключа для подписания ключей. Когда ключ изменяется, и старый, и новый ключи на определенном отрезке времени считаются действующими. Как только срок действия кешированных значений в Интернете истечет, старый ключ будет считаться отмененным.

Запись RRSIG — это подпись набора записей о ресурсах (т.е. набора всех записей одного и того же типа и с одним и тем же именем внутри зоны). Записи RRSIG генерируются программным обеспечением, предназначенным для подписания зоны, и добавляются в подписанную версию файла зоны.

Запись RRSIG содержит много информации.

- Тип записей о ресурсах, входящих в подписываемый набор.
- Алгоритм, используемый для подписания и закодированный небольшим числом.
- Количество меток (фрагментов, разделенных точками) в поле имени.
- Значение TTL для подписываемого набора записей.
- Срок действия подписи (в формате *ггггммддччссс*).
- Время подписания набора записей (также в формате *ггггммддччссс*).
- Идентификатор ключа (пятизначный номер).
- Имя подписавшего (имя домена).
- Сама цифровая подпись (64-разрядная).

Рассмотрим пример.

```
RRSIG   NS 5 2 57600 20090919182841 (
          20090820182841 23301 example.com.
          pMKZ76waPVTbIguEQNUojNVlVewHau4p...==)
```

При подписании зоны также генерируются записи NSEC или NSEC3. В отличие от подписанных наборов записей, они сертифицируют интервалы *между* именами наборов записей и тем самым позволяют подписывать ответ типа "нет такого домена" или "нет такого набора записей о ресурсах". Например, сервер может ответить на запрос записей A с именем bork.atrust.com, послав запись NSEC, подтверждающую отсутствие любых записей A между именами bork.atrust.com и borrelia.atrust.com.

К сожалению, включение в записи NSEC конечных точек позволяет обойти зону и выяснить все ее действующие хосты. В версии NSEC3 этот недостаток был устранен путем включения в запись хешированных имен конечных точек, а не самих имен. Правда, это было сделано за счет более интенсивных вычислений: чем выше безопасность, тем ниже производительность. В настоящее время используются как записи NSEC, так и записи NSEC3, и вы должны будете сделать выбор между ними при генерировании своих ключей и подписании своих зон.

Если защита от блуждания по зоне не является критически важной для вашей организации, мы рекомендуем пока использовать запись NSEC.

Настройка протокола DNSSEC

Использование подписанных зон связано с двумя разными рабочими потоками: один из них создает ключи и подписывает зоны, а второй обслуживает содержимое этих подписанных зон. Эти функции не обязательно реализовывать на одном и том же компьютере.

На самом деле лучше изолировать закрытые ключи и процесс подписания зоны, сильно загружающий центральный процессор, на компьютере, который не доступен из Интернета. (Разумеется, компьютер, обрабатывающий данные, должен быть виден из Интернета.)

На первом этапе настройки протокола DNSSEC следует организовать файлы зон так, чтобы все файлы данных для зоны находились в одном каталоге. Это необходимо для правильной работы инструментов, управляющих зонами по протоколу DNSSEC.

Затем следует включить протокол DNSSEC на своих серверах с помощью опиций файла **named.conf**.

```
options {
     dnssec-enable yes;
}
```

для авторитетного сервера и

```
options {
     dnssec-enable yes;
     dnssec-validation yes;
}
```

для рекурсивных серверов. Опция `dnssec-enable` приказывает вашему авторитетному серверу включать подписи наборов записей DNSSEC в свои ответы на запросы, поступающие от серверов имен, работающих по протоколу DNSSEC. Опция `dnssec-validation` заставляет демон **named** проверять легитимность подписей, которые он получает в ответ от других серверов.

Генерирование пар ключей

Необходимо сгенерировать две пары ключей для каждой зоны, которую хотите подписать, — для подписания зоны (ZSK) и для подписания ключей (KSK). Каждая пара состоит из открытого и закрытого ключей. Закрытый ключ KSK подписывает ключ ZSK и создает безопасную точку входа для зоны. Закрытый ключ ZSK подписывает записи о ресурсах зоны. Открытые ключи затем публикуются, чтобы позволить другим сайтам проверить ваши подписи.

Команды

```
$ dnssec-keygen -a RSASHA256 -b 1024 -n ZONE example.com
Kexample.com.+008+29718
$ dnssec-keygen -a RSASHA256 -b 2048 -n ZONE -f KSK example.com
Kexample.com.+008+05005
```

генерируют для сайта `example.com` пару 1024-битовых ключей ZSK, использующую алгоритмы RSA и SHA-256, а также соответствующую пару 2048-битовых ключей KSK.[23] Серьезная проблема, связанная с ограничением на предельный размер пакета UDP, вынуждает отдавать предпочтение коротким ключам для подписания зоны и часто их менять. Для повышения уровня безопасности можно использовать более длинные ключи для подписания ключей.

Генерирование ключей занимает немного времени — всего несколько секунд. Как правило, ограничивающим фактором является не мощность центрального процессора, а энтропия, доступная для рандомизации. В системе Linux для повышения энтропии за счет дополнительных источников используется демон **haveged**, который повышает скорость генерирования ключей.

[23] Рекомендации разных организаций, касающиеся длин криптографических ключей, приведены на веб-сайте `keylength.com`.

Команда **dnssec-keygen** выводит в стандартный поток вывода базовое имя файла для сгенерированного ключа. В нашем примере **example.com** — это имя ключа, **008** — идентификатор набора алгоритмов RSA/SHA-256, а **29718** и **05005** — хешированные значения, которые называются идентификаторами ключей, слепками ключей или дескрипторами ключей. При каждом запуске команда **dnssec-keygen** создает два файла (**.key** и **.private**).

```
Kexample.com+008+29718.key      # Открытый ключ для подписи зоны
Kexample.com+008+29718.private  # Закрытый ключ для подписи зоны
```

Существует несколько алгоритмов шифрования, каждый из которых имеет свой диапазон длин ключей. Для того чтобы увидеть список доступных алгоритмов, можно выполнить команду **dnssec-keygen** без аргументов. Кроме того, сервер BIND может использовать ключи, сгенерированные другим программным обеспечением.

В зависимости от версии вашего программного обеспечения, имена некоторых доступных алгоритмов могут содержать имя NSEC3 в качестве префикса или окончания. Если вы хотите использовать записи NSEC3, а не NSEC для подписания отрицательных ответов, должны сгенерировать NSEC3-совместимые ключи одним из NSEC3-совместимых алгоритмов; подробности можно найти на справочной странице, посвященной команде **dnssec-keygen**.

Каждый файл **.key** содержит по одной записи о ресурсах DNSSEC для сайта example.com. Например, ниже приведен открытый ключ для подписания зоны, усеченный по ширине страницы. Его можно назвать ключом ZSK, потому что поле файлов равно 256, а не 257, как для ключей KSK.

```
example.com. IN DNSKEY 256 3 8 AwEAAcyLrgENt8OJ4PIQiv2ZhWwSviA...
```

Эти открытые ключи должны быть вставлены в файлы зон с помощью директивы $INCLUDE или другим способом либо в конец, либо сразу после записи SOA. Для того чтобы скопировать ключи в файл зоны, можно добавить их с помощью команды **cat**[24] и вставить с помощью текстового редактора.

В идеале закрытая часть любой пары ключей должна храниться в компьютере, отключенном от сети, или хотя бы в компьютере, недоступном из Интернета. Для динамически обновляемых зон это требование практически невыполнимо, а для ключей, предназначенных для подписания зон, еще и непрактично. Тем не менее для ключей, предназначенных для подписания других ключей, данное требование абсолютно разумно, поскольку эти ключи имеют долгий срок действия. Подумайте о скрытом главном сервере, недоступном извне для ключей ZSK. Распечатайте закрытый ключ KSK или запишите его на флешку, а затем спрячьте в сейфе, пока он не потребуется в следующий раз.

"Заперев" новые закрытые ключи, целесообразно записать информацию о новых ключах в системный файл ключей. При этом не обязательно записывать туда сами ключи. Достаточно записать их идентификаторы, алгоритмы, дату, назначение и т.д.

По умолчанию срок действия подписи ограничен одним месяцем для записей RRSIG (ZSK-подписи наборов записей о ресурсах) и тремя месяцами для записей DNSSIG (KSK-подписи ключей ZSK). В настоящее время рекомендуется использовать 1024-битовые ключи ZSK от трех месяцев до одного года, а 1280-битовые ключи KSK — от одного до двух лет.[25] Поскольку рекомендуемые сроки действия ключей дольше, чем сроки, установленные для них по умолчанию, эти параметры необходимо указывать явно при подписании или периодическом переподписании зон, даже если сами ключи не изменились.

[24] Используйте команду наподобие cat Kexmaple.com+*.key >> zonefile. Операция >> означает добавление ключа в конец файла zonefile, а не его замену, как операция >. (Не перепутайте их!)

[25] Рекомендации разных организаций, касающиеся длин криптографических ключей, приведены на веб-сайте keylength.com.

Подписание зоны

Итак, теперь, когда у вас есть ключи, вы можете подписывать зоны с помощью команды **dnssec-signzone** , добавляющей записи RRSIG и NSEC или NSEC3 в каждый набор записей о ресурсах. Эти команды считывают оригинальный файл зоны и создают отдельную подписанную копию с именем *файл_зоны*.**signed**.

Синтаксис команды **dnssec-signzone** для сервера BIND имеет следующий вид.

```
dnssec-signzone [-o зона][-N increment][-k KSK-файл]
      файл_зоны [ZSK-файл]
```

где параметр *зона* по умолчанию равен *файл_зоны*, а ключевые файлы по умолчанию задаются командой **dnssec-keygen**, как описано выше.

Если имена ваших файлов зоны совпадают с именами зон, а имена ключевых файлов не изменялись, то команда сокращается до следующего варианта:

```
dnssec-signzone [-N increment] файл_зоны
```

Флаг `-N increment` автоматически увеличивает порядковый номер в записи SOA, так что забыть это сделать будет невозможно. Кроме того, можно указать значение **unixtime**, чтобы установить порядковый номер равным текущему времени UNIX (количеству секунд, прошедших с 1 января 1970 года), или значение **keep**, чтобы предотвратить изменение оригинального порядкового номера командой **dnssec-signzone**. Порядковый номер увеличивается в файле подписанной зоны, но не в оригинальном файле зоны.

Рассмотрим пример, в котором используются сгенерированные ранее ключи.

```
$ sudo dnssec-signzone -o example.com -N increment
   -k Kexample.com+008+05005 example.com Kexample.com+008+29718
```

Подписанный файл упорядочивается в алфавитном порядке и содержит записи DNSKEY, добавленные вручную, а также записи RRSIG и NSEC, сгенерированные в ходе подписания. Порядковый номер зоны увеличен.

Если вы сгенерировали свои ключи с помощью алгоритма, совместимого с NSEC3, то должны подписать зону так, как показано выше, но указав флаг **-3** *salt*. Другие полезные опции команды **dnssec-signzone** приведены в табл.16.6.

Таблица 16.6. Флаги команды dnssec-signzone

Флаг	Описание
`-g`	Генерирование записи (записей) DS, которая должна быть включена в родительскую зону
`-s` *начальный_момент*	Установка момента времени, с которого подписи считаются действительными
`-e` *конечный_момент*	Установка момента времени, после которого подписи считаются недействительными
`-t`	Вывод статистических показателей

Данные о сроках действия подписей можно выразить в виде абсолютного времени в формате ггггммддччммсс или относительного времени, считая от текущего момента, в формате +*N*, где *N* — количество секунд. По умолчанию период действия подписи изменяется от одного часа в прошлое до 30 дней в будущее. Рассмотрим пример, в котором мы указываем, что подписи должны быть действительными до окончания календарного 2017 года.

```
$ sudo dnssec-signzone -N increment -e 20171231235959 example.com
```

Размеры файлов подписанных зон от четырех до десяти раз больше, чем размеры файлов исходных зон, и все попытки навести логический порядок напрасны. Строка наподобие

```
mail-relay          A     63.173.189.2
```

превращается в несколько следующих строк.

```
mail-relay.example.com.    57600    A    63.173.189.2
   57600     RRSIG    A 8 3 57600 20090722234636 (
                        20150622234636 23301 example.com.
                        Y7s9jDWYuuXvozeU7zGRdFCl+rzU8cLiwoev
                        0I2TGfL1bhsRgJfkpEYFVRUB7kKVRNguEYwk
                        d2RSkDJ9QzRQ+w== )
   3600      NSEC     mail-relay2.example.com. A RRSIG NSEC
   3600      RRSIG    NSEC 8 3 3600 20090722234636 (
                        20150622234636 23301 example.com.
                        42QrXP8vpoChsGPseProBMZ7twf7eS5WK+4O
                        WNsN84hF0notymRxZRIZypqWzLIPBZAUJ77R
                        HP0hLfBDoqmZYw== )
```

С практической точки зрения файл подписанной зоны больше нельзя назвать понятным для человека и его невозможно отредактировать вручную из-за записей `RRSIG` и `NSEC` или `NSEC3`. Этот файл не содержит ни одного фрагмента, который пользователь мог бы изменить вручную!

За исключением записей DNSSEC, каждый набор записей о ресурсах (ресурсных записей с одинаковым типом и именем) получает одну подпись от ключа ZSK. Записи о ресурсах `DNSSEC` подписываются как ключом ZSK, так и ключом KSK, поэтому они содержат две записи `RRSIG`. Тем не менее 64-разрядное представление подписи заканчивается несколькими знаками равенства, потому что длина записи должны быть кратной четырем.

Как только ваша зона подписана, остается лишь указать сервер имен в подписанных версиях файлов зоны. Если вы используете сервер BIND, поищите инструкцию `zone`, соответствующую каждой зоне в файле **`named.conf`**, и измените параметр `file` с `example.com` на `example.com.signed`.

В заключение перезапустите демон сервера имен, заставив его прочитать снова свой файл конфигурации. Для сервера BIND следует выполнить команды **`sudo rndc reconfig`** и **`sudo rndc flush`**.

Теперь мы обслуживаем подписанную зону DNSSEC! Для того чтобы внести изменения, вы можете отредактировать либо исходный неподписанный файл зоны, либо подписанный файл зоны, а затем переподписать зону. Редактирование подписанной зоны иногда оказывается чрезвычайно сложной задачей, но это проще, чем повторное подписание всей зоны. Удалите записи `RRSIG`, соответствующие всем изменяемым записям. Для того чтобы избежать путаницы между версиями, вероятно, следует внести идентичные изменения в неподписанную зону.

При передаче подписанной зоны в качестве аргумента команде **`dnssec-signzone`** все неподписанные записи подписываются, а подписи всех записей, срок действия которых подходит к концу, обновляются. Выражение "срок действия подходит к концу" означает, что прошло три четверти периода действия подписи. Повторное подписание, как правило, приводит к изменениям, поэтому следует изменить порядковый номер зоны вручную или автоматически с помощью сервера BIND, используя паратмер **`-N increment`** в командной строке **`dnssec-signzone`**.

Это все, что касается локальной части конфигурации протокола DNSSEC. Осталось только решить трудную проблему: соединить наш безопасный "DNS-островок" с другими надежными и подписанными частями иерархии DNS. Мы должны либо передать наши записи DS в подписанную родительскую зону, либо использовать динамическую проверку доменов. Решение этих задач описывается в следующем разделе.

Цепочка доверия в протоколе DNSSEC

Продолжим наш пример, связанный с настройкой протокола DNSSEC. Сайт `example.com` теперь подписан, и его серверы имен поддерживают протокол DNSSEC. Это значит, что при отправке запросов они используют протокол EDNS0, расширенный протокол DNS, а в DNS-заголовке пакета устанавливают флаг, включающий протокол DNSSEC. Отвечая на запросы, которые приходят с таким установленным битом в заголовке, они включают в свой ответ подписанные данные.

Клиент, получающий подписанные запросы, может оценить корректность ответа, проверив его подпись с помощью соответствующего открытого ключа. Однако он получает этот ключ опять же из записи зоны `DNSKEY`, что, если вдуматься, довольно подозрительно. Что может помешать постороннему лицу предоставить ложные записи и ложный открытый ключ, чтобы пройти проверку?

Каноническое решение заключается в том, чтобы передать вашей родительской зоне запись `DS`, которую она должна включить в свой файл зоны. Запись `DS`, приходящая из родительской зоны, сертифицируется родительским закрытым ключом. Если клиент доверяет своей родительской зоне, он должен верить в то, что запись `DS` родительской зоны правильно отражает открытый ключ вашей зоны.

В свою очередь, родительская зона сертифицируется своей родительской зоной и так вплоть до корня.

Смена ключей DNSSEC

Для протокола DNSSEC смена ключей всегда была трудной задачей. Его исходные спецификации были, по существу, изменены для того, чтобы решить проблемы, связанные с обеспечением взаимодействия между родительской и дочерней зонами для создания, изменения или удаления ключей. Новые спецификации называются DNSSEC-bis.

Смена ключей ZSK представляет собой относительно несложную задачу и не предусматривает наличия родительской зоны или какой-либо точки доверия. Единственным "скользким" местом является расписание. Ключи имеют определенный срок действия, поэтому их замену необходимо производить до истечения этого срока. Однако ключи имеют период TTL, определенный в файле зоны. Для иллюстрации предположим, что период TTL равен одному дню и что ключи на следующей неделе еще будут действовать. Придется выполнить следующие действия.

- Сгенерировать новый ключ ZSK.
- Включить его в файл зоны.
- Впервые или повторно подписать зону с помощью ключа KSK либо *старого* ключа ZSK.
- Попросить сервер имен перезагрузить зону; теперь в ней будет действовать новый ключ.
- Подождать 24 часа (период TTL); теперь все могут использовать как старый ключ, так и новый.

- Подписать зону снова с помощью ключа KSK и *нового* ключа ZSK.
- Попросить сервер имен перезагрузить зону.
- Подождать 24 часа; теперь все имеют новую зону.
- Удалить старый ключ ZSK, например при следующем изменении зоны.

Эта схема называется *предварительной публикацией* (prepublishing). Совершенно очевидно, что до того, пока все будут использовать новый ключ, приходится запускать этот процесс как минимум дважды после истечения двух периодов TTL. Эти периоды ожидания гарантируют, что любой сайт с кешированными значениями всегда будет иметь кешированный ключ, соответствующий этим кешированным данным.

Еще одной переменной, которая влияет на этот процесс, является время, которое потребуется вашему самому слабому подчиненному серверу для обновления своей копии вашей зоны после получения уведомления от главного сервера. По этой причине не следует ждать до последней минуты, чтобы начать процесс смены ключей или повторного подписания зон, срок действия подписей которых истек. Просроченные подписи недействительны, поэтому сайты, проверяющие подписи DNSSEC, не смогут выполнить поиск имен для вашего домена.

Механизм смены ключей KSK называется *двойным подписанием* (double signing) и также довольно прост. Однако вам придется переслать новую запись `DS` родительской зоне или послать запись `DLV` своему "суррогатному родителю". Убедитесь, что родительская зона или репозитарий точек доверия вас признали, прежде чем переключиться на новый ключ. Процесс смены ключей KSK состоит из следующих этапов.

- Создать новый ключ KSK.
- Включить его в файл зоны.
- Подписать зону и новым, и старым ключами KSK и ZSK.
- Попросить сервер имен перезагрузить зону.
- Подождать 24 часа (период TTL); теперь у всех есть новый ключ.
- Уведомить все точки доверия о новом значении KSK.
- После подтверждения удалить старую запись KSK из зоны.
- Заново подписать зону новыми ключами KSK и ZSK.

Инструменты DNSSEC

В дистрибутивный набор BIND 9.10 входит новый инструмент для отладки. Domain Entity Lookup and Validation Engine (DELV) выполняет поиск аналогично утилите **`dig`**, но при этом он лучше согласован с протоколом DNSSEC. Фактически утилита **`delv`** проверяет цепочку доверия DNSSEC с помощью того же самого кода, который использует демон **`named`** в пакете BIND 9.

Кроме инструментов DNSSEC, поставляемых с пакетом BIND, существует четыре набора инструментов для развертывания и тестирования: **`ldns`**, DNSSEC-Tools (бывший Sparta), RIPE и OpenDNSSEC (`opendnssec.org`).

Инструменты `ldns`, `nlnetlabs.nl/projects/ldns`

По словам сотрудников компании NLnet Labs, **`ldns`** — это библиотека программ для разработки инструментов DNS, включающая примеры, демонстрирующие использование этой библиотеки. Эти инструменты перечислены ниже вместе с кратким описа-

нием каждого из них. Все они находятся в каталоге **examples**, за исключением утилиты **drill**, имеющей свой собственный каталог в дистрибутивном пакете. Команды сопровождаются справочными страницами. Файл **README**, относящийся к верхнему уровню иерархии каталогов, содержит очень краткие инструкции по инсталляции.

- **ldns-chaos** показывает идентификатор сервера имен, хранящийся в классе CHAOS.
- **ldns-compare-zones** демонстрирует разницу между двумя файлами зон.
- **ldns-dpa** анализирует пакеты DNS с помощью файлов слежения **tcpdump**.
- **ldns-key2ds** преобразовывает запись DNSSEC в запись DS.
- **ldns-keyfetcher** извлекает открытые ключи DNSSEC для зон.
- **ldns-keygen** генерирует пары ключей TSIG и DNSSEC.
- **ldns-notify** проверяет обновления подчиненных серверов зон.
- **ldns-nsec3-hash** распечатывает запись NSEC для заданного имени в хешированном виде.
- **ldns-read-zone** читает зону и распечатывает ее в разных форматах.
- **ldns-revoke** устанавливает флаг отмены для ключа RR в протоколе DNSSEC (см. документ RFC54011).
- **ldns-rrsig** распечатывает в удобном для чтения виде даты истечения сроков действия ключей из записей RRSIG.
- **ldns-signzone** подписывает файл зоны с записью NSEC или NSEC3.
- **ldns-update** посылает пакет динамического обновления.
- **ldns-verify-zone** проверяет состояние записей RRSIG, NSEC или NSEC3.
- **ldns-walk** совершает обход зоны, используя записи NSEC протокола DNSSEC.
- **ldns-zcat** объединяет файлы зон, разбитые утилитой **ldns-zsplit** на фрагменты.
- **ldns-zsplit** разбивает зону на фрагменты, чтобы подписывать их параллельно.

Многие из этих инструментов очень просты и выполняют только одну рутинную операцию для системы DNS. Они были написаны в качестве примеров использования библиотеки **ldns** и демонстрируют, насколько простым становится код, когда библиотека берет всю тяжелую работу на себя.

Инструменты dnssec-tools.org

Комплект инструментов DNSSEc-Tools создан на основе инструментов сервера BIND, предназначенных для поддержки протокола DNSSEC, и включает в себя следующие команды.

- **dnspktflow** отслеживает поток пакетов DNS в последовательности запросов и ответов, перехваченных командой **tcpdump**, и создает диаграмму.
- **donuts** анализирует файлы зон в поисках ошибок и несоответствий.
- **donutsd** выполняет команду **donuts** через определенные интервалы времени и предупреждает о проблемах.
- **mapper** отображает файлы зон, демонстрируя защищенные и незащищенные фрагменты.
- **rollerd**, **rollctl** и **rollinit** осуществляют автоматическую смену ключей, используя схему предварительной публикации для ключей ZSK и метод двойного подписания для ключей KSK.

- **trustman** управляет точками доверия и включает реализацию смены ключей RFC5011.
- **validate** — инструмент для проверки подписи из командной строки.
- **zonesigner** генерирует ключи и подписи зон.

Веб-сайт содержит хорошую документацию и учебные пособия для всех этих инструментов. Исходный код доступен для загрузки и защищен лицензией BSD.

Инструменты RIPE, ripe.net

Инструменты компании RIPE функционируют как внешний компонент инструментов пакета BIND, предназначенных для поддержки протокола DNSSEC, и основное внимание уделяют вопросам управления. Их подробные сообщения при выполнении и упаковке многих аргументов и команд имеют более понятную форму.

Инструменты Open DNSSEC, opendnssec.org

OpenDNSSEC — это набор инструментов, которые получают неподписанные зоны, добавляют подписи и другие записи для протокола DNSSEC, а затем передают их авторитетным серверам имен для этой зоны. Эта автоматизация значительно упрощает начальную настройку протокола DNSSEC.

Отладка протокола DNSSEC

Протокол DNSSEC был разработан для того, чтобы обеспечить взаимодействие между подписанными и неподписанными зонами, а также между серверами имен, поддерживающими протокол DNSSEC и игнорирующими его. Следовательно, возможно постепенное развертывание протокола, что часто и происходит, хотя и не всегда.

DNSSEC — это распределенная система с многочисленными изменчивыми частями. Все проблемы порождают авторитетные серверы, распознаватели клиентов и линии связей между ними. Проблема, которая кажется локальной, может отразиться на удаленном пользователе, поэтому такие инструменты, как SecSpider и Vantages, отслеживающие распределенное состояние системы, могут оказаться очень полезными. Эти инструменты, а также утилиты, перечисленные выше, и регистрационные файлы вашего сервера имен являются основными орудиями отладки.

Сначала убедитесь, что вы перенаправили категорию вывода журнала регистрации по протоколу DNSSEC, указанную в файле **named.conf**, в файл на локальном компьютере. Полезно отделить все сообщения, связанные с протоколом DNSSEC, чтобы не записывать в этот файл никаких других категорий журнала регистрации. Рассмотрим пример спецификации журнала регистрации для демона **named**.

```
channel dnssec-log {
    file "/var/log/named/dnssec.log" versions 4 size 10m ;
    print-time yes ;
    print-category yes ;
    print-severity yes ;
    severity debug 3;
}
category dnssec { dnssec-log;}
```

В пакете BIND следует установить уровень отладки равным 3 или выше, чтобы увидеть этапы проверки, выполняемые рекурсивным сервером BIND при попытках проверить подпись. Этому уровню регистрации соответствуют две страницы регистрационной

информации для одной проверяемой подписи. Если вы отслеживаете работу загруженного сервера, то регистрационная информация от нескольких запросов будет перемежаться другими данными. Разбираться в этой путанице — очень сложная и утомительная задача.

Команда **drill** имеет два особенно полезных флага: **-T** — для отслеживания цепочки доверия от корня до указанного хоста и **-S** — для отслеживания подписей от указанного хоста обратно к корню. Рассмотрим практический пример вывода, полученного от команды **drill -S**, позаимствованный из документа *DNSSEC HOWTO* компании NLnet Labs.

```
$ drill -S -k ksk.keyfile example.net SOA
DNSSEC Trust tree:
example.net (SOA)
|---example.net. (DNSKEY keytag: 17000)
    |---example.net. (DNSKEY keytag: 49656)
    |---example.net. (DS keytag: 49656)
        |---net. (DNSKEY KEYTAG: 62972)

            |---net. (DNSKEY KEYTAG: 13467)
            |---net. (DS KEYTAG: 13467)
                |---. (DNSKEY KEYTAG: 63380)
                    |---net. (DNSKEY KEYTAG: 63276) ;; Chase successful
```

Если сервер имен, выполняющий проверку, не может проверить подпись, он возвращает сигнал SERVFAIL. Проблема может заключаться в неправильной конфигурации одной из зон, входящих в цепочку доверия, в фальшивых данных, поступивших от злоумышленника, или в настройке самого рекурсивного сервера, выполняющего проверку. Попробуйте выполнить команду **drill** и отследить подписи вдоль цепочки доверия, чтобы обнаружить источник проблемы.

Если все подписи окажутся верифицированными, то попытайтесь послать запрос проблемному сайту с помощью команды **dig**, а затем **dig +cd**. (Флаг **cd** отключает проверку подписей.) Примените их к каждой зоне, входящей в цепочку доверия, чтобы выяснить, в чем проблема. Вы можете перемещаться по цепочке доверия как вверх, так и вниз. Чаще всего причиной ошибки становятся устаревшие точки доверия или просроченные подписи.

16.11. Отладка сервера BIND

Сервер BIND предоставляет три основных инструмента для отладки: журнальная регистрация, управляющая программа и инструмент запросов их командной строки.

Журнальная регистрация на сервере BIND

📖 Система Syslog описывается в главе 10.

Возможности подсистемы журнальной регистрации демона **named** действительно впечатляют. Сервер BIND использует систему Syslog для записи сообщений об ошибках и аномалиях. В новейших версиях концепция журнальной регистрации обобщена: добавлен еще один уровень переадресации и поддерживается направление сообщений непосредственно в файлы. Прежде чем переходить к деталям, приведем мини-словарь терминов, связанных с журнальной регистрацией в пакете BIND (табл. 16.7).

Таблица 16.7. Лексикон пакета BIND

Термин	Что означает
Канал	Место, куда направляются сообщения, — система Syslog, файл или устройство **/dev/null**[a]
Категория	Класс сообщений, генерируемых демоном **named**, например сообщения о динамических обновлениях или об ответах на запросы
Модуль	Имя исходного модуля, который сгенерировал сообщение
Средство	Название средства системы Syslog; за DNS не закреплено собственное средство, но можно выбрать любое стандартное
Важность	Степень важности сообщения об ошибке

[a] **/dev/null/** — псевдоустройство, отбрасывающее всю входящую информацию.

Подсистема журнальной регистрации конфигурируется при помощи инструкции `logging` в файле **named.conf**. Сначала определяются каналы — возможные пункты доставки сообщений. Затем для различных категорий сообщений задаются каналы, куда они будут поступать.

При формировании сообщения ему назначаются категория, модуль и уровень важности. После этого оно рассылается по всем связанным с категорией и модулем каналам. В каждом канале имеется фильтр, определяющий, сообщения какого уровня важности можно пропускать. Каналы, ведущие в систему Syslog, подвергаются дополнительной фильтрации в соответствии с правилами, установленными в файле **/etc/syslog.conf**.

Общий вид инструкции `logging` таков.

```
logging {
     определение_канала
     определение_канала
     ...
     category имя_категории {
        имя_канала
        имя_канала
     ...
    };
};
```

Каналы

Аргумент *определение_канала* выглядит по-разному, в зависимости от того, ведет он к файлу или к системе Syslog. Для каждого канала необходимо выбрать либо тип `file`, либо тип `syslog`; совмещать их канал не может.

```
channel имя_канала {
    file путь [versions число_версий | unlimited] [size размер];
    syslog средство;
    severity важность;
    print-category yes | no;
    print-severity yes | no;
    print-time yes | no;
};
```

Для файлового канала аргумент *число_версий* сообщает о том, сколько резервных копий файла нужно хранить. Аргумент *размер* задает предельно допустимый размер файла (например, `2048`, `100k`, `20m`, `15g`, `unlimited`, `default`), по достижении кото-

рого произойдет автоматическая ротация. К примеру, если файловый канал называется **mylog**, то в процессе ротации появятся версии **mylog.0**, **mylog.1** и т.д.

Для каналов системы Syslog задается имя средства, указываемое при регистрации сообщений. Это может быть любое стандартное средство, но на практике единственным разумным выбором являются средства `daemon` и `local0–local7`.

📖 Список средств системы Syslog приводился в главе 10.

Остальные разделы в определении канала являются необязательными. Аргумент *важность* может принимать следующие значения (в порядке уменьшения приоритета): `critical`, `error`, `warning`, `notice`, `info` и `debug` (здесь может добавляться номер уровня, например `severity debug 3`). Значение `dynamic` соответствует текущему уровню отладки сервера.

Параметры `print` устанавливают или отменяют вывод различных префиксов сообщений. Система Syslog вставляет перед каждым сообщением метку времени, а также имя хоста, но не уровень важности или категорию. Существует также параметр, позволяющий отображать имя исходного файла (модуля), сгенерировавшего сообщение. Параметр `print-time` рекомендуется включать только для файловых каналов, поскольку в Syslog произойдет ненужное дублирование информации.

В табл. 16.8 перечислены четыре канала, которые определены по умолчанию. В большинстве случаев создавать новые каналы не требуется.

Таблица 16.8. Стандартные каналы журнальной регистрации пакета BIND

Имя канала	Назначение
`default_syslog`	Направляет сообщения уровня `info` и выше в систему Syslog от имени средства `daemon`
`default_debug`	Направляет сообщения в файл **named.run**; уровень важности устанавливается равным `dynamic`
`default_stderr`	Направляет сообщения в стандартный канал ошибок демона **named** с уровнем важности `info`
`null`	Отбрасывает все сообщения

Категории

В момент создания программы категории определяются программистом. Они организовывают сообщения по темам или функциональным возможностям, а не по важности. В табл. 16.9 приведен текущий список категорий сообщений.

Таблица 16.9. Категории сообщений пакета BIND

Категория	Что охватывает
`client`	Клиентские запросы
`config`	Ошибки анализа и обработки конфигурационного файла
`database`	Сообщения об операциях с базой данных
`default`	Категории, для которых не был явно назначен канал
`delegation-only`	Запросы, посланные в NXDOMAIN зонами, выполняющими только делегирование
`dispatch`	Диспетчеризация входящих пакетов среди модулей сервера
`dnssec`	Сообщения протокола DNSSEC
`edns-disabled`	Информация о серверах, вышедших из строя

Окончание табл. 16.9

Категория	Что охватывает
`general`	Неклассифицированные сообщения
`lame-servers`	Сообщения о серверах, которые, как предполагается, обслуживают зону, но на самом деле это не так[a]
`network`	Сетевые операции
`notify`	Сообщения об изменениях зон
`queries`	Короткое сообщение для каждого (!) запроса, принимаемого сервером
`resolver`	Операции преобразования имен, например рекурсивный поиск, выполняемый от имени клиента
`security`	Принятые или непринятые запросы
`unmatched`	Запросы, которые демон **named** не может классифицировать (неправильный класс, нет представления)
`update`	Сообщения о динамических обновлениях
`update-security`	Одобрение или отклонение запросов на обновление
`xfer-in`	Сообщения о передачах зон , принимаемых сервером
`xfer-out`	Сообщения о передачах зон , отправляемых сервером

[a]Это может быть как родительская, так и дочерняя зона.

Журнал запросов

Стандартный вид инструкции `logging` таков.

```
logging {
   category default { default_syslog; default_debug; };
};
```

После внесения существенных изменений в систему BIND необходимо просматривать журнальные файлы; возможно, стоит также повышать уровень отладки. После того как демон **named** вернется в стабильное состояние, настройте систему так, чтобы регистрировались только важные сообщения.

Журнал запросов — источник весьма полезной информации. На его основании можно проверить, работают ли директивы `allow`, узнать, кто посылает вам запросы, идентифицировать неправильно сконфигурированные клиенты и т.д.

Для того чтобы включить режим регистрации запросов, назначьте категории `queries` какой-нибудь канал. Регистрация в системе Syslog менее эффективна, чем непосредственно в файле, поэтому при регистрации каждого запроса создайте файловый канал, связанный с локальным диском. Зарезервируйте для журнала достаточно дискового пространства и будьте готовы отключить регистрацию, когда накопится достаточный объем данных (команда **rndc querylog** включает и отключает регистрацию запросов).

Иногда отладка представления является довольно сложной задачей, но, к счастью, представление, соответствующее конкретному запросу, можно занести в журнал вместе с запросом.

Ниже перечислены наиболее часто регистрируемые сообщения.

- *Неправильно сконфигурированный сервер* ("Lame server resolving xxx"). Если подобное сообщение поступает от одной из внутренних зон, значит, в конфигурации системы имеется ошибка. Если же в сообщении говорится о какой-то зоне в Интернете, то можно не беспокоиться: это "чужая" ошибка. Сообщения второго вида вполне можно отбрасывать, направляя их в канал `null`.

- *Отклонение запроса* ("...query (cache) xxx denied"). Причиной этого сообщения может быть неправильная конфигурация удаленного сайта, нарушение правил или ситуация, в которой некто делегировал вам зону, но вы ее не конфигурировали.
- *Просроченное время при распознавании: отключение EDNS* ("too many timeouts resolving xxx: disabling EDNS"). Это сообщение может возникнуть вследствие отказа брандмауэра, не пропускающего пакеты UDP, размер которых превышает 512 байт, и не допускающего фрагментирование. Кроме того, оно может свидетельствовать о проблемах на конкретном хосте. Следует убедиться, что проблема не связана с вашим брандмауэром, и рассмотреть возможность переадресации этих сообщений в нулевой канал.
- *Неожиданное распознавание RCODE (SERVFAIL)* ("unexpected RCODE (SEVFAIL) resolving xxx"). Это сообщение может быть признаком атаки или, что более вероятно, сигналом о том, что кто-то постоянно посылает запросы к неправильно сконфигурированной зоне.
- *Неправильная отсылка* ("Bad referral"). Это сообщение свидетельствует о неправильном взаимодействии серверов имен зоны.
- *Неавторитетен* ("Not authoritative for"). Подчиненный сервер не может получить авторитетные данные о зоне. Возможно, у него хранится неправильный адрес главного сервера или же тот не смог загрузить зону.
- *Зона отклонена* ("Zone rejected"). Демон **named** отказался загружать файл зоны, поскольку тот содержит ошибки.
- *Не найдены записи NS* ("No NS RR for"). В файле зоны после записи `SOA` не найдены записи `NS`. Возможно, они отсутствуют либо не начинаются с табуляции или какого-нибудь другого пробельного символа. Во втором случае они не присоединяются к зоне и, следовательно, интерпретируются неправильно.
- *Не задано стандартное значение TTL* ("No default TTL set"). Желательно задавать стандартное значение TTL посредством директивы `$TTL`, располагаемой в начале файла зоны. Ошибка свидетельствует о том, что такая директива отсутствует. В версии BIND 9 наличие этой директивы обязательно.
- *Нет корневых серверов имен для класса* ("No root nameservers for class"). Демону **named** не удается найти корневые серверы имен. Следует проверить файл корневых "подсказок" и подключение сервера к Интернету.
- *Адрес уже используется* ("Address already in use"). Порт, который нужен для работы демона **named**, занят другим процессом, возможно, другой копией демона. Если демон отсутствует в списке выполняющихся процессов, то, очевидно, он перед этим аварийно завершил свою работу и оставил открытым управляющий сокет утилиты **rndc**, который нужно найти и удалить. Хороший способ устранить проблему — остановить процесс с помощью утилиты **rndc** и перезапустить процесс **named**.

```
$ sudo rndc stop
$ sudo /usr/sbin/named ...
```

- *Обновление не выполнено* ("...updating zone xxx: update unsuccessful"). Имела место попытка динамического обновления зоны, которая была отвергнута вследствие установки `allow-update` или `update-policy` для зоны в файле **named.conf**. Это очень распространенное сообщение об ошибке, которая часто вызывается неправильной конфигурацией системы Windows.

Пример конфигурации журнала запросов в пакете BIND

Приведенный ниже фрагмент взят из файла **named.conf** одного из загруженных серверов имен TLD и представляет собой полную конфигурацию журнала запросов.

```
logging
    channel default_log { # Default channel, to a file
        file "log/named.log" versions 3 size 10m;
        print-time yes;
        print-category yes;
        print-severity yes;
        severity info;
    };
    channel xfer-log { # Zone transfers channel, to a file
        file "log/xfer.log" versions 3 size 10m;
        print-time yes;
        print-category yes;
        print-severity yes;
        severity info;
    };
    channel dnssec-log { # DNSSEC channel, to a file
        file "log/dnssec.log" versions 3 size 10m;
        severity debug 1;
        print-severity yes;
        print-time yes;
    };
    category default { default_log; defaula_debug; }
    category dnssec { dnssec-log; }
    category xfer-in { xfer-log; }
    category xfer-out { xfer-log; }
    category notify { xfer-log; }
};
```

Уровни отладки в пакете BIND

Уровни отладки демона **named** обозначаются целыми числами от 0 до 100. Чем выше число, тем больше текста содержит выходная информация. Уровень 0 выключает отладку. Уровни 1 и 2 отлично подходят для отладки конфигурационного файла и базы данных. Уровни выше 4 предназначены для разработчиков кода демона.

Отладку можно включить из командной строки, запуская демон **named** с флагом **-d**. Например, команда

```
# sudo named -d2
```

запускает демон **named** на уровне отладки 2. По умолчанию отладочная информация записывается в файл **named.run**, находящийся в каталоге запуска демона. Этот файл растет очень быстро, поэтому во время отладки будьте внимательны.

Отладку можно также включать в процессе работы демона **named** с помощью команды **rndc trace**, которая увеличивает уровень отладки на единицу. Команда **rndc notrace** выключает режим отладки. Кроме того, можно создать канал регистрации сообщений, в определение которого входит строка следующего вида.

```
severity debug 3;
```

Эта строка обеспечивает направление в канал всех отладочных сообщений вплоть до уровня 3. Другие директивы в определении канала указывают на то, куда в конечном

итоге попадут сообщения. Чем выше уровень важности, тем больше информации регистрируется.

Просматривая журнальные файлы и отладочные сообщения, можно заметить, как часто делаются ошибки в конфигурации DNS. Забыв поставить точку в конце имени, можно спровоцировать угрожающий рост трафика DNS. Помните: точка нужна в конце каждого полностью определенного имени домена.

Управление сервером имен с помощью команды `rndc`

Опции команды **rndc** перечислены в табл. 16.10. Если ввести имя команды **rndc** без аргументов, то она выведет на экран список доступных подкоманд и их краткое описание. В предыдущих версиях программы **rndc** использовались сигналы. Однако количество доступных подкоманд уже "перевалило" за 25, поэтому разработчики пакета BIND отказались от использования сигналов. Подкоманды, приводящие к созданию файлов, размещают их в каталоге, указанном в файле **named.conf** как начальный каталог демона **named**.

Таблица 16.10. Опции команды `rndc`[a]

Инструкция	Назначение
dumpdb	Записывает образ базы данных DNS в файл **named_dump.db**
flush [*представление*]	Очищает все кеши или кеши, заданные *представлением*
flushname *имя* [*представление*]	Удаляет *имя* из кеша сервера
freeze *зона* [*класс* [*представление*]]	Приостанавливает обновления динамической *зоны*
thaw *зона* [*класс* [*представление*]]	Возобновляет обновления динамической *зоны*
halt	Останавливает демон **named** без обработки отложенных обновлений
querylog	Переключает режим трассировки входящих запросов
notify *зона* [*класс* [*представление*]]	Повторно посылает *зоне* уведомления
notrace	Отключает режим отладки
reconfig	Повторно загружает конфигурационный файл и все новые зоны
recursing	Записывает текущие рекурсивные запросы в файл **named.recursing**
refresh *зона* [*класс* [*представление*]]	Планирует обновления для *зоны*
reload	Повторно загружает файл **named.conf** и файлы зон
reload *зона* [*класс* [*представление*]]	Повторно загружает только указанную *зону* или *представление*
retransfer *зона* [*класс* [*представление*]]	Повторно копирует данные для *зоны* из главного сервера
stats	Записывает статистическую информацию в файл **named.stats**
status	Отображает на экране текущий статус выполнения демона **named**
stop	Сохраняет отложенные обновления и останавливает демон **named**

Окончание табл. 16.10

Инструкция	Назначение
trace	Увеличивает уровень отладки на единицу
trace *приращение*	Увеличивает уровень отладки на *приращение*
validation *новое_состояние*	Включает/отключает проверку DNSSEC на лету

[a] Аргумент *класс* означает то же, что и в случае записей о ресурсах; для Интернета он обычно равен IN.

Команда **rndc reload** заставляет демон **named** заново прочитать конфигурационный файл и повторно загрузить файлы зон. Команду **reload** *зона* удобно применять на интенсивно эксплуатируемом сервере, когда изменению подвергается только одна зона, а остальные трогать не нужно. Кроме того, можно задать параметры *класс* и *представление*, чтобы загрузить только указанное представление данных зоны.

Обратите внимание на то, что команды **rndc reload** недостаточно, чтобы добавить совершенно новую зону; для этого требуется, чтобы демон **named** прочитал файл **named.conf** и новый файл зоны. Для новых зон следует использовать команду **rndc reconfig**, которая заново прочитывает файл конфигурации и загружает любую новую зону, не трогая существующие.

Команда **rndc freeze** *зона* останавливает динамические обновления и согласовывает журнал динамических обновлений с файлами зон. После замораживания зоны ее данные можно редактировать вручную. Поскольку зона заморожена, динамическое обновление происходить не будет. Завершив редактирование, выполните команду **rndc thaw** *зона*, чтобы принимать динамические обновления снова.

Команда **rndc dumpdb** заставляет демон **named** записать образ своей базы данных в файл **named_dump.db**. Этот файл очень большой и содержит не только локальные данные, но и данные, накопленные в кеше сервера имен.

Версии демона **named** и программы **rndc** должны совпадать, иначе будет выдано сообщение о несовпадении версий протокола. Обычно они устанавливаются на один и тот же компьютер, и расхождение версий может вызвать проблемы, если попытаться управлять демоном **named**, запущенном на другом компьютере.

Некорректное делегирование

Подавая заявку на регистрацию доменного имени, вы просите, чтобы основному серверу имен и администратору DNS была выделена (делегирована) часть дерева имен. Если не поддерживать работу домена или изменить адреса серверов имен, не обновив связующие записи родительского домена, будет иметь место так называемое *некорректное делегирование* (lame delegation).

Последствия некорректного делегирования могут оказаться весьма негативными. Если хотя бы один из ваших серверов окажется некорректным, то вся ваша система DNS снизит эффективность работы. Если все серверы имен являются некорректными, то ни один из них не будет доступен. Все запросы будут начинаться с корня, пока ответы будут кешироваться, поэтому некорректные серверы и неисправное программное обеспечение, которое не делает негативного кеширования ошибок SERVFAIL, увеличивают нагрузку на каждый хост, находящийся на пути от корня к некорректному домену.

Некорректное делегирование можно обнаружить с помощью инструмента под названием **doc** (domain obscenity control — проверка корректности домена) либо путем не-

посредственного просмотра записей из журнальных файлов.[26] Рассмотрим пример регистрационных записей.

```
Jul 19 14:37:50 nubark named[757]: lame server resolving 'w3w3.com'(in
   'w3w3.com'?): 216.117.131.52#53
```

Послав с помощью команды **dig** запрос о серверах имен для домена `w3w3.com` одному из серверов gTLD-домена `.com`, мы получим следующую информацию.

```
$ dig @e.gtld-servers.net w3w3.com ns
;; ANSWER SECTION:
w3w3.com    172800  IN  NS  ns0.nameservices.net.
w3w3.com    172800  IN  NS  ns1.nameservices.net.
```

Если же теперь опросить каждый из этих серверов по очереди, задав им этот же вопрос, то мы получим ответ от сервера `ns0` и не получим ответа от сервера `ns1`.

```
$ dig @ns0.nameservices.net w3w3.com ns
;; ANSWER SECTION:
w3w3.com    14400  IN  NS  ns0.nameservices.net.
w3w3.com    14400  IN  NS  ns1.nameservices.net.

$ dig @ns1.nameservices.net w3w3.com ns
;; QUESTION SECTION:
;; w3w3.com.    IN  IN

;; AUTHORITY RECORDS:
com.        92152 IN  NS  M.GTLD-SERVERS.NET.
com.        92152 IN  NS  I.GTLD-SERVERS.NET.
com.        92152 IN  NS  E.GTLD-SERVERS.NET.
```

Сервер имен `ns1.nameservices.net` делегировал ответственность за домен `w3w3.com` серверам домена `.com`, но они эту ответственность не приняли. Эта конфигурация является неправильной, и возникло некорректное делегирование. Клиенты, пытающиеся попасть в домен `w3w3.com`, будут обслуживаться медленно. Если домен `w3w3.com` оплачивает услуги DNS домену `nameservices.net`, то он заслуживает компенсации!

Иногда, посылая запрос с помощью команды **dig** на авторитетный сервер в поисках некорректности, можно не получить никакой информации. В этом случае следует попытаться повторить запрос с флагом **+norecurse**, чтобы можно было увидеть точно, что знает искомый сервер.

16.12. Литература

Система доменных имен и пакет BIND описаны во многих источниках, включая документацию, входящую в состав пакета, отдельные главы в книгах об Интернете, целую книгу серии *In a Nutshell* издательства O'Reilly, а также многочисленные ресурсы в глобальной сети.

[26] Во многих организациях в качестве канала `lame-servers` для журналов регистрации указан каталог **/dev/null**, чтобы не беспокоиться о некорректном делегировании других доменов, не принадлежащих этой организации. Это допустимо, если ваш домен находится в полном порядке и не является ни источником, ни жертвой некорректного делегирования.

Книги и другая документация

- The Nominum BIND Development Team. *BINDv9 Administrator Reference Manual.* Документ включен в дистрибутив BIND (**doc/arm**), а также доступен на веб-сайте `www.isc.org`. В нем в общих чертах описаны принципы администрирования пакета BIND 9.
- Liu, Cricket, and Paul Albitz. *DNS and BIND (5th Edition).* Sebastopol, CA: O'Reilly Media, 2006. Эта книга стала настольным справочником по серверу BIND, хотя и написана довольно тяжелым языком.
- Liu, Cricket. *DNS & BIND Cookbook.* Sebastopol, CA: O'Reilly Media, 2002. Это упрощенная версия книги издательства O'Reilly о системе DNS, содержащая четкие инструкции и примеры работы с серверами имен. Довольно старая, но все еще полезная.
- Liu, Cricket. *DNS and BIND on IPv6.* Sebastopol, CA: O'Reilly Media, 2011. Книга ориентирована на DNS и BIND в рамках протокола IPv6. Она невелика и содержит исключительно материал, связанный с протоколом IPv6.
- Lucas, Michael W. *DNSSEC Mastery: Securing the Domain Name System with BIND.* Grosse Point Woods, MI: Tilted Windmill Press, 2013.

Ресурсы в Интернете

Веб-сайты `isc.org`, `dns-oarc.net`, `ripe.net`, `nlnetlabs.nl`, `f.root-servers.org` и `k.root-servers.org` содержат много информации о системе DNS, исследованиях, результатах измерений, презентациях и др.

Подробная информация о протоколе DNS, записях о ресурсах и других аспектах DNS приведена на сайте `iana.org/assignments/dns-parameters`. Этот документ содержит информацию о том, в каком документе RFC описан тот или иной факт, касающийся работы системы DNS.

Справочник *DNSSEC HOWTO*, написанный Олафом Колкманом (Olaf Kolkman), — это 70-страничный документ, в котором изложены все аспекты развертывания и отладки протокола DNSSEC. Его можно загрузить с сайта `nlnetlabs.nl/dnssec_howto/dnssec_howto.pdf`.

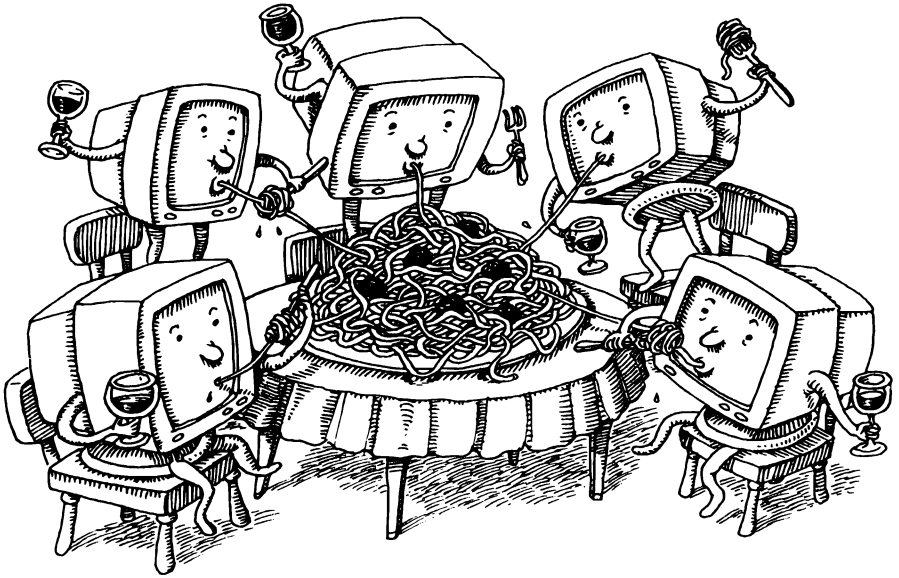
Документы RFC

Документы RFC, в которых описывается система доменных имен, доступны на веб-сайте `www.rfc-editor.org`. Мы использовали только самые важные из них, хотя в настоящее время есть около 100 документов и около 50 черновиков, опубликованных в Интернете, поэтому рекомендуем читателям зайти на сайт `www.rfc-editor.org` и изучить весь архив.

Для того чтобы увидеть все документы, касающиеся текущей версии BIND, найдите каталоги **doc/rfc** и **doc/draft**.

Глава 17

Система единого входа



Пользователи и системные администраторы хотели бы, чтобы информация об учетной записи волшебным образом распространялась на все компьютеры сети и пользователь мог войти в любую систему с одинаковыми учетными данными. Эта функция называется "единым входом" (single sing-on — SSO) и потребность в ней повсеместна.

SSO включает в себя две основные концепции безопасности: идентификационную информацию и аутентификацию. Идентификационная информация пользователя — это абстрактное представление лица, которому необходим доступ к системе или приложению. Обычно она включает такие атрибуты, как имя пользователя, пароль, идентификатор пользователя и адрес электронной почты. Аутентификация — это доказательство того, что лицо является законным владельцем идентификационной информации.

В этой главе основное внимание уделяется системе SSO как компоненту систем UNIX и Linux в рамках единой организации. Для единого входа в нескольких организациях (который, например, может потребоваться для интеграции ваших систем и систем поставщика программного обеспечения в рамках модели обслуживания "программное обеспечение как услуга" (Software-as-a-Service — SaaS)) доступны несколько решений на основе стандартов и коммерческих решений SSO. В этих случаях в качестве первого шага мы рекомендуем изучить язык Security Assertion Markup Language (SAML).

17.1. Основные элементы системы единого входа

Хотя существует множество способов создания единого входа, в каждом сценарии обычно требуется четыре элемента.

- Централизованное хранилище каталогов, которое содержит идентификационные данные пользователя и информацию авторизации. Наиболее распространенными решениями являются службы каталогов, основанные на протоколе LDAP (Lightweight Directory Access Protocol). В средах, в которых сочетаются системы Windows, UNIX и Linux, хорошим выбором является популярная служба Microsoft Active Directory, включающая настраиваемый, нестандартный интерфейс LDAP.
- Инструмент для управления информацией пользователя в каталоге. Для собственных реализаций LDAP мы рекомендуем phpLDAPadmin или Apache Directory Studio. Оба являются простыми в использовании веб-инструментами, которые позволяют импортировать, добавлять, изменять и удалять записи в каталоге. Если вы являетесь приверженцем службы Microsoft Active Directory, то для управления информацией в каталоге вы можете использовать интегрируемый модуль Windows "Active Directory Users and Computers".
- Механизм аутентификации идентификационной информации пользователей. Вы можете аутентифицировать пользователей непосредственно в хранилище LDAP, но помимо этого часто используется система аутентификации на основе мандатов на основе протокола Kerberos, первоначально разработанная в MIT.[1] В средах Windows Active Directory предоставляет доступ LDAP к идентификаторам пользователей и использует настраиваемую версию Kerberos для аутентификации.
- Аутентификация в современных системах UNIX и Linux выполняется системой Pluggable Authentication Module, также известной как PAM. Для агрегирования доступа к службам идентификации и аутентификации пользователя вы можете использовать демон System Security Service Daemon (**sssd**), а затем указать PAM в **sssd**.
- Библиотеки подпрограмм на языке C для централизованной идентификации и аутентификации, которые ищут пользовательские атрибуты. Эти утилиты (например, **getpwent**) традиционно читают обычные файлы, такие как **/etc/passwd** и **/etc/group**, и отвечают на запросы, основываясь на их содержании. В настоящее время источники данных настраиваются в файле переключения службы имен, **/etc/nsswitch.conf**.

На рис. 17.1 показаны отношения высокого уровня между различными компонентами SSO в типичной конфигурации. В этом примере в качестве сервера каталогов используется Active Directory. Обратите внимание на то, что как временная синхронизация (NTP), так и отображение имен хостов (DNS) имеют решающее значение для сред, использующих Kerberos, поскольку мандаты аутентификации имеют временные метки и ограниченный срок действия.

В этой главе мы рассмотрим основные концепции системы LDAP и представим два конкретных LDAP-сервера для систем UNIX и Linux. Затем мы обсудим шаги, необходимые для того, чтобы машина использовала централизованную службу каталогов для обработки авторизации.

[1] Сообщество специалистов по системам безопасности делится на тех, кто считает, что наилучшую защиту аутентификации обеспечивает служба LDAP, и сторонников протокола Kerberos. Дорога жизни усеяна расплющенными белками, которые не могли сделать выбор. Выберите свой вариант и не оглядывайтесь назад.

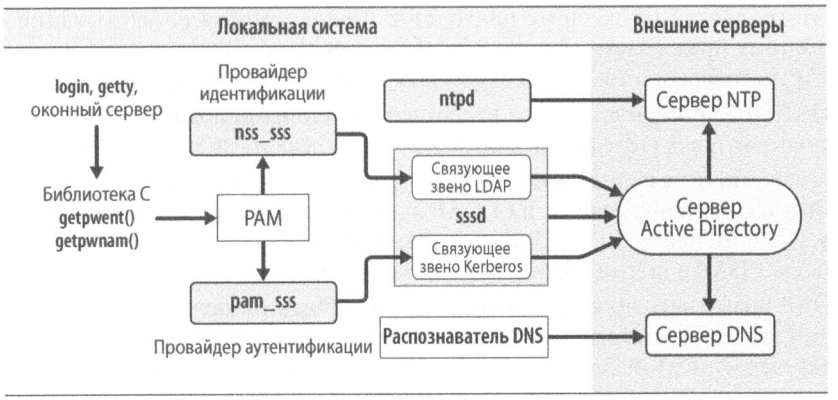


Рис. 17.1. Компоненты SSO

17.2. LDAP: "облегченные" службы каталогов

Служба каталогов — это просто база данных, но такая, в которой сделан ряд дополнительных предположений. Любой набор данных, характеристики которых подпадают под эти предположения, становится кандидатом на включение в базу данных. Основные предположения таковы:

- информационные объекты относительно невелики;
- база данных будет реплицироваться и кешироваться на множестве компьютеров;
- у информации есть атрибуты;
- данные извлекаются часто, но записываются редко;
- операции поиска выполняются очень часто.

Текущая стандартная система, предложенная организацией IETF для этих целей, называется LDAP (Lightweight Directory Access Protocol — упрощенный протокол доступа к каталогам).[2] Изначально LDAP задумывался как простой шлюзовой протокол, который позволял бы клиентам TCP/IP взаимодействовать с серверами каталогов X.500, теперь уже устаревшими.

Наиболее распространенной реализацией протокола LDAP является служба Active Directory компании Microsoft. Многие организации используют эту службу для аутентификации как в системе Windows, так и в системах UNIX и Linux. Для таких организаций стандартной реализацией стал пакет OpenLDAP (`openldap.org`). Служба каталогов уровня предприятия, остроумно названная *389 Directory Server* (ранее известная как Fedora Directory Server и Netscape Directory Server), также является проектом с открытым исходным кодом, к которому можно получить доступ на сайте `port389.org`.[3]

Особенности LDAP

Для уверенной работы с LDAP вам нужен хотя бы небольшой опыт. Протокол LDAP сам по себе не решает какую-либо административную проблему. Нет какой-то "главной задачи", которую можно было бы решить исключительно с помощью LDAP; к тому

[2]Как это ни парадоксально, но протокол LDAP можно назвать каким угодно, но только не упрощенным.

[3]TCP-порт 389 является портом по умолчанию для всех реализаций протокола LDAP.

же на разных сайтах по-разному подходят к необходимости развертывания серверов LDAP. Поэтому прежде чем перейти к особенностям инсталляции и конфигурирования OpenLDAP, поговорим о том, в каких случаях целесообразно использовать LDAP.

- LDAP можно использовать в качестве хранилища дополнительной информации о пользователях (телефонных номеров, домашних и рабочих адресов).
- Многие почтовые системы, включая **sendmail**, Exim и Postfix, могут получить данные о маршрутизации из LDAP, и этим отчасти можно объяснить популярность применения LDAP. Дополнительную информацию об использовании протокола LDAP в почтовой системе **sendmail** см. в разделе 18.8.
- LDAP позволяет упростить процесс аутентификации пользователей для приложений (даже тех, которые написаны другими командами программистов и в других подразделениях), избавляя разработчиков от необходимости беспокоиться о точных деталях управления учетными записями.
- Изменения, внесенные в LDAP-данные, немедленно вступают в силу и мгновенно становятся видимыми для всех хостов и приложений клиентов.
- LDAP всесторонне поддерживается языками написания сценариев вроде Perl и Python (посредством использования библиотек). Следовательно, LDAP удобно использовать для передачи информации о конфигурации локально написанным сценариям и утилитам администрирования.
- LDAP хорошо поддерживается и как служба общедоступных каталогов. Многие ведущие почтовые клиенты применяют LDAP для доступа к каталогам пользователей. Простой поиск с использованием LDAP поддерживается во многих веб-браузерах посредством типа LDAP URL.

Структура данных LDAP

Данные протокола LDAP принимают форму списков свойств, которые в мире LDAP называют "записями" (entry). Каждая запись состоит из набора именованных атрибутов (например, `uid`, `description`) и значений этих атрибутов. Пользователи, работающие в среде Windows, вероятно, заметят, что эта структура напоминает структуру записей в системном реестре. Как и в реестре Windows, отдельный атрибут может иметь несколько различных значений.

В качестве примера рассмотрим обычную (и упрощенную) строку файла **/etc/passwd**, выраженную в виде записи LDAP.

```
dn: uid=ghopper,ou=People,dc=navy,dc=mil
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: shadowAccount
uid: ghopper
cn: Grace Hopper
userPassword: {crypt}$1$pZaGA2RL$MPDJoc0afuhHY6yk8HQFp0
loginShell: /bin/bash
uidNumber: 1202
gidNumber: 1202
homeDirectory: /home/ghopper
```

Здесь мы видим простой пример формата LDIF (LDAP Data Interchange Format — формат обмена данными LDAP), который применяется в большинстве инструментов, связан-

ных с LDAP, и в реализациях серверов. Причиной успеха этого формата является то обстоятельство, что LDAP можно без труда преобразовывать в простой текст и обратно.

Записи образуют иерархию по "отличительным именам" (имя атрибута: `dn` — distinguished name), которые формируют некоторую разновидность пути поиска. Как и в системе DNS, "самый старший бит" находится справа. Например, в приведенном выше примере имя DNS `navy.mil` используется для построения верхних уровней иерархии LDAP. Оно разбивается на два компонента домена: `navy` и `mil`, хотя это всего лишь одно из некоторых обычных соглашений.

Каждая запись имеет только одно отличительное имя. Записи совершенно изолированы друг от друга и не образуют иерархии, за исключением иерархии, неявно определяемой атрибутами `dn`. Это гарантирует их уникальность и подсказывает реализации протокола, как эффективно индексировать и искать данные. Виртуальную иерархию, созданную атрибутами `dn`, применяют разные пользователи протокола LDAP, но она является скорее соглашением о структуризации данных, чем явной функциональной возможностью протокола LDAP. В то же время существуют возможности для создания символических связей между записями и ссылками на другие серверы.

Записи LDAP обычно составляются на основе использования атрибута `objectClass`. Классы объектов определяют атрибуты, которые может содержать запись, причем некоторые из них могут требоваться для проверки достоверности. Каждому атрибуту назначается тип данных. Схема вложения и комбинирования классов объектов ничем не отличается от схемы, принятой в традиционном объектно-ориентированном программировании. Верхний уровень дерева класса объектов называется `top`; он означает лишь то, что запись должна иметь атрибут `objectClass`.

В табл. 17.1 приведены некоторые обычные атрибуты LDAP, назначение которых с первого взгляда может быть непонятным.

Таблица 17.1. Некоторые имена атрибутов, которые можно встретить в иерархиях LDAP

Атрибут	Расшифровка	Назначение
`o`	Organization (организация)	Часто идентифицирует запись верхнего уровня сайта[a]
`ou`	Organization unit (организационная единица)	Логическое подразделение, например `marketing` (отдел маркетинга)
`cn`	Common name (обычное имя)	Наиболее естественное имя, с помощью которого можно передать смысл записи
`dc`	Domain component (компонент домена)	Используется в сайтах, в которых иерархия построена на основе DNS
`objectClass`	Object class (класс объектов)	Схема, в соответствии с которой формируются атрибуты данной записи

[a] Обычно не используется системами, в которых LDAP-иерархия построена на основе DNS.

OpenLDAP: традиционный LDAP-сервер с открытым исходным кодом

OpenLDAP — это расширение проекта, выполненного в Мичиганском университете. В настоящее время он продолжает оставаться проектом с открытым исходным текстом и распространяется с большинством дистрибутивов Linux (хотя и не всегда включается в стандартный вариант инсталляции). Его документацию, вероятно, лучше всего характеризует слово "оживленная".

В дистрибутиве OpenLDAP стандартным демоном LDAP-сервера является **slapd**. В среде с несколькими серверами OpenLDAP демон **slurpd** выполняется на главном сервере и отрабатывает механизм репликации посредством внесения изменений на подчиненные серверы. Утилиты командной строки позволяют выполнять запросы и модифицировать данные LDAP.

Настройка довольно простая. В первую очередь потребуется создать файл **/etc/openldap/slapd.conf**, скопировав образец, который был инсталлирован вместе с сервером OpenLDAP. Необходимо обратить внимание на следующие строки.

```
database bdb
suffix "dc=mydomain,dc=com"
rootdn "cn=admin,dc=mydomain,dc=com"
rootpw {crypt}abJnggxhB/yWI
directory /var/lib/ldap
```

По умолчанию выбирается формат базы данных Berkley DB, отлично подходящий для данных, манипуляция которыми будет производиться в системе OpenLDAP. Вы можете использовать разнообразные интерфейсы, включая специальные методы (например, сценарии, создающие данные "на лету").

Переменная **suffix** задает "базовое имя" LDAP. Это корень вашей части пространства имен LDAP, подобный тому, что в DNS называется доменным именем. Этот пример показывает обычное использование доменного имени в качестве базового имени LDAP.

Переменная **rootdn** задает административное имя, а переменная **rootpw** — хешированный пароль администратора. Обратите внимание на то, что доменные компоненты, восходящие к административному имени, также должны быть указаны. Для генерирования значения для этого поля можно использовать утилиту **slappaswd**; просто скопируйте и вставьте в поле результат работы этой утилиты.

Поскольку пароль хешируется, проверьте, что файл **slapd.conf** принадлежит привилегированному пользователю, а его уровень допуска равен 600.

Нужно также отредактировать файл **/etc/openldap/ldap.conf**, чтобы указать сервер по умолчанию и базовое имя для клиентских запросов LDAP. Это несложно: просто задайте аргумент записи **host** равным имени сервера, а в переменную **base** занесите то же значение, что и в переменной **suffix** файла **slapd.conf** (убедитесь, что обе строки не являются комментариями). Вот как выглядит пример для сайта atrust.com.

```
BASE   dc=atrust,dc=com
URI    ldap://atlantic.atrust.com
```

С этого момента можно запускать демон **slapd** без аргументов.

389 Directory Server: альтернативный LDAP-сервер с открытым исходным кодом

Подобно OpenLDAP, служба каталогов уровня предприятия 389 Directory Server (**port389.org**) является расширением проекта, выполненного в Мичиганском университете. Однако прошло несколько лет (в компании Netscape Communications), прежде чем этот проект снова стал открытым.

Существует множество причин рассматривать проект 389 Directory Server как альтернативу для OpenLDAP, но среди его явных преимуществ все же стоит выделить более совершенную документацию. Проект 389 Directory Server поставляется с инструкциями профессионального уровня по администрированию и использованию, включая подробные руководства по инсталляции и внедрению.

К ключевым особенностям проекта 389 Directory Server можно отнести следующие:

- использование нескольких полностью равноправных мастер-серверов, что позволяет обеспечить отказоустойчивость и высокую скорость выполнения операций записи;
- возможность синхронизации пользователей, групп и паролей с контроллерами домена Active Directory;
- консоль администрирования с графическим интерфейсом, управления из командной строки и через веб-интерфейс;
- поддержка протокола LDAP, оперативное (без потерь времени) обновление схемы, конфигурации и управления, а также механизм Access Control Information (ACI).

Проект 389 Directory Server имеет больше активных разработчиков, чем проект OpenLDAP. Поэтому мы обычно рекомендуем отдавать предпочтение именно проекту 389 Directory Server.

С административной точки зрения эти два сервера с открытым исходным кодом поразительно сходны в том, что касается структуры и функционирования. И это не удивительно, поскольку оба пакета были построены на базе одного и того же исходного кода.

Создание LDAP-запросов

Для администрирования LDAP вам не обойтись без просмотра содержимого базы данных и управления им. Для этого вам очень пригодится упомянутый выше свободно распространяемый пакет phpLDAPadmin, который предоставляет удобный интерфейс, работающий по принципу "указал и щелкнул". Помимо phpLDAPadmin, можно использовать утилиту **ldapsearch** (распространяемую с OpenLDAP и 389 Directory Server), которая предназначена для поиска информации в службе каталога и является аналогичным инструментом командной строки, генерирующим результат в формате LDIF. Утилита **ldapsearch** особенно полезна для вызова из сценариев, а также для сред отладки, в которых служба Active Directory действует в качестве сервера LDAP.

В следующем примере запроса используется утилита **ldapsearch** для просмотра информации о каталогах тех пользователей, у которых обычное имя `cn` (*c*ommon *n*ame) начинается с "`ned`". В данном случае найден только один результат, соответствующий такому запросу. Назначения используемых здесь флагов рассматриваются ниже.

```
$ ldapsearch -h atlantic.atrust.com -p 389
    -x -D "cn=trent,cn=users,dc=boulder,dc=atrust,dc=com" -W
    -b "cn=users,dc=boulder,dc=atrust,DC=com" "cn=ned*"

Enter LDAP Password:  пароль

# LDAPv3
# base <cn=users,dc=boulder,dc=atrust,dc=com> with scope sub
# filter: cn=ned*
# requesting: ALL
#
# ned, Users, boulder.atrust.com
dn: cn=ned,cn=Users,dc=boulder,dc=atrust,dc=com
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: user
cn: ned
```

```
sn: McClain
telephoneNumber: 303 245 4505
givenName: Ned
distinguishedName: cn=ned,cn=Users,dc=boulder,dc=atrust,dc=com
displayName: Ned McClain
memberOf: cn=Users,cn=Builtin,dc=boulder,dc=atrust,dc=com
memberOf: cn=Enterprise Admins,cn=Users,dc=boulder,dc=atrust,dc=com
name: ned
sAMAccountName: ned
userPrincipalName: ned@boulder.atrust.com
lastLogonTimestamp: 129086952498943974
mail: ned@atrust.com
```

Флаги **-h** и **-p** команды **ldapsearch** позволяют указать в запросе хост и порт сервера LDAP соответственно.

Обычно вам потребуется аутентифицировать себя на сервере LDAP. В этом случае используются специальные флаги. Флаг **-x** запрашивает простую аутентификацию (в отличие от SASL). Флаг **-D** идентифицирует отличительное имя учетной записи пользователя, который имеет права доступа, необходимые для выполнения запроса. Наконец, флаг **-W** предписывает утилите **ldapsearch** предложить вам ввести соответствующий пароль.

Флаг **-b** указывает утилите **ldapsearch**, где именно в иерархии LDAP начать поиск. Этот параметр известен как **baseDN** (отсюда и имя флага "**b**"). По умолчанию утилита **ldapsearch** возвращает все соответствующие критерию поиска строки, найденные ниже базовой точки дерева **baseDN**, т.е. поиск будет выполняться только в дочерних элементах базы поиска (включая ее саму). Уточнить характер такого поиска можно с помощью флага **-s**.

Последний аргумент представляет собой "фильтр", который описывает объект вашего поиска. Он не требует использования флагов. Этот фильтр, **cn=ned***, обеспечивает возвращение всех строк LDAP, "обычное имя" которых начинается с "**ned**". Этот фильтр заключается в кавычки (**"cn=ned*"**), чтобы специальные символы универсализации в строке поиска (в данном случае это "звездочка") не были восприняты системой как управляющие символы командной оболочки.

Если вы хотите извлечь все записи ниже заданной базы `baseDN`, просто используйте в качестве фильтра поиска выражение **objectClass=*** — или же опустите фильтр вообще, поскольку такой характер поиска действует по умолчанию.

Любые аргументы, которые указаны за фильтром, обеспечивают выбор конкретных атрибутов для возвращаемого результата. Например, если бы вы добавили в конец приведенной выше командной строки аргумент **mail givenName**, утилита **ldapsearch** возвратила бы только эти атрибуты отфильтрованных записей.

Преобразования файлов паролей и групп LDAP

Если вы переходите на протокол LDAP и у вас уже есть информация о пользователях и группах, то вам, возможно, потребуется перенести существующие данные. В документе RFC2307 определено стандартное преобразование традиционных наборов данных UNIX, к которым относятся файлы **passwd** и **group**, в пространство имен LDAP. Это полезный справочный документ (по крайней мере с теоретической точки зрения). На практике все эти спецификации читать гораздо проще компьютерам, чем людям; вам же лучше просто просмотреть примеры.

Фирма Padl Software предлагает бесплатную коллекцию сценариев, которые переводят существующие текстовые файлы или карты NIS в LDAP. Эти сценарии можно най-

ти по адресу: **padl.com/OSS/MigrationTools.html**. Работать с ними очень просто. Сценарии можно использовать в качестве фильтров для генерации LDIF, а также можно запускать, чтобы загружать данные прямо с сервера. Например, сценарий **migrate_group** преобразовывает взятую из **/etc/group** строку

```
csstaff:x:2033:evi,matthew,trent
```

в следующий блок LDIF:

```
dn: cn=csstaff,ou=Group,dc=domainname,dc=com
cn: csstaff
objectClass: posixGroup
objectClass: top
userPassword: {crypt}x
gidNumber: 2033
memberuid: evi
memberuid: matthew
memberuid: trent
```

17.3. Использование служб каталогов для входа в систему

После инсталляции службы каталогов выполните следующие задачи по настройке, чтобы ваша система могла использовать SSO.

- Если вы планируете использовать службу Active Directory с протоколом Kerberos, настройте Kerberos и присоедините систему к домену Active Directory.
- Настройте утилиту **sssd** для связи с соответствующими хранилищами идентификации и аутентификации (LDAP, Active Directory или Kerberos).
- Настройте ключ службы имен, **nsswitch.conf**, чтобы использовать утилиту **sssd** в качестве источника информации о пользователе, группе и пароле.
- Настройте PAM для обслуживания запросов аутентификации с помощью демона **sssd**.[4]

Ниже мы рассмотрим эти процедуры.

Система Kerberos

Kerberos — это система аутентификации на основе мандатов, использующая криптографию с симметричным ключом. Его популярность в последнее время была обеспечена прежде всего компанией Microsoft, которая использует ее как часть системы аутентификации в службе Active Directory и операционной системе Windows. В контексте SSO мы описываем, как интегрироваться со средой Active Directory Kerberos в системах Linux и FreeBSD. Если вы используете LDAP-сервер, отличный от Active Directory, или если вы хотите пройти аутентификацию в службе Active Directory с помощью LDAP, а не Kerberos, вы можете сразу перейти к обсуждению демона **sssd**.

🕮 Дополнительную информацию о системе Kerberos см. в разделе 27.6.

[4]Одна часть программного обеспечения использует традиционное семейство библиотечных процедур **getpwent** для поиска информации о пользователе, но современные службы часто напрямую вызывают процедуры аутентификации PAM. Чтобы обеспечить полностью функциональную среду, настройте и PAM, и **nsswitch.conf**.

Конфигурация Linux Kerberos для интеграции службы Active Directory



Системные администраторы часто хотят, чтобы их системы Linux были членами домена Active Directory. Раньше сложность этой конфигурации приводила к тому, что некоторые из системных администраторов приходили в отчаяние. К счастью, появление пакета **realmd** намного упростило эту задачу. Пакет **realmd** действует в качестве инструмента настройки как для демона **sssd**, так и для системы Kerberos.

Прежде чем пытаться присоединиться к домену Active Directory, проверьте следующие условия.

- В системе Linux, которую вы присоединяете к домену, инсталлирован пакет **realmd**.
- Демон **sssd** инсталлирован (см. ниже).
- Демон **ntpd** установлен и запущен.
- Вы знаете правильное имя своего домена Active Directory.
- У вас есть учетные данные для учетной записи Active Directory, которой разрешено присоединяться к системе в домене. Это действие приводит к выдаче мандата на выдачу мандатов в Kerberos (ticket granting ticket — TGT), чтобы он мог выполнять операции аутентификации в будущем без доступа к паролю администратора.

Например, если ваше доменное имя Active Directory — ULSAH.COM, а учетная запись Active Directory под именем `trent` имеет право на присоединение системы к домену, вы можете использовать следующую команду для присоединения вашей системы к домену.

```
$ sudo realm join --user=trent ULSAH.COM
```

Затем вы можете проверить результат.

```
$ realm list
ulsah.com
  type: kerberos
  realm-name: ULSAH.COM
  domain-name: ulsah.com
  configured: kerberos-member
  server-software: active-directory
  client-software: sssd
  required-package: sssd
  required-package: adcli
  required-package: samba-common
  login-formats: %U@ulsah.com
  login-policy: allow-real logins
```

Конфигурация Kerberos FreeBSD для интеграции AD



Система Kerberos печально известна своим сложным процессом настройки, особенно на стороне сервера. К сожалению, система FreeBSD не имеет надежного инструмента, похожего на пакет **realmd** в системе Linux, который настраивает Kerberos и объединяет домен Active Directory за один шаг. Однако вам необходимо настроить только клиентскую сторону Kerberos. Конфигурационным файлом является **/etc/krb5.conf**.

Во-первых, дважды проверьте, что полное доменное имя системы включено в файл `/etc/hosts`, а протокол NTP настроен и работает. Затем отредактируйте файл **`krb5.conf`**, чтобы добавить область, как показано в следующем примере. Замените имя домена Active Directory вашего сайта для ULSAH.COM.

```
[logging]
    default = FILE:/var/log/krb5.log
[libdefaults]
    clockskew = 300
    default_realm = ULSAH.COM
    kdc_timesync = 1
    ccache_type = 4
    forwardable = true
    proxiable = true
[realms]
    ULSAH.COM = {
        kdc = dc.ulsah.com
        admin_server = dc.ulsah.com
        default_domain = ULSAH
    }
[domain_realm]
    .ulsah.com = ULSAH.COM
    ulsah.com = ULSAH.COM
```

В приведенном выше примере интерес представляют несколько значений. Пятиминутное рассогласование часов разрешено, даже если время установлено с помощью протокола NTP. Эта свобода позволяет системе работать даже в случае проблем с протоколом NTP. Область по умолчанию установлена в домене Active Directory, а центр распределения ключей (или KDC) настроен как контроллер домена Active Directory. Для отладки может пригодиться файл `krb5.log`.

Запросите мандат из контроллера Active Directory, запустив команду **`kinit`**. Укажите действительную учетную запись пользователя домена. Аккаунт `administrator` обычно является хорошим тестом, но подойдет и любая учетная запись. При появлении запроса введите пароль домена.

```
$ kinit administrator@ULSAH.COM
Password for administrator@ULSAH.COM: <password>
```

Используйте команду **`klist`**, чтобы показать мандат Kerberos.

```
$ klist
Ticket cache: FILE:/tmp/krb5cc_1000
Default principal: administrator@ULSAH.COM

Valid starting     Expires            Service principal
04/30/17 13:40:19  04/30/17 23:40:21  krbtgt/ULSAH.COM@ULSAH.COM
        renew until 05/01/17 13:40:19

Kerberos 4 ticket cache: /tmp/tkt1000
klist: You have no tickets cached
```

Если мандат отображается на экране, то аутентификация прошла успешно. В этом случае мандат действителен в течение 10 часов и может быть продлен на 24 часа. Для аннулирования мандата можно использовать команду **`kdestroy`**.

Последний шаг — присоединить систему к домену, как показано ниже. Используемая учетная запись администратора (в данном случае `trent`) должна иметь соответствующие полномочия на сервере Active Directory для присоединения систем к домену.

```
$ net ads join -U trent
Enter trent's password: <password>
Using short domain -- ULSAH
Joined 'example.ulsah.com' to domain 'ULSAH.COM'
```

Дополнительные параметры конфигурации приведены на справочной странице `krb5.conf`.

Демон `sssd`: служба системной безопасности



В прошлом путь к установке системы SSO в операционных системах UNIX и Linux был долог и труден. Несколько лет назад было принято устанавливать независимую аутентификацию для каждой службы или приложения. Такой подход часто приводил к появлению запутанных конфигураций и недокументированных зависимостей, которые со временем невозможно было контролировать. Пароли пользователей будут работать с одним приложением, но не с другим, что вызовет разочарование для всех.

Ранее компания Microsoft опубликовала расширения (первоначально называемые "Services for UNIX", затем "Windows Security and Directory Services for UNIX" и, наконец, "Identity Management for UNIX" в Windows Server 2012), которые облегчили размещение пользователей и групп UNIX в службе Active Directory. Однако использование полномочий для управления этими атрибутами в системе, отличной от UNIX, было неестественным. К облегчению многих пользователей компания Microsoft прекратила поддержку этой функции в версии Windows Server 2016.

Эти проблемы нуждались в каком-то комплексном решении, и именно это мы получили с помощью демона **`sssd`**, Demon System Security Services. Доступный как для Linux, так и для FreeBSD демон **`sssd`** — это универсальный инструмент для проверки прав пользователей, аутентификации и сопоставления учетных записей. Он также может кешировать учетные данные в автономном режиме, что полезно для мобильных устройств.

Демон **`sssd`** поддерживает аутентификацию как с помощью собственного протокола LDAP, так и протокола Kerberos. Демон **`sssd`** настраивается с помощью файла **`sssd.conf`**. Ниже приведен базовый пример для среды, которая использует Active Directory в качестве службы каталогов:

```
[sssd]
services = nss, pam
domains = ULSAH.COM

[domain/ULSAH.COM]
id_provider = ad
access_provider = ad
```

Если вы используете сервер LDAP, не задействующий службу Active Directory, то ваш файл **`sssd.conf`** может выглядеть примерно так:

```
[sssd]
services = nss, pam
domains = LDAP
```

```
[domain/LDAP]
id_provider = ldap
auth_provider = ldap
ldap_uri = ldap://ldap.ulsah.com
ldap_user_search_base = dc=ulsah,dc=com
tls_reqcert = demand
ldap_tls_cacert = /etc/pki/tls/certs/ca-bundle.crt
```

По очевидным причинам безопасности демон **sssd** не разрешает аутентификацию по незашифрованному каналу, поэтому требуется использование LDAPS/TLS. Установка атрибута `tls_reqcert` для запроса в приведенном выше примере заставляет демон **sssd** дополнительно проверять сертификат сервера. Демон **sssd** отключает соединение, если обнаруживается, что сертификат недостаточен.

Когда демон **sssd** запущен и работает, вы должны сообщить системе, чтобы она использовала его в качестве источника для идентификации и аутентификации. Следующим шагом в этом процессе является настройка ключа службы имен и настройка PAM.

`nsswitch.conf`: переключатель службы имен

Коммутатор службы имен (name service switch — NSS) был разработан для упрощения выбора между различными базами данных конфигурации и механизмами разрешения имен. Вся конфигурация находится в файле **`/etc/nsswitch.conf`**.

Синтаксис прост: для определенного типа поиска достаточно просто перечислить источники в том порядке, в котором им следует обратиться. Вначале следует запрашивать локальные системные файлы **passwd** и **group** (заданные атрибутом `files`), а затем можно привязаться к службе Active Directory или другой службе каталогов с помощью демона **sssd** (указанного атрибутом **sss**). Этот трюк описан в следующих строках.

```
passwd: files sss
group:  files sss
shadow: files sss
```

После настройки файла **`nsswitch.conf`**, конфигурацию можно проверить с помощью команды **`getent passwd`**. Эта команда выводит список учетных записей пользователей, определенных во всех источниках в формате **`/etc/passwd`**:

```
$ getent passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
...
bwhaley:x:10006:10018::/home/bwhaley:/bin/sh
guest:*:10001:10001:Guest:/home/ULSAH/guest:/bin/bash
ben:*:10002:10000:Ben Whaley:/home/ULSAH/ben:/bin/bash
krbtgt:*:10003:10000:krbtgt:/home/ULSAH/krbtgt:/bin/bash
```

Как видно из последних трех записей, приведенных выше, единственный способ отличить локальных пользователей от учетных записей домена — это идентификатор пользователя и путь к домашнему каталогу.

Модули PAM: украшение или чудо аутентификации?

Аббревиатура PAM означает "подключаемые модули аутентификации" (Pluggable Authentication Modules — PAM). Эти модули освобождают программистов от необходимости выполнять рутинные операции для реализации систем аутентификации.

Концепция и термин были придуманы в компании Sun Microsystems (которая в настоящее время является частью компании Oracle) и описаны в 1996 году в статье, авторами которой были Самар (Samar) и Лаи (Lai) из компании SunSoft.

В далеком прошлом команды вроде **login** включали встроенный код аутентификации, который предлагал пользователю ввести пароль, сравнивал его с зашифрованной записью из файла **/etc/shadow** (в настоящее время этот файл называется **/etc/passwd**) и делал вывод об их совпадении или несовпадении. Разумеется, другие команды (например, **passwd**) содержали такой же код. Не имея доступа к открытому коду, было невозможно изменить метод аутентификации, поэтому администраторы имели мало возможностей для управления настройками паролей (например, задавать правила, описывающие правильные пароли) или не имели их вообще. Появление модулей PAM в корне изменило положение дел.

Системные процедуры аутентификации из модулей PAM были помещены в совместно используемую библиотеку, которую может вызывать программа **login** и другие программы. Выделение функций аутентификации в отдельную подсистему облегчает внедрение новых методов аутентификации и шифрования в компьютерную систему. Например, многофакторная аутентификация может поддерживаться без внесения изменений в программы **login** и **passwd**.

Для системного администратора выбор правильного уровня безопасности для аутентификации стал простой задачей конфигурирования. Программисты также получили выгоду: они больше не обязаны писать сложный код для аутентификации и, что еще более важно, их системы аутентификации правильно реализуются с первой попытки. Модули PAM могут аутентифицировать все виды деятельности: регистрацию пользователей, другие формы доступа к системе, использование защищенных веб-сайтов и даже настройку приложений.

Конфигурация модулей PAM

Файлы конфигурации модулей PAM содержат по одной строке, каждая из которых представляет собой имя модуля PAM, используемого в системе.

Общий формат этих файлов имеет следующий вид.

тип_модуля управляющий_флаг путь_к_модулю [*аргументы*]

Поля разделяются пробелами.

Порядок полей в файле конфигурации PAM имеет значение. Например, модуль, предлагающий пользователю ввести пароль, должен выполняться раньше модуля, проверяющего корректность паролей. Один модуль передает результаты своей работы другому, устанавливая значения переменных окружения или переменных PAM.

Поле *тип_модуля* может иметь значения `auth`, `account`, `session` или `password`. Модуль типа `auth` идентифицирует пользователя и может предоставлять ему право группового доступа. Модуль типа `account` выполняет действия, не связанные с аутентификацией, например предоставляет доступ в зависимости от времени суток, ограничивает количество одновременно работающих пользователей или количество портов, на которых может происходить регистрация. (Например, можно использовать модуль типа `account` для ограничения регистрации суперпользователя на консоли.) Действия, которые необходимо выполнить до или после того, как пользователь получит доступ к требуемой службе, например монтирование файловой системы, реализуются модулем типа `session`. Наконец, модуль типа `password` применяется, когда пользователь должен ввести аутентификационную информацию (например, пароль или кодовое словосочетание).

Поле *управляющий_флаг* определяет, как должны взаимодействовать модули, образующие стек (табл. 17.2).

Таблица 17.2. Управляющие флаги модулей PAM

Флаг	Остановка в случае ошибки	Остановка в случае успеха	Комментарии
include	–	–	Включает другой файл конфигурации в данную точку стека
optional	Нет	Нет	Имеет значение, только если модуль единственный
required	Нет	Нет	Ошибка приводит со временем к отказу всего стека
requisite	Да	Нет	Так же как и required, но отказ стека наступает мгновенно
sufficient	Нет	Да	Неудачное название; см. пояснения в тексте

Если бы модуль PAM мог просто возвращать код ошибки, как только первый модуль в стеке потерпит неудачу, то система этих флагов была бы проще. К сожалению, система разработана так, что большинство модулей имеет шанс на выполнение независимо от успеха или сбоя других модулей, находящихся с ними на одном и том же уровне, и этот факт имеет определенные тонкости, связанные с потоком управления. (Это было сделано для того, чтобы хакеры не могли понять, какие модули PAM в стеке вызывают отказ.)

Модули required должны следовать один за другим; отказ одного из них гарантирует, что весь стек со временем даст сбой. Однако отказ модуля, помеченного флагом required, не приводит к немедленному сбою стека. Если вы хотите, чтобы сбой наступал моментально, вместо флага required используйте флаг requisite.

Успех модуля типа sufficient немедленно прекращает работу стека. Однако окончательный результат работы стека не обязательно должен быть успешным, поскольку модуль типа sufficient не может компенсировать ошибку выполненного ранее модуля типа required. Если ранее выполненный модуль типа required выдал ошибку, успешно выполненный модуль типа sufficient прекращает работу стека *и* возвращает ошибку в качестве итогового результата.

Прежде чем изменять параметры безопасности системы, еще раз убедитесь, что вы понимаете, что делаете. (Вы можете настраивать модули PAM хоть каждый день, но как долго вы способны помнить, какая версия указана в атрибуте requisite, а какая — в атрибуте required?)

Пример модуля PAM

В качестве примера ниже приведен файл **/etc/pam.d/login** из системы Linux с запущенным демоном **sssd**. Мы раскрыли включенные файлы, чтобы сделать пример более связным.

```
auth      requisite     pam_nologin.so
auth      [user_unknown=ignore success=ok ignore=ignore auth_err=die
             default=bad] pam_securetty.so
auth      required pam_env.so
auth      sufficient pam_unix2.so
auth      sufficient pam_sss.so use_first_pass

account   required pam_unix2.so
account   [default=bad success=ok user_unknown=ignore] pam_sss.so
password  requisite  pam_pwcheck.so nullok cracklib
```

```
password required   pam_unix2.so use_authtok nullok
password sufficient pam_sss.so use_authtok

session  required   pam_loginuid.so
session  required   pam_limits.so
session  required   pam_unix2.so
session  sufficient pam_sss.so
session  optional   pam_umask.so
session  required   pam_lastlog.so nowtmp
session  optional   pam_mail.so standard
session  optional   pam_ck_connector.so
```

Стек `auth` содержит несколько модулей. В первой строке модуль `pam_nologin` проверяет существование файла **`/etc/nologin`**. Если он существует, модуль немедленно прекращает регистрацию, если пользователь не является `root`. Модуль `pam_securetty` гарантирует, что суперпользователь может регистрироваться только с терминалов, перечисленных в файле **`/etc/securetty`**. В этой строке используется альтернативный синтаксис системы Linux, описанный на справочной странице **`pam.conf`**. В этом случае функционирование модуля аналогично реакции на флаг `required`. Модуль `pam_env` устанавливает переменные окружения в файле **`/etc/security/pam_env.conf`**, и наконец, модуль `pam_unix2` проверяет мандаты пользователя, выполняя стандартную аутентификацию системы UNIX. Если пользователь не имеет учетной записи UNIX, модуль `pam_sss` пытается выполнить аутентификацию с помощью демона **`sssd`**. Если оба модуля вернут ошибку, стек `auth` вернет сообщение об ошибке.

Стек `account` содержит только модули `pam_unix2` и `pam_sss`, которые в данном контексте оценивают корректность самой учетной записи. Например, они возвращают ошибку, если учетная запись просрочена или пароль должен быть изменен. В последнем случае соответствующий модуль принимает от пользователя новый пароль и передает его модулям `password`.

Строка `pam_pwcheck` проверяет устойчивость предложенных паролей, вызывая библиотеку **`cracklib`**. Она возвращает ошибку, если новый пароль не соответствует установленным требованиям. Однако она допускает пустые пароли, потому что установлен флаг `nullok`. Строки `pam_unix2` и `pam_sss` обновляют действующий пароль.

В заключение модули `session` выполняют несколько рутинных процедур. Модуль `pam_loginuid` устанавливает атрибут процесса ядра `loginuid` равным идентификатору пользователя, модуль `pam_limits` считывает лимиты использования ресурсов из файла **`/etc/security/limits.conf`** и устанавливает соответствующие параметры процесса, чтобы ввести их в действие. Модуль `pam_unix2` регистрирует доступ пользователя в систему, а модуль `pam_unmask` устанавливает режим создания начального файла. Модуль `pam_lastlog` выводит время последней регистрации пользователя, а модуль `pam_mail` — сообщение, если пользователь получил новое электронное сообщение. В заключение, модуль `pam_ck_connector` уведомляет демон **`ConsoleKit`** (системный демон, управляющий сеансами регистрации) о новой регистрации.

В результате пользователь был успешно аутентифицирован, и модули PAM вернули управление программе **`login`**.

17.4. Альтернативные подходы

Хотя система LDAP в настоящее время является самым популярным методом централизации информации о пользователях и аутентификации внутри организации, за мно-

гие десятилетия появилось много других подходов. Два старых варианта, служба NIS и утилита **rsync**, все еще используются в отдельных изолированных пакетах.

NIS: сетевая информационная служба

Административная база данных NIS (Network Information Service — сетевая информационная служба), выпущенная компанией Sun в 80-х годах, была первой базой данных такого рода. Сначала она называлась Sun Yellow Pages (желтые страницы Sun), но по причинам правового характера ее пришлось переименовать. Команды NIS до сих пор начинаются с префикса **yp**, поскольку имя, данное при рождении, забыть трудно. Служба NIS была с воодушевлением принята поставщиками UNIX-систем и поддерживается во всех дистрибутивах Linux и FreeBSD.

И все же в наши дни для новых систем не следует использовать NIS, причем не только из-за неизбежной интеграции с системами Windows, но и из-за несовершенства NIS-системы безопасности и масштабируемости, в то время как есть альтернатива в виде LDAP.

Утилита rsync: более безопасная рассылка файлов

Утилита **rsync**, написанная Эндрю Триджеллом (Andrew Tridgell) и Полом Маккерасом (Paul Mackerras), напоминает улучшенную версию команды **scp**, пытающуюся сохранить ссылки, время модификации и права доступа. Утилита **rsync** более эффективна для работы в сети, поскольку анализирует отдельные файлы и пытается передавать только изменения между версиями.

Проще всего распространять файлы, такие как **/etc/passwd** и **/etc/group**, настроив планировщик **cron** на запуск утилиты **rsync** на главном сервере. Хотя эту схему несложно реализовать и использовать, она требует, чтобы все изменения происходили непосредственно на главном сервер, в том числе изменения паролей пользователей.

Например, команда

```
# rsync -gopt -e ssh /etc/passwd /etc/shadow lollipop:/etc
```

посылает файлы **/etc/passwd** и **/etc/shadow** на компьютер **lollipop**. Флаг **-gopt** указывает на сохранение прав доступа, идентификаторов владельцев и времени модификации файла. Для передачи файлов утилита **rsync** использует оболочку **ssh**, поэтому соединение является зашифрованным. Однако демон **sshd** на компьютере `lollipop` должен быть настроен так, чтобы не требовать пароля для запуска команд. Разумеется, это требование нарушает правила безопасности. Будьте осторожны!

Флаги **--include** и **--exclude** позволяют указать список регулярных выражений, с которым будут сравниваться имена файлов. Благодаря этому можно сформировать довольно сложный набор правил копирования. Если командная строка становится слишком громоздкой, поместите регулярные выражения в отдельные файлы. Для этого существуют опции **--include-file** и **--exclude-file**.

Инструменты управления конфигурацией, такие как Ansible, являются еще одним популярным способом распространения файлов между системами (см. главу 23).

17.5. Литература

В качестве общего введения в систему LDAP мы можем порекомендовать неплохой "хрестоматийный учебник" *LDAP for Rocket Scientists*, который содержит описание архитектуры и протокола LDAP. Эта книга опубликована в электронном виде на сайте `zytrax.com/books/ldap`. Кроме того, существует множество разнообразных докумен-

тов RFC, посвященных протоколу LDAP, каждый из которых является довольно сложным. Наиболее важные документы перечислены в табл. 17.3.

Таблица 17.3. Документы RFC, посвященные протоколу LDAP

RFC	Название
2307	An Approach for Using LDAP as a Network Information Service
2820	Access Control Requirements for LDAP
2849	LDAP Data Interchange Format (LDIF) — Technical Specification
3112	LDAP Authentication Password Schema
3672	Subentries in the Lightweight Directory Access Protocol (LDAP)
4511	LDAP: The Protocol
4512	LDAP: Directory Information Models
4513	LDAP: Authentication Methods and Security Mechanisms
4514	LDAP: String Representation of Distinguished Names
4515	LDAP: String Representation of Search Filters
4516	LDAP: Uniform Resource Locator
4517	LDAP: Syntaxes and Matching Rules
4519	LDAP: Schema for User Applications

Кроме того, существует несколько старых, но полезных книг по LDAP.

- Carter, Gerald. *LDAP System Administration.* Sebastopol, CA: O'Reilly Media, 2003.
- Volgmaier, Reinhard. *The ABCs of LDAP: How to Install, Run, and Administer LDAP Services.* Boca Raton, FL: Auerbach Publications, 2004. Очень хорошая книга о модулях PAM.
- Lucas, Michael. *PAM Mastery.* North Charleston, SC: CreateSpace, 2016.

В заключение укажем на прекрасную книгу издательства O'Reilly о службе Active Directory:

- Desmond, Brian, Joe Richards, Robbie Allen, and Alistair G. Lowe-Norris. *Active Directory: Designing, Deploying, and Running Active Directory.* Sebastopol, CA: O'Reilly Media, 2013.

Глава 18

Электронная почта



В далеком прошлом, чтобы приготовить блюдо из курицы, необходимо было не просто обжарить курицу, а пойти в курятник, выбрать цыпленка, свернуть ему шею, выщипать перья и т.д. Сегодня большинство из нас просто покупают полуфабрикат из курицы в продуктовом магазине и тем самым избегают лишних хлопот.

Электронная почта эволюционировала аналогичным образом. С давних времен организации вручную работали с электронной почтой, иногда вплоть до предопределения точной маршрутизации почты. Сегодня многие организации используют пакетные облачные службы электронной почты, такие как Google Gmail или Microsoft Office 365.

Даже если ваша электронная почта работает в облаке, вам все равно придется разбираться, поддерживать и взаимодействовать с ней в качестве администратора. Если ваша организация использует локальные почтовые серверы, рабочая нагрузка еще больше расширяется, включая в себя настройку, мониторинг и тестирование.

Если вы узнаете себя в одном из этих практичных сценариев, то эта глава для вас. В противном случае пропустите этот материал и потратьте свое время, посвященное администрированию электронной почты, отвечая на сообщения от состоятельных иностранцев, которым нужна помощь в перемещении миллионов долларов в обмен на большую награду.[1]

18.1. Архитектура почтовой системы

Система электронной почты состоит из нескольких компонентов.

- *Пользовательский агент* (Mail User Agent— MUA, или UA), который дает пользователям возможность читать и составлять сообщения.

[1] Шутка!

- *Агент передачи электронной почты* (Mail Submission Agent — MSA), принимающий почту, исходящую от агента MUA, обрабатывающий ее и передающий транспортной системе.
- *Транспортный агент* (Mail Transport Agent — MTA), который пересылает сообщения с одного компьютера на другой.
- *Агент доставки* (Aelivery Agent — DA), который помещает сообщения в локальное хранилище[2].
- Необязательный *агент доступа* (Acess Agent — AA), который связывает пользовательский агент с хранилищем сообщений (например, посредством протокола IMAP или POP).

К некоторым из этих агентов присоединяются средства для распознавания спама, вирусов и (исходящих) внутренних секретов компании. Схема взаимодействия всех указанных компонентов представлена на рис. 18.1.

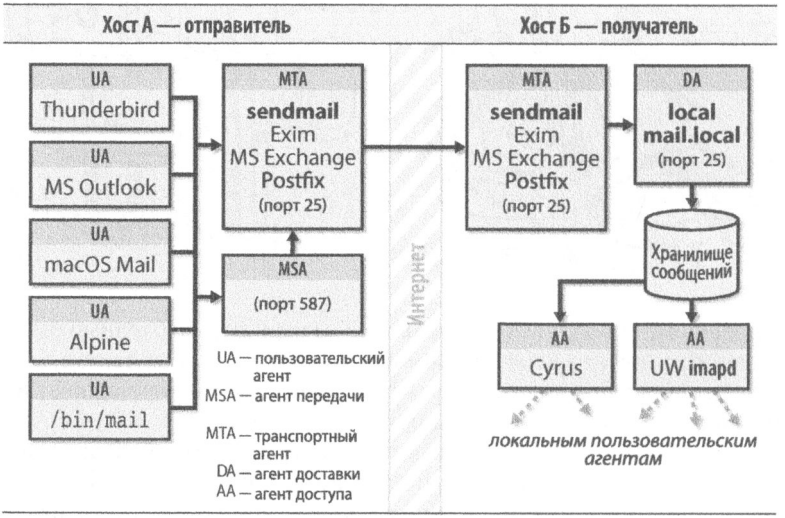


Рис. 18.1. Компоненты почтовой системы

Пользовательские агенты

Пользовательский агент (иногда говорят "клиент электронной почты") применяется для чтения и составления электронных сообщений. Первоначально сообщения могли содержать только простой текст, однако благодаря стандарту MIME (Multipurpose Internet Mail Extensions — многоцелевые расширения электронной почты в сети Интернет) появилась возможность включать в них форматированный текст и присоединять разные файлы (в том числе вирусы). Стандарт MIME поддерживается большинством пользовательских агентов. Поскольку он не влияет на процесс адресации и доставки почты, мы не будем его рассматривать.

Самым первым пользовательским агентом была утилита `/bin/mail`, которая и сейчас остается удобным инструментом для чтения текстовых сообщений. Поскольку электронная почта в Интернете давно вышла за пределы "текстовой эры", пользовательские агенты, ориентированные на текст, для большинства пользователей являются непрак-

[2]Иногда это почтовые ящики пользователей, иногда — база данных.

тичными. Но не следует отбрасывать утилиту `/bin/mail`; она остается удобным интерфейсом для сценариев и других программ.

Одна из элегантных функциональных возможностей, проиллюстрированных на рис. 18.1, состоит в том, что пользовательский агент не обязан выполняться в той же самой системе и даже на той же самой платформе, на которой выполняется остальная часть почтовой системы. Пользователи могут получить свои сообщения электронной почты на ноутбуках или смартфонах, работающих под управлением системы Windows, с помощью протоколов агентов доступа IMAP или POP.

Агенты передачи

Агенты MSA — последнее новшество в пантеоне электронной почты — были изобретены для того, чтобы разгрузить агентов MTA от некоторых задач. Агенты MSA облегчают серверам-концентраторам задачу различения входящих и исходящих сообщений (например, для принятия решения об ответе) и обеспечивают пользовательским агентам единообразную и простую конфигурацию исходящей почты.

Агент MSA напоминает секретаря, который ведет прием новых сообщений и внедрен в систему локальными пользовательскими агентами. Агент MSA располагается между пользовательским и транспортным агентами и выполняет несколько функций, которые ранее относились к компетенции агента MTA. Агент MSA реализует безопасную (зашифрованную и аутентифицированную) связь с пользовательскими агентами и часто немного изменяет заголовки и удаляет входящие сообщения. Во многих ситуациях агент MSA просто прослушивает агентов MTA на разных портах, применяя разные конфигурации.

Агенты MSA используют тот же протокол передачи почты, что и агенты MTA, поэтому с точки зрения пользовательских агентов они выглядят как агенты MTA. Однако они обычно прослушивают соединение на порту 587, а не 25, который является стандартным для агентов MTA. При такой схеме работы пользовательские агенты должны соединяться с портом 587, а не 25. Если ваш пользовательский агент не может использовать порт 587, вы можете запустить агент MSA на порту 25, но в системе, которая отличается от системы, в которой был запущен агент MTA; в каждый момент времени конкретный порт может прослушивать только один процесс.

Если вы используете агент MSA, проверьте, что конфигурация вашего транспортного агента задана так, что он не дублирует работу, выполняемую агентом MSA. Обработка дубликатов никак не влияет на корректность обработки почты, но заставляет систему выполнять дополнительную ненужную работу.

Поскольку агент MSA для передачи сообщений использует агент MTA, для взаимной аутентификации оба агента должны использовать протокол SMTP-AUTH. В противном случае возникнет так называемый "открытый ретранслятор" (open relay), который могут использовать спамеры, в результате чего другие почтовые серверы занесут ваш адрес в "черный список".

🕮 Дополнительную информацию о протоколе SMTP см. в разделе 18.3.

Транспортные агенты

Задача транспортного агента — принимать почту от пользовательского агента или агента передачи, интерпретировать адреса получателей и перенаправлять почту на соответствующие почтовые серверы для последующей доставки. Транспортные агенты работают по протоколу SMTP (Simple Mail Transport Protocol — простой протокол передачи

электронной почты), который вначале был определен в документе RFC821, а затем дополнен в документе RFC5321. Расширенная версия этого протокола называется ESMTP (Extended SMTP).

Список заданий транспортных агентов как отправителя, так и получателя содержит следующие пункты.

- Получение сообщений электронной почты от удаленных почтовых серверов.
- Распознавание адресов получателей.
- Перезапись адресов получателей в форме, понятной для агента доставки.
- Перенаправление сообщения следующему ответственному серверу или передача его локальному агенту доставки для сохранения в почтовом ящике пользователя.

Большая часть работы, связанной с настройкой почтовой системы, сводится к конфигурированию транспортного агента. В книге мы рассматриваем три агента MTA с открытым кодом: **sendmail**, Exim и Postfix.

Локальные агенты доставки

Агент доставки, которого иногда называют также *локальным агентом доставки* (Local Delivery Agent — LDA), отвечает за прием почты от транспортного агента и ее передачу соответствующим получателям на локальном компьютере. Почта может доставляться конкретному пользователю, в список рассылки, в файл и даже в программу. Однако два последних получателя могут ослабить конфиденциальность и безопасность вашей системы.

Транспортные агенты обычно содержат встроенных локальных агентов доставки. Например, программы **procmail** (`procmail.org`) и Maildrop (`courier-mta.org/maildrop`) являются локальными агентами доставки, которые способны фильтровать и сортировать почту перед ее доставкой. Некоторые агенты доступа (AA) также имеют встроенных агентов доставки, выполняющих как доставку, так и другие задания.

Хранилища сообщений

Хранилище сообщений — пункт назначения, находящийся в конце долгого пути, который сообщение электронной почты проходит по Интернету от отправителя к получателю.

Почта обычно хранится в формате **mbox** или **Maildir**. В первом случае вся почта хранится в одном файле, как правило, **/var/mail/***имя_пользователя*, а индивидуальные сообщения отделяются специальной строкой **From**. Во втором варианте каждое сообщение хранится в отдельном файле. Хранить сообщения в отдельных файлах намного удобнее, но при этом в каталоге находится очень много маленьких файлов; некоторые файловые системы этого не одобряют.

Эти простые файлы до сих пор используются в качестве хранилищ сообщений, но интернет-провайдеры, имеющие тысячи и миллионы клиентов электронной почты, ищут другие технологии для реализации своих хранилищ, как правило, на основе баз данных. К сожалению, это приводит к тому, что хранилища сообщений становятся все более сложными.

Агенты доступа

Для доступа к хранилищам и загрузки сообщений электронной почты на локальное устройство (рабочую станцию, ноутбук, телефон и т.п.) используются два протокола: Internet Message Access Protocol версии 4 (IMAP4) и Post Office Ptorocol версии 3 (POP3).

Ранние версии этих протоколов имели проблемы с безопасностью. Убедитесь, что вы используете версию (IMAPS или POP3S), которая реализует шифрование SSL и не передает через Интернет пароли в открытом виде.

Протокол IMAP значительно лучше, чем протокол POP, поскольку доставляет ваши почтовые сообщения по одному, а не все сразу, что более удобно для работы в сети (особенно если каналы связи не обладают высокой пропускной способностью) и комфортнее для людей, перемещающихся с места на место. Протокол IMAP также лучше обрабатывает огромные файлы, которые многие любят присоединять к своим сообщениям: вы можете просматривать заголовки своих сообщений и не загружать вложенные файлы, пока не будете готовы к работе с ними.

18.2. Структура почтового сообщения

Структура электронного сообщения состоит из трех частей.

- Конверт
- Заголовки
- Тело

Конверт определяет, куда должно быть доставлено сообщение или куда его требуется возвратить в случае, если доставка невозможна. Обычно конверт невидим для пользователя и не является частью самого сообщения; он используется транспортным агентом.

Если отправителем и получателем являются обычные люди, то адреса конверта обычно согласованы со строками `From` и `To` заголовка. Если же сообщение направлено списку рассылки или было сгенерировано спамером, пытающимся подделать идентичность отправителя, то конверт и заголовки могут быть не согласованы друг с другом.

Заголовки — это набор пар "свойство-значение", отформатированных в соответствии с документом RFC5322. В них содержится различная информация о сообщении, включая дату и время его отправки, а также сведения о транспортных агентах, через которые оно прошло на своем пути. Заголовки являются неотъемлемой частью сообщения, но пользовательские агенты часто скрывают некоторые менее интересные заголовки при отображении сообщения.

Тело сообщения — это та информация, которую, собственно, и требуется переслать. Оно должно содержать обычный текст, однако часто он представляет собой двоичные данные в специальной почтовой кодировке.

Системный администратор должен уметь анализировать заголовки почтовых сообщений и выявлять с их помощью возникшие проблемы. Многие пользовательские агенты скрывают заголовки, но существует способ их увидеть, открыв хранилище сообщений в текстовом редакторе. Ниже приведены заголовки (с сокращениями, обозначенными многоточиями) из обычного сообщения, которое не является спамом. Мы удалили половину страницы заголовков, которые система Gmail использует для фильтрации спама.

```
Delivered-To: sailingevi@gmail.com
Received: by 10.231.39.205 with SMTP id...; Fri, 24 May 2013 08:14:27
   -700 (PDT)[3]
Received: by 10.114.163.26 with SMTP id...; Fri, 24 May 2013 08:14:26
   -700 (PDT)
Return-Path: <david@schweikert.ch>
```

[3] В память об Эви, которая была автором этого раздела, данный исторический пример приводится без изменений.

```
Received: from mail-relay.atrust.com
   (mail-relay.atrust.com [63.173.189.2]) by mx.google.com with
   ESMTP id 17si2166978pxi.34.2009.10.16.08.14.20; Fri, 24 May 2013
   08:14:25 -0700 (PDT)
Received-SPF: fail (google.com: domain of david@schweikert.ch does not
   designate 63.173.189.2 as permitted sender) client-ip=63.173.189.2;
   Authentication-Results: mx.google.com; spf=hardfail (google.com: domain
   of david@schweikert.ch does not designate 63.173.189.2 as permitted
   sender) smtp.mail=david@schweikert.ch
Received: from mail.schweikert.ch (nigel.schweikert.ch [88.198.52.145])
   by mail-relay.atrust.com (8.12.11/8.12.11) with ESMTP id n9GFEDKA0
   for <evi@atrust.com>; Fri, 24 May 2013 09:14:14 -0600
Received: from localhost (localhost.localdomain [127.0.0.1]) by mail.
   schweikert.ch (Postfix) with ESMTP id 3251112DA79; Fri, 24 May 2013
   17:14:12 +0200 (CEST)
X-Virus-Scanned: Debian amavisd-new at mail.schweikert.ch
Received: from mail.schweikert.ch ([127.0.0.1]) by localhost (mail.
   schweikert.ch [127.0.0.1]) (amavisd-new, port 10024) with ESMTP id
   dV8BpT7rhJKC; Fri, 24 May 2013 17:14:07 +0200 (CEST)
Received: by mail.schweikert.ch (Postfix, from userid 1000)
   id 2A15612DB89; Fri, 24 May 2013 17:14:07 +0200 (CEST)
   Date: Fri, 24 May 2013 17:14:06 +0200
From: David Schweikert <david@schweikert.ch>
To: evi@atrust.com
Cc: Garth Snyder <garth@garthsnyder.com>
Subject: Email chapter comments
Hi evi
I just finished reading the email chapter draft, and I was pleased to see
...
```

Для того чтобы прочитать эту тарабарщину, начнем со строк Received, но в направлении снизу вверх (т.е. со стороны отправителя). Это сообщение пришло с домашнего компьютера Дэвида Швайкерта (David Schweikert), находящегося в домене `schweikert.ch`, на его почтовый сервер (`mail.schweikert.ch`), где оно прошло сканирование на вирусы. Затем оно было переслано получателю по адресу `evi@atrust.com`. Однако получатель `mail-relay.atrust.com` послал его по адресу `sailingevi@gmail.com`, где находился почтовый ящик Эви.

🕮 Более подробная информация о технологии SPF приведена в разделе 18.4.

Посреди заголовков мы видим, что проверка сообщения на основе технологии SPF завершилась неудачно. Это произошло потому, что система Google проверила IP-адрес сервера `mail-relay.atrust.com`, сравнила его с записью SPF в домене `schweirkert.ch` и они, разумеется, не совпали. Это недостаток технологии SPF — она не распознает сообщения, которые предназначены для пересылки.

Мы можем видеть следы частого использования транспортных агентов (Postfix — в домене `schweirkert.ch` и **sendmail** 8.12 — в домене `atrust.com`). В данном случае сканирование вирусов было выполнено с помощью агента **amavisd-new** на порту 10024 на компьютере, работающем под управлением операционной системы Debian Linux. Мы можем проследить путь, который проделало сообщение из часового пояса центрально-европейского летнего времени (`CEST +0200`) в Колорадо (`-0600`) и на сервер Google (`PDT -0700`); эти числа представляют собой разницу между местным временем и всемирным координированным временем (Coordinated Universal Time — UTC). В заголовках можно найти много скрытой информации!

Рассмотрим заголовки (тоже сокращенные) сообщения, которое является спамом.

```
Delivered-To: sailingevi@gmail.com
Received: by 10.231.39.205 with SMTP id…; Fri, 19 Oct 2009 08:59:32
   -0700…
Received: by 10.231.5.143 with SMTP id…; Fri, 19 Oct 2009 08:59:31
   -0700…
Return-Path: <smotheringl39@sherman.dp.ua>
Received: from mail-relay.atrust.com (mail-relay.atrust.com
   [63.173.189.2]) …
Received-SPF: neutral (google.com: 63.173.189.2 is neither
   permitted nor denied by best guess record for domain of
   smotheringl39@sherman.dp.ua) client-ip=63.173.189.2;
Authentication-Results: mx.google.com; spf=neutral (google.
   com: 63.173.189.2 is neither permitted nor denied by best
   guess record for domain of smotheringl39@sherman.dp.ua)
   smtp.mail=smotheringl39@sherman.dp.ua
Received: from SpeedTouch.lan (187-10-167-249.dsl.telesp.net.br
   [187.10.167.249] (may be forged)) by mail-relay.atrust.com …
Received: from 187.10.167.249 by relay2.trifle.net; Fri, 19 Oct 2009
   13:59: …
From: "alert@atrust.com" <alert@atrust.com>
To: <ned@atrust.com>
Subject: A new settings file for the ned@atrust.com mailbox
Date: Fri, 19 Oct 2009 13:59:12 -0300 …
```

В соответствии с заголовком `From`, отправителем этого сообщения является почтовый ящик `alert@atrust.com`. Однако заголовок `Return-Path`, содержащий копию конверта отправителя, свидетельствует о том, что источником сообщения является почтовый ящик `smotheringl39@sherman.dp.ua`, адрес которого указывает на Украину. Первый транспортный агент, обработавший сообщение, имеет IP-адрес 187.10.167.249 и находится в Бразилии. Хитрые спамеры...[4]

Проверка по технологии SPF, которую выполнил сервер Google, снова завершилась провалом, но на этот раз с "нейтральным" результатом, потому что домен `sherman.dp.ua` не имеет записи SPF, с которой можно было бы сравнить IP-адрес сервера `mail-relay.atrust.com`.

Информация о получателе также не совсем правдива. Заголовок `To` указывает, что сообщение направлено по адресу `ned@atrust.com`. Однако для того, чтобы сообщение было направлено по адресу `sailingevi@gmail.com`, среди адресов получателя на конверте должен находиться адрес `evi@atrust.com`.

18.3. Протокол SMTP

Протокол SMTP (Simple Mail Transport Protocol — простой протокол передачи электронной почты) и его расширенная версия ESMTP были стандартизованы в документах RFC (в частности, RFC5321) и используются для передачи сообщений между разными частями почтовой системы.

- От пользовательского агента к агенту передачи или транспортному агенту, когда сообщение поступает в почтовую систему.
- От агента передачи к транспортному агенту, когда сообщение начинает свой путь.

[4]Следует отметить, что многие строки заголовка, включая строки Received, могут оказаться поддельными. С этими данными необходимо быть крайне осторожными.

- От транспортного агента или агента передачи к программам сканирования спама и вирусов.
- От одного транспортного агента к другому при передаче сообщений от одной организации другой.
- От транспортного агента к агенту доставки, когда сообщение поступает в локальное хранилище.

Поскольку формат сообщений и протокол передачи стандартизированы, транспортные агенты отправителя и получателя не обязательно должны быть одинаковыми и даже знать друг о друге; просто они оба должны придерживаться протоколов SMTP и ESMTP. На разных почтовых серверах могут работать различные транспортные агенты, и их взаимодействие будет безошибочным.

В соответствии со своим именем протокол SMTP является довольно простым. Транспортный агент связывается с вашим почтовым сервером и, по существу, сообщает следующее: "Вот сообщение; пожалуйста, доставь его по адресу `user@your.domain`". Ваш транспортный агент отвечает: "Хорошо".

Требование строго придерживаться протокола SMTP стало основой для борьбы со спамом и вредоносными программами, поэтому системные администраторы должны хорошо в нем разбираться. Его язык имеет всего несколько команд; самые важные перечислены в табл. 18.1.

Таблица 18.1. Команды протокола SMTP

Команда	Функция
`HELO` *имя_компьютера*	Проверяет, поддерживает ли компьютер протокол SMTP
`EHLO` *имя_компьютера*	Проверяет, поддерживает ли компьютер протокол ESMTP
`MAIL FROM:` *обратный_адрес*	Идентифицирует конверт отправителя
`RCPT TO:` *прямой_адрес*[a]	Идентифицирует конверт получателя
`VRFY` *адрес*	Проверяет корректность *адреса* (возможность доставки)
`EXPN` *адрес*	Демонстрирует расширение альтернативных имен и отображения файла **`.forward`**
`DATA`	Отмечает начало тела сообщения[б]
`QUIT`	Завершает диалог с сервером и прерывает соединение
`RSET`	Восстанавливает состояние соединения
`HELP`	Выводит на экран описание команд SMTP

[a] У сообщения может быть несколько команд `RCPT`.

[б] Тело завершается пустой строкой, содержащей одну точку.

Вы прислали мне привет (EHLO)

Серверы, использующие протокол ESMTP, начинают общение, посылая команду `EHLO`, а не `HELO`. Если процесс на другом конце соединения понимает эту команду и отвечает "`OK`", то участники согласовывают расширения и находят набор общих параметров для обмена. Если в ответ на команду `EHLO` собеседник присылает ошибку, то сервер, использующий протокол ESMTP, переключается на протокол SMTP. Однако в настоящее время практически все серверы используют протокол ESMTP.

Типичное общение по протоколу SMTP для доставки сообщения состоит из следующих команд: `HELO` или `EHLO`, `MAIL FROM:`, `RCPT TO:`, `DATA` и `QUIT`. Большая часть

команды выполняется отправителем. Получатель лишь высылает коды ошибок и подтверждения.

В протоколах SMTP и ESMTP применяются текстовые команды, поэтому их можно использовать непосредственно в процессе отладки почтовой системы. Достаточно просто запустить утилиту `telnet`, подключиться к TCP-порту 25 или 587 и начать ввод команд SMTP. Пример приведен чуть ниже.

Коды ошибок протокола SMTP

Кроме протокола SMTP, в документах RFC определена совокупность кодов временных и постоянных ошибок. Изначально эти коды состояли из трех цифр (например, 550), каждая из которых интерпретировалась отдельно. Если первая цифра равнялась 2, значит, все прошло успешно, если 4 — произошла временная ошибка, а если 5 — постоянная ошибка.

Трехзначная система кодирования ошибок слишком жесткая, поэтому в документе RFC3463 (который позднее был обновлен в RFC3886, 4468, 4865, 4954 и 5248) была описана ее гибкая модификация. В этом документе определен расширенный формат кодирования ошибок, получивший название *уведомление о состоянии доставки* (delivery status notification — DSN). Коды DNS имеют формат X.X.X, а не XXX, как раньше, причем каждый символ X может означать многозначное число. Первое число X по-прежнему может принимать значения 2, 4 и 5. Второе число означает тему, а третье кодирует подробности. В новой системе кодирования второе число используется для того, чтобы отличать ошибки компьютера от ошибок почтового ящика. Некоторые коды DSN перечислены в табл. 18.2. Все коды перечислены в приложении А к документу RFC3463.

Таблица 18.2. Команды протокола SMTP

Временная ошибка	Постоянная ошибка	Смысл
4.2.1	5.2.1	Почтовый ящик недоступен
4.2.2	5.2.2	Почтовый ящик полон
4.2.3	5.2.3	Слишком длинное сообщение
4.4.1	5.4.1	Нет ответа от компьютера
4.4.4	5.4.4	Невозможно выполнить маршрутизацию
4.5.3	5.5.3	Слишком много получателей
4.7.1	5.7.1	Доставка не авторизована, сообщение отклонено
4.7.*	5.7.*	Нарушение правил организации

Аутентификация SMTP

Документ RFC4954 определяет расширение исходного протокола SMTP, позволяющее SMTP-клиенту идентифицировать себя и проходить аутентификацию у почтового сервера. После этого сервер может позволить клиенту использовать себя для пересылки почты. Этот протокол поддерживает несколько механизмов аутентификации. Обмен информацией состоит из следующих этапов.

1. Клиент посылает команду `EHLO`, сообщая, что он использует протокол ESMT.
2. Сервер отвечает и уведомляет о своих механизмах аутентификации.
3. Клиент посылает команду `AUTH` и называет конкретный механизм аутентификации, который он хочет использовать, включая свои данные для аутентификации.

4. Сервер принимает данные, присланные с командой AUTH, или начинает последовательность команд "вызов-ответ" для обмена информацией с клиентом.
5. Сервер либо принимает, либо отвергает попытку аутентификации.

Для того чтобы узнать, какой механизм аутентификации поддерживает сервер, можно применить утилиту **telnet** к порту 25 и выполнить команду EHLO. Например, ниже приведен усеченный вариант обмена сообщениями с почтовым сервером mail-relay.atrust.com (команды набраны полужирным шрифтом).

```
$ telnet mail-relay.atrust.com 25
Trying 192.168.2.1...
Connected to mail-relay.atrust.com.
Escape character is '^]'.
220 mail-relay.atrust.com ESMTP AT Mail Service 28.1.2/28.1.2; Mon, 12
    Sep 2016 18:05:55 -0600
ehlo booklab.atrust.com
250-mail-relay.atrust.com Hello [192.168.22.35], pleased to meet you
250-ENHANCEDSTATUSCODES
250-PIPELINING
250-8BITMIME
250-SIZE
250-DSN
250-ETRN
250-AUTH LOGIN PLAIN
250-DELIVERBY
250 HELP
```

В данном случае почтовый сервер поддерживает механизмы аутентификации LOGIN и PLAIN. Серверы **sendmail**, Exim и Postfix поддерживают аутентификацию SMTP; детали их конфигураций описаны в разделах 18.8, 18.9 и 18.10.

18.4. Спам и вредоносные программы

Спам — это жаргонное название макулатурной почты, которую также называют *непрошеной коммерческой электронной почтой* (unsolicited commercial email — UCE).

Когда-то системные администраторы каждую неделю проводили много часов, вручную редактируя списки блокировки и корректируя разрешения в домашних фильтрах. К сожалению, спамеры стали настолько хитрыми и коммерциализированными, что эти меры больше не являются эффективными.

В этом разделе мы рассмотрим основные функции по борьбе со спамом каждого транспортного агента. Тем не менее бороться со спамом в одиночку бесполезно. Лучше заплатить за облачную службу по борьбе со спамом (такую как McAfee SaaS Email Protection, Google G Suite или Barracuda) и предоставить это дело профессионалам. Они лучше разбираются в состоянии глобальной электронной почты и могут быстрее реагировать на новую информацию, чем вы.

Спам стал серьезной проблемой, потому что, несмотря на невысокий процент отвечающих, стоимость ответа в расчете на доллар высока. (Рассылка на список, состоящий из 30 миллионов адресов электронной почты, стоит около 20 долл.) Если бы это не было выгодно для спамеров, то проблема не достигла бы таких масштабов. Исследования показывают, что 95–98% всех почтовых отправлений являются спамом.

Существуют даже венчурные компании, чья миссия состоит в том, чтобы рассылать спам дешевле и эффективнее (хотя обычно они называются "маркетинг электронной

почты", а не спамеры). Если вы работаете в одной из этих компаний или покупаете их услуги, мы не знаем, как вы спите по ночам.

В любом случае, посоветуйте своим пользователям просто удалять спам, который они получают. Многие сообщения, являющиеся спамом, содержат инструкции о том, как удалить свой адрес из списка рассылки. Если последовать этим инструкциям, спамеры могут действительно удалить ваш адрес из текущего списка, но они немедленно добавят его в несколько других списков с аннотацией "доставлено реальному человеку, прочитавшему это сообщение". С этого момента ваш электронный адрес будет стоить еще дороже.

Подделки

Подделать электронное письмо очень просто; многие пользовательские агенты позволяют заполнять поле адреса отправителя чем угодно. Транспортные агенты могут использовать аутентификацию по протоколу SMTP между локальными серверами, но они не могут этого сделать в масштабе всего Интернета. Некоторые транспортные агенты добавляют предупреждающие заголовки в исходящие локальные сообщения, которые могут быть подделаны.

В электронном сообщении личность любого отправителя может быть фальсифицирована. Будьте очень осторожны, если в вашей организации электронные сообщения используются в качестве средства авторизации, например, ключей от дверей, карточек доступа и денег. Вы должны предупредить об этом администраторов и полагаться на то, что они просматривают подозрительные письма, поступающие от авторитетных отправителей, и проверяют корректность сообщений. Следует быть еще более осторожным, если в сообщении предлагается предоставить необычной персоне непропорциональные привилегии.

Технология SPF и спецификации Sender ID

Лучший способ борьбы со спамом — остановить работу его источника. Это звучит легко и просто, но в реальной жизни это практически невозможно сделать. Структура Интернета затрудняет отслеживание реального источника сообщения и проверку его аутентичности. Обществу необходим надежный способ верификации отправителя и его намерений. Для решения этой проблемы было сделано много предложений, но технология SPF и спецификации Sender ID оказались наиболее удачными.

Технология SPF (Sender Policy Framework — инфраструктура политики отправителей) описана организацией IETF в документе RFC7208. Протокол SPF определяет набор DNS-записей, с помощью которых организация может указать свои официальные почтовые серверы для исходящей корреспонденции. Транспортные агенты могут отвергать сообщения, исходящие из домена организации, если они не отправлены из указанных источников. Разумеется, эта система работает хорошо только в том случае, когда большинство организаций публикует записи SPF.

Спецификации Sender ID и технология SPF практически идентичны по форме и функциям. Однако ключевые части технологии Sender ID запатентованы компанией Microsoft, следовательно, являются предметом полемики. Во время написания книги (2017 г.) компания Microsoft все еще пыталась склонить индустрию к принятию своих запатентованных стандартов. Организация IETF решила не делать однозначный выбор и опубликовала документы RFC4406 (о спецификациях Sender ID) и RFC7208 (о технологии SPF). Обе технологии названы экспериментальными, так что рынок сам должен решить, какая из них будет выбрана в качестве стандартной.

Пересылаемые сообщения, не соответствующие ни спецификациям Sender ID, ни протоколу SPF, являются серьезным недостатком обеих систем. Получатель проверяет запись SPF об оригинальном отправителе, чтобы открыть его список авторизованных серверов. Однако эти адреса не обязательно соответствуют серверам, выполняющим пересылку писем и задействованным в транспортировке данного сообщения. Следует быть осторожным, принимая решения на основе отказов системы SPF.

Системы DKIM

DKIM (DomainKeys Identified Mail — почта, идентифицированная доменными ключами) — это система криптографических подписей для сообщений электронной почты. Она позволяет получателю верифицировать не только личность отправителя, но и тот факт, что сообщение не было подделано по пути. Система использует DNS-записи для публикации доменных криптографических ключей и правил подписи сообщений. Система DKIM поддерживается всеми транспортными агентами, описанными в этой главе, но на практике встречается крайне редко.

18.5. Конфиденциальность и шифрование сообщений

По умолчанию вся почта отправляется незашифрованной. Научите своих пользователей, что они никогда не должны отправлять конфиденциальные данные по электронной почте, если они не используют внешний пакет шифрования или ваша организация не предоставила централизованное решение для шифрования для электронной почты. Даже при шифровании конфиденциальность электронного сообщения никогда не может быть гарантирована на 100%.[5] Вы платите свои деньги, и у вас есть свои права.

Исторически наиболее распространенными внешними пакетами шифрования были Pretty Good Privacy (PGP), его свободный клон GPG и S/MIME. Как S/MIME, так и PGP документированы в серии спецификаций RFC, причем S/MIME отслеживает текущие стандарты. Наиболее распространенные пользовательские агенты поддерживают дополнительные модули для обоих решений.

Эти стандарты обеспечивают основу для конфиденциальности электронной почты, аутентификации, обеспечения целостности сообщений и невозможности отказа от авторства.[6] Но хотя PGP/GPG и S/MIME являются потенциально жизнеспособными решениями для специалистов, которые заботятся о конфиденциальности, они оказались слишком громоздкими для неискушенных пользователей. Оба стандарта требуют некоторого мастерства в управлении криптографическим ключом и понимания базовой стратегии шифрования.

Большинство организаций, которые обрабатывают конфиденциальные данные в электронной почте (особенно те, которые общаются с общественностью, например медицинские учреждения), выбирают централизованную службу, использующую закрытые технологии для шифрования сообщений. Такие системы могут использовать либо локальные решения (например, IronPort от Cisco), которые развертываются в ва-

[5] "Эксперт по компьютерной безопасности" Дональд Дж. Трамп как-то сказал: "Я не верю в это [в электронную почту], потому что, по-моему, во-первых, ее можно взломать. Но если я все же отправлю электронное письмо — если я вообще отправлю его когда-нибудь, — то сделаю это в порядке исключения. Я просто не верю в электронную почту". Мудрые слова.

[6] Кстати, если вы используете системы PGP/GPG и S/MIME, то вы можете повысить степень безопасности, часто меняя открытый ключ или сертификат.

шем центре обработки данных, либо облачные службы (такие как Zix, `zixcorp.com`), которые могут быть сконфигурированы для шифрования исходящих сообщений в соответствии с их содержимым или другими правилами. Централизованное шифрование электронной почты — это одна категория услуг, для которой лучше всего использовать коммерческое решение, а не изобретать свои собственные.

По крайней мере в области электронной почты системы предотвращения потери данных (data loss prevention — DLP) тесно связаны с централизованным шифрованием. DLP-системы стремятся предотвратить или как минимум обнаружить утечку конфиденциальной информации в потоке электронной почты, исходящей из вашей организацию. Они сканируют исходящую электронную почту в поисках потенциально секретного содержания. Подозрительные сообщения могут быть помечены, заблокированы или возвращены отправителям. Мы рекомендуем выбрать централизованную платформу шифрования, включающую возможности DLP, — одной проблемой будет меньше.

В дополнение к шифрованию транспорта между транспортными агентами важно обеспечить, чтобы связь между пользователем и агентом-получателем всегда была зашифрована, особенно потому, что этот канал обычно использует определенные формы учетных данных пользователя для установления связи. Убедитесь, что агентами доступа разрешены только безопасные версии протоколов IMAP и POP, поддерживающие технологию TLS. (Они известны как IMAPS и POP3S соответственно.)

18.6. Почтовые псевдонимы

Другой концепцией, которую широко используют агенты MTA, являются псевдонимы. Псевдонимы позволяют системному администратору или отдельным пользователям переадресовывать почту.[7] Их можно применять для задания списков рассылки, для пересылки почты между компьютерами, а также для того, чтобы к пользователям можно было обращаться по нескольким именам. Псевдонимы обрабатываются рекурсивно, поэтому могут указывать на объекты, которые, в свою очередь, тоже являются псевдонимами.

Системные администраторы часто используют роли или функциональные псевдонимы (например, `printers@example.com`), чтобы направлять сообщения определенной темы именно тому пользователю, который ею сейчас занимается. Кроме того, псевдонимы используются для пересылки результатов ночного сканирования на вирусы и для указания администратора, отвечающего за электронную почту.

Самый распространенный способ задать псевдоним — использовать простой файл, такой как **`/etc/mail/aliases`** (см. ниже). Этот метод впервые был внедрен в агенте передачи почты **`sendmail`**, но почтовые серверы Exim и Postfix тоже поддерживают его.

Большинство пользовательских агентов в некоторой степени поддерживают концепцию псевдонимов (обычно называемых "моя группа", "мои письма" или как-то еще). Однако пользовательский агент раскрывает такие псевдонимы еще до того, как почта достигнет агента передачи или транспортного агента. Эти псевдонимы являются внутренними для пользовательского агента и не требуют поддержки от остальных компонентов почтовой системы.

[7] По техническим причинам псевдонимы настраивает только системный администратор. Пользователи управляют маршрутизацией почты при помощи файла **`.forward`**, который в действительности не имеет прямого отношения к псевдонимам, но мы использовали оба этих средства.

Псевдоним можно также определить в файле пересылки в рабочем каталоге каждого пользователя (**~/.forward**). Эти псевдонимы, имеющие несколько необычную синтаксическую структуру, применяются ко всей почте, поступающей конкретному пользователю. Они часто используются для пересылки почты по другому адресу или для реализации автоответчика.

Транспортные агенты ищут псевдонимы в глобальных файлах **aliases** (**/etc/mail/aliases** или **/etc/aliases**), а затем в файлах пересылки получателя. Псевдонимы применяются только к сообщениям, которые транспортный агент считает локальными.

Формат записи в файле **alises** имеет следующий вид.

локальное_имя: получатель1,получатель2, ...

Здесь *локальное_имя* — оригинальный адрес, с которым будут сравниваться входящие сообщения, а список получателей содержит либо адреса получателей, либо другие псевдонимы. Строки, начинающиеся с отступа, считаются продолжением предыдущих строк.

С точки зрения почтовой системы файл **alises** заменяет файл **/etc/passwd**, поэтому запись

```
david: david@somewhere-else.edu
```

не позволит локальному пользователю `david` вообще получать сообщения электронной почты. Следовательно, при выборе новых имен для пользователей администраторы и инструменты **adduser** должны проверять как файл **passwd**, так и файл **aliases**.

Файл **aliases** всегда должен содержать псевдоним **postmaster**, перенаправляющий сообщение тому, кто обслуживает почтовую систему. Аналогично псевдоним **abuse** подходит для ситуаций, когда кто-то извне вашей организации хочет прислать вам жалобу по поводу спама или любых других подозрительных действий, проистекающих из вашей сети. Кроме того, в файле должен присутствовать псевдоним для автоматических сообщений от транспортного агента; обычно этот агент называется `Mailer-Daemon` и часто имеет псевдоним **postmaster**.

К сожалению, в настоящее время в почтовой системе происходит столько злоупотреблений, что некоторые организации настраивают эти стандартные контактные адреса так, чтобы отбрасывать почту вообще, вместо ее пересылки индивидуальным пользователям. Записи вроде

```
# Basic system aliases - these MUST be present
mailer-daemon:  postmaster
postmaster:     "/dev/null"
```

стали обычными. Мы не рекомендуем эту практику, поскольку люди, испытывающие проблемы с пересылкой почты в вашу организацию, будут сообщать об этом администратору почтового сервера.

Мы рекомендуем использовать такие записи.

```
# Basic system aliases - these MUST be present
mailer-daemon:  "/dev/null"
postmaster:     root
```

Вы должны перенаправлять почту, поступающую на пользователю `root`, системным администраторам или тому лицу, которое ведет ежедневные регистрационные записи. Учетные записи `bin`, `sys`, `daemon`, `nobody` и `hostmaster` (а также любые другие учетные записи псевдопользователей) должны иметь похожие псевдонимы.

Помимо списков пользователей, псевдонимы могут ссылаться на следующие файлы и программы.

- Файл, содержащий список адресов.
- Файл, в который должны добавляться сообщения.
- Программу, на вход которой должны передаваться сообщения.

Два последних адресата должны вызывать подозрения с точки зрения безопасности, поскольку отправитель сообщения полностью определяет его содержание. Иметь возможность добавлять это содержание в файл или передавать его командам в качестве аргументов — значит подвергать систему опасности. Многие транспортные агенты либо не разрешают передавать сообщения таким адресатам, либо строго ограничивают список разрешенных команд и файлов.

Псевдонимы могут создавать циклические ссылки. Транспортные агенты пытаются распознавать циклы, которые вызывают бесконечную циркуляцию сообщений от получателя к отправителю и обратно, и возвращать ошибочные сообщения их отправителям. Для того чтобы распознать циклическую ссылку, транспортный агент может подсчитывать количество строк `Received` в заголовке сообщения и прекращать ретрансляцию, если это количество превышает установленный предел (обычно 25). Каждое посещение нового компьютера на жаргоне электронной почты называется "скачком" ("hop"). Возвращение сообщения его отправителю называется "рикошетом" ("bouncing"). Итак, типичное выражение на этом жаргоне может звучать следующим образом: "Письмо рикошетит после 25 скачков".[8] Другой способ распознавания циклов заключается в добавлении заголовка `Delivered-To` для каждого компьютера, которому пересылается сообщение. Если транспортный агент обнаружит в этом заголовке адрес, который уже упоминался ранее, то он будет знать, что письмо "ходит по кругу".

Загрузка псевдонимов из файла

Директива `:include:` в файле **aliases** (или пользовательском файле **.forward**) является прекрасным средством управления псевдонимами с помощью специального файла. Это отличное средство, позволяющее пользователям создавать свои собственные списки рассылки. Этот файл управляется только пользователем и его можно менять без вмешательства системного администратора. Однако такой псевдоним легко может стать пособником при рассылке спама, поэтому не следует разрешать ретрансляцию сообщений, поступающих извне, на эти адреса.

Для создания такого списка рассылки с помощью директивы `:include:` системный администратор должен ввести псевдоним в глобальный файл **aliases**, а затем создать внешний файл и при помощи команды **chown** сделать его владельцем пользователя, управляющего списком рассылки. Например, файл **aliases** может содержать следующую строку.

```
sa-book: :include:/usr/local/mail/ulsah.authors
```

Файл **ulsah.authors** должен находиться в локальной файловой системе. Кроме того, запись в этот файл должна быть разрешена только его владельцу. Для полноты следует также создать псевдоним, соответствующий владельцу списка рассылки, чтобы сообщения об ошибках ("рикошеты") посылались владельцу, а не отправителю сообщения.

```
owner-sa-book: evi
```

[8] Мы не придерживались последовательной терминологии в этой главе, иногда называя возврат сообщения "рикошетом", а иногда "ошибкой". На самом деле мы имеем в виду, что в этом случае генерируется уведомление о состоянии доставки (DSN, представляющее собой сообщение, форматированное специальным образом). Такое уведомление обычно означает, что сообщение не было доставлено и, следовательно, возвращается отправителю.

Направление почты в файл

Если объект, на который ссылается псевдоним, — полное имя файла (заключенное в двойные кавычки при наличии специальных символов), то сообщения добавляются в конец указанного файла. Сам файл должен существовать. Вот пример такого псевдонима.

```
cron-status: /usr/local/admin/cron-status-messages
```

Возможность посылать почту в файл или на вход программы очень полезна, но она ослабляет безопасность и потому должна быть ограничена. Приведенный синтаксис допустим только в файле **aliases** и пользовательском файле **.forward** (а также в файлах, которые внедряются в них посредством директивы `:include:`). Имя файла не трактуется как адрес, поэтому почта, направленная по адресу `/etc/passwd@host.domain`, будет возвращена.

Если ссылка на внешний файл содержится в файле **aliases**, то адресуемый файл либо должен быть полностью открыт для записи (что нежелательно), либо иметь установленный бит смены идентификатора пользователя (SUID) и быть неисполняемым, либо принадлежать основному пользователю транспортного агента. Этот пользователь задается в файле конфигурации транспортного агента.

Если же ссылка на файл содержится в файле **.forward**, то адресуемый файл должен принадлежать и быть доступным для записи основному получателю сообщения. Необходимо, чтобы у этого пользователя была своя запись в файле **/etc/passwd**, а его интерпретатор команд был упомянут в файле **/etc/shells**. Для файлов, принадлежащих пользователю `root`, следует задать режим доступа 4644 или 4600, т.е. установить бит `setuid` и отменить возможность выполнения.

Направление почты в программу

Благодаря псевдонимам можно организовать пересылку почты на вход заданной программы. Это реализуется с помощью строки примерно следующего вида.

```
autoftp: "|/usr/local/bin/autologger"
```

Однако использование такого способа доставки почты создает еще более серьезные бреши в защите, чем направление почты в файл, поэтому данный механизм тоже разрешен только для файлов **aliases** и **.forward**, а также для файлов, которые подключаются к ним посредством директивы `:include:`, и часто требует использования ограниченной оболочки (restricted shell).

Хешированная база данных псевдонимов

Поскольку записи файла **aliases** не упорядочены, прямой поиск в данном файле был бы для агента MTA неэффективным. По этой причине средствами системы Berkeley DB создается хешированная версия файла **aliases**. Хеширование значительно ускоряет поиск псевдонимов, особенно в больших файлах.

Файлы, образованные из файла **/etc/mail/aliases**, называются **aliases.db**. При каждом изменении файла **aliases** хешированную базу данных следует перестраивать с помощью команды **newaliases**. Если команда **newaliases** вызывается в автоматическом режиме, сохраняйте выводимые ею сообщения об ошибках, поскольку могут возникнуть проблемы с форматированием.

18.7. Конфигурация электронной почты

Ядром системы электронной почты является ее транспортный агент. Программа **sendmail** была первым транспортным агентом системы UNIX, написанной много лет тому назад выпускником университета Беркли (UC Berkeley) Эриком Аллманом (Eric Allman). С тех пор были разработаны другие транспортные агенты. Одни из них являются коммерческими продуктами, а другие — программами с открытым кодом. В этой главе мы рассмотрим три почтовых транспортных агента с открытым кодом: **sendmail**, Postfix, написанный Витцем Венема (Wietse Venema) из компании IBM Research, и Exim, написанный Филипом Гейзелом (Philip Hazel) из Кембриджского университета.

После высокоуровневого проектирования почтовой системы ее настройка является второй по важности заботой системного администратора. К счастью, образцы конфигурации или примеры настроек, поставляемые вместе с транспортными агентами, часто оказываются очень близкими к тому, что требуется для среднестатистического сервера. Конфигурацию транспортного агента не следует начинать с нуля.

Сайт SecuritySpace (`securityspace.com`) ежемесячно проводит исследования, чтобы определить рыночную долю разных транспортных агентов. В отчете за июнь 2017 года было указано, что из 2 миллионов опрошенных транспортных агентов были получены ответы от 1,7 миллиона, в которых указана строка идентификации MTA. Результаты этого исследования приведены в табл. 18.3 наряду с результатами исследований, проведенных в 2009 году, и некоторыми значениями, полученными в 2001 году.

Таблица 18.3. Рыночная доля почтовых транспортных агентов

			Рыночная доля, %		
MTA	**Источник**	**ОС**	**2017**	**2009**	**2001**
Exim	`exim.org`	Debian	56	30	8
Postfix	`postfix.org`	Red Hat, Ubuntu	33	20	2
MS Exchange	`microsoft.com/exchange`	—	1	20	4
sendmail	`sendmail.org`	FreeBSD	5	19	60
Другие	—	—	<3%	<3%	<3%

Очевиден переход лидерства от программы **sendmail** к пакетам Exim и Postfix, в то время как компания Microsoft сначала достигла лидерства на рынке транспортных агентов, а потом уступила его. Обратите внимание на то, что эти данные учитывают только тех транспортных агентов, которые имели выход в Интернет.

Для каждого из упомянутых транспортных агентов мы приводим детали конфигурации, необходимой для выполнения их функций, включая следующие.

- Конфигурация простых клиентов.
- Конфигурация почтового сервера, имеющего выход в Интернет.
- Управление маршрутизацией входящей и исходящей корреспонденции.
- Штемпелевание почты, поступающей от центрального сервера или самого домена.
- Обеспечение безопасности.
- Отладка.

Если вы реализуете почтовую систему с нуля и не придерживаетесь заранее установленных правил или предпочтений, вам может быть трудно выбрать транспортный агент.

Программа **sendmail** определенно является самой сложной, за исключением, возможно, сайтов, использующих исключительно систему FreeBSD. Пакет Exim — мощный инструмент с тонкими настройками, но тоже слишком сложный. Пакет Postfix проще, быстрее и в основном ориентирован на обеспечение безопасности. Если ваша организация или системные администраторы ранее работали с какими-то транспортными агентами, они, вероятно, не захотят переключаться на что-то другое, если необходимые вам функциональные возможности не реализованы в старых агентах.

Конфигурация агента **sendmail** описывается в следующем разделе. Конфигурации пакета Exim описаны в разделе 18.9, а конфигурация пакета Postfix — в разделе 18.10.

18.8. Почтовый агент sendmail

Почтовый агент **sendmail** распространяется в виде открытого кода и доступен на сайте `sendmail.org`, но в настоящее время его редко приходится собирать с самого начала.[9] При желании сделать это можно обратиться за инструкциями, находящимися в файле **INSTALL** в корневом каталоге дистрибутива. Если необходимо обойти определенные установки, сделанные по умолчанию, то можно найти их в файле **devtools/OS/***имя-вашей-операционной-системы*. Можно также добавить новые функциональные свойства, отредактировав файл **devtools/Site/site.config.m4**.

Программа **sendmail** использует препроцессор макросов **m4** не только во время компиляции, но и для конфигурирования. Файл конфигурации **m4** обычно называется *имя_компьютера*.**mc**. Он компилируется из довольно понятного синтаксиса в совершенно невразумительный низкоуровневый язык и записывается в файл *имя_компьютера*.**cf**, который в свою очередь инсталлируется как **/etc/mail/sendmail.cf**.

Для того чтобы увидеть версию программы **sendmail**, инсталлированной на вашем компьютере, а также узнать, как она скомпилировалась, попытайтесь выполнить следующую команду.

```
linux$ /usr/sbin/sendmail -d0.1 -bt < /dev/null
Version 8.13.8
Compiled with: DNSMAP HESIOD HES_GETMAILHOST LDAPMAP LOG
   MAP_REGEX MATCHGECOS MILTER MIME7TO8 MIME8TO7 NAMED_BIND
   NETINET NETINET6 NETUNIX NEWDB NIS PIPELINING SASLv2 SCANF
   SOCKETMAP STARTTLS TCPWRAPPERS USERDB USE_LDAP_INI
============ SYSTEM IDENTITY (after readcf) =================
        (short domain name) $w = ross
    (canonical domain name) $j = ross.atrust.com
           (subdomain name) $m = atrust.com
                (node name) $k = ross.atrust.com
=============================================================
```

Эта команда переводит программу **sendmail** в режимы проверки адреса (**-bt**) и отладки (**-d0.1**), но не вводит никакого адреса для тестирования (**</dev/null**). Побочный эффект заключается в том, что программа **sendmail** сообщает номер своей версии и флаги компилятора, с которыми она компилировалась. Зная номер версии, вы можете зайти на веб-сайт `sendmail.org` и увидеть, существуют ли у этой версии программы скрытые недостатки.

[9] По состоянию на октябрь 2013 г. программа **sendmail** поддерживалась и распространялась акционерной компанией Proofprint, Inc.

Для того чтобы найти файлы программы **sendmail** в своей системе, посмотрите в начало инсталлированного файла **/etc/mail/sendmail.cf**. Комментарии, которые там находятся, содержат имя каталога, в котором была собрана данная конфигурация. Этот каталог, в свою очередь, приведет вас к файлу **.mc**, являющемуся исходным источником конфигурации.

Большинство поставщиков программы **sendmail** включают в дистрибутивный пакет не только исполняемый модуль, но и каталог **cf** из дистрибутивного дерева, который они прячут среди файлов операционной системы. Найти его поможет табл. 18.4.

Таблица 18.4. Местоположение каталога конфигурации программы sendmail

Система	Каталог
Ubuntu	**/usr/share/sendmail**
Debian	**/usr/share/sendmail**
Red Hat	**/etc/mail**
CentOS	**/etc/mail**
FreeBSD	**/etc/mail**

Файл переключения

🕮 Переключение служб более подробно описано в разделе 17.3.

В большинстве систем существует конфигурационный файл "переключения служб" **/etc/nsswitch.conf**, в котором перечисляются методы, удовлетворяющие разным стандартным запросам, например запросам поиска пользователя и компьютера. Если для данного типа запросов в файле указано несколько методов распознавания, то файл переключения служб также определит порядок, в котором будут проверяться эти методы.

Существование переключения служб в программном обеспечении обычно не заметно для программ. Однако программа **sendmail** осуществляет очень плотный контроль над своими процедурами поиска, поэтому в настоящее время она игнорирует системный файл переключения служб и использует свой собственный внутренний файл конфигурации служб (**/etc/mail/service.switch**).

На почтовую систему оказывают влияние два поля в файле переключения служб: `aliases` и `hosts`. Поле `hosts` может иметь значения `dns`, `nis`, `nisplus` и `files`, а поле `aliases` — `files`, `nis`, `nisplus` и `ldap`. Вспомогательные файлы для механизмов, которые вы используете (за исключением `files`), должны быть скомпилированы в программе **sendmail** до того, как данная служба будет использована.

Запуск программы sendmail

Программа **sendmail** не должна запускаться демонами **inetd** или **systemd**, поэтому ее необходимо явно запускать во время загрузки. Подробности этой процедуры описаны в главе 2.

Функционирование программы **sendmail** определяется флагами, с которыми она запускается. Программу **sendmail** можно запускать в нескольких режимах, выбранных с помощью флага **-b**, который означает слово "be" ("быть") или "become" ("становиться") и всегда используется в сочетании с другим флагом, определяющим роль программы **sendmail**. Допустимые значения флага **-b** и флага **-A**, который производит выбор между агентами MTA и MSA, приведены в табл. 18.5.

Таблица 18.5. Флаги командной строки для основных режимов работы программы sendmail

Флаг	Описание
`-Ac`	Использует файл конфигурации `submit.cf` и действует как агент доставки
`-Am`	Использует файл конфигурации `sendmail.cf` и действует как транспортный агент
`-ba`	Функционирует в режиме ARPANET (ожидает управляющих символов `CR/LF` в конце строки)
`-bd`	Функционирует в режиме демона и прослушивает соединения на порту 25
`-bD`	Функционирует в режиме демона, но явно, а не в фоновом режиме[a]
`-bh`	Просматривает информацию о последнем соединении (аналогично команде `hoststat`)
`-bH`	Стирает с диска копию информации об устаревших соединениях (аналогично команде `purgestat`)
`-bi`	Инициализирует хешированные псевдонимы (аналогично команде `newaliases`)
`-bm`	Функционирует как почтальон, доставляет почту как обычно (по умолчанию)
`-bp`	Выводит на печать очередь почтовых сообщений (аналогично команде `mailq`)
`-bP`	Выводит на печать количество записей в очередях, расположенных в совместно используемой памяти
`-bs`	Устанавливает режим сервера SMTP (для стандартного входа, но не для порта 25)
`-bt`	Устанавливает режим проверки адреса
`-bv`	Только проверяет почтовые адреса; не посылает почту

[a] Этот режим используется для отладки, чтобы увидеть сообщения об ошибках.

Если вы конфигурируете сервер, который будет принимать почту из Интернета, запустите программу **sendmail** в режиме демона (**-bd**). В этом режиме программа **sendmail** прослушивает сетевой порт 25 и ожидает задания.[10] Обычно, кроме него, устанавливается флаг **-q**, задающий интервал времени, в течение которого программа **sendmail** обрабатывает почтовую очередь. Например, флаг **-q30m** заставляет программу проверять очередь каждые 30 минут, а **-q1h** — каждый час.

Программа **sendmail** обычно пытается доставить сообщение немедленно, сохраняя его в очереди только на короткий промежуток времени, чтобы гарантировать надежность работы. Однако, если ваш сервер перегружен или адресат недоступен, программа **sendmail** размещает сообщение в очереди и пытается послать его снова. Программа **sendmail** использует обработчиков персистентной очереди, которые обычно запускаются во время загрузки. Она выполняет блокировку, поэтому параллельное выполнение нескольких обработчиков очереди вполне безопасно. Функциональная возможность "группы очередей" позволяет работать со списками рассылки и очередями.

Программа **sendmail** считывает свой конфигурационный файл **sendmail.cf** только в момент запуска. Следовательно, если вы изменили конфигурационный файл, то должны либо прекратить выполнение и запустить программу **sendmail** заново, либо послать ей сигнал HUP. Программа **sendmail** создает файл **sendmail.pid**, содержащий идентификатор процесса и команду его запуска. Вы должны запускать программу **sendmail**, используя полностью определенный путь, потому что она, получив сигнал HUP, применяет сама к себе команду **exec**. Файл **sendmail.pid** позволяет отправить сигнал HUP работающему процессу с помощью следующей команды.

```
$ sudo kill -HUP `head -1 sendmail.pid`
```

[10] Порты, которые прослушивает программа **sendmail**, определяются параметром DAEMON_OPTIONS; порт 25 задается по умолчанию.

Местоположение файла **.pid** зависит от операционной системы. Обычно он находится по пути **/var/run/sendmail.pid** или **/etc/mail/sendmail.pid**, но его можно задать в конфигурационном файле с помощью параметра confPID_FILE.

```
define(confPID_FILE, `/var/run/sendmail.pid')
```

Почтовые очереди

Программа **sendmail** использует по крайней мере две очереди: **/var/spool/mqueue**, когда функционирует как транспортный агент на порту 25, и **/var/spool/clientmqueue**, когда функционирует как агент доставки на порту 587.[11] Все сообщения, как минимум на короткое время оказываются в очереди, прежде чем отправиться в дальнейший путь.

Сообщения, находящиеся в очереди, сохраняются в виде фрагментов в разных файлах. Каждое имя файла имеет двухбуквенный префикс, идентифицирующий фрагмент, за ним следует случайный идентификатор, построенный на основе идентификатора процесса в программе **sendmail**. Шесть возможных фрагментов приведены в табл. 18.6.

Таблица 18.6. Префиксы для файлов из очереди

Префикс	Содержимое файла
qf	Заголовок сообщения и управляющий файл
df	Тело сообщения
tf	Временная версия файла **df** в момент его обновления
Tf	Информация о том, что было выполнено более 32 неудачных попыток блокировки
Qf	Информация о том, что сообщение было отправлено обратно и не может быть восстановлено
xf	Временный файл транскрипции сообщений об ошибках, поступивших от программы для рассылки писем

Если подкаталоги **qf**, **df** и **xf** в каталоге очереди уже существуют, то эти фрагменты сообщения помещаются в соответствующий подкаталог. Файл **qf** содержит не только заголовок сообщения, но и адреса, указанные на конверте, дату, при наступлении которой сообщение следует вернуть как недоставленное, приоритет сообщения в очереди и причину, по которой сообщение оказалось в очереди. Каждая строка начинается с однобуквенного кода, идентифицирующего остальную часть строки.

Каждое сообщение, помещенное в очередь, должно иметь файлы **qf** и **df**. Все остальные префиксы программа **sendmail** использует, пытаясь доставить сообщение. Если компьютер дает сбой и перезагружается, то последовательность загрузочных команд для программы **sendmail** должна удалить файлы **tf**, **xf** и **Tf** из всех очередей. Системный администратор, ответственный за почту, должен иногда проверять файлы **Qf**, если локальная конфигурация вызывает рикошет. Эпизодическая проверка каталогов очереди позволит исправить проблемы до того, как произойдет катастрофа.

Почтовая очередь создает несколько предпосылок для неприятностей. Например, файловая система может переполниться (избегайте размещения каталогов **/var/spool/mqueue** и **/var/log** в одном и том же разделе диска), очередь может "засориться", а беспризорные почтовые сообщения могут "застрять" в очереди. В конфигурации программы **sendmail** есть параметры, обеспечивающие высокую производительность на очень загруженных компьютерах.

[11] Для повышения производительности работы программа **sendmail** может использовать несколько очередей в каталоге **mqueue**.

Конфигурация программы **sendmail**

Действия программы **sendmail** определяются одним конфигурационным файлом, который обычно называется **/etc/mail/sendmail.cf**, если программа **sendmail** запускается как транспортный почтовый агент, или **/etc/mail/submit.cf**, если программа **sendmail** действует как агент доставки. Флаги, с которыми запускается программа **sendmail**, определяют, какой файл конфигурации используется: флаги **-bm**, **-bs** и **-bt** означают использование файла **submit.cf**, если он существует, а все другие режимы используют файл **sendmail.cf**. Эти имена можно изменить с помощью флагов в командной строке или параметров конфигурационного файла, но лучше этого не делать.

Формат конфигурационного файла был разработан для того, чтобы облегчить его разбор с помощью компьютера, а не человека. Исходный файл препроцессора **m4** (с расширением **.mc**), который порождает файл с расширением **.cf**, является усовершенствованием, но его капризный и строгий синтаксис очень неудобен. К счастью, многие парадигмы, которые вы, возможно, захотите использовать, уже были предусмотрены и реализованы в дистрибутивном пакете в качестве готовых решений.

Настройка конфигурации программы **sendmail** состоит из нескольких этапов.

1. Выбор роли для компьютера, который вы конфигурируете: клиент, сервер, получатель почты через Интернет и т.д.
2. Выбор функциональных возможностей, необходимых для реализации этой роли, и создание файла с расширением **.mc** для данной конфигурации.
3. Компиляция файла с расширением **.mc** с помощью препроцессора **m4**, чтобы создать файл конфигурации **.cf**.

Мы опишем функциональные возможности, типичные для общедоступных в рамках организации серверов, имеющих выход в Интернет, и для небольших настольных клиентов. Более подробное описание можно найти в документации программы **sendmail**, в книге *Sendmail* Брайана Косталеса и его соавторов (Bryan Costales et al.), а также в файле **cf/README** из дистрибутивного пакета.

Препроцессор m4

Изначально предназначенный для использования в сочетании с языками программирования, препроцессор **m4** позволяет пользователям писать более удобочитаемые (или, наоборот, запутанные) программы. Это достаточно мощный инструмент, чтобы оказаться полезным во многих ситуациях, связанных с преобразованием входной информации, он прекрасно работает с конфигурационными файлами программы **sendmail**.

Макрос препроцессора **m4** имеет следующую форму.

```
name(arg1, arg2, ..., argn)
```

Между именем и открывающей скобкой не должно быть пробелов. Левая и правая одинарные кавычки обозначают строки в качестве аргументов. Соглашения о кавычках в макросах препроцессора **m4** являются странными, поскольку левая и правая кавычки — разные символы. Кроме того, кавычки могут быть вложенными.

Препроцессор **m4** имеет несколько встроенных макросов, и пользователи могут определять свои собственные макросы. Основные встроенные макросы, используемые в конфигурации программы **sendmail**, приведены в табл. 18.7.

Таблица 18.7. Макросы препроцессора m4, часто используемые программой sendmail

Макрос	Функция
`define`	Определяет макрос с именем *arg1* со значением *arg2*
`divert`	Управляет выходными потоками
`dnl`	Игнорирует все символы до начала новой строки
`include`	Включает (интерполирует) файл с именем *arg1*
`undefine`	Отменяет предыдущее определение макроса с именем *arg1*

Фрагменты конфигурации программы sendmail

Дистрибутивный пакет программы **sendmail** содержит подкаталог **cf**, в котором находятся все фрагменты, необходимые для конфигурации препроцессора **m4**. Если вы не самостоятельно инсталлируете программу **sendmail** на основе исходного текста, а полагаетесь на поставщика, то используйте местоположение каталога **cf**, указанное в табл. 18.4. Документация о конфигурации программы **sendmail** записана в файле **README**. Подкаталоги, перечисленные в табл. 18.8, содержат примеры и фрагменты, которые можно включать в свою конфигурацию.

Таблица 18.8. Подкаталоги конфигурации программы sendmail

Каталог	Содержимое
cf	Эталонные файлы **.mc** (мастера конфигурации)
domain	Эталонные файлы m4 для разных доменов в домене Berkeley
feature	Фрагменты, реализующие разные функциональные возможности
hack	Особые функциональные возможности, ценность или реализация которых сомнительна
m4	Основной файл конфигурации и другие важные файлы
ostype	Местоположение файлов, зависящее от операционной системы, и другая полезная информация
mailer	Файлы m4, описывающие основные программы для рассылки (агенты доставки)
sh	Сценарии оболочки, используемые в препроцессоре m4

Каталог **cf/cf** содержит примеры файлов с расширением **.mc**. Фактически он содержит так много примеров, что в них легко запутаться. Мы рекомендуем хранить свои файлы с расширением **.mc** отдельно от примеров, размещенных в каталоге **cf**. Для этого следует либо создать новый каталог для своего сайта (**cf/***имя_сервера*), либо разместить каталог **cf** отдельно под именем **cf.examples** и создать в нем новый подкаталог **cf**. Если вы сделаете это, скопируйте в свой новый каталог сценарии **Makefile** и **Build**, чтобы инструкции из файла **README** сохранили свою работоспособность. В качестве альтернативы можно скопировать все собственные файлы конфигурации с расширением **.mc** в центральное место хранения, а не оставлять их в дистрибутивном пакете программы **sendmail**. Сценарий **Build** использует относительные пути, поэтому, если вы хотите создать файл с расширением **.cf** из файла с расширением **.mc** и не использовать имена их дистрибутивной иерархии программы **sendmail**, придется их изменить.

Файлы из каталога **cf/ostype** конфигурируют программу **sendmail** для любой конкретной операционной системы. Многие из них являются заранее заданными, но если вы изменили настройки своей системы, то вам придется либо модифицировать их, либо

создавать новые. Скопируйте тот из них, который лучше соответствует вашей системе, и присвойте ему новое имя.

Каталог **cf/feature** — это место, где вы будете искать все фрагменты конфигурации, которые вам нужны. Там найдутся все функциональные возможности, которые могут оказаться полезными для сайта, использующего программу **sendmail**.

Остальные подкаталоги каталога **cf** служат, скорее, как заглушки. Их не надо исследовать или анализировать — просто используйте их.

Конфигурационный файл, построенный на основе эталонного файла с расширением .mc

Перед тем как погрузиться в детали различных конфигурационных макросов, свойств и параметров, которые могут использоваться при конфигурировании программы **sendmail**, создадим простую конфигурацию, тем самым проиллюстрировав весь процесс. Предметом нашего примера будет краевой хост `myhost.example.com`; главный конфигурационный файл называется **myhost.mc**. Рассмотрим полный файл с расширением **.mc**.

```
divert(-1)
#### basic .mc file for example.com
divert(0)
VERSIONID(`$Id$')
OSTYPE(`linux')
MAILER(`local')
MAILER(`smtp')
```

За небольшим исключением, каждая строка содержит вызов готового макроса. Первые четыре строки представляют собой шаблон; они вставляют комментарий в скомпилированный файл, чтобы указать версию программы **sendmail**, созданный каталог конфигурации и т.д. Макрос `OSTYPE` интерполирует файл **../ostype/linux.m4**. Строки, содержащие вызов макроса `MAILER`, разрешают локальную доставку сообщений (пользователям, имеющим учетные записи на сервере `myhost.example.com`), а также доставку почтовым серверам в Интернете.

Для того чтобы создать реальный конфигурационный файл, просто примените команду **Build** к новому скопированному вами каталогу **cf**.

```
$ ./Build myhost.cf
```

В заключение инсталлируйте файл **myhost.cf** в соответствующем месте — в большинстве случаев как файл **/etc/mail/sendmail.cf**, хотя некоторые поставщики могут поместить его в другое место. Чаще всего поставщики размещают его в каталоге **/etc** и **/usr/lib**.

Для более крупных почтовых серверов можно создать отдельный файл препроцессора **m4**, чтобы хранить там значения, заданные по умолчанию; поместите его в каталог **cf/domain**. Отдельные хосты могут затем включать его содержимое, используя макрос `DOMAIN`. Отдельный файл конфигурации нужен не каждому компьютеру, но каждая группа похожих компьютеров (с одинаковой архитектурой и одинаковыми ролями: сервер, клиент и т.д.), вероятно, нуждается в собственной конфигурации.

Порядок следования макросов в файле с расширением **.mc** не случаен. Он должен быть таким.

```
VERSIONID
OSTYPE
```

```
DOMAIN
FEATURE
определения локальных макросов
MAILER
```

Даже имея простую систему конфигурации **m4** для программы **sendmail**, вам, возможно, придется принимать несколько разных решений, связанных с конфигурацией вашего почтового сервера. Читая о функциональных возможностях, описанных ниже, подумайте о том, как их применять к вашей сети. Небольшая сеть, вероятно, будет иметь один концентратор и концевые хосты, а значит, только два варианта файла конфигурации. В более крупной сети могут понадобиться отдельные концентраторы для входящей и исходящей почты и, вероятно, отдельный сервер POP/IMAP.

Независимо от сложности вашей сети и ее подключения ко внешнему миру (открытой, за брандмауэром или виртуальной частной сети, например), скорее всего, вы найдете в каталоге **cf** готовые фрагменты, подходящие для настройки и использования.

Примитивы конфигурации программы sendmail

Команды конфигурации **sendmail** чувствительны к регистру клавиатуры. По общепринятому соглашению, имена заранее определенных макросов состоят только из прописных букв (например, OSTYPE), команды препроцессора **m4** — только из строчных букв (например, define). Названия параметров конфигурации начинаются с префикса conf, набранного в нижнем регистре, и заканчиваются именем переменной, набранным в верхнем регистре (например, confFALLBACK_MX). Макросы обычно ссылаются на файл препроцессора **m4** с именем *../имя_макроса/arg1*.**m4**. Например, ссылка OSTYPE(`linux') включает файл **../ostype/linux.m4**.

Таблицы и базы данных

Перед тем как погрузиться в детали, связанные с конкретными примитивами конфигурации, сначала следует обсудить таблицы (иногда называемые *ассоциативными массивами* (maps) или *базами данных*), которые программа **sendmail** может использовать для маршрутизации почты или перезаписи адреса. Большинство из них используется в сочетании с макросом FEATURE.

Таблица — это кеш (как правило, текстовый файл), содержащий информацию о маршрутизации, псевдонимах, правилах, а также другие данные, которые можно преобразовать в формат базы данных с помощью команды **makemap**, а затем использовать как источник информации при выполнении одной или нескольких операций программы **sendmail**. Хотя данные для программы **sendmail** обычно хранятся в текстовом файле, они могут также поступать из системы DNS, от сервера NIS, от протокола LDAP или из других источников. Использование централизованного сервера IMAP освобождает программу **sendmail** от выполнения рутинных операций, связанных с розыском пользователей и их устаревших таблиц.

Программа **sendmail** предусматривает три типа ассоциативных массивов.

- dbm — использует расширяемые алгоритмы хеширования (**dbm/ndbm**).
- hash — использует стандартную схему хеширования (DB).
- btree — использует B-дерево (DB).

В большинстве случаев использования таблиц в программе **sendmail** наилучшим является формат базы данных hash, заданный по умолчанию. Для создания базы данных

из текстового файла следует использовать команду **makemap**, задав тип и имя выходного файла базы данных. Текстовая версия базы данных должна передаваться на стандартный вход команды **makemap**. Рассмотрим пример.

```
$ sudo makemap hash /etc/mail/access < /etc/mail/access
```

На первый взгляд эта команда выглядит ошибкой, которая может вызвать перезапись входного файла пустым результирующим файлом. Однако команда **makemap** добавляет специальный суффикс, так что реальный результирующий файл будет называться **/etc/mail/access.db** и конфликт не возникнет. Каждый раз при изменении текстового файла файл базы данных необходимо создавать заново с помощью команды **makemap** (но программа **sendmail** не требует использования сигнала HUP).

В текстовых файлах, на основе которых создаются ассоциативные массивы, могут находиться комментарии. Они начинаются символом # и продолжаются до конца строки.

В большинстве случаев в качестве ключа базы данных используется самое длинное из возможных совпадений. Как и в любой другой хешированной структуре, порядок следования записей во входном текстовом файле не имеет значения. Примитивы `FEATURE`, ожидающие файл базы данных как параметр, по умолчанию предусматривают в качестве типа базы данных тип `hash`, а в качестве имени базы данных — имя **/etc/mail/***имя_таблицы*`.db`.

Обобщенные макросы и функциональные возможности

В табл. 18.9 перечислены основные примитивы конфигурации с указанием степени их использования (да, нет, возможно) и кратким описанием их действий. Более подробно они будут рассмотрены ниже.

Таблица 18.9. Основные примитивы конфигурации программы `sendmail`

Примитив	Используется?	Описание
`OSTYPE`	Да	Включает пути, специфичные для операционной системы, и флаги программы рассылки
`DOMAIN`	Нет	Включает детали конфигурации, специфичные для сайта
`MAILER`	Да	Допускает использование программ рассылки, обычно `smtp` и `local`
`FEATURE`	Возможно	Допускает использование набора функциональных возможностей программы **sendmail**
`use_cw_file`	Да (серверы)	Список хостов, для которых сервер принимает почту
`redirect`	Возможно (серверы)	Перенаправляет письма при перемещении пользователей
`always_add_domain`	Да	Полностью определяет имена компьютеров, если сервер UA этого не сделал
`access_db`	Возможно (серверы)	Задает базу данных хостов, для которых выполняется ретрансляция сообщений
`virtusertable`	Возможно (серверы)	Включает псевдонимы доменов (виртуальные домены)
`ldap_routing`	Возможно (серверы)	Маршрутизирует входящую почту с помощью сервера LDAP
`MASQUERADE_AS`	Да	Маскирует всю почту так, будто она исходит из одного источника
`EXPOSED_USER`	Да	Содержит список пользователей, для которых не следует выполнять маскировку сообщений

Окончание табл. 18.9

Примитив	Используется?	Описание
MAIL_HUB	Да (серверы)	Задает почтовый сервер, предназначенный для входящей почты
SMART_HOST	Да (клиенты)	Задает почтовый сервер, предназначенный для исходящей почты

Макрос OSTYPE

Файл OSTYPE содержит разнообразную информацию от производителей, например ожидаемое местонахождение файлов, связанных с почтовой системой, пути к командам, которые необходимы для программы **sendmail**, флаги для программ рассылки и т.д. Список всех переменных, которые можно определить в файле OSTYPE, перечислены в файле **cf/README**. [12]

Макрос DOMAIN

Директива DOMAIN позволяет указывать всю общую информацию о сайте в одном месте (**cf/domain/***имя_файла*.**m4**), а затем включать в каждый конфигурационный файл компьютера.

```
DOMAIN(`имя_файла')
```

Макрос MAILER

Этот макрос следует включать в каждый агент доставки, который вы хотите использовать. Полный список программ рассылки можно найти в каталоге **cf/mailers**, но обычно необходимы только local, smtp и, возможно, cyrus. Строки, содержащие макрос MAILER, обычно являются последними в файле **.mc**.

Макрос FEATURE

Рассматриваемый макрос позволяет реализовать большое количество сценариев (по последним данным, 56!), включая файлы препроцессора **m4** из каталога **feature**. Его синтаксическая конструкция выглядит следующим образом.

```
FEATURE(keyword, arg, arg, ...)
```

Здесь аргумент *keyword* соответствует файлу *keyword*.**m4** в каталоге **cf/feature**. Количество остальных аргументов не должно превышать девяти.

Функциональная возможность use_cw_file

Внутренний класс w программы **sendmail** (отсюда имя **cw**) содержит имена всех локальных компьютеров, к которым есть доступ из данного хоста и может доставляться почта. Эта функциональная возможность позволяет доставлять почту хостам, построчно перечисленным в списке, находящемся в файле **/etc/mail/local-host-names**. Эту функциональную возможность активизирует конфигурационный файл.

```
FEATURE(`use_cw_file')
```

Клиентская машина на самом деле не нуждается в этой возможности, если у нее есть псевдоним. Файл **local-host-names** должен содержать имена всех локальных хостов

[12] Так где же все-таки определен макрос OSTYPE? В одном из файлов в каталоге **cf/m4**, который таинственным образом добавляется к вашему конфигурационному файлу при запуске сценария **Build**.

и виртуальных доменов, которым можно доставлять почту, включая сайты, резервные записи MX которых ссылаются на ваш сервер.

Если эта функциональная возможность не подключена, то программа **sendmail** выполняет локальную доставку почты, только если она адресована компьютеру, на котором выполняется сама программа **sendmail**.

Если вы добавили в сеть новый компьютер, то должны добавить его имя в файл **local-host-names** и послать сигнал HUP программе **sendmail**, чтобы изменения вступили в силу.

Функциональная возможность `redirect`

Когда люди покидают вашу организацию, вы обычно либо пересылаете им почту на новый адрес, либо возвращаете ее отправителю с сообщением об ошибке. Функциональная возможность `redirect` позволяет реализовать более элегантное решение для возврата почты.

Если Джо Смит окончил университет `oldsite.edu` (регистрационное имя `smithj`) и перешел в организацию `newsite.com` (регистрационное имя `joe`), то, включив функциональную возможность `redirect` с помощью примитива

```
FEATURE(`redirect')
```

и добавив строку

```
smithj: joe@newsite.com.REDIRECT
```

в файл **aliases** на сайте `oldsite.edu`, вы сможете возвращать почту, адресованную пользователю `smithj`, отправителю вместе с сообщением об ошибке, в котором указан новый адрес получателя `joe@newsite.com`. Само письмо автоматически не перенаправляется.

Функциональная возможность `always_add_domain`

Эта функциональная возможность задает полное определение всех почтовых адресов. Ее следует использовать постоянно.

Функциональная возможность `access_db`

С помощью этой функциональной возможности можно управлять ретрансляцией и другими правилами работы с почтой. Обычно данные, которые управляют этой функциональной возможностью, либо поступают от сервера LDAP, либо хранятся в текстовом файле **/etc/mail/access**. Во втором случае текстовый файл должен быть преобразован в индексированный формат с помощью команды **makemap**, как показано выше. Для использования простого файла в конфигурационном файле следует указать инструкцию `FEATURE(`access_db')`; для работы с сервером LDAP следует использовать инструкцию `FEATURE(`access_db', `LDAP')`.[13]

Полем ключа для доступа к базе данных является IP-адрес сети или имя домена, сопровождаемое необязательным дескриптором, например `Connect:`, `To:` или `From:`. Поле значения указывает, что делать с сообщением.

Наиболее широко используются такие значения, как `OK` для приема сообщений, `RELAY` для разрешения ретрансляции, `REJECT` для отказа с сообщением об ошибке общего характера или `ERROR:` *"код_ошибки и сообщение"* для отказа с сообщением о конкретной ошибке. С помощью других значений можно установить более точный контроль. Рассмотрим фрагмент из файла **/etc/mail/access**.

[13] В этом примитиве используется схема LDAP, определенная в файле **cf/sendmail.schema**; если вы хотите использовать другой файл схемы, укажите дополнительные аргументы в инструкции `FEATURE`.

```
localhost       RELAY
127.0.0.1       RELAY
192.168.1.1     RELAY
192.168.1.17    RELAY
66.77.123.1     OK
fax.com         OK
61              ERROR:"550 We don't accept mail from spammers"
67.106.63       ERROR:"550 We don't accept mail from spammers"
```

Функциональная возможность `virtusertable`

Эта функциональная возможность позволяет задавать псевдонимы доменов для входящей корреспонденции с помощью ассоциативного массива, хранящегося в файле **/etc/mail/virtusertable**. Она также позволяет размещать несколько виртуальных доменов на одной машине и использовать их для веб-хостинга. Поле ключа таблицы содержит либо адрес электронной почты (*user@host.domain*), либо спецификацию домена (*@domain*). Поле значения содержит локальный или внешний адрес электронной почты. Если ключом является домен, то значение может либо передавать поле `user` как переменную `%1`, либо направлять почту другому пользователю. Рассмотрим несколько примеров.

```
@appliedtrust.com       %1@atrust.com
unix@book.admin.com     sa-book-authors@atrust.com
linux@book.admin.com    sa-book-authors@atrust.com
webmaster@example.com   billy.q.zakowski@colorado.edu
info@testdomain.net     ausername@hotmail.com
```

Все ключи в левой части отображения данных должны быть перечислены в файле **cw**, т.е. **/etc/mail/local-host-names**, или находиться в списке `VIRTUSER_DOMAIN`. Если это условие не выполняется, то программа **sendmail** не сможет принять локальную почту и попытается найти адресата в Интернете. Однако записи DNS `MX` вернут программу **sendmail** назад на тот же самый сервер, и в результате рикошета вы получите сообщение "`local configuration error`" ("ошибка локальной конфигурации"). К сожалению, программа **sendmail** не может сообщить, что это сообщение на самом деле означает "ключа виртуального пользователя в файле **cw** нет".

Функциональная возможность `ldap_routing`

📖 Протокол LDAP описан в разделе 17.2.

Протокол LDAP (Lightweight Directory Access Protocol — облегченный протокол доступа к каталогам) может быть источником данных о псевдонимах или информации о маршрутизации почты, а также общих данных, хранящихся в таблицах. Файл **cf/README** содержит большой раздел о протоколе LDAP с множеством примеров.

Для того чтобы использовать сервер LDAP указанным выше способом, необходимо скомпилировать программу **sendmail** так, чтобы она включала поддержку протокола LDAP. Для этого в файл **.mc** следует добавить такие строки.

```
define(`confLDAP_DEFAULT_SPEC', `-h server -b searchbase')
FEATURE(`ldap_routing')
LDAPROUTE_DOMAIN(`my_domain')
```

Эти строки означают, что вы хотите использовать базу данных протокола LDAP для маршрутизации входящей почты, адресованной указанному домену. Параметр `LDAP_DEFAULT_SPEC` идентифицирует сервер LDAP и базовое имя LDAP для поисков.

Протокол LDAP использует порт 389, если вы не указали явным образом другой порт, добавив в инструкцию `define` параметр `-p` *порт_ldap*.

Программа **sendmail** использует значения двух дескрипторов в базе данных LDAP.

- `mailLocalAddress` — для адресата входящей почты.
- `mailRoutingAddress` — для пункта назначения, в который должна быть доставлена почта.

Кроме того, программа **sendmail** использует дескриптор `mailHost`, который направляет почту обработчику, заданному в поле `MX` для указанного хоста. Адресом получателя остается значение дескриптора `mailRoutingAddress`.

Записи базы данных LDAP позволяют использовать шаблонный символ `@`*domain*, который перенаправляет почту, адресованную кому-либо в указанном домене (как и параметр `virtusertable`).

По умолчанию при пересылке почты, адресованной получателю `user@host1.mydomain`, сначала будет выполняться поиск по ключу `user@host1.mydomain`. Если такой ключ не найден, программа **sendmail** может попытаться выполнить поиск по домену `@host1.mydomain`, но не по `user@mydomain`. Включив строку

```
LDAPROUTE_EQUIVALENT(`host1.mydomain')
```

можно также выполнить поиск по ключу `user@mydomain` и `@mydomain`. Эта функциональная возможность допускает использование единой базы данных для маршрутизации почты на сложных сайтах. Вы также можете использовать записи для разделов `LDAPROUTE_EQUIVALENT` из файла, что делает это свойство еще более полезным.

Синтаксис этой инструкции выглядит следующим образом.

```
LDAPROUTE_EQUIVALENT_FILE(`filename')
```

Кроме того, дополнительные аргументы свойства `ldap_routing` позволяют более подробно описать схему LDAP и точнее управлять обработкой имен адресатов, для которых предусмотрен раздел *+detail*. Как всегда, более подробную информацию следует искать в файле **cf/README**.

Маскирующие функциональные возможности

Адрес электронной почты обычно состоит из имени пользователя, хоста и домена, но многие организации не хотят демонстрировать имена своих хостов в Интернете. Макрос **MASQUERADE_AS** позволяет скрыть все машины за единственной сущностью. Вся появляющаяся почта будет исходить от указанного компьютера или домена. Это очень удобно для обычных пользователей, но системные пользователи, такие как пользователь `root`, не должны участвовать в маскировке.

Например, последовательность инструкций

```
MASQUERADE_AS(`atrust.com')
EXPOSED_USER(`root')
EXPOSED_USER(`Mailer-Daemon')
```

"проштамповывает" почту так, будто она исходит от адреса `user@atrust.com`, если она не была послана пользователем `root` или почтовой системой; в этих случаях почта должна иметь истинный адрес отправителя. Инструкция `MASQUERADE_AS` — это верхушка айсберга, за которой скрываются десятки вариантов и исключений. Функциональные возможности `allmasquerade` и `masquerade_envelope` (в сочетании с `MASQUERADE_AS`) скрывают требуемую часть локальной информации. Детали см. в файле **cf/README**.

Макросы MAIL_HUB и SMART_HOST

Маскировка изменяет заголовки и, возможно, конверт, чтобы вся почта выглядела исходящей от одного компьютера или домена. Однако большинство организаций *действительно* желают, чтобы вся почта исходила от одного компьютера или поступала на один компьютер, чтобы можно было контролировать поток вирусов, спама и секретов компании. Этот контроль можно обеспечить с помощью сочетания записей MX в системе DNS, макроса **MAIL_HUB** для входящей почты и макроса **SMART_HOST** для исходящей почты.

🕮 Более подробная информация о записях DNS MX приведена в разделе 16.5.

Например, в структуре электронной почты записи MX могут направлять входящую электронную почту из Интернета транспортному агенту в демилитаризованной зоне. После проверки того факта, что полученная почта не содержит вирусы и спам и была направлена правильному локальному пользователю, почту можно передать с помощью инструкции define внутреннему транспортному агенту, выполняющему маршрутизацию.

```
define(`MAIL_HUB', `smtp:routingMTA.mydomain')
```

🕮 Функциональная возможность nullclient рассматривается в следующем разделе.

Аналогично клиентские компьютеры могут направлять свою почту компьютеру SMART_HOST, назначенному в конфигурации с помощью свойства nullclient. Компьютер SMART_HOST затем может профильтровать вирусы и спам, чтобы почта из вашей сети не просочилась в Интернет.

Синтаксис инструкции SMART_HOST аналогичен синтаксису инструкции MAIL_HUB, причем агентом доставки тоже является relay.

```
define(`SMART_HOST', `smtp:outgoingMTA.mydomain')
```

Ту же самую машину можно использовать как сервер для входящей и исходящей почты. Инструкции SMART_HOST и MAIL_HUB должны разрешать ретрансляцию почты: первая — от клиентов внутри вашего домена, а вторая — от транспортного агента в зоне DMZ.

Конфигурация клиентов

Большинство ваших компьютеров необходимо конфигурировать как клиентов, желающих посылать исходящую почту, генерируемую пользователями, и не получать почту вообще. Одна из функциональных возможностей программы **sendmail** под названием nullclient предназначена именно для таких ситуаций. Она создает конфигурационный файл, который перенаправляет всю почту на центральный концентратор по протоколу SMTP. Содержимое всего конфигурационного файла, следующее за строками с макросами VERSIONID и OSTYPE, сводится к таким строкам.

```
FEATURE(`nocanonify')
FEATURE(`nullclient', `mailserver')
EXPOSED_USER(`root')
```

Здесь *mailserver* — имя центрального концентратора почты. Параметр nocanonify сообщает программе **sendmail**, что она не должна выполнять поиск в системе DNS или перезаписывать адреса с полностью определенными именами доменов. Всю эту работу будет выполнять почтовый сервер. Этот параметр похож на параметр SMART_HOST и предполагает, что клиент применяет к почтовому серверу инструкцию MASQUERADE_AS. Инструкция EXPOSED_USER исключает пользователя root из маскировки и открывает возможности для отладки.

Компьютер *mailserver* должен разрешать ретрансляцию почты от своих нулевых клиентов. Это разрешение предоставляется с помощью примитива access_db, описанного выше. Нулевой клиент должен иметь соответствующую запись MX, указывающую на компьютер *mailserver*, а также содержаться в файле **cw** на компьютере *mailserver* (который обычно называется **/etc/mail/local-host-names**). Эти настройки позволяют компьютеру *mailserver* принимать почту для клиента.

Если пользовательские агенты на клиентской машине будут использовать только порт 587 для отправки почтовых сообщений, то программа **sendmail** должна запускаться как агент доставки (MSA, без флага **-bd**). В противном случае вы должны запустить программу **sendmail** в режиме демона (**-bd**), задав при этом параметр конфигурации DAEMON_OPTIONS так, чтобы программа **sendmail** обрабатывала только соединения, поступающие через интерфейс замыкания (loopback interface).

Параметры конфигурации препроцессора m4

Параметры файла конфигурации задаются с помощью команды define, относящейся к препроцессору **m4**. Полный список параметров, доступных как переменные препроцессора **m4** (вместе с их значениями, заданными по умолчанию), приведен в файле **cf/README**.

Значения по умолчанию подходят для типичной организации, не страдающей от паранойи и не слишком обеспокоенной вопросами производительности. Параметры, заданные по умолчанию, предназначены для защиты от спама: они отключают ретрансляцию и требуют, чтобы адреса были полностью определенными, а домены отправителей соответствовали их IP-адресам. Если ваш почтовый концентратор слишком загружен и обслуживает слишком много списков рассылки, может потребоваться более тонкая настройка.

Параметры, которые приходится настраивать чаще всего, перечислены в табл. 18.10 (это около 10% от почти 175 параметров конфигурации). В скобках указаны их значения по умолчанию. В целях экономии места префикс conf в названиях параметров опущен. Например, параметр FALLBACK_MX в действительности называется confFALLBACK_MX. Таблица разделена на части, в соответствии с тем, для каких целей применяются те или иные параметры: общих, связанных с ресурсами, производительностью, безопасностью, спамом и т.д. Некоторые параметры попадают сразу в несколько категорий, однако мы указали их только в одном месте.

Таблица 18.10. Основные параметры конфигурации

	Параметр	Описание (значение по умолчанию)
Ресурсы	MAX_DAEMON_CHILDREN	Максимальное количество порожденных процессов[a] (ограничений нет)
	MAX_MESSAGE_SIZE	Максимальный размер в байтах одного сообщения (ограничений нет)
	MIN_FREE_BLOCKS	Минимальное пространство в файловой системе для хранения поступающей почты (100)
	TO_*имя*	Время тайм-аута для разных ситуаций (варьируется)
Производительность	DELAY_LA	Показатель средней загруженности, при котором доставка почты замедляется (0 — нет ограничений)
	FAST_SPLIT	Подавляет поиск записей MX, так как получатели отсортированы и распределены по очередям (1 — включено)

Окончание табл. 18.10

	Параметр	Описание (значение по умолчанию)
	MCI_CACHE_SIZE	Количество открытых кешированных TCP-соединений (2)
	MCI_CACHE_TIMEOUT	Время, в течение которого кешированные соединения остаются открытыми (5m)
	MIN_QUEUE_AGE	Минимальный период времени, в течение которого сообщение должно находиться в очереди; позволяет улучшить обработку очереди на перегруженном компьютере (0)
	QUEUE_LA	Показатель средней загруженности, при котором сообщения помещаются в очередь, а не отправляются немедленно (8**число_процессоров*)
	REFUSE_LA	Показатель средней загруженности, при котором почта перестает приниматься (12**число_процессоров*)
Безопасность/спам	AUTH_MECHANISMS	Список механизмов аутентификации по протоколу SMTP для библиотеки Cyrus SASL [6]
	CONNECTION_RATE_THROTTLE	Предотвращает атаки типа "отказ от обслуживания", ограничивая количество соединений в секунду, при котором разрешается устанавливать почтовое соединение (ограничений нет)
	DONT_BLAME_SENDMAIL	Отменяет ряд функций программы **sendmail**, связанных с безопасностью и проверкой файлов; не изменяйте значение этой опции без особой необходимости (safe)
	MAX_MIME_HEADER_LENGTH	Максимальный размер заголовков MIME (ограничений нет) [в]
	MAX_RCPTS_PER_MESSAGE	Препятствует отправке спама; обработка избыточного числа получателей приостанавливается и генерируется временное сообщение об ошибке (число получателей не ограничено)
	PRIVACY_FLAGS	Налагает ограничения на информацию, выдаваемую по протоколу SMTP (authwarnings)
Разное	DOUBLE_BOUNCE_ADDRESS	Перехватывает множество спама; некоторые организации используют каталог **/dev/null**, но это может привести к серьезным проблемам (почтовый сервер)
	LDAP_DEFAULT_SPEC	Спецификация базы данных LDAP, включая имя компьютера, на котором запущен сервер, и номер порта (не определена)

[а] Точнее говоря, это максимальное количество дочерних процессов, которые можно выполнить одновременно. При достижении предельного значения программа **sendmail** отказывается принимать запросы. Эта опция позволяет предотвратить атаки типа "отказ от обслуживания".

[б] Значение по умолчанию таково: "EXTERNAL GSSAPI KERBEROS_V4 DIGEST-MD5 CRAM-MD5". Не добавляйте "PLAIN LOGIN", поскольку пароль передается открытым текстом. Если соединение также не защищено с помощью криптографического протокола SSL, то изнутри все может выглядеть благополучно, а со стороны Интернета — нет.

[в] Эта опция защищает буфер пользовательского агента от переполнения. Рекомендуемое значение — "256/128", что означает "256 байт на заголовок и 128 байт на каждый параметр заголовка".

Средства программы sendmail для борьбы со спамом

Программа **sendmail** имеет разнообразные возможности для борьбы со спамом и вирусами.

- *Правила управления открытой ретрансляцией*, под которой понимается использование почтового сервера для отправки сообщения одним сторонним пользователем другому, тоже стороннему. Спамеры часто прибегают к ретрансляции, пытаясь таким способом замаскировать действительного адресата сообщения и тем самым избежать обнаружения со стороны своих собственных провайдеров. Это также позволяет им перекладывать финансовую и вычислительную нагрузку на чужие плечи.
- *База данных доступа*, позволяющая фильтровать почту согласно адресам, содержащимся в базе данных. Это напоминает применение брандмауэра для электронной почты.
- *Черные списки* открытых ретрансляторов и известных спамеров, проверяемые программой **sendmail**.
- *Дроссели, которые могут замедлять прием почты при обнаружении неправильного поведения.*
- *Проверка заголовков* и *фильтрация входящей почты* посредством универсальной библиотеки фильтров **libmilter**. Эта библиотека позволяет сканировать заголовки и содержимое сообщений и отбрасывать сообщения, удовлетворяющие определенным критериям (см. `milter.org`).

Ретрансляция

Программа **sendmail** получает входящую почту, просматривает адреса конверта, принимает решение о том, куда переслать сообщение, после чего отправляет его в соответствующий пункт назначения. Это может быть не только локальный получатель, но и другой транспортный агент, расположенный далее по цепи доставки. Когда входящее сообщение не имеет локального получателя, транспортный агент, обрабатывающий его, играет роль *ретранслятора* (relay).

Только хостам, помеченным ключевым словом `RELAY` в базе данных доступа (см. выше), а также хостам, перечисленным в файле **/etc/mail/relay-domains**, разрешается ретранслировать почту. Впрочем, некоторые типы ретрансляции необходимы. Как определить, какое сообщение следует ретранслировать, а какое — отклонить? Ретрансляция нужна лишь в двух случаях.

- *Когда транспортный агент выступает в качестве шлюза для хостов, недоступных никаким другим способом.* Это могут быть периодически выключаемые компьютеры (PPP-хосты, персональные компьютеры, работающие под управлением Windows) и виртуальные хосты. В этом случае все получатели, которым требуется переслать сообщение, должны находиться в одном домене.
- *Когда транспортный агент является сервером исходящей почты для других хостов.* В этом случае имена и IP-адреса всех хостов-отправителей являются локальными (или, по крайней мере, заданы явно).
- *Когда вы согласны служить получателем для резервных записей MX для другой организации.*

Любые другие случаи ретрансляции, скорее всего, свидетельствуют о плохой конфигурации сервера (исключение можно сделать в отношении мобильных пользователей).

Чтобы избежать ретрансляции в первом из упомянутых вариантов, организуйте прием почты на центральном сервере (с использованием протокола POP или IMAP для клиентского доступа). Второй вариант должен быть всегда допустим, но только для локальных хостов. Лучше проверять IP-адреса, чем имена компьютеров, так как последние проще подделать; впрочем, программа **sendmail** следит за этим. В третьем случае вы можете указать в своей базе данных другую организацию и разрешить ретрансляцию для блоков ее IP-адресов.

Несмотря на то что в программе **sendmail** ретрансляция запрещена по умолчанию, с помощью специальных средств ее можно разрешить либо в полном объеме, либо в ограниченном и контролируемом режиме. Эти средства перечислены ниже. Их можно использовать, но мы рекомендуем проявлять максимум осторожности и не злоупотреблять ими. Безопаснее всего организовать ограниченную ретрансляцию с помощью средства access_db, рассматриваемого в следующем подразделе.

- FEATURE(`relay_entire_domain') — разрешает ретрансляцию для хостов текущего домена.
- RELAY_DOMAIN(`*домен*,...') — дополнительно разрешает ретрансляцию для указанных доменов.
- RELAY_DOMAIN_FILE(`*имя_файла*') — то же самое, но список доменов берется из файла.
- FEATURE(`relay_hosts_only') — влияет на работу макроса RELAY_DOMAIN и средства access_db.

Если благодаря макроконстантам SMART_HOST и MAIL_HUB почта направляется на конкретный почтовый сервер, придется сделать исключение. Этот сервер должен быть настроен на ретрансляцию всей почты, поступающей от локальных компьютеров.

```
FEATURE(`relay_entire_domain')
```

Если вы рассматриваете возможность разрешения ретрансляции в какой-то форме, обратитесь к документации программы **sendmail** в файле **cf/README**, чтобы ненароком не стать подручным спамеров. Если решите включить ретрансляцию, то обратитесь к одному из специализированных сайтов с просьбой проверить, что вы случайно не создали открытую ретрансляцию, в частности можно обратиться на сайт spamhelp.org.

Использование черных списков

Если необходимо блокировать почту каких-либо локальных пользователей или хостов, воспользуйтесь средством

```
FEATURE(`blacklist_recipients')
```

Ему в базе данных доступа соответствуют записи следующих типов.

```
To:nobody@                    ERROR:550 Mailbox disabled for this user
To:printer.mydomain           ERROR:550 This host does not accept mail
To:user@host.mydomain         ERROR:550 Mailbox disabled for this user
```

Эти записи блокируют почту, приходящую на адрес пользователя **nobody** (на любом хосте), а также адресованную сетевому принтеру и указанному пользователю конкретного компьютера. Дескриптор To: разрешает этим пользователям посылать сообщения (подобной возможностью обладают некоторые принтеры), но не принимать их.

Для того чтобы подключить черные DNS-списки для входящей почты, используется параметр dnsbl.

```
FEATURE(`dnsbl', `zen.spamhaus.org')
```

Эта функциональная возможность заставляет программу **sendmail** отклонять почту, приходящую от хоста, IP-адрес которого указан в списке известных спамеров (SBL, XBL и PBL), поддерживаемых сайтом `spamhaus.org`. В других списках указаны адреса спамеров, работающих по коммутируемым линиям, и хостов, поддерживающих открытую ретрансляцию. Черные списки распространяются через службу DNS с помощью особого приема.

Средству `dnsbl` можно передать третий аргумент, который задает требуемое сообщение об ошибке. Если третий аргумент пропущен, будет выдано фиксированное сообщение (из базы данных DNS, содержащей проверяемые записи).

Функциональную возможность `dnsbl` можно включать несколько раз, чтобы проверять разные списки злоумышленников.

Дроссели, скорость и ограничения на количество соединений

В табл. 18.11 перечислены несколько параметров программы **sendmail**, которые могут замедлить обработку почты, если поведение клиентов покажется подозрительным.

Таблица 18.11. Основные параметры конфигурации

Примитив	Описание
`BAD_RCPT_THROTTLE`	Замедляет коллекционирование адресов спамерами
`MAX_RCPTS_PER_MESSAGE`	Откладывает доставку, если сообщение имеет слишком много получателей
`ratecontrol`	Ограничивает скорость входящих соединений
`conncontrol`	Ограничивает количество одновременных соединений
`greet_pause`	Задерживает ответ `HELO`, требует точного подчинения протоколу SMTP

После того как счетчик `no-such-login` достигает предельного значения, заданного параметром `BAD_RCPT_THROTTLE`, программа **sendmail** выполняет односекундную задержку после получения каждой команды `RCPT`, замедляя сбор адресов спамерской программой. Для того чтобы задать это предельное значение равным 3, используйте следующую команду.

```
define(`confBAD_RCPT_THROTTLE', `3')
```

Параметр `MAX_RCPTS_PER_MESSAGE` заставляет отправителя поставить дополнительных получателей в конец очереди. Это простая форма серого списка для сообщений, имеющих подозрительно много получателей.

Параметры `ratecontrol` и `conncontrol` позволяют установить для каждого компьютера или каждой сети скорость приема входящих соединений и количество одновременно устанавливаемых соединений соответственно. Оба параметра можно задать в файле **/etc/mail/access**, чтобы указать пределы и домены, к которым они относятся. Первый параметр сопровождается дескриптором `ClientRate:` в поле ключа, а второй — дескриптором `ClientConn:`. Для того чтобы включить режим контроля скорости, вставьте следующие строки в свой файл с расширением **.mc**.[14]

```
FEATURE(`ratecontrol', `nodelay',`terminate')
FEATURE(`conncontrol', `nodelay',`terminate')
```

Затем добавьте в свой файл **/etc/mail/access** список компьютеров или сетей, подлежащих контролю, а также соответствующие предельные значения. Например, строки

```
ClientRate:192.168.6.17  2
ClientRate:170.65.3.4    10
```

[14] Наряду с инструкцией `FEATURE(`access_db')`.

ограничивают хосты 192.168.6.17 и 170.65.3.4 двумя новыми соединениями в минуту и десятью новыми соединениями в минуту соответственно. Строки

```
ClientConn:192.168.2.8   2
ClientConn:175.14.4.1    7
ClientConn:              10
```

устанавливают следующие пределы: два соединения для хоста 192.168.2.8, семь — для хоста 175.14.4.1 и десять — для одновременных соединений со всеми другими хостами.

Еще одним интересным параметром является `greet_pause`. Когда удаленный транспортный агент соединяется с вашим сервером **sendmail**, протокол SMTP заставляет его подождать и поприветствовать ваш сервер, прежде чем начать обмен сообщениями. Однако программы рассылки спама обычно мгновенно выдают команду `EHLO/HELO`. Это поведение частично объясняется плохой реализацией протокола SMTP в инструментах для рассылки спама, но это может экономить время для спамера. Как бы то ни было, такое поведение является подозрительным и называется *навязыванием* ("slamming").

Параметр `greet_pause` заставляет программу **sendmail** подождать указанный период времени в начале соединения, прежде чем поздороваться с новообретенным другом. Если удаленный транспортный агент не останавливается, чтобы правильно поздороваться и обменяться командами `EHLO` или `HELO` в назначенный момент времени для знакомства, программа **sendmail** регистрирует ошибку и отказывается от выполнения последующих команд, поступающих от удаленного транспортного агента.

Вы можете задать паузу для приветствия с помощью следующей записи в файле с расширением **.mc**.

```
FEATURE(`greet_pause', `700')
```

Эта строка устанавливает задержку в 700 миллисекунд в начале каждого нового соединения. Вы можете устанавливать задержки для хостов или сетей с помощью префикса `GreetPause:` в базе данных доступа, но большинство сайтов для этого использует общее значение для данного параметра.

Безопасность и программа sendmail

С началом лавинообразного роста Интернета программы наподобие **sendmail**, принимающие произвольные входные данные и направляющие их локальным пользователям, в файлы или в интерпретаторы команд, превратились в излюбленные мишени для хакеров. В настоящее время программа **sendmail**, наряду со службой DNS и даже протоколом IP, начала "обзаводиться" встроенными средствами аутентификации и шифрования, позволяющими решать основные проблемы безопасности.

Программа **sendmail** поддерживает систему SMTP-аутентификации и шифрование по протоколу TSL (Transport Layer Security — безопасный протокол транспортного уровня), ранее известному как протокол SSL (Secure Sockets Layer — протокол защищенных сокетов). Протокол TSL содержит шесть новых конфигурационных параметров, связанных с файлами сертификатов и ключей. Кроме того, в базе данных доступа появились новые операции сравнения, связанные с аутентификацией.

В этом разделе мы опишем модель безопасности программы **sendmail** и систему для защиты конфиденциальности. Затем кратко рассмотрим протоколы TLS и SASL (Simple Authentication and Security Layer — простой протокол аутентификации и защиты) и их применение в сочетании с программой **sendmail**.

Прежде чем довериться содержимому, скажем, файлов **.forward** или **aliases**, программа **sendmail** тщательно проверяет права доступа к ним. Как правило, подобные

меры предосторожности действительно необходимы, но иногда все же требуется их ослабить. С этой целью в программе появился параметр DontBlameSendmail, название которого ("не осуждай **sendmail**") подсказывает системным администраторам, что после изменения опции действия программы станут небезопасными.

У параметра DontBlameSendmail очень много возможных значений (по последним подсчетам, 55). По умолчанию он равен safe. Полный список значений можно найти в файле **doc/op/op.ps** дистрибутива **sendmail** или в книге издательства O'Reilly, посвященной программе **sendmail**. Впрочем, лучше всего оставить этот параметр заданным по умолчанию.

Владельцы файлов

Программа **sendmail** различает трех специальных пользователей: DefaultUser, RunAsUser и TrustedUser.

По умолчанию все агенты доставки работают от имени пользователя DefaultUser, если не установлены специальные флаги. При наличии в файле **/etc/passwd** учетной записи **mailnull**, **sendmail** или **daemon** именно эта запись будет соответствовать пользователю DefaultUser. В противном случае пользователю будут соответствовать значения UID и GID, равные 1. Мы рекомендуем применять учетную запись **mailnull**. Добавьте ее в файл **/etc/passwd** со звездочкой в поле пароля, с пустым полем идентификатора команд, с пустым полем начального каталога и группой **mailnull**. Понадобится также добавить запись **mailnull** в файл **/etc/group**. Этой учетной записи не должны принадлежать никакие файлы. Если программа **sendmail** не выполняется с правами суперпользователя, для агентов доставки должен быть установлен бит setuid.

Если задан пользователь RunAsUser, программа **sendmail** работает от его имени, игнорируя пользователя DefaultUser. Если программа запускается с установленным битом setgid (идентификатор группы меняется на **smmsp**), то агент передачи почты (программа **sendmail**) передает транспортному агенту (другой экземпляр **sendmail**) сообщения по протоколу SMTP. Транспортный агент **sendmail** не имеет установленного бита setuid, но вызывается с правами корневого пользователя из сценариев запуска системы.

Пользователю RunAsUser соответствует значение UID, которое программа **sendmail** использует после открытия сокета, подключенного к порту 25. Привилегированные порты (их номера не превышают 1024) могут открываться только суперпользователем, следовательно, программа **sendmail** первоначально должна работать от имени пользователя root. Тем не менее по завершении указанной операции программа может взять себе другое значение UID. Это уменьшает вероятность возможных злоупотреблений, если программа **sendmail** запускается обманным путем. Не применяйте средство RunAsUser на компьютерах, где есть другие пользовательские учетные записи и службы; оно предназначено для брандмауэров и бастионных хостов. [15]

По умолчанию программа не меняет идентификационную информацию и продолжает выполняться от имени пользователя root. Если же учетная запись RunAsUser не эквивалентна суперпользователю, придется многое изменить. Пользователю RunAsUser должна принадлежать очередь почтовых сообщений, он должен иметь возможность читать все хеш-файлы и файлы баз данных, запускать нужные программы и т.д. На поиск всех файлов и каталогов, владелец которых должен быть изменен, может уйти несколько часов.

[15] Бастионные хосты — это специально укрепленные хосты, предназначенные для отражения атаки и размещенные в зоне DMZ или за пределами брандмауэра.

Пользователь TrustedUser может владеть файлами баз данных и псевдонимов. Ему разрешается запускать демон и перестраивать файл **aliases**. Этот пользователь необходим для поддержки графических надстроек к программе **sendmail**, которые вынуждены предоставлять ограниченный административный контроль определенным пользователям. Обязательно контролируйте учетную запись, соответствующую пользователю TrustedUser, поскольку через нее можно легко получить права суперпользователя. Пользователь TrustedUser не соответствует классу TRUSTED_USERS, который определяет, кто имеет право менять заголовок "From" сообщений.[16]

Права доступа

Для безопасности программы **sendmail** большое значение имеют права доступа к определенным файлам и каталогам. Установки, приведенные в табл. 18.12, считаются безопасными.

Таблица 18.12. Владельцы и права доступа для каталогов, связанных с программой sendmail

Каталог	Владелец	Режим	Назначение/содержимое
/var/spool/clientmqueue	smmsp:smmsp	770	Почтовая очередь для начальной подачи почты
/var/spool/mqueue	RunAsUser	700	Очередь почтовых сообщений
/, /var, /var/spool	root	755	Родительские каталоги подкаталога **mqueue**
/etc/mail/*	TrustedUser	644	Хеш-таблицы, конфигурационный файл, файлы псевдонимов
/etc/mail	TrustedUser	755	Родительский каталог для хеш-файлов
/etc	root	755	Путь к каталогу **mail**

Программа **sendmail** больше не читает файл **.forward**, если число ссылок на него больше единицы, а путь к каталогу небезопасен (права доступа ослаблены). В такую ловушку однажды попалась Эви Немет, когда ее файл **.forward**, представляющий собой жесткую ссылку либо на файл **.forward.to.boulder**, либо на файл **.forward.to.sandiego**, перестал поддерживать пересылку почты от небольшого хоста, с которого ей иногда приходили сообщения. Прошли месяцы, прежде чем Эви поняла, что это происходит по ее вине и фраза "Я никогда не получала от вас почту" не является оправданием.

Отключить многие ограничения на доступ к файлам можно с помощью опции DontBlameSendmail. Но не забывайте о безопасности.

Безопасная пересылка почты в файлы и программы

В качестве агента программной доставки мы рекомендуем применять утилиту **smrsh**, а не **/bin/sh**; агентом локальной доставки лучше назначить утилиту **mail.local**, а не **/bin/mail**. Обе утилиты входят в дистрибутив **sendmail**. Для их активации необходимо добавить в **mc**-файл следующие директивы.

```
FEATURE(`smrsh', `путь-к-smrsh')
FEATURE(`local_lmtp', `путь-к-mail.local')
```

Если пути не указаны, по умолчанию принимается каталог **/usr/libexec**. Можно воспользоваться опцией confEBINDIR, чтобы изменить стандартный каталог исполняемых файлов. Найти агентов доставки вам поможет табл. 18.13.

[16]Средство TRUSTED_USERS обычно применяется для поддержки программ, работающих со списками рассылки.

Таблица 18.13. Местоположение ограниченных агентов доставки программы sendmail

Операционная система	smrsh	mail.local	sm.bin
Ubuntu	/usr/lib/sm.bin	/usr/lib/sm.bin	/usr/adm
Debian	/usr/lib/sm.bin	/usr/lib/sm.bin	/usr/adm
Red Hat	/usr/sbin	-	/etc/smrsh
CentOS	/usr/sbin	-	/etc/smrsh
FreeBSD	/usr/libexec	/usr/libexec	/usr/adm

Утилита **smrsh** — это ограниченная оболочка, запускающая программы, находящиеся только в одном каталоге (по умолчанию — **/usr/adm/sm.bin**). Она игнорирует заданные пользователями имена каталогов и деревьев, выполняя поиск требуемых команд в собственном безопасном каталоге. Утилита **smrsh** также блокирует некоторые метасимволы интерпретатора команд, в частности '<' — оператор переадресации входного потока. В каталоге **sm.bin** допускается наличие символических ссылок, поэтому создавать копии программ не потребуется. Программа **vacation** — подходящий кандидат для включения в каталог **sm.bin**. Не размещайте в нем программу **procmail**; она небезопасна.

Приведем несколько примеров команд вместе с их возможными интерпретациями утилитой **smrsh**.

```
vacation eric                 # Выполняется команда /usr/adm/sm.bin/vacation
cat /etc/password             # Отклоняется, так как cat нет в каталоге sm.bin
vacation eric < /etc/passwd   # Отклоняется, так как оператор < не разрешен
```

Опция `SafeFileEnvironment` программы **sendmail** контролирует, куда можно записывать сообщения, если в файле **aliases** или **.forward** задано перенаправление почты в файл. Это опция заставляет программу осуществить системный вызов **chroot**, вследствие чего корневым каталогом файловой системы станет не **/**, а **/safe** или любой другой указанный в опции каталог. Например, если в файле псевдонимов задается перенаправление почты в файл **/etc/passwd**, то на самом деле сообщения будут записываться в файл **/safe/etc/passwd**.

Опция `SafeFileEnvironment` защищает также файлы устройств, каталоги и другие специальные файлы, позволяя записывать почту только в обычные файлы. Это полезно не только в плане улучшения безопасности, но и с точки зрения борьбы с ошибками пользователей. В некоторых системах значение этой опции задается равным **/home**, что открывает доступ к начальным каталогам пользователей, но оставляет системные файлы вне досягаемости.

Агенты доставки тоже могут запускаться в виртуальном каталоге.

Опции безопасности

В программе **sendmail** есть опции безопасности, которые определяют следующее:

- какие сведения о системе можно узнать из внешнего мира по протоколу SMTP;
- что требуется от хоста на противоположном конце SMTP-соединения;
- могут ли пользователи просматривать или обрабатывать очередь почтовых сообщений.

В табл. 18.14 приведены текущие значения опций безопасности. Самую последнюю информацию можно узнать в файле **doc/op/op.ps** дистрибутива.

Таблица 18.14. Значения переменной PrivacyOption

Значение	Интерпретация
authwarnings	К сообщениям добавляется заголовок Authentication-Warning (это установка по умолчанию)
goaway	Отменяются все SMTP-запросы состояния (EXPN, VRFY и т.д.)
needexpnhelo	Не допускается раскрытие адреса (команда EXPN) без команды HELO
needmailhelo	Требуется SMTP-команда HELO (идентифицирует удаленный хост)
needvrfyhelo	Не допускается проверка адреса (команда VRFY) без команды HELO
nobodyreturn	При выдаче кода состояния доставки не возвращается тело сообщения
noetrn[a]	Не допускается асинхронная обработка очереди
noexpn	Не допускается SMTP-команда EXPN
noreceipts	Запрещается выдача кодов состояния доставки
noverb[6]	Не допускается "многословный" режим команды EXPN
novrfy	Не допускается SMTP-команда VRFY
public	Проверка конфиденциальности/безопасности не осуществляется
restrictexpand	Ограничивается объем информации, выдаваемой при наличии флагов **-bv** и **-v**[в]
restrictmailq	Только пользователи группы, которой принадлежит каталог **mqueue**, могут просматривать очередь сообщений
restrictqrun	Только владелец каталога **mqueue** может обрабатывать очередь сообщений

[a] ETRN — это команда протокола ESMTP, которая используется хостами с коммутируемым доступом. Она запрашивает обработку очереди только для сообщений данного хоста.

[6] В "многословном" режиме команда EXPN отслеживает переадресацию в файле **.forward** и выдает дополнительную информацию о местонахождении пользовательской почты. Необходимо включать опцию noverb или, еще лучше, noexpn на любом компьютере, имеющем выход во внешний мир.

[в] Если только программа не выполняется от имени пользователя **root** или **TrustedUser**.

Мы рекомендуем сделать в **mc**-файле "консервативные" установки.

```
define(`confPRIVACY_FLAGS', ``goaway, authwarnings,
       restrictmailq, restrictqrun'')
```

По умолчанию установлен лишь параметр authwarnings. Обратите внимание на двойные кавычки: некоторые версии препроцессора **m4** требуют этого для предотвращения ненужной интерпретации запятых в списке значений.

Выполнение программы sendmail в виртуальном каталоге (для настоящих параноиков)

Те, кого беспокоит влияние программы **sendmail** на файловую систему, могут запускать ее в виртуальной среде. Создайте в этом каталоге минимальную файловую систему, включив в нее файл **/dev/null**, важные файлы каталога **/etc** (**passwd**, **group**, **resolv.conf**, **sendmail.cf**, таблицы баз данных, **mail/***), необходимые программе **sendmail** совместно используемые библиотеки, исполняемый файл **sendmail**, каталог очереди почтовых сообщений и журнальные файлы. Вероятно, вы захотите поэкспери-

ментировать со списком. Для запуска программы **sendmail** в виртуальном окружении используйте команду **chroot**.

```
# sudo chroot /jail /usr/sbin/sendmail -bd -q30m
```

Отражение атак типа "отказ от обслуживания"

Атаки типа "отказ от обслуживания" невозможно предотвратить, потому что нельзя заранее предсказать, является ли сообщение оружием атаки или нет. Злоумышленники могут поступать по-разному, например переполнять SMTP-порт ложными запросами на подключение, "забивать" разделы диска гигантскими сообщениями, забивать выходные соединения, бомбардировать систему почтовыми сообщениями и т.д. В программе **sendmail** имеются конфигурационные параметры, позволяющие ограничить влияние этих атак на систему, но в результате может пострадать и легитимная почта. Определенную помощь может оказать и новая библиотека функций фильтрации почты.

Параметр `MaxDaemonChildren` ограничивает число процессов **sendmail**. Он не позволяет программе захватить все вычислительные ресурсы системы, но зато дает возможность хакерам очень легко одолеть службу SMTP. Опция `MaxMessageSize` защищает каталог очереди от переполнения, но если задать значение опции слишком маленьким, жертвами могут оказаться обычные сообщения. (Следует уведомить пользователей о существующем пределе на размер сообщения, чтобы они не удивлялись, если письмо вдруг вернется обратно. Этот предел обычно устанавливается достаточно большим.) Параметр `ConnectionRateThrottle` задает предельно допустимое число соединений в секунду. Это может привести к небольшому замедлению работы программы **sendmail**. Наконец, опция `MaxRcptsPerMessage` определяет максимальное число получателей сообщения.

Программа **sendmail** всегда умела отказываться от приема сообщений (опция `REFUSE_LA`) и ставить сообщения в очередь (опция `QUEUE_LA`) в зависимости от уровня загруженности системы. Дополнительный параметр `DELAY_LA` позволяет выполнять обработку почты с меньшей скоростью.

Несмотря на все меры предосторожности, трудно предотвратить ситуацию, когда кто-то начинает бомбардировать систему письмами. Это может стать очень неприятной проблемой.

TLS: безопасносный протокол транспортного уровня

Еще одна система аутентификации и шифрования — TLS — определена в документе RFC3207. Она реализована в программе **sendmail** в виде расширения протокола SMTP под названием STARTTLS.

🕮 Протокол TLS описан в разделе 27.6.

Протокол TLS труднее настроить, так как требуется поддержка со стороны центра сертификации. Можно заплатить компании VeriSign, которая выдаст вам необходимые сертификаты (подписанные открытые ключи, идентифицирующие их предъявителя), или организовать собственный центр сертификации. В процессе ретрансляции почты и приема запросов от других хостов аутентификация осуществляется не на основе имени хоста или его IP-адреса, а путем обмена ключами. Записи вида

```
TLS_Srv:secure.example.com ENCR:112
TLS_Clt:laptop.example.com PERM+VERIFY:112
```

в базе данных доступа указывают на то, что используется механизм STARTTLS. Почта, направляемая в домен `secure.example.com`, должна шифроваться как минимум 112-битовыми ключами. Почта, поступающая от хоста `secure.example.com`, будет приниматься лишь в том случае, если клиент пройдет процедуру аутентификации.

Хотя протокол STARTTLS обеспечивает надежное шифрование, обратите внимание, что его защита охватывает только текущий сеанс связи с транспортным агентом "следующего перехода" передачи почты. Как только сообщение поступит в следующий переход, оно может быть перенаправлено другому транспортному агенту, который не использует безопасный транспортный метод. Если вы контролируете все возможные транспортные агенты на своем пути, то сможете создать безопасную почтовую транспортную сеть. В противном случае вам придется полагаться на пакет шифрования для конечного пользователя (например, PGP/GPG) или централизованную службу шифрования электронной почты (см. раздел 18.5).

Грег Шапиро и Клаус Ассманн из компании Sendmail, Inc. накопили некоторый объем дополнительной документации (хотя и немного устаревшей), относящийся к системе безопасности и работе программы **sendmail** в веб. С ней можно ознакомиться по адресам `sendmail.org/~gshapiro` и `sendmail.org/~ca`. Особенно полезна вторая ссылка.

Тестирование и отладка программы sendmail

Конфигурация, основанная на применении препроцессора **m4**, в определенной степени тестируется автоматически. Низкоуровневая отладка, в основном, не требуется. Единственное, что невозможно контролировать при помощи флагов отладки, — это смысловые ошибки.

В процессе подготовки главы мы обнаружили несколько подобного рода ошибок в конфигурационных файлах. Иногда, к примеру, вызывалось средство без сопутствующего ему макроса (в частности, активизировалось средство `masquerade_envelope`, но не включался режим маскирования адресов с помощью макроса `MASQUERADE_AS`). Или конфигурация программы **sendmail** не соответствовала настройкам брандмауэра, который определял, какую почту разрешается пропускать.

Нельзя проектировать почтовую систему саму по себе. Ее необходимо согласовать с имеющимися записями `MX` базы данных DNS и настройками брандмауэра.

Мониторинг очереди

С помощью команды **mailq** (эквивалент команды **sendmail -bp**) можно просматривать состояние сообщений, помещенных в очереди. Сообщение могут помещаться в очередь после доставки или при неудачной попытке доставки.

Команда **mailq** выводит отчет о файлах в каталоге **/var/spool/mqueue** в любой заданный момент. Результат позволяет определить, почему сообщение могло быть задержано. Если выясняется, что в системе увеличивается поток почтовых сообщений, системный администратор может отследить попытки программы **sendmail** разобрать этот завал.

По умолчанию существует две очереди: для сообщений, полученных через порт 25, и для сообщений, полученных через порт 587 (очередь подачи для клиентов). Клиентскую очередь можно увидеть с помощью команды **mailq -Ac**.

Ниже приведен типичный результат работы команды **mailq**. В отчете указано, что доставки ожидают три сообщения.

```
$ sudo mailq
/var/spool/mqueue (3 requests)
-----Q-ID----- --Size-- -----Q-Time----- ------------Sender/Recipient-------
k623gYYk008732   23217       Sat Jul 1 21:42         MAILER-DAEMON
    8BITMIME (Deferred: Connection refused by agribusinessonline.com.)
                                        <Nimtz@agribusinessonline.com>
k5ULkAHB032374   279         Fri Jun 30 15:46 <randy@atrust.com>
    (Deferred: Name server: k2wireless.com.: host name lookup fa)
                                        <relder@k2wireless.com>
k5UJDm72023576 2485 Fri Jun 30 13:13 MAILER-DAEMON
    (reply: read error from mx4.level3.com.)
                                         <lfinist@bbnplanet.com>
```

Если вы считаете, что разбираетесь в ситуации лучше, чем программа **sendmail**, или просто хотите, чтобы программа **sendmail** попыталась немедленно повторить попытку доставки сообщений, попавших в очередь, можете запустить повторную обработку очереди с помощью команды **sendmail -q**. Если вы используете команду **sendmail -q -v**, то программа **sendmail** выведет результаты для каждой попытки доставки, что часто оказывается полезным для отладки. Программа **sendmail** повторяет попытки доставить сообщения в каждой очереди по истечении заданного интервала времени (как правило, каждые 30 минут).

Журнальная регистрация

Программа **sendmail** регистрирует сообщения об ошибках и своем состоянии через систему Syslog. Это делается от имени средства `mail` на уровнях от `debug` до `crit`. Все сообщения помечаются строкой "`sendmail`". Изменить эту установку можно с помощью флага командной строки **-L**, что довольно удобно, если отлаживается одна из копий программы, в то время как остальные копии продолжают нормально работать.

🕮 Подробнее о системе Syslog рассказывалось в главе 10.

Опция `confLOG_LEVEL`, заданная в командной строке или в файле конфигурации, определяет уровень важности, который программа **sendmail** примет в качестве порогового. Высокие пороговые значения соответствуют низким уровням важности и обеспечивают регистрацию большего объема информации.

В табл. 18.15 показано приблизительное соответствие между уровнями журнальной регистрации в программе **sendmail** и уровнями важности Syslog.

Таблица 18.15. Соотношение между пороговыми уровнями программы `sendmail` и уровнями важности Syslog

Порог	Уровень важности
0	Регистрация отключена
1	`alert` или `crit`
2	`crit`
3	`err` или `warning`
4	`notice`
5-11	`info`
>=12	`debug`

Напомним, что сообщение, зарегистрированное в системном журнале на конкретном уровне, относится как к этому уровню, так и ко всем находящимся выше уровням. Файл `/etc/syslog.conf` определяет окончательный пункт назначения каждого сообщения. Местоположения, заданные по умолчанию, показаны в табл. 18.16.

Таблица 18.16. Местоположения регистрационных журналов программы sendmail

Система	Местоположение регистрационного файла
Debian	`/var/log/mail.log`
Ubuntu	`/var/log/mail.log`
Red Hat	`/var/log/maillog`
CentOS	`/var/log/maillog`
FreeBSD	`/var/log/maillog`

Существует несколько программ, которые могут обрабатывать регистрационные файлы **sendmail**, создавая итоговые отчеты, которые варьируются от простых количественных и текстовых таблиц (**mreport**) до сложных веб-страниц (Yasma). У системного администратора может возникнуть необходимость или желание ограничить доступ к этим данным или по крайней мере проинформировать пользователей о сборе данных о них.

18.9. Почтовый агент Exim

Агент транспорта и доставки почты Exim был написан в 1995 году Филипом Гейзелом из Кембриджского университета и распространен под универсальной общедоступной лицензией GNU. Текущий на момент русского издания книги выпуск, Exim версии 4.92, появился в весной 2019 года. В интерактивном доступе находятся тонны документации о программе, кроме того, было опубликовано несколько книг, написанных автором программного обеспечения.

Запросы в поисковой системе Google о почтовом сервере Exim часто приводят к устаревшим, а иногда и вообще неприемлемым ответам, поэтому в первую очередь следует обращаться к официальной документации. В дистрибутивный пакет входит документ о спецификации и конфигурации (**doc/spec.txt**), состоящий из более чем 400 страниц. Этот документ также доступен на сайте `exim.org` в виде файла PDF. Он представляет собой исчерпывающий справочник по программе Exim, который неукоснительно обновляется с каждым новым выпуском.

Существует два подхода к конфигурации постового сервера Exim: Debian и альтернативный. Операционная система Debian использует препроцессор **m4** и управляет своим собственным набором списков рассылки для поддержки пользователей. Мы не будем рассматривать расширения конфигурации, характерные для системы Debian.

Почтовый сервер Exim похож на программу **sendmail** тем, что он реализован как единственный процесс, выполняющий все почтовые операции. Однако его авторы отказались от исторического багажа программы **sendmail** (поддержки устаревших форматов адресов, необходимости получать почту для хостов, не находящихся в Интернете, и т.д.). Многие аспекты функционирования почтового сервера Exim определяются во время компиляции, при этом ориентирами являются база данных почтового сервера Exim и форматы хранения сообщений.

Основные рабочие инструменты в почтовом сервере Exim называются *маршрутизаторами* и *транспортными средствами*. Оба они относятся к общей категории "драйве-

ров". Маршрутизаторы решают, как сообщения должны быть переданы, а транспортные средства выбирают способ доставки. Маршрутизаторы — это упорядоченный список адресов, которые следует проверить, а транспортные средства — неупорядоченный набор способов доставки.

Инсталляция почтового сервера Exim

Последнюю версию дистрибутивного пакета можно загрузить с сайта `exim.org` или, если ваш сайт работает под управлением системы Linux, из вашего любимого репозитария пакетов. Место, где следует инсталлировать программу, идентификаторы пользователей и другие параметры указаны в файлах **README** и **src/EDITME**. Файл **EDITME** содержит более 1000 строк, которые по большей части представляют собой комментарии к процессу компиляции; требуемые изменения помечены. После редактирования сохраните файл как **../Local/Makefile** или **../Local/Makefile-***имя_ОС* (если в одном и том же каталоге вы создаете конфигурации для разных операционных систем), а затем запустите команду **make**.

Ниже перечислены несколько важных переменных (по нашему мнению) и предлагаемые значения (по мнению разработчиков почтового сервера Exim) из файла **EDITME**. Первые пять строк являются обязательными, остальные только рекомендуются.

```
BIN_DIRECTORY=/usr/exim/bin             # Место для исполняемого модуля exim
SPOOL_DIRECTORY=/var/spool/exim         # Каталог для почтовой очереди
CONFIGURE_FILE=/usr/exim/configure      # Файл конфигурации программы Exim
SYSTEM_ALIASES_FILE=/etc/aliases        # Место для файлов псевдонимов
EXIM_USER=ref:exim                      # Пользователь программы EXIM

ROUTER_ACCEPT=yes                       # Драйверы маршрутизатора
ROUTER_DNSLOOKUP=yes
ROUTER_IPLITERAL=yes
ROUTER_MANUALROUTE=yes
ROUTER_QUERYPROGRAM=yes
ROUTER_REDIRECT=yes
TRANSPORT_APPENDFILE=yes                # Драйверы транспорта
TRANSPORT_AUTOREPLY=yes
TRANSPORT_PIPE=yes
TRANSPORT_SMTP=yes
SUPPORT_MAILDIR=yes                     # Разрешенные форматы почтовых ящиков
SUPPORT_MAILSTORE=yes
SUPPORT_MBX=yes
LOOKUP_DBM=yes                          # Системы управления базами данных
LOOKUP_LSEARCH=yes                      # Метод последовательного поиска
LOOKUP_DNSDB=yes                        # Разрешает почти все поиски в DNS
USE_DB=yes                              # Использовать Berkeley DB (из README)
DBMLIB=-ldb                             # (из файла README)
WITH_CONTENT_SCAN=yes                   # Включает сканирование содержимого
                                        # через списки ACL
EXPERIMENTAL_SPF=yes                    # Включает поддержку SPF, нужна libspf2
CFLAGS += -I/usr/local/include          # Из www.libspf2.org
LDFLAGS += -lspf2
LOG_FILE_PATH=/var/log/exim_%slog       # Файлы журнала: файл, Syslog или оба
LOG_FILE_PATH=syslog
LOG_FILE_PATH=syslog:/var/log/exim_%slog
EXICYCLOG_MAX=10                        # Количество сохраненных файлов журнала
                                        # равно 10
```

Если вы собираетесь использовать маршрутизаторы и транспортные средства, то их следует скомпилировать в код. В настоящее время, которое характеризуется большими объемами доступной памяти, их можно использовать все вместе. Некоторые пути, заданные по умолчанию, являются нестандартными: например бинарный файл в каталоге **/usr/exim/bin** и файл PID в каталоге **/var/spool/exim**. Эти параметры можно изменить в соответствии со своим инсталлированным программным обеспечением.

Существует около десяти доступных систем управления базами данных, включая MySQL, Oracle и LDAP. Если вы хотите выбрать систему LDAP, то должны указать переменную LDAP_LIB_TYPE, которая сообщит программе Exim о том, что вы используете библиотеку LDAP. Кроме того, вы должны указать путь для включения файлов и библиотек системы LDAP.

В файле **EDITME** хранится информация о любых зависимостях, которые могут понадобиться при выборе базы данных. В комментариях о зависимостях, которые сопровождают каждую запись, есть фраза "(from README)", в которой указан не файл **src/EDITME**, а именно **README**.

Файл **EDITME** имеет много дополнительных параметров безопасности, которые позволяют активизировать дополнительные возможности, такие как SMTP AUTH, TLS, SASL, PAM, а также параметры управления владением файлом и разрешениями доступа к нему. Некоторые возможности программы Exim можно отключить на этапе компиляции, чтобы ограничить последствия ее взлома злоумышленниками.

До завершения инсталляции мы рекомендуем прочитать файл **EDITME** полностью. Это даст вам приятное ощущение, что вы управляете выполнением программы посредством конфигурационного файла. Файл **README**, находящийся в корне дистрибутивного каталога, содержит много информации о деталях, специфичных для операционных систем, которые иногда полезно включать в файл **EDITME**.

Модифицировав файл **EDITME** и инсталлировав его в каталог **Local/Makefile**, запустите команду **make** на вершине дистрибутивной иерархии, а затем инструкцию **sudo make install**. На следующем этапе необходимо протестировать исполняемый модуль **exim** и убедиться, что он доставляет почту так, как ожидается. Полезная тестовая документация содержится в файле **doc/spec.txt**.

Убедившись, что почтовый сервер Exim работает правильно, свяжите модуль **/usr/sbin/sendmail** с модулем **exim**, чтобы почтовый сервер Exim мог эмулировать традиционный интерфейс командной строки для обмена информацией с почтовой системой, используемой многими почтовыми агентами. Кроме того, следует сделать так, чтобы модуль **exim** запускался во время загрузки системы.

Загрузка почтового сервера Exim

На компьютере, выполняющем роль концентратора почты, модуль **exim** обычно запускается во время загрузки в режиме демона и работает непрерывно, прослушивая порт 25 и принимая сообщения в рамках протокола SMTP. Загрузка операционных систем описана в главе 2.

Как и программа **sendmail**, почтовый сервер Exim может иметь несколько режимов работы, и при запуске с разными флагами она выполняет разные функции. Флаги режима почтового сервера Exim похожи на флаги режима программы **sendmail**, потому что разработчики модуля **exim** очень стараются обеспечить совместимость при вызове почтовых агентов и других средств. Некоторые из наиболее распространенных флагов перечислены в табл. 18.17.

Таблица 18.17. Распространенные флаги командной строки модуля Exim

Флаг	Смысл
`-bd`	Запускается в режиме демона и ожидает соединения на порту 25
`-bf` *или* `-bF`	Запускается в пользовательском режиме или в режиме тестирования системного фильтра
`-bi`	Перестраивает хешированные псевдонимы (эквивалентен команде **newaliases**)
`-bp`	Выводит на печать почтовую очередь (эквивалентен команде **mailq**)
`-bt`	Включает режим тестирования адресов
`-bV`	Проверяет синтаксические ошибки в конфигурационном файле
`-d+`*категория*	Запускается в режиме отладки; очень гибкая конфигурация, учитывающая категории
`-q`	Запускает обработчик очереди (эквивалентен команде **runq**)

Любые ошибки, существующие в конфигурационном файле, можно обнаружить на этапе синтаксического анализа с помощью команды **exim -bV**, но некоторые ошибки можно выявить только на этапе выполнения. Распространенная ошибка — неправильно расставленные скобки.

Все нюансы, связанные с флагами и параметрами команды **exim**, включая обширную информацию об отладке, можно найти на соответствующей справочной странице.

Утилиты почтового сервера Exim

Дистрибутивный пакет программы Exim включает в себя набор утилит, облегчающих слежение, отладку и проверку инсталляции. Ниже приведен список с кратким описанием каждой из этих утилит. Более подробная информация о них изложена в документации.

- **exicyclog** — выполняет ротацию журнальных файлов.
- **exigrep** — просматривает основной журнал.
- **exilog** — визуализирует регистрационные файлы на нескольких серверах.
- **exim_checkaccess** — проверяет доступность заданного IP-адреса.
- **exim_dbmbuild** — создает файл DBM.
- **exim_dumpdb** — распечатывает базу данных уточнений (hints).
- **exim_fixdb** — исправляет базу данных уточнений .
- **exim_lock** — блокирует файл почтового ящика.
- **exim_tidydb** — очищает базу данных уточнений .
- **eximstats** — извлекает статистические показатели из журнала.
- **exinext** — извлекает информацию о повторных попытках.
- **exipick** — ищет сообщения по заданным критериям.
- **exiqgrep** — выполняет поиск в очереди.
- **exiqsumm** — создает отчет об очереди.
- **exiwhat** — выводит список выполняемых процессов программы Exim.

Еще одна утилита, которая входит в пакет Exim, называется **eximon**. Она представляет собой приложение для системы X Windows, которое выводит на экран состояние программы Exim, состояние ее очереди и несколько последних строк из файла журнала. Как и основной дистрибутивный пакет, она создается с помощью редактирования хорошо комментированного файла **EDITME** в каталоге **exim_monitor** и запуска команды **make**. Параметры, заданные по умолчанию для утилиты **eximon**, подобраны очень удачно, поэтому настройка этого приложения обычно не вызывает затруднений. Кроме

того, конфигурирование и управление очередью можно осуществлять с помощью графического интерфейса утилиты **eximon**.

Язык конфигурации программы Exim

Рассматриваемый здесь язык конфигурации программы Exim (точнее, языки: один для фильтров, другой для регулярных выражений и так далее) напоминает довольно старый (разработанный в 1970-х годах) язык Forth.[17] При первом чтении конфигурации программы иногда трудно отличить ключевые слова и имена параметров, которые являются фиксированными в языке Exim, от имен переменных, определенных системными администраторами.

Несмотря на то что программа Exim рекламируется как легкая для конфигурирования и хорошо документированная, ее довольно сложно освоить. Раздел спецификации "Как программа Exim получает и доставляет почту" требует от новичков больших усилий. Тем не менее в ней изложены все основные концепции, лежащие в основе системы.

После присвоения стандартной переменной конкретного значения язык Exim иногда активизирует определенные действия. В нем есть около 120 стандартных переменных, значения которых можно изменять в результате одного из действий. Эти переменные можно включать в условные выражения.

Выражение для вычисления инструкции `if` и подобных ей может напомнить обратную польскую запись, характерную для времен расцвета калькуляторов Hewlett-Packard. Рассмотрим простой пример. В строке

```
acl_smtp_rcpt = ${if ={25}{$interface_port} \
    {acl_check_rcpt} {acl_check_rcpt_submit} }
```

установка параметра `acl_smtp_rcpt` вызывает реализацию списка управления доступом для каждого пользователя (т.е. выполнение команды SMPT RCPT). Значение, присвоенное этому параметру, может быть равным `acl_check_rcpt` или `acl_check_rcpt_submit`, в зависимости от того, имеет ли переменная `$interface_port` значение 25.

Мы не будем углубляться в детали языка конфигурации программы Exim, лишь посоветуем читателям обратиться к обширной документации. В частности, обратите внимание на раздел расширения строк в спецификации программы Exim.

Файл конфигурации программы Exim

Поведением программы Exim во время выполнения управляет файл конфигурации, который обычно называется **/usr/exim/configure**. Его имя представляет собой одну из обязательных переменных, задаваемых в файле **EDITME** и компилируемых в бинарный код.

Файл конфигурации **src/configure.default**, поставляемый по умолчанию, сопровождается подробными комментариями и хорошо подходит для новичков. Мы рекомендуем не слишком далеко отклоняться от этого файла, пока вы не освоитесь с парадигмой Exim и не научитесь создавать более сложные конфигурации для специальных ситуаций. Разработчики программы Exim хорошо продумали поддержку наиболее распространенных ситуаций и создали вполне разумные стандартные конфигурации.

Рекомендуем также не изменять имена переменных, использованных в файле конфигурации, заданном по умолчанию, поскольку именно их ожидают увидеть в списках рассылки люди, которые будут отвечать на ваши вопросы о конфигурации.

---

[17]Для экспертов в компьютерных науках заметим, что этот язык является полным по Тьюрингу; в переводе для простых смертных это значит "мощный и сложный".

Программа **exim** выводит сообщения в поток `stderr` и прекращает работу, если в вашем файле конфигурации есть синтаксическая ошибка. Однако она не выявляет все синтаксические ошибки немедленно, поскольку не обращается к переменным, пока в них нет потребности.

Порядок записей в файле конфигурации не совсем произвольный: раздел глобальных параметров конфигурации должен существовать и быть первым. Все остальные разделы являются необязательными и могут следовать в любом порядке.

Перечислим некоторые разделы.

- Глобальные параметры конфигурации (обязательны).
- `acl` — списки управления доступом, фильтрующие адреса и сообщения.
- `authenticators` — раздел для параметров команды SMPT `AUTH` или протокола аутентификации TLS.
- `routers` — упорядоченная последовательность, предназначенная для определения места назначения сообщения.
- `transports` — определения драйверов, предназначенных для фактической доставки.
- `retry` — настройки правил для решения проблем, связанных с сообщениями.
- `rewrite` — правила перезаписи глобальных адресов.
- `local_scan` — ловушка для спама.

Каждый раздел, за исключением первого, начинается с инструкции `begin` *`имя-раздела`*, например `begin acl`. Инструкции `end` *`имя-раздела`* не существует; признаком конца раздела является инструкция `begin` следующего раздела. Отступы между разделами облегчают чтение, но для программы Exim значения не имеют.

Некоторые инструкции конфигурации присваивают имена объектам, которые впоследствии будут использованы для управления потоками сообщений. Эти имена должны начинаться с буквы и содержать только буквы, цифры и символ подчеркивания. Если первым символом, отличающимся от разделителей строки, является символ #, то вся остальная часть строки считается комментарием. Это значит, что вы не можете поместить комментарий в строку, в которой помещается инструкция; он не будет распознан как комментарий, поскольку его первый символ не является символом #.

Программа Exim позволяет включать файлы в любое место файла конфигурации. Существует две формы включения файлов.

```
.include полный-путь
.include_if_exists полный-путь
```

Первая форма генерирует ошибку, если файла нет. Несмотря на то что включение файлов позволяет сохранять небольшой объем конфигурационного файла, за время обработки сообщения они считываются несколько раз, поэтому лучше просто включить их содержимое прямо в файл конфигурации.

Глобальные параметры

В разделе глобальных параметров задается множество данных, включая рабочие параметры (пределы, размеры, тайм-ауты, свойства почтового сервера на указанном хосте), определения списков (локальных хостов, локальных хостов для перенаправления, удаленных доменов для перенаправления) и макросы (имя хоста, контакт, местоположение, сообщения об ошибках, баннер SMTP).

Параметры

Параметры устанавливаются с помощью следующей синтаксической конструкции.

имя_параметра = значение[я]

Здесь *значения* могут быть булевыми или строчными, целыми или десятичными числами, а также временными интервалами. Допускается также несколько значений, в этом случае они разделяются двоеточиями.

Использование двоеточия в качестве разделителя создает проблему, потому что в адресах IPv6 оно является частью адреса. Решить эту проблему можно путем удваивания количества двоеточий, но намного проще и удобнее для чтения переопределить символ разделения и задать его равным < во время присвоения параметру его значения. Например, в следующих строках задаются значения параметра `localhost_interfaces`, содержащие IPv4- и IPv6-адреса локального хоста.

```
local_interfaces = 127.0.0.1 : ::::1
local_interfaces = <; 127.0.0.1 ; ::1
```

Вторая форма, в которой в качестве разделителя используется точка с запятой, является более удобной для чтения и менее уязвимой для ошибок.

Существует огромное количество параметров — в предметном указателе документации их более 500. А мы говорили, что программа **sendmail** была слишком сложной! Большинство параметров имеет вполне разумные значения по умолчанию, и все параметры имеют информативные имена. Для того чтобы легко находить новый параметр, целесообразно скопировать файл **doc/spec.txt** из дистрибутивного пакета в свой привычный текстовый редактор. Мы не собираемся описывать все параметры, а упомянем лишь те из них, которые встречаются в наших примерах.

Списки

Программа Exim использует четыре вида списков, которые объявляются с помощью ключевых слов `hostlist`, `domainlist`, `addresslist` и `localpartslist`. Ниже приведены два примера использования ключевого слова `hostlist`.

```
hostlist my_relay_list = 192.168.1.0/24 : myfriend.example.com
hostlist my_relay_list = /usr/local/exim/relay_hosts.txt
```

Члены списка могут быть перечислены непосредственно в строке или загружены из файла. Если они указываются непосредственно в строке, то разделяются двоеточиями. Существует до 16 именованных списков каждого типа. В примерах, приведенных выше, мы включили все компьютеры в локальной сети /24 и указали конкретное имя локального компьютера.

Символ `@` может быть членом списка; он означает имя локального хоста и помогает написать единственный обобщенный конфигурационный файл, который будет применяться на всех компьютерах вашей сети. Не менее полезным является обозначение `@[]`, которое означает все IP-адреса, прослушиваемые программой Exim; иначе говоря, все IP-адреса локального хоста.

Для того чтобы сослаться на список, следует просто поставить символ + перед его именем, если хотите сравнивать его члены, или !+, если хотите сравнивать элементы, не являющиеся членами списка; например, `+my_relay_list`. Между знаком + и именем списка пробела быть не должно.

Списки могут содержать ссылки на другие списки, а знак ! означает отрицание. Списки, содержащие ссылки на переменные (например, `$variable_name`), замедляют обработку, потому что по умолчанию программа Exim не может кешировать результаты сравнений по списку.

Макрос

Макросы можно использовать для определения параметров, сообщений об ошибках и т.д. Грамматический разбор макросов очень примитивен, поэтому вы не можете определить макрос, имя которого является подмножеством другого макроса, не получив непредсказуемых результатов.

Синтаксис макроса выглядит следующим образом.

```
MACRO_NAME = остальная часть строки
```

Например, в первой из приведенных ниже строк определяется макрос с именем ALIAS_QUERY, который выполняет поиск записи о псевдониме пользователя в базе данных MySQL. Вторая строка демонстрирует использование макроса для выполнения реального поиска, результат которого хранится в переменной с именем data.

```
ALIAS_QUERY = \
   select mailbox from user where login = '${quote_mysql:$local_part}';
data = ${lookup mysql{ALIAS_QUERY}}
```

Имена макросов не обязательно набирать только прописными буквами, но они должны начинаться с прописной буквы. Однако соглашение, по которому макросы состоят только из прописных букв, повышает ясность текста. Файл конфигурации может содержать директиву ifdef, вычисляющую макрос. Система использует ее, чтобы определить, включать или нет фрагмент конфигурационного файла. Поддерживаются все возможные формы инструкции ifdef; все они начинаются с точки.

Списки управления доступом (ACL)

Списки управления доступом фильтруют адреса входящих сообщений и либо принимают их, либо отклоняют. Программа Exim разделяет адреса входящих сообщений на локальную часть, соответствующую пользователю, и доменную, соответствующую домену получателя.

Списки управления доступом можно применять на разных стадиях обмена информацией по протоколу SMTP: HELO, MAIL, RCPT, DATA и т.д. Обычно список управления доступом требует строгого следования протоколу SMTP на стадии HELO, проверяет отправителя и его домен на стадии MAIL, проверяет получателя на стадии RCPT и сканирует содержимое сообщения на стадии DATA.

Для того чтобы определить, какой список управления доступом должен применяться после каждой команды в протоколе SMTP, используется множество параметров с именами, имеющими формат acl_smtp_*команда*. Например, параметр acl_smtp_rcpt означает, что список управления доступом должен применяться к каждому адресу, указанному как получатель данного сообщения. Списки управления доступом можно задать в разделе acl в файле конфигурации, в файле, на который ссылается параметр acl_smtp_*команда*, или в командной строке.

Ниже приведен пример списка управления доступом my_acl_check_rcpt. Его можно вызвать, присвоив это имя параметру acl_smtp_rcpt в разделе глобальных параметров в файле конфигурации. Если этот список управления доступом отвергает адрес, указанный в команде RCPT, то сервер отправителя должен прекратить отправку сообщения на этот адрес. Другой параметр списка управления доступом — acl_smtp_data — применяется к сообщению после его получения, например для сканирования его содержимого.

```
begin acl
    my_acl_check_rcpt:
        accept   hosts = :
                 control = dkim_disable_verify
```

Этому списку управления доступом по умолчанию назначено имя acl_check_rcpt; вероятно, это имя изменять не следует. Тем не менее мы изменили его, чтобы продемонстрировать, что оно задается вами, а не фиксируется заранее в параметрах конфигурации программы Exim.

Первая строка, содержащая раздел accept и одно двоеточие, представляет собой пустой список. Пустой список удаленных хостов применяется в ситуациях, когда локальный пользовательский почтовый агент (MUA) подает сообщение на стандартный вход транспортного агента. Если проверяемый адрес соответствует данному условию, список управления доступом принимает адрес и отключает проверку подписи DKIM, которая по умолчанию включена. Если же адрес не соответствует данному оператору address, то управление переходит к следующему разделу в определении списка.

```
deny     message = Restricted characters in address
         domains = +local_domains
         local_parts = ^[.] : ^.*[@%!/|]

deny     message = Restricted characters in address
         domains = !+local_domains
         local_parts = ^[./|] : ^.*[@%!] : ^.*/\\.\\./
```

Первая строка конфигурации deny предназначена для сообщений, поступающих на ваши локальные домены. Она отклоняет любой адрес, в котором локальная часть (имя пользователя) начинается с точки или содержит специальные символы @, %, !, / или |. Вторая строка deny применяется к сообщениям, посланным во внешний мир вашими пользователями. Она также не допускает использование некоторых специальных символов и их комбинаций в качестве части адреса, поскольку это свидетельствует о том, что ваши компьютеры были заражены вирусом или другими вредоносными программами. В прошлом такие адреса использовались спамерами для запутывания списков управления доступом или для создания проблем в системе безопасности.

В целом, если вы хотите использовать выражения $local_parts (например, в качестве пользовательского имени получателя) в имени каталога (для хранения почты или просмотра файла автоответчика, например), будьте очень осторожны, чтобы ваши списки управления доступом, отфильтровывающие любые специальные символы, не привели к непредвиденным последствиям. (В этом примере выполняется поиск последовательности символов /../, которые могут создать проблемы, если имя пользователя является частью пути.)

```
accept   local_parts = postmaster
         domains = +local_domains
```

Следующий раздел accept гарантирует, что почта, посланная администратору электронной почты, всегда будет доставляться, если она была послана в локальный домен; это облегчает отладку.

```
require  verify = sender
```

Строка require проверяет, можно ли вернуть сообщение о недоставке, но проверка относится только к домену отправителя.[18] Если пользовательское имя отправителя было подделано, то сообщение о недоставке может породить новое сообщение о недоставке. В этом месте проверку можно усилить, вызвав другую программу, но некоторые сайты рассматривают такие вызовы как злонамеренные действия и могут внести ваш почтовый сервер в черный список или в список серверов с плохой репутацией.

```
accept   hosts = +relay_from_hosts
         control = submission
         control = dkim_disable_verify
```

[18]Раздел require означает "deny, если не совпало".

Следующий раздел accept проверяет хосты, для которых разрешается выполнять ретрансляцию, т.е. локальные хосты, подающие почту в систему. Эта строка означает, что программа **exim** должна действовать как агент передачи почтовых сообщений и исправлять любые недостатки в заголовках при поступлении сообщений от пользовательского агента. Адрес отправителя не проверяется, потому что многие пользовательские агенты сбиваются с толку ошибочными возвратами. (Это приемлемо только для локальных машин, которые выполняют ретрансляцию на интеллектуальный хост, не во внешние домены.) Верификация подписи DKIM отключена, поскольку эти сообщения отправляются вашими пользователями или их коллегами по ретрансляции.

```
accept   authenticated = *
         control = submission
         control = dkim_disable_verify
```

Последний раздел accept относится к локальным хостам, выполняющим аутентификацию с помощью протокола SMTP AUTH. Эти сообщения снова обрабатываются как передачи от пользовательских агентов.

```
require  message = Relay not permitted
         domains = +local_domains : +relay_to_domains

require verify = recipient
```

Затем мы проверяем домен адресата, который указан в заголовке письма. Он должен содержаться либо в нашем списке local_domains, либо в списке relay_to_domains, в котором перечислены домены, имеющие право на ретрансляцию. (Списки этих доменов определяются в другом месте.) Письмо с адресом, который не входит ни в один из этих списков, порождает особое сообщение об ошибке.

```
accept
```

Итак, если все наши сформулированные выше предположения были удовлетворены и ни одно из более тонких правил accept или deny не было нарушено, мы верифицируем получателя и принимаем сообщение. В эту категорию попадает большинство сообщений, поступающих из Интернета.

В приведенный выше пример мы не включили сканирование черных списков. Для доступа к черным спискам следует использовать один из примеров, описанных в файле конфигурации, предоставленном по умолчанию, или что-нибудь наподобие следующего кода.

```
deny     condition = ${if isip4{$sender_host_address}}
         !authenticated = *
         !hosts = +my_whitelist_ips
         !dnslists = list.dnswl.org
         domains = +local_domains
         verify = recipient
         message = You are on RBL $dnslist_domain: $dnslist_text
         dnslists = zen.spamhaus.org
         logwrite = Blacklisted sender [$sender_host_address] \
             $dnslist_domain: $dnslist_text
```

В переводе на обычный язык этот фрагмент означает, что если сообщение соответствует *всем* следующим критериям, то оно отклоняется с сообщением о пользовательской ошибке и регистрируется в журнале (также вместе с сообщением о пользовательской ошибке).

- Сообщение поступило с IPv4-адреса (некоторые списки неправильно обрабатывают IPv6-адреса).
- Сообщение не ассоциировано с аутентифицированным сеансом SMTP.

- Сообщение поступило от отправителя, которого нет в локальном белом списке.
- Сообщение поступило от отправителя, которого нет в глобальном (из Интернета) белом списке.
- Сообщение адресовано корректному локальному получателю.
- Хост отправителя занесен в черный список сайта `zen.spamhaus.org`.

Переменные `dnslist_text` и `dnslist_domain` задаются путем присваивания параметру `dnslists`, который выполняет переключение поиска по черным спискам. Этот раздел `deny` должен размещаться сразу после проверки необычных символов в адресах.

Рассмотрим еще один пример списка управления доступом, который отклоняет сообщение, поступившее от удаленного сайта, неправильно ответившего на команду `HELO`.

```
acl_check_mail:
    deny message = 503 Bad sequence of cmds - must send HELO/EHLO first
         condition = ${if !def:sender_helo_name}
    accept
```

Программа Exim решает проблему "болтуна" (более специальный случай сайта, "неправильно отвечающего на команду `HELO`") с помощью параметра `smtp_enforce_sync`, который включается по умолчанию.

Сканирование содержимого на этапе применения списков управления доступом

Программа Exim поддерживает мощное сканирование содержимого в нескольких точках на пути сообщения внутри почтовой системы: на этапе применения списков управления доступом (после команды SMTP `DATA`), во время доставки с помощью параметра `transport_filter` или функции `local_scan` после выполнения всех проверок списков управления доступом. Для поддержки сканирования необходимо выполнить компиляцию данного модуля в систему Exim с помощью переменной `WITH_CONTENT_SCAN` в файле **`EDITME`**; по умолчанию она закомментирована. Этот параметр наделяет списки управления доступом дополнительными мощью и гибкостью, добавляя два новых параметра конфигурации: `spamd_address` и `av_scanner`.

Сканирование на этапе проверки списков управления доступом позволяет отклонять сообщения на лету с помощью обмена информацией между транспортным агентом и хостом отправителя. Сообщение никогда не будет принято к доставке, поэтому сообщение о недоставке не генерируется. Это очень хороший способ отклонения сообщений, поскольку он позволяет избежать обратной рассылки спама, вызванного сообщениями о недоставке, по поддельным адресам отправителей.

Аутентификаторы

Аутентификаторы (authenticators) — это драйверы, взаимодействующие с последовательностью "вызов-ответ" команд SMTP `AUTH` и идентифицирующие механизм аутентификации, приемлемый для клиента и сервера. Программа Exim поддерживает перечисленные ниже механизмы.

- `AUTH_CRAM_MD5` (RFC2195).
- `AUTH_PLAINTEXT`, включающий макросы `PLAIN` и `LOGIN`.
- `AUTH_SPA`, поддерживающий механизм аутентификации Secure Password Authentication компании Microsoft.

Когда агент **exim** получает электронную почту, он действует как сервер SMTP AUTH. Посылая электронную почту, он является клиентом. Параметры, появляющиеся в определениях экземпляров аутентификатора, сопровождаются префиксами `server_` или `client_`, позволяющими задавать разные конфигурации в зависимости от роли, которую играет агент Exim.

Аутентификаторы используются в списках управления доступом, как показано в одном из примеров, приведенных выше.

```
deny  !authenticated = *
```

Ниже приведен пример, демонстрирующий клиентский и серверный механизмы LOGIN. В этом простом примере используются фиксированные имя пользователя и пароль, что приемлемо для небольших организаций, но, вероятно, это не следует рекомендовать для крупных сетей.

```
begin authenticators

  my_client_fixed_login:
     driver = plaintext
     public_name = LOGIN
     client_send = : myusername : mypasswd

  my_server_fixed_login:
     driver = plaintext
     public_name = LOGIN
     server_advertise_condition = ${if def:tis_cipher}
     server_prompts = User Name : Password
     server_condition = ${if and {{eq{$auth1}{имя_пользователя}} \
          {eq{$auth2}{пароль}}}}
     server_set_id = $auth1
```

Данные для аутентификации могут поступать из многих источников: LDAP, PAM, **/etc/passwd** и т.д. Раздел `server_advertise_condition` в указанном фрагменте кода предотвращает открытую пересылку по линии связи паролей почтовых клиентов по требованию протокола TLS (через STARTTLS или SSL). Если вы хотите, чтобы программа **exim** действовала аналогично, будучи клиентский системой, используйте параметр `client_condition` в разделе `client`, тоже вместе с параметром `tis_cipher`.

Детальное описание всех возможных параметров аутентификации программы Exim и примеры их использования можно найти в документации.

Маршрутизаторы

Маршрутизаторы обрабатывают адреса электронной почты, либо перезаписывая их, либо переприсваивая транспортному агенту и посылая в пункт назначения. Конкретный маршрутизатор может иметь несколько экземпляров с разными параметрами.

Вы указываете последовательность маршрутизаторов. Сообщение начинает свой путь с первого маршрутизатора и проходит по списку, пока сообщение не будет принято или отклонено. Принимающий маршрутизатор обычно передает сообщения драйверу транспорта. Маршрутизаторы обрабатывают как входящую, так и исходящую корреспонденцию. Их можно считать чем-то вроде подпрограмм в языке программирования.

Маршрутизатор может возвращать любые из уведомлений о статусе сообщения, перечисленных в табл.18.18.

Таблица 18.18. Распространенные флаги командной строки модуля Exim

Флаг	Смысл
`accept`	Маршрутизатор принимает адрес и передает его драйверу транспорта
`decline`	Маршрутизатор решает не обрабатывать адрес; следующий маршрутизатор, пожалуйста!
`defer`	Оставляет сообщение в очереди для последующей обработки
`error`	Обнаружена ошибка в спецификации маршрутизатора; сообщение откладывается
`fail`	Адрес является неправильным; маршрутизатор ставит его в очередь для генерации сообщения о недоставке
`pass`	Данный маршрутизатор не может обработать указанный адрес; управление передается следующему маршрутизатору

Если сообщение получило от всех маршрутизаторов в очереди уведомления `pass` или `decline`, оно возвращается отправителю как письмо, имеющее немаршрутизируемый адрес.

Если сообщение удовлетворяет всем предусловиям для маршрутизатора и маршрутизатор закончил работу, выполнив инструкцию `no_more`, то это сообщение не передается никаким дополнительным маршрутизаторам, независимо от того, какой статус оно получило у текущего маршрутизатора. Например, если ваш удаленный SMTP-маршрутизатор выполнил предусловие `domains = !+local_domains` и установил параметр `no_more`, то следующему маршрутизатору в последовательности отправляются только сообщения, адресованные локальным пользователям (т.е. сообщения, нарушающие предусловие домена).

Маршрутизаторы имеют много параметров; наиболее распространенными являются параметры предусловий или условий отказов, возвращаемых сообщений об ошибках и используемых драйверов транспорта.

В следующих разделах подробно описываются маршрутизаторы `accept`, `dnslookup`, `manualroute` и `redirect`. Эти фрагменты конфигурации основаны на предположении, что агент **`exim`** выполняется на локальном компьютере в домене `example.com`. Все эти фрагменты довольно просты; если вы захотите использовать более изощренные маршрутизаторы, обратитесь к документации.

Маршрутизатор `accept`

Этот маршрутизатор помечает адрес как приемлемый и передает соответствующее сообщение драйверу транспорта. Ниже приведен пример экземпляров маршрутизатора `accept` с именами `localusers` (для доставки локальной почты) и `save_to_file` (для отправки сообщения в архив).

```
localusers:
     driver = accept
     domains = example.com
     check_local_user
     transport = my_local_delivery
save_to_file:
     driver = accept
     domains = dialup.example.com
     transport = batchsmtp_appendfile
```

Экземпляр маршрутизатора с именем `localusers` проверяет, совпадает ли доменная часть адреса доставки с именем домена `example.com` и является ли локальная часть адреса регистрационным именем локального пользователя. Если оба условия выполнены, маршрутизатор передает сообщение экземпляру драйвера транспорта с именем

my_local_delivery, определенному в разделе транспорта. Экземпляр save_to_file предназначен для абонентов автоматической телефонной линии; он добавляет сообщение в файл, указанный в определении транспорта batchsmtp_appendfile.

Маршрутизатор dnslookup

Как правило, маршрутизатор dnslookup используется для исходящих сообщений. Он ищет запись MX о домене отправителя и передает сообщение драйверу транспорта SMTP для доставки. Ниже приведен экземпляр маршрутизатора с именем remoteusers.

```
remoteusers:
    driver = dnslookup
    domains = !+example.com
    transport = my_remote_delivery
```

🕮 Более подробно пространство частных адресов RFC1918 описано в разделе 13.4.

Код dnslookup ищет записи MX об адресате. Если их нет, он проверяет запись A. В итоге данный экземпляр маршрутизатора может запрещать доставку сообщений на определенные IP-адреса; например, частные адреса RFC1918 не могут маршрутизироваться в Интернет. Больше информации об этом можно найти в описании параметра ignore_target_hosts.

Маршрутизатор manualroute

Гибкий драйвер manualroute может посылать электронную почту куда угодно. Информация о маршрутизации может быть представлена в виде таблицы правил соответствия для домена получателя (route_list) или отдельного правила, которое применяется для всех доменов (route_data). Ниже приведены два экземпляра маршрутизатора manualroute. Первый экземпляр реализует концепцию "интеллектуального хоста", в которой все исходящие сообщения посылаются центральному "интеллектуальному" хосту для обработки. Этот экземпляр называется smarthost и применяется ко всем доменам получателей, не указанным (используется символ !) в списке local_domains.

```
smarthost:
    driver = manualroute
    domains = !+local_domains
    transport = remote_snmp
    route_data = smarthost.example.com
```

Экземпляр маршрутизатора firewall, приведенный ниже, использует протокол SMTP для отправки входящих сообщений хостам, находящимся за брандмауэром (возможно, после сканирования их на спам и вирусы). Он ищет данные о маршрутизации для каждого домена получателя в базе данных DBM, содержащей имена локальных хостов.

```
firewall:
    driver = manualroute
    transport = remote-smtp
    route_data = ${lookup{$domain} dbm {/internal/host/routes}}
```

Маршрутизатор redirect

Драйвер redirect перезаписывает адрес, используя псевдонимы из системного файла **aliases** или пользовательского файла **~/.forward**. Обычно он не присваивает пере-

записанный адрес транспорту, а оставляет эту работу другим маршрутизаторам, находящимся в цепочке.

Первый экземпляр system_aliases, приведенный ниже, ищет псевдонимы методом последовательного перебора (lsearch) в файле **/etc/aliases**. Это удобно, если файл **aliases** небольшой, но если он огромный, то метод последовательного перебора следует заменить поиском в базе данных. Второй экземпляр с именем forwardfile сначала проверяет, что почта адресована локальному пользователю, а затем просматривает пользовательский файл **.forward**.

```
system_aliases:
    driver = redirect
    data = ${lookup{$local_part} lsearch {/etc/aliases}}

user_forward:
    driver = redirect
    check_local_user
    file = $home/.forward
    no_verify
```

Параметр check_local_user гарантирует, что получатель является корректным локальным пользователем. Параметр no_verify означает, что проверка правильности адреса, на который перенаправляется сообщение, не нужна, следует просто выполнить доставку.

Фильтрация пользователей с помощью файлов .forward

Программа Exim позволяет не только перенаправлять сообщения, используя файлы **.forward**, но и фильтровать их, основываясь на содержимом пользовательского файла **.forward**. Она поддерживает собственную фильтрацию, а также фильтрацию методом решета (Sieve filtering), являющуюся стандартом IETF. Если первая строка пользовательского файла **.forward** выглядит как

```
#Exim filter
```

или

```
#Sieve filter
```

то последующие команды фильтрации (их около 15) можно использовать для выработки решения, куда следует доставить сообщение. Фильтрация на самом деле не подразумевает доставки сообщения, она просто выясняет пункт назначения. Рассмотрим пример.

```
#Exim filter
if   $header_subject: contains SAGE or $header_subject: contains sysadmin
then
     save $home/mail/sage-sysadmin
endif
```

Для того чтобы управлять тем, что пользователи могут делать в своих файлах **.forward**, а чего они делать не могут, используется множество параметров **.forward**. Имена этих параметров начинаются с префиксов forbid_ или allow_. Важно предотвратить запуск пользователями оболочек, загрузку библиотек в бинарных кодах или использование встроенного интерпретатора языка Perl, если они не должны этого делать. Производя обновление почтовой системы, проверьте новые параметры forbid_*, чтобы убедиться, что пользователи не имеют возможности экспериментировать со своими файлами **.forward**.

Транспортные механизмы

Маршрутизаторы решают, куда должны следовать сообщения, а транспортные механизмы выполняют их фактическое перемещение в пункт назначения. Локальные транспорты обычно добавляют сообщения в файл, передают их локальной программе или обмениваются информацией по протоколу LMTP с серверами IMAP. Удаленные транспорты обмениваются информацией со своими партнерами в Интернете по протоколу SMTP.

В агенте Exim существует пять транспортных механизмов: `appendfile`, `lmtp`, `smtp`, `autoreply` и `pipe`; мы опишем транспортные механизмы `appendfile` и `smtp`. Транспорт `autoreply` обычно используется для отправки сообщений автоответчика, а транспорт `pipe` передает сообщения на вход команд через канал UNIX. Как и при работе с маршрутизаторами, необходимо определить экземпляры транспортов, причем желательно иметь несколько экземпляров транспорта одного и того же типа. Порядок важен для маршрутизаторов, но не для транспортных механизмов.

Транспортный механизм `appendfile`

Драйвер `appendfile` сохраняет сообщения в формате **`mbox`**, **`mbx`**, **`Maildir`** или **`mailstore`** в заданном файле или каталоге. Компилируя программу Exim, необходимо включать соответствующие форматы почтового ящика; по умолчанию они закомментированы в файле **`EDITME`**. В следующем примере определен транспорт `my_local_delivery` (экземпляр транспорта `appendfile`), который упоминался в определении экземпляра маршрутизатора `localusers` в одном из предыдущих примеров.

```
my_local_delivery:
   driver = appendfile
   file = /var/mail/$local_part
   delivery_date_add
   envelope_to_add
   return_path_add
   group = mail
   mode = 0660
```

Строки `*_add` добавляют заголовки в сообщение. Разделы `group` и `mode` гарантируют, что транспортный агент может записывать данные в файл.

Транспортный механизм `smtp`

Это основной компонент всей почтовой системы. Здесь мы определили два экземпляра — для стандартного SMPT-порта (25) и для порта подачи почты (587).

```
my_remote_delivery:
   driver = smtp

my_remote_delivery_port587:
  driver = smtp
  port = 587
  headers_add = X-processed-by: MACRO_HEADER port 587
```

Второй экземпляр, `my_remote_delivery_port587`, задает порт и указывает, что в сообщение должен быть добавлен заголовок, включающий индикацию исходящего порта. Кроме того, где-то в файле конфигурации должен быть определен макрос `MACRO_HEADER`.

Конфигурация `retry`

В файле конфигурации должен существовать раздел `retry`, иначе программа Exim никогда не будет повторять попытку доставить почту, которая не была доставлена с пер-

вого раза. Можно указать три временных интервала, каждый из которых длиннее предыдущего. После истечения последнего интервала сообщения будут возвращены обратно отправителю как недоставленные.

В инструкциях `retry` для обозначения минут, часов, дней и недель используются суффиксы `m`, `h`, `d` и `w`. Для разных хостов и доменов можно задавать разные интервалы.

Вот как выглядит раздел `retry`.

```
begin retry
    *      *     F, 2h, 15m;     F, 24h, 1h;     F, 4d, 6h
```

Этот пример означает следующее: "Для любого домена доставку почты на временно недоступный адрес следует повторять каждые 15 минут в течение двух часов, каждый час в течение следующих 24 часов, а затем каждые 6 часов в течение 4 дней и в заключение отправить сообщение обратно как недоставленное".

Конфигурация перезаписи

Раздел перезаписи в файле конфигурации начинается словами `begin rewrite`. Он используется для исправления адресов, а не для перенаправления сообщений. Например, его можно использовать для исходящих адресов.

- Для того чтобы точно указать, что сообщение исходит из вашего домена, а не от индивидуальных хостов.
- Для отображения имен пользователей в стандартном формате, например *Имя.Фамилия*.

Перезапись не следует применять для адресов входящей почты.

Функция локального сканирования

Если вы хотите еще более точно настроить программу **exim**, например, чтобы фильтровать самые современные и самые распространенные вирусы, можете написать функцию на языке C, реализующую ваш собственный алгоритм сканирования, и инсталлировать ее в разделе `local_scan` файла конфигурации. Детали и примеры описаны в документации программы Exim.

Регистрация

По умолчанию программа Exim записывает три разных регистрационных файла: главный журнал, журнал отказов и журнал тревоги. Каждая запись в журнале содержит время ее создания. Местоположение регистрационных файлов задается в файле **EDITME** (до компиляции программы **exim**) или в файле конфигурации времени выполнения с помощью параметра `log_file_path`. По умолчанию файлы регистрации хранятся в каталоге **/var/spool/exim/log**.

Параметр `log_file_path` может принимать до двух значений, разделенных двоеточием. Каждое значение должно быть либо ключевым словом `syslog`, либо абсолютным путем со встроенными символами %, вместо которых подставляются имена **main**, **reject** и **panic**. Например, инструкция

```
log_file_path = syslog : /var/log/exim_%s
```

устанавливает, что в качестве регистрационных файлов должны использоваться системный журнал (с функцией "`mail`") и файлы **exim_main**, **exim_reject** и **exim_panic** в каталоге **/var/log**. Программа Exim передает записи журнала **main** в системный жур-

нал в качестве первоочередной информации, записи **reject** — в качестве первоочередных уведомлений, а записи **panic** — в качестве первоочередных сигналов.

Журнал **main** содержит по одной строке для поступления и доставки каждого сообщения. Отчет об информации, содержащейся в этом журнале, можно сгенерировать с помощью сценария **eximstats** на языке Perl, включенного в дистрибутивный пакет Exim. Записи из журнала **reject** хранят информацию о сообщениях, которые были отклонены из-за нарушения правил, — злонамеренные программы, спам и пр. Этот журнал содержит итоговую строку для сообщения из журнала **main**, а также исходные заголовки отклоненных сообщений. Если правила были изменены, проверьте журнал **reject**, чтобы убедиться, что все в порядке.

Журнал **panic** предназначен для регистрации серьезных ошибок программного обеспечения; программа **exim** записывает в него информацию непосредственно перед тем, как завершить работу. Если проблемы не возникли, журнал **panic** существовать не должен. Попросите утилиту **cron** проверить, существует ли файл **panic**, устраните проблему, вызвавшую тревогу, а затем удалите этот файл. Программа **exim** заново создаст его, если снова возникнет тревожная ситуация. При отладке можно увеличить объем и расширить набор типов регистрируемых данных с помощью параметра `log_selector`. Рассмотрим пример.

```
log_selector = +smtp_connection +snmp_incomplete_transaction +...
```

Категории регистрируемых данных могут затем включаться или исключаться с помощью механизма `log_selector`, описанного в спецификации программы Exim в разделе "Log files". Существует примерно 35 возможностей, включая `+all`, которая реально переполнит информацией ваши диски!

Кроме того, программа **exim** хранит временный регистрационный файл для каждого обработанного сообщения. Он называется по идентификатору сообщения и находится в каталоге **/var/spool/exim/msglog**. Если возникают проблемы с конкретным пунктом назначения, следует проверить информацию в этом каталоге.

Отладка

Программа Exim имеет мощные средства для отладки. Вы можете управлять объемом информации, которую хотите видеть для каждой настраиваемой возможности. Команда **exim -d** заставляет программу **exim** переключиться в режим отладки, в котором она работает на переднем плане и не отключается от терминала. К параметру **-d** можно добавить дополнительные категории отладки, вставляя символы **+** или **-** перед названием категории. Например, параметры **-d+expand+acl** предусматривают обычный вывод информации об отладке, который сопровождается дополнительными деталями, касающимися расширений строк и интерпретации списков управления доступом. (Эти две категории являются самыми проблемными.) Для отладочной информации можно настроить более 30 категорий; их полный список находится на соответствующей справочной странице.

При отладке почтовой системы обычно запускают почтовый транспортный агент на нестандартном порту, а затем обращаются к нему с помощью команды **telnet**. Например, для того чтобы запустить программу **exim** в режиме демона, прослушивающего порт 26, и включить вывод информации об отладке, можно использовать следующую команду.

```
$ sudo exim -d -oX 26 -bd
```

Затем можно подключиться через **telnet** к порту 26 и набрать команды SMTP, пытаясь воспроизвести проблемы, решение которых следует отладить.

В качестве альтернативы можно использовать команду **swaks**, которая вступит в обмен информацией с протоколом SMTP. Эта команда запускает сценарий на языке Perl, ускоряющий и облегчающий отладку протокола SMTP. Инструкция **swaks --help** выводит на экране короткое описание, а полную информацию можно найти на сайте `jetmore.org/john/code/swaks`.

Если в ваших регистрационных файлах фиксируются тайм-ауты примерно каждые 30 с, следует подозревать проблемы в системе DNS.

18.10. Почтовый агент Postfix

Постфикс — популярная альтернатива программе **sendmail**. Виетс Венема (Wietse Venema) начал работу над проектом Postfix в 1996 году, когда проводил годичный творческий отпуск в Исследовательском центре им. Т. Дж. Уотсона компании IBM (T. J. Watson Research Center), и до сих пор активно работает над ним. Целью разработки программы Postfix является не только безопасность (хотя это ее первоочередная и самая главная цель!), но и реализация политики распространения открытого программного обеспечения, высокое быстродействие, надежность и гибкость. Все основные дистрибутивные пакеты системы Linux включают Postfix, а начиная с версии 10.3 система Mac OS X по умолчанию оснащается почтовой системой Postfix, а не **sendmail**.

🕮 Более подробно регулярные выражения описаны в разделе 7.3.

О системе Postfix следует знать два самых важных факта: во-первых, она поставляется в практически работоспособном виде (простейшие файлы конфигурации состоят из одной-двух строк) и, во-вторых, эффективно использует ассоциативные массивы регулярных выражений для фильтрации почты, особенно в сочетании с библиотекой PCRE (Perl Compatible Regular Expression). Система Postfix совместима с программой **sendmail** в том смысле, что файлы **aliases** и **.forward** системы Postfix имеют тот же формат и семантику, что и аналогичные файлы в системе **sendmail**.

Система Postfix поддерживает протокол ESMTP. Кроме того, поддерживаются виртуальные домены и фильтация спама. Для перезаписи адресов система Postfix использует поиск в таблицах, которые могут быть записаны в обычных файлах или в форматах Berkeley DB, DBM, LDAP, NIS, NetInfo и SQL.

Архитектура системы Postfix

Система Postfix состоит из нескольких небольших взаимодействующих программ, посылающих сообщения по сети, получающих сообщения, выполняющих локальную доставку почты и т.д. Взаимодействие между ними осуществляется через сокеты локальных доменов или именованные каналы FIFO. Эта архитектура довольно сильно отличается от архитектуры программ **sendmail** и Exim, в которых основную работу выполняет большая монолитная программа.

Запуск системы Postfix и слежение за всеми ее процессами выполняет программа **master**. В ее конфигурационном файле, **master.cf**, перечислены все вспомогательные программы, а также информация о том, как они должны запускаться. Значения, по умолчанию установленные в этом файле, удовлетворяют большую часть потребностей пользователей, так что изощряться не обязательно. В основном, приходится отключать ненужные программы, например демон **smtpd**, когда клиенту не требуется прослушивать порт SMTP.

Самые важные серверные программы, вовлеченные в доставку электронной почты, показаны на рис. 18.2.

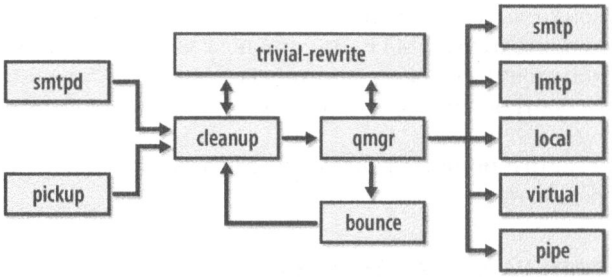


Рис. 18.2. Серверные программы системы Postfix

Получение почты

Программа **smtpd** получает почту, поступающую в систему по протоколу SMTP. Она также проверяет, имеют ли право клиенты, установившие соединение, отправлять почту, которую они пытаются доставить. Если электронная почта послана локально с помощью программы **/usr/lib/sendmail**, то файл записывается в каталог **/var/spool/postfix/maildrop**. Этот каталог периодически сканируется программой **pickup**, обрабатывающей любой найденный новый файл.

Вся входящая почта проходит через программу **cleanup**, которая добавляет пропущенные заголовки и перезаписывает адреса в соответствии с таблицами `canonical` и `virtual`. Прежде чем поместить сообщение в очередь входящих сообщений, программа **cleanup** передает его программе **trivial-rewrite**, которая выполняет небольшое редактирование адресов, например добавляет почтовый домен к не полностью заданным адресам.

Управление почтовыми очередями ожидания

Программа **qmgr** управляет пятью очередями, содержащими почту, которая ожидает доставки.

- `incoming` — поступающая почта.
- `active` — доставляемая почта.
- `deferred` — почта, доставка которой завершилась неудачей.
- `hold` — почта, заблокированная в очереди администратором.
- `corrupt` — почта, которую невозможно прочитать или проанализировать.

Менеджер очереди обычно выбирает следующее сообщение для обработки, руководствуясь простой стратегией FIFO, но помимо нее он также поддерживает сложный алгоритм приоритетного обслуживания, который среди всей массы сообщений отдает предпочтение сообщениям, направленным избранным получателям.

Для того чтобы не перегружать хост, предназначенный для получения почты, особенно после восстановления его работы, система Postfix использует алгоритм медленного пуска, управляющий скоростью доставки почты.

Отложенная почта помечается специальной временной меткой о необходимости повторной доставки. Ее длительность увеличивается по экспоненциальному закону, что позволяет не тратить лишние ресурсы на недоставленные недавно сообщения. Избежать излишних попыток доставить почту, которую пока что невозможно доставить, позволяет кеширование состояния.

Отправка сообщений

Программа **qmgr** с помощью программы **trivial-rewrite** решает, куда следует послать сообщение. Вместо решений, принимаемых программой **trivial-rewrite**, можно использовать поиск в таблицах (`transport_maps`).

Доставка почты удаленным серверам по протоколу SMTP выполняется программой **smtp**. Программа **lmtp** доставляет почту, используя протокол LMTP (Local Mail Transfer Protocol), определенный в документе RFC2033. Протокол LMTP основан на протоколе SMTP, но был модифицирован так, чтобы почтовый сервер не управлял почтовой очередью. Этот обработчик почты очень полезен для доставки сообщений на почтовые серверы, обслуживающие клиентские ящики, такие как Cyrus IMAP.

Программа **local** предназначена для локальной доставки электронной почты. Она находит адреса в таблице **aliases** и следует инструкциям, указанным в файлах получателя **.forward**. Сообщения пересылаются на другой адрес, передаются внешней программе для обработки или сохраняются в почтовых папках пользователя.

Программа **virtual** доставляет почту в "виртуальные почтовые ящики", т.е. в почтовые ящики, не связанные с локальной учетной записью в системе Linux, но имеющие корректные адреса. И, наконец, программа **pipe** выполняет доставку посредством внешних программ.

Безопасность

Система Postfix обеспечивает безопасность на нескольких уровнях. Большинство серверных программ системы Postfix могут выполняться в окружении, созданном с помощью команды **chroot**. Это отдельные программы, не имеющие отношений типа "предок-потомок". Ни одна из них не имеет механизма `setuid`. Каталог `maildrop` разрешен по записи для группы `postdrop`, для которой в программе **postdrop** установлен идентификатор `setgid`.

Команды и документация системы Postfix

Взаимодействие пользователя с системой обеспечивают несколько утилит, запускаемых из командной строки.

- **postalias** – создает, модифицирует и ставит в очередь таблицы псевдонимов.
- **postcat** – отображает содержимое файлов очереди.
- **postconf** – отображает и редактирует главный файл конфигурации **main.cf**.
- **postfix** – запускает и останавливает почтовую систему (должна запускаться с правами пользователя `root`).
- **postmap** – создает, модифицирует или запрашивает таблицы поиска.
- **postsuper** – управляет почтовыми очередями.
- **sendmail**, **mailq**, **newaliases** – это замененные команды, совместимые с системой **sendmail**.

Дистрибутивный пакет Postfix содержит набор справочных страниц, описывающих все программы и их параметры. Документация, размещенная на сайте `postfix.org`, объясняет, как конфигурировать разные аспекты программы Postfix и управлять ими. Эти документы также входят в дистрибутивный пакет Postfix и находятся в каталоге **README_FILES**.

Конфигурация системы Postfix

Файл **main.cf** – главный конфигурационный файл системы Postfix. Файл **master.cf** конфигурирует серверные программы. Он также определяет разные таблицы поиска, на которые ссылается файл **main.cf**, и поддерживает разные типы служебных отображений.

Справочная страница **postconf**(5) описывает каждый параметр, заданный в файле **main.cf**. Существует также программа **postconf**, которая с помощью команды **man postconf** позволяет получить справочную страницу, вместо страницы **postconf**(5). Для того чтобы получить правильную версию справочной страницы, следует использовать команду **man -s 5 postconf**.

Язык конфигурирования системы Postfix выглядит как ряд комментариев оболочки **sh** и инструкций присваивания. В определении одной переменной можно ссылаться на другие переменные с помощью префикса `$`. Определения переменных заносятся в память сразу, как только они обнаруживаются в файле конфигурации; они не раскрываются, пока не будут использованы, и в это время могут производиться любые подстановки.

Можно создавать новые переменные, присваивая им значения. При выборе имен следует быть осторожным, чтобы не создать конфликт с существующими переменными конфигурации.

Все файлы конфигурации системы Postfix, включая таблицы поиска, интерпретируют строки, начинающиеся с пробела, как продолжение предыдущей строки. Это позволяет создавать очень удобные для чтения файлы конфигурации, но при этом новую строку приходится начинать строго с первой позиции.

Что должно находиться в файле ***main.cf***

В файле **main.cf** можно задать более 500 параметров. Однако для среднестатистического почтового сервера необходимы только некоторые из них. Автор системы Postfix настоятельно рекомендует включать в свои файлы конфигурации только те параметры, которые имеют значения, отличные от заданных по умолчанию. Таким образом, если в будущем значение параметра, заданное по умолчанию, изменится, ваша конфигурация учтет новое значение автоматически.

Образец файла **main.cf**, поставляемый в дистрибутивном пакете, содержит много закомментированных параметров, а также их короткое описание. Исходную версию файла лучше всего оставить в качестве образца. Начните создание собственной конфигурации с пустого файла, чтобы ваши установки не затерялись среди многочисленных комментариев.

Основные установки

Начнем с самой простой конфигурации: пустого файла. Как это ни удивительно, это вполне допустимая конфигурация системы Postfix. Результатом такой конфигурации является почтовый сервер, доставляющий электронную почту локально внутри того же самого домена, в котором находится локальная хост-система, и посылающий все сообщения на нелокальные адреса непосредственно на соответствующий удаленный сервер.

Нулевой клиент

Еще одна простая конфигурация называется *нулевым клиентом*; иначе говоря, система, не выполняющая локальной доставки никаких электронных сообщений и перенаправляющая внешнюю почту на предназначенный для этого центральный сервер. Для реализации этой конфигурации определим несколько параметров: параметр `mydomain`, задающий доменную часть имени хоста, и `myorigin`, представляющий собой имя по-

чтового домена, добавляемое в не полностью заданные адреса. Если эти два параметра совпадают, то можно написать, например, следующие строки.

```
mydomain = cs.colorado.edu
myorigin = $mydomain
```

Еще один параметр, который необходимо определить, — `mydestination`, задающий локальные почтовые домены. Если в адресе получателя сообщения имя почтового домена совпадает со значением параметра `mydestination`, сообщение доставляется с помощью программы **`local`** соответствующему пользователю (в предположении, что не найдены ни релевантный псевдоним, ни файл **`.forward`**). Если в параметре `mydestination` указаны имена нескольких доменов, то все они считаются псевдонимами одного и того же домена.

Для нулевого клиента локальная доставка не предусмотрена, поэтому этот параметр должен оставаться пустым.

```
mydestination =
```

В заключение, параметр `relayhost` означает, что система Postfix должна посылать все нелокальные сообщения на указанный хост, а не непосредственно в их заданные явно пункты назначения.

```
relayhost = [mail.cs.colorado.edu]
```

Квадратные скобки означают, что система Postfix должна интерпретировать указанную строку как имя хоста (запись `A` в DNS), а не как имя домена (запись `MX` в DNS).

Поскольку нулевые клиенты не должны получать почту от других систем, в конфигурации нулевого клиента осталось только закомментировать строку **`smtpd`** в файле **`master.cf`**. Это изменение не позволяет системе Postfix вообще запускать программу **`smtpd`**. Итак, с помощью всего нескольких строк мы определили полнофункциональный нулевой клиент!

Для "реального" почтового сервера необходимо задать немного больше параметров, а также некоторые таблицы отображений. Эти темы рассмотрим в следующих подразделах.

Использование утилиты `postconf`

Утилита **`postconf`** — это удобный инструмент, помогающий конфигурировать систему Postfix. Если запустить ее без аргументов, она выводит на список всех параметров, которые она сконфигурировала к данному моменту. Если в качестве аргумента указать конкретный параметр, утилита **`postconf`** выведет его значение.

Флаг **`-d`** заставляет утилиту **`postconf`** выводить значения параметров, заданные по умолчанию, а не текущие значения, указанные в конфигурации. Рассмотрим пример.

```
$ postconf mydestination
mydestination =
$ postconf -d mydestination
mydestination = $myhostname, localhost.$mydomain, localhost
```

Еще один полезный флаг **`-n`** заставляет утилиту **`postconf`** выводить только те параметры, которые отличаются от заданных по умолчанию. Если вам необходима помощь от участников списка рассылки Postfix, именно эту информацию следует указать в вашем электронном письме.

Таблицы поиска

Многие аспекты функционирования системы Postfix зависят от использования таблиц поиска, которые могут отображать ключи в значения или реализовать простые списки. Например, настройки по умолчанию для таблицы `alias_maps` имеют следующий вид.

```
alias_maps = dbm:/etc/mail/aliases
```

Источники данных задаются с помощью обозначения *тип:путь*. Несколько значений можно разделять запятыми, пробелами или и тем, и другим. Доступные источники данных перечислены в табл. 18.19; команда **postconf -m** также выводит эту информацию.

Таблица 18.19. Источники информации для таблиц поиска в системе Postfix

Тип	Описание
cidr	Сетевые адреса в формате CIDR
dbm/sdbm	Обычные файлы баз данных **dbm** или **gdbm**
hash/btree	Хеш-таблицы Berkeley DB или файл B-дерева (эквивалент **dbm**)
ldap	Служба каталогов LDAP
mysql	База данных MySQL
nis	Служба каталогов NIS
pcre	Регулярные выражения, совместимые с языком Perl
pgsql	База данных PostgreSQL
proxy	Доступ через сервер **proxymap**, например, чтобы избежать использования команды **chroot**
regexp	Регулярные выражения POSIX
static	Возвращает значение, указанное как *путь*, независимо от ключа
unix	Файлы **/etc/passwd** и **/etc/group**; использует синтаксис NIS[a]

[a] **unix:passwd.byname** — это файл **passwd**, а **unix:group.byname** — файл **group**.

Типы dbm и sdbm следует использовать только для обеспечения совместимости с обычной таблицей псевдонимов программы **sendmail**. Berkeley DB (hash) — более современная реализация; она безопаснее и быстрее. Ее совместимость — это не проблема, используйте следующие инструкции.

```
alias_database = hash:/etc/mail/aliases
alias_maps = hash:/etc/mail/aliases
```

Параметр alias_database задает таблицу, которая перестраивается с помощью команды **newaliases** и должна соответствовать таблице, заданной параметром alias_maps. Причина, по которой используются два параметра, заключается в том, что таблица alias_maps может содержать источники, не являющиеся базами данных, например mysql или nis, которые не требуют перестройки.

Все таблицы баз данных (dbm, sdbm, hash и btree) представляют собой текстовые файлы, которые компилируются в эффективный бинарный формат, предназначенный для поиска. Синтаксис этих текстовых файлов похож на синтаксис файлов конфигурации тем, что в них также используются комментарии и продолжения строк. Записи задаются как простые пары "ключ-значение", разделенные пробелами, за исключением таблиц псевдонимов, в которых используются двоеточия после ключей, чтобы обеспечить совместимость с программой **sendmail**. Например, следующие строки являются характерными для таблицы псевдонимов.

```
postmaster: david, tobias
webmaster: evi
```

В следующем примере задается таблица доступа для ретрансляции почты от любого клиента, имя хоста которого заканчивается строкой cs.colorado.edu.

```
.cs.colorado.edu    OK
```

Текстовые файлы компилируются в бинарный формат с помощью команд **postmap** (для обычных таблиц) и **postalias** (для таблиц псевдонимов). Спецификация таблицы (включая ее тип) должна передаваться как первый аргумент. Рассмотрим пример.

```
$ sudo postmap hash:/etc/postfix/access
```

Команда **postmap** может также запрашивать значения в таблице поиска (нет совпадения — нет вывода на экран).

```
$ postmap -q blabla hash:/etc/postfix/access
$ postmap -q .cs.colorado.edu hash:/etc/postfix/access
OK
```

Локальная доставка

Программа **local** доставляет почту локальным получателям. Кроме того, она обрабатывает локальные псевдонимы. Например, если параметр `mydestination` задан равным `cs.colorado.edu`, а почта поступает на адрес `evi@cs.colorado.edu`, то программа **local** сначала проверяет таблицы `alias_maps`, а затем рекурсивно заменяет все соответствующие записи.

Если в таблице нет соответствия псевдонимов, программа **local** просматривает файл **.forward** в рабочем каталоге пользователя `evi` и следует инструкциям, заданным в этом файле, если он существует. (Синтаксис совпадает с правой частью таблицы псевдонимов.) В заключение, если файл **.forward** не найден, то электронное сообщение доставляется в локальный почтовый ящик пользователя `evi`.

По умолчанию программа **local** записывает данные в файлы, имеющие формат **mbox** и находящиеся в каталоге **/var/mail**. Эти настройки можно изменить с помощью параметров, указанных в табл. 18.20.

Таблица 18.20. Параметры для доставки почты в локальный почтовый ящик (задаются в файле `main.cf`)

Параметр	Описание
`home_mailbox`	Доставляет почту в каталог *~пользователь* по заданному относительному пути
`mail_spool_directory`	Доставляет почту в центральный каталог, обслуживающий всех пользователей
`mailbox_command`	Доставляет почту с помощью внешней программы, обычно **procmail**
`mailbox_transport`	Доставляет почту с помощью службы, определенной в файле **master.cf** [a]
`recipient_delimiter`	Разрешает расширенные имена (см. описание ниже)

[a] Этот параметр взаимодействует с почтовыми серверами, такими как Cyrus **imapd**.

Параметры `mail_spool_directory` и `home_mailbox` обычно генерируют почтовые ящики в формате **mbox**, но они также могут создавать почтовые ящики **Maildir**. Для того чтобы сделать это, необходимо добавить косую черту в конце имени пути.

Если параметр `recipient_delimiter` равен `+`, то почта, адресованная `evi+whatever@cs.colorado.edu`, принимается для доставки на учетную запись `evi`. С помощью этой функциональной возможности пользователь может создавать специальные адреса и сортировать свою почту по адресу получателя. Система Postfix сначала пытается выполнить поиск по полному адресу, и только если поиск оказался безуспешным, она разбирает дополнительные компоненты и "возвращается" к базовому адресу. Система Postfix также ищет соответствующий файл пересылки **.forward+***что_угодно* для дальнейшей обработки псевдонимов.

Виртуальные домены

Есть три возможности организовать почтовый домен на почтовом сервере Postfix.

- Указать домен в параметре `mydestination`. Доставка выполняется, как описано выше: псевдонимы раскрываются и почта доставляется на соответствующие учетные записи.
- Указать домен в параметре `virtual_alias_domains`. Это позволяет оснастить домен собственным пространством адресов, которое не зависит от системных учетных записей пользователей. Все адреса внутри домена должны отображаться (с помощью ассоциативных массивов) во внешние реальные адреса.
- Указать домен в параметре `virtual_mailbox_domains`. Как и в случае параметра `virtual_alias_domains`, домен имеет собственное пространство имен. Все почтовые ящики должны находиться в указанном каталоге.

Домен можно указать только в одном из трех описанных параметров. Решение следует хорошенько обдумать, потому что от него зависят многие элементы конфигурации. Мы уже описывали применение метода `mydestination`. Рассмотрим остальные возможности.

Псевдонимы виртуальных доменов

Если домен указан как значение параметра `virtual_alias_domains`, то почта, получаемая для этого домена, будет приниматься системой Postfix и должна направляться реальному получателю либо на локальном компьютере, либо где-то еще.

Перенаправление адресов в виртуальный домен должно определяться в таблице поиска, указанной в параметре `virtual_alias_maps`. Записи этой таблицы в левой части содержат адрес виртуального домена, а в правой — реальный адрес получателя.

Не полностью определенное имя в правой части интерпретируется как локальное имя пользователя.

Рассмотрим следующий пример из файла **`main.cf`**.

```
myorigin = cs.colorado.edu
mydestination = cs.colorado.edu
virtual_alias_domains = admin.com
virtual_alias_maps = hash:/etc/mail/admin.com/virtual
```

В файле **`/etc/mail/admin.com/virtual`** могут содержаться следующие строки.

```
postmaster@admin.com    evi, david@admin.com
david@admin.com         david@schweikert.ch
evi@admin.com           evi
```

Почта, направляемая на адрес `evi@admin.com`, будет перенаправлена на адрес `evi@cs.colorado.edu` (добавлен параметр `myorigin`) и в итоге будет доставлена в почтовый ящик пользователя `evi`, потому что имя домена `cs.colorado.edu` указано как значение параметра `mydestination`.

Определения могут быть рекурсивными: правая часть может содержать адреса, которые в дальнейшем определяются в левой части. Обратите внимание на то, что правая часть может быть только списком адресов. Если необходимо выполнить внешнюю программу или использовать подстановку файлов с помощью инструкции `:include:`, то нужно перенаправить почту на псевдоним, который впоследствии может быть раскрыт в соответствии с потребностями пользователя.

Для того чтобы хранить всю информацию в одном файле, можно присвоить параметру `virtual_alias_domains` имя той же таблицы поиска, которая задана параметром `virtual_alias_maps`, и внести в эту таблицу специальную запись, которая отмечала

бы этот файл как псевдоним виртуального домена. В файле **main.cf** соответствующие строки выглядели бы следующим образом.

```
virtual_alias_domains = $virtual_alias_maps
virtual_alias_maps = hash:/etc/mail/admin.com/virtual
```

Файл **/etc/mail/admin.com/virtual** содержал бы следующие данные.

```
admin.com                notused
postmaster@admin.com     evi, david@admin.com
...
```

Правая часть записи для почтового домена (admin.com) на самом деле никогда не используется; наличия имени admin.com в этой таблице в качестве независимого поля уже достаточно для того, чтобы система Postfix рассматривала его как псевдоним виртуального домена.

Виртуальные почтовые домены

Домены, перечисленные в списке значений параметра virtual_mailbox_domains, аналогичны локальным доменам, но список пользователей и их соответствующие почтовые ящики должны управляться независимо от системных учетных записей пользователей.

Параметр virtual_mailbox_maps указывает на таблицу, в которой перечислены все корректные пользователи в домене. Соответствующее отображение имеет следующий вид.

```
user@domain /путь/к/почтовому_ящику
```

Если имя пути заканчивается косой чертой, почтовые ящики хранятся в формате **Maildir**. Значение параметра virtual_mailbox_base всегда является префиксом заданных путей.

Часто возникает необходимость заменить псевдонимом некоторые адреса в виртуальных почтовых доменах. Для этого следует использовать параметр virtual_alias_map. Рассмотрим полный пример. В файле **main.cf**

```
virtual_mailbox_domains = admin.com
virtual_mailbox_base = /var/mail/virtual
virtual_mailbox_maps = hash:/etc/mail/admin.com/vmailboxes
virtual_alias_maps = hash:/etc/mail/admin.com/valiases
```

имеется ссылка на файл **/etc/mail/admin.com/vmailboxes**, который может содержать следующую запись.

```
evi@admin.com            nemeth/evi/
```

Файл **/etc/mail/admin.com/valiases** может содержать следующую строку.

```
postmaster@admin.com     evi@admin.com
```

Отображения виртуальных псевдонимов можно применять даже к адресам, которые не являются псевдонимами виртуальных доменов. Отображения виртуальных псевдонимов позволяют перенаправлять любой адрес от любого домена независимо от типа домена (канонического, виртуального псевдонима или виртуального почтового ящика). Поскольку пути к почтовому ящику могут находиться только в правой части отображения виртуального почтового ящика, использование этого механизма является единственным способом задать псевдоним для такого домена.

Управление доступом

Почтовые серверы должны ретранслировать почту для третьих лиц только со стороны доверенных клиентов. Если почтовый сервер перенаправляет почту от неизвестных

клиентов на другие серверы, то возникает так называемая *открытая ретрансляция* (open relay), а это плохо. Более подробно открытая ретрансляция описана в разделе 18.8.

К счастью, система Postfix по умолчанию не допускает открытой ретрансляции. На самом деле она имеет довольно строгие настройки, установленные по умолчанию; их приходится скорее смягчать, чем ужесточать. Управление доступом для транзакций протокола SMTP конфигурируется в системе Postfix с помощью "списков ограничения доступа". Параметры, указанные в табл. 18.21, управляют тем, что следует проверять на разных этапах сеанса SMTP.

Таблица 18.21. Параметры системы Postfix для ограничения доступа в рамках протокола SMTP

Параметр	Когда применяется
`smtpd_client_restrictions`	По запросу на соединение
`smtpd_data_restrictions`	К команде **DATA** (тело почты)
`smtpd_etrn_restrictions`	К команде **ETRN** [a]
`smtpd_helo_restrictions`	К команде **HELO**/**EHLO** (начало сессии)
`smtpd_recipient_restrictions`	К команде **RCPT TO** (спецификация получателя)
`smtpd_sender_restrictions`	К команде **MAIL FROM** (спецификация отправителя)

[a] Это специальная команда, используемая для повторной отправки сообщений из очереди.

Наиболее важным параметром является `smtpd_recipient_restrictions`, поскольку управление доступом легче всего реализовать, если адрес получателя известен и можно определить, локальный он или нет. Все другие параметры из табл. 18.21 в конфигурации, заданной по умолчанию, остаются пустыми. По умолчанию параметр `smtpd_recipient_restrictions` имеет следующее значение.

```
smtpd_recipient_restrictions = permit_mynetworks,
    reject_unauth_destination
```

Каждое из указанных ограничений проверяется по очереди, пока не будет принято определенное решение о том, что делать с почтой. Общие ограничения перечислены в табл. 18.22.

Таблица 18.22. Общие ограничения доступа в системе Postfix

Ограничение	Функция
`check_client_access`	Проверяет адрес хоста клиента, просматривая таблицу поиска
`check_recipient_access`	Проверяет почтовый адрес получателя, просматривая таблицу поиска
`permit_mynetworks`	Предоставляет доступ адресам, перечисленным в списке `mynetworks`
`reject_unauth_destination`	Отклоняет почту для нелокальных получателей; отменяет ретрансляцию

С помощью этих ограничений можно проверить все, а не только специфическую информацию наподобие адреса отправителя в параметре `smtpd_sender_restrictions`. Следовательно, для простоты можно наложить все ограничения, используя только один параметр, которым должен быть параметр `smtpd_recipient_restrictions`, поскольку только он позволяет проверять все (за исключением части `DATA`).

Параметр `smtpd_recipient_restriction` позволяет также проверять ретранслируемую почту. Ограничения, заданные параметром `reject_unauth_destination`, следует сохранить и с осторожностью выбирать "разрешающие" ограничения, предшествующие ему.

Таблицы доступа

Каждое ограничение возвращает одно действие, указанное в табл. 18.23. Таблицы доступа используются в ограничениях `check_client_access` и `check_recipient_access` для выбора действия, зависящего от адреса клиентского хоста или адреса получателя соответственно.

Таблица 18.23. Действия для таблицы доступа

Действие	Описание
`4nn` *текст*	Возвращает код временной ошибки `4nn` и сообщение *text*
`5nn` *текст*	Возвращает код постоянной ошибки `5nn` и сообщение *text*
`DEFER_IF_PERMIT`	Если результатом оценки ограничения является действие `PERMIT`, изменяет его на временную ошибку
`DEFER_IF_REJECT`	Если результатом оценки ограничения является действие `REJECT`, изменяет его на временную ошибку
`DISCARD`	Принимает сообщение, но скрытно удаляет его
`DUNNO`	Делает вид, что ключ не был найден; проверяет дальнейшие ограничения
`FILTER` *транспорт:пункт_назначения*	Пропускает почту через фильтр *транспорт:пункт_назначения*
`HOLD`	Блокирует почту в очереди
`OK`	Принимает почту
`PREPEND` *заголовок*	Добавляет заголовок в сообщение
`REDIRECT` *адрес*	Перенаправляет данное сообщение на конкретный адрес
`REJECT`	Отклоняет почту
`WARN` *сообщение*	Заносит указанное сообщение в журнал регистрации

В качестве примера предположим, что мы хотим разрешить ретрансляцию для всех компьютеров в домене `cs.colorado.edu` и отправку сообщения списку внутренней рассылки `newsletter@cs.colorado.edu` только доверенным клиентам. Эти правила можно реализовать с помощью следующих строк в файле **`main.cf`**.

```
smtpd_recipient_restrictions =
          permit_mynetworks
          check_client_access hash:/etc/postfix/relaying_access
          reject_unauth_destination
          check_recipient_access hash:/etc/postfix/restricted_recipients
```

Обратите внимание на то, что запятые являются необязательными, если указан список значений параметра.

В файл **`/etc/postfix/relaying_access`** следует внести следующую строку.

```
.cs.colorado.edu OK
```

В файл **`/etc/postfix/restricted_recipients`** следует внести такую строку.

```
newsletter@cs.colorado.edu REJECT Internal list
```

Текст, следующий за словом `REJECT`, является необязательной строкой, которая посылается клиенту вместе с кодом ошибки. Он сообщает отправителю, почему сообщение было отклонено.

Аутентификация клиентов и шифрование

Для пользователей, отправляющих сообщения из дома, обычно проще всего направлять исходящую почту через домашний почтовый сервер интернет-провайдера, незави-

симо от адреса отправителя сообщения. Большинство интернет-провайдеров доверяют своим непосредственным клиентам и разрешают ретрансляцию. Если эта конфигурация невозможна или используется система Sender ID или SPF, то следует убедиться, что мобильные пользователи, находящиеся вне сети, могут быть авторизованы для передачи сообщений демону **smtpd**.

Решить эту проблему можно, используя механизм SMTP AUTH непосредственно на уровне протокола SMTP. Для этого систему Postfix необходимо скомпилировать с библиотекой SASL. Затем эту функциональную возможность можно конфигурировать следующим образом.

```
smtpd_sasl_auth_enable = yes
smtpd_recipient_restrictions =
   permit_mynetworks
   permit_sasl_authenticated
   ...
```

Вам также понадобится поддержка зашифрованных соединений, чтобы избежать отправки паролей открытым текстом. Добавьте в файл **main.cf** строки, похожие на следующие.

```
smtpd_tls_security_level = may
smtpd_tls_auth_only = yes
smtpd_tls_loglevel = 1
smtpd_tls_received_header = yes
smtpd_tls_cert_file = /etc/certs/smtp.pem
smtpd_tls_key_file = $smtpd_tls_cert_file
smtpd_tls_protocols = !SSLv2
```

Необходимо также поместить правильно подписанный сертификат в файл **/etc/certs/smtp.pem**. Кроме того, рекомендуется подключить шифрование на исходящие соединения SMTP.

```
smtp_tls_security_level = may
smtp_tls_loglevel = 1
```

Отладка

Если в системе Postfix возникает проблема, в первую очередь следует проверить регистрационные файлы. Ответы на возможные вопросы, скорее всего, найдутся в этих файлах; просто надо их поискать. Каждая программа в системе Postfix обычно создает регистрационную запись для каждого обработанного сообщения. Например, ниже приведен пример регистрационных записей для исходящего сообщения.

```
Aug 18 22:41:33 nova postfix/pickup: 0E4A93688: uid=506
   from=<dws@ee.ethz.ch>
Aug 18 22:41:33 nova postfix/cleanup: 0E4A93688: message-id=
   <20040818204132.GA11444@ee.ethz.ch>
Aug 18 22:41:33 nova postfix/qmgr: 0E4A93688: from=<dws@ee.ethz.ch>,
   size=577,nrcpt=1 (queue active)
Aug 18 22:41:33 nova postfix/smtp: 0E4A93688:
   to=<evi@ee.ethz.ch>,relay=tardis.ee.ethz.ch[129.132.2.217],delay=0,
   status=sent (250 Ok: queued as 154D4D930B)

Aug 18 22:41:33 nova postfix/qmgr: 0E4A93688: removed
```

Легко видеть, что интересующая нас информация разбросана по нескольким строкам. Обратите внимание на то, что идентификатор 0E4A93688 встречается на каждой

строке: система Postfix присваивает идентификатор очереди сразу, как только сообщение поступает в почтовую систему, и никогда не изменяет его. Следовательно, при поиске регистрационных записей об истории сообщения в первую очередь следует определить идентификационный номер сообщения в очереди. Узнав его, легко применить команду **grep** ко всем релевантным записям.

Система Postfix делает подробные записи о замеченных ею проблемах. Однако иногда трудно найти важные строки среди тысяч сообщений о нормальном состоянии. В этом случае целесообразно использовать некоторые инструменты, перечисленные в разделе 10.6.

Просмотр очереди

Еще одно место, где следует искать решение проблем, — почтовая очередь. Как и в системе **sendmail**, команда **mailq** выводит на экран содержимое очереди. Его можно использовать для того, чтобы выяснить причину задержки отправки сообщения.

Полезным инструментом является также сценарий **qshape**, который поставляется с последними версиями системы Postfix. Он демонстрирует суммарные статистические показатели о содержимом очереди. Его вывод выглядит следующим образом.

```
$ sudo qshape deferred
                    T  5 10 20 40 80 160 320 640 1280 1280+
            TOTAL  78  0  0  0  7  3   3   2  12    2    49
          expn.com  34  0  0  0  0  0   0   0   9    0    25
    chinabank.ph     5  0  0  0  1  1   1   2   0    0     0
 prob-helper.biz     3  0  0  0  0  0   0   0   0    0     3
```

Сценарий **qshape** создает отчет о заданной очереди (в данном случае об очереди отложенных сообщений), упорядоченной по доменам получателя. В столбцах отчета указывается, сколько минут релевантное сообщение провело в очереди. Например, в отчете видно, что 25 сообщений, направленных в домен `expn.com`, провели в очереди более 1280 минут. Все пункты назначения в данном примере вызывают подозрения, что сообщения были посланы автоответчиком в ответ на спам.

Сценарий **qshape** также создает отчет в соответствии с доменами отправителей, если он запущен с флагом **-s**.

Мягкий рикошет

Если параметр `soft_bounce` установлен равным `yes`, система Postfix посылает сообщения о временных ошибках, а не только сообщения о постоянных ошибках, таких как "пользователь неизвестен" или "ретрансляция запрещена". Этот параметр предоставляет отличную возможность для тестирования; он позволяет отслеживать местонахождение сообщения после изменения конфигурации без риска потери законной электронной почты. В этом случае любой отказ в конце концов приводит к повторной попытке доставки.

Не забудьте отключить эту функциональную возможность, когда закончите тестирование, иначе попытки доставки отвергнутой почты будут повторяться снова и снова.

18.11. Литература

Для того чтобы не смешивать все ссылки в один список, мы упорядочили их по транспортным агентам и темам.

Литература по программе sendmail

- Costales B., Assmann C., Jansen G., Shapiro G. *Sendmail, 4th Edition.* — Sebastopol, CA: O'Reilly Media, 2007. Эта книга представляет собой внушительный том о конфигурации программы **sendmail** размером более 1300 страниц. Она содержит руководство системного администратора, а также полный список библиографических ссылок. Кроме того, существует электронный вариант этой книги. В коллектив авторов входят два ключевых разработчика программы **sendmail** (Claus и Greg), что является гарантией технической корректности и авторитетности книги.
- В разделе *Sendmail Installation and Operation Guide* изложены инструкции по инсталляции и хорошее описание файла конфигурации. Файл с этим разделом можно найти в подкаталоге **doc/op** дистрибутивного пакета **sendmail**. Этот документ является достаточно полным, и в сочетании с файлом **README** из каталога **cf** он дает конкретное представление о системе **sendmail**.
- Сайты `sendmail.org`, `sendmail.org/~ca` и `sendmail.org/~gshapiro` содержат документы, ответы на вопросы и справочные материалы по программе **sendmail**.

Литература о системе Exim

- Hazel P. The *Exim SMTP Mail Server: Official Guide for Release 4, 2nd Edition.* — Cambridge, UK: User Interface Technologies, Ltd., 2007.
- Hazel P. *Exim: The Mail Transfer Agent.* — Sebastapol, CA: O'Reilly Media, 2001.
- Meers J. *Getting started with EXIM.* — `exim-new-users.co.uk`, 2007.
- Спецификация системы Exim представляет собой определяющий документ о ее конфигурации. Он является исчерпывающим и обновляется с появлением каждой новой версии. Текстовая версия включена в файл **doc/spec.txt** из дистрибутивного пакета, а соответствующий файл PDF доступен на сайте `exim.org`. Кроме того, на этом веб-сайте можно найти несколько справочных документов.

Литература о системе Postfix

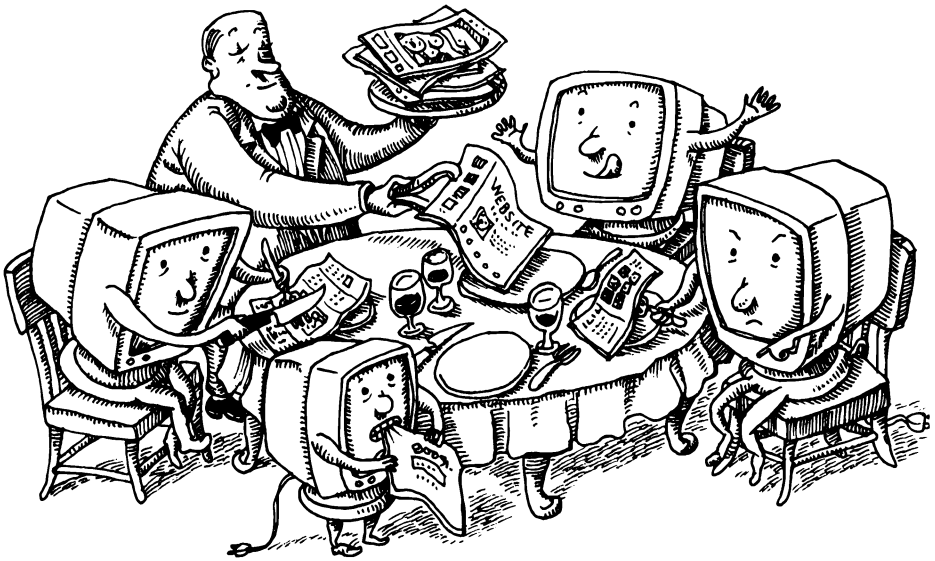
- Dent K.D. *Postfix: The Definitive Guide.* — Sebastopol, CA: O'Reilly Media, 2003.
- Hildebrandt R., Koetter P. *The Book of Postfix: State of the Art Message Transport.* — San Francisco, CA: No Starch Press, 2005. Эта книга лучше всех; она посвящает читателя во все тонкости конфигурации системы Postfix, даже в сложных средах. Авторы являются активными членами сообщества пользователей системы Postfix и регулярно участвуют в почтовых рассылках группы пользователей Postfix. К сожалению, эта книга не переиздается, но ее можно приобрести в букинистических магазинах.

Документы RFC

Текущими версиями документов RFC821 и RFC822 являются документы RFC5321 (обновлен в RFC7504) и RFC5322 (обновлен в RFC6854). В них определен протокол SMTP и форматы сообщений и адресов для электронной почты в Интернете. Документы RFC6531 и RFC6532 посвящены вопросам интернационализации адресов электронной почты. Существует около 90 документов RFC, связанных с электронной почтой. Их слишком много, чтобы перечислять в данной книге. Все эти документы можно найти на сайте `rfc-editor.org` с помощью поисковой системы.

Глава 19

Веб-хостинг



UNIX и Linux являются доминирующими платформами для обслуживания веб-приложений. По данным сайта `w3techs.com`, 67% ведущих веб-сайтов, занимающих первый миллион мест, обслуживаются операционной системой Linux или FreeBSD. Кроме того, программное обеспечение веб-серверов с открытым исходным кодом занимает более 80% рынка.

Как правило, веб-приложения не работают только в одной системе. Вместо этого создаются коллекции программных компонентов, распределенных в разных системах и взаимодействующих друг с другом, чтобы отвечать на запросы как можно быстрее и гибче. Каждая часть этой архитектуры должна быть устойчивой к сбоям сервера, нагрузкам, сетевым разделениям и целенаправленным атакам.

Удовлетворить эти потребности помогает облачная инфраструктура. Ее способность быстро обеспечивать пропускную способность в ответ на запросы идеально подходит для внезапных, а иногда и неожиданных приливов пользователей, которые возникают в Интернете. Кроме того, дополнительные службы облачных провайдеров включают в себя множество удобных решений, отвечающих общим требованиям, что значительно упрощает разработку, развертывание и работу веб-систем.

19.1. HTTP: протокол передачи гипертекста

HTTP — это основной сетевой протокол для связи в Интернете. За обманчиво простым фасадом запросов и ответов без фиксации состояния скрываются слои уточнений,

которые привносят как гибкость, так и сложность. Глубокое понимание протокола HTTP является основной обязанностью всех системных администраторов.

В своей простейшей форме HTTP — это клиент-серверный протокол с одним запросом и одним ответом. Клиенты, также называемые *пользовательскими агентами*, отправляют HTTP-серверу запросы на ресурсы. Серверы получают входящие запросы и обрабатывают их, получая файлы с локальных дисков, повторно отправляя запросы на другие серверы, запрашивая базы данных или выполняя любое количество других возможных вычислений. Типичный просмотр одной веб-страницы в Интернете предполагает десятки или сотни таких обменов.

Как и большинство интернет-протоколов, протокол HTTP также изменялся со временем, хотя и медленно. Главенствующее место этого протокола в современном Интернете делает его обновления критически важными. Официальные пересмотры — это многочисленные заседания комитетов, переписка с участниками переговоров, периоды публичного оценивания и маневрирование заинтересованных сторон, имеющих противоречивые интересы. Во время длительных периодов ожидания официальных версий, зафиксированных в спецификациях RFC, по необходимости появляются неофициальные расширения протокола, которые становятся повсеместными и в конечном итоге включаются в качестве функций в следующую спецификацию.

В версиях протокола HTTP 1.0 и 1.1 стороны обмениваются информацией в виде обычного текста. Отважные администраторы могут взаимодействовать с серверами напрямую, запустив утилиты **telnet** или **netcat**. Они также могут наблюдать и собирать обмены данными по протоколу HTTP с помощью такого программного обеспечения для сбора пакетов, как **tcpdump**, независимого от используемого протокола.

🕮 Общую информацию о TLS см. в разделе 27.6.

Веб находится в процессе внедрения HTTP/2, основной версии протокола, которая сохраняет совместимость с предыдущими версиями, но предоставляет ряд новых решений для повышения производительности. Стремясь обеспечить универсальное использование протокола HTTPS (защищенного, зашифрованного протокола HTTP) для следующего поколения Интернета, основные браузеры, такие как Firefox и Chrome, решили поддерживать HTTP/2 только через TLS-зашифрованные соединения.

Чтобы упростить синтаксический анализ и повысить эффективность сети, протокол HTTP/2 переводится из текстового формата в двоичный. Семантика HTTP остается прежней, но поскольку переданные данные больше не поддаются непосредственному анализу людьми, универсальные средства, такие как **telnet**, теперь бесполезны. Восстановить определенную интерактивность и возможность отладки соединений HTTP/2 помогает удобная утилита командной строки **h2i**, входящая в состав сетевого репозитория языка Go `github.com/golang/net`. Многие HTTP-специфичные инструменты, такие как **curl**, также изначально поддерживают протокол HTTP/2.

Унифицированные указатели ресурсов (URL)

URL-адрес — это идентификатор, определяющий способ и место доступа к ресурсу. URL-адреса не являются HTTP-специфичными; они также используются для других протоколов. Например, мобильные операционные системы используют URL-адреса для облегчения связи между приложениями.

Иногда встречаются такие аббревиатуры, как URI (Uniform Resource Identifier) и URN (Uniform Resource Name). Точные различия и отношения между терминами

URL, URI и URN являются неопределенными и несущественными. Просто используйте термин "URL".

Общий шаблон для URL-адресов имеет вид *схема:адрес*, где *схема* идентифицирует протокол или целевую систему, а *адрес* — это некая строка, которая имеет смысл в рамках этой схемы. Например, URL `mailto:ulsah@admin.com` инкапсулирует адрес электронной почты. Если он вызывается в качестве целевой ссылки в Интернете, большинство браузеров будут открывать окно почтового клиента для отправки электронной почты по указанному адресу.

Для Интернета релевантными являются схемы `http` и `https`. На практике можно также встретить схемы `ws` (WebSockets), `wss` (WebSockets over TLS), `ftp`, `ldap` и многие другие.

Адресная часть URL-адреса для веба позволяет определить довольно внушительную внутреннюю структуру. Вот как выглядит общий шаблон:

схема://[имя_пользователя[:пароль]@]имя_хоста[:порт][[/путь][?запрос][#якорь]

Все элементы, кроме *схемы* и *имени_хоста*, являются необязательными.

🕮 Более подробную информацию о базовой аутентификации HTTP см. в разделе 19.4.

Использование *имени_пользователя* и *пароля* в URL-адресе обеспечивает базовую аутентификацию HTTP, которая поддерживается большинством пользовательских агентов и серверов. В общем, довольно плохая идея встраивать пароли в URL-адреса, потому что URL-адреса могут регистрироваться в журналах, публиковаться в открытых источниках, помещаться в закладки, они видны в результатах работы команды **ps** и т.д. Пользовательские агенты могут получать свои учетные данные из источника, отличного от URL-адреса, и обычно это лучший вариант. В веб-браузере достаточно просто не указывать учетные данные и позволить ему запрашивать их отдельно.

Базовая аутентификация HTTP не является защищенной, а это означает, что пароль доступен для всех, кто прослушивает транзакцию. Таким образом, базовая аутентификация должна использоваться только для безопасных соединений HTTPS.

Параметр *имя_хоста* может быть именем домена или IP-адресом, а также фактическим именем хоста. Параметр *порт* — это номер TCP-порта для подключения. В схемах `http` и `https` по умолчанию используются порты 80 и 443 соответственно.

Раздел *запроса* может включать в себя несколько параметров, разделенных амперсандами. Каждый параметр представляет собой пару *ключ=значение*. Например, пользователи Adobe InDesign могут столкнуться со следующим URL-адресом:

```
http://adobe.com/search/index.cfm?term=indesign+crash&loc=en_us
```

Как и в случае с паролями, конфиденциальные данные никогда не должны появляться в качестве параметра запроса URL-адреса, поскольку пути URL-адресов часто регистрируются в журналах в виде обычного текста. Альтернативой является передача параметров как части тела запроса. (Вы не можете контролировать это в программном обеспечении других людей, но можете убедиться, что ваш собственный сайт ведет себя правильно.)

Параметр *якорь* идентифицирует частную цель определенного URL-адреса. Например, Википедия широко использует якоря в качестве заголовков разделов, позволяя напрямую связывать отдельные части статей.

Структура транзакции протокола HTTP

HTTP-запросы и ответы похожи по структуре. После начальной строки оба содержат последовательность заголовков, пустую строку и, наконец, тело сообщения, называемое полезной нагрузкой.

HTTP-запросы

Первая строка запроса указывает действие, которое должен выполнить сервер. Она состоит из метода запроса (также известного как глагол), пути для выполнения действия и используемой версии HTTP. Например, запрос на получение HTML-страницы верхнего уровня может выглядеть так:

```
GET /index.html HTTP/1.1
```

Основные методы HTTP-запроса показаны в табл. 19.1. Глаголы, отмеченные как "безопасные", не должны изменять состояние сервера. Однако это скорее пожелание, чем требование. В конечном итоге это зависит от программного обеспечения, которое обрабатывает запрос и решает, как интерпретировать глагол.

Таблица 19.1. Методы HTTP-запроса

Глагол	Безопасный?	Цель
GET	Да	Получает указанный ресурс
HEAD	Да	Аналогичен **GET**, но не требует никакой нагрузки; извлекает только метаданные
DELETE	Нет	Удаляет указанный ресурс
POST	Нет	Применяет данные запроса к данному ресурсу
PUT	Нет	Аналогичен **POST**, но подразумевает замену существующего содержимого
OPTIONS	Да	Показывает, какие методы поддерживает сервер для указанного пути

Метод `GET`, безусловно, является наиболее часто используемым HTTP-глаголом, за ним следует глагол `POST`.[1] В интерфейсе прикладного программирования REST, обсуждаемом в разделе 19.2, чаще используются более экзотические глаголы, такие как `PUT` и `DELETE`.

HTTP-ответы

Начальная строка в ответе, называемая *строкой состояния*, указывает текущую ситуацию с запросом. Это выглядит так:

```
HTTP/1.1 200 OK
```

Важной частью является трехзначный цифровой код состояния. Последующая фраза — это полезный комментарий на английском языке, который программное обеспечение игнорирует.

Первая цифра в коде определяет его класс, т.е. общий характер результата. В табл. 19.2 показаны пять определенных классов. Остальные две цифры в классе дают дополнительную информацию. В протоколе определено более 60 кодов состояния, но только некоторые из них обычно встречаются на практике.

[1] Различие между глаголами `POST` и `PUT` является тонким и в значительной степени связано с разработкой интерфейсов прикладного программирования для веба. Глагол `PUT` должен быть идемпотентным, т.е. `PUT` можно повторить, не вызывая вредных эффектов. Например, транзакция, которая заставляет сервер отправлять электронную почту, не должна быть представлена как `PUT`. Правила кеширования HTTP для `PUT` и `POST` также значительно различаются между собой. Более подробную информацию см. в спецификации. RFC2616.

Таблица 19.2. Классы HTTP-запроса

Код	Общие указания	Примеры
`1xx`	Полученный запрос; обработка продолжается	`101 Switching protocols`
`2xx`	Успех	`200 OK`
		`201 Created`
`3xx`	Необходимы дополнительные действия	`301 Moved permanently`
		`302 Found`[a]
`4xx`	Недопустимый запрос	`403 Forbidden`
		`404 Not Found`
`5xx`	Ошибка сервера или среды выполнения запроса	`503 Service Unavailable`

[a] Чаще всего используется (хотя и нецелесообразно, если следовать спецификации) для временных переадресаций.

Заголовки и тело сообщения

Заголовки указывают метаданные о запросе или ответе (например о разрешении сжатия), какие типы содержимого принимаются, ожидаются или предоставляются и как промежуточные кеши должны обрабатывать данные. Для запросов единственным требуемым заголовком является `Host`, который используется программным обеспечением веб-сервера для определения того, к какому именно сайту происходит обращение.

Основные заголовки приведены в табл. 19.3.

Таблица 19.3. Основные HTTP-заголовки

Имя: пример	Направление[a]	Содержимое
`Host: www.admin.com`	→	Имя домена и запрашиваемый порт
`Content-Type: application/json`	↔	Желаемый или существующий формат данных
`Authorization: Basic QWx...FtZ==`	→	Сертификаты для базовой аутентификации протокола HTTP
`Last-Modified: Wed, Sep 7 2016...`	←	Последняя известная дата модификации объекта
`Cookie: flavor=oatmeal`	→	Объект `cookie`, возвращаемый пользовательским агентом
`Content-Length: 423`	↔	Длина тела в байтах
`Set-Cookie: flavor=oatmeal`	←	Объект `cookie`, сохраняемый пользовательским агентом
`User-Agent: curl/7.37.1`	→	Пользовательский агент, пославший запрос
`Server: nginx/1.6.2`	←	Программное обеспечение сервера, отвечающее на запрос
`Upgrade: HTTP/2.0`	↔	Запрос на переход к другому протоколу
`Expires: Sat, 15 Oct 2016 14:02:...`	←	Допустимая продолжительность кеширования ответа
`Cache-Control: max-age=7200`	↔	Аналогичен `Expires`, но допускает больше контроля

[a] Направление: → только запрос, ← только ответ или ↔ и запрос, и ответ.

Список в табл. 19.3 ни в коем случае не является окончательным. Фактически обе стороны транзакции могут включать любые заголовки, которые они пожелают. Обе стороны должны игнорировать заголовки, которые они не понимают.[2]

Заголовки отделяются от тела сообщения пустой строкой. Для запросов тело сообщения может включать параметры (для `POST` или `PUT`) или содержимое загружаемого файла. Для ответов тело сообщения является полезной нагрузкой запрашиваемого ресурса (например, файл в формате HTML, данные изображения или результаты запроса). Тело сообщения не обязательно читается человеком, поскольку оно может содержать изображения или другие двоичные данные. Тело также может быть пустым, например для запросов `GET` или большинства ответов об ошибках.

Утилита `curl`: инструмент командной строки для работы с HTTP

Утилита **`curl`** (cURL) — удобный HTTP-клиент командной строки, доступный для большинства платформ.[3] Здесь мы используем ее для изучения обмена HTTP.

Ниже приведен вызов **`curl`**, запрашивающий корень веб-сайта `admin.com` на TCP-порту 80, который по умолчанию используется для незашифрованных (не HTTPS) запросов. Полезная нагрузка отклика (т.е. домашняя страница `admin.com`) и некоторые информационные сообщения от самой утилиты **`curl`** были скрыты флагами **`-o /dev/null`** и **`-s`**. Мы также установили флаг **`-v`**, чтобы запросить подробный вывод **`curl`** на дисплей, включая заголовки.

```
$ curl -s -v -o /dev/null http://admin.com
* Rebuilt URL to: http://admin.com/
* Hostname was NOT found in DNS cache
*   Trying 54.84.253.153...
* Connected to admin.com (54.84.253.153) port 80 (#0)
> GET / HTTP/1.1
> User-Agent: curl/7.37.1
> Host: admin.com
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Mon, 27 Apr 2015 18:17:08 GMT
* Server Apache/2.4.7 (Ubuntu) is not blacklisted
< Server: Apache/2.4.7 (Ubuntu)
< Last-Modified: Sat, 02 Feb 2013 03:08:20 GMT
< ETag: "66d3-4d4b52c0c1100"
< Accept-Ranges: bytes
< Content-Length: 26323
< Vary: Accept-Encoding
< Content-Type: text/html
<
{ [2642 bytes data]
* Connection #0 to host admin.com left intact
```

[2]По соглашению, пользовательские и экспериментальные заголовки изначально имели префикс "`X-`". Однако некоторые заголовки с префиксом "`X-`" (такие как `X-Forwarded-For`) стали стандартами де-факто, и тогда стало невозможно удалить префикс, потому что это нарушило бы совместимость. Использование префикса "`X-`" в спецификации RFC6648 объявлено устаревшим.

[3]Администраторы также должны помнить о клиентской библиотеке **`libcurl`**, которую разработчики могут использовать для создания собственных программных аналогов утилиты **`curl`**.

Строки, начинающиеся с символов > и <, обозначают запрос и ответ соответственно. В запросе клиент сообщает серверу, что пользовательский агент представляет собой утилиту **curl**, что он ищет сайт `admin.com` и что в качестве ответа он будет принимать любой тип содержимого. Сервер идентифицирует себя как Apache 2.4.7 и отвечает содержимым типа HTML, а также множеством других метаданных.

Существует возможность явно задавать заголовки с помощью аргумента **-H** утилиты **curl**. Эта функция особенно удобна для непосредственных запросов по IP-адресам, минуя DNS. Например, установив заголовок `Host` в командной строке пользовательского агента, мы можем проверить, что веб-сервер для сайта `www.admin.com` отвечает идентично запросам, адресованным сайту `admin.com`:

```
$ curl -H "Host: www.admin.com" -s -v -o /dev /null 54.84.253.153
<тот же вывод, что и в предыдущем примере, но с другим заголовком Host>
```

Для загрузки файла используется аргумент **-O** . В данном примере в текущий каталог загружается архив `tar` исходного кода **curl**:

```
$ curl -O http://curl.haxx.se/snapshots/curl-7.46.0-20151105.tar.gz
```

Мы только коснулись возможностей утилиты **curl**. Она может обрабатывать другие методы запроса, такие как `POST` и `DELETE`, сохранять и отправлять файлы `cookie`, загружать файлы и помогать в различных сценариях отладки.

Браузер Chrome компании Google предлагает функцию "Copy as cURL" ("Копировать как cURL"), которая создает команду **curl**, чтобы имитировать собственное поведение браузера, включая заголовки, файлы `cookie` и другие детали. С ее помощью легко повторять запросы с различными настройками и видеть результаты точно так же, как и в браузере. (Щелкните правой кнопкой мыши на имени ресурса на вкладке **Network** (Сеть) панели инструментов разработчика, чтобы открыть эту опцию.)

Повторное использование TCP-соединений

TCP-соединения дороги. В дополнение к памяти, необходимой для их поддержки, трехстороннее квитирование, используемое для установки каждого нового соединения, добавляет задержку, эквивалентную полному раунду пересылки пакетов еще до того как HTTP-запрос вообще начнет выполняться.[4]

По оценкам HTTP Archive, проекта по сбору веб-статистики, при загрузке страницы среднестатистического сайта выполняются запросы на 99 ресурсов. Если каждому ресурсу потребовалось бы новое TCP-соединение, производительность сети была бы очень низкой. На самом деле так и было в первое время после появления Интернета.

Исходная спецификация HTTP/1.0 не содержала никаких предложений для повторного использования соединений, но некоторые авантюрные разработчики добавили экспериментальную поддержку в качестве расширения. К клиентам и серверам неформально был добавлен заголовок `Connection: Keep-Alive`, который затем был улучшен и принят по умолчанию в версии HTTP/1.1. Благодаря постоянным подключениям (также называемым *персистентными*) HTTP-клиенты и серверы отправляют несколько запросов по одному соединению, что позволяет сэкономить часть ресурсов и времени на установку и разрыв нескольких соединений.

Накладные расходы протокола TCP оказываются нетривиальными даже при включенной поддержке постоянных соединений в протоколе HTTP/1.1. Для повышения производи-

[4]TCP Fast Open (TFO) — это предложение, целью которого является улучшение этой ситуации, позволяющее также передавать пакеты `SYN` и `SYN-ACK` трехстороннего квитирования протокола TCP (см. спецификацию RFC7413).

тельности большинство браузеров открывают до шести параллельных подключений к серверу. Занятые серверы, в свою очередь, должны поддерживать много тысяч TCP-соединений в разных состояниях, что приводит к перегрузке сети и простою ресурсов.

Для решения этой проблемы версия HTTP/2 представляет мультиплексирование, позволяя чередовать несколько транзакций по одному соединению. Таким образом, серверы HTTP/2 могут поддерживать больше клиентов для каждой системы, поскольку каждый клиент создает меньше накладных расходов.

HTTP на основе протокола TLS

Сам по себе протокол HTTP не обеспечивает безопасность на уровне сети. Указатели URL, заголовки и полезная нагрузка открыты для проверки и модификации в любой точке между клиентом и сервером. Злоумышленники могут перехватывать сообщения, изменять их содержимое или перенаправлять запросы на серверы по своему выбору.

Для решения этой проблемы был предложен протокол TLS (Transport Layer Security — безопасный протокол транспортного уровня), который работает как отдельный уровень между протоколами TCP и HTTP.[5] Протокол TLS предоставляет только средства безопасности и шифрования на уровне соединения; он никак не связан с уровнем протокола HTTP.

Пользовательский агент проверяет подлинность сервера как часть процесса подключения по протоколу TLS, исключая возможность подмены поддельными серверами. Как только соединение установлено, его содержимое становится защищенным от слежения и подмены на время обмена. Атакующие все еще могут видеть хост и порт, используемые на уровне TCP, но они не могут получить доступ к HTTP-данным, таким как URL-адрес запроса или заголовки, которые его сопровождают.

🕮 Дополнительную информацию о криптографии TLS см. в разделе 27.6.

Виртуальные хосты

В первые дни Интернета на веб-сервере обычно размещался только один веб-сайт. Например, при запросе `admin.com` клиенты выполняли поиск в DNS, чтобы найти IP-адрес, связанный с этим именем, а затем отправляли HTTP-запрос на порт 80 по этому адресу. Сервер по этому адресу знал, что он был связан с сайтом `admin.com` и соответствующим образом обрабатывал результаты.

По мере увеличения использования Интернета администраторы поняли, что они могут достичь эффекта масштабирования, если на одном сервере будут размещаться сразу несколько сайтов. Но как отличить запросы, связанные с сайтом `admin.com`, от тех, которые связаны с сайтом `example.com`, если оба трафика идут через один и тот же сетевой порт?

Одна из возможностей состоит в том, чтобы определить виртуальные сетевые интерфейсы, что по сути привязывает несколько различных IP-адресов к одному физическому соединению. В большинстве систем это допускается и работает нормально, но такая схема неудобна и требует управления на нескольких уровнях.

Лучшее решение (виртуальные хосты) было предоставлено в протоколе HTTP 1.1, описанном в спецификации RFC2616. Эта схема определяет HTTP-заголовок `Host`, который пользовательские агенты задают явно, чтобы указать, с каким именно сайтом они

[5] Предшественник протокола TLS был известен под именем SSL (Secure Sockets Layer). Все версии SSL официально объявлены устаревшими, но аббревиатура SSL все еще широко используется. Вне криптографического контекста мы будем считать, что SSL на самом деле означает TLS.

пытаются связаться. Серверы проверяют этот заголовок и ведут себя соответственно. Это соглашение позволяет сэкономить пространство IP-адресов и упрощает управление, особенно для веб-серверов, на которых обслуживаются сотни или тысячи веб-сайтов на одном физическом компьютере.

Протокол HTTP 1.1 *требует*, чтобы пользовательские агенты предоставляли заголовок `Host`, поэтому виртуальные хосты теперь являются стандартным способом, с помощью которого веб-серверы и администраторы обеспечивают консолидацию серверов.

Использование виртуальных хостов на основе имени в сочетании с протоколом TLS организовать довольно сложно. Сертификаты TLS выдаются для определенных имен хостов, которые выбираются при создании сертификата. Соединение TLS должно быть установлено до того, как веб-сервер сможет прочитать заголовок `Host` из HTTP-запроса, но без этого заголовка веб-сервер не знает, каким именно виртуальным хостом он должен прикинуться и, следовательно, какой сертификат выбрать.

Решением является механизм SNI (Server Name Indication — указание имени сервера), с помощью которого клиент отправляет имя требуемого хоста в качестве части исходного сообщения TLS-соединения. Современные серверы и клиенты обрабатывают расширение SNI автоматически.

19.2. Основы программного обеспечения для веба

Богатая библиотека программного обеспечения с открытым исходным кодом облегчает создание гибких, устойчивых веб-приложений. В табл. 19.4 перечислено несколько общих категорий служб, работающих в рамках протокола HTTP и выполняющих определенные функции в стеке веб-приложений.

Таблица 19.4. Частичный список типов HTTP-серверов

Тип	Цель	Примеры
Сервер приложений	Запускает код веб-приложения, интерфейсы для веб-серверов	Unicorn, Tomcat
Кеш	Ускоряет доступ к часто запрашиваемому материалу	Varnish, Squid
Балансировщик нагрузки	Перенаправляет запросы в нижестоящие системы	Pound, HAProxy
Брандмауэр веб-приложений[a]	Проверяет HTTP-трафик в поисках распространенных атак	ModSecurity
Веб-сервер	Обслуживает статическое содержимое, связывается с другими серверами	Apache, NGINX

[a] Часто сокращается как WAF (Web App Firewall).

Веб-прокси — это посредник, получающий HTTP-запросы от клиентов, (необязательно) выполняющий определенную обработку и передающий запросы в конечный пункт назначения.

Прокси-серверы обычно прозрачны для клиентов. Балансировщики нагрузки, брандмауэры веб-приложений и серверы кешей — это специализированные типы прокси-серверов. Веб-сервер также действует как своего рода прокси-сервер, если он передает запросы серверам приложений.

На рис. 19.1 показана роль, которую каждая служба играет в обмене данными по протоколу HTTP. Запросы могут выполняться на верхних уровнях стека, если запрошенный ресурс может быть удовлетворен, или отклоняться с кодом `4xx` или `5xx`, если возникает проблема. Запросы, требующие запроса к базе данных, проходят все уровни.

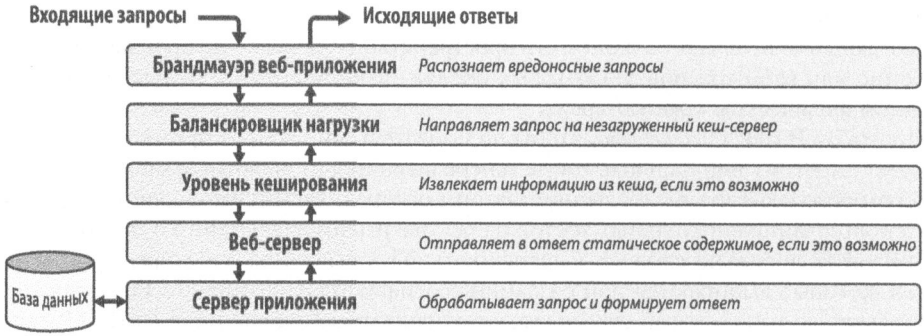


Рис. 19.1. Компоненты стека веб-приложений

Чтобы максимизировать доступность, каждый уровень должен работать на нескольких серверах одновременно. В идеальном случае избыточность должна охватывать географические регионы, чтобы схема в целом не зависела ни от одного физического центра обработки данных. Эту цель намного проще достичь, используя облачную платформу, которая предлагает четко определенные географические регионы в качестве фундаментального строительного блока.

В реальном мире архитектура обычно бывает не такой понятной, как предполагается на рис. 19.1. Кроме того, в большинстве компонентов веб-приложений реализованы функции более чем одного уровня. Например, NGINX — это и веб-сервер, и высокопроизводительный кеш, и балансировщик нагрузки. Веб-сервер NGINX с включенными функциями кеширования более эффективен, чем стек отдельных серверов, работающих на отдельных виртуальных машинах.

Веб-серверы и прокси-сервер протокола HTTP

На большинстве сайтов веб-серверы используются в качестве прокси-серверов для перенаправления HTTP-трафика серверам приложений или для непосредственной выборки и доставки клиенту статического содержимого. Ниже приводится список некоторых функций, предоставляемых веб-серверами.

- Виртуальные хосты, позволяющие многим веб-сайтам мирно сосуществовать на одном веб-сервере.
- Обработка соединений TLS.
- Настраиваемая регистрация запросов в системном журнале, которая позволяет зафиксировать запросы клиентов и ответы сервера.
- Базовая аутентификация протокола HTTP.
- Маршрутизация в разные нисходящие системы в соответствии с запрошенными URL-адресами.
- Выполнение динамического содержимого через серверы приложений.

Ведущими веб-серверами с открытым исходным кодом являются HTTP-сервер Apache, известный в разговорной речи как `httpd`, и NGINX, который произносится как "энджин икс" ("движок икс").

Интернет-исследовательская и сетевая компания Netcraft публикует ежемесячную статистику доли рынка для веб-серверов. По состоянию на июнь 2017 г. Netcraft пока-

зывает, что около 46% активных веб-сайтов используют Apache. Доля NGINX составляет 20% и неуклонно растет с 2008 года.

Apache **httpd** — это оригинальный проект организации Apache Software Foundation, которая известна тем, что поддерживает множество отличных проектов с открытым исходным кодом. Проект **httpd** находится в активной разработке с 1995 г. и многими рассматривается как эталонная реализация HTTP-сервера.

NGINX — это универсальный сервер, предназначенный для обеспечения скорости и эффективности. Как и **httpd**, NGINX поддерживает обслуживание статического веб-содержимого, балансировку нагрузки, мониторинг подчиненных серверов, проксирование, кеширование и другие связанные функции.

Некоторые системы разработки, в частности Node.js и язык Go, реализуют внутренние веб-серверы и могут обрабатывать множество рабочих процессов по протоколу HTTP без необходимости использования отдельного веб-сервера. Эти системы включают в себя сложные функции управления подключением и достаточно надежны для производственных нагрузок.

Сервер H2O (`h2o.example.net`, обратите внимание на то, что в слове **example** стоит цифра 1) — это новый проект веб-сервера, который в полной мере использует функции HTTP/2 и обеспечивает еще более высокую производительность, чем NGINX. Поскольку он был впервые выпущен в 2014 г., он не может претендовать на послужной список Apache или NGINX. С другой стороны, он никак не связан историческими ограничениями в плане реализации, а также с проектом **httpd**. Его, безусловно, стоит рассматривать как вариант для новых развертываний.

Трудно сделать хорошие рекомендации относительно этих вариантов, потому что они все неплохие. Тем не менее для основного производственного сервера мы используем сервер NGINX. Он представляет исключительную производительность и относительно простую и современную систему конфигурации.

Балансировщики нагрузки

Невозможно запустить высокодоступный веб-сайт на одном сервере. Эта конфигурация не только создаст вашим пользователям все возможные проблемы, связанные с сервером, но также не дает возможности обновлять программное обеспечение, операционную систему или конфигурацию без простоя.

Отдельные серверы также особенно уязвимы для перегрузок и преднамеренных атак. Чем больше перегружается сервер, тем больше времени он тратит вместо того, чтобы выполнять полезную работу. Пройдя определенный порог нагрузки (который еще нужно будет обнаружить путем горького опыта!), его производительность резко падает, а не плавно снижается.

Чтобы избежать этих проблем, можно использовать балансировщик нагрузки, который является особым типом прокси-сервера, распределяющего входящие запросы среди множества нисходящих веб-серверов. Балансировщики нагрузки также контролируют состояние этих серверов, чтобы обеспечить своевременную и корректную обработку клиентских запросов.

На рис. 19.2 показано размещение балансировщика нагрузки на архитектурной диаграмме.

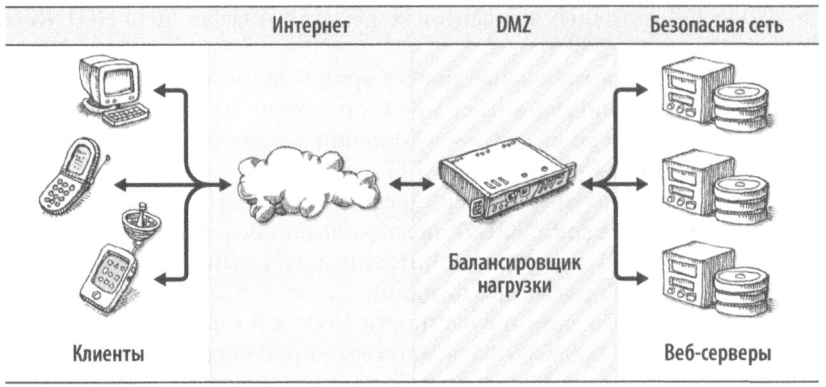


Рис. 19.2. Роль балансировщика нагрузки

Балансировщики нагрузки решают многие проблемы, присущие односистемному дизайну.

- Балансировщики нагрузки не обрабатывают запросы, а просто направляют их в другие системы. В результате они могут обрабатывать намного больше одновременных запросов, чем обычный веб-сервер.
- Когда веб-сервер нуждается в обновлении программного обеспечения или должен быть отключен по какой-либо другой причине, его можно легко удалить из ротации.
- Если на одном из серверов возникла проблема, механизм проверки работоспособности на балансировщике нагрузки обнаруживает проблему и удаляет ошибочную систему из пула серверов, пока он снова не станет "здоровым".

Чтобы самим не стать единственной точкой отказа, балансировщики нагрузки обычно работают парами. В зависимости от конфигурации один балансировщик может действовать как пассивная резервная копия, в то время как другой обслуживает основной трафик, или оба балансировщика могут одновременно обслуживать запросы.

Способ распределения запросов обычно настраивается. Вот несколько общих алгоритмов.

- Циклический алгоритм, в котором запросы распределяются между активными серверами в фиксированном порядке ротации.
- Алгоритм выравнивания нагрузки, в котором новые запросы переходят на нисходящий сервер, в настоящее время обрабатывающий наименьшее количество подключений или запросов.
- Алгоритм разделения, в котором балансировщик нагрузки выбирает сервер в соответствии с хешем IP-адреса клиента. Этот метод гарантирует, что запросы одного и того же клиента всегда достигают одного и того же веб-сервера.

Балансировщики нагрузки обычно работают на четвертом уровне модели OSI, на котором они перенаправляют запросы с учетом только IP-адреса и номера порта запроса. Тем не менее они также могут работать на втором уровне, проверяя запросы и маршрутизируя их в соответствии с их целевым URL, значениями `cookie` или другими заголовками HTTP. Например, запросы на сайт `example.com/linux` могут направляться на одну совокупность серверов, а запросы на сайт `example.com/bsd` — на другую.

🕮 Дополнительную информацию о сетевых DMZ (демилитаризованных зонах) см. в разделе 27.8.

В качестве дополнительного бонуса балансировщики нагрузки могут повысить безопасность. Они обычно находятся в демилитаризованной зоне (DMZ) сети и выполняют функции прокси-серверов, пересылающих запросы веб-серверам, защищенным внутренним брандмауэром. Если используется протокол HTTPS, они также являются конечной точкой защищенного соединения TLS: при подключении клиента к балансировщику нагрузки используется TLS, но при подключении балансировщика нагрузки к веб-серверам может использоваться обычный протокол HTTP. Эта компоновка позволяет снять часть нагрузки с веб-серверов и освободить их от выполнения некоторой вспомогательной работы.

Балансировщики нагрузки могут распространять другие виды трафика в дополнение к (или вместо) протоколу HTTP. Общепринятым является добавление балансировщика нагрузки, который распределяет запросы к базам данных, таким как MySQL или Redis.

Наиболее распространенными балансировщиками с открытым исходным кодом для систем UNIX и Linux являются NGINX, уже представленный как веб-сервер, и HAProxy, высокопроизводительный прокси-сервер для протоколов TCP и HTTP, любимый ветеранами сетевого администратрирования за его гибкую конфигурацию, стабильность и высокую производительность. Они оба превосходны и хорошо документированы, и оба имеют большие сообщества пользователей. (Apache `httpd` также имеет модуль балансировки нагрузки, хотя мы не видели его в массовой эксплуатации.)

Коммерческие балансировщики нагрузки, такие как F5 и Citrix, доступны как в качестве аппаратных устройств, которые будут установлены в дата-центре, так и в качестве программных решений. Они обычно предлагают графический интерфейс конфигурации, больше возможностей, чем инструменты с открытым исходным кодом, дополнительные функции кроме простой балансировки нагрузки и высокие ценовые ориентиры.

Компания Amazon предлагает специальную службу гибкой балансировки нагрузки, Elastic Load Balancer (ELB) для использования с виртуальными машинами EC2. ELB — это полностью управляемая служба; для самого балансировщика нагрузки виртуальная машина не требуется. Служба ELB обрабатывает чрезвычайно большое количество одновременных подключений и может балансировать трафик между несколькими зонами доступности.

В терминологии ELB "слушатель" принимает соединения от клиентов и "проксирует" их на внешние экземпляры EC2, которые выполняют фактическую работу. Слушатели могут проксировать трафик TCP, HTTP или HTTPS. Нагрузка распределяется по алгоритму наименьшего числа соединений.

Служба ELB не является самым полнофункциональным балансировщиком нагрузки, но мы рекомендуем это решение для систем, размещенных на AWS, потому что оно не требует практически никакого административного внимания.

Кеши

Веб-кеши появились в результате наблюдения, что клиенты часто повторно обращаются к одному и тому же содержимому за короткое время. Кеши находятся между клиентами и веб-серверами и хранят результаты самых частых запросов, иногда в оперативной памяти. Затем они могут вмешиваться, чтобы ответить на запросы, для которых они знают правильный ответ, тем самым снижая нагрузку на авторитетные веб-серверы и улучшая время отклика для пользователей.

На жаргоне кеширования источник — это исходный поставщик содержимого, источник подлинной информации о нем. Кеши получают свое содержимое непосредственно из источника или из другого вышестоящего кеша.

На кеширование влияют несколько факторов.

- Значения заголовков HTTP, включая `Cache-Control`, `ETag` и `Expires`.
- Выполняется ли запрос протоколом HTTPS, для которого кеширование носит более тонкий характер.[6]
- Код состояния ответа; некоторые из них не кешируемы (см. спецификацию RFC2616).
- Содержимое HTML-дескритора `<meta>`.[7]

Статические большие двоичные объекты, такие как изображения, видео, таблицы стилей CSS и файлы JavaScript, хорошо подходят для кеширования, потому что они редко меняются. Динамическое содержимое, загружаемое из базы данных или другой системы в режиме реального времени, кешировать сложнее, но почти всегда возможно. Для страниц с высокой степенью изменчивости, которые никогда не должны кешироваться, разработчики устанавливают следующий HTTP-заголовок:

```
Cache-Control: no-cache, no-store
```

На рис. 19.3 показано размещение нескольких потенциальных уровней кеша при обработке HTTP-запроса.

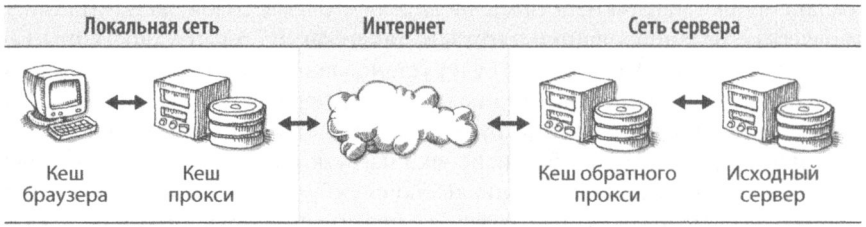


Рис. 19.3. Кеширование участников обработки HTTP-запроса

Кеш браузера

Все современные веб-браузеры хранят недавно использованные ресурсы (изображения, таблицы стилей, файлы JavaScript и некоторые HTML-страницы) локально, чтобы ускорить выборку данных и отображение веб-страниц. Теоретически кеши браузера должны следовать точно таким же правилам кеширования, как и любой другой HTTP-кеш.[8]

---

[6] Поскольку данные в протоколе HTTPS зашифрованы, ответы не могут быть кешированы, если сервер кеша не является конечным пунктом соединения TLS и может дешифровать полезную нагрузку. При подключении с сервера кеша к источнику может потребоваться отдельно зашифрованное TLS-соединение (или нет, в зависимости от того, есть ли доверие к соединению между ними).

[7] Они не учитываются всеми кешами, поэтому, как правило, менее эффективны.

[8] Вот почему после щелчка на кнопке Back (Назад) вашего браузера обычно происходит небольшая задержка, а не мгновенно отображается предыдущая страница. Несмотря на то что большинство ресурсов, необходимых для создания страницы, кешируются локально, HTML-оболочка верхнего уровня страницы обычно является динамической и некешируемой, поэтому по-прежнему требуется обращение к серверу. Браузер мог бы просто отобразить данные, полученные за время предыдущего визита — и некоторые браузеры именно так и делали, но этот прием нарушает правильность кеширования и приводит к множеству тонких проблем.

Кеширующий прокси-сервер

Чтобы ускорить доступ для всех пользователей, можно установить кеширующий прокси-сервер на границе сети организации. Когда пользователь загружает веб-сайт, запросы сначала принимаются кеширующим прокси-сервером. Если запрашиваемый ресурс находится в кеши, он немедленно возвращается пользователю без обращения к удаленному веб-серверу.

Настроить кеширующий прокси-сервер можно двумя способами: активно, изменяя настройки браузера пользователей, указывая адрес и порт прокси-сервера вручную, или пассивно, изменив конфигурацию сетевого маршрутизатора так, чтобы тот "заворачивал" всесь веб-трафик на кеширующий прокси-сервер. Последняя конфигурация известна как перехватывающий прокси-сервер. Существуют также методы, с помощью которых пользовательские агенты могут автоматически обнаруживать соответствующие прокси-серверы.

Кеширующий обратный прокси-сервер

Операторы веб-сайта настраивают кеширующий обратный прокси-сервер для перехвата трафика, отправляемого клиенту со своих веб-серверов и серверов приложений. Входящие запросы сначала направляются кеширующему обратному прокси-серверу, из которого они могут быть немедленно удовлетворены, если запрашиваемые ресурсы доступны. Обратный прокси-сервер передает запросы на некешированные ресурсы соответствующим веб-серверам.

Серверные сайты используют кеширующие обратные прокси-серверы прежде всего потому, что они уменьшают нагрузку на исходные серверы. Они также могут иметь благоприятный побочный эффект за счет ускорения времени отклика для клиентов.

Проблемы с кешем

Веб-кеши чрезвычайно важны для работы в Интернете, но они также создают сложности. Проблема на любом уровне кеширования может привести к появлению устаревшего по отношению к исходному серверу содержимого. Проблемы с кешем могут возникнуть как у пользователей, так и у администраторов, и иногда их трудно отлаживать.

Записи старого кеша лучше всего обнаруживаются с помощью прямого запроса на каждый сервер, находящийся по пути к веб-сайту. Если вы являетесь оператором сайта, попробуйте использовать команду `curl`, чтобы запросить проблемную страницу непосредственно из источника, затем из кеширующего обратного прокси-сервера и, если получится, из кеширующего прокси-сервера и из любых других кешей, расположенных на на пути прохождения запроса.

Чтобы корректно запросить обновление кеша, можно использовать команду `curl -H "Cache-Control: no-cache"`.[9] Соответствующие кеши выполнят это обновление, но если вы все еще видите старые данные, не следует полагать, что ваш запрос на перезагрузку был выполнен, пока не проверите это на сервере.

Программное обеспечение для кеширования

В табл. 19.5 перечислены некоторые из программных реализаций кеширования с открытым исходным кодом. Из них мы часто используем NGINX. Его кеширование легко настраивается, кроме того NGINX часто используется в качестве прокси-сервера или веб-сервера.

[9]Это то же самое, что выбрать команду <Shift+Reload> в веб-браузере.

Таблица 19.5. Программное обеспечение для кеширования с открытым исходным кодом

Сервер	Описание
Squid	Одна из первых реализаций кеша с открытым исходным кодом
	Обычно используется в качестве кеширующего прокси-сервера
	Включает важные функции, такие как антивирус и TLS
Varnish	Необычный язык настройки
	Многопоточный
	Модульный и расширяемый
Apache `mod_cache`	Хороший выбор для уже развернутых сайтов на `httpd`
NGINX	Хороший выбор для сайтов, уже работающих на NGINX
	Имеет репутацию хорошей производительности
Apache Traffic Server	Работает на сайтах с чрезвычайно высоким трафиком
	Поддерживает HTTP/2
	Финансируется фондом Apache Foundation компании Yahoo!

Сети доставки контента

Сеть доставки содержимого (CDN) представляет собой глобально распределенную систему, которая улучшает производительность веба путем перемещения содержимого ближе к пользователям. Серверы в сети CDN распределены географически в сотни или тысяче мест. Когда клиенты запрашивают содержимое сайта, использующего CDN, они направляются к ближайшему серверу (называемому *пограничным сервером*), тем самым уменьшая задержку и сетевую нагрузку на источник.

Пограничные серверы аналогичны кеширующим прокси-серверам. Они хранят копии содержимого локально. Если у них нет локальной копии запрашиваемого ресурса или срок их действия истек, они извлекают ресурс из источника, отвечают клиенту и обновляют свой кеш.

Сети CDN используют систему DNS для перенаправления клиентов на географически ближайший сервер. На рис. 19.4 показано, как это работает.

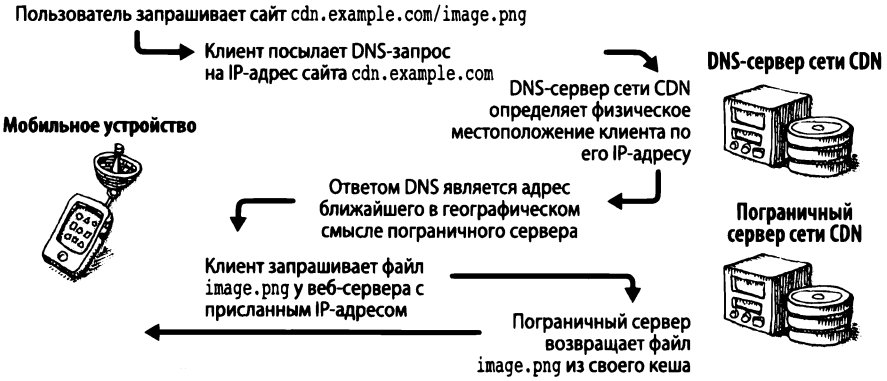


Рис. 19.4. Роль DNS в сети доставки содержимого

Сети CDN теперь могут размещать динамическое содержимое, но традиционно они лучше всего подходят для статического содержимого, такого как изображения, таблицы стилей, библиотеки JavaScript, файлы HTML и загружаемые объекты. Потоковые службы, такие как Netflix и YouTube, используют CDN для обслуживания больших медиафайлов.

Сети CDN также предлагают дополнительные услуги, помимо повышения производительности. Большинство CDN предоставляют функции обеспечения безопасности, такие как предотвращение отказа в обслуживании и брандмауэры веб-приложений. Некоторые специальные CDN предлагают другие инновации, которые оптимизируют обновление страниц и уменьшают нагрузку на исходные серверы.

Значительная часть содержимого в Интернете сегодня обслуживается сетями CDN. Если вы запускаете большой сайт, то готовьтесь вытащить свой кошелек, чтобы заплатить за высокую скорость. Если вы запускаете меньшую службу, оптимизируйте свои локальные уровни кеширования, прежде чем обращаться к сетям CDN.

Одним из старейших и самых престижных (читай: дорогих) поставщиков сетей CDN является компания Akamai со штаб-квартирой в Массачусетсе. Среди клиентов компании Akamai находятся крупнейшие государственные и частные компании. Она имеет самую большую глобальную сеть, а также поддерживает некоторые из самых передовых функций CDN. Еще никто не был уволен за то, что выбрал Akamai.

Компания CloudFlare — еще один популярный поставщик CDN. В отличие от Akamai, компания CloudFlare имеет историю продаж меньшим по масштабу клиентам, хотя ее целевой рынок в последнее время перешел на корпорации. Ценообразование четко указано на их веб-сайте, и они предлагают одни из лучших доступных функций безопасности. CloudFlare был одним из первых крупных поставщиков для развертывания протокола HTTP/2 для всех своих клиентов.

Служба CDN компании Amazon называется CloudFront. Она интегрируется с другими службами AWS, такими как S3, EC2 и ELB, но также может работать на сайтах, размещенных вне облака Amazon. Как и в случае с другими продуктами AWS, ценообразование является конкурентоспособным и зависит от использования.

Языки веба

Веб прошел эволюцию от преимущественно статичной сети до весьма интерактивной. Веб-приложения, которые обеспечивают это преимущество, кодируются на разных языках программирования, каждый из которых связан с соответствующими инструментами и уникальными особенностями. Администраторам необходимо управлять библиотеками программного обеспечения, запускать серверы приложений и настраивать веб-приложения в соответствии со стандартами, установленными для экосистемы каждого языка.

Все языки, упомянутые в следующих разделах, сегодня широко используются в Интернете. Все они имеют сообщества разработчиков, обширные библиотеки поддержки и хорошо документированные передовые методы использования. Сайты, как правило, выбирают те языки и библиотеки, с которыми их команды чувствуют себя наиболее комфортно.

Ruby

🕮 Краткое описание языка Ruby приведено в разделе 7.6.

Язык Ruby хорошо известен в среде DevOps и в кругах системного администрирования, поскольку используется в инструментах Chef и Puppet. Он также лежит в основе каркаса Ruby on Rails, широко используемого для создания веб-приложений.

Rails — хороший выбор для быстрых процессов разработки и часто используется для создания прототипов новых идей.

Rails имеет посредственную производительность и ориентирован на поддержку монолитных приложений. Со временем многие приложения Rails имеют тенденцию

накапливать ошибки, что затрудняет их модификацию; конечный результат часто постепенно снижает производительность.

Язык Ruby имеет большую коллекцию библиотек, называемых *геммами*[10] (gems), которые разработчики могут использовать для упрощения своих проектов. Большинство из них размещено на сайте `rubygems.org`. Они курируются сообществом, но многие из них имеют крайне невысокое качество. Управление установленными версиями системы Ruby и ее различными зависимостями от гемм может быть утомительным и неприятным.

Python

🕮 Краткое описание языка Python см. в разделе 7.5.

Python — это язык общего назначения, используемый не только для веб-разработки, но и в широком круге научных дисциплин. Его программы легко читать и изучать. Наиболее широко развернутым веб-каркасом для Python является Django, который имеет много преимуществ и недостатков, как и Ruby on Rails.

Java

Язык Java, теперь контролируемый компанией Oracle, чаще всего используется в корпоративных средах с более медленными технологическими процессами разработки. Java обеспечивает быструю производительность за счет сложных, неуклюжих инструментов и множества уровней абстракции. Сложные требования к лицензированию Java и сложные соглашения могут помешать новичкам.

Node.js

Язык JavaScript известен прежде всего как язык написания клиентских сценариев, которые работают в веб-браузерах. Он был жестоко осмеян как поспешно разработанный, трудно читаемый и часто противоречивый язык. После выхода платформы Node.js появилась возможность запустить сценарии JavaScript на серверах. Она также позволяет использовать язык JavaScript в центрах обработки данных.

Справедливости ради следует отметить, что платформа Node.js может похвастаться высоким параллелизмом и возможностью обмена сообщениями в реальном времени. Поскольку это новая платформа, она пока что не обременена кучей неуклюжих решений, используемых на протяжении многих лет в других системах.

PHP

Язык PHP очень прост, и по этой причине он привлекает начинающих и неопытных программистов. Приложения PHP печально известны тем, что их трудно сопровождать. Старые версии PHP предоставляли разработчиками слишком много возможностей создавать большие дыры в системе безопасности своих приложений, но в последних версиях эта ситуация была существенно исправлена. PHP — это язык, используемый системами WordPress, Drupal и несколькими другими системами управления содержимым (CMS).

Go

Go — это язык более низкого уровня, разработанный компанией Google. Он приобрел популярность в последние годы благодаря использованию в крупных проектах с открытым исходным кодом, таких как Docker. Он отлично подходит для системно-

[10] Gem — драгоценный камень, ruby — рубин. — *Примеч. перев.*

го программирования, но также применяется для веб-приложений благодаря мощным встроенным примитивам параллелизма. Одним из преимуществ для администраторов является то, что программное обеспечение на языке Go обычно компилируется в автономный двоичный файл, что упрощает его развертывание.

Интерфейсы прикладного программирования (API)

Веб-API — это интерфейсы приложений, предназначенные для использования не людьми, а программными агентами. Интерфейс API определяет набор методов, с помощью которых удаленная система может использовать данные и службы приложения. Интерфейсы API стали повсеместными в Интернете, потому что они способствуют взаимодействию между многими разными клиентами.

В интерфейсах API ничего нового. Операционные системы определяют API, позволяя приложениям пользовательского пространства взаимодействовать с ядром. Почти все программные пакеты используют определенные интерфейсы для облегчения модульности и разделения функций внутри кодовой базы. Тем не менее веб-API отличаются, поскольку раскрываются в общедоступной сети, чтобы поощрить их использование сторонними разработчиками.

Вызовы веб-API — это обычные HTTP-запросы. Они называются именно так, поскольку клиент и сервер согласились, что некоторые URL-адреса и глаголы имеют определенные значения и эффекты в области их взаимодействия.

Веб-API обычно используют какой-то текстовый формат сериализации для кодирования данных для обмена. Эти форматы относительно просты и могут анализироваться приложениями, написанными на любом языке программирования. Существует множество форматов, но наиболее распространенными являются JavaScript Object Notation (JSON)[11] и Extensible Markup Language (XML).

Интерфейсы HTTP API, возможно, проще всего объяснить на примере. Музыкальная служба Spotify предлагает интерфейс API, который представляет ее библиотеку произведений. Клиент API может запрашивать информацию об альбомах, исполнителях и треках, выполнять поиск, а также выполнять другие связанные с этим действия. Этот интерфейс API используется как собственными клиентскими приложениями Spotify (его браузером, настольными и мобильными клиентами), так и третьими лицами, которые захотят включить поддержку службы Spotify в свои приложения.

Поскольку веб-API состоит из HTTP-запросов, с этой службой можно взаимодействовать с помощью всех обычных средств, поддерживающих протокол HTTP, включая веб-браузеры и утилиту `curl`. Например, мы можем получить запись JSON Spotify для группы The Beatles:[12]

```
$ curl https://api.spotify.com/v1/artists/3WrFJ7ztbogyGnTHbHJFl2 | jq '.'
{
   "external_urls": {
       "spotify": "https://open.spotify.com/artist/3WrFJ7ztbogyGnTHbHJFl2"
    },
    "followers": {
```

[11]Дуглас Крокфорд (Douglas Crockford), который назвал и продвинул формат JSON, говорит, что это название произносится как имя Джейсон, но почему-то в техническом сообществе более широкое распространение получило произношение "JAY-sawn" (джейсоун).

[12]Как мы узнали, что `3WrFJ7ztbogyGnTHbHJFl2` — это идентификатор Spotify для группы The Beatles? Попробуйте выполнить поиск на странице `https://api.spotify.com/v1/search?type=artist&q=beatles`.

```
      "href": null,
      "total": 1566620
    },
    "genres": [ "british invasion" ],
    "href": "https://api.spotify.com/v1/artists/3WrFJ7ztbogyGnTHbHJFl2",
    "id": "3WrFJ7ztbogyGnTHbHJFl2",
    "images": [ <removed for concision> ],
    "name": "The Beatles",
    "popularity": 91,
    "type": "artist",
    "uri": "spotify:artist:3WrFJ7ztbogyGnTHbHJFl2"
}
```

Здесь мы передали результат в формате JSON на обработку утилите `jq`, чтобы немного улучшить форматирование.[13] На терминале утилита `jq` раскрашивает результат.

Интерфейс API Spotify также является примером стиля RESTful, который сегодня является преобладающим. REST (Representational State Transfer — передача состояния представления) — это архитектурный стиль проектирования API, предложенный Роем Филдингом (Roy Fielding) в его докторской диссертации.[14] Этот термин первоначально был весьма специфичным, но теперь он менее тесно связан с веб-службами, которые 1) явно используют HTTP-глаголы для передачи намерений и 2) используют каталогизированную структуру пути для поиска ресурсов. Большинство API-интерфейсов REST используют формат JSON в качестве основного средства для представления данных.

Стиль REST резко контрастирует с протоколом SOAP (Simple Object Access Protocol — простой протокол доступа к объектам), более ранней системой для реализации HTTP API, которая определяет строгие и сложные многоуровневые рекомендации для взаимодействия между системами. API-интерфейсы SOAP используют сложный формат XML, который объединяет все вызовы с помощью нескольких конкретных URL-адресов, что приводит к большим объемам пересылок данных по протоколу HTTP, низкой производительности и бесконечным трудностям при разработке, отладке и развертывании.[15]

19.3. Облачный веб-хостинг

Облачные провайдеры предлагают десятки служб для размещения веб-приложений, а их набор меняется еженедельно. Мы не можем охватить все, но некоторые моменты представляют особый интерес для веб-администраторов.

Небольшие сайты с небольшим количеством пользователей, допускающие случайные отключения, могут обойтись одним виртуальным облачным экземпляром в качестве веб-сервера (или, возможно, двумя экземплярами за балансировщиком нагрузки для по-

[13]Утилита `jq` делает гораздо больше, чем простое улучшение форматирования, и настоятельно рекомендуется для синтаксического анализа и фильтрации данных в формате JSON из командной строки. Вы можете загрузить ее с сайта `stedolan.github.io/jq`.

[14]Филдинг также является основным автором спецификации HTTP.

[15]Разработка экосистемы SOAP представляет собой интересное тематическое исследование путей, с помощью которых технические инициативы могут сойти с рельс. В частности, это иллюстрирует риски, связанные с попыткой разработать системы для туманного и неопределенного будущего. SOAP приложил немало усилий для сохранения нейтральной платформы, языка, данных и транспорта, и в значительной степени достиг этих целей — даже базовые типы данных, такие как целые, оставались открытыми для определения. К сожалению, получившаяся система оказалась сложной и не соответствовала потребностям практики.

вышения надежности). Однако облако предлагает множество возможностей для улучшения этих простых конфигураций без значительного увеличения стоимости и сложности администрирования.

Сборка или покупка

Администраторы, работающие на облачной платформе, могут создавать пользовательские самодостаточные веб-приложения из "сырых" виртуальных машин. В качестве альтернативы, они могут самостоятельно выполнять часть проектирования облачных служб, тем самым сокращая затраты на разработку, настройку и обслуживание всей службы вручную. Для повышения эффективности мы предпочитаем по возможности полагаться на услуги поставщика.

Балансировщики нагрузки — хороший пример этого компромисса. Например, на сервере AWS можно запускать экземпляр EC2 с программным обеспечением для балансировки нагрузки с открытым исходным кодом или подписываться на прилагаемый AWS модуль балансировки нагрузки Elastic Load Balancer (ELB). Первый предлагает большие возможности настройки, но при этом требует от пользователей управлять операционной системой балансировки нагрузки, настраивать программное обеспечение балансировки нагрузки, настраивать производительность и оперативно устанавливать исправления безопасности, по мере их выпуска в будущем. Кроме того, соединить все это в одно целое, чтобы правильно обрабатывать отказы программного обеспечения или экземпляра, будет несколько сложнее.

С другой стороны, ELB может быть создан за считанные секунды и не требует никаких дополнительных административных действий. AWS обрабатывает все за кулисами. Если только в ELB не отсутствует определенная функция, которая вам нужна, то это, безусловно, целесообразный выбор.

В конечном итоге это решение заключается в создании службы или аутсорсинге ее поставщику. Ради здравого смысла избегайте сборки, если эта функция не является основной компетенцией для вашего бизнеса.

Платформа как услуга

Концепция PaaS (Platform-as-a-Service) упрощает веб-хостинг для разработчиков, устраняя инфраструктуру как проблему. Разработчики упаковывают свой код в определенном формате и загружают его поставщику PaaS, который предоставляет соответствующие системы и запускает его автоматически. Поставщик выдает конечную точку DNS, подключенную к запущенному приложению клиента, которую клиент может затем настроить с помощью записи DNS `CNAME`.

Хотя предложения PaaS значительно упрощают управление инфраструктурой, они жертвуют гибкостью и настройкой. Большинство предложений либо не позволяют административный доступ к оболочке, либо активно препятствуют ему. Пользователи PaaS должны принять определенные проектные решения, принятые поставщиком. Возможность пользователей реализовать некоторые функции может быть ограничена.

Google App Engine впервые разработал концепцию PaaS и остается одним из самых известных продуктов. App Engine поддерживает Python, Java, PHP и Go и имеет множество вспомогательных функций, таких как выполнение запланированных заданий на основе планировщика заданий **`cron`**, программный доступ к журналам, обмен сообщениями в реальном времени и доступ к различным базам данных. Google App Engine считается лидером в области предложений PaaS.

Конкурирующий продукт от компании AWS называется Elastic Beanstalk. В дополнение ко всем языкам, поддерживаемым App Engine, он поддерживает контейнеры Ruby, Node.js, Microsoft .NET и Docker. Он интегрируется с гибкими балансировщиками нагрузки ELB и функцией автоматического масштабирования AWS, используя мощность экосистемы AWS.

На практике мы обнаружили, что Elastic Beanstalk представляет собой смешанную картину. Настройка возможна через каркас расширения, который является запатентованным и запутанным. Пользователи по-прежнему несут ответственность за запуск экземпляров EC2, на которых размещено приложение. Поэтому, хотя Elastic Beanstalk отлично подходит для прототипирования, мы считаем, что система не является хорошим выбором для рабочих нагрузок, состоящих из множества служб.

Компания Heroku — еще один уважаемый поставщик в этой области. Приложение Heroku разворачивается в легком динамическом контейнере для Linux. Пользователи контролируют развертывание динамического контейнера. Компания Heroku имеет разветвленную сеть партнеров, которые предлагают базы данных, балансировку нагрузки и другие интеграции. Цена Heroku выше, чем на некоторые другие предложения, отчасти потому, что его собственная инфраструктура скрытно работает на серверах AWS.

Статический хостинг содержимого

Кажется нецелесообразным запускать операционную систему только ради размещения статических веб-сайтов. К счастью, облачные провайдеры могут разместить их для вас. В AWS S3 вы создаете отдельную область памяти для своего содержимого (так называемую "корзину"), а затем настраиваете запись CNAME из своего домена на конечную точку внутри провайдера. В Google Firebase используется инструмент командной строки для копирования вашего локального содержимого на серверы компании Google, которая предоставляет сертификат SSL и размещает ваши файлы. В обоих случаях можно использовать свое содержимое из CDN для повышения производительности.

🕮 Дополнительную информацию о сетях доставки содержимого см. в разделе 19.2.

Бессерверные веб-приложения

AWS Lambda — это вычислительная служба на основе событий. Разработчики, использующие службу Lambda, пишут код, который запускается в ответ на событие, такое как приход сообщения в очередь, новый объект в корзине или даже HTTP-запрос. Служба Lambda передает полезную нагрузку и метаданные события в качестве входных данных для пользовательской функции, которая выполняет обработку и возвращает ответ. Управлять экземплярами или операционными системами не требуется.

Чтобы обрабатывать HTTP-запросы, служба Lambda используется совместно с другой службой AWS под названием API Gateway, прокси-сервером, который может масштабироваться до сотен тысяч одновременных запросов. Перед источником вставлен интерфейс API Gateway, чтобы добавить такие функции, как управление доступом, ограничение скорости и кеширование. HTTP-запросы принимаются шлюзом API, и когда приходит запрос, шлюз запускает функции службы Lambda.

В сочетании со статическим хостингом на S3, Lambda и API Gateway могут привести к полностью бессерверной платформе для запуска веб-приложений, как показано на рис. 19.5.

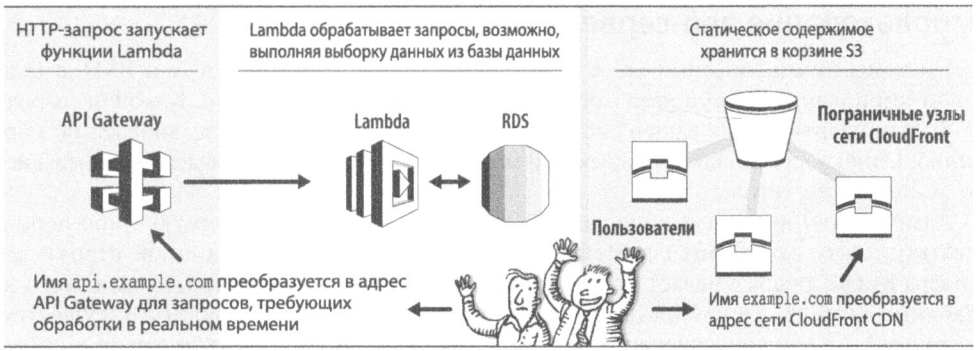


Рис. 19.5. Бессерверный веб-хостинг на основе AWS Lambda, API Gateway и S3

Эта технология все еще находится на начальном этапе, но уже меняет механику размещения веб-приложений. Мы ожидаем, что усовершенствования, каркасы, конкурирующие службы и передовая практика появятся уже в ближайшие годы.

19.4. Веб-сервер Apache httpd

Веб-сервер **httpd** является вездесущим среди множества разновидностей, реализованных в системах UNIX и Linux. Он переносится во многие архитектуры, а готовые пакеты существуют для всех основных систем. К сожалению, поставщики операционных систем имеют разные и сильно укоренившиеся подходы к конфигурированию **httpd**.

Модульная архитектура является основным фактором успеха Apache. Динамические модули можно подключать с помощью файла конфигурации, предлагая альтернативные варианты аутентификации, повышенную безопасность, поддержку для запуска кода, написанного на большинстве языков, перезапись URL-адресов и многие другие функции.

В основном по историческим причинам Apache имеет подключаемую систему обработки соединений на основе многопроцессных модулей (multi-processing modules — MPM), которая определяет, как HTTP-соединения обрабатываются на сетевом уровне ввода-вывода. Событийная (`event`) MPM — это современная технология, которая рекомендуется к использованию вместо устаревших `worker` и `prefork`.[16]

Для привязки к привилегированным портам (с номерами меньше 1024, такими как HTTP-порт 80 и HTTPS-порт 443), начальный процесс **httpd** должен выполняться с правами `root`. Затем этот процесс запускает дополнительные рабочие модули под локальной учетной записью с более низкими привилегиями для обработки фактических запросов.

Серверы **httpd**, которые не должны прослушивать порты 80 или 443, могут выполняться полностью без привилегий `root`.

Веб-сервер **httpd** настраивается с помощью директив в текстовых файлах конфигурации, которые используют отличительный синтаксис Apache. Хотя существуют сотни директив, администраторы обычно должны настраивать только несколько. Директивы и их значения документируются непосредственно в стандартных файлах конфигурации, которые поставляются с операционной системой, а также доступны на веб-сайте Apache.

[16] Устаревшее программное обеспечение, которое не считается потокобезопасным, например `mod_php`, должно использовать модуль `prefork` MPM. При этом для обработки каждого соединения будет использоваться отдельный процесс, а не поток.

Использование веб-сервера httpd

Системы инициализации **init** в операционных системах System V и BSD, а также демон инициализации **systemd** могут управлять веб-сервером **httpd**. Какой бы вариант ни был стандартным для вашей системы, один из них должен быть установлен по умолчанию. Однако для отладки и настройки можно взаимодействовать с демоном независимо от сценариев запуска.

Администраторы могут либо запускать веб-сервер **httpd** напрямую, либо использовать утилиту **apachectl**.[17] Непосредственный запуск из командной строки веб-сервера **httpd** подразумевает прямой контроль над демоном сервера, но для этого необходимо помнить (и набирать!) все опции, а это нелегко. Помимо этого, существует **apachectl** — оболочка сценария вокруг **httpd**. Каждый поставщик операционной системы настраивает оболочку **apachectl**, чтобы она соответствовала соглашениям процесса **init**. Она может запускать, останавливать, перезагружать и показывать состояние веб-сервера Apache.

Например, вот как можно запустить сервер с конфигурацией по умолчанию:

```
# apachectl start
Performing sanity check on apache24 configuration:
Syntax OK
Starting apache24.
# apachectl status
apache24 is running as pid 1337.
```

В этом выводе из системы FreeBSD оболочка **apachectl** сначала выполняет проверку конфигурации, подобно **lint**, путем выполнения команды **httpd -t**, а затем запускает демон.[18]

Команда **apachectl graceful** ждет, пока закроются все открытые соединения, а затем перезагружает сервер. Эта функция удобна для обновления конфигурации сервера без прерывания активных подключений. Она доступна через системные сценарии запуска и остановки.

Используйте флаг **-f** утилиты **apachectl** для запуска Apache с настраиваемой конфигурацией, например:

```
# apachectl -f /etc/httpd/conf/custom-config.conf -k start
```

Некоторые поставщики отказываются от использования утилиты **apachectl** в пользу прямого запуска веб-сервера **httpd**. Обратитесь к главе 2, чтобы узнать, как настроить веб-сервер **httpd** для автоматического запуска во время начальной загрузки системы.

Конфигурация логистики веб-сервера httpd

Хотя вся конфигурация веб-сервера **httpd** может содержаться в одном файле, разработчики операционных систем обычно используют директиву `Include` для разделения стандартной конфигурации на несколько файлов и каталогов. Эта архитектура упроща-

[17] **httpd** — это имя двоичного файла демона и проекта. Разработчики системы Ubuntu взяли на себя смелость переименовать **httpd** в **apache2**, что соответствует названию пакета **apt**, но в целом приводит к небольшой путанице.

[18] Утилита **lint** — это программа UNIX, которая оценивает код на языке C в поисках возможных ошибок. Этот термин теперь более широко применяется к любому инструменту, который проверяет файлы программного обеспечения и конфигурации на наличие ошибок, опечаток и других проблем.

ет управление сервером и лучше подходит для автоматизации. Как и ожидалось, особенности иерархии конфигурации отличаются в зависимости от системы. В табл. 19.6 перечислены настройки сервера Apache для каждой из наших демонстрационных платформ, предусмотренные по умолчанию.

Таблица 19.6. Детали конфигурации Apache в зависимости от платформы

	RHEL/CentOS	Debian/Ubuntu	FreeBSD
Имя пакета	`httpd`	`apache2`	`apache24`
Корневой каталог	`/etc/httpd`	`/etc/apache2`	`/usr/local/etc/apache24`
Основной файл конфигурации	`conf/httpd.conf`	`apache2.conf`	`httpd.conf`
Конфигурация модулей	`conf.modules.d/`	`mods-available/` `mods-enabled/`	`modules.d/`
Конфигурация виртуальных хостов	`conf.d/`	`sites-available/` `sites-enabled/`	`Includes/`
Расположение журнала	`/var/log/httpd`	`/var/log/apache2`	`/var/log/httpd-*.log`
Пользователь	`apache`	`www-data`	`www`

Во время запуска веб-сервер **httpd**, обращается к основному файлу конфигурации, обычно к **httpd.conf**, и включает любые дополнительные файлы, которые указаны в директивах `Include`. По умолчанию файл **httpd.conf** содержит много комментариев и служит в качестве краткого справочника. Параметры конфигурации в этом файле можно сгруппировать по трем категориям:

- глобальные параметры, такие как путь к корневому каталогу конфигурации сервера **httpd**, пользователь и группа для запуска, модули для активации, а также сетевые интерфейсы и порты для прослушивания;
- разделы `VirtualHost`, которые определяют, как будут обрабатываться запросы к текущему домену (обычно перенесены в подкаталоги и включаются в основную конфигурацию с помощью директивы `Include`);
- инструкции для ответов на запросы, которые не соответствуют определению `VirtualHost`.

Многие администраторы обычно удовлетворены глобальными настройками и ограничиваются управлением несколькими разделами `VirtualHosts`.

Модули существуют независимо от ядра **httpd** и часто имеют свои собственные параметры конфигурации. Большинство поставщиков операционных систем предпочитают переносить конфигурацию модуля в отдельный подкаталог.

Настройка веб-сервера Apache в системах Debian и Ubuntu имеет особенности. Их структура подкаталогов, файлов конфигурации и символических ссылок создает более гибкую систему управления сервером, по крайней мере теоретически.

Эту загадку мы попытались прояснить на рис. 19.6. Главный файл **apache2.conf** включает все файлы из подкаталогов *** -enabled** в каталоге **/etc/apache2**. Эти файлы на самом деле являются символическими ссылками на файлы в каталогах *** -available**. Для каждого набора подкаталогов предоставляется пара команд конфигурации, которые создают и удаляют символические ссылки.

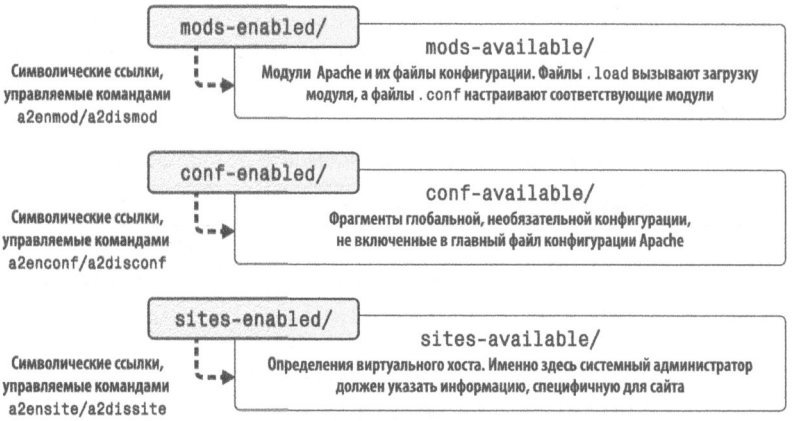


Рис. 19.6. Подкаталоги **/etc/apache2** *в системах на основе Debian*

По нашему опыту, система Debian не нужна и слишком сложна. Простой подкаталог конфигурации сайта **site-configuration** обычно обеспечивает достаточную структуру. Однако, если вы используете Debian или Ubuntu, имеет смысл придерживаться своих настроек по умолчанию.

Настройка виртуального хоста

Львиная доля конфигурации веб-сервера **httpd** заключается в определениях виртуальных хостов. Как правило, рекомендуется создавать отдельный файл конфигурации для каждого сайта.

Когда поступает HTTP-запрос, веб-сервер **httpd** определяет, какой виртуальный хост следует выбрать, обратившись к HTTP-заголовку `Host` и сетевому порту. Затем он сопоставляет часть пути запрашиваемого URL-адреса с директивами `Files`, `Directory` и `Location`, чтобы определить, как обслуживать запрошенное содержимое. Этот процесс отображения известен как маршрутизация запросов.

В следующем примере показана конфигурация протоколов HTTP и HTTPS для сайта `admin.com`.

```
<VirtualHost *:80>
   ServerName admin.com
   ServerAlias www.admin.com
   ServerAlias ulsah.admin.com
   Redirect / https://admin.com/
</VirtualHost>

<VirtualHost *:443>
   ServerName              admin.com
   ServerAlias             www.admin.com
   ServerAlias             ulsah.admin.com
   DocumentRoot            /var/www/admin.com/
   CustomLog               /var/log/apache2/admin_com_access combined
   ErrorLog                /var/log/apache2/admin_com_error
   SSLEngine               on
   SSLCertificateFile      "/etc/ssl/certs/admin.com.crt"
   SSLCertificateKeyFile   "/etc/ssl/private/admin.com.key"
```

```
   <Directory "/var/www/admin.com">
      Require all granted
   </Directory>
   <Directory "/var/www/admin.com/photos">
      Options +Indexes
   </Directory>
   <IfModule mod_rewrite.c>
      RewriteEngine on
      RewriteRule ^/(usah|lsah)$ /ulsah
   </IfModule>
   ExtendedStatus On
   <Location /server-status>
     SetHandler server-status
     Require ip 10.0.10.10/32
   </Location>
</VirtualHost>
```

Большая часть этого кода не нуждается в разъяснениях, но стоит отметить несколько деталей.

- Первая директива **VirtualHost** отвечает при обращении к порту 80 и перенаправляет все HTTP-запросы к сайтам `admin.com`, `www.admin.com` и `ulsah.admin.com` в HTTPS-запросы.
- Запросы к каталогу `admin.com/photos` выводят список всех файлов в этом каталоге.
- Запросы к каталогам `/usah` и `/lsah` переписываются в `/ulsah`.

Состояние сервера, отображаемое в этой конфигурации при доступе по URL `www.admin.com/server-status`, представляет собой модуль, который показывает полезную информацию о производительности во время выполнения, включая статистику о потреблении процессорного времени и памяти демона, состоянии запроса, среднем количестве запросов в секунду и т.д. Системы мониторинга могут использовать эту функцию для сбора данных о веб-сервере для оповещения, отчетности и визуализации HTTP-трафика. Здесь доступ к состоянию сервера ограничен одним IP-адресом, 10.0.10.10.

Базовая аутентификация протокола HTTP

В базовой схеме аутентификации протокола HTTP клиенты передают в HTTP-заголовке `Authorization` имя пользователя и пароль, зашифрованный по стандарту Base64. Если пользователь включает имя и пароль в URL (например, `https://user:pass@www.admin.com/server-status`), браузер автоматически выполняет перекодировку и помещает закодированное значение в заголовок `Authorization`.

Имя пользователя и пароль не зашифрованы, поэтому базовая аутентификация не обеспечивает конфиденциальности. Таким образом, ее безопасно использовать только в сочетании с протоколом HTTPS.

Обычная аутентификация в веб-сервере Apache настраивается в блоках `Location` или `Directory`. Например, следующий фрагмент требует аутентификации для доступа к URL `/server-status` (лучшая практика) и ограничения доступа к нему всего из одной подсети:

```
<Location /server-status>
   SetHandler server-status
   Require ip 10.0.10.0/24
```

```
    AuthType Basic
    AuthName "Restricted"
    AuthUserFile /var/www/.htpasswd
    Require valid-user
</Location>
```

Обратите внимание на то, что информация об учетной записи не хранится в файле конфигурации. Используйте утилиту **htpasswd** для создания учетной записи:

```
# htpasswd -c /var/www/.htpasswd ben
New password: <password>
Re-type new password: <password>
Adding password for user ben
# cat /var/www/.htpasswd
ben:$apr1$mPh0x0Cj$hfqMavkdHfVRVscE678Sp0
# chown www-data /var/www/.htpasswd          # Установка владельца
# chmod 600 /var/www/.htpasswd               # Ограничение доступа
```

Файлы паролей являются обычно скрытыми файлами, название которых начинается с точки, например, **.htpasswd**, но их можно назвать как угодно. Несмотря на то что пароли зашифрованы, для файла **.htpasswd** следует установить доступ только для чтения для того пользователя, от имени которого запущен веб-сервер. Эта предосторожность ограничивает способность злоумышленников видеть имена пользователей и запускать пароли через программное обеспечение для взлома.

Настройка TLS

Хотя имя SSL было заменено на TLS, в интересах обратной совместимости Apache сохраняет имя SSL для своих параметров конфигурации (как и многие другие пакеты программного обеспечения). Для настройки TLS требуется несколько строк:

```
SSLEngine                on
SSLProtocol              all -SSLv2 -SSLv3
SSLCertificateFile       "/etc/ssl/certs/admin.com.crt"
SSLCertificateKeyFile    "/etc/ssl/private/admin.com.key"
```

Здесь сертификат и ключ TLS находятся в самом сердце системы Linux, в каталоге `/etc/ssl`. Открытые сертификаты могут быть прочитаны кем угодно, но ключ должен быть доступен только пользователю мастер-процесса Apache, обычно `root`. Мы предпочитаем устанавливать разрешения `444` для сертификата и `400` для ключа.

Все версии фактического протокола SSL (предшественника TLS), как известно, являются небезопасными и должны быть отключены с помощью директивы SSLProtocol, показанной выше.

🕮 Полное описание руководства TLS на стороне сервера см. в разделе 19.7.

В некоторых из алгоритмов шифрования выявлены слабые места. Поддерживаемые алгориты шифрования на веб-сервере можно настроить с помощью директивы `SSLCipherSuite`. Рекомендации по точному использованию тех или иных настроек постоянно меняются. Руководство Mozilla *Server Side TLS* — лучший ресурс, о котором мы знаем, чтобы оставаться в курсе передовых методов использования TLS. Он также имеет удобный справочник о синтаксисе конфигурации для Apache, NGINX и HAProxy.

Запуск веб-приложений в Apache

Веб-сервер **httpd** может быть расширен с помощью модульной системы для запуска программ, написанных на Python, Ruby, Perl, PHP и других языках. Модули выполня-

ются внутри процессов Apache и имеют доступ к полному жизненному циклу HTTP-запроса/ответа.

Модули предоставляют дополнительные директивы конфигурации, которые позволяют администраторам управлять характеристиками среды выполнения приложений. В табл. 19.7 перечислены некоторые распространенные модули сервера приложений.

Таблица 19.7. Модули сервера приложений для `httpd`

Модуль	Язык	
`mod_php`	PHP	Устарел; использовать только с MPM `prefork`
`mod_wsgi`	Python	Интерфейс шлюза веб-сервера, стандарт Python
`mod_passenger`	Несколько	Гибкий, коммерчески поддерживаемый сервер приложений для нескольких языков, включая Ruby, Python и Node.js
`mod_proxy_fcgi`	Любой	Стандартный серверный интерфейс, который позволяет запустить программы на любом языке
`mod_perl`	Perl	Perl-интерпретатор, который реализован в **`httpd`**

В следующем примере настройки приложения Django Python для сайта `api.admin.com` используется модуль `mod_wsgi`:

```
LoadModule wsgi_module mod_wsgi.so

<VirtualHost *:443>
   ServerName api.admin.com
   CustomLog /var/log/apache2/api_admin_com_access combined
   ErrorLog  /var/log/apache2/api_admin_com_error

   SSLEngine on
   SSLCertificateFile    "/etc/ssl/certs/api_admin.com.crt"
   SSLCertificateKeyFile "/etc/ssl/private/api_admin.com.key"

   WSGIDaemonProcess admin_api user=user1 group=group1 threads=5
   WSGIScriptAlias / /var/www/api.admin.com/admin_api.wsgi

   <Directory /var/www/api.admin.com>
       WSGIProcessGroup admin_api
       WSGIApplicationGroup %{GLOBAL}
       Require all granted
   </Directory>
</VirtualHost>
```

После того как модуль **`mod_wsgi.so`** был загружен веб-сервером Apache, стали доступными несколько директив конфигурации WSGI. Файл `WSGIScriptAlias` в приведенной выше конфигурации **`admin_api.wsgi`** содержит код на Python, необходимый модулю WSGI.

Ведение журнала

Веб-сервер **`httpd`** предлагает лучшие в своем классе возможности ведения журнала, с детальным контролем над данными, которые регистрируются, и возможностью разделить данные журналов для разных виртуальных хостов. Администраторы используют эти журналы для отладки проблем конфигурации, обнаружения потенциальных угроз безопасности и анализа информации об использовании.

Пример сообщения журнала **`admin.com.access.log`** выглядит так:

```
127.0.0.1 - - [19/Jun/2015:15:21:06 +0000] "GET /search HTTP/1.1" 200
    20892 "-" "curl/7.38.0"
```

Это сообщение показывает:

- источник запроса; в данном случае `127.0.0.1`, т.е. локальный хост;
- временную метку;
- путь запрашиваемого ресурса (`/search`) и метода HTTP (`GET`);
- код состояния ответа (`200`);
- размер ответа;
- пользовательский агент (инструмент командной строки **`curl`**).

В документации для **`mod_log_config`** есть все сведения о том, как настроить формат журнала.

Загруженный веб-сайт генерирует большое количество журналов запросов, которые могут быстро заполнить диск. Администраторы несут ответственность за то, чтобы этого не произошло. Храните журналы веб-сервера в выделенном разделе диска, чтобы разросшийся до больших размеров журнальный файл не повлиял на работу остальной части системы.

🕮 Дополнительную информацию об утилите **`logrotate`** см. в разделе 10.5.

В большинстве дистрибутивов Linux установка пакета Apache по умолчанию включает в себя соответствующую конфигурацию **`logrotate`**. Система FreeBSD не имеет такого значения по умолчанию, и администраторы должны вместо этого добавлять запись в файл **`/etc/newsyslog.conf`** для журналов Apache.

Каталог журналов и файлы внутри него должны быть доступны для записи только пользователю основного процесса **`httpd`**, который обычно является суперпользователем `root`. Если бы у остальных пользователей был доступ для записи, то они могли бы создать символическую ссылку на другой файл, в результате чего он был бы перезаписан фиктивными данными. Системные значения по умолчанию настроены безопасно, поэтому избегайте изменений настройки владельца и группы.

19.5. Веб-сервер NGINX

Загруженный веб-сервер должен отвечать на многие тысячи одновременных запросов. Большая часть времени, необходимого для обработки каждого запроса, расходуется на ожидание поступления данных из сети или диска. Время, затрачиваемое на активную обработку запроса, намного короче времени ожидания.

Чтобы эффективно обрабатывать эту рабочую нагрузку, веб-сервер NGINX использует систему на основе событий, в которой всего несколько рабочих процессов обрабатывают множество запросов одновременно. Когда запрос или ответ (событие) готов к обслуживанию, рабочий процесс быстро завершает обработку и переходит к обработке следующего события. Прежде всего NGINX стремится избежать блокировки сетевого или дискового ввода-вывода.

Событие MPM, включенное в новые версии Apache, использует аналогичную архитектуру, но для сайтов с большим объемом и высокой производительностью NGINX остается основным программным обеспечением.

Администраторы, работающие с веб-сервером NGINX, заметят как минимум два процесса: главный и рабочий. Главный процесс выполняет основные обязанности, такие как открытие сокетов, считывание конфигурации и поддержание других процессов

NGINX. Рабочие процессы выполняют большую часть тяжелой работы с помощью обработки запросов. В некоторых конфигурациях используются дополнительные процессы, предназначенные для кеширования. Как и в Apache, главный процесс выполняется как `root`, поэтому он может открывать сокеты для любых портов с номерами меньше 1024. Другие процессы выполняются как менее привилегированный пользователь.

Количество рабочих процессов можно задавать. Хорошее эмпирическое правило состоит в том, чтобы запускать столько рабочих процессов, сколько процессорных ядер есть у системы. В системах Debian и Ubuntu именно так и настраивается NGINX по умолчанию, если он установлен из пакета. В системах FreeBSD и RHEL по умолчанию запускается один рабочий процесс.

Установка и запуск NGINX

Хотя популярность веб-сервера NGINX продолжает расти и он является основным продуктом среди самых загруженных веб-сайтов в мире, дистрибутивы операционных систем по-прежнему отстают от его поддержки. Версии, доступные в официальных хранилищах для систем Debian и RHEL, обычно устаревшие, хотя система FreeBSD обычно является более актуальной. Веб-сервер NGINX является продуктом с открытым исходным кодом, поэтому его можно построить и установить вручную. Веб-страница проекта, `nginx.org`, предлагает пакеты для утилит **`apt`** и **`yum`**, которые, как правило, более позднюю версию, чем те, которые поставляются с дистрибутивами ОС.

🕮 Дополнительную информацию о сигналах см. в разделе 4.2.

Для повседневной работы с веб-сервером **`nginx`** не требуется никаких особых приемов управления службами. Можно также запустить демон **`nginx`** во время разработки и отладки. Для указания другого файла конфигурации используйте аргумент **`-c`**. Параметр **`-t`** выполняет проверку синтаксиса файла конфигурации.

Веб-сервер **`nginx`** использует сигналы для запуска различных действий по обслуживанию, перечисленные в табл. 19.8. Убедитесь, что вы передаете их основному процессу **`nginx`** (обычно тому, у которого самый низкий PID).

Таблица 19.8. Сигналы, понятные демону nginx

Сигнал	Функция
`TERM` или `INT`	Немедленное прекращает работу
`QUIT`	Завершает и закрывает все текущие соединения, затем прекращает работу
`USR1`	Повторно открывает файлы журналов (используется для облегчения ротации журналов)
`HUP`	Перезагружает конфигурацию [а]
`USR2`	Изящно заменяет двоичный файл сервера без прерывания службы [б]

[а] Этот параметр проверяет синтаксис новой конфигурации, и, если синтаксис корректный, запускает новые рабочие процессы с новой конфигурацией, а затем аккуратно закрывает старые рабочие процессы.

[б] Подробнее о том, как это работает, см. в документации **`nginx`**.

Настройка веб-сервера NGINX

Формат конфигурации NGINX напоминает язык C. Он использует фигурные скобки, чтобы различать блоки строк конфигурации и точки с запятой для разделе-

ния строк. Основной файл конфигурации по умолчанию называется **nginx.conf**. Наиболее важные для конкретной системы аспекты конфигурации NGINX приведены в табл. 19.9.

Таблица 19.9. Детали конфигурации NGINX в зависимости от платформы

	RHEL/CentOS	Debian/Ubuntu	FreeBSD
Имя пакета	nginx[a]	nginx	nginx
Путь к демону	/sbin/nginx	/usr/sbin/nginx	/usr/local/sbin/nginx
Корневой каталог конфигурации	/etc/nginx	/etc/nginx	/usr/local/etc/nginx
Конфигурация виртуального хоста[б]	conf.d	site-available/ site-enabled/	Нет заранее заданного места
Пользователь по умолчанию	nginx	www-data	nobody

[a] Необходимо включить репозиторий программного обеспечения EPEL; см. fedoraproject.org/wiki/EPEL.

[б] Относительно корневого каталога конфигурации.

В файле **nginx.conf** блоки директив конфигурации, окруженные фигурными скобками, называются *контекстами*. Контекст содержит директивы, специфичные для данного блока конфигурации. Например, вот минимальная (но полная) конфигурация NGINX, которая показывает три контекста:

```
events { }

http {
    server {
        server_name www.admin.com;
        root /var/www/admin.com;
    }
}
```

Внешний контекст (называемый `main`) является неявным и настраивает базовую функциональность. События и контексты `http` находятся в контексте `main`. Контекст `event` необходим для настройки обработки соединений. Поскольку в этом примере этот контекст пуст, подразумеваются значения по умолчанию. К счастью, значения по умолчанию вполне разумные и определяют следующие действия.

- Запускается один рабочий процесс (используется непривилегированная учетная запись пользователя).
- Прослушивается порт 80, если сервер запущен с правами `root`, или порт 8000 в противном случае.
- Файлы журналов записываются в каталог **/var/log/nginx** (выбранный во время компиляции).

Контекст `http` содержит все директивы, относящиеся к HTTP-службам веб и прокси-серверу. Контексты `server`, определяющие виртуальные хосты, вложены в **http**. Lkz настройки нескольких виртуальных хостов используется несколько контекстов `server` в разделе `http`.

В раздел `server_name` можно включить псевдонимы, чтобы установить соответствие между заголовком `Host` и группой поддоменов.

```
http {
   server {
      server_name admin.com www.admin.com;
      root /var/www/admin.com;
   }
   server {
      server_name example.com www.example.com;
      root /var/www/example.com;
   }
}
```

🕮 Обзор регулярных выражений см. в разделе 7.3.

Значение `server_name` также может быть регулярным выражением, а совпадение может быть даже перехвачено и присвоено переменной, которая будет использоваться в конфигурации в дальнейшем. Используя эту функцию, можно реорганизовать предыдущую конфигурацию следующим образом.

```
http {
   server {
      server_name ~^(www\.)?(?<domain>(example|admin).com)$;
      root /var/www/$domain;
   }
}
```

Регулярное выражение, которое начинается с тильды, соответствует строкам `example.com` или `admin.com`, перед которыми могут стоять символы `www`. Значение соответствующего регулярному выражению домена сохраняется в переменной `$domain`, которая затем используется для определения того, корень какого сервера выбрать.[19]

Виртуальные хосты на основе имен можно отличить от хостов на основе IP-адреса, используя вместе директивы `listen` и `server_name`.

```
server {
   listen 10.0.10.10:80
   server_name admin.com www.admin.com;
   root /var/www/admin.com/site1;
}
server {
   listen 10.0.10.11:80
   server_name admin.com www.admin.com;
   root /var/www/admin.com/site2;
}
```

Эта конфигурация создает две версии сайта `admin.com`, которые обслуживаются из разных корневых каталогов. IP-адрес интерфейса, на который был получен запрос, определяет, какую версию сайта видит клиент.

Каталог `root` — это базовый каталог, в котором хранятся HTML, изображения, таблицы стилей, сценарии и другие файлы для виртуального хоста. По умолчанию веб-сервер NGINX просто выбирает файлы из каталога `root`, но для более сложной марш-

[19]Имейте в виду, что использование этого синтаксиса обязывает веб-сервер NGINX выполнять проверку регулярного выражения для каждого HTTP-запросом. Мы используем его здесь, чтобы продемонстрировать гибкость NGINX, но на практике вы, вероятно, захотите просто перечислить все возможные имена хостов в виде простого текста. Разумно использовать регулярные выражения в файле **`nginx.conf`**, но убедитесь, что они соответствуют реальным значениям, и старайтесь поддерживать их в нижней части иерархии конфигурации, чтобы они активизировались только в определенных ситуациях.

рутизации запросов можно использовать директиву `location`. Если заданный путь не соответствует директиве `location`, веб-сервер NGINX автоматически возвращается к базовому каталогу `root`.

В следующем примере используются директивы `location` и `proxy_pass`. Они инструктируют NGINX обслуживать большинство запросов из корневого каталога веб-хоста, но перенаправляют запросы к веб-сайту `http://www.admin.com/nginx` на сайт `nginx.org`.

```
server {
   server_name admin.com www.admin.com;
   root /var/www/admin.com;
   location /nginx/ {
       proxy_pass http://nginx.org/;
   }
}
```

Директива `proxy_pass` инструктирует веб-сервер NGINX действовать как прокси-сервер и перенаправлять запросы от клиентов на другой сервер. Мы еще не раз столкнемся с директивой `proxy_pass`, когда будем описывать использование NGINX в качестве балансировщика нагрузки в разделе "Балансировка нагрузки с помощью NGINX".

Директива `location` может использовать регулярные выражения для выполнения полноценной маршрутизации в разные источники на основе запрошенного содержимого. Описание того, как веб-сервер NGINX выполняет директивы `server_name`, `listen` и `location` для маршрутизации запросов, приводится в его официальной документации.

Чаще всего в дистрибутивах устанавливаются разумные стандартные значения для многих директив в глобальном контексте `http`, а затем используются директивы `include` для добавления виртуальных хостов для конкретного сайта в окончательную конфигурацию. Например, файл **`nginx.conf`** в системе Ubuntu по умолчанию содержит строку

```
include /etc/nginx/conf.d/*.conf;
```

Эта структура помогает устранить избыточность, поскольку все дочерние объекты наследуют глобальные настройки. Администраторам в простых средах, возможно, не нужно ничего делать, кроме как описать конфигурацию виртуального хоста, выраженную в контексте `server`.

Настройка TLS для NGINX

Несмотря на то что стиль конфигурации веб-сервера NGINX мало похож на стиль конфигурации Apache, его конфигурация TLS — это одна из областей, в которой это не так. Как и в Apache, все ключевые слова конфигурации относятся к SSL, более раннему имени TLS.

Включение TLS, а также указание файла сертификата и закрытого ключа выполняется следующим образом.

```
server {
   listen 443;
   ssl on;
   ssl_certificate /etc/ssl/certs/admin.com.crt;
   ssl_certificate_key /etc/ssl/private/admin.com.crt;
   ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
   ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE... # строка обрезана
   ssl_prefer_server_ciphers on;
```

```
    server_name admin.com www.admin.com;
    root /var/www/admin.com/site1;
}
```

Должны быть включены только фактические протоколы TLS (а не старые версии SSL); все протоколы SSL устарели. Права доступа к файлам сертификата и ключа должны соответствовать рекомендациям, изложенным в подразделе, посвященном настройке протокола TLS для веб-сервера Apache, раздела 19.4.

Используйте директиву `ssl_ciphers`, чтобы затребовать использование криптографически сильных алгоритмов шифрования и отключать более слабые шифры. Параметр `ssl_prefer_server_ciphers` в сочетании с `ssl_ciphers` инструктирует NGINX выбирать шифр из списка сервера, а не из клиента. В противном случае клиент мог бы предложить любой шифр, который ему понравился. (В предыдущем примере не отображается полный список алгоритмов шифрования, потому что соответствующий список довольно длинный; подробности представлены в руководстве Mozilla *Server Side TSL*. Если вы предпочитаете более короткий список шифров, загляните на сайт `cipherli.st`.)

Балансировка нагрузки с помощью NGINX

Веб-сервер NGINX также выступает мощным балансировщиком нагрузки. Его стиль конфигурации является гибким, но несколько неочевидным.

Для создания именованных групп серверов используйте модуль `upstream`. Например, следующий раздел определяет группу `admin-servers` как набор из двух серверов.

```
upstream admin-servers {
    server web1.admin.com:8080 max_fails = 2;
    server web2.admin.com:8080 max_fails = 2;
}
```

На группы `upstream` можно ссылаться из определения виртуальных хостов. В частности, они могут использоваться в качестве прокси-адресов, как и имена хостов.

```
http {
    server {
        server_name admin.com www.admin.com;
        location / {
              proxy_pass http://admin-servers;
              health_check interval=30 fails=3 passes=1 uri=/health_check
                  match=admin-health # в оригинале строка не разрывается
        }
    }
    match admin-health {
        status 200;
        header Content-Type = text/html;
        body ~ "Red Leader, Standing By";
    }
}
```

Здесь трафик сайтов `admin.com` и `www.admin.com` обрабатывается серверами `web1` и `web2` в циклическом порядке (по умолчанию).

Эта конфигурация также устанавливает проверки работоспособности для рабочих веб-серверов. Проверки выполняются каждые 30 с (`interval=30`) для каждого сервера в конечной точке `/health_check` (`uri=/health_check`). NGINX отмечает сервер как неработоспособный, если проверка работоспособности завершится неудачно после трех

последовательных попыток (fail=3), но вернет сервер в ротацию, если он пройдет проверку хотя бы один раз (pass=1).

Ключевое слово match особенно характерно для NGINX. Оно определяет условия, при которых проверка работоспособности считается успешной. В этом случае веб-сервер NGINX должен получить код ответа 200, заголовок Content-Type должен быть установлен равным text/html, а тело ответа должно содержать фразу "Red Leader, Standing By".

Мы добавили в контекст upstream дополнительное условие, которое устанавливает, что максимальное количество попыток подключения должно быть равным двум. Иначе говоря, если веб-сервер NGINX вообще не может подключиться к серверу после двух попыток, то он отказывается от этого сервера и удаляет его из пула. Эта проверка соединения дополняет более структурированные проверки из раздела health_check.

19.6. Программное обеспечение HAProxy

HAProxy является наиболее широко используемым программным обеспечением для балансировки нагрузки с открытым исходным кодом. Оно позволяет проксировать трафик по протоколам HTTP и TCP, поддерживает личные сеансы для привязки данного клиента к определенному веб-серверу и предлагает расширенные возможности проверки работоспособности. Последние версии также поддерживают протоколы TLS, IPv6 и сжатие для протокола HTTP. Поддержка HTTP/2 еще не завершена и, как ожидается, появится в версии HAProxy 1.7.

Конфигурация HAProxy обычно содержится в одном файле **haproxy.cfg**. Она настолько простая, что поставщики операционных систем обычно не усложняют себе жизнь и используют структуру каталогов по умолчанию, рекомендованную проектом.

В системах Debian и RHEL конфигурация находится в файле **/etc/haproxy/haproxy.cfg**. Система FreeBSD не предоставляет значение по умолчанию, так как на самом деле для него не существует разумной рекомендации — все полностью зависит от вашей конфигурации. Пример конфигурации в системе FreeBSD можно найти в каталоге **/usr/local/share/examples/haproxy** после установки пакета HAProxy.

Следующая простая примерная конфигурация позволяет HAProxy прослушивать порт 80 и распределять запросы с помощью циклического режима между двумя веб-серверами web1 и web2, запущенными на порту 8080.

```
global
   daemon
   maxconn 5000
defaults
   mode http
   timeout connect 5000 # Milliseconds
   timeout client 10000
   timeout server 10000
frontend http-in
   bind *:80
   default_backend webservers
backend webservers
   balance        roundrobin
   server web1    10.0.0.10:8080
   server web2    10.0.0.11:8080
```

Использование ключевых слов `frontend` и `backend` для конфигурирования HAProxy показано на рис. 19.7.

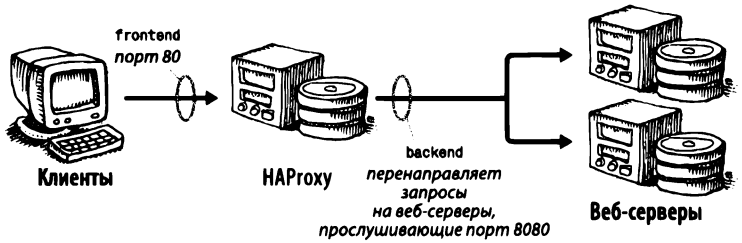


Рис. 19.7. Спецификации внешних и внутренних серверов HAProxy

Директива `frontend` определяет, как программа HAProxy будет обрабатывать запросы от клиентов: какие адреса и порты использовать, какие типы трафика обслуживать и другие ориентированные на клиента моменты.

Директива `backend` настраивает набор серверов, которые фактически обрабатывают запросы. В одной конфигурации могут существовать несколько пар директив `frontend/backend`, позволяя одному балансировщику нагрузки HAProxy обслуживать несколько сайтов.

Настройки `timeout` позволяют осуществлять тщательный контроль над тем, как долго система должна ожидать при попытке открытия нового соединения с сервером и как долго нужно держать соединения открытыми после их установки. Точная настройка этих значений важна для загруженных веб-серверов. В локальных сетях значение `timeout connect` может быть довольно низким (500 мс или меньше), потому что новые соединения устанавливаются очень быстро.

Проверки работоспособности

Хотя предыдущая конфигурация обеспечивает базовые функции, она не проверяет состояние нижестоящих веб-серверов. Если один из серверов `web1` или `web2` перестанет отвечать на запросы, половина входящих запросов даст сбой.

Функция проверки состояния HAProxy выполняет регулярные HTTP-запросы для определения работоспособности каждого сервера. Пока серверы отвечают кодом ответа HTTP 200, они считаются в работоспособном состоянии и продолжают получать запросы от балансировщика нагрузки.

Если сервер не проходит проверку работоспособности (возвращая что-либо, кроме кода состояния 200), то программа HAProxy удаляет его из пула. Однако HAProxy продолжает выполнять проверки работоспособности этого сервера. Если он снова начнет отвечать успешно, то программа HAProxy вернет его в пул.

Все параметры проверки работоспособности, такие как метод запроса, интервал между проверками и URL запроса, могут быть скорректированы. В следующем примере HAProxy выполняет запрос `GET` для каталога / на каждом сервере каждые 30 с:

```
backend webservers
   balance roundrobin
   option httpchk GET /
   server web1 10.0.0.10:8080 check inter 30000
   server web2 10.0.0.11:8080 check inter 30000
```

Конечно, неплохо, что существует возможность связаться с веб-сервером машины, но это едва ли последнее слово в отношении получения состояния сервера. Хорошо спроектированные веб-приложения обычно предоставляют конечную точку для проверки работоспособности, которая позволяет тщательно проанализировать приложение, чтобы определить его истинное состояние. Эти проверки могут включать проверку подключения к базе данных или кешу, а также мониторинг производительности. Используйте эти более сложные проверки, если они доступны.

Статистика сервера

Программное обеспечение HAProxy предлагает удобный веб-интерфейс, который отображает статистику сервера, так же как модуль **mod_status** в веб-сервере Apache. Версия HAProxy показывает состояние каждого сервера в пуле и позволяет вручную включать и отключать серверы по мере необходимости.

Синтаксис прост:

```
listen stats :8000
   mode http
   stats enable
   stats hide-version
   stats realm HAProxy\ Statistics
   stats uri /
   stats auth myuser:mypass
   stats admin if TRUE
```

Статистика сервера может быть настроена как внутри конкретного слушателя, так и внутри блоков `frontend` и `backend`, чтобы ограничить функцию только этой конфигурацией.

Липкие сессии

HTTP — это протокол без сохранения состояния, поэтому каждая транзакция является независимым сеансом. С точки зрения протокола запросы от одного и того же клиента не связаны между собой.

В то же время большинство веб-приложений нуждаются в возможности отслеживать поведение пользователей с течением времени. Классическим примером состояния является корзина для покупок. Пользователи просматривают магазин, добавляют товары в корзину, и, когда они готовы оплатить покупки, отправляют свои платежные данные. Веб-приложение должно иметь возможность для отслеживания содержимого корзины на протяжении нескольких просмотров страниц.

Для отслеживания состояния большинство веб-приложений используют систему `cookie`. Веб-приложение создает сеанс для пользователя и помещает идентификатор сеанса в параметр `cookie`, который отправляется обратно пользователю в заголовке ответа. Каждый раз, когда клиент отправляет запрос на сервер, вместе с ним отправляется и параметр `cookie`. Сервер использует значение параметра `cookie` для восстановления контекста клиента.

В идеале веб-приложения должны хранить свою информацию о состоянии в постоянном и общем хранилище, таком как база данных. Однако некоторые плохо управляемые веб-приложения сохраняют свои данные сеанса локально, в памяти сервера или на локальном диске. Когда они размещаются за балансировщиком нагрузки, эти приложения прерываются, потому что запросы одного клиента могут направляться на несколько серверов, в зависимости от капризов алгоритма планирования балансировки нагрузки.

Чтобы решить эту проблему, HAProxy может вставить собственный параметр `cookie` в ответы, что называется *липкими сессиями* (sticky sessions). Любые будущие запросы от одного и того же клиента будут содержать этот параметр `cookie`. HAProxy может использовать значение `cookie` для перенаправления запроса на один и тот же сервер.

Версия предыдущей конфигурации, модифицированная для поддержки липких сессий, выглядит следующим образом. Обратите внимание на добавление директивы `cookie`.

```
backend webservers
   balance roundrobin
   option httpchk GET /
   cookie SERVERNAME insert httponly secure
   server web1 10.0.0.10:8080 cookie web1 check inter 30000
   server web2 10.0.0.11:8080 cookie web2 check inter 30000
```

В этой конфигурации HAProxy поддерживает параметры `cookie SERVERNAME` для отслеживания сервера, с которым работает клиент. Ключевое слово `secure` указывает, что значение `cookie` следует отправлять только через TLS-соединения, а ключевое слово `httponly` сообщает браузерам использовать `cookie` только через протокол HTTP. Для получения дополнительной информации об этих атрибутах обратитесь к спецификации RFC6265.

Прекращение использования TLS

HAProxy 1.5 и более поздние версии включают поддержку TLS. Общая конфигурация заключается в прекращении использования протокола TLS на сервере HAProxy и взаимодействия с внутренними веб-серверами через обычный протокол HTTP. Этот подход разгружает внутренние веб-сервера от расшифровки криптографического содержимого и уменьшает количество систем, которым необходим закрытый ключ.

Для сайтов с особыми требованиями к безопасности также можно использовать протокол HTTPS для взаимодействия HAProxy с внутренними серверами. Для этого можно использовать один и тот же сертификат TLS или другой; в любом случае прокси-сервер должен будет прервать сеанс протокола TLS и повторно возобновить его для взаимодействия с внутренними веб-серверами.

Поскольку HAProxy прерывает соединение TLS с клиентами, необходимо добавить соответствующие параметры в блок конфигурации интерфейса.

```
frontend https-in
   bind *:443 ssl crt /etc/ssl/private/admin.com.pem
   default_backend webservers
```

Веб-серверы Apache и NGINX требуют, чтобы закрытый ключ и сертификат хранились в отдельных файлах в формате PEM, но HAProxy ожидает, что оба компонента будут присутствовать в одном файле. Можно просто объединить отдельные файлы для создания составного файла:

```
# cat /etc/ssl/{private/admin.com.key,certs/admin.com.crt}>
      /etc/ssl/private/admin.com.pem
# chmod 400 /etc/ssl/private/admin.com.pem
# ls -l /etc/ssl/private/admin.com.pem
-r -------- 1 root root 3660 Jun 18 17:46 /etc/ssl/private/admin.com.pem
```

Поскольку закрытый ключ является частью составного файла, убедитесь, что файл принадлежит пользователю `root` и не доступен для чтения другим пользователем. (Если

вы не запускаете HAProxy от имени `root`, поскольку вам не нужно получать доступ к каким-либо привилегированным портам, убедитесь, что право собственности на файл ключа совпадает с идентификатором, под которым запускается HAProxy.)

Все обычные рекомендации для TLS применяются к HAProxy: отключить протоколы эпохи SSL и явно настроить приемлемые алгоритмы шифрования.

19.7. Литература

- Adrian, David, et al. *Weak Diffie-Hellman and the Logjam Attack.* `weakdh.org`. На этой странице описывается атака Logjam на протокол обмена ключами Диффи–Хеллмана и предлагаются способы обеспечения безопасности систем.
- CloudFlare, Inc. `blog.cloudflare.com`. Это корпоративный блог сети доставки содержимого CloudFlare. Некоторые сообщения — всего лишь маркетинговая информация, но многие из них содержат информацию о последних тенденциях и технологиях в Интернете.
- Google, Inc. *Web Fundamentals.* `developers.google.com/web/fundamentals`. Это полезное руководство по различным передовым методам веб-разработки, включая разделы по проектированию сайта, пользовательским интерфейсам, безопасности, производительности и другим темам, представляющим интерес для разработчиков и администраторов. Особенно полезно описание кеширования.
- Grigorik, Ilya. *High Performance Browser Networking.* O'Reilly Media. 2013. Исключительное руководство по протоколам, преимуществам, ограничениям и аспектам производительности сети. Полезно как для разработчиков, так и для системных администраторов.
- IANA. *Index of HTTP Status Codes.* `www.iana.org/assignments/http-status-codes`.
- International Engineering Task Force. *Hypertext Transfer Protocol Version 2.* `http2.github.io/http2-spec`. Рабочий проект спецификации HTTP2.
- Mozilla. *Security/Server Side TLS.* `wiki.mozilla.org/Security/Server_Side_TLS`. Отличный ресурс, который документирует лучшие методы настройки TLS на многих платформах.
- Stenberg, Daniel. `daniel.haxx.se/blog`. Это блог Даниэля Стенберга, автора утилиты **`curl`** и авторитетного эксперта по HTTP.
- van Elst, Remy. *Strong Ciphers for Apache, nginx, and Lighthttpd.* `cipherli.st`. Правильная и безопасная настройка шифрования для Apache **`httpd`**, NGINX и веб-серверов **`lighttpd`**, а также инструмента для тестирования конфигурации TLS.

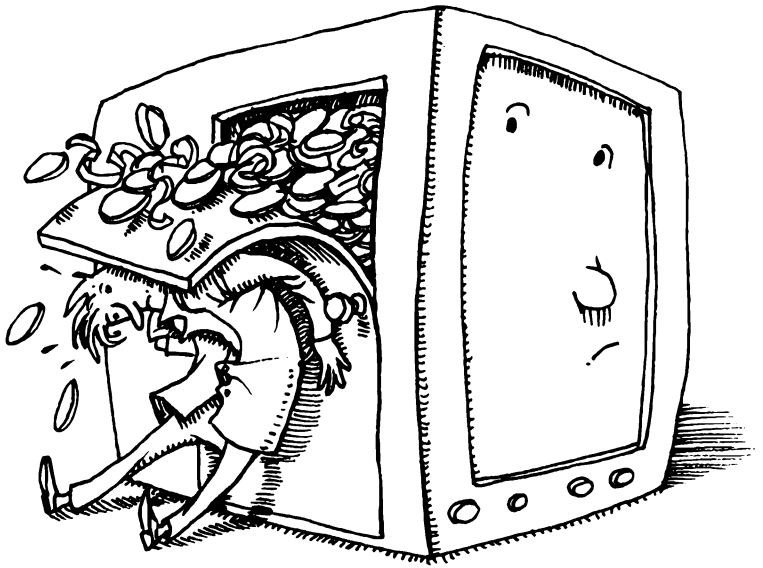
ЧАСТЬ III

Хранение данных



Глава 20

Дисковая память



Системы хранения данных все больше напоминают фрагменты огромной головоломки, которые можно соединять друг с другом, создавая бесчисленное множество вариантов. Из них можно собрать что угодно: от быстродействующего хранилища для критически важной базы данных до огромного архивного хранилища, в котором хранятся по три копии всех данных с возможностью отката в любую точку в прошлом.

Механические жесткие диски остаются доминирующим средством для создания интерактивных хранилищ данных, но при создании высокопроизводительных приложений они все чаще вытесняются твердотельными дисками (solid state drive — SSD), которые лучше подходят для приложений, предъявляющих повышенные требования к производительности. Лучшие стороны этих двух технологий можно объединить в одно целое с помощью программных и аппаратных кеширующих систем.

На облачных серверах обычно есть выбор оборудования для хранения, но за виртуальные диски с поддержкой SSD придется платить больше. Можно также выбирать из множества специфических типов хранения, таких как хранилища объектов, бесконечно расширяемые сетевые диски и реляционные базы данных как услуга.

Работу этого реального и виртуального оборудования обеспечивает множество программных компонентов, взаимодействующих с физическими устройствами хранения и иерархией файловой системы, которые видят пользователи. Эти компоненты включают в себя драйверы устройств, соглашения о разделении, массивы RAID, менеджеры логических томов, системы для виртуализации дисков по сети и сами реализации файловой системы.

В главе обсуждаются административные проблемы и решения, возникающие на каждом из этих уровней. Мы начнем с простых инструкций, позволяющих добавить базовый

диск в операционных системах Linux или FreeBSD. Затем опишем технологии, лежащие в основе современного аппаратного обеспечения для дисковой памяти, и рассмотрим общую архитектуру программного обеспечения для дисковой памяти. Затем мы опишем проблемы, связанные с дисковой памятью, начиная с низкоуровневого форматирования и заканчивая файловой системой. По ходу изложения рассмотрим вопросы, связанные с логическим разделением дисков, системами RAID и менеджерами логических томов.

Несмотря на то что все производители используют стандартизованное дисковое оборудование, в области программного обеспечения наблюдается огромное разнообразие. В главах 21 и 22 рассматриваются две широко распространенные файловые системы: NFS для совместного использования файлов в системах UNIX и Linux, и SMB для взаимодействия системах Windows и macOS.

20.1. Добавление диска

Прежде чем погрузимся в многостраничное повествование об архитектуре и теории дисковых систем, рассмотрим самый обычный сценарий: мы хотим инсталлировать жесткий диск и сделать его доступным посредством файловой системы. В этом сценарии нет ничего особенного: нет никакого массива RAID, все пространство диска находится на одном логическом томе, а тип файловой системы выбирается по умолчанию.

Шаг первый — подключить накопитель. Если рассматриваемая машина является облачным сервером, пользователь обычно создает виртуальный диск требуемого размера с помощью административного GUI-интерфейса провайдера (или с помощью своего интерфейса API), а затем присоединяет его к существующему виртуальному серверу в качестве отдельного шага. Обычно нет необходимости перезагружать сервер, поскольку облачные (и виртуальные) ядра распознают такие аппаратные изменения "на лету".

В случае физического оборудования диски, которые обмениваются данными через USB-порт, могут просто подсоединяться и включаться в работу на лету. Приводы SATA и SAS необходимо монтировать в отсеке, корпусе или специальной корзине. Хотя некоторые аппаратные средства и драйверы предназначены для обеспечения горячего добавления дисков SATA, эта функция требует аппаратной поддержки и необычна в оборудовании массового рынка. Перезагрузите систему, чтобы убедиться, что операционная система сконфигурирована правильно и адекватно реагирует на внесенные изменения во время загрузки.

Если вы используете настольную машину с оконной системой и все настроено правильно, система может предложить вам отформатировать новый диск при его подключении. Это особенно вероятно, если вы подключаете внешний USB-диск или флеш-накопитель. Опция автоматического форматирования обычно работает нормально; используйте ее по возможности. Однако после этого проверьте информацию о монтировании (запустив команду `mount` в окне терминала), чтобы убедиться, что диск не смонтирован с ограничениями, которых вы не хотели (например, с заблокированным выполнением программ или запретом полноценного владения файлами).

Если вы добавили диск вручную, крайне важно определить и отформатировать правильное дисковое устройство. Недавно добавленный диск необязательно представлен файлом устройства с наивысшим номером, а в некоторых системах добавление нового диска может изменить имена устройств существующих дисков (обычно после перезагрузки). Дважды проверьте идентификационные данные нового диска, просмотрев его производителя, размер и номер модели, прежде чем делать что-либо потенциально разрушительное! Используйте команды, упомянутые в следующих двух разделах.

Рецепт для Linux



Сначала выполните команду **lsblk**, чтобы получить список дисков системы и определить новый диск. Если этот вывод не дает вам достаточной информации, чтобы окончательно определить новый диск, вы можете указать модель и серийные номера с помощью команды **lsblk -o +MODEL,SERIAL**.

🕮 Описание таблицы разделов GPT см. в разделе 20.6.

Как только вы узнаете, какой файл устройства относится к новому диску (предположим, что это **/dev/sdb**), создайте таблицу разделов на диске. Это можно сделать с помощью нескольких команд и утилит, включая **parted**, **gparted**, **fdisk**, **cfdisk** и **sfdisk**; не имеет значения, какую из них вы используете, если она понимает таблицы разделов в стиле GPT. Утилита **gparted**, вероятно, самый простой вариант в системе с графическим интерфейсом пользователя. Ниже мы показываем рецепт на основе утилиты **fdisk**, которая работает во всех Linux-системах. (В некоторых системах команда **parted** по-прежнему не поддерживает схему разбиения диска в стиле GPT.)

```
$ sudo fdisk /dev/sdb
Welcome to fdisk (util-linux 2.23.2).

Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Command (m for help): g
Building a new GPT disklabel (GUID:
   AB780438-DA90-42AD-8538-EEC9626228C7)

Command (m for help): n
Partition number (1-128, default 1): <Enter>
First sector (2048-1048575966, default 2048): <Enter>
Last sector, +sectors or +size{K,M,G,T,P} (2048-1048575966, default
   1048575966): <Enter>
Created partition 1

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
```

Подкоманда **g** создает таблицу разделов GPT. Подкоманда **n** создает новый раздел; нажатие клавиши <Enter> в ответ на все вопросы утилиты **fdisk** выделяет все свободное пространство для нового раздела (раздел 1). Наконец, подкоманда **w** записывает новую таблицу разделов на диск.

Файл устройства для вновь созданного раздела совпадает с файлом устройства для диска с добавлением **1** к его имени. В приведенном выше примере разделом является **/dev/sdb1**.

Теперь вы можете создать файловую систему на устройстве **/dev/sdb1** с помощью команды **mkfs**. Опция **-L** присваивает файловой системе короткую метку (здесь, `spare`, т.е. запасной). Эта метка не меняется, даже если устройству, содержащему файловую систему, будет назначено другое имя во время последующей загрузки.

```
$ sudo mkfs -t ext4 -L spare /dev/sdb1
mke2fs 1.42.9 (28-Dec-2013)
Discarding device blocks: done
Filesystem label=spare
OS type: Linux
Block size=4096 (log=2)
...
```

Далее мы создаем точку монтирования и монтируем новую файловую систему.

```
$ sudo mkdir /spare
$ sudo mount LABEL=spare /spare
```

Как способ идентификации раздела мы могли бы указать **/dev/sdb1** вместо **LABEL=spare**, но это имя не обязательно будет работать в будущем.

Чтобы файловая система автоматически монтировалась во время загрузки, отредактируйте файл **/etc/fstab** и продублируйте одну из существующих записей. Измените имя устройства и точку монтирования в соответствии с параметрами приведенной выше команды монтирования. Например,

```
LABEL=spare       /spare       ext4        errors=remount-ro     0     0
```

Для идентификации файловой системы можно также использовать идентификатор UUID (см. раздел 20.10).

Подробное описание файлов устройств системы Linux приведено в разделе 20.4. Информацию о разделении диска см. в разделе 20.6. Информация о файловой системе ext4 приведена в разделе 20.10.

Рецепт для FreeBSD



Выполните команду **geom disk list** для отображения дисковых устройств, о которых известно ядру. К сожалению, система FreeBSD не раскрывает никакой информации, кроме имен и размеров устройств. С помощью команды **geom disk list** можно устранить любую двусмысленность относительно диска и узнать, какие устройства имеют существующие разделы. Неформатированный диск не должен иметь разделов.

Как только вы узнаете имя диска, вы можете создать на нем таблицу разделов и файловую систему. В этом примере мы предполагаем, что имя диска **ada1** и что мы хотим подключить новую файловую систему как **/spare**.

```
$ sudo gpart create -s GPT ada1     # Создаем таблицу разделов GPT
ada1 created

$ sudo gpart add -l spare -t freebsd-ufs -a 1M ada1 # Создаем раздел
ada1p1 added

$ sudo newfs -L spare /dev/ada1p1       # Создаем файловую систему
/dev/ada1p1: 5120.0MB (10485680 sectors) block size 32768, fragment
   size 4096
   using 9 cylinder groups of 626.09MB, 20035 blks, 80256 inodes.
super-block backups (for fsck_ffs -b #) at:
 192, 1282432, 2564672, 3846912, 5129152, 6411392, 7693632,
   8975872, 10258112
...
```

Параметр **-l** команды **gpart add** назначает текстовую метку к новому разделу. Метка делает раздел доступным через путь **/dev/gpt/spare** независимо от имени устройства, которое ядро назначает базовому дисководу. Опция **-L** команды **newfs** применяет аналогичную (но другую) метку к новой файловой системе, чтобы сделать раздел доступным как **/dev/ufs/spare**.

Смонтируйте файловые системы с помощью следующих команд.

```
$ sudo mkdir /spare
$ sudo mount /dev/ufs/spare /spare
```

Для того чтобы смонтировать файловую систему во время загрузки, добавьте ее в файл **/etc/fstab** (см. раздел 20.10).

20.2. Аппаратное обеспечение для хранения данных

Даже после появления Интернета существует всего несколько основных способов хранения компьютерных данных: на жестких дисках, во флеш-памяти, на магнитных лентах и на оптических носителях. Последние две технологии имеют существенные ограничения, которые не позволяют использовать их в качестве основной файловой системы. Тем не менее они по-прежнему иногда используются для резервного копирования и для автономных хранилищ, в которых мгновенный доступ и возможность перезаписи не являются первоочередными.

После 40-летнего господства традиционной технологии на магнитных дисках разработчики систем, ориентированных на производительность, наконец получили практическую альтернативу в виде твердотельных дисков (SSD). Эти устройства на основе флеш-памяти предлагают различные комбинации компромиссов со стандартным диском, которые будут влиять на архитектуру баз данных, файловых систем и операционных систем в течение многих лет.

В то же время традиционные жесткие диски продолжают экспоненциальное увеличение своей емкости. Тридцать лет назад, на заре форм-фактора 5,25″, который остается в употреблении и сегодня, жесткий диск емкостью 60 Мбайт стоил 1000 долл. Сегодня обычный накопитель емкостью 4 Тбайт стоит около 125 долл. Таким образом, сегодня за те же деньги можно сохранить более чем в 500 тысяч раз больше информации, т.е. отношение объема памяти к стоимости удваивалось каждые 1,6 года. В течение того же периода пропускная способность приводов массового рынка последовательно увеличивалась с 500 Кбайт/с до 200 Мбайт/с, что характеризуется сравнительно незначительным множителем, равным 400. Время поиска с произвольным доступом почти не изменилось. Чем больше меняется мир, тем больше в нем остается прежнего.

Размеры дисков указываются в гигабайтах, которые равны миллиардам байтов, в отличие от памяти, которая указывается гибибайтах, т.е. 2^{30} (1077741824) байтов. Разница составляет около 7%. Обязательно проверяйте свои устройства при оценке и сравнении емкости.

Жесткие диски и SSD-устройства очень похожи в том смысле, что они могут заменять друг друга, по крайней мере на аппаратном уровне. Они используют те же аппаратные интерфейсы и интерфейсные протоколы. И все же они имеют очень разные сильные стороны, как видно из табл. 20.1.

🕮 Дополнительную информацию о единицах IEC (гибибайтах и прочем) см. в разделе 1.6.

Таблица 20.1. Сравнение технологий HDD и SSD[а]

Характеристика	HDD	SDD
Типичный размер	< 16 ТБайт	< 2 Тбайт
Время произвольного доступа	8 мс	0,25 мс
Последовательное чтение	200 Мбайт/с	450 Мбайт/с
Произвольное чтение	2 Мбайт/с	450 Мбайт/с
IOPS[б]	150 операций/с	100000 операций/с
Стоимость	0,03 долл./Гбайт	0,26 долл./Гбайт
Надежность	Низкая	Низкая[в]
Ограничение количества записей	Нет	Теоретически

[а] Производительность и стоимость приведены по состоянию на середину 2017 г.

[б] Операций ввода-вывода в секунду.

[в] Меньше полных отказов устройств, чем у HDD, но более частые потери данных

В следующих разделах мы подробно рассмотрим каждую из этих технологий, а также более новую категорию устройств хранения данных — гибридные приводы.

Жесткие диски

Обычный жесткий диск состоит из нескольких вращающихся пластин, покрытых магнитным слоем. Считывание и запись данных осуществляются с помощью маленьких плавающих над поверхностью диска головок, смонтированных на металлическом рычаге, который перемещает их вперед и назад. Головки расположены близко к поверхности пластин, но не прикасаются к ним.

Считывание с пластины осуществляется довольно быстро. Однако перед этим нужно механически переместить головку к требуемому сектору диска, что существенно снижает скорость произвольного доступа к данным. При этом существует два основных источника задержки.

Рычаг, на котором расположены головки, должен резко перескакивать на соответствующую дорожку. Время, которое требуется для этого, называется *задержкой поиска* (seek delay). Затем система должна подождать, пока требуемый сектор не окажется под головкой при вращении пластины. Время, которое требуется для этого, называется *временем ожидания* (rotational latency). Если последовательность операций чтения организована оптимально, то диски могут передавать данные со скоростью до сотен мегабайтов в секунду, но скорость произвольного доступа в лучшем случае составляет несколько мегабайтов в секунду.

Совокупность дорожек на разных пластинах, которые находятся на одинаковом расстоянии от оси вращения, называется *цилиндром*. Данные, расположенные на цилиндре, можно считывать, не перемещая рычаг. Несмотря на то что головки работают чрезвычайно быстро, они перемещаются намного медленнее, чем вращается диск. Следовательно, любой доступ к диску, не требующий перемещения головки в новую позицию, будет выполняться быстрее.

Со временем скорость вращения пластин увеличилась. В настоящее время стандартная скорость вращения дисков массового производства, ориентированных на высокую скорость доступа, составляет 7200 оборотов в минуту (revolutions per minute — RPM), а лучшие диски достигают скорости 10 и 15 тысяч оборотов в минуту. Более высокая скорость вращения уменьшает время ожидания и увеличивает пропускную способность при передаче данных, но при этом диски сильнее нагреваются.

Надежность жестких дисков

Жесткие диски довольно часто выходят из строя. В 2007 году подразделение Google Labs, изучив 100 тыс. дисков, удивило техническое сообщество сообщением, что средний ежегодный процент отказов (annual failure rate — AFR) жестких дисков, проработавших более двух лет, превышает 6%, что намного больше показателя, предсказанного производителями на основе экстраполяции результатов краткосрочных тестирований. Поведение дисков можно описать так: несколько месяцев риска "детской смертности", два года безупречной работы, а затем ежегодный процент отказов взлетает до 6–8%. В целом вероятность пятилетнего "выживания" жестких дисков в исследовании компании Google не превышает 75%.

Интересно, что компания Google не нашла корреляции между процентом отказов и двумя факторами внешней среды, которые ранее считались важными, — температурный режим работы и активность диска. Полный отчет можно найти по адресу: `goo.gl/Y7Senk`.

Совсем недавно компания Backblaze, поставщик облачных хранилищ, стала регулярно публиковать свои сообщения о надежности различных моделей жестких дисков на сайте `backblaze.com/blog`. Эти данные на 10 лет более поздние, чем оригинальное исследование Google, но предлагают одну и ту же основную схему: сначала наблюдается высокий уровень сбоев, затем два-три года бесперебойной работы, а затем стремительный рост среднегодовой частоты отказа. Абсолютные цифры тоже довольно близки.

Виды сбоев и показатели надежности

Сбои жестких дисков обычно возникают из-за дефектов поверхности диска (плохих блоков) или механических повреждений. Приводы пытаются автоматически исправлять ошибки первого вида и переназначать восстановленные данные на другую часть диска. Когда секторы с ошибками становятся видны на уровне операционной системы (т.е. отражаются в системных журналах), это означает, что данные уже потеряны. Это плохой признак; вытащите привод из компьютера и замените его.

Прошивка и аппаратный интерфейс диска после сбоя обычно остаются работоспособными, и часто бывает интересно попытаться получить подробную информацию о том, что произошло с диском (см. раздел 20.4). Тем не менее диски настолько дешевы, что редко стоит делать это по экономическим соображениям, за исключением, возможно, обучения.

Надежность диска производители обычно измеряют с помощью среднего времени наработки на отказ (mean time between failures — MTBF), которое измеряется часами. Обычно значение показателя MTBF у промышленного диска составляет 1,2 млн часов. Однако этот показатель носит статистический характер и означает, что конкретный диск проработает 140 лет до первого сбоя.

Показатель MTBF является обратным по отношению к показателю AFR (Annualized Failure Rate, или частота сбоев за год), вычисленному для установившегося режима работы диска, т.е. после включения в систему, но до износа. Показатель MTBF, указываемый производителями и равный 1,2 млн часов, соответствует показателю AFR, равному 0,7% в год. Это значение почти (но не совсем) совпадает с показателем, выявленным компанией Google (1-2%) на протяжении первых двух лет работы их тестовых дисков.

Возможно, показатель MTBF, указываемый производителями, точен, но он получен на основе преднамеренного подбора данных о самом благополучном периоде работы дисков. Следовательно, этот показатель можно считать лишь верхней границей надежности, но его нельзя использовать для прогнозирования фактического процента отказов в долгосрочной перспективе.[1] Основываясь на приведенных выше данных, следует

[1] Наш технический рецензент Джон Корбет (Jon Corbet) называет этот показатель "уровнем надежности, достоверно не превышающим определенное значение".

разделить показатель MTBF, объявленный производителями, на 7,5 или перейти к более реалистичным методам выявления частоты отказов за пять лет.

Типы накопителей

Остались только два производителя жестких дисков: Seagate и Western Digital. Вы можете увидеть несколько других товарных марок для продажи, но все они в конечном итоге принадлежат тем же двум компаниям, которые на протяжении десяти лет занимались поглощением конкурентов.

Товарные марки жестких дисков подразделяются на нескольких общих категорий.

- **Эксклюзивные диски.** Эти продукты предлагают большое количество памяти для хранения данных при минимально возможной цене. Производительность не является приоритетом, но она обычно приличная. Сегодняшние низкопроизводительные приводы часто оказываются быстрее, чем высокопроизводительные приводы пять или десять лет назад.
- **Диски для массового рынка.** Это продукты, предназначенные для конечных пользователей (часто компьютерных игроков), имеют более высокую скорость вращения шпинделя и большую кеш-память, чем те, которые имеют их эквиваленты. В большинстве тестов они лучше, чем эксклюзивные диски. Как и в случае с эксклюзивными дисками, настройка прошивки подчеркивает однопользовательские шаблоны доступа, такие как большие последовательности операций чтения и записи. Эти диски часто работают в напряженном температурном режиме.
- **Накопители NAS.** NAS означает "сетевое хранилище" (network-attached storage), но эти диски предназначены для использования во всех типах серверов, RAID-системах и массивах — везде, где размещаются несколько дисков, работающих параллельно. Они предназначены для постоянной работы без выключения, а также для балансировки нагрузки, характеризуются высокой надежностью и низкой теплоотдачей. Контрольные показатели производительности, которые характеризуют однопользовательские шаблоны доступа, могут не выявить большой разницы в производительности по сравнению с эксклюзивными дисками, но NAS-накопители обычно более интеллектуально обрабатывают несколько потоков независимых операций ввода-вывода благодаря настройке прошивки. Накопители NAS часто имеют более длительную гарантию, чем эксклюзивные диски; их цена находится где-то между эксклюзивными и массовыми дисками.
- **Промышленные диски.** Слово "промышленный" (enterprise) в контексте жестких дисков может означать много разных вещей, но чаще всего оно означает "дорогой". Здесь вы найдете диски с интерфейсами, отличными от SATA, и необычные характеристики, такие как шпиндель с частотой вращения более 10 000 оборотов в минуту. Это, как правило, накопители премиум-класса с длительной (часто до пяти лет) гарантией.

Различия между этими категориями дисков наполовину реальные, а наполовину маркетинговые. Все классы дисков отлично работают во всех приложениях, но производительность и надежность могут различаться. Накопители NAS — это, пожалуй, лучший универсальный выбор для дисков, чтобы удовлетворить всевозможные потенциальные потребности.

Жесткие диски представляют собой товар. Одна модель может оказаться лучше другой, имея одинаковые частоту вращения шпинделя, интерфейс аппаратного обеспече-

ния и уровень надежности. В настоящее время существуют специализированные лаборатории, которые проводят сравнительный анализ конкурирующих дисков.

Надежность — другое дело. Данные компаний Google и Backblaze демонстрируют существенные различия между моделями. Наименее надежные чаще дают сбой, чем лучшие. К сожалению, нет никакого способа идентифицировать неудачные модели, пока они не будут проданы в течение года или двух и не создадут себе репутацию в реальном мире.[2]

Впрочем, это не имеет значения. Даже самые лучшие диски могут выйти из строя. Невозможно избежать необходимости резервного копирования и дополнительного хранилища, когда на карту поставлены важные данные. Проектируйте свою инфраструктуру на основе предположения, что диски могут выйти из строя, а затем выясните, сколько в этом контексте стоит более надежный диск.

Гарантии и списание

Поскольку жесткие диски чаще требуют гарантийного обслуживания, чем другие типы аппаратного обеспечения, длительность гарантии является важным фактором покупки. Промышленный стандарт сократился до ничтожных двух лет, подозрительно близко к длине среднегодового периода бесперебойной работы. Значительное преимущество имеет трехлетняя гарантия, предлагаемая на многие диски NAS.

Обмены жестких дисков по гарантии выполняются просто, если вы сможете продемонстрировать, что диски не прошли диагностический тест, предоставленный производителем. Тестовые программы обычно запускаются только в системе Windows и неприменимы в средах виртуализации и при использовании промежуточного аппаратного обеспечения для подключения дисков, такого как USB-переходники. Если ваши операции влекут за собой частые обмены дисков, вы можете счесть необходимым поддерживать выделенный компьютер с операционной системой Windows в качестве станции для тестирования дисков.

Как правило, следует проявлять особое внимание к появляющимся признакам выхода дисков из строя, даже если вы не можете достоверно подтвердить, что они достаточно неисправны, чтобы иметь право на обмен по гарантии. Даже, казалось бы, незначительные признаки (например, странные шумы или ошибки дискового сектора во временных файлах), вероятно, указывают на то, что диск скоро выйдет из строя.

Твердотельные диски

Твердотельные диски (SSD) распределяют операции считывания и записи по группам ячеек памяти, каждая из которых по отдельности работает медленнее, чем современные жесткие диски. Однако благодаря параллельной работе таких ячеек скорость обмена данными дисков SSD в целом соответствуют скорости работы традиционных жестких дисков и даже в несколько раз превосходит ее. Огромное преимущество SSD-дисков состоит в том, что они сохраняют свою скорость работы даже тогда, когда происходит произвольное обращение к данным для чтения или записи. Эта особенность является очень привлекательной в реальных приложениях.

Производители жестких дисков любят приводить скорость последовательной передачи данных для своей продукции, поскольку эти числа впечатляюще высоки. Однако

[2]Компания Hitachi (HGST, часть компании Western Digital) заслуживает признания как особо высоконадежная торговая марка. В течение последнего десятилетия ее диски последовательно лидировали на диаграммах надежности. Тем не менее диски HGST продаются значительно дороже своих конкурентов.

для традиционных жестких дисков этот показатель практически не имеет ничего общего с реальной скоростью его работы, наблюдаемой при чтении и записи произвольных секторов диска.[3]

Высокая скорость работы SSD-дисков не дается даром. Накопители SSD не только дороже в пересчете на один гигабайт хранимой информации, чем жесткие диски, но и порождают новые сложности и неопределенности. В марте 2009 г. Ананд Шимпи (Anand Shimpi) опубликовал статью о технологии SSD, которую мы рекомендуем как замечательное введение в эту тему. Ее можно найти на сайте `tinyurl.com/dexnbt`.

Ограниченное количество перезаписей

Каждая страница флеш-памяти на диске SSD (в современных дисках ее размер равен четырем двоичным килобайтам) может быть перезаписана ограниченное количество раз (обычно 100 тысяч, в зависимости от использованной технологии). Для того чтобы ограничить износ страницы, программное обеспечение дисков SSD ведет таблицы отображений и распределяет записи по всем страницам накопителя. Эти отображения остаются скрытыми от операционной системы, которая рассматривает накопитель как последовательный ряд блоков. Такой накопитель можно считать виртуальной памятью.

Теоретический предел для перезаписи флеш-памяти, вероятно, не так важен, как это может показаться. Для того чтобы его превысить, вам нужно в течение 15 лет непрерывно записывать на диск SSD 500 Гбайт данных со скоростью 100 Мбайт/с. Намного важнее долгосрочная надежность дисков SSD. На этот вопрос ответа пока нет. Нам достоверно известно, как работает диск SSD, произведенный пять лет назад, но современные продукты имеют совершенно другие свойства.

Флеш-память и типы контроллеров

Диски SSD производятся на основе нескольких типов флеш-памяти. Основное различие между типами связано с тем, сколько бит информации хранится в каждой отдельной ячейке флеш-памяти. Одноуровневые ячейки памяти (single-level memories — SLC) хранят один бит; это самый быстрый, но и самый дорогой вариант. Также в спектре дисков присутствуют многоуровневые ячейки (multilevel cells — MLC) и трехуровневые ячейки (triple-level cells — TLC).

Обзоры, посвященные дискам SSD, подробно описывают эти детали реализации как само собой разумеющееся, но неясно, почему покупатели должны об этом думать. Некоторые SSD работают быстрее, чем другие, но для понимания этого факта не требуется никакой конкретной технической подготовки. Стандартные тесты довольно хорошо отражают различия в производительности.

Теоретически флеш-память SLC имеет преимущество в надежности над другими типами. На практике надежность, по-видимому, больше связана с тем, насколько хорошо прошивка накопителя управляет памятью и сколько памяти производитель зарезервировал для замены проблемных ячеек.

Контроллеры, которые координируют SSD-компоненты, все еще эволюционируют. Некоторые из них лучше других, но в наши дни все основные предложения, как правило, вполне приемлемы. Если вы хотите потратить время на изучение аппаратного обе-

[3]Здесь нужно учитывать ваши реальные рабочие нагрузки. В обычных ситуациях, для которых на самом деле в значительной степени характерны последовательные операции чтения и записи, жесткие диски все еще могут конкурировать с твердотельными, особенно если учитывать затраты на оборудование.

спечения SSD, то лучше изучить репутацию контроллеров флеш-памяти, используемых для реализации твердотельных накопителей, чем рассматривать отдельные марки и модели дисков SSD. Производители дисков SSD обычно стараются не скрывать информацию о контроллерах, которые они используют. Однако если они сами не скажут вам об этом, то вы обязательно узнаете это из независимых обзоров.

Кластеры страниц и предварительное стирание

Еще одна сложность заключается в том, что информация со страниц флеш-памяти должна быть предварительно стерта перед записью в нее новых данных. Стирание памяти представляет собой отдельную операцию, которая выполняется медленнее, чем сама запись. Кроме того, невозможно стереть одну страницу, поскольку все смежные страницы кластера (как правило, 128 штук или 512 кибибайт) стираются одновременно. Если пул заранее стертых страниц исчерпан, то скорость записи SSD-накопителя может значительно снизиться и устройство должно восстановить страницы на лету, чтобы продолжить запись.

Перестроить буфер стертых страниц труднее, чем кажется, потому что файловые системы, разработанные для традиционных дисков, никак не помечают и не стирают блоки данных, которые больше не нужны. Накопитель не знает, что файловая система считает данный блок свободным; он знает только, что кто-то когда-то записал в него данные для хранения. Для того чтобы SSD-накопитель мог поддерживать работу кеша предварительно стертых страниц (а значит, обеспечивать высокую скорость записи), файловая система должна иметь возможность сообщать SSD-накопителю, что определенные страницы больше не нужны. Этой возможностью, которая называется операцией TRIM, в настоящий момент обладают практически все файловые системы. В рассматриваемых в книге примерах операционных систем единственной файловой системой, которая еще не поддерживает операцию TRIM, является файловая система ZFS в Linux.

Надежность дисков SSD

В 2016 г. Бьянка Шредер (Bianka Schroeder) и соавторы (`goo.gl/lzuX6c`) обобщили обширный набор данных, связанных с дисками SSD, полученный из центров обработки данных Google. Основные выводы приведены ниже.

- Технология памяти не имеет отношения к надежности. Надежность сильно варьируется среди моделей, но, как и в случае с жесткими дисками, ее можно оценить только ретроспективно.
- Большинство ошибок чтения происходят на уровне бит и корректируются с помощью технологии избыточного кодирования. Эти грубые (но исправляемые) ошибки чтения являются общими и ожидаемыми. Они происходят на большинстве накопителей SSD в течение всего времени их работы.
- Наиболее распространенным видом отказа является обнаружение в блоке данных битов, которые больше уже нельзя скорректировать, потому что это не позволяет принятая система кодирования. Эти ошибки обнаруживаются, но не могут быть исправлены; они обязательно влекут потерю данных.
- Даже среди самых надежных моделей SSD у 20% накопителей наблюдалась по крайней мере одна неисправимая ошибка чтения. Среди наименее надежных моделей — у 63%.
- Несмотря на то что возраст и интенсивность использования диска *должны влиять* на количество возникающих неисправимых ошибок, эта связь так и не была вы-

явлена до конца. В частности, в исследовании не было обнаружено никаких доказательств того, что старые твердотельные накопители представляют собой тикающие бомбы замедленного действия, которые постепенно приближаются к своему полному отказу.

- Поскольку неисправимые ошибки только незначительно коррелируют с рабочей нагрузкой, стандартная цифра надежности, указанная производителями, — уровень неисправимых ошибок в битах или UBER — не имеет смысла. Рабочая нагрузка мало влияет на количество наблюдаемых ошибок, поэтому надежность не следует характеризовать этим уровнем.

Наиболее заметным из этих выводов является тот, что нечитаемые блоки появляются часто и обычно возникают изолированно. При этом, как правило, диск SSD сообщает об ошибке блока, но затем продолжает нормально функционировать.

🕮 Дополнительную информацию о создании всеобъемлющей программы обследования см. в главе 28.

Конечно, ненадежные устройства хранения данных — не новость; поэтому следует всегда делать несколько резервных копий, независимо от того, какое оборудование вы используете. Однако уровень отказов SSD-дисков остается существенно ниже по сравнению с отказами жестких дисков. В отличие от жесткого диска, диски SSD редко будут требовать вашего внимания, пока сбой не станет явным и очевидным. Диски SSD требуют структурированного и систематического мониторинга.

Ошибки развиваются со временем независимо от рабочего цикла накопителя, поэтому твердотельные накопители, вероятно, не являются хорошим выбором для архивного хранения. И наоборот, отдельный сбойный блок не является признаком того, что диск SSD испортился или срок его полезной службы приближается к концу. В отсутствие более крупных сбоев целесообразно переформатировать такой диск и вернуть его в строй.

Гибридные диски

Проведя много лет в категории "экзотики", жесткие диски SSHD со встроенной флеш-памятью становятся все более востребованными. В настоящее время все их используют все больше и больше потребителей.

Аббревиатура SSHD означает "solid state hybrid drive" ("твердотельный гибридный диск" и является чем-то вроде триумфа маркетинга, спроектированным так, чтобы способствовать путанице с дисками SSD. SSHD — это просто традиционные жесткие диски с некоторыми дополнительными функциями на плате системной логики; в действительности, они являются такими же твердотельными, как и обычная посудомоечная машина.

Тесты современных продуктов SSHD, как правило, были не очень впечатляющими, особенно когда эталонные тесты пытались воспроизвести реальные сценарии их работы. Во многом это связано с тем, что в текущих устройствах обычно встроен только символический объем кеша из флеш-памяти .

Несмотря на то что эффективность современной технологии SSHD невелика, основная идея многоуровневого кеширования вполне обоснована и хорошо используется в таких системах, как ZFS и Apple Fusion Drive. Поскольку цена флеш-памяти продолжает падать, мы ожидаем, что диски на основе традиционных вращающихся пластин будут продолжать включать все больше и больше кеша. Эти продукты могут продаваться в качестве SSHD как явно, так и неявно.

Расширенный формат и блоки по 4 КиБ

На протяжении десятилетий стандартный размер дискового блока был зафиксирован на уровне 512 байт. Это слишком мало и неэффективно с точки зрения работы большинства файловых систем, поэтому в самих файловых системах 512-байтовые сектора объединяются в длинные кластеры страниц, размером от 1 до 8 КиБ, которые читаются и записываются вместе.

Поскольку реально ни в одном программном обеспечении при обмене данных с диском не выполняются операции чтения и записи данных по 512 байт, слишком неэффективно и расточительно для оборудования поддерживать такие крошечные сектора. Поэтому за последнее десятилетие индустрия хранения данных перешла на новый стандартный размер блока 4 КиБ, известный как *расширенный формат* (Advanced Format). Все современные устройства хранения используют внутри сектора по 4 КиБ, хотя большинство из них продолжают эмулировать 512-байтовые блоки для совместимости с устаревшими файловыми системами и ради удобства для конечного потребителя.

В настоящее время существуют три разных "мира", в которых может существовать устройство хранения данных.

- 512n, или реальные 512, — это старые устройства, которые фактически имеют 512-байтовые сектора. Эти устройства больше не производятся, но, конечно, они все еще есть в реальном мире. Эти диски ничего не знают об расширенном формате.
- 4Kn, или реальные 4K, — устройства, поддерживающие расширенный формат, которые имеют сектора по 4 КиБ (или страницы, как в случае SSD), и сообщают на главный компьютер, что их размер блока равен 4 КиБ. Все интерфейсные аппаратные средства и все программное обеспечение, которое напрямую связано с устройством, должны знать и готовиться к работе с блоками по 4 КиБ.
  4Kn — это волна будущего, но, поскольку она требует как аппаратной, так и программной поддержки, ее принятие будет постепенным. Приводы корпоративного уровня с интерфейсами 4Kn стали доступны в 2014 г., но на данный момент вы не рискуете столкнуться с диском 4Kn, если вы явно не закажете его.
- 512e, или эмулированные 512, — это устройства, использующие внутренние блоки по 4 КиБ, но сообщающие на главный компьютер, что их размер блока равен 512 байт. Прошивка в устройстве объединяет 512-байтовые блочные операции в операции с фактическими блоками хранения по 4 КиБ.

Переход с устройств 512n на устройства 512e был завершен в 2011 г. Эти две системы с точки зрения главного компьютера выглядят практически идентичными, поэтому устройства 512e отлично работают со старыми компьютерами и старыми операционными системами.

Единственное, что нужно знать об устройствах 512e, это то, что они чувствительны к несогласованности между размером кластера страниц файловой системы и блоками аппаратного диска. Поскольку диск может читать или записывать только страницы по 4 КиБ (несмотря на его эмуляцию традиционных 512-байтовых блоков), границы кластеров файловой системы и границы блоков жесткого диска должны совпадать. Крайне нежелательно, чтобы логический кластер файловой системы размером в 4 КиБ физически располагался в половине одного дискового блока в 4 КиБ и половине другого. В таком случае на диске потребуется выполнить в два раза больше операций чтения или записи физических блоков, чем требуется для обслуживания определенного количества логических кластеров.

Поскольку обычно кластеры файловой системы располагаются с самого начала дисковой памяти, выделяемой для них, можно решить проблему выравнивания, поместив начало дискового раздела на границу, адрес которой равен степени двойки и которая является большой по сравнению с вероятным размером диска и размера кластера файловой системы (например, 64 КиБ). Средства разделения дисков в современных версиях операционных систем Windows, Linux и BSD автоматически обеспечивают такое выравнивание. Тем не менее диски 512e, которые были неправильно распределены в устаревших системах, не могут быть безболезненно исправлены; вам придется запустить утилиту выравнивания, чтобы изменить границы раздела и физически переместить данные или же просто полностью стереть устройство и распределить его заново.

20.3. Интерфейсы устройств для хранения данных

В настоящее время существует лишь несколько наиболее распространенных стандартов интерфейса. Если система поддерживает несколько разных интерфейсов, следует использовать тот, который наилучшим образом удовлетворяет требования, предъявляемые к скорости, надежности, мобильности и стоимости.

Интерфейс SATA

Последовательный интерфейс ATA, SATA, является основным аппаратным интерфейсом для устройств хранения данных. Помимо поддержки высоких скоростей передачи (в настоящее время 6 Гбит/с), SATA имеет встроенную поддержку для "горячей" замены и (необязательно) поддержку очереди команд, две функции, которые в конечном итоге делают ATA жизнеспособной альтернативой интерфейсу SAS в серверных средах.

Кабели SATA легко входят в свои соединительные разъемы, но они могут так же легко выскользнуть оттуда. Доступны кабели с фиксаторами, но они имеют неоднозначную репутацию. На материнских платах с шестью или восемью разъемами SATA, упакованными вместе, может быть трудно отсоединить блокирующие соединители без пары узконосых плоскогубцев.

Интерфейс SATA также ввел стандарт кабелей для подключения внешних устройств, называемый eSATA. Эти кабели электрически идентичны стандартным SATA, но их разъемы немного отличаются. Вы можете добавить порт eSATA в систему с внутренними разъемами SATA, установив недорогой преобразователь на кронштейне.

Будьте внимательны при работе с внешними многодисковыми корпусами, которые имеют только один порт eSATA, потому что некоторые из них содержат интеллектуальные (RAID) контроллеры, для работы которых требуется собственный драйвер, а драйверы таких устройств редко поддерживаются в UNIX или Linux. Существуют и обычные корпуса, в которых встроен разветвитель портов SATA. Они потенциально могут использоваться в системах UNIX, но поскольку не все хост-адаптеры SATA поддерживают разветвление портов, обратите пристальное внимание на информацию о совместимости. Корпуса с несколькими портами eSATA — по одному на отсек для дисков — можно всегда использовать без особых проблем.

Интерфейс PCI Express

Шина PCI Express (Peripheral Component Interconnect Express — PCIe) используется на материнских платах персональных компьютеров более десяти лет. В настоящее время это основной стандарт для подключения всех типов дополнительных плат, а также видеокарт.

По мере развития рынка SSD стало ясно, что скорость 6 Гбит/с работы интерфейса SATA скоро станет узким местом при взаимодействии с самыми быстрыми устройствами хранения данных. Поэтому вместо традиционной формы 2,5-дюймового жесткого диска для ноутбуков, передовые SSD стали принимать форму печатных плат, которые подключаются непосредственно к шине PCIe системы.

Интерфейс PCIe был привлекательным благодаря своей гибкой архитектуре и быстрой скорости передачи. Версия, которая теперь является основной, PCIe 3.0, имеет скорость передачи 8 гигатранзакций в секунду (Гт/с). Фактическая пропускная способность зависит от того, сколько каналов передачи данных имеет устройство; их может быть всего лишь один или целых 16. Устройства с самой широкой полосой пропускания могут достигать пропускной способности более 15 Гб/с.[4] Ожидаемый в ближайшем будущем стандарт PCIe 4.0 удвоит базовую скорость передачи данных до 16 Гт/с.

Сравнивая интерфейсы PCIe и SATA имейте в виду, что скорость SATA, равная 6 Гбит/с указывается в *гига*битах в секунду. Полноценная PCIe на самом деле более чем в 20 раз быстрее, чем SATA.

Стандарт SATA подвергается давлению. К сожалению, экосистема SATA ограничена прошлыми вариантами дизайна и необходимостью поддерживать существующие кабели и разъемы. Маловероятно, что скорость интерфейсов SATA может быть значительно улучшена в течение следующих нескольких лет.

Вместо этого в последнее время основная работа была сосредоточена на попытке унификации SATA и PCIe на уровне разъемов. Стандарт M.2 для подключаемых плат маршрутизирует SATA, PCIe (с четырьмя каналами передачи данных) и USB 3.0 через стандартный разъем. Один или два из этих слотов теперь являются стандартными для портативных компьютеров, но их также можно найти на настольных системах.

Платы стандарта M.2 имеют ширину около дюйма и могут составлять до четырех дюймов в длину. Они тонкие, с обеих сторон разрешены только несколько миллиметров для монтажа компонентов.

U.2 — более новая версия M.2; она только начинает внедряться. Вместо USB, в дополнение к интерфейсам SATA и PCIe, через разъем U.2 можно подключать устройства с интерфейсом SAS.

Интерфейс SAS

Аббревиатура SAS означает Serial Attached SCSI, в которой часть SCSI представляет интерфейс Small Computer System Interface, общий канал данных, который когда-то соединял множество разных типов периферийных устройств. В наши дни рынок для периферийных соединений захватил интерфейс USB, и интерфейс SCSI можно встретить только в виде SAS, промышленного интерфейса, используемого для подключения большого количества устройств хранения данных.

Теперь, когда названия SAS и SCSI в значительной степени являются синонимами, обширная история различных технологий SCSI, относящихся к 1986 г., служит в основном для создания путаницы. Операционные системы вносят дополнительную неоднозначность, фильтруя весь доступ к диску через "подсистему SCSI" независимо от того, задействовано действительно устройство SCSI или нет. Наш совет — игнорировать всю эту историю и рассматривать SAS как отдельную систему.

[4]Эта скорость совсем немного не дотягивает до 16 Гб/с, потому что часть полосы пропускания занимают служебные сигналы. Однако объем накладных расходов настолько мал (около 1,5%), что его можно безопасно игнорировать.

Подобно SATA, интерфейс SAS представляет собой систему с прямыми связями (point-to-point system): вы подключаете диск в порт SAS через кабельную или прямую соединительную плату. Однако SAS позволяет "расширителям" подключать несколько устройств к одному хост-порту. Они аналогичны разветвителям портов SATA, но в то время как поддержка этих разветвителей портов иногда отсутствует, расширения SAS поддерживаются всегда.

В настоящее время SAS работает со скоростью 12 Гбит/с, что вдвое превышает скорость SATA.

В прошлых изданиях этой книги SCSI был очевидным выбором интерфейса для серверных приложений. Он предлагал самую широкую доступную полосу пропускания, выполнение команды вне очереди (прием, называемый очередью размеченных команд), более экономное использование центрального процессора, упрощение обработки большого количества устройств хранения данных и доступ к самым передовым жестким дискам рынка.

Появление SATA нивелировало или минимизировало большинство из этих преимуществ, поэтому SAS просто не дает явных преимуществ, которыми пользовался SCSI. Диски SATA конкурируют с эквивалентными дисками SAS почти в каждой категории (а в некоторых случаях превосходят их). В то же время как устройства SATA, так и интерфейсы и кабели, используемые для их подключения, дешевле и гораздо более широко доступны.

Интерфейс SAS все еще имеет несколько козырей.

- Производители продолжают использовать разделение SATA/SAS для стратификации рынка хранения. Чтобы оправдать премиальное ценообразование, самые быстрые и надежные диски по-прежнему доступны только с интерфейсами SAS.
- SATA ограничивает глубину очереди 32 операциями, в то время как SAS может обрабатывать тысячи.
- SAS может обрабатывать многие устройства хранения (сотни или тысячи) на одном интерфейсе хоста. Но имейте в виду, что все эти устройства подключаются к одному каналу; поэтому вы по-прежнему ограничены совокупной пропускной способностью в 12 Гбит/с.

Обсуждение SAS и SATA может в конечном итоге быть спорным, поскольку стандарт SAS включает поддержку дисков SATA. Соединители SAS и SATA настолько похожи, что через одну объединительную панель SAS можно подключить диски любого типа. На логическом уровне команды SATA просто туннелируются по шине SAS.

Эта конвергенция — удивительный технический подвиг, но его экономический смысл неясен. Затраты на установку SAS в основном связаны с хост-адаптером, объединительной панелью и инфраструктурой; диски SAS сами по себе не слишком дорогие. Вложив средства в настройку SAS, вы можете придерживаться этой технологии SAS до упора. (С другой стороны, возможно, скромные цены на диски SAS являются результатом того, что вместо них можно легко установить диски SATA.)

Интерфейс USB

Универсальная последовательная шина (USB) является популярной альтернативой для подключения внешних жестких дисков. Текущая скорость составляет от 4 Гб/с для USB 3.0 и до 10 Гб/с для USB 3.1.[5] Обе системы достаточно быстрые, чтобы обеспечить максимальную скорость передачи всех данных, уступая только самым быстрым

[5]Скорость USB 3.0 часто упоминается как 5 Гбит/с, но из-за накладных расходов, связанных с обязательной кодировкой, более вероятно, что фактическая скорость передачи составляет 4 Гбит/с.

дискам SSD. Однако избегайте использования интерфейса USB 2.0; его скорость достигает 480 Мбит/с, что мало даже для работы обычного жесткого диска.

У самих накопителей никогда не бывает встроенных интерфейсов USB. Внешние диски, продаваемые с этими интерфейсами, — это, как правило, диски SATA с преобразователем протокола (переходником), встроенным в корпус. Вы также можете приобрести эти корпуса отдельно и поместить в них свой жесткий диск.

USB-адаптеры также доступны в виде специальных корзин для подключения дисков и кабельных адаптеров. Это особенно удобно, когда диски часто меняются: просто вытаскиваете старый диск и вставляете новый.

USB-флешки являются совершенно законными устройствами хранения данных. Они представляют собой блочный интерфейс, аналогичный интерфейсу любого другого диска, хотя их скорость работы обычно посредственная. Основополагающая технология похожа на технологию SSD, но без некоторых преимуществ, которые дают дискам SSD их превосходную скорость и надежность.

20.4. Подключение и низкоуровневое управление накопителями

Способ подключения диска к системе зависит от используемого интерфейса. Остальное определяется монтажными кронштейнами и кабелями. К счастью, современные разъемы SAS и SATA имеют встроенную "защиту от дурака".

🕮 Дополнительную информацию о динамической работе с устройствами см. в разделе 11.3.

Интерфейс SAS поддерживает "горячую" замену дисков. Поэтому при его использовании вы можете сразу подключать и отключать диски без прерывания работы системы. Ядро ОС должно автоматически распознать новое устройство и создать для него соответствующий файл. Интерфейс SATA также может теоретически поддерживать "горячую" замену устройств. Следует заметить, что в стандарте SATA на этот счет нет никаких специальных требований, поэтому большинство производителей ради экономии не поддерживают эту возможность.

Даже если используемые вами интерфейсы допускают подключение устройств без перезагрузки системы (т.е. "горячее" подключение), все же безопаснее обесточить систему перед тем, как вносить изменения в ее аппаратное обеспечение. При работе с интерфейсами SATA "горячее" подключение зависит от реализации. Некоторые системные платы компьютеров не поддерживают эту функциональную возможность.

Целесообразно попытаться подключить жесткий диск к интерфейсу SATA, чтобы выяснить, работает ли горячее подключение в конкретной системе. Вы ничего не повредите. В худшем случае система просто проигнорирует диск.[6]

Проверка инсталляции на уровне аппаратного обеспечения

После инсталляции нового диска следует убедиться, что система знает о его существовании на самом низком из возможных уровней. Для персональных компьютеров это несложно: система BIOS показывает диски SATA и USB, подключенные к системе. Здесь

[6] "Горячее" подключение может показаться очень привлекательной функцией, которая открывает множество возможностей, например замены плохого диска без прерывания работы программ. Однако обеспечить безопасность и надежность этого приема на более высоких уровнях реализации хранилища данных очень сложно. В этой книге мы не описываем управление "горячим" подключением.

также могут быть включены диски SAS, если материнская плата поддерживает их напрямую. Если в системе имеется отдельная интерфейсная плата SAS, вам может потребоваться вызвать настройку BIOS для этой платы, чтобы просмотреть информацию о диске.

На облачных серверах и системах, поддерживающих горячее подключение дисков, вам, возможно, придется немного поработать. Проверьте диагностический вывод ядра после опроса устройства. Например, одна из наших тестовых систем вывела следующие сообщения для устаревшего диска SCSI, подключенного к хост-адаптеру BusLogic SCSI.

```
scsi0: BusLogic BT-948
csi: 1 host.
  vendor: SEAGATE  Model: ST446452W     Rev: 0001
  Type:   Direct-Access              ANSI SCSI revision: 02
Detected scsi disk sda at scsi0, channel 0, id 3, lun 3
scsi0: Target 3: Queue Depth 28, Asynchronous
SCSI device sda: hdwr sector=512 bytes. Sectors=91923356 [44884 MB]
     [44.8 GB]
```

Эту информацию можно увидеть после завершения загрузки системы: просто загляните в файлы системного журнала. Для получения дополнительной информации об обработке загрузочных сообщений из ядра обратитесь к разделу 10.3. Существуют несколько команд, выводящих список дисков, о которых система знает. В системах Linux лучшим вариантом обычно является команда **lsblk**, которая является стандартной для всех дистрибутивов. Для получения дополнительной информации запросите модель и серийные номера:

```
lsblk -o +MODEL,SERIAL
```

В системе FreeBSD используйте команду **geom disk list**.

Файлы дисковых устройств

Вновь добавленный диск представляется в системе в виде файлов дисковых устройств в каталоге **/dev**. Общую информацию о файлах дисковых устройств см. в разделе 5.4.

Все рассмотренные нами системы создают такие файлы автоматически, но системный администратор по-прежнему обязан знать, где искать файлы дисковых устройств и какие из них соответствуют новому диску. Форматирование не того накопителя — это прямой путь к катастрофе. В табл. 20.2 приведены соглашения об именах дисков, принятые в рассматриваемых операционных системах. Вместо демонстрации абстрактного шаблона, по которому дискам присваиваются имена, в табл. 20.2 приводится типичный пример выбора имени первого диска в системе.

Таблица 20.2. Стандарты именования дисков

Система	Диск	Раздел
Linux	**/dev/sda**	**/dev/sda1**
FreeBSD	**/dev/ada0**	**/dev/ada0p1**

Столбец, соответствующий дискам, содержат пути к диску, а столбец, соответствующий разделам, демонстрирует путь к конкретному разделу.

Например, **/dev/sda** в системе Linux — это первый диск, управляемый драйвером **sd**. Следующим диском будет **/dev/sdb** и т.д. В системе FreeBSD используются другие имена драйверов и числа вместо букв, но принцип остается тот же самый.

Не придавайте слишком большого значения именам драйверов, которые отображаются в файлах дисковых устройств. Современные ядра осуществляют управление дис-

ками SATA и SAS через общий уровень SCSI, поэтому не удивляйтесь, если SATA-диски маскируются как устройства SCSI. Названия драйверов также различаются в облачных и виртуализованных системах; виртуальный диск SATA может иметь или не иметь то же имя драйвера, что и фактический диск SATA.

В файлы устройств для разделов добавляется дополнительная информация, чтобы указать номер раздела. Нумерация разделов обычно начинается с 1, а не 0.

Непостоянные имена устройств

Имена дискам назначаются последовательно, по мере того, как ядро опрашивает различные интерфейсы и устройства в системе. Добавление диска может привести к тому, что существующие диски изменят свои имена. На самом деле даже перезагрузка системы может иногда приводить к изменениям имен.

Ниже приводится пара правил хорошего тона для системных администраторов.

- Никогда не вносите изменения в диски, разделы или файловые системы, не проверяя идентичность диска, на котором вы работаете, даже в стабильной системе.
- Никогда не указывайте имя файла дискового устройства ни в каком файле конфигурации, поскольку оно может измениться в самый неожиданный момент.

Последняя проблема наиболее примечательна при внесении изменений в файл **/etc/fstab**, содержащий перечень файловых систем, которые система монтирует во время загрузки. Когда-то было типично идентифицировать разделы диска по именам файлов своих устройств в **/etc/fstab**, но это уже не безопасно. Альтернативные подходы описаны в разделе 20.10.



В системе Linux предлагается несколько способов решения проблемы непостоянных имен дисковых устройств. Подкаталоги каталога **/dev/disk** содержат характеристики дисков, например их идентификационные номера, присвоенные производителем, или информацию о подключении. Эти имена устройств (которые на самом деле представляют собой обычные ссылки в каталоге верхнего уровня **/dev**) являются постоянными, но они длинные и громоздкие.

На уровне файловых систем и массивов дисков система Linux использует уникальные текстовые строки, которые постоянно идентифицируют объекты. Во многих случаях эти длинные строки скрыты, так что вам не придется работать с ними непосредственно.

Команда **parted -l** выводит размеры, таблицы разделов, номера моделей и производителей каждого диска, существующего в системе.

Форматирование дисков и управление сбойными секторами

Пользователи иногда под термином "форматирование" понимают процесс записи на диск таблицы разделов и создание в этих разделах файловых систем. Однако в данном случае мы используем термин "форматирование" для более фундаментальной операции настройки дисковой среды на аппаратном уровне. Мы предпочитаем называть эту операцию *инициализацией*, однако в реальном мире эти термины часто используются как синонимы, поэтому их смысл необходимо определять по контексту.

Процесс форматирования записывает на пластинах информацию об адресе и метки синхронизации, позволяющие определить положение каждого сектора. Он также выявляет сбойные секторы, т.е. дефекты носителя, которые приводят к появлению областей,

в которых невозможно надежно выполнять операции чтения и записи. Во всех современных дисках встроено управление сбойными секторами, поэтому ни вам, ни драйверу не нужно беспокоиться об управлении этими дефектами. Прошивка накопителя автоматически заменяет сбойные секторы хорошими, взятыми из области на диске, которая зарезервирована для этой цели.

Все жесткие диски поставляются заранее отформатированными, а заводское форматирование не хуже любого другого, которое вы можете выполнить в процессе эксплуатации диска. Лучше избегать использования низкоуровневого форматирования, если это не требуется. Не стоит переформатировать новые диски, как самоцель.

Если на диске обнаруживаются ошибки чтения или записи, сначала необходимо проверить кабели, терминаторы и адресацию. Каждый из них может вызвать симптомы, похожие на существование сбойных секторов. Если после проверки вы все еще уверены, что на диске есть дефект, лучше заменить его сразу на новый диск, а не ждать много часов, пока он отформатируется, в надежде, что проблемы исчезнут сами по себе.

Сбойные секторы, проявившиеся после форматирования диска, могут обрабатываться автоматически, но могут и не обрабатываться. Если микропрограмма управления накопителем полагает, что эти блоки можно восстановить надежным образом, то вновь выявленный дефект может быть исправлен на лету, а данные записаны в новое место. При выявлении более серьезных или менее очевидных ошибок накопитель прекращает выполнение операций чтений или записи и сообщает об ошибке операционной системе компьютера.

Диски SATA обычно не предназначены для форматирования за пределами завода изготовителя. Однако вы можете получить информацию о программном обеспечении для форматирования дисков от его производителей, как правило, для системы Windows. Убедитесь, что программное обеспечение предназначено именно для того накопителя, который вы собираетесь форматировать, и точно следуйте инструкциям производителя.[7]

Диски SAS форматируются сами по стандартной команде, поступающей от компьютера. Процедура отправки этой команды зависит от системы. На персональных компьютерах часто существует возможность посылать эту команду из системы BIOS контроллера SCSI. Для того чтобы отдать команду на форматирование диска SCSI от имени операционной системы, следует выполнить команду `sg_format` в системе Linux или `camcontrol` — в системе FreeBSD.

Для проверки целостности диска существуют разнообразные утилиты, которые выполняют запись данных в случайно выбранные области и считывают их оттуда. Тщательные тесты занимают много времени и, к сожалению, имеют небольшое прогностическое значение. Если вы подозреваете, что диск испорчен, и можете его просто заменить (или заказать новый в течение часа), то эти тесты можно проигнорировать. В противном случае запустите тест на ночь. Не беспокойтесь об "изнашивании" диска вследствие избыточного или агрессивного тестирования. Промышленные диски предназначены для постоянной работы.

Безопасное стирание дисков ATA

Начиная с 2000 года диски PATA и ATA реализуют команду "безопасного стирания", которая переписывает уничтожить все данные на диске, используя метод, разработанный производителями для защиты от несанкционированного извлечения информации. Безопасное стирание сертифицировано Национальным институтом стандартов и технологии (NIST) и в большинстве случаев вполне удовлетворяет пользователей. По класси-

[7] С другой стороны, если накопитель емкостью 1 Тбайт стоит 80 долл., зачем беспокоиться?

фикации Министерства обороны США, оно одобрено к использованию на уровне конфиденциальности ниже, чем "секретно".

Зачем вообще нужна такая функциональная возможность? Во-первых, файловые системы сами ничего не стирают, поэтому команды вроде **rm -rf *** оставляют данные, находящиеся на диске, нетронутыми и допускают их восстановление с помощью специальных программ.[8] Об этом всегда нужно помнить при избавлении от дисков, независимо от того, продаете ли вы их на аукционе eBay, или выбрасываете в мусорник.

Во-вторых, даже ручное перезаписывание каждого сектора на диске может оставлять магнитные следы, которые злоумышленник может выявить и восстановить в лаборатории. Безопасное стирание переписывает данные столько раз, сколько нужно для того, чтобы уничтожить эти следы. Остаточные магнитные сигналы для большинства организаций не составляют большой проблемы, но всегда полезно знать, что вы защищены от разглашения конфиденциальных данных организации. Некоторые организации выполняют безопасное стирание по требованию регуляторных органов или исходя из своих бизнес-правил.

В заключение, безопасное стирание делает накопители SSD абсолютно пустыми. Это позволяет повысить скорость их работы в ситуациях, когда нельзя использовать команду TRIM стандарта ATA (команду для физического стирания блока) либо потому, что файловая система, используемая в накопителе SSD, не способна ее выполнить, либо потому, что накопитель SSD подключен через адаптер компьютера или контроллер RAID, которые не поддерживают команду TRIM.

Команда безопасного удаления ATA защищена паролем на уровне диска, чтобы снизить риск ее непреднамеренного использования. Поэтому перед вызовом команды вы должны задать свой пароль на диске. Однако не торопитесь записывать пароль; при желании вы можете его всегда сбросить. Опасности блокировки диска не существует.



В системе Linux можно использовать команду **hdparm**.

```
$ sudo hdparm --user-master u --security-set-pass пароль /dev/диск
$ sudo hdparm --user-master u --security-set-erase пароль /dev/диск
```



В системе FreeBSD можно использовать команду **camcontrol**.

```
$ sudo camcontrol security диск -U user -s пароль -e пароль
```

Команда безопасного стирания в стандарте ATA не имеет аналогов в стандарте SCSI, но команда "форматировать модуль" в стандарте SCSI, описанная выше, представляется вполне разумной альтернативой.

Многие системы имеют утилиту **shred**, которая выполняет безопасное стирание отдельных файлов. К сожалению, ее работа основана на предположении, что блоки файлов могут быть перезаписаны на том же месте. Однако во многих ситуациях это условие не выполняется (в частности, это относится к любым файловым системам на любых накопителях SSD, любым логическим томам, имеющим механизм мгновенных копий, и, возможно, ко всей файловой системе ZFS), поэтому полезность универсальной утилиты **shred** вызывает сомнения.

Для очистки всей дисковой системы на персональном компьютере существует программа Дарика (Darik) Boot and Nuke (`dban.org`). Этот инструмент запускается со своего загрузочного диска, поэтому он не используется каждый день. Тем не менее он довольно удобен для вывода из эксплуатации старого аппаратного обеспечения.

[8] Теперь, когда большинство файловых систем поддерживают команду TRIM для информирования диска SSD о блоках, которые больше не нужны системе, это утверждение стало не совсем точным. Однако команда TRIM является необязательной; диск SSD не обязан стирать что-либо в ответ.

Команды hdparm и camcontrol: параметры диска и интерфейса (Linux)

Команды **hdparm** (Linux) и **camcontrol** (FreeBSD) имеют немного больше возможностей, чем команды безопасного стирания. Обычно они используются для взаимодействия со встроенными микропрограммами управления жесткими дисками SATA и SAS.

Так как эти утилиты работают на уровне, близком к аппаратному, они работают правильно только в невиртуализованных системах. На традиционном физическом сервере они на самом деле являются лучшим способом получить информацию о дисковых устройствах системы (**hdparm -I** и **camcontrol devlist**); мы не упоминаем их в другом месте (например, в рецептах добавления диска, приведенных в начале этой главы) только потому, что они не работают в виртуальных системах.

Команда **hdparm** происходит из доисторического мира дисков IDE и постепенно расширяется, охватывая возможности SATA и SCSI. Команда **camcontrol** задумывалась как альтернатива SCSI и была усовершенствована для покрытия некоторых функций SATA. Их синтаксисы различны, но в наши дни эти средства примерно эквивалентны.

Среди прочего эти инструменты могут устанавливать параметры мощности электропитания, влиять на уровень шума при работе диска, устанавливать флаг только для чтения и выводить подробную информацию о диске.

Мониторинг жесткого диска с помощью стандарта SMART

Жесткие диски представляют собой отказоустойчивые системы, использующие кодирование с исправлением ошибок, и встроенные микропрограммы с развитой логикой для сокрытия своих дефектов от операционной системы компьютера. В некоторых случаях жесткие диски сообщают операционной системе о неисправимой ошибке только после многочисленных попыток исправить ее. Было бы хорошо заранее распознавать такие ошибки, пока не разразился кризис.

Устройства SATA реализуют подробную форму отчетов о состоянии накопителя, которая иногда позволяет предсказать сбой. Этот стандарт под названием SMART ("self-monitoring, analysis, and reporting technology") предусматривает более 50 параметров, которые может анализировать компьютер.

Ссылаясь на исследование дисков, проведенное компанией Google (см. раздел 20.2), часто утверждают, что данные стандарта SMART не могут предсказать отказ накопителя. Это не совсем так. На самом деле компания Google обнаружила, что четыре параметра стандарта SMART обладают высокой прогнозной точностью, но сам сбой невозможно предотвратить с помощью изменения настроек стандарта SMART. Среди накопителей, давших сбой, 56% не содержали изменений в этих четырех параметрах. С другой стороны, мы считаем, что предсказание сбоя даже у половины дисков является неплохим результатом!

Четырьмя чувствительными параметрами стандарта SMART являются:

- количество ошибок сканирования (scan error count);
- количество повторного распределения памяти (reallocation count);
- количество автономного повторного распределения памяти (off-line reallocation count);
- количество секторов "с испытательным сроком" (probation).

Все эти значения должны быть равны нулю. Ненулевые значения этих параметров повышают вероятность отказа в течение 60 дней в 39, 14, 21 и 16 раз соответственно.

Для того чтобы извлечь пользу из параметров стандарта SMART, необходимо иметь специальное программное обеспечение, которое опрашивает накопители и прогнозирует вероятность сбоя. К сожалению, стандарты отчетов варьируются в зависимости от производителей накопителей, поэтому декодирование параметров не всегда является простой задачей. Большинство мониторов SMART накапливают основные параметры, а затем ищут внезапные изменения в неблагоприятном направлении, вместо того чтобы интерпретировать их абсолютные величины. (В соответствии с отчетом компании Google учет этих "мягких" параметров стандарта SMART в дополнение к "большой четверке" позволяет предсказать 64% сбоев.)

Стандартным программным обеспечением стандарта SMART для контроля накопителей в системах UNIX и Linux является пакет **smartmontools** с сайта `smartmontools.org`. В системах Red Hat, CentOS и FreeBSD он инсталлируется по умолчанию и обычно входит в репозиторий пакетов в других системах.

Пакет **smartmontools** состоит из демона **smartd**, который постоянно следит за накопителями, и команды **smartctl**, которую можно использовать для выполнения интерактивных запросов или сценариев. Демон имеет один конфигурационный файл, как правило, **/etc/smartd.conf**, содержащий подробные комментарии и много примеров.

Накопители SCSI имеют собственную систему отчетов о нестандартном состоянии, но, к сожалению, они менее подробные, чем отчеты системы SMART. Разработчики пакета **smartmontools** попытались включить накопители SCSI в свою схему, но прогнозируемая значимость данных стандарта SCSI является менее ясной.

20.5. Программное обеспечение накопителей

Если вам когда-нибудь приходилось подключать новый диск и система Windows спрашивала, не желаете ли вы его отформатировать, вы могли поразиться тому, насколько сложным является управление накопителями в системах UNIX и Linux. Почему же оно такое сложное?

Для начала отметим, что в системах UNIX и Linux все не настолько и сложно. Вы можете подключиться к своему настольному компьютеру с графическим интерфейсом, присоединить к нему накопитель USB и работать с ним практически так же, как в системе Windows. Вы получите возможность выполнить простую процедуру установки накопителя для хранения персональных данных. Если это все, что вам нужно, то все в порядке.

Как обычно, в этой книге нас интересуют промышленные системы хранения данных: файловые системы, доступ к которым имеет множество пользователей (локальных и удаленных) и которые в то же время должны быть надежными, высокопроизводительными, допускающими простое резервирование и обновление. Такие системы требуют немного больше внимания, и системы UNIX и Linux дают для этого достаточно оснований.

Типичный набор компонентов программного обеспечения, осуществляющих взаимодействие между накопителем и его пользователями, приведен на рис. 20.1. Архитектура, показанная на рис. 20.1, относится к системе Linux, но и другие операционные системы обладают аналогичными возможностями, хотя и не обязательно в точно таком же виде.

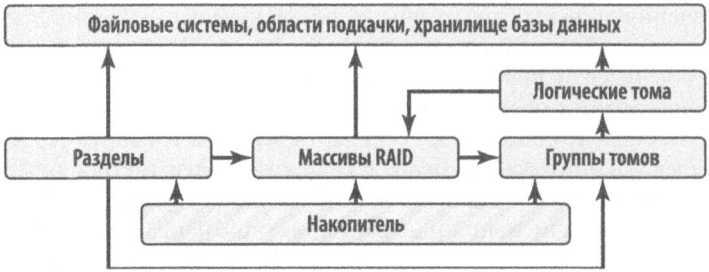


Рис. 20.1. Уровни управления накопителем

Стрелки на рис. 20.1 означают "могут быть созданы на основе". Например, файловая система Linux может быть создана на основе раздела, массива RAID или логического тома. Создание стека модулей, соединяющих каждый накопитель с его окончательным приложением, является одной из задач администратора.

Внимательный читатель заметит, что этот граф имеет цикл, а в реальных конфигурациях циклов не бывает. Система Linux допускает создание стеков из массивов RAID и логических томов в любом порядке, но ни один из компонентов не может использоваться больше одного раза (хотя с технической точки зрения это возможно).

Рассмотрим фрагменты рис. 20.1.

- **Накопитель** (storage device) — это все, что напоминает диск. Это может быть жесткий диск, флеш-накопитель, диск SSD, внешний массив RAID, реализованный в виде аппаратного обеспечения, и даже сетевая служба, обеспечивающая доступ к удаленному устройству на уровне блока. Конкретный вид оборудования значения не имеет, поскольку устройство допускает прямой доступ, обрабатывает блочный ввод-вывод и представлено файлом устройства.
- **Раздел** (partition) — это фрагмент накопителя, имеющий фиксированный размер. Каждый раздел имеет собственный файл устройства и действует как независимый накопитель. Для обеспечения эффективности разделы создаются с помощью того же драйвера, который используется для управления устройством. Большинство схем разделения занимает несколько блоков в начале накопителя для записи диапазонов блоков, которые занимает каждый раздел.
- **Группы томов и логические тома** ассоциируются с менеджерами логических томов (logical volume manager — LVM). Эти системы объединяют физические устройства и формируют пул накопителей, которые называются *группами томов*. Администратор может разделить этот пул на логические тома почти так же, как диски разбиваются на разделы. Например, диск объемом 6 Тбайт и диск объемом 2 Тбайт можно объединить в группу томов объемом 8 Тбайт, а затем разбить на два логических тома по 4 Тбайт. В результате один логический том будет содержать блоки данных, относящиеся к обоим жестким дискам.

  Поскольку менеджер логических дисков LVM добавляет уровень абстракции между логическими и физическими блоками, с его помощью можно зафиксировать логическое состояние тома, просто сделав копии таблицы отображения. Следовательно, менеджеры логических томов часто обеспечивают возможность получения "мгновенных копий" томов. После этого все последующие операции записи на том приводят к тому, что данные будут направляться в новые блоки, а менеджер LVM будет отслеживать как старую, так и новую таблицы отображения. Разумеется, менед-

жер LVM должен хранить как исходный образ, так и все модифицированные блоки, поэтому в конце концов он исчерпает память, если мгновенные копии никогда не удаляются.

- **Массив RAID** (избыточный массив недорогих и независимых дисков) объединяет многочисленные накопители в одно виртуальное устройство. В зависимости от настройки массива, эта конфигурация может повысить производительность (путем чтения и записи дисков в параллельном режиме) или надежность (путем дублирования и проверки данных с контролем четности на всех дисках) либо оба этих показателя.

  Как следует из названия, массив RAID обычно интерпретируется как совокупность обычных дисков, но современные реализации позволяют использовать в качестве компонента этого массива все, что функционирует как диск.

- **Файловая система** служит посредником между набором блоков, представленных разделом, массивом RAID, логическим блоком и стандартным интерфейсом файловой системы, ожидаемым программами: путями вроде `/var/spool/mail`, типами файлов в системах UNIX, правами доступа и т.п. Файловая система решает, где и как хранится содержимое файлов, как представить пространство имен и организовать поиск на диске, а также как избежать сбоя (и восстанавливаться после него).

  Большую часть накопителя занимает файловая система. Однако в некоторых операционных системах (не в текущих версиях Linux) на диске могут существовать специальные разделы для файла подкачки или хранилища баз данных. Такая конфигурация (т.е. без "помощи" файловой системы) может потенциально работать чуть быстрее. При этом ядро ОС или базы данных создает на диске собственные структуры данных, делая использование файловой системы излишним.

Если вы считаете, что описанная выше система содержит слишком много маленьких компонентов, которые просто реализуют один блочный накопитель на основе другого, то вы совершенно правы. В последнее время наблюдается тенденция в сторону консолидации этих компонентов, чтобы повысить эффективность и устранить дублирование. Хотя менеджеры логических томов изначально не функционировали как контроллеры RAID, большинство из них воплотили некоторые функциональные возможности массивов RAID (в частности, чередование данных по дискам и зеркалирование).

В настоящее время на передовом крае находятся системы, объединяющие файловую систему, контроллер RAID и систему LVM в один интегрированный пакет. Первым примером такого рода является файловая система ZFS, хотя файловая система Btrfs в Linux предназначена для решения аналогичных задач. Системы ZFS и Btrfs описываются в разделе 20.11. (Забегая наперед, скажем, что если у вас будет возможность использовать одну из этих систем, то обязательно попробуйте.)

Отображение устройств в системе Linux



Для простоты на рис. 20.1 мы опустили центральный компонент стека хранения Linux — механизм отображения устройств. Это многообещающий механизм, охватывающий несколько контекстов, среди которых основными являются реализация LVM2, уровни файловой системы для контейнеризации (см. главу 25) и механизм шифрования всего диска (ищите в поисковых машинах слово LUKS).

Механизм отображения устройств позволяет реализовать абстрактную идею создания одного блочного устройства на основе набора других блочных устройств. Он создает таблицу отображения устройств, на основе которой выполняется преобразование входящих запросов к диску и определение размещения конкретного блока данных на соответствующем устройстве.

Чаще всего механизм отображения устройств является частью реализации устройств хранения данных в системах Linux и не предполагает прямое взаимодействие с пользователем. Тем не менее вы можете увидеть его следы всякий раз, получая доступ к устройствам в каталоге **/dev/mapper**. Вы также можете настроить собственные таблицы отображения с помощью команды **dmsetup**, хотя случаи, в которых вам может потребоваться сделать это, относительно редки.

В следующих разделах мы более подробно рассмотрим уровни, относящиеся к конфигурации хранилища данных: разбиение на разделы, RAID, управление логическими томами и файловые системы.

20.6. Разбиение диска

Разбиение логического тома и управление логическим томом — это два способа разделения диска (или совокупности дисков в случае менеджера LVM) на отдельные части определенного размера. Операционные системы Linux и FreeBSD поддерживают оба этих метода.

Традиционно на самом низком уровне управления диском использовалось его разбиение на разделы, причем только диски можно было так разделить. Например, отдельные диски можно подключить к RAID-контроллеру или диспетчеру логических томов, но тогда нельзя будет разбить на разделы результирующие логические тома или тома RAID.

Правило, что на разделы можно разбить только диски, все чаще нарушается в пользу более общей модели, в рамках которой диски, разделы, пулы LVM и массивы RAID могут быть созданы на основе друг друга, в любом порядке или их комбинации. С точки зрения архитектуры программного обеспечения это красиво и элегантно. Но с точки зрения практичности у него есть неудачный побочный эффект, подразумевающий наличие разумной причины разбивать сущности, отличающиеся от дисков.

Фактически разбиение в большинстве случаев менее желательно, чем управление логическим томом. Этот подход грубый и ненадежный, к тому же он не имеет таких функций, как управление моментальными снимками. Решения о разбиении диска трудно пересмотреть впоследствии. Единственными заметными преимуществами разбиения над управлением логическим томом являются его простота и тот факт, что системы Windows и прошивки BIOS понимают эту структуру и могут с ней работать напрямую. В некоторых версиях систем UNIX, работающих на проприетарном оборудовании, полностью отказались от разбиения дисков, и никто, похоже, по нему не скучает.

И разбиение диска, и логические тома облегчают резервное копирование, предотвращая незаконный захват пользователями дискового пространства и потенциальные повреждения от программ, вышедших из-под контроля. Все системы имеют корневой раздел, монтирующийся на "/" и содержащий большинство данных о конфигурации локального компьютера. Теоретически для того, чтобы перевести систему в однопользовательский режим работы, достаточно одного корневого раздела. Различные подкаталоги (самые распространенные из них — **/var**, **/usr**, **/tmp**, **/share** и **/home**) можно поместить на собственные разделы диска или тома. Большинство систем имеет по крайней мере одну область подкачки.

Не существует единого мнения о наилучшем способе разбиения диска для конкретной системы. Как правило в каждой системе приняты свои правила разбиения диска по умолчанию, которые сравнительно просты. На рис. 20.2 показана традиционная схема разбиения двух дисков в системе Linux, при которой в одном разделе размещается одна файловая система. (При этом загрузочный диск не показан.)



Рис. 20.2. Традиционная схема разбиения дисков в системе Linux

Рассмотрим ряд факторов, влияющих на схему разбиения дисков.

- В далеком прошлом целесообразно было иметь резервный корневой накопитель, с которого можно было загрузиться в ситуации, когда с обычным корневым разделом пошло что-то не так. В настоящее время загружаемые диски USB и инсталляционные диски DVD операционных систем обладают более сильными функциями по восстановлению данных в большинстве систем. Резервный корневой радел создает больше проблем, чем решает.
- Размещение каталога **/tmp** в отдельной файловой системе ограничивает размеры временных файлов и позволяет не выполнять их резервное копирование. Некоторые системы для повышения производительности используют для хранения каталога **/tmp** файловую систему, расположенную в оперативной памяти. Файловые системы, расположенные в оперативной памяти, с случае необходимости переносятся в область подкачки, поэтому они хорошо работают в большинстве ситуаций.
- Поскольку файлы системного журнала хранятся в каталоге **/var/log**, целесообразно, чтобы каталог **/var** или **/var/log** занимал отдельный раздел диска. Если каталог **/var** расположен в маленьком корневом разделе, то легко переполнить этот раздел и вызвать аварийный останов всего компьютера.
- Пользовательские рабочие каталоги полезно размещать в отдельном разделе или каталоге. В этом случае, даже если корневой раздел будет поврежден или разрушен, пользовательские данные с высокой вероятностью останутся целыми. И наоборот, система сможет продолжать работу, даже если ошибочный пользовательский сценарий переполнит каталог **/home**.
- Разделение области подкачки между несколькими физическими дисками повышает скорость работы системы. Этот метод работает и по отношению к файловым системам; размещайте интенсивно используемые файловые системы на разных дисках. Эта тема освещается в разделе 29.2.
- Добавляя модули памяти в компьютер, увеличьте область подкачки. Более подробно виртуальная память описывается в разделе 29.6.
- Старайтесь размещать кластеры с быстро изменяющейся информацией в нескольких разделах диска, которые часто подвергаются резервному копированию.

- Организация Center for Internet Security публикует руководства по настройке для различных операционных систем по адресу `www.cisecurity.org/cis-benchmarks`. Они являются критериями лучшей практики. Документы включают полезные рекомендации для разбиения дисков на разделы и компоновки файловой системы.

Традиционное разбиение

Системы, допускающие разбиение, реализуют его, записывая "метку" в начало диска, чтобы определить диапазон блоков, включенных в каждый раздел. Подробности этой процедуры могут варьироваться; метка часто может существовать одновременно с другой установочной информацией (например, загрузочным блоком) и часто содержит дополнительную информацию, например имя или уникальный идентификатор всего диска.

Драйвер устройства, представляющий диск, считывает эту метку и использует таблицу разбиения диска для вычисления физического местоположения каждого раздела. Как правило, каждому разделу соответствует один или два файла устройства (для блочного устройства и для устройства посимвольного ввода-вывода; в системе Linux есть только блочные устройства). Кроме того, отдельный набор файлов устройств представляет диск в целом.

Несмотря на всеобщую доступность менеджеров логических томов, в некоторых ситуациях необходимо или желательно традиционное разбиение.

- В настоящее время используются только две схемы разбиения: MBR и GPT. Мы обсудим эти схемы в следующих разделах.
- На персональных компьютерах загрузочный диск должен иметь таблицу разбиения. Системы, произведенные до 2012 г., обычно требуют схему MBR, а некоторые из более современных — GPT. Большинство новых систем поддерживают обе схемы.
- Инсталлирование таблиц MBR или GPT делает диск понятным для системы Windows, даже если содержание его отдельных разделов остается недоступным. Если у вас нет конкретных планов по взаимодействию с системой Windows, то все же учтите повсеместную распространенность системы Windows, большую популярность виртуальных машин и вероятность того, что ваш диск в один прекрасный день может вступить с ней в контакт.
- Разделы занимают точно определенное место на диске и гарантируют локальность ссылок. Логические тома этого не обеспечивают (по крайней мере по умолчанию). В большинстве случаев это не имеет большого значения. Однако следует учитывать тот факт, что в традиционных дисках доступ к данным, расположенных в смежных цилиндрах, выполняется быстрее, чем если данные разнесены по удаленным цилиндрам. Кроме того, скорость обмена данными, расположенными на наружных цилиндрах диска (т.е. цилиндрах, содержащих блоки с наименьшими номерами) примерно на 30% выше, чем на внутренних.
- В системах RAID (см. раздел 20.8) используются диски или разделы сопоставимого размера. Хотя в конкретной реализации системы RAID могут использоваться диски разных размеров, скорее всего при этом будут использоваться только те диапазоны номеров блоков, которые являются общими для всех дисков. Чтобы не терять избыточное дисковое пространство из него можно сделать отдельный раздел для редко используемых данных. Если же поместить в этот раздел часто используемые данные, то такая схема разбиения снизит производительность работы всего массива RAID.

Разбиение диска по схеме MBR

Разбиение диска по схеме MBR (Master Boot Record) является старым стандартом компании Microsoft, созданным еще в 1980-х годах. Это неудобный и плохо продуманный формат, который не может поддерживать диски размером более 2 Тб. Кто тогда знал, что диски могут стать такими большими?

Схема MBR не имеет преимуществ перед схемой GPT, за исключением того, что это единственный формат, из которого старое компьютерное оборудование может загружать Windows. Если у вас нет веских причин использовать разделы MBR, то, как правило, не нужно этого делать. К сожалению, схема MBR по-прежнему используется по умолчанию при установке многих дистрибутивов операционных систем.

Метка MBR находится в одном 512-байтовом дисковом блоке, большую часть которого занимает программа начальной загрузки. При этом хватило места только для определения четырех разделов диска. Эти разделы называются *первичными*, поскольку они определяются непосредственно схемой MBR.

Один из первичных разделов можно определить как "расширенный". Это значит, что он содержит собственную вспомогательную таблицу разделов. К сожалению, расширенные разделы вызвали множество сложных проблем, поэтому их следует избегать.

Схема разбиения дисков в системе Windows позволяет пометить один из разделов как "активный". Загрузчики ищут активный раздел и пытаются загрузить операционную систему из него.

Кроме того, каждый раздел имеет однобайтовый атрибут типа, который обозначает содержимое раздела. Как правило, эти коды представляют либо типы файловой системы, либо операционные системы. Эти коды не присваиваются централизованно, но со временем были приняты определенные соглашения. Они сформулированы Андрисом Э. Брауэром (Andries E. Brouwer) на сайте `goo.gl/ATi3`.

Команда MS DOS, осуществляющая разбиение жесткого диска, называется **`fdisk`**. Большинство операционных систем, поддерживающих разбиение в стиле MBR, унаследовали это имя для обозначения своих собственных команд разбиения, но они очень разнообразны. Сама система Windows от нее отказалась, ее современная версия инструмента для разбиения диска называется **`diskpart`**. Кроме того, система Windows имеет графический пользовательский интерфейс для разбиения дисков, который доступен благодаря утилите Управление дисками (Disk Management) из консоли управления **`mmc`**.

Не имеет значения, с помощью какой операционной системы вы разбили диск на разделы — Windows или какой-то другой. Окончательный результат будет таким же.

Схема GPT: таблица разделов GUID

Проект компании Intel по созданию расширяемого микропрограммного интерфейса (extensible firmware interface — EFI) был направлен на замену неустойчивых соглашений, касающихся систем BIOS персональных компьютеров, более современной и функциональной архитектурой.[9] Схема разбиения EFI в настоящее время стала стандартом в новом аппаратном обеспечении персональных компьютеров и получила широкую поддержку со стороны операционных систем.

Схема разделения EFI, известная как "таблица разделов GUID", или GPT, устраняет очевидные недостатки метки MBR. Она определяет только один тип разбиения (больше

[9]Проект EFI недавно был переименован в UEFI. Буква U означает "unified" (универсальный). Этот проект поддерживается многими производителями. Однако название EFI употребляется чаще. По существу, названия UEFI и EFI являются синонимами.

нет никаких логических разделов в расширенном разделе), а самих разделов может быть сколько угодно. Каждый раздел имеет тип, определенный 16-байтовым идентификатором (глобально уникальным ID, или GUID), который не требует централизованного управления.

Следует подчеркнуть, что таблица GPT сохраняет простейшую совместимость с системами, использующими метку MBR, записывая метку MBR в первый блок таблицы разделов. Эта "ложная" метка MBR позволяет диску выглядеть так, будто он содержит только один раздел MBR (по крайней мере в пределах до 2 Тбайт). Само по себе это не имеет значения, но присутствие "ложной" метки MBR защищает диск от случайного переформатирования в обычных системах.

Версии системы Windows, начиная с версии Vista, открыли эру дисков GPT для хранения данных, но только системы, поддерживающие расширяемый микропрограммный интерфейс EFI, могут загружаться с этих дисков. Система Linux и ее загрузчик GRUB продвинулись дальше: они поддерживают диски GPT, с которых могут загружаться любые системы. Системы Mac OS для процессоров компании Intel, поддерживают обе схемы разбиения EFI и GPT.

Несмотря на то что схема разбиения диска GPT хорошо поддерживается во многих ядрах операционных систем, ее поддержка среди утилит управления дисками остается эпизодической. Убедитесь, что все утилиты, которые вы запускаете на диске с разбиением по схеме GPT, действительно поддерживают эту схему.

Разбиение дисков в системе Linux



В системах семейства Linux предусмотрено несколько вариантов разбиения дисков, которые только запутывают конечного пользователя, учитывая то, что некоторые из них не поддерживают GPT. По умолчанию используется команда **parted** — утилита командной строки, понимающая разные форматы меток (включая те, что использовались в системах Solaris). С ее помощью можно не только создать или удалить раздел диска, но и переместить его в другое место и даже изменить размер. В системе с графическим интерфейсом пользователя GNOME существует ее специальная версия, называемая **gparted**.

В целом мы рекомендуем использовать утилиту **gparted**, а не **parted**. Обе утилиты просты, но **gparted** позволяет задавать размеры разделов, а не начало и конец диапазона блоков. Для разбиения загрузочного диска лучше всего использовать графические инсталляторы из дистрибутивных пакетов, так как они обычно предлагают разбиение, которое является оптимальным для данного дистрибутива.

Разбиение дисков в системе FreeBSD



Как и Linux, система FreeBSD имеет несколько инструментов для разбиения дисков на разделы. Игнорируйте все, кроме **gpart**. Остальные существуют только для того, чтобы спровоцировать вас совершить какую-нибудь ужасную ошибку.

Таинственные объекты **geom**, которые вы увидите на справочной странице **gpart** (и в других связанных с хранилищем контекстах в системе FreeBSD), — это абстракция устройств хранения FreeBSD. Не все объекты **geom** — это дисковые накопители, но все

дисковые накопители являются объектами **geom**, поэтому при вызове команды **geom** вы можете использовать общее имя диска, такое как **ada0**.

В рецепте, посвященном добавлению диска в системе FreeBSD из раздела 20.1, для настройки таблицы разделов на новом диске используется команда **gpart**.

20.7. Управление логическими томами

Представьте себе ситуацию, в которой вы не знаете, насколько большим должен быть раздел. Через шесть месяцев после создания раздела выясняется, что он слишком большой, а в соседнем разделе недостаточно памяти. Знакомо? Менеджер логического тома позволяет вам динамически перераспределить память между переполненным разделом и разделом, испытывающим нехватку памяти.

Управление логическими томами, по существу, является максимально эффективной и абстрактной версией процедуры разделения диска. Оно группирует отдельные накопители в "группу томов". Блоки в группе томов могут быть затем выделены "логическим томам", которые представлены файлами блочных устройств и функционируют как разделы диска.

Однако логические тома являются более гибкими и мощными, чем разделы дисков. Перечислим ряд операций, которые можно выполнять с помощью менеджера тома.

- Перемещение логических томов между несколькими физическими устройствами.
- Увеличение и уменьшение размеров логических томов на лету.
- Получение "моментальных снимков" логических томов в результате выполнения операции копирования при записи.
- Замена накопителей без прекращения работы системы.
- Создание зеркал или чередования данных по дискам на логических томах.

Компоненты логического тома можно объединить в одно целое разными способами. При конкатенации физические блоки устройств объединяются вместе путем их последовательного расположения друг за другом. При чередовании данных компоненты располагаются так, чтобы соседние виртуальные блоки по возможности располагались на разных физических дисках. Это позволяет устранить узкие места, связанные с производительностью одного диска, увеличить в разы скорость обмена данными и снизить задержку доступа к данным.

Если вы уже сталкивалась с массивами RAID (см. раздел 20.8), то можете заметить, что чередование данных поразительно напоминает RAID 0. Однако реализации LVM с чередованием, как правило, более гибкие, чем RAID. Например, они могут автоматически оптимизировать чередование или разрешать чередование на устройствах разного размера, даже если чередование фактически не произойдет в 100% случаев. Граница между LVM и RAID стала размытой, и даже схемы с контролем четности, такие как RAID 5 и RAID 6, уже регулярно появляются в менеджерах томов.

Управление логическими томами в системе Linux



Менеджер логических томов в системе Linux (LVM2) на самом деле представляет собой клон программы Veritas из системы HP-UX. Команды для этих двух систем практически одинаковы и перечислены в табл. 20.3.

Таблица 20.3. Команды LVM в системе Linux

Сущность	Операция	Команда
Физический том	Создание	`pvcreate`
	Отображение	`pvdisplay`
	Модификация	`pvchange`
	Проверка	`pvck`
Группа томов	Создание	`vgcreate`
	Модификация	`vgchange`
	Расширение	`vgextend`
	Отображение	`vgdisplay`
	Проверка	`vgck`
	Ввод в строй	`vgscan`
Логический том	Создание	`lvcreate`
	Модификация	`lvchange`
	Изменение размера	`lvresize`
	Отображение	`lvdisplay`

Архитектура LVM верхнего уровня состоит в том, что отдельные диски и разделы (физические тома) собираются в пулы устройств хранения данных, называемые *группами томов*. Затем группы томов подразделяются на логические тома, которые являются блочными устройствами, в которых хранятся файловые системы.

Физический том должен иметь метку LVM, задаваемую в команде `pvcreate`. Применение такой метки является первым шагом для доступа к устройству через менеджер LVM. Помимо информации об использовании системных ресурсов, метка содержит уникальный идентификатор для идентификации устройства.

Физический том — несколько неточный термин, поскольку физические тома не обязаны иметь прямое соответствие физическим устройствам. Они могут быть дисками, но они также могут быть дисковыми разделами или RAID-массивами. Менеджеру LVM все равно.

Вы можете управлять менеджером LVM либо большой группой простых команд, перечисленных в табл. 20.3, либо с помощью команды `lvm` и ее различных подкоманд. Эти варианты по существу идентичны; отдельные команды — это просто ссылки на команду `lvm`, которые созданы только для того, чтобы своим названием говорить вам, что они делают. Хорошее введение в систему и ее инструменты можно найти на справочной странице команды `lvm`.

Процесс настройки менеджера логических томов в системе Linux состоит из нескольких этапов.

- Создание (на самом деле определение) и инициализация физических томов.
- Добавление физических томов в группу томов.
- Создание логических томов из группы томов.

Команды LVM начинаются с буквы, определяющей уровень абстракции, на котором они действуют: команды с префиксом `pv` манипулируют физическими томами, `vg` — группами томов, а `lv` — логическими томами. Команды, имеющие префикс `lvm` (например, `lvmchange`), оперируют системой в целом.

В следующем примере мы настроили жесткий диск **/dev/sdb** объемом 1 Тбайт для использования в менеджере логических томов и создали логический том. Предполагается, что диск содержит единственный раздел, поэтому мы пропустим этот этап и будем использовать диск как простое физическое устройство, хотя это и неэффективно. Разбиение диска на разделы делает его доступным для самого широкого спектра программного обеспечения и операционных систем.

Для начала пометим раздел **sdb1** как физический том LVM.

```
$ sudo pvcreate /dev/sdb1
Physical volume "/dev/sdb1" successfully created
```

Наше физическое устройство теперь готово для добавления в группу томов.

```
$ sudo vgcreate DEMO /dev/sdb1
Volume group "DEMO" successfully created
```

Несмотря на то что в этом примере используется только одно физическое устройство, мы, разумеется, могли добавить дополнительные накопители. Для проверки наших действий используем команду **vgdisplay**.

```
$ sudo vgdisplay DEMO
--- Volume group ---
VG Name               DEMO
System ID
Format                lvm2
Metadata Areas        1
Metadata Sequence No  1
VG Access             read/write
VG Status             resizable
MAX LV                0
Cur LV                0
Open LV               0
Max PV                0
Cur PV                1
Act PV                1
VG Size               1000.00 GiB
PE Size               4.00 MiB
Total PE              255999
Alloc PE / Size       0 / 0
Free PE / Size        255999 / 1000.00 GiB
VG UUID               n26rxj-X5HN-x4nv-rdnM-7AWe-OQ21-EdDwEO
```

Аббревиатура "PE" обозначает физический экстент (physical extent) — единицы выделения дисковой памяти, на которые разделяется группа томов.

На последних этапах создается логический том в группе **DEMO**, а затем — файловая система внутри этого тома. В итоге мы получим логический том размером 100 Гбайт.

```
$ sudo lvcreate -L 100G -n web1 DEMO
Logical volume "web1" created
```

Большинство интересных возможностей менеджера LVM2 относится к уровню логического тома. В частности, они включают в себя возможности создания дисков с чередованием, зеркал и объединения нескольких устройств в один большой том.

Теперь к логическому тому можно обращаться по имени **/dev/DEMO/web1**. Мы обсудим файловые системы в разделе 20.9, а пока кратко опишем создание стандартной файловой системы для демонстрации некоторых особенностей работы менеджера LVM.

```
$ sudo mkfs /dev/DEMO/web1
...
$ sudo mkdir /mnt/web1
$ sudo mount /dev/DEMO/web1 /mnt/web1
```

Мгновенные копии тома

Вы можете создать копию при записи любого логического тома LVM2, независимо от того, содержит он файловую систему или нет. Это свойство удобно для создания статического образа файловой системы, предназначенного для резервного копирования, но, в отличие от мгновенных копий систем ZFS и Btrfs, мгновенные снимки LVM2, к сожалению, не очень полезны для контроля версий.

Проблема заключается в том, что логические тома имеют фиксированный размер. Когда создается логический том, пространство на диске выделяется из группы томов. Копирование при записи изначально не потребляет дисковую память, но при модификации блоков менеджер тома должен найти место для хранения как старой, так и новой версии. При создании мгновенных копий это пространство, необходимое для модифицированных блоков, должно резервироваться, причем, как и любой том LVM, выделенная память должна иметь фиксированный размер.

При этом не имеет значения, что модифицируется — оригинальный том или его мгновенная копия (которая по умолчанию перезаписывается). В любом случае затраты на дублирование блоков перекладываются на мгновенные копии. Выделение памяти для мгновенных копий могут быть сведены на нет при активности исходного тома, даже если при этом создаются пустые мгновенные копии.

Если для мгновенной копии не выделяется максимальный объем памяти, занимаемый томом, для которого она создается, то вы можете столкнуться с нехваткой дискового пространства в мгновенной копии. Это намного хуже, чем кажется, поскольку менеджер томов не сможет обеспечить хранение согласованного образа мгновенной копии; потребуется дополнительная память *только лишь для хранения самой копии*. При нехватке выделенного пространства менеджер LVM прекращает поддерживать мгновенные копии, и они безвозвратно повреждаются.

Итак, как показывает практика, мгновенные копии должны быть либо краткосрочными, либо такими же большими, как их источники. Эти аргументы свидетельствуют в пользу "многочисленных дешевых виртуальных копий".

Для того чтобы создать том **/dev/DEMO/web1-snap** как мгновенную копию тома **/dev/DEMO/web1**, можно использовать следующую команду.

```
$ sudo lvcreate -L 100G -s -n web1-snap DEMO/web1
Logical volume "web1-snap" created.
```

Обратите внимание на то, что мгновенная копия имеет самостоятельное имя, а ее источник должен быть задан в виде ***группа_томов/том***.

Теоретически том **/mnt/web1** сначала необходимо размонтировать, чтобы гарантировать согласованность файловой системы. На практике файловая система ext4 достаточно хорошо защищена от повреждения данных, хотя существует вероятность потерять самые свежие изменения блоков данных. Это идеальный компромисс для мгновенных копий, используемых в качестве источника для резервного копирования.

Для того чтобы проверить состояние мгновенных копий, следует выполнить команду **lvdisplay**. Если она сообщает, что мгновенная копия "не активна", это значит, что для нее уже не хватило места на диске, и ее следует удалить, поскольку с такой мгновенной копией практически ничего сделать уже нельзя.

Изменение размера файловых систем

Переполнение файловых систем происходит гораздо чаще, чем сбои дисков, и одно из преимуществ логических томов заключается в том, что манипулировать ими гораздо легче, чем физическими разделами на жестком диске. Мы сталкивались с разными ситуациями, когда пользователи хранили на сервере личные видео-файлы, либо когда почтовые ящики организации были просто забиты спамом.

Менеджер логических дисков ничего не знает о содержимом своих томов, но изменять размеры необходимо на уровне как тома, так и файловой системы. Порядок зависит от конкретной операции. При уменьшении размера следует сначала работать с файловой системой, а при увеличении — сначала с томом. Это правило запоминать не стоит: просто следуйте здравому смыслу.

Предположим, что в нашем примере том **/mnt/web1** увеличился больше, чем предполагалось, и ему нужно дополнительно 100 Гбайт дисковой памяти. Сначала проверим группу томов, чтобы убедиться, что дополнительное место существует.

```
$ sudo vgdisplay DEMO
--- Volume group ---
VG Name               DEMO
System ID
Format                lvm2
Metadata Areas        1
Metadata Sequence No  4
VG Access             read/write
VG Status             resizable
Open LV               1
Max PV                0
Cur PV                1
Act PV                1
VG Size               1000.00 GiB
PE Size               4.00 MiB
Total PE              255999
Alloc PE / Size       51200 / 200.00 GiB
Free PE / Size        204799 / 800.00 GiB
VG UUID               n26rxj-X5HN-x4nv-rdnM-7AWe-OQ21-EdDwEO
```

Обратите внимание на то, что мы использовали 200 Гбайт пространства: 100 Гбайт — для исходной файловой системы и 100 Гбайт — для моментального снимка. Тем не менее нам по-прежнему доступно много места. Мы размонтируем файловую систему и используем команду **lvresize** для добавления места в логический том.

```
$ sudo umount /mnt/web1
$ sudo lvchange -an DEMO/web1
$ sudo lvresize -L +100G DEMO/web1
Size of logical volume DEMO/web1 changed from 100.00 GiB (25600
    extents) to 200.00 GiB (51200 extents).
Logical volume DEMO/web1 successfully resized.
$ sudo lvchange -ay DEMO/web1
```

Команды **lvchange** необходимы для того, чтобы деактивировать том, изменить его объем, а потом вновь активировать. Эта часть необходима только потому, что существует мгновенная копия тома `web1` из предыдущего примера. После изменения объема копия "увидит" дополнительные 100 Гбайт, но, поскольку файловая система содержит только 100 Мбайт, копию можно по-прежнему использовать.

Теперь мы можем изменить размер файловой системы с помощью команды **resize2fs**. (Цифра 2 взята из имени оригинальной файловой системы ext2, но команда поддерживает все версии этой файловой системы.) Поскольку команда **resize2fs** может определять объем новой файловой системы по тому, нам не обязательно указывать новый объем явным образом. Это необходимо только при уменьшении размера файловой системы.

```
$ sudo resize2fs /dev/DEMO/web1
resize2fs 1.43.3 (04-Sep-2016)
Resizing the filesystem on /dev/DEMO/web1 to 52428800 (4k) blocks.
The filesystem on /dev/DEMO/web1 is now 52428800 (4k) blocks long.
```

Ну вот! Проверка информации, которая выводится командой **df**, снова показывает изменения.

```
$ sudo mount /dev/DEMO/web1 /mnt/web1
$ df -h /mnt/web1
Filesystem              Size   Used  Avail  Use%   Mounted on
/dev/mapper/DEMO-web1   197G   60M   187G     1%   /mnt/web1
```

Аналогичным образом работают команды для изменения размера в других файловых системах. Для файловых систем XFS (по умолчанию для систем Red Hat и CentOS) используйте команду **xfs_growfs**; для файловых систем UFS (по умолчанию в системе FreeBSD) используйте команду **growfs**. Файловые системы XFS должны быть смонтированы так, чтобы они допускали расширение. Как показывают названия этих команд, файловые системы XFS и UFS могут быть расширены, но не уменьшены. Если вам нужно освободить место, скопируйте содержимое файловой системы в новую, меньшую по размеру файловую систему.

Стоит отметить, что "диски", которые вы выделяете и прикрепляете к виртуальным машинам в облаке, являются по существу логическими томами, хотя сам диспетчер томов находится в другом месте облака. Эти объемы обычно изменяются с помощью консоли управления облачного провайдера или утилиты командной строки.

Процедура изменения размеров облачных файловых систем такая же, как и описанная выше, но имейте в виду, что, поскольку эти виртуальные устройства выдают себя за диски, у них, вероятно, есть таблицы разделов. Вам нужно будет изменить размер на трех отдельных слоях: на уровне облачного провайдера, уровне раздела и уровне файловой системы.

Управление логическими томами в FreeBSD



У системы FreeBSD есть полноценный менеджер логических томов. Предыдущие версии были известны под именем Vinum, но теперь, когда система была переписана, чтобы соответствовать обобщенной архитектуре `geom` FreeBSD для устройств хранения, имя было изменено на GVinum. Как и LVM2, GVinum реализует множество типов RAID.

Разработчики систем FreeBSD в последнее время приложили много усилий для поддержки файловой системы ZFS, и хотя GVinum официально не устарел, общедоступные комментарии разработчиков показывают, что ZFS является рекомендуемым подходом для управления логическим томом и RAID в будущем. Соответственно, мы не обсуждаем здесь GVinum. Описание файловой системы ZFS приводится в разделе 20.12.

20.8. RAID: избыточные массивы недорогих дисков

Даже при наличии резервных копий последствия сбоя диска на сервере могут быть катастрофическими. RAID (redundant arrays of inexpensive disks — избыточные массивы недорогих дисков) — это система, распределяющая или дублирующая данные по набору нескольких дисков.[10] Система RAID не только помогает избежать потери данных, но и минимизирует время простоя, связанного с отказом аппаратуры (часто сводя его к нулю), и потенциально повышает скорость работы.

Систему RAID можно реализовать с помощью специального аппаратного контроллера, благодаря которому операционная система будет видеть группу жестких дисков как один составной накопитель. Кроме того, ее можно реализовать так, чтобы операционная система просто считывала или записывала данные на жестких дисках по правилам системы RAID.

Программная и аппаратная реализации системы RAID

Поскольку диски сами по себе являются узким местом в реализации системы RAID, нет причин предполагать, что аппаратная реализация системы RAID всегда будет работать быстрее, чем программная на уровне операционной системы. В прошлом аппаратная система RAID была доминирующей по двум причинам: из-за недостатка программной поддержки (отсутствие прямой поддержки со стороны операционных систем) и способности аппаратного обеспечения буферизовать записываемые данные в энергонезависимой памяти.

Последнее свойство, упомянутое выше, повышает скорость работы дисковой подсистемы, поскольку операции записи выполняется очень быстро. Кроме того, оно защищает дисковую подсистему от потенциального повреждения, которое получило название "Дырка при записи в RAID 5". Этот эффект описывается ниже. Однако будьте внимательны: многие широко распространенные недорогие RAID-контроллеры, используемые в персональных компьютерах, вообще не имеют энергонезависимой памяти. На самом деле они представляют собой старый добрый интерфейс SATA с элементами встроенного программного обеспечения RAID. Реализации системы RAID на материнских платах персональных компьютеров также относятся к этой категории. В таких случаях лучше вообще отказаться от использования подобного RAID в системах Linux или FreeBSD. А лучше всего используйте файловые системы ZFS или Btrfs.

Однажды на важном промышленном сервере произошел отказ контроллера диска. Несмотря на то что данные были распределены по нескольким физическим накопителям, неисправный контроллер RAID уничтожил данные на всех дисках. В результате пришлось долго и нудно восстанавливать данные на сервере. Поэтому теперь для управления средой RAID на восстановленном сервере используется встроенное в ядро программное обеспечение, что позволило исключить возможность нового сбоя контроллера RAID.

Уровни системы RAID

Система RAID выполняет две важные вещи. Во-первых, она может повысить скорость работы дисковой подсистемы, "разбросав" данные по многим накопителям и позволив нескольким дискам одновременно передавать или получать поток данных. Во-вторых, она может реплицировать данные по многим физическим устройствам, уменьшая риск потери данных, связанный со сбоем одного диска.

[10]Аббревиатуру RAID иногда расшифровывают как "redundant arrays of independent disks" ("избыточные массивы независимых дисков"). Оба варианта с исторической точки зрения являются правильными.

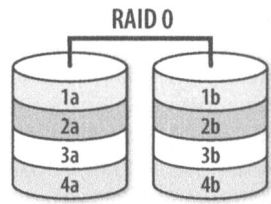
Репликация может осуществляться в двух видах: зеркальное отражение (mirroring), при котором блоки данных побитово репродуцируются на нескольких накопителях, и использование схем четности (parity schemes), когда один или несколько накопителей содержат контрольную сумму блоков данных, расположенных на других дисках и позволяющую восстановить эти данные в случае отказа одного из дисков. Зеркальное отражение быстрее, но занимает много места на диске. Схемы четности более экономно используют дисковую память, но работают медленнее.

Система RAID традиционно описывается в терминах "уровней", определяющих конкретные детали параллелизма и избыточности. Возможно, этот термин является неудачным, потому что более высокий уровень не обязательно оказывается лучшим. Уровни — это просто разные конфигурации; вы можете использовать любой из них.

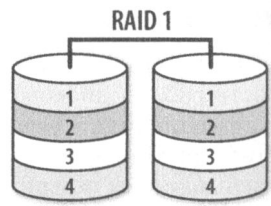
На рисунках, приведенных ниже, числа обозначают логические сегменты, а буквы `a`, `b` и `c` — блоки данных в этих сегментах. Блоки, помеченные буквами `p` и `q`, являются блоками четности.

- "Линейный режим", известный также как JBOD (just a bunch of disks — просто группа дисков), нельзя даже назвать реальным уровнем системы RAID. Его реализует каждый контроллер RAID. В режиме JBOD сектора нескольких накопителей объединяются в один большой виртуальный накопитель. Это не обеспечивает ни избыточности данных, ни повышения производительности. В настоящее время функциональная возможность JBOD лучше обеспечивается с помощью менеджера логических томов, а не контроллера RAID.
- Уровень RAID 0 используется только для повышения производительности. Он объединяет два или более накопителей одинакового размера, но вместо их объединения последовательно один за другим он распределяет данные между дисками, входящими в пул. Операции последовательного чтения и записи в этом случае оказываются распределенными между несколькими физическими дисками, что снижает время записи и доступа к диску.

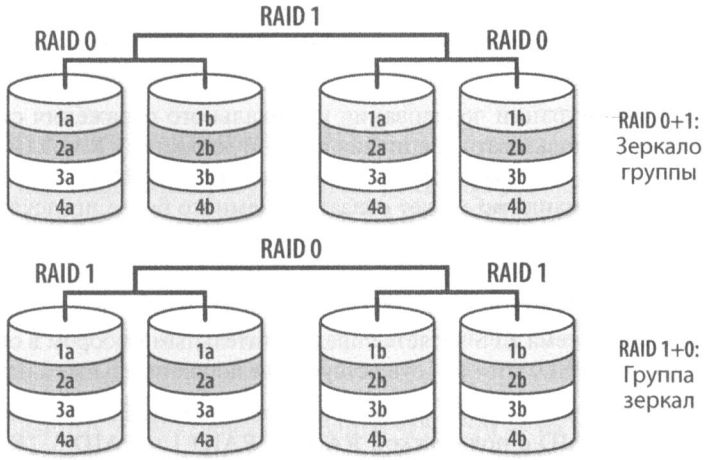
  Обратите внимание на то, что надежность уровня 0 значительно *ниже* надежности отдельных дисков. Вероятность отказа массива из двух дисков в течение года примерно в два раза выше, чем у отдельного диска, и т.д.



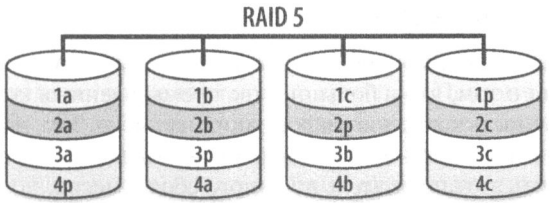
- Уровень RAID 1 известен как "зеркальное отражение". Записи дублируются одновременно на нескольких дисках. Эта схема замедляет процесс записи по сравнению с записью данных на отдельный диск. Однако она обеспечивает скорость считывания, сравнимую с уровнем RAID 0, потому что чтение можно распределять между несколькими дублирующими накопителями.



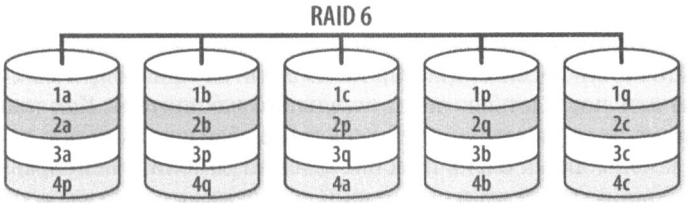
- Уровни RAID 1+0 и 0+1 представляют собой группы зеркал или зеркала групп. С логической точки зрения они представляют собой сочетание уровней RAID 0 и RAID 1, но многие контроллеры и программы обеспечивают их непосредственную поддержку. Цель обоих режимов — одновременно достичь производительности уровня RAID 0 и избыточности уровня RAID 1.



- Уровень RAID 5 распределяет и данные, и информацию о четности, добавляя избыточность и одновременно повышая производительность чтения. Кроме того, уровень RAID 5 более эффективно использует дисковое пространство, чем уровень RAID 1. Если массив состоит из N накопителей (требуется не менее трех), то N–1 из них могут хранить данные. Следовательно, эффективность использования дисковой памяти на уровне RAID 5 превышает 67%, в то время как зеркальное отражение не может превысить 50%.



- Уровень RAID 6 аналогичен уровню RAID 5 с двумя дисками четности. Массив RAID 6 может сохранить работоспособность при полном отказе двух накопителей без потери данных.



Уровни RAID 2, 3 и 4 хотя и определены, но используются редко. Менеджеры логических томов обычно могут и чередовать данные (RAID 0), и осуществлять зеркальное отражение (RAID 1).

Системы ZFS и Btrfs, представляющие собой системы RAID, менеджеров логических дисков и файловые системы вместе взятые, обеспечивают группирование, зеркалирование и настройку аналогично уровням RAID 5 и RAID 6. Подробнее об этих возможностях см. в разделе 20.11.



Система Linux поддерживает файловые системы ZFS и Btrfs, хотя систему ZFS нужно инсталлировать отдельно. Дополнительную информацию см. в разделе 20.11.

Для обычных конфигураций чередования и зеркального отражения система Linux предоставляет выбор: использовать специализированную систему RAID (например, команду **md**; см. ниже) или менеджер логического диска. Подход LVM, вероятно, является более гибким, хотя команда **md** может оказаться немного более предсказуемой. Если у вас есть возможность использовать команду **md**, вы можете продолжать использовать механизм LVM для управления пространством на томе RAID. Для программной реализации RAID уровней 5 и 6 вы должны использовать команду **md**.



Файловая система ZFS является предпочтительным выбором в операционной системе FreeBSD, хотя там существуют две дополнительные альтернативы.

На уровне драйвера диска система **geom** в файловой системе FreeBSD может объединять диски в массивы RAID с поддержкой RAID 0, RAID 1 и RAID 3. (RAID 3 похож на RAID 5, но использует выделенный диск для хранения блоков четности вместо распределения блоков четности среди всех дисков.) Можно даже создавать стеки систем **geom**, так что возможны комбинации RAID 1 + 0 и RAID 0 + 1.

Система FreeBSD также включает поддержку RAID 0, RAID 1 и RAID 5 в своем логическом диспетчере томов GVinum. Однако с появлением полной встроенной поддержки ZFS во FreeBSD будущее GVinum, похоже, под вопросом. Он еще официально не устарел, но больше не поддерживается.

Восстановление диска после сбоя

Исследование причин отказов дисков, проведенное компанией Google и описанное выше, свидетельствует о том, что в большинстве промышленных компаний необходимо обеспечивать избыточность хранения информации на диске. Так, при ежегодном уровне отказов, равном 8%, вашей организации потребуется около 150 жестких дисков только для того, чтобы снизить этот показатель до одного сбоя в месяц.

Режимы JBOD и RAID 0 в плане надежности ничем помочь не могут, когда возникает проблема с отказом диска, — здесь просто нужно почаще выполнять резервное копирование данных. Другие конфигурации RAID при аппаратном сбое отмечают вышедшие из строя диски как неисправные. С точки зрения клиентов массивы RAID продолжают нормально функционировать, хотя и с более низкой скоростью.

Неисправные диски необходимо как можно быстрее заменить новыми, чтобы восстановить избыточность массива. Массив RAID 5 или двухдисковый массив RAID 1 может только смягчить последствия отказа отдельного накопителя. Как только один сбой произошел, массив подвергается опасности второго сбоя с полной потерей данных.

Процедура замены диска очень простая. Следует заменить неисправный диск другим диском, имеющим такой же или больший объем памяти, а затем сообщить системе RAID о том, что старый диск был заменен новым. Последует продолжительный период, в течение которого информация о четности будет переписана на новый, чистый, диск. Часто эту операцию выполняют в ночное время. В течение этого периода массив остается доступным для клиентов, но производительность, скорее всего, будет очень низкой.

Для того чтобы ограничить время простоя и уменьшить уязвимость массива при втором сбое, большинство систем RAID позволяет назначать один или несколько дисков на роль "горячего резерва". Когда возникает сбой, неисправный диск автоматически

заменяется резервным и немедленно начинается процесс повторной синхронизации массива. Если в массиве поддерживаются диски "горячего резерва", то их, разумеется, обязательно следует использовать.

Недостатки конфигурации RAID 5

RAID 5 — популярная конфигурация, но у нее есть ряд недостатков. Все, что будет сказано ниже, относится и к конфигурации RAID 6, но для простоты мы ограничим обсуждение системой RAID 5.

Во-первых, что следует особо подчеркнуть, конфигурация RAID 5 не отменяет регулярное автономное резервное копирование данных. Она защищает систему лишь от сбоя диска, и все. Она не защищает систему ни от случайного удаления файлов, ни от сбоев контроллера, пожара, хакерской атаки и других опасностей.

Во-вторых, конфигурация RAID 5 не отличается высокой производительностью. Она хранит блоки данных на N–1 дисках, а блоки четности — на N-м диске.[11] После записи произвольного блока должны быть обновлены по крайней мере один блок данных и один блок четности для указанного логического сегмента . Более того, система RAID не знает, что именно должен содержать новый блок четности, пока не прочитает старый блок четности и старые данные. Следовательно, каждая запись в случайно выбранное место дисковой подсистемы состоит из четырех операций: двух операций считывания и двух операций записи. (Если реализация обладает развитой логикой, то последовательные записи могут оказаться намного эффективнее.)

И наконец, конфигурация RAID 5 уязвима к повреждениям при определенных обстоятельствах. Ее инкрементальное обновление данных о четности является более эффективным, чем чтение всего логического сегмента и повторное вычисление для него информации о четности на основе исходных данных. С другой стороны, это значит, что данные о четности больше нигде не вычисляются и не хранятся. Поэтому если будет нарушена синхронизация какого-нибудь блока в логическом сегменте, содержащего блок четности, этот факт никак не проявится при нормальной работе для конечного пользователя; операции считывания блоков данных по-прежнему будут возвращать правильные данные.

Эта проблема выяснится только при сбое диска. Блок четности, вероятно, будет перезаписан много раз с момента возникновения исходной рассинхронизации. Следовательно, реконструированный блок данных на запасном диске будет состоять, по существу, из случайных данных.

Эта разновидность рассинхронизации между блоками данных и четности не единственная проблема. Дисковые накопители не являются транзакционными устройствами. Без дополнительного уровня защиты сложно гарантировать, что на разных дисках будут правильно обновлены либо два блока, либо ни одного. Для нарушения синхронизации между блоком данных и блоком четности достаточно, чтобы произошел сбой диска, отключение электропитания или разрыв связи в неподходящий момент.

Эта проблема известна под названием "дырка при записи на RAID 5" (RAID 5 "write hole"). Она является объектом пристального изучения в течение последних десяти лет. Авторы файловой системы ZFS утверждают, что благодаря переменной длине логических сегментов в системе ZFS она защищена от проблемы "дырки при записи на RAID 5".

[11]Данные о четности распределяются между всеми накопителями массива; каждый логический сегмент имеет свою информацию о четности, которая хранится на отдельном диске. Поскольку в массиве нет специального диска для хранения информации о четности, маловероятно, что любой отдельный диск может затормозить работу системы.

Кстати, это одна из причин, по которой авторы назвали свою реализацию системы RAID в ZFS именем RAID-Z, а не RAID 5, хотя на практике эти концепции похожи.

Еще одним потенциальным решением является "чистка" ("scrubbing") диска, которая состоит из поочередной проверки блоков четности во время простоя дискового массива. Многие реализации системы RAID имеют ту или иную функцию чистки. От вас только требуется не забывать ее регулярно запускать, например через утилиту **cron** или таймер **systemd**.

Команда mdadm: программное обеспечение RAID в системе Linux



Стандартная реализация программного обеспечения RAID для системы Linux называется **md**, драйвер "нескольких дисков" ("multiple disks" driver). Его внешний интерфейс обеспечивает команда **mdadm**. Драйвер **md** поддерживает все конфигурации RAID, перечисленные выше, а также RAID 4. Предыдущая система под названием **raidtools** больше не используется.

Вы также можете реализовать RAID в системе Linux с помощью диспетчера логических томов (LVM2), или Btrfs, или другой файловой системы со встроенными функциями управления томами и поддержки RAID. Мы рассматриваем менеджер LVM2 в разделе 20.7, а файловые системы следующего поколения — в разделе 20.11. Как правило, эти многочисленные реализации представляют разные эпохи разработки программного обеспечения, причем **mdadm** относится к первому, а ZFS/Btrfs — к последнему поколению.

Все эти системы активно поддерживаются, поэтому выбирайте в зависимости от того, что вы предпочитаете. Сайтам без установленной базы лучше всего перейти непосредственно на систему "все-в-одном", например Btrfs.

Создание массива

В следующем сценарии выполняется конфигурация массива RAID 5, состоящего из трех идентичных жестких дисков объемом 1 Тбайт. Несмотря на то что драйвер **md** может использовать в качестве компонентов неформатированные диски, мы предпочитаем снабдить каждый диск таблицей разделов, поэтому начинаем с запуска утилиты **gparted**, создающей таблицу разделов GPT на каждом диске, и выделения всего дискового пространства одному разделу типа "Linux RAID". Совершенно не обязательно задавать тип раздела, но это будет полезной информацией для тех, кто станет просматривать таблицу разделов позднее.

Следующая команда создает массив RAID 5 из наших трех разделов диска.

```
$ sudo mdadm --create /dev/md/extra --level=5 --raid-devices=3
    /dev/sdf1 /dev/sdg1 /dev/sdh1

mdadm: Defaulting to version 1.2 metadata
mdadm: array /dev/md/extra started.
```

Виртуальный файл **/proc/mdstat** всегда содержит краткое описание состояния драйвера **md**, а также состояния всех массивов RAID, существующих в системе. Особенно полезно проанализировать содержимое файла **/proc/mdstat** после добавления нового диска или замены неисправного. Для этой цели рекомендуем использовать команду **cat /proc/mdstat**.)

```
$ cat /proc/mdstat
Personalities : [linear] [multipath] [raid0] [raid1] [raid6] [raid5]
```

```
   [raid4] [raid10]
md127 : active raid5 sdh1[3] sdg1[1] sdf1[0]
   2096886784 blocks super 1.2 level 5, 512k chunk, algo 2 [3/2] [UU_]
   [>....................] recovery = 0.0% (945840/1048443392)
      finish=535.2min speed=32615K/sec
   bitmap: 0/8 pages [0KB], 65536KB chunk
unused devices: <none>
```

Драйвер **md** не следит за тем, какой блок в массиве был использован, поэтому он должен вручную синхронизировать все блоки четности с их соответствующими блоками данных. Драйвер **md** запускает операцию "восстановления" ("recovery") данных, потому что, по существу, она совпадает с той процедурой, которая выполняется после замены неисправного диска. Для больших массивов она может выполняться довольно длительное время (несколько часов, или даже дней).

В файлах **/var/log/messages** или **/var/log/syslog** также записаны полезные сообщения.

```
kernel: md: bind<sdf1>
kernel: md: bind<sdg1>
kernel: md: bind<sdh1>
kernel: md/raid:md127: device sdg1 operational as raid disk 1
kernel: md/raid:md127: device sdf1 operational as raid disk 0
kernel: md/raid:md127: allocated 3316kB
kernel: md/raid:md127: raid level 5 active with 2 out of 3 devices,
     algorithm 2
kernel: RAID conf printout:
kernel: --- level:5 rd:3 wd:2
kernel: disk 0, o:1, dev:sdf1
kernel: disk 1, o:1, dev:sdg1
kernel: created bitmap (8 pages) for device md127
mdadm[1174]: NewArray event detected on md device /dev/md127
mdadm[1174]: DegradedArray event detected on md device /dev/md127
kernel: md127: bitmap initialized from disk: read 1 pages, set 15998 of
     15998 bits
kernel: md127: detected capacity change from 0 to 2147212066816
kernel: RAID conf printout:
kernel: --- level:5 rd:3 wd:2
kernel: disk 0, o:1, dev:sdf1
kernel: disk 1, o:1, dev:sdg1
kernel: disk 2, o:1, dev:sdh1
kernel: md: recovery of RAID array md127
kernel: md: minimum _guaranteed_ speed: 1000 KB/sec/disk.
kernel: md: using maximum available idle IO bandwidth (but not more than
     200000 KB/sec) for recovery.
kernel: md: using 128k window, over a total of 1048443392k.
mdadm[1174]: RebuildStarted event detected on md device /dev/md127
```

Команда первоначального создания также предназначена для "активизации" массива (т.е. для приведения его в состояние, пригодное для использования). При последующих перезагрузках в большинстве дистрибутивов ОС, включая те, которые рассматриваются в книге, поиск существующих массивов и их повторная активизация выполняются автоматически.

Обратите внимание на то, что при выполнении команды **mdadm -create** указывается путь к устройству для составного массива. Пути **md**-устройств старого стиля выглядели как **/dev/md0**, но когда вы указываете путь в каталоге **/dev/md**, как это было сделано в данном примере, **mdadm** записывает выбранное имя в суперблок массива. Эта

мера гарантирует, что вы всегда можете найти массив по логическому пути, даже если после перезагрузки массив активизирован автоматически и ему может быть присвоен другой номер. Как видно из записей журнала, приведенных выше, массив также имеет традиционное имя (здесь, **/dev/md127**), а **/dev/md/extra** — это просто символическая ссылка на реальное устройство массива.

mdadm.conf: документирование конфигурации массива

Формально, команда **mdadm** не требует наличия файла конфигурации, но если он есть, она его использует (как правило, это файл **/etc/mdadm/mdadm.conf** или **/etc/mdadm.conf**). Мы настоятельно рекомендуем использовать файл конфигурации и добавить в него элементы ARRAY при создании нового массива. В результате вы создадите в стандартном месте документацию по конфигурации системы RAID, позволяя администратору просмотреть информацию при возникновении проблемы. В качестве альтернативы использованию файла конфигурации можно задавать конфигурацию в командной строке при каждой активизации массива.

Команда **mdadm --detail --scan** записывает текущие настройки системы RAID в файл конфигурации **mdadm.conf**.

```
$ sudo mdadm --detail --scan
ARRAY /dev/md/extra metadata=1.2 name=ubuntu:extra UUID=b72de2fb:60b30
    3af:3c176048:dc5b6c8b
```

Теперь команда **mdadm** сможет прочитать файл **mdadm.conf** при загрузке или завершении работы системы, что намного упростит управление массивом. Например, для того чтобы отключить массив, созданный выше, достаточно запустить следующую команду.

```
$ sudo mdadm -S /dev/md/extra
```

Для того чтобы запустить массив снова, можно выполнить такую команду.

```
$ sudo mdadm -As /dev/md/extra
```

Первая из приведенных выше команд сработает, даже если файла **mdadm.conf** не существует, а вторая — нет.

В предыдущих изданиях этой книги мы рекомендовали добавлять записи DEVICE для компонентов каждого массива в файл **mdadm.conf**. Теперь мы берем свои слова обратно. Названия устройств в наши дни стали слишком непостоянными, и команда **mdadm** лучше находит и идентифицирует компоненты массива, чем раньше. Теперь мы не считаем, что записи DEVICE — лучшая практика.

Команда **mdadm** имеет режим **--monitor**, в котором она выполняется постоянно как демон и сообщает по электронной почте о проблемах, возникших в массиве RAID. Используйте эту возможность! Для того чтобы ее настроить, добавьте строку MAILADDR или PROGRAM в свой файл **mdadm.conf**. Конфигурация MAILADDR указывает адресата, которому следует отправлять предупреждения по электронной почте, а конфигурация PROGRAM запускает внешнюю программу для формирования отчетов, которую вы установили (это полезно для интеграции с системой мониторинга; см. главу 28).

Кроме того, во время загрузки необходимо запустить демон мониторинга. Во всех рассматриваемых нами дистрибутивах ОС это делается с помощью сценария **init**, но имена и процедуры могут слегка отличаться.

```
debian$ sudo update-rc.d mdadm enable
ubuntu$ sudo update-rc.d mdadm enable
```

```
redhat$ sudo systemctl enable mdmonitor
centos$ sudo systemctl enable mdmonitor
```

Имитация сбоя

Что произойдет, если диск действительно выйдет из строя? Попробуем разобраться! Команда **mdadm** предоставляет нам прекрасную возможность сымитировать сбойный диск.

```
$ sudo mdadm /dev/md/extra -f /dev/sdg1
mdadm: set /dev/sdg1 faulty in /dev/md/extra

$ sudo tail -1 /var/log/messages
Apr 10 16:18:39 ubuntu kernel: md/raid:md127: Disk failure on sdg1,
    disabling device.#012md/raid:md127: Operation continuing on 2
    devices.

$ cat /proc/mdstat
Personalities : [linear] [multipath] [raid0] [raid1] [raid6] [raid5]
    [raid4] [raid10]
md127 : active raid5 sdh1[3] sdf1[0] sdg1[1](F)
    2096886784 blocks super 1.2 level 5, 512k chunk, algo 2 [3/2] [UU_]

unused devices: <none>
```

Поскольку RAID 5 представляет собой устойчивую к отказу одного диска конфигурацию, массив продолжает функционировать в режиме деградации, так что пользователи могут не заметить возникшей проблемы вовсе.

Для того чтобы удалить накопители из конфигурации RAID, выполните команду **mdadm -r**.

```
$ sudo mdadm /dev/md/extra -r /dev/sdg1
mdadm: hot removed /dev/sdg1 from /dev/md/extra
```

Как только диск будет логически удален, вы можете остановить систему и заменить накопитель. Аппаратное обеспечение, допускающее "горячую замену", позволяет производить замену дисков без остановки системы и ее перезагрузки.

Если ваш RAID-массив построен на основе физических дисков, вы должны заменять их только идентичными дисками. Если вместо дисков используются разделы, то их можно заменять любыми разделами подходящего размера. Последнее утверждение является достаточно веской причиной для того, чтобы создавать массивы на основе разделов, а не физических дисков. С точки зрения производительности лучше, чтобы RAID-массивы создавались из дисков одного производителя. (Если ваша конфигурация RAID построена на основе разделов, вы должны запустить утилиту разбиения диска и правильно создать разделы, прежде чем добавлять диск в массив.)

В нашем примере сбой всего лишь смоделирован, поэтому мы можем вернуть накопитель обратно в массив, не заменяя никакого физического диска.

```
$ sudo mdadm /dev/md/extra -a /dev/sdg1
mdadm: hot added /dev/sdg1
```

Драйвер **md** немедленно начнет перестраивать массив. Как обычно, за выполнением этой процедуры вы можете следить с помощью файла **/proc/mdstat**. Перестройка может идти часами, так что учтите этот факт, составляя план по восстановлению (и тестированию!) системы после сбоя.

20.9. Файловые системы

Даже после разбиения жесткого диска на разделы или логические тома на нем еще нельзя хранить файлы. Для этого необходимо реализовать все абстракции и функции, описанные в главе 5, в терминах блоков. Они реализуются кодом, который называется *файловой системой* и требует дополнительных затрат ресурсов и данных.

В ранних версиях операционных систем реализация файловой системы была встроена в ядро, но вскоре стало ясно, что поддержка многочисленных типов файловых систем является важной целью. Разработчики систем UNIX создали хорошо определенный интерфейс ядра, позволявший одновременно активировать несколько типов файловых систем. Кроме того, базовое аппаратное обеспечение было абстрагировано с помощью интерфейса файловой системы, так что файловые системы видели примерно такой же интерфейс для накопителей, что и другие программы системы UNIX, имеющие доступ к дискам через файлы устройств в каталоге `/dev`.

Изначально поддержка нескольких типов файловых систем объяснялась необходимостью поддерживать систему NFS и файловые системы для сменных носителей. Однако, как только шлюз был открыт, началась эра поисков; многие группы стали работать над улучшением файловых систем. Некоторые из этих файловых систем были предназначены для конкретных операционных систем, а другие (например, ReiserFS) не были связаны ни с какими определенными операционными системами.

В большинстве операционных систем по умолчанию установлены одна или две файловые системы. Эти файловые системы тщательно тестируются вместе с остальной частью системы до выпуска стабильных версий.

Как правило, системы официально поддерживают одну файловую систему традиционного стиля (UFS, ext4 или XFS) и одну файловую систему следующего поколения, которая включает в себя функции управления томами и RAID (ZFS или Btrfs). Поддержка последних опций обычно лучше всего реализуется на физическом оборудовании; в облачных системах они могут использоваться для разделов данных, но для загрузочного диска только изредка.

Несмотря на то что реализации других файловых систем как правило устанавливаются в виде обычных пакетов, применение дополнительных файловых систем часто приносит риск и потенциальную нестабильность в работе для всей ОС. Файловые системы являются основополагающими, поэтому они должны быть на 100% стабильны и надежны при всех сценариях использования. Разработчики файловых систем упорно трудятся, чтобы достичь такого уровня надежности, но риск не может быть полностью исключен.[12]

Если только вы не создаете огромный пул для хранения данных или диск для работы специфического приложения, мы рекомендуем использовать только те файловые системы, которые поддерживаются в вашей ОС. Именно это, скорее всего, и предполагается в документации и средствах администрирования вашей ОС.

В следующих разделах более подробно описаны наиболее распространенные файловые системы и их управление. Сначала мы опишем традиционные файловые системы UFS, ext4 и XFS, а затем перейдем к системам следующего поколения ZFS (раздел 20.12) и Btrfs (раздел 20.13).

---

[12]Недавно компания Apple перевела iOS-устройства (которых насчитывают более миллиарда) на совершенно новую написанную с нуля файловую систему под названием APFS. То, что этот переход был выполнен невидимо и без заметных катастроф, было поистине историческим достижением инженерного искусства.

20.10. Традиционные файловые системы: UFS, ext4 и XFS

Файловые системы UFS, ext4 и XFS имеют разный код и историю, но со временем они стали очень похожими друг на друга с точки зрения администрирования. Эти файловые системы иллюстрируют старый подход, в котором управление томами и RAID реализовано отдельно от самой файловой системы. Эти системы предназначены исключительно для реализации старых добрых файловых хранилищ на блочных устройствах заранее определенного размера. Их возможности более или менее ограничены теми, которые описаны в главе 5.

Старые файловые системы в этой категории имеют скрытый дефект: если в середине операции записи было прервано питание, то дисковые блоки могут содержать некорректные структуры данных. Для решения этой проблемы и автоматического исправления других наиболее распространенных ошибок во время загрузки для проверки файловых систем использовалась команда **fsck**.

Современные файловые системы включают функцию, называемую *журналированием*, которая предотвращает возможность такого рода повреждений. Когда происходит операция файловой системы, необходимые изменения сначала записываются в журнал. После завершения обновления журнала записывается специальная "фиксирующая запись" (commit record), помечающая конец элемента данных. И только после этого будет изменена нормальная файловая система. Если во время обновления произошел сбой, файловая система может позже воспроизвести журнал для восстановления идеально согласованной файловой системы.[13]

Журналирование уменьшает время, необходимое для выполнения проверки целостности файловой системы (см. раздел, посвященный команде **fsck**) примерно до одной секунды для каждой файловой системы. Если произошел аппаратный сбой, состояние файловой системы может быть практически мгновенно оценено и восстановлено.

Система Berkeley Fast File System, реализованная Мак-Кьюзиком и другими в 1980-х годах, была первым стандартом, распространяющимся на многие системы UNIX. С некоторыми небольшими корректировками она в конечном итоге стала известна как файловая система UNIX (UFS) и легла в основу нескольких других реализаций файловой системы, включая серию файловых системе ext для Linux. UFS остается стандартной файловой системой, используемой в системе FreeBSD.

"Вторая расширенная файловая система" ("second extended filesystem"), получившая название ext2, долгое время была основным стандартом в системе Linux. Она была разработана и реализована Реми Кардом (Rémy Card), Теодором Тс'о (Theodore Ts'o) и Стивеном Твиди (Stephen Tweedie). Помимо того, что код файловой системы ext2 был написан специально для системы Linux, с функциональной точки зрения он похож на файловую систему Berkeley Fast File System.

Файловая система ext3 расширяла возможности журналирования, а система ext4 представляет собой определенное улучшение своих предшественников. Она расширяет пределы максимально допустимой памяти, увеличивает производительность некоторых операций и позволяет использовать для выделения памяти не отдельные дисковые бло-

[13] В большинстве случаев журналируются только изменения метаданных. Фактические данные, которые нужно сохранить, записываются непосредственно в файловую систему. В некоторых файловых системах также может использоваться журнал для фиксации данных, но это требует значительных эксплуатационных расходов.

ки, а целые экстенты (логические диапазоны дисковых блоков). Файловая система ext4 де-факто стала стандартной для операционных систем Debian и Ubuntu.

Файловая система XFS была разработана компанией Silicon Graphics, Inc., позже известной как SGI. Это была стандартная файловая система для IRIX, версии UNIX от SGI. Она была одной из первых файловых систем, использовавших экстенты. Это сделало ее особенно подходящей для приложений, обрабатывающих большие медиафайлы, как это делали многие клиенты SGI. XFS является стандартной файловой системой для операционных систем Red Hat и CentOS.

Терминология файловых систем

Благодаря своему родству многие файловые системы используют общую описательную терминологию. Реализации базовых объектов часто изменяются, но термины по-прежнему широко употребляются системными администраторами как обозначения фундаментальных понятий. Например, индексные дескрипторы ("inodes") — это записи фиксированной длины в таблице, каждый элемент которой хранит информацию об отдельном файле. Ранее они заносились в эту таблицу в момент создания файловой системы, но в настоящее время некоторые файловые системы создают их динамически при необходимости. В любом случае индексный дескриптор обычно имеет идентификационный номер, который можно увидеть с помощью команды `ls -i`.

Индексные дескрипторы это те объекты, на которые указывают элементы каталога. При создании жесткой ссылки на существующий файл создается новый элемент каталога, но не новый индексный дескриптор.

Суперблок (superblock) — это запись, описывающая характеристики файловой системы. Она содержит информацию о длине дискового блока, размере и местоположении таблиц индексных дескрипторов, карту блоков и информацию об их использовании, размер групп блоков и некоторые другие важные параметры файловой системы. Поскольку повреждение суперблока может привести к потере очень важной информации, несколько его копий хранятся в разных местах.

Для повышения производительности работы в ядре кешируются дисковые блоки. Кешировать можно все дисковые блоки, включая суперблоки, блоки индексных дескрипторов и информацию о каталоге. Кеши обычно не работают в режиме "сквозной записи" ("write-through"), поэтому может возникнуть некоторая задержка между моментом, когда приложение "считает", что блок записан, и моментом, в который блок действительно записывается на диск. Приложения могут затребовать от файла более предсказуемого поведения, но в этом случае резко снижается их скорость работы.

Системный вызов **sync** направляет модифицированные блоки в место их постоянного хранения на диске, стараясь моментально обеспечить полную согласованность дисковой файловой системы. Это периодическое сохранение минимизирует потерю данных, которая может произойти при сбое из-за многочисленных несохраненных блоков. Файловые системы могут самостоятельно выполнять синхронизацию по собственному расписанию или предоставить это операционной системе. Современные файловые системы имеют механизм журналирования, минимизирующий или исключающий возможность повреждения их структуры при сбое, поэтому частота синхронизации должна зависеть от того, сколько блоков могут оказаться потерянными при сбое.

В файловой системе карта дисковых блоков — это таблица содержащихся в ней свободных блоков. При записи новых файлов система анализирует эту карту, чтобы найти эффективную схему хранения. В файловой системе также хранится сводка по использованию дисковых блоков, где содержится важная информация об уже распределенных блоках.

Полиморфизм файловых систем

Файловая система — это программное обеспечение, состоящее из многочисленных компонентов. Одна ее часть находится в ядре (или же в пользовательском пространстве системы Linux; наберите в поисковой машине Google аббревиатуру FUSE) и реализует основные операции, связанные с трансляцией вызовов стандартного интерфейса API файловой системы в операции чтения и записи блоков диска. Другие части состоят из пользовательских команд для форматирования новых томов, проверки целостности файловой системы и выполнения других специальных операций, связанных с форматированием.

Ранее стандартные пользовательские команды "знали все" о файловой системе, которую использовала операционная система, и просто реализовали требуемые функциональные возможности. Так, команда **mkfs** создавала новые файловые системы, команда **fsck** устраняла проблемы, а команда **mount**, в основном, просто вызывала системные функции.

В настоящее время существует множество файловых систем, поэтому операционные системы должны решить, как использовать их возможности. Долгое время операционная система Linux стремилась подогнать все файловые системы под один стандарт и скрыть их за командами-оболочками **mkfs** и **fsck**. Эти команды вызывали отдельные команды, например **mkfs.***имя_файловой_системы* и **fsck.***имя_файловой_системы* в зависимости от конкретной файловой системы. В настоящее время желание унифицировать все файловые системы ушло в прошлое, и большинство операционных систем предлагают пользователям вызывать команды файловых систем непосредственно.

Форматирование файловых систем



Для создания новой файловой системы следует выполнить следующую команду.

mkfs.*имя_файловой_системы* **[-L** *метка* **]** [*другие_параметры*] *устройство*

В системе FreeBSD процесс создания файловой системы UFS аналогичен предыдущему, но вместо команды **mfks** выполняется команда **newfs**.

newfs [-L *метка* **]** [*другие_параметры*] *устройство*

Параметр **-L** в обоих вариантах задает метку тома для файловой системы, например, **spare**, **home** или **extra**. Это только один из многих параметров, но мы рекомендуем его использовать во всех файловых системах. Он позволяет избавиться от ручного отслеживания устройства, на котором расположена файловая система, при ее монтировании. Это особенно удобно, если имя дискового устройства может изменяться.

Остальные опции, указываемые в *другие_параметры*, зависит от конкретной файловой системы, и используется редко.

Команда fsck: проверка и исправление файловых систем

Из-за буферизации блоков и того факта, что дисковые накопители на самом деле не являются транзакционными устройствами, структуры данных на файловых системах могут стать рассогласованными. Если эти проблемы не устранить как можно скорее, то это превращается в снежный ком.

Первоначально проблемы с файловой системой устранялись с помощью вызова команды **fsck** ("filesystem consistency check"; читается как "FS check" или "fisk"), которая тщательно проверяла все структуры данных и просматривала дерево выделения для каж-

дого файла. Она использовала набор эвристических правил, описывающих возможный внешний вид файловой системы после повреждения, в разные моменты времени в процессе ее обновления.

Оригинальная схема **fsck** работала удивительно хорошо, но поскольку она предусматривала чтение всех данных с диска, для больших дисков процедура могла занимать несколько часов. Одно из первых решений этой задачи было основано на концепции бита "чистой файловой системы", который устанавливался в суперблоке, когда файловая система корректно демонтировалась. При повторном монтировании файловой системы этот бит проверялся, и таким образом можно было узнать, что проверку с помощью команды **fsck** можно пропустить.

В настоящее время журналы файловой системы позволяют команде **fsck** сосредоточить свое внимание на том, что произошло во время сбоя. С их помощью команда **fsck** может просто восстановить предыдущее согласованное состояние файловой системы.

Как правило, диски обрабатываются командой **fsck** автоматически во время начальной загрузки системы, если они указаны в системном файле **/etc/fstab**. В файле **fstab** есть уже устаревшие поля, с помощью которых задавался порядок проверки файловых систем с помощью команды **fsck** и возможность распараллеливания этой работы. Однако в настоящее время команда **fsck** выполняется настолько быстро, что для нее имеет значение только один фактор — чтобы корневая файловая система проверялась первой.

Команду **fsck** можно запустить вручную, чтобы выполнить глубокую проверку, которая была характерна для ее первых версий, но для этого потребуется время.



Семейство файловых систем ext в операционной системе Linux может быть настроено так, чтобы повторная проверка выполнялась после демонтирования определенное количество раз или по истечении определенного периода времени, даже если все демонтированные файловые системы были "чистыми". Это полезная процедура, и в большинстве ситуаций приемлемым является значение, задаваемое по умолчанию (около 20 монтирований). Однако если файловые системы монтируются часто, как это происходит в настольных рабочих станциях, то даже эта частота выполнения команды **fsck** может стать чрезмерной. Для того чтобы увеличить интервал времени до 50 монтирований, следует использовать команду **tune2fs**.

```
$ sudo tune2fs -c 50 /dev/sda3
tune2fs 1.43.3 (04-Sep-2016)
Setting maximal mount count to 50
```

Если файловая система повреждена и команда **fsck** не может восстановить ее автоматически, то *не следует с ней экспериментировать*, пока не будет создана надежная резервная копия. Лучшая политика страхования — применить ко всему диску команду **dd** и создать резервный файл или диск.

Большинство файловых систем создает каталог **lost+found** в корневом разделе каждой файловой системы, где команда **fsck** может хранить файлы, для которых невозможно определить родительский каталог. Каталогу **lost+found** предварительно выделяется дополнительное дисковое пространство, чтобы команда **fsck** могла хранить "осиротевшие" файлы, не создавая дополнительных каталогов в нестабильной файловой системе. Не удаляйте этот каталог.[14]

[14] В некоторых системах для восстановления этого каталога после удаления используется команда **mklost+found**.

Поскольку имя, присвоенное файлу, записывается только в родительском каталоге, имена "осиротевших" файлов недоступны, и файлы, хранящиеся в каталоге **lost+found**, именуются по названиям их индексных дескрипторов. Однако таблица индексных дескрипторов содержит идентификаторы UID владельца файла, что позволяет легко возвращать файл его владельцу.

Монтирование файловой системы

Файловую систему необходимо смонтировать до того, как она станет видимой для процессов. Точкой монтирования для файловой системы может быть любой каталог, поэтому файлы и подкаталоги, которые расположены ниже этой точки в иерархии, останутся недоступными, пока другая файловая система будет смонтирована в этой точке. Более подробную информацию можно найти в разделе 5.2.

После инсталляции нового диска необходимо смонтировать новые файловые системы вручную, чтобы убедиться, что все работает правильно. Например, команда

```
$ sudo mount /dev/sda1 /mnt/temp
```

монтирует файловую систему в разделе, представленном файлом устройства **/dev/sda1** (имена устройств зависят от операционной системы), в каталоге **/mnt**, который является традиционным для временных точек монтирования.

Размер файловой системы можно проверить с помощью команды **df**. В приведенном ниже примере флаг **-h** системы Linux используется для выдачи результатов в понятном для человека виде. К сожалению, большинство системных значений, заданных по умолчанию для команды **df**, относятся к бесполезным сущностям, таким как "дисковые блоки", но есть флаг, заставляющий команду **df** выдавать конкретную информацию, например двоичные кило- или гигабайты.

```
$ df -h /mnt/web1
Filesystem              Size  Used  Avail  Use%  Mounted on
/dev/mapper/DEMO-web1   197G   60M   187G    1%  /mnt/web1
```

Настройка автоматического монтирования

Обычно операционные системы настраиваются так, чтобы локальные файловые системы автоматически монтировались на этапе начальной загрузки. В файле конфигурации (среди прочей информации) **/etc/fstab** перечислены имена устройств и точки монтирования для всех системных дисков.

Команды **mount**, **umount**, **swapon** и **fsck** читают содержимое файла **fstab**, поэтому желательно, чтобы представленные в этом файле данные были правильными и полными. Команды **mount** и **umount** используют этот файл, чтобы выяснить, что мы хотим, указывая в командной строке только имя раздела или точку монтирования. Например, при конфигурации файла **fstab** в системе Linux, приведенной далее, команда

```
$ sudo mount /media/cdrom0
```

эквивалентна следующей команде.

```
$ sudo mount -t udf -o user,noauto,exec,utf8 /dev/scd0 /media/cdrom0
```

Команда **mount -a** монтирует все регулярные файловые системы, перечисленные в файле **fstab**; обычно эта команда выполняется в рамках сценария запуска на этапе

начальной загрузки.[15] Аргумент **-t** *тип_файловой_системы* ограничивает операции над файловыми системами только указанного типа. Например, команда

```
$ sudo mount -at ext4
```

монтирует все локальные файловые системы ext4. Команда **mount** читает файл **fstab** последовательно. Следовательно, файловые системы, смонтированные под другими файловыми системами, должны следовать за своими родительскими разделами, указанными в файле **fstab**. Например, строка для файла **/var/log** должна следовать за строкой **/var**, если **/var** — отдельная файловая система.

Команда **umount**, предназначенная для демонтирования файловых систем, имеет аналогичный синтаксис. Файловую систему, которую некий процесс использует в качестве своего текущего каталога или которая содержит открытый файл, демонтировать нельзя. Существуют команды, позволяющие идентифицировать процессы, препятствующие попыткам выполнить команду **umount**; см. раздел 5.2.



Файл **fstab** в системе FreeBSD является наиболее традиционным из наших примеров систем. Рассмотрим его записи для системы, имеющей только одну файловую систему, смонтированной под корневой файловой системой (**/spare**).

```
# Device          Mountpoint      FStype    Options    Dump    Pass#
/dev/gpt/rootfs   /               ufs       rw         1       1
/dev/gpt/swap-a   none            swap      sw         0       0
/dev/gpt/swap-b   none            swap      sw         0       0
fdesc             /dev/fd         fdescfs   rw         0       0
proc              /proc           procfs    rw         0       0
/dev/gpt/spare    /spare          ufs       rw         0       0
```

Каждая строка содержит шесть полей, разделенных пробелами, и описывает отдельную файловую систему. Для повышения читабельности поля обычно выравниваются, но в принципе это не требуется.

🕮 Дополнительную информацию о системе NFS см. в главе 21.

Первое поле содержит имя устройства. Файл **fstab** может содержать точки монтирования, находящиеся в удаленных системах. В этом случае первое поле содержит путь NFS. Обозначение *server:/export* означает каталог */export* на машине с именем *server*.

Второе поле задает точку монтирования, а третье — тип файловой системы. Точное название типа, идентифицирующего локальные файловые системы, зависит от конкретной машины.

Четвертое поле задает опции команды **mount**, которые должны применяться по умолчанию. Существует множество возможностей; опции, общие для всех типов файловых систем, можно найти на справочной странице для команды **mount**. Отдельные файловые системы обычно имеют собственные опции.

Пятое и шестое поля являются рудиментарными. Предположительно они задавали "частоту создания копии" и параметр управления параллельным выполнением команд **fsck**. Ни одна из этих возможностей в современных системах не актуальна.

Устройства, указанные для файловых систем **/dev/fd** и **/proc**, являются фиктивными элементами. Эти виртуальные файловые системы зависят от конкретной операционной системы и не требуют дополнительной информации для монтирования. Другие

[15] Опция `noauto` команды **mount** исключает указанную файловую систему из процесса автоматического монтирования с помощью команды **mount -a**.

устройства идентифицируются их метками разделов GPT, что является более надежным вариантом, чем использование фактических имен устройств. Чтобы узнать метку существующего раздела, выполните команду

gpart show -l *диск*

для вывода таблицы разделов соответствующего диска. Чтобы установить метку для раздела, используйте команду

gpart modify -i *индекс* **-l** *метка диск*[16]

Файловые системы UFS также имеют собственные метки, и они отображаются в каталоге **/dev/ufs**. Метки UFS и метки разделов являются разными, но им можно (и, вероятно, следует) присвоить одинаковые значения. В этом примере файловая система **/dev/ufs/spare** будет работать так же, как **/dev/gpt/spare**. Чтобы определить текущую метку файловой системы, выполните команду

tunefs -p *устройство*

Чтобы установить метку, выполните команду

tunefs -L *метка устройство*

Размонтируйте файловую систему перед установкой метки.



Ниже приводятся примеры файла **fstab** из системы Ubuntu. Общий формат тот же, но системы семейства Linux часто имеют разные особенности.

```
# <file system> <mount point>   <type>       <options>              <d>   <p>
proc            /proc           proc         defaults                0     0
UUID=a8e3...8f8a /              ext4         errors=remount-ro       0     1
UUID=13e9...b8d2 none           swap         sw                      0     0
/dev/scd0       /media/cdrom0   udf,iso9660  user,noauto,exec,utf8   0     0
```

Первая строка относится к файловой системе **/proc**, которая фактически представлена драйвером ядра и не имеет вспомогательного запоминающего устройства. Устройство `proc`, указанное в первом столбце, просто занимает место.

Вторая и третья строки для идентификации томов используют идентификаторы разделов (идентификаторы UUID, которые мы сократили, чтобы не снизить читабельность текста), а не имена устройств. Эта система похожа на систему меток UFS, используемую в операционной системе FreeBSD, за исключением того, что идентификаторы представляют собой длинные случайные числа вместо текстовых строк. Используйте команду **blkid** для определения UUID конкретной файловой системы.

Файловым системам также можно назначать административные метки; используйте команды **e2label** или **xfs_admin** для их чтения или установки. Если вы хотите использовать метки в файле **fstab** (этот подход является предпочтительным), вместо `UUID=`*длинное_случайное_число* укажите `LABEL=`*метка* .

Разделы диска GPT могут иметь идентификатор UUID и собственные метки, которые не зависят от UUID и меток файловых систем, которые они содержат. Чтобы использовать эти опции для идентификации разделов в файле **fstab**, применяйте параметры `PARTUUID=` и `PARTLABEL=`. Однако обычная практика, похоже, сводится к использованию UUID файловой системы.

[16]Предупреждение: таблицы разделов иногда называются *метками диска*. Убедитесь, что при чтении документации вы различаете метку отдельного раздела и метку самого диска. Перезапись таблицы разделов диска потенциально катастрофична.

Вы также можете идентифицировать устройства по их именам, расположенным в каталоге **/dev/disk**. Поддиректории, такие как **/dev/disk/by-uuid** и **/dev/disk/by-partuuid**, автоматически обслуживаются менеджером устройств udev.

Монтирование USB-накопителя

Применение устройств USB весьма разнообразно: например, персональные флешки, цифровые фото и видеокамеры, большие внешние диски. Большинство из них поддерживается в системах семейства UNIX в качестве накопителей данных.

В прошлом для управления USB-накопителями приходилось прибегать к специальным трюкам. Однако в настоящее время, когда операционные системы реализуют управление динамическими устройствами в качестве фундаментального требования, USB-накопители представляют собой просто еще один тип накопителей, которые появляются и исчезают без предупреждения.

С точки зрения управления с USB-накопителями связаны следующие проблемы.

- Ядро должно распознавать устройство и назначать ему файл устройства.
- Необходимо выяснять, какие назначения были сделаны.

Первый этап обычно выполняется автоматически. После назначения файла устройства с ним можно выполнять операции, описанные выше в разделе 20.4. Дополнительная информация по динамическому управлению устройствами приведена в главе 11.

Включение подкачки

Как правило, в качестве области подкачки используются разделы диска или логические тома, а не структурированные файловые системы. Вместо использования файловой системы для слежения за содержимым области подкачки, ядро поддерживает собственное упрощенное отображение блоков памяти в блоки области подкачки.

В некоторых системах можно выполнять подкачку с помощью файла, находящегося в разделе файловой системы. В старых операционных системах эта конфигурация может работать медленнее, чем при использовании специального раздела подкачки, но в некоторых случаях это бывает очень удобным. В любом случае с помощью менеджера логических томов можно решить большинство проблем, по которым вам может понадобиться использовать файл подкачки, а не том подкачки.

Чем больше область подкачки, которую вы используете, тем больше виртуальной памяти могут занимать ваши процессы. Максимальная производительность виртуальной памяти достигается, когда область подкачки разделена между несколькими физическими накопителями. Разумеется, лучше всего вообще не использовать подкачку; если вам необходимо оптимизировать производительность подкачки, лучше подумайте об увеличении оперативной памяти компьютера.

Конкретный размер области подкачки зависит о того, как используется компьютер. Нет ничего плохого в том (кроме потери дисковой памяти), что будет создан большой раздел для подкачки. Но, как правило, размер области подкачки должен составлять половину размера оперативной памяти компьютера и не быть меньше 2 Гбайт.

Если система может работать в спящем режиме (как правило, на персональных компьютерах), она должна иметь возможность сохранять все содержимое памяти в области подкачки, кроме сохранения страниц при обычной работе. На этих машинах следует увеличить размер области подкачки, рекомендованный выше, на объем оперативной памяти.

Облачные и виртуализированные экземпляры серверов имеют свои особенности в отношении пространства подкачки. Подкачка всегда снижает их производительность, поэтому некоторые источники рекомендуют полностью ее исключить; если вам нужно больше памяти, вам нужен больший экземпляр сервера. С другой стороны, для небольших экземпляров серверов обычно назначаются настолько скудные объемы оперативной памяти, что они могут только едва загрузиться без использования области подкачки. Поэтому, как правило, следует выделять место для области подкачки для любых экземпляров серверов, если только вы не планируете его использовать для других целей, либо вам не нужно за него платить. Какой бы подход вы ни выбрали, проверьте свои базовые образы, чтобы узнать, как они настроены. Некоторые из них поставляются с предварительно настроенной подкачкой, а некоторые нет.

Некоторые экземпляры Amazon EC2 поставляются с локальным хранилищем экземпляров. Это, по сути, кусок локального жесткого диска на машине, на которой запущен гипервизор. Содержимое хранилища экземпляров не сохраняется между сеансами работы. Это пространство входит в цену экземпляра, поэтому вы можете использовать ее для пространства подкачки, если нет ничего другого.



В системах Linux область подкачки инициализируется с помощью команды **mkswap**, которой нужно передать имя устройства в качестве аргумента. Команда **mkswap** записывает некоторую информацию в область заголовка раздела подкачки. Эти данные включают идентификатор UUID, поэтому разделы подкачки считаются файловыми системами с точки зрения **/etc/fstab** и могут быть идентифицированы по UUID.

Вы можете вручную включить подкачку на конкретном устройстве с помощью команды **swapon** *устройство*. Однако лучше, чтобы эта функция выполнялась автоматически во время загрузки. Просто укажите области подкачки в файле **fstab** и присвойте им тип `swap` для файловой системы. Чтобы просмотреть активную конфигурацию системы подкачки, выполните команду **swapon -s** в системе Linux или **swapctl -l** в системе FreeBSD.

20.11. Файловые системы следующего поколения: ZFS и Btrfs

Несмотря на то что ZFS и Btrfs обычно называются файловыми системами, они представляют собой вертикально интегрированные подходы к управлению хранилищами, которые включают функции диспетчера логических томов и RAID-контроллера. Хотя текущие версии обеих систем имеют несколько ограничений, большинство из них относятся к категории "еще не реализованы", а не к категории "не могут быть реализованы по архитектурным причинам".

Копирование при записи

Как ZFS, так и Btrfs избегают перезаписи данных на месте и вместо этого используют схему, известную как "копирование при записи". Чтобы обновить блок метаданных, например, файловая система модифицирует его копию в памяти, а затем записывает ее в ранее свободный блок диска. Конечно, этот блок данных, вероятно, имеет родительский блок, который указывает на него, поэтому родитель также перезаписывается, как и родитель родителя и так далее до самого верхнего уровня файловой системы. (На

практике кеширование и тщательный дизайн структур данных оптимизируют бо́льшую часть этих записей, по крайней мере в краткосрочной перспективе.)

Преимущество этой архитектуры заключается в том, что копия файловой системы на диске остается всегда постоянно согласованной. Перед обновлением корневого блока файловая система выглядит так же, как и в последний раз, когда был обновлен корень. Несколько пустых блоков будут изменены, но поскольку на них никто не ссылается, это не имеет никакого значения. Файловая система в целом перемещается напрямую из одного согласованного состояния в другое.

Обнаружение ошибок

Файловые системы ZFS и Btrfs также воспринимают целостность данных гораздо более серьезно, чем традиционные файловые системы. В этих системах хранятся контрольные суммы для каждого дискового блока, и в них проверяются все прочитанные блоки, чтобы убедиться, что получены корректные данные. В пулах хранения, которые поддерживают зеркалирование или контроль по четности, испорченные данные автоматически восстанавливаются из заведомо хорошей копии.

В дисковых накопителях реализован собственный уровень обнаружения ошибок и исправления ошибок, и хотя они довольно часто выходят из строя, предполагается, что они должны сообщать об ошибке основному компьютеру. Тем не менее они иногда возвращают испорченные данные без указания ошибки.

Одно из часто упоминаемых эмпирических правил гласит, что на каждые 75 Тбайт данных приходится одна скрытая ошибка. В 2008 г. Байравасундарам и соавторы (Bairavasundaram et al.) опубликовали результаты исследования, в котором они проанализировали служебные записи более чем на 1,5 миллиона дисков на серверах NetApp и обнаружили, что на 0,5% дисков возникают скрытые ошибки чтения в течение каждого года обслуживания.[17]

Частота ошибок мала, но по всем признакам она остается примерно постоянной, несмотря на то, что как и емкость дисков, так и объемы данных, хранящихся на дисках, экспоненциально увеличиваются. Вскоре у нас будут такие большие диски, что будет невозможно прочитать все содержимое, не избежав скрытой ошибки. Дополнительная проверка, выполняемая файловыми системами ZFS и Btrfs, начинает выглядеть очень важной.[18]

Контроль по четности в массивах RAID не решает эту проблему, по крайней мере в обычном режиме. Четность не может быть проверена без чтения содержимого всего логического сегмента, а выполнять чтение всего логического сегмента при каждой дисковой операции неэффективно. Выполнение проверки всего диска (команда **scrub**) может помочь найти скрытые ошибки, но только если они воспроизводимы.

Производительность

Все традиционные файловые системы, которые по-прежнему широко используются, имеют схожую производительность. Можно создать рабочие нагрузки, при которых одна файловая система имеет преимущество, но тесты общего назначения редко показывают большую разницу.

---

[17] Интересно отметить, что одним из ключевых результатов этого исследования было то, что жесткие диски корпоративного уровня на порядок меньше подвержены этим ошибкам.

[18] Связанная, а также недооцененная проблема — это риск случайных ошибок в оперативной памяти. Они нечастые, но они действительно происходят. Все производственные серверы должны использовать (и контролировать!) память с коррекцией ошибок (error correction code — ECC).

Файловые системы с копированием при записи несколько отличаются от традиционных файловых систем, и им не хватает десятилетий постепенного совершенствования, которые привели старые файловые системы в их текущее состояние. Обычно производительность традиционных файловых систем выше производительности файловых систем ZFS и Btrfs.

Во многих тестах ZFS и Btrfs показывают производительность, сравнимую с традиционными файловыми системами. Но в худшем случае эти файловые системы могут быть примерно в два раза медленнее, чем традиционные.

Судя по тестам системы Linux (единственной платформы, на которой возможно прямое сравнение, поскольку файловая система Btrfs предназначена только для Linux), Btrfs в настоящее время имеет небольшое преимущество над ZFS. Однако результаты широко варьируются в зависимости от шаблона доступа. Это не редкость, когда одна из этих файловых систем хорошо работает в определенном контрольном тесте, а другая сильно отстает.

Анализ производительности осложняется тем фактом, что каждая из этих файловых систем "в своем рукаве" имеет некоторые потенциальные трюки для ее повышения. Тесты обычно не учитывают эти возможности. Система ZFS позволяет добавлять кеширование SSD в пул хранения; она автоматически копирует часто считываемые данные в кеш и избегает частого обращения к жестким дискам. В системе Btrfs можно использовать команду `chattr +C`, чтобы отключить семантику копирования при записи для данных в определенных файлах (обычно больших или часто модифицированных), тем самым обойдя некоторые типичные неэффективные сценарии.

Для общего использования в качестве корневых файловых систем и хранилища домашних каталогов ZFS и Btrfs работают хорошо и предлагают множество полезных преимуществ. Они также могут хорошо работать как хранилище данных для конкретных рабочих нагрузок сервера. Однако в этих последних сценариях стоит потратить некоторое время, чтобы тщательно проверить их поведение в конкретной среде.

20.12. Файловая система ZFS: все проблемы решены

Система ZFS появилась в 2005 г. как компонент операционной системы OpenSolaris и быстро вошла в состав системы Solaris 10 и различных дистрибутивных пакетов, основанных на лицензии BSD. В 2008 г. она уже могла использоваться как корневая файловая система и с тех пор стала основным выбором для операционной системы Solaris. UFS продолжает быть корневой файловой системой в операционной системе FreeBSD, но уже версия FreeBSD 10 официально поддерживает файловую систему ZFS как один из возможных вариантов.

Система ZFS — не только файловая система со встроенными функциями менеджера логических томов и контроллера RAID. Как первоначально было задумано для операционной системы OpenSolaris, в ней осуществлен всесторонний пересмотр прежнего подхода к управлению памятью: от способа монтирования файловых систем до их экспортирования в другие системы на основе NFS и SMB.

Современные системы BSD и Linux должны учитывать разнообразие файловых систем и поэтому они были вынуждены немного отступить от оригинального универсального подхода ZFS. Однако ZFS остается тщательно спроектированной системой, которая решает довольно много административных проблем на основе архитектуры, а не с помощью добавления функций.

ZFS в системе Linux

Файловая система ZFS — это бесплатное программное обеспечение. Ее использование в системе Linux было стимулировано тем, что ее исходный код защищен лицензией Common Development and Distribution License (CDDL) компании Sun Microsystems. Организация The Free Software Foundation (FFS) сохраняет несовместимость лицензии CDDL с лицензией GNU Public License, которая защищает ядро Linux. Хотя модульные версии ZFS для Linux долго были доступны через проект OpenZFS (`openzfs.org`), позиция организации FSF препятствовала внедрению файловой системы ZSF в базовые дистрибутивы Linux.

После почти десятилетней паузы позиция организации FSF была оспорена компанией Canonical Ltd., разработчиком операционной системы Ubuntu. Проведя юридический анализ, компания Canonical формально оспорила позицию FSF и включила файловую системы ZFS в версию Ubuntu 16.04 в форме загружаемого модуля ядра. Пока (середина 2017 г.) ни один судебный процесс не закончился. Если компания Canonical останется безнаказанной, файловая система ZFS сможет стать полноценно поддерживаемой корневой файловой системой в операционной системе Ubuntu и другие дистрибутивы смогут последовать ее примеру.[19]

Архитектура ZFS

На рис. 20.3 показаны основные объекты в системе ZFS и их отношения друг с другом. Пул системы ZFS аналогичен группе томов в других системах с механизмами управления логическими томами. Каждый пул состоит из виртуальных устройств, представляющих собой накопители (диски, разделы, устройства SAN и так далее), группы зеркал или массивов RAID. Массив ZFS RAID по существу похож на массив RAID 5 тем, что в нем используется одно или несколько устройств хранения четности для обеспечения избыточности массива. Однако в системе ZFS эта схема называется RAID-Z и в ней используется последовательность логических сегментов переменных размеров, чтобы исключить появление "дырки при записи RAID 5". Все операции записи в пул хранения распределяются по виртуальным устройствам пула, поэтому пул, содержащий только отдельные накопители, по существу представляет собой схему RAID 0, хотя накопители в этой конфигурации не обязаны иметь одинаковые объемы.

К сожалению, текущая версия массива ZFS RAID имеет недостатки. Например, в ней невозможно ни добавить новые устройства в массив после того, как он был определен, ни удалить устройство ZFS. Как и в большинстве реализаций массива RAID, накопители в нем должны иметь одинаковые объемы; вы можете заставить систему ZFS принять накопители переменных объемов, но общий размер массива будет определяться по наименьшему объему накопителя. Для того чтобы эффективно использовать диски переменных объемов в сочетании со схемой ZFS RAID, необходимо заранее разбить диски на разделы и определить оставшиеся области в качестве отдельных устройств.

Большая часть работы по настройке системы ZFS и управлению ею выполняется посредством двух команд: **`zpool`** и **`zfs`**. Команда **`zpool`** используется для создания пулов хранения и управления ими, а команда **`zfs`** предназначена для создания сущностей, возникших из пула, в основном файловых систем и томов, используемых в качестве области подкачки, хранилищ баз данных, и поддержки томов SAN.

[19] История ZFS — интересный случай, когда лицензия GPL активно препятствует разработке пакета программного обеспечения с открытым исходным кодом и блокирует его принятие пользователями и дистрибьюторами. Если вас интересуют юридические данные, почитайте новости Ричарда Фонтаны (Richard Fontana) за 2016 г. на сайте **`goo.gl/PC9i3t`**.



Рис. 20.3. Архитектура файловой системы ZFS

Пример: добавление диска

Прежде чем погрузиться в детали системы ZFS, рассмотрим небольшой пример. Допустим, что мы добавили новый диск в систему FreeBSD и он получил имя **/dev/ada1**. (Для того чтобы определить требуемое устройство, следует выполнить команду **geom disk list**.)

На первом этапе необходимо создать на этом диске новый пул хранения.

```
$ sudo zpool create demo ada1
```

На втором этапе... Ну, на самом деле второго этапа нет. Система ZFS создает пул **demo**, корневую файловую систему внутри этого пула и монтирует эту файловую систему как каталог **/demo** за один этап. Файловая система будет смонтирована автоматически при загрузке системы.

```
$ ls -a /demo
...
```

Пример был бы еще более впечатляющим, если бы мы могли просто добавить новый диск в существующий пул хранения корневого диска, который по умолчанию называется **zroot**. (Эта команда могла бы выглядеть так: **sudo zpool add zroot ada1**.) К сожалению, корневой пул может содержать только одно виртуальное устройство. Тем не менее остальные пулы допускают безболезненное расширение.

Файловые системы и свойства

Одним из замечательных свойств ZFS является то, что она позволяет автоматически создавать файловые системы по умолчанию в новом пуле хранения, причем эти файловые системы не будут занимать какой-то предопределенный объем памяти. Все файловые системы, существующие в пуле, могут занимать доступное пулу пространство.

В отличие от традиционных файловых систем, которые не зависят друг от друга, файловые системы ZFS являются иерархическими и могут взаимодействовать со своими родительскими и дочерними файловыми системами несколькими способами. Новая файловая система создается с помощью команды **zfs create**.

```
$ sudo zfs create demo/new_fs
$ zfs list -r demo
```

```
NAME          USED    AVAIL    REFER    MOUNTPOINT
demo          432K     945G      96K    /demo
demo/new_fs   96K      945G      96K    /demo/new_fs
```

Файл **-r** в команде **zfs list** означает, что она рекурсивно применяется ко всем дочерним файловым системам. Большинство остальных подкоманд **zfs** также понимают флаг **-r**. Еще более полезно то, что система ZFS автоматически монтирует новую файловую систему сразу же после ее создания.

Для того чтобы имитировать традиционные файловые системы, имеющие фиксированный размер, к свойствам файловой системы можно добавить параметры `reservation` (объем дисковой памяти, зарезервированный для пула хранения и предназначенный для использования файловой системой) и `quota`. Эта настройка является ключевой в управлении системой ZFS и представляет собой определенную смену парадигмы для администраторов, работающих с другими системами. Для примера зададим оба значения равными 1 Гбайт.

```
$ sudo zfs set reservation=1g demo/new_fs
$ sudo zfs set quota=1g demo/new_fs
$ zfs list -r demo
NAME          USED    AVAIL   REFER   MOUNTPOINT
demo          1.00G    944G     96K   /demo
demo/new_fs     96K   1024M     96K   /demo/new_fs
```

Новое значение параметра **quota** отражается в столбце `AVAIL` для файловой системы **/demo/new_fs**. Значение параметра **reservation** показано в столбце `USED` для файловой системы **/demo**. Это объясняется тем, что резерв для файловых систем, дочерних по отношению к системе **/demo**, должен быть учтен в ее размере. [20]

Оба изменения этих свойств отражаются только в системе учета. Единственное изменение, которое действительно касается пула хранения, — это изменение одного или нескольких блоков данных при записи новых параметров. При этом не запускается ни один процесс, чтобы отформатировать один гигабайт дискового пространства, зарезервированного для файловой системы **/demo/new_fs**. Большинство операций ZFS, включая создание нового пула хранения и новых файловых систем, выполняется так же легковесно.

Используя иерархическую систему управления дисковым пространством, можно легко группировать несколько файловых систем, чтобы гарантировать, что их общий объем не превысит заданного порога; нет необходимости задавать этот порог для каждой файловой системы отдельно.

Параметры **quota** и **reservation** следует задавать так, чтобы они правильно имитировали традиционную файловую систему фиксированного размера. [21] Сам по себе параметр **reservation** просто гарантирует, что файловая система будет иметь достаточно дисковой памяти, чтобы достичь *по крайней мере* указанного размера. Параметр **quota** ограничивает размер файловой системы *без гарантии*, что она будет иметь этот объем памяти для своего увеличения; другая файловая система может занять все свободное пространство пула, не оставив места для расширения файловой системы **/demo/new_fs**.

---

[20] Столбец `REFER` содержит объем данных в активной копии каждой файловой системы. Файловые системы **/demo** и **/demo/new_fs** имеют близкие значения `REFER`, поскольку обе они пустые, а не потому что между ними существует какая-то связь.

[21] Параметры **reservation** и **quota** учитывают весь объем дисковой памяти, занимаемый файловой системой, включая пространство, занятое мгновенными копиями. Если вы хотите ограничить размер только активной копии файловой системы, следует использовать параметры **refreservation** и **refquota**. Префикс **ref** означает "объем данных, на которые ссылается файловая система". Именно это значение показано в столбце `REFER` в результатах работы команды **zfs list**.

С другой стороны, в реальной жизни существует очень мало причин для такой настройки файловой системы. Эти свойства следует использовать только для демонстрации возможностей системы учета памяти в ZFS и чтобы подчеркнуть, что система ZFS при необходимости может быть совместимой с традиционной моделью.

Наследование свойств

Многие свойства естественным образом наследуются дочерними файловыми системами. Например, если мы хотим смонтировать корень пула файловой системы **demo** в каталоге **/opt/demo**, а не **/demo**, можно просто задать соответствующий параметр **mountpoint**.

```
$ sudo zfs set mountpoint=/mnt/demo demo
$ zfs list -r demo
NAME          USED    AVAIL   REFER    MOUNTPOINT
demo          1.00G   944G      96K    /mnt/demo
demo/new_fs     96K   1024M     96K    /mnt/demo/new_fs

$ ls /mnt/demo
new_fs
```

Параметр **mountpoint** автоматически демонтирует файловые системы, а изменение точки монтирования влияет на дочерние файловые системы предсказуемым и очевидным образом. Обычные правила, касающиеся файловой системы, тем не менее, остаются в силе (см. раздел 5.2).

Для того чтобы увидеть действительное значение конкретного свойства, используется команда **zfs get**, а команда **zfs get all** выводит список значений всех параметров. Столбец SOURCE содержит информацию, объясняющую, почему каждое свойство имеет конкретное значение: слово `local` означает, что свойство было задано явно, а дефис — что свойство предназначено только для чтения. Если значение свойства унаследовано от родительской файловой системы, то в столбце SOURCE будут указаны детали того наследования.

```
$ zfs get all demo/new_fs
NAME         PROPERTY       VALUE                  SOURCE
demo/new_fs  type           filesystem             -
demo/new_fs  creation       Mon Apr 03 0:12 2017   -
demo/new_fs  used           96K                    -
demo/new_fs  available      1024M                  -
demo/new_fs  referenced     96K                    -
demo/new_fs  compressratio  1.00x                  -
demo/new_fs  mounted        yes                    -
demo/new_fs  quota          1G                     local
demo/new_fs  reservation    1G                     local
demo/new_fs  mountpoint     /mnt/new_fs            inherited from demo
demo/new_fs  checksum       on                     default
demo/new_fs  compression    off                    default
... <и т.д. всего 55 строк>
```

Внимательные читатели могут заметить, что свойства `available` и `referenced` подозрительно похожи на столбцы AVAIL и REFER, которые отображаются командой **zfs list**. Фактически команда **zfs list** — это просто другой способ отображения свойств файловой системы. Если бы выше мы привели полный список результатов, полученных с помощью команды **zfs get**, то среди них было бы и свойство `used`. Для того чтобы

указать, какие именно свойства нас интересуют, их можно указать с помощью опции **-o** команды **zfs list**.

Поскольку нет никакого смысла задавать явно значения свойств `used` и других свойств, связанных с объемом, эти свойства предназначаются только для чтения. Если конкретные правила вычисления значения свойства `used` вас по каким-то причинам не устраивают, возможно, следует использовать другие свойства, такие как `usedbychildren` и `usedbysnapshots`, чтобы увидеть, как расходуется дисковая память.

Для собственного использования и создания локальных сценариев можно задавать дополнительные, нестандартные свойства файловых систем. Процесс их задания ничем не отличается от задания стандартных свойств. Имена пользовательских свойств должны содержать двоеточие, чтобы отличать их от стандартных свойств.

Один пользователь — одна файловая система

Поскольку файловые системы не расходуют дисковую память и не требуют много времени на свое создание, их оптимальное количество может быть скорее ближе к "много", чем к "мало". Если рабочие каталоги пользователей находятся в пуле хранения системы ZFS, рекомендуется создать отдельный рабочий каталог в отдельной файловой системе.

Существует несколько преимуществ такой схемы.

- Если необходимо установить квоты по использованию дискового пространства, то рабочие каталоги пользователей являются естественным объектом для этой цели. Квоты можно установить как для файловых систем индивидуальных пользователей, так и для файловых систем, содержащих всех пользователей.
- Мгновенные копии соответствуют отдельным файловым системам. Если каждый рабочий каталог пользователя представляет собой отдельную файловую систему, то пользователь может получить доступ к старым мгновенным копиям с через каталог **~/.zfs**.[22] Одно это сэкономит администратору много времени, поскольку пользователи смогут самостоятельно обслуживать свои потребности в восстановлении файлов.
- Система ZFS позволяет делегировать разрешение выполнять разные операции, такие как поиск мгновенных копий и откат системы в предыдущее состояние. При желании можно передать пользователям контроль над этими операциями, которые выполняются в их собственных каталогах. В этой книге мы не будем описывать детали механизма управления разрешениями в системе ZFS; их можно найти в справочнике *ZFS Administration Guide*, а также на страницах электронной справки, посвященной команде **zfs allow**.

Мгновенные копии и клоны

Как и менеджер логических томов, система ZFS обеспечивает копирование при записи на пользовательском уровне, разрешая создавать мгновенные копии. Однако есть важное отличие: мгновенные копии системы ZFS реализуются для файловых систем, а не для логических томов, поэтому они имеют произвольную глубину детализации.

Мгновенную копию можно создать с помощью команды **zfs snapshot**. Например, следующая последовательность команд иллюстрирует создание мгновенной копии, ее

[22] Этот каталог по умолчанию является скрытым; он не появляется среди результатов работы команды **ls -a**. Его можно сделать видимым с помощью команды **zfs set snapdir=visible** *файловая_система*.

использование с помощью каталога **.zfs/snapshot** и возвращение файловой системы в предыдущее состояние.

```
$ sudo touch /mnt/demo/new_fs/now_you_see_me
$ ls /mnt/demo/new_fs
now_you_see_me
$ sudo zfs snapshot demo/new_fs@snap1
$ sudo rm /mnt/demo/new_fs/now_you_see_me
$ ls /mnt/demo/new_fs
$ ls /mnt/demo/new_fs/.zfs/snapshot/snap1
now_you_see_me
$ sudo zfs rollback demo/new_fs@snap1
$ ls /opt/demo/new_fs
now_you_see_me
```

Каждой мгновенной копии в момент ее создания можно присвоить имя. Полная спецификация мгновенной копии обычно записывается в виде *файловая_система@мгновенная_копия*.

Для рекурсивного создания мгновенных копий используется команда **zfs snapshot -r**. Ее эффект эквивалентен выполнению команды **zfs snapshot** для каждого объекта по отдельности. При этом для каждого подкомпонента будет создана собственная мгновенная копия. Все мгновенные копии имеют одинаковые имена, но с логической точки зрения они отличаются друг от друга, поскольку их часть *файловая_система* будет другой.

Мгновенные копии в системе ZFS предназначены только для чтения, несмотря на определенные свойства, они все же не являются файловыми системами в подлинном смысле. Однако вы можете инициировать мгновенную копию как полноценную и допускающую запись файловую систему, "клонировав" ее.

```
$ sudo zfs clone demo/new_fs@snap1 demo/subclone
$ ls /mnt/demo/subclone
now_you_see_me
$ sudo touch /mnt/demo/subclone/and_me_too
$ ls /mnt/demo/subclone
and_me_too now_you_see_me
```

Мгновенная копия, которая стала оригиналом для клона, остается нетронутой и предназначается только для чтения. Однако новая файловая система (в нашем примере **demo/subclone**) сохраняет связь как с мгновенной копией, так и с файловой системой, на которой она основана, и ни одну из них нельзя удалить, пока существует клон.

Операция клонирования применяется относительно редко, но это единственный способ создать новую ветвь на дереве эволюции файловой системы. Операция **zfs rollback**, продемонстрированная выше, может только восстановить файловую систему в состоянии, соответствующем самой последней по времени мгновенной копии. Поэтому перед ее использованием вам нужно будет перманентно удалить (**zfs destroy**) все мгновенные копии, сделанные за время, прошедшее после создания мгновенной копии, к которой хотите вернуться. Клонирование позволяет вам вернуться в прошлое без потери изменений, внесенных недавно.

Предположим, вы обнаружили дыру в системе безопасности, возникшую в течение прошедшей недели. Для обеспечения безопасности вы захотите вернуть файловую систему в состояние, в котором она находилась неделю назад, чтобы быть уверенным, что она не содержит лазеек, оставленных хакерами. В то же время вы не хотите потерять результаты работы за последнюю неделю или данные для аналитического отчета. Решением этой проблемы является клонирование мгновенной копии, сделанной неде-

лю назад, в виде новой файловой системы, применение команды **zfs rename** к старой файловой системе и команды **zfs rename** — к клону, заменяющему исходную файловую систему.

Полезно также применить к клону команду **zfs promote**; эта операция инвертирует отношения между клоном и оригиналом файловой системы. После активации основная файловая система имеет доступ ко всем старым мгновенным копиям файловой системы, а старая файловая система становится "клонированным" ответвлением.

Неразмеченные логические тома

Области подкачки и неразмеченные области для хранения данных создаются с помощью команды **zfs create** точно так же, как создаются файловые системы. Аргумент **-V** *размер* заставляет команду **zfs** обрабатывать новый объект как неразмеченный логический том, а не файловую систему. В параметре *размер* может использоваться любая известная единица измерения, например **128m**.

Поскольку том не содержит файловую систему, он не монтируется; вместо этого он появляется в каталоге **/dev/zvol** и ссылаться на него можно так, будто он представляет собой жесткий диск или раздел. Система ZFS выполняет зеркальную копию иерархической структуры пула хранения в этих каталогах, поэтому команда **sudo zfs create -V 128m demo/swap** создает том подкачки размером 128 Мбайт, размещенный в каталоге **/dev/zvol/demo/swap**.

Мгновенные копии неразмеченных томов можно создавать точно так же, как мгновенные копии файловых систем, но поскольку в данном случае нет иерархии файловой системы, в которой размещается каталог **.zfs/snapshot**, мгновенные копии оказываются в том же самом каталоге, что и их исходные тома. Клоны функционируют точно так же, как ожидается.

По умолчанию неразмеченные тома получают резерв дисковой памяти, равный его указанному объему. Можно уменьшить резерв или отказаться от него вообще, но в этом случае попытки записи на том могут вызвать ошибку нехватки памяти. Клиенты неразмеченных томов могут не справиться с обработкой таких ошибок.

Управление пулом памяти

Рассмотрев свойства системы ZFS, относящиеся к файловой системе и уровню "блок-клиент", исследуем ее пулы хранения.

До сих пор мы использовали пул с именем `demo`, который создали на отдельном диске. Команда **zpool list** отображает следующие результаты.

```
$ zpool list
NAME   SIZE  ALLOC   FREE   EXPANDSZ   FRAG  CAP   DEDUP  HEALTH  ALTROOT
demo   976M   516K   976G          -    0%   0%   1.00x  ONLINE  -
zroot  19.9G  16.3G  3.61G          -   24%  81%   1.00x  ONLINE  -
```

Пул с именем **zroot** содержит загружаемую корневую файловую систему. На загружаемые пулы в настоящее время наложено несколько ограничений: они могут содержать только одно виртуальное устройство, и это устройство должно быть либо зеркальным массивом, либо отдельным диском; оно не может состоять из расщепленных логических сегментов и быть массивом RAID-Z. (Пока неясно, как это интерпретировать: как предел реализации или как значительное повышение надежности корневой файловой системы.)

Команда **zpool status** добавляет несколько деталей о виртуальных устройствах, из которых состоит пул хранения, и сообщает об их текущем состоянии.

```
$ zpool status demo
  pool: demo
state: ONLINE
  scan: none requested
config:

   NAME      STATE   READ  WRITE  CKSUM
   demo      ONLINE    0      0      0
     ada1    ONLINE    0      0      0

errors: No known data errors
```

Оставим пул **demo** и создадим нечто более сложное. Мы подключили к нашей системе пять накопителей емкостью 1 Тбайт. Сначала создадим пул с именем `monster`, содержащий три таких накопителя, в конфигурации RAID-Z с одним разрядом контроля четности (single-parity configuration).

```
$ sudo zpool destroy demo
$ sudo zpool create monster raidz1 ada1 ada2 ada3
$ zfs list monster
NAME     USED    AVAIL  REFER   MOUNTPOINT
monster  87.2K   1.84T  29.3K   /monster
```

Система ZFS также распознает схемы **raidz2** и **raidz3** для конфигурации с двумя и тремя разрядами контроля четности. Минимальное количество дисков всегда на единицу больше, чем количество битов контроля четности. В данном случае один из трех накопителей предназначен для контроля четности, поэтому для файловой системы доступно примерно два терабайта памяти.

Для иллюстрации добавим оставшиеся два накопителя, сконфигурированных как зеркало.

```
$ sudo zpool add monster mirror ada4 ada5
invalid vdev specification
use '-f' to override the following errors:
mismatched replication level: pool uses raidz and new vdev is mirror
$ sudo zpool add -f monster mirror ada4 ada5
```

Команда **zpool** сначала "спотыкается" на этой конфигурации, потому что эти два виртуальных устройства имеют разные схемы избыточности. Данная конфигурация работает хорошо, поскольку оба виртуальных устройства обладают определенной избыточностью. В реальных условиях не следует смешивать виртуальные устройства, обладающие и не обладающие избыточностью, поскольку невозможно предсказать, какие блоки на каких устройствах будут находиться; частичная избыточность здесь бесполезна.

```
$ zpool status monster
  pool: monster
state: ONLINE
  scan: none requested
config:

    NAME          STATE    READ  WRITE  CKSUM
    monster       ONLINE   0     0      0
       raidz1-0   ONLINE   0     0      0
       ada1       ONLINE   0     0      0
       ada2       ONLINE   0     0      0
```

```
      ada3      ONLINE   0     0     0
    mirror-1    ONLINE   0     0     0
      ada4      ONLINE   0     0     0
      ada5      ONLINE   0     0     0

errors: No known data errors
```

Система ZFS распределяет записи между всеми виртуальными устройствами пула. Как показано в нашем примере, виртуальные устройства не обязательно должны иметь одинаковый размер.[23] Однако компоненты внутри избыточной группы должны иметь одинаковые размеры. Если это условие не выполняется, то каждый компонент будет иметь минимальный размер. При использовании нескольких простых дисков в пуле хранения важную роль играет конфигурация RAID 0.

Дополнительные виртуальные устройства можно добавить в пул в любое время. Однако существующие данные не будут перераспределены, чтобы воспользоваться преимуществом параллелизма. К сожалению, в настоящее время невозможно добавлять дополнительные устройства в существующий массив RAID или зеркало. И как раз здесь файловая система Btrfs имеет определенные преимущества, поскольку она позволяет выполнять все виды реорганизации пулов относительно безболезненно и в автоматическом режиме.

Система ZFS имеет особенно хорошую реализацию кеширования по чтению, которое обеспечивает эффективное использование накопителей SSD. Для того чтобы установить эту конфигурацию, достаточно просто добавить накопители SSDs в пул хранения как виртуальные устройства типа **cache**. Система кеширования использует алгоритм адаптивной замены, разработанный компанией IBM, который эффективнее обычного алгоритма кеширования LRU (least recently used — наиболее давно использовавшийся). Ей известна частота обращения к блокам и момент времени, когда они использовались в последний раз, поэтому чтение больших файлов не предполагает очистку кеша.

Горячее резервирование реализуется с помощью виртуальных устройств типа **spare**. Один и тот же диск можно добавлять в несколько пулов хранения; при этом диск из пула, который первый даст сбой, будет заменен резервным.

20.13. Файловая системы Btrfs: облегченная версия ZFS для Linux

Проект файловой системы Btrfs компании Oracle (аббревиатура "B-tree file system" официально произносится как "butter FS" ("баттэ эфэс") или "better FS" ("беттэ эфэс), хотя трудно не думать о "butter face" ("баттэ фейс"[24])) стремился повторить многие достижения ZFS на платформе Linux в течение долгой паузы, когда разработчикам ZFS показалось, что эта система может быть потеряна для Linux из-за проблем с лицензированием.

Хотя файловая система Btrfs по-прежнему активно разрабатывается, она является стандартной частью ядра Linux с 2009 г. Она доступна и готова к использованию практически во всех Linux-системах, а в системе SUSE Enterprise Linux она даже поддерживается для корневой файловой системы. Поскольку кодовая база развивается быстро, вероятно, сейчас лучше избегать использования Btrfs в системах, для которых важна стабильность работы, таких как Red Hat; старые версии Btrfs имеют известные проблемы.

[23] В данном примере диски имеют одинаковые размеры, а виртуальные устройства нет (2 Тбайт против 1 Тбайт).

[24] "Butter face" на сленге означает "девушка с красивой фигурой, но некрасивым лицом". — *Примеч. ред.*

Btrfs или ZFS

Поскольку файловые системы Btrfs и ZFS имеют общий технический фундамент, их сравнение, вероятно, неизбежно. Однако Btrfs не является клоном ZFS и не пытается воспроизвести ее архитектуру. Например, тома Btrfs монтируются точно так же, как и в других файловых системах, с помощью команды `mount` или указания в файле `/etc/fstab`.

Хотя тома и подтома файловой системы Btrfs существуют в общем пространстве имен, между ними нет иерархических отношений. Чтобы внести изменения в группу подтомов Btrfs, необходимо изменить каждый из них по отдельности. Команды Btrfs не работают рекурсивно, а свойства тома не наследуются. Это не упущение: по мнению разработчиков, незачем загружать файловую систему функциями, которые можно эмулировать в сценарии оболочки.

Система Btrfs отражает это стремление к простоте различными способами. Например, пулы памяти Btrfs могут включать только одну группу дисков в одной конкретной конфигурации (например, RAID 5), тогда как пулы ZFS могут включать в себя несколько групп дисков, а также кешировать диски, журналы регистрации намерений и горячие резервные копии.

Как обычно бывает в области программного обеспечения, горячие споры относительно сравнительных достоинств ZFS и Btrfs в основном сосредоточены на стилистических различиях. Однако несколько важных различий между этими двумя системами поднимаются выше уровня придирок и личных предпочтений.

- Файловая система Btrfs является явным победителем, когда дело доходит до изменения конфигурации вашего оборудования; ZFS даже не пытается соревноваться в этом. Вы можете добавлять или удалять диски в любое время или даже изменять тип RAID, а система Btrfs соответственно перераспределяет существующие данные, оставаясь в режиме онлайн. В системе ZFS такие изменения, как правило, невозможны без копирования данных на внешние носители и их повторной перезаписи.
- Даже без использования функций, требующих большого объема памяти (например, дедупликации), система ZFS лучше всего работает с большим объемом оперативной памяти. Рекомендуемый минимум — 2 Гбайт. Это большой объем памяти для виртуального сервера.
- Способность ZFS кешировать часто считываемые данные на отдельных кеширующих SSD-дисках является уникальным преимуществом во многих сценариях использования, система Btrfs в настоящее время не имеет аналогичных функций.
- По состоянию на 2017 г. реализации RAID-массивов с проверкой четности (RAID 5 и 6) в файловой системе Btrfs еще не были готовы к использованию в производственной среде. Это не наше мнение; это официальное мнение разработчиков и существенный недостаток.

Настройка и преобразование хранилища

В этом разделе мы демонстрируем несколько общих процедур Btrfs, аналогичных тем, которые показаны для ZFS в предыдущих разделах. Сначала создадим файловую систему Btrfs для использования на наборе из двух жестких дисков объемом в 1 Тбайт, настроенных как RAID 1 (зеркалирование):

```
$ sudo mkfs.btrfs -L demo -d raid1 /dev/sdb /dev/sdc
Label:              demo
UUID:
Node size:          16384
Sector size:        4096
Filesystem size:    1.91TiB
Block group profiles:
Data:               RAID1           1.00GiB
Metadata:           RAID1           1.00GiB
System:             RAID1           8.00MiB
SSD detected:       no
Incompat features:  extref, skinny-metadata
Number of devices:  2
Devices:
ID    SIZE        PATH
1     978.00GiB   /dev/sdb
2     978.00GiB   /dev/sdc
$ sudo mkdir /mnt/demo
$ sudo mount LABEL=demo /mnt/demo
```

Мы могли бы назначить имя для любых компонентов устройств в командной строке **mount**, но проще всего использовать метку **demo**, которую мы назначили группе.

Команда **btrfs filesystem usage** показывает, как в данный момент используется пространство на этих дисках.

```
$ sudo btrfs filesystem usage /mnt/demo
Overall:
   Device size:                   1.91TiB
   Device allocated:              4.02GiB
   Device unallocated:            1.91TiB
   Device missing:                  0.00B
   Used:                          1.25MiB
   Free (estimated):            976.99GiB      (min: 976.99GiB)
   Data ratio:                       2.00
   Metadata ratio:                   2.00
   Global reserve:                16.00MiB      (used: 0.00B)

Data,RAID1:   Size:1.00GiB, Used:512.00KiB
   /dev/sdb          1.00GiB
   /dev/sdc          1.00GiB

Metadata,RAID1: Size:1.00GiB, Used:112.00KiB
   /dev/sdb          1.00GiB
   /dev/sdc          1.00GiB

System,RAID1: Size:8.00MiB, Used:16.00KiB
   /dev/sdb          8.00MiB
   /dev/sdc          8.00MiB

Unallocated:
   /dev/sdb        975.99GiB
   /dev/sdc        975.99GiB
```

Интересно отметить небольшие начальные объемы памяти в группе RAID 1, выделенные для данных, метаданных и системных блоков. Большая часть дискового пространства остается в нераспределенном пуле, который не имеет внутренней структуры. Запрошенное зеркалирование не затрагивает диски в целом, только блоки, которые

на самом деле используются. Это не такая жесткая структура, как политика, которая будет реализована на уровне групп блоков.

Это различие является ключом к пониманию того, как файловая система Btrfs может адаптироваться к изменяющимся требованиям и замене оборудования. Вот что происходит, когда мы храним некоторые файлы в новой файловой системе, а затем добавляем третий диск.

```
$ mkdir /mnt/demo/usr
$ cd /usr; tar cf - . | (cd /mnt/demo/usr; sudo tar xfp -)
$ sudo btrfs device add /dev/sdd /mnt/demo
$ sudo btrfs filesystem usage /mnt/demo[25]
Overall:
<пропущено для экономии места>

Data,RAID1:  Size:3.00GiB, Used:2.90GiB
   /dev/sdb      3.00GiB
   /dev/sdc      3.00GiB

Metadata,RAID1: Size:1.00GiB, Used:148.94MiB
   /dev/sdb      1.00GiB
   /dev/sdc      1.00GiB

System,RAID1: Size:8.00MiB, Used:16.00KiB
   /dev/sdb      8.00MiB
   /dev/sdc      8.00MiB

Unallocated:
   /dev/sdb    973.99GiB
   /dev/sdc    973.99GiB
   /dev/sdd    978.00GiB
```

Новый диск **/dev/sdd** стал доступным для пула, но существующие группы блоков остались на своих местах, так как ни в одном из них не используется новый диск. Будущие операции выделения памяти автоматически воспользуются новым диском. При необходимости, мы можем заставить файловую систему Btrfs равномерно распределить данные по всем дискам.

```
$ sudo btrfs balance start --full-balance /mnt/demo
Starting balance without any filters.
Done, had to relocate 5 out of 5 chunks
```

Преобразование между уровнями RAID также является одной из форм балансировки. Теперь, когда у нас есть три диска, мы можем конвертировать массив RAID 1 в RAID 5:

```
$ sudo btrfs balance start -dconvert=raid5 -mconvert=raid5 /mnt/demo
Done, had to relocate 5 out of 5 chunks
```

Если бы мы взглянули на данные об использовании во время преобразования, то увидели бы, что группы блоков одновременно активны и для RAID 1, и для RAID 5. Процедуры удаления диска работают аналогично: система Btrfs постепенно копирует все блоки в группы, которые не включают в себя удаляемый диск, и в конечном итоге данных на нем не остается.

[25] Подкоманды команды **btrfs** могут быть сокращены до любого уникального префикса. Например, вместо команды **btrfs filesystem usage** можно использовать ее сокращенный вариант: **btrfs f u**. Для ясности мы используем полный вариант команды.

Тома и подтома

Мгновенные снимки и квоты являются объектами уровня файловой системы Btrfs, поэтому полезно иметь возможность определять части дерева файлов как разные объекты. Система Btrfs называет эти части "подтомами". Подтом очень похож на обычный каталог файловой системы, и на самом деле он остается доступным как подкаталог его родительского тома, как показано ниже.

```
$ sudo btrfs subvolume create /mnt/demo/sub
Create subvolume '/mnt/demo/sub'
$ sudo touch /mnt/demo/sub/file_in_a_subvolume
$ ls /mnt/demo/sub
file_in_a_subvolume
```

Подтома не монтируются автоматически; они отображаются здесь как часть родительского тома. Тем не менее существует возможность смонтировать подтом независимо от своего родителя, указав в командной строке опцию монтирования **subvol**. Например, так.

```
$ mkdir /sub
$ sudo mount LABEL=demo -o subvol=/sub /sub
$ ls /sub
file_in_a_subvolume
```

Невозможно предотвратить появление подтома в его родительском томе при монтировании родителя. Чтобы создать иллюзию нескольких независимых, невзаимодействующих томов, просто сделайте их подчиненными корнями и смонтируйте каждый из них отдельно с опцией **subvol**. Сам корневой каталог не требуется монтировать нигде. Фактически система Btrfs позволяет указать том, отличный от корня, который должен быть целевым объектом монтирования по умолчанию, если не запрошен подтом; см. команду **btrfs subvolume set-default**.

Чтобы увидеть или обработать всю иерархию Btrfs в этой конфигурации, просто смонтируйте корень в пустой каталог с помощью команды **subvol=/**. Это обычная практика, которые нужно монтировать несколько раз и которые доступны через несколько путей.

Снимки тома

Версия моментальных снимков Btrfs работает так же, как и команда **cp**, за исключением того, что копии являются поверхностными и изначально сохраняются в хранилище родительского тома.

```
$ sudo btrfs subvolume snapshot /mnt/demo/sub /mnt/demo/sub_snap
Create a snapshot of '/mnt/demo/sub' in '/mnt/demo/sub_snap'
```

В отличие от моментальных снимков ZFS, моментальные снимки Btrfs доступны для записи по умолчанию. На самом деле в файловой системе Btrfs нет такой вещи, как моментальный снимок в чистом виде; моментальный снимок — это просто том, который совместно использует некоторое хранилище с другим томом.

```
$ sudo touch /mnt/demo/sub/another_file
$ ls /mnt/demo/sub
another_file file_in_a_subvolume
$ ls /mnt/demo/sub_snap
file_in_a_subvolume
```

Для создания неизменяемого моментального снимка просто передайте параметр **-r** команде **btrfs subvolume snapshot**. Система Btrfs не проводит принципиального различия между моментальными снимками, доступными только для чтения, и записываемыми копиями, как это делает система ZFS. (В ZFS записываемые копии являются клонами. Чтобы создать такой клон, сначала создается снимок только для чтения, а затем клон на основе этого моментального снимка.) Система Btrfs не применяет какие-либо конкретные соглашения об именах или местоположении, когда дело доходит до определения подтомов и моментальных снимков, поэтому вам решать, как эти сущности должны быть организованы и названы. Документация по файловой системе Btrfs на сайте `btrfs.wiki.kernel.org` предлагает несколько вариантов на выбор.

В системе Btrfs также нет операции отката, которая восстанавливает состояние тома в соответствии с конкретным моментальным снимком. Вместо этого можно просто переместить исходный том в другое место, а затем выполнить команду **mv** или скопировать снимок на свое место:

```
$ ls /mnt/demo/sub
another_file file_in_a_subvolume
$ sudo mv /mnt/demo/sub /mnt/demo/sub.old
$ sudo btrfs subvolume snapshot /mnt/demo/sub_snap /mnt/demo/sub
Create a snapshot of '/mnt/demo/sub_snap' in '/mnt/demo/sub'
$ ls /mnt/demo/sub
file_in_a_subvolume
```

Обратите внимание на то, что это изменение мешает прямому монтированию подтома. После этого его нужно будет перемонтировать.

Поверхностные копии

Аналогия между моментальными снимками Btrfs и командой **cp** больше, чем просто совпадение. Невозможно создавать моментальные снимки в чистом виде для файлов или каталогов, которые не являются корнями подтома. Но, что интересно, можно создавать поверхностные копии произвольных файлов и каталогов с помощью команды **cp --reflink**, даже пересекая границы подтома.

Эта опция активизирует специфичный для системы Btrfs механизм команды **cp**, которая напрямую связывается с файловой системой, чтобы организовать дублирование записи с копированием. Ее семантика идентична семантике обычной команды **cp** и также очень напоминает семантику моментального снимка.

Система Btrfs не отслеживает поверхностные копии автоматически, как это было бы с моментальными снимками, и, следовательно, не гарантирует идеальную согласованность по времени активно изменяемых иерархий каталогов. Однако в остальном две операции заметно похожи. Одна приятная особенность поверхностных копий заключается в том, что они не требуют специальных разрешений; ими может воспользоваться любой пользователь.

Если вы укажете параметр команды **cp** в форме **--reflink=auto**, команда **cp** при возможности выполнит поверхностное копирование, а в противном случае будет вести себя как обычно. Это делает ее заманчивой целью для псевдонима в файле **~/.bashrc**:

```
alias cp="cp --reflink=auto"
```

20.14. Стратегия резервного копирования данных

В благополучных условиях основной задачей управления дисковой памятью является обеспечение хорошей производительности и достаточного свободного пространства. К сожалению, такие условия выполняются не всегда. По исследованиям Google Labs, у диска меньше 75% шансов выжить в течение пяти лет, поэтому всегда создавайте системы для защиты ценных данных от катастрофических потерь и будьте готовы активизировать процедуру восстановления данных в кратчайшие сроки.

Массивы RAID и другие схемы репликации данных защищают от сбоя одного объекта или части оборудования. Однако есть много других вариантов потерять данные, которые эти технологии не предотвращают. Например, если у вас возникло нарушение системы безопасности или произошло заражение с помощью вредоносного программного обеспечения, ваши данные могут быть изменены или повреждены, даже если физический уровень остается совершенно неповрежденным. Автоматическая репликация скомпрометированных данных на несколько дисков или сайтов только усилит ваши страдания. Вам нужны надежные резервные копии критических данных, которые можно использовать в качестве резервного варианта.

В последние десятилетия популярным методом хранения автономных резервных копий были такие носители, как магнитные ленты. Однако емкость этих носителей оказалась неспособной идти в ногу с экспоненциально растущими размерами жестких дисков и твердотельных накопителей. Наряду с физическими проблемами транспортировки и хранения лент и поддержания в рабочем состоянии капризных механических ленточных накопителей, проблемы с объемом в конечном итоге оттеснили ленточные носители до статуса 35-миллиметровой пленочной фотокамеры: она по-прежнему формально находится на рынке, но вы должны задаться вопросом, кто на самом деле ее покупает.

Сегодня большинство облачных платформ позволяют создавать мгновенные резервные копии в виде снимков, обычно в автоматическом режиме. Внося ежемесячную плату за память, занимаемую каждым снимком, вы можете устанавливать свои собственные правила хранения.

Независимо от конкретной технологии, которую вы используете для реализации резервных копий, вам нужен письменный план, который отвечает хотя бы на следующие вопросы.

Общая стратегия

- Какие данные необходимо скопировать?
- Какая система или технология будут выполнять резервное копирование?
- Где будут храниться данные резервного копирования?
- Будут ли зашифрованы резервные копии? Если да, где будут храниться ключи шифрования?
- Сколько будет стоить хранение резервных копий с течением времени?

Хронологическая стратегия

- Как часто будут выполняться резервные копии?
- Как часто будут проверяться резервные копии и выполняться их тест по восстановлению данных?
- Как долго будут сохраняться резервные копии?

Персонал

- Кто будет иметь доступ к данным резервного копирования?
- Кто будет иметь доступ к ключам шифрования, которые защищают данные резервного копирования?
- Кто будет отвечать за проверку выполнения резервных копий?
- Кто будет отвечать за подтверждение факта резервного копирования и их тестирования?

Использование и защита

- Как можно получить доступ к резервным данным или восстановить их в чрезвычайной ситуации?
- Как убедиться, что ни хакер, ни вредоносный процесс не могут повредить, изменить или удалить резервные копии? (Как обеспечить неизменность данных?)
- Как защитить резервные данные от захвата конкурирующим провайдером облака, поставщиком оборудования или правительством?

Точные ответы на эти вопросы зависят, как минимум, от вида организации, типа данных, нормативов, технологической платформы и бюджета. Найдите время, чтобы составить план резервного копирования для вашей организации или просмотреть существующий план резервного копирования.

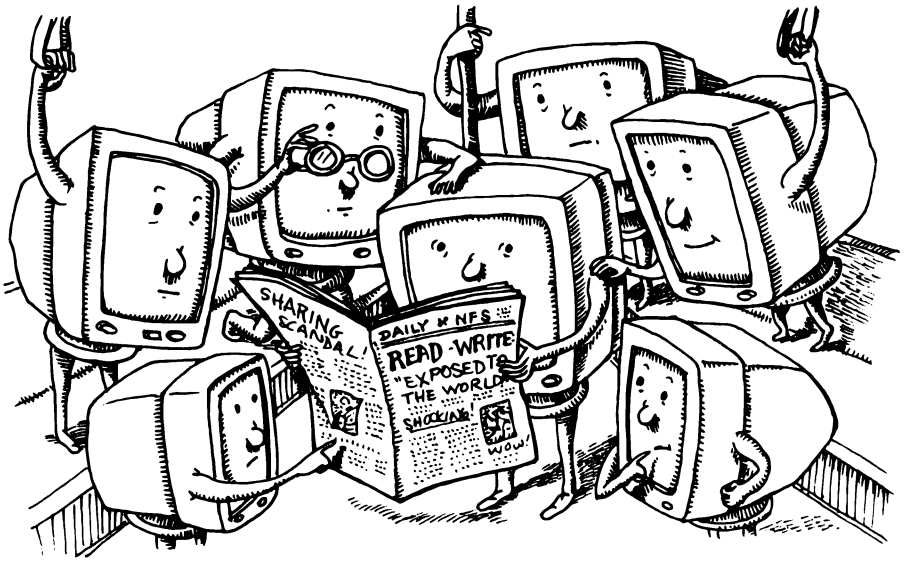
20.15. Литература

- Lucas, Michael W., and Allan Jude. *FreeBSD Mastery: ZFS*. Tilted Windmill Press, 2015.
- Jude, Allan, and Michael W. Lucas. *FreeBSD Mastery: Advanced ZFS*. Tilted Windmill Press, 2016.

  Две упомянутые выше книги — современные справочники по ZFS. Хотя они и ориентированы на специфику системы FreeBSD, большинство материалов относится также к ZFS в системе Linux. Книга *Advanced ZFS* особенно полезна освещением таких разнообразных тем, как механизм Jail, делегирование полномочий, стратегии кеширования и анализ производительности.
- Lucas, Michael W., and Allan Jude. *FreeBSD Mastery: Storage Essentials*. Tilted Windmill Press, 2015.
- McKusick, Marshall Kirk, George V. Neville-Neil, and Robert N. M. Watson. The *Design and Implementation of the FreeBSD Operating System (2nd Edition)*. Upper Saddle River, NJ: Addison-Wesley Professional, 2014. В этой книге рассматриваются различные темы, связанные с ядром, но в них содержатся также полные главы о UFS, ZFS и уровне VFS.

Глава 21

Сетевая файловая система NFS



Протокол сетевой файловой системы (Network File System, или NFS),, позволяет пользователям совместно работать с файлами, расположенными на разных компьютерах. Протокол NFS почти прозрачен для пользователей, т.е. при сбое сервера, поддерживающего протокол NFS, информация не пропадает. Клиенты просто ждут, когда сервер вновь начнет функционировать, а затем продолжают работать так, будто ничего не произошло.

Протокол NFS разработала компания Sun Microsystems в 1984 году. Первоначально протокол NFS был реализован как суррогат файловой системы для бездисковых клиентов, однако предложенный протокол оказался столь удачным, что со временем стал универсальным решением проблемы совместного использования файлов. Все дистрибутивные пакеты систем UNIX и Linux содержат версию протокола NFS; многие из них используют лицензию компании Sun. В настоящее время протокол NFS является открытым стандартом и описан в документах RFC (см. RFC 1094, RFC 1983 и особенно RFC 7530).

21.1. Введение в протокол NFS

Цель сетевой файловой службы — предоставить общий доступ к файлам и каталогам, которые хранятся на дисках удаленных систем. Пользовательские приложения должны иметь возможность читать и записывать эти файлы с помощью тех же системных вызовов, которые они используют для локальных файлов; файлы, хранящиеся в других местах сети, должны быть прозрачными для приложений. Если несколько сетевых клиентов или приложений пытаются одновременно изменить файл, служба совместного доступа к файлам должна разрешать возникающие конфликты.

Конкуренция

NFS — не единственная система совместного использования файлов. Существует протокол Server Message Block (SMB), встроенный в операционные системы Windows и macOS и предназначенный для этой же цели. Однако протокол SMB также может использоваться и в системах UNIX и Linux, если запустить дополнительный пакет Samba. Если вы запускаете гибридную сеть, которая включает в себя множество различных операционных систем, то можете обнаружить, что протокол SMB обеспечивает наилучшую совместимость.

🕮 Подробную информацию о протоколе SMB и пакете Samba см. в главе 22.

Протокол NFS чаще всего используется в средах, где преобладают UNIX и Linux. В этих ситуациях он предлагает несколько более естественную форму использования и более высокую степень интеграции. Но даже в этих условиях протокол SMB остается вполне законным вариантом. Для сетей, состоящих исключительно из систем на основе UNIX и Linux, довольно необычно использовать протокол SMB в качестве основного протокола совместного использования файлов (по крайне мере, мы об этом еще не слышали!).

Совместное использование файлов в сети выглядит простой задачей, но на самом деле она представляет собой сложную проблему, имеющую множество вариантов решения и нюансов. В качестве свидетельства сложности этой задачи напомним, что многочисленные ошибки протокола NFS проявились в необычных ситуациях только спустя четверть века его использования. Современные администраторы могут быть уверены в том, что протоколы совместного использования файлов (NFS и SMB) не портят данные и не вызывают ярость пользователей, но для того, чтобы этого достичь, пришлось немало потрудиться.

Сети хранения данных (storage area network — SAN) — еще один вариант высокопроизводительного управления хранением данных в сети. Серверы SAN не должны поддерживать никакие файловые системы, потому что они обрабатывают только запросы на операции с дисковыми блоками. И этим они отличаются от протоколов NFS и SMB, которые работают на уровне файловых систем и файлов, а не на уровне устройств хранения. SAN обеспечивает быстрый доступ для чтения и записи, но он не может управлять параллельным доступом нескольких клиентов без помощи кластерной файловой системы.

Для проектов с большими данными широко используются распределенные файловые системы с открытым исходным кодом. GlusterFS и Ceph реализуют как POSIX-совместимые файловые системы, так и хранилища объектов RESTful, распределенные между кластерами хостов для повышения отказоустойчивости. Коммерческие версии обеих систем продаются компанией Red Hat, которая приобрела обоих разработчиков. Обе системы представляют собой готовые к производству, высокопроизводительные файловые системы, заслуживающие рассмотрения в таких случаях, как обработка больших данных и высокопроизводительные вычисления.

Облачные системы имеют дополнительные возможности (см. раздел 9.3).

Проблемы, связанные с состоянием

При разработке файловой системы необходимо решить, какая ее часть будет отслеживать файлы, открываемые каждым клиентом. Информация об этих файлах называется *состоянием* (state). Сервер, не фиксирующий состояние файлов и клиентов, называется *сервером без сохранения состояния* (stateless), а сервер, который решает эту задачу, — *сервером с сохранением состояния* (stateful). Многие годы использовались оба этих подхода, причем каждый из них имеет как преимущества, так и недостатки.

Серверы с сохранением состояния отслеживают все открытые файлы в сети. Этот режим работы вызывает множество проблем (больше, чем можно было бы ожидать) и затрудняет восстановление работоспособности сети в случае аварии. Когда сервер восстанавливает свою работу, клиент и сервер должны заново согласовать, какое состояние следует считать последним перед аварией. Серверы с сохранением состояния позволяют клиентам лучше контролировать файлы и облегчают управление файлами, открытыми в режиме чтения-записи.

На сервере без сохранения состояния каждый запрос не зависит от предшествующих запросов. Если сервер или клиент терпит крах, то никаких потерь для процесса это не влечет. В этом случае сервер безболезненно терпит крах и перезагружается, поскольку никакого контекста нет. Однако в этом случае сервер не может знать, какие клиенты открыли файлы для записи, поэтому не способен управлять параллельной работой.

Проблемы производительности

Пользовательский интерфейс сетевых файловых систем не должен отличаться от пользовательского интерфейса локальных файловых систем. К сожалению, глобальные сети имеют большое время ожидания, что приводит к неправильному выполнению операций и падению скорости передачи данных. Все это в итоге приводит к снижению производительности работы с большими файлами. В большинстве протоколов файловых служб, включая NFS, реализованы методы минимизации потерь в производительности как в локальных, так и глобальных сетях.

В большинстве протоколов пытаются минимизировать количество запросов в сети. Например, стратегия упреждающего кеширования предусматривает загрузку фрагментов файла в локальный буфер памяти, чтобы избежать задержки при считывании нового раздела файла. Часть полосы пропускания сети используется для того, чтобы избежать задержки при двухстороннем обмене данными с сервером.

Кроме того, в некоторых системах операции записи данных кешируются в памяти, и в пакетах посылаются только их обновления, уменьшая тем самым задержку, вызванную необходимостью выполнить операции записи на сервере. Эти пакетные операции обычно называют *объединением запросов* (request coalescing).

Безопасность

Любая служба, предоставляющая удобный доступ к файлам в сети, является источником серьезных угроз для безопасности. Локальные файловые системы реализуют сложные алгоритмы управления доступом, наделяя пользователей разными правами. В сети эти проблемы многократно усложняются, поскольку существуют требования, предъявляемые к быстродействию машин, а также имеются различия между их конфигурациями, ошибки в программном обеспечении файловых систем и нерешенные вопросы, связанные с протоколами совместного использования файлов.

Развитие служб каталогов и централизованной аутентификации повысило безопасность сетевых файловых систем. По существу, ни один клиент не может аутентифицировать себя самостоятельно, поэтому необходима доверенная централизованная система, верифицирующая личности и предоставляющая им доступ к файлам. Большинство служб совместного использования файлов могут быть интегрированы с разнообразными провайдерами аутентификации.

Протоколы совместного использования файлов обычно не затрагивают проблемы конфиденциальности и целостности — или по крайней мере они не делают этого на-

прямую. Как и при аутентификации, эта ответственность обычно передается на другой уровень, например протоколам Kerberos, SSH или VPN-туннелю. В последних версиях протокола SMB добавлена поддержка надежного шифрования и проверки целостности данных. Однако во многих организациях, использующих протокол NFS в защищенной сети, обычно отказываются от этих средств, потому что они довольно громоздкие при реализации и замедляют работу сети.

21.2. Основные идеи, лежащие в основе протокола NFS

Новейшая версия протокола NFS является платформенно-независимой, обеспечивает высокую производительность в глобальных сетях, таких как Интернет, а также гарантирует высокую безопасность. Большинство его реализаций также содержат диагностические средства для решения проблем, связанных с настройкой и производительностью.

NFS — это сетевой протокол, поэтому теоретически он может быть реализован в пользовательском пространстве, как и большинство других сетевых служб. Тем не менее в рамках традиционного подхода протокол NFS реализован как на стороне сервера, так и на стороне клиента, чтобы работать внутри ядра, в основном для повышения производительности. Этот общий шаблон сохраняется даже в системе Linux, где функции блокировки и некоторые системные вызовы оказались трудными для экспорта в пространство пользователя. К счастью, части ядра NFS не нуждаются в конфигурации и в значительной степени прозрачны для администраторов.

NFS не является готовым решением всех проблем совместного доступа к файлам. Высокая доступность может быть достигнута только с помощью "горячего резерва", но протокол NFS не имеет встроенных средств синхронизации с резервными серверами. Внезапное исчезновение сервера NFS из сети может привести к тому, что клиенты будут хранить устаревшие дескрипторы файлов, которые могут быть удалены только при перезагрузке. При этом реализация средств повышенной безопасность возможна, но слишком сложна. Несмотря на эти недостатки, протокол NFS остается наиболее распространенным выбором для совместного использования файлов в локальной сети в системах UNIX и Linux.

Версии и история протокола

Первая открытая версия протокола NFS, появившаяся в 1989 году, имела номер 2. Оригинальный протокол имел несколько дорогостоящих компромиссов, отдавая предпочтение согласованности, а не производительности, и быстро был заменен. В настоящее время эта версия встречается крайне редко.

В версии 3, появившейся в начале 1990-х годов, этот недостаток был устранен за счет схемы согласования, которая позволяла выполнять запись асинхронно. Были улучшены также другие аспекты протокола, связанные с производительностью и обработкой больших файлов, вследствие чего версия NFS 3 стала работать гораздо быстрее, чем NFS 2.

Версия NFS 4, выпущенная в 2003 г., но получившая широкое распространение лишь к концу этого десятилетия, является результатом крупной переработки протокола; она содержит много усовершенствований и новых функциональных возможностей, перечисленных ниже.

- Совместимость и взаимодействие со всеми брандмауэрами и устройствами NAT.
- Интегрирование в основном протоколе NFS протоколов блокировки и монтирования.

- Поддержка операций с сохранением состояния.
- Высокая интегрированная безопасность.
- Поддержка репликации и миграции.
- Поддержка как UNIX-, так и Windows-клиентов.
- Списки управления доступом (ACL).
- Поддержка файлов в кодировке Unicode.
- Хорошая производительность даже при низкой скорости передачи данных по сети.

Разные версии протокола не совместимы друг с другом, но на файловых серверах, поддерживающих протокол NFS (включая все рассмотренные нами в книге операционные системы), обычно реализованы все три версии. На практике все клиенты и серверы протокола NFS могут взаимодействовать с помощью одной из этих версий. Если и клиентская, и серверная стороны поддерживают протокол NFS 4, следует использовать именно эту версию.

NFS продолжает активно развиваться и широко распространяться. Версия 4.2, написанная некоторыми из первоначальных заинтересованных сторон еще во времена расцвета Sun, достигла состояния чернового стандарта RFC в начале 2015 г. Служба Elastic File System от AWS, которая стала общедоступной в середине 2016 г., позволяет использовать файловые системы NFSv4.1 в экземплярах EC2.

Хотя версия 4 протокола NFS во многих отношениях и является значительным шагом вперед, в ней не сильно изменен процесс настройки и администрирования клиентов и серверов. В некотором смысле это ее особенность; например, до сих пор используются одни и те же файлы конфигурации и команды для администрирования всех версий протокола NFS. С другой стороны, это проблема; некоторые аспекты процесса настройки явно являются импровизацией (особенно в системе FreeBSD), а некоторые функции оказались неоднозначными или перегруженными, с разным смыслом или форматами файлов конфигурации в зависимости от того, какую версию NFS вы используете.

Удаленный вызов процедур

Когда компания Sun разработала первые версии протокола NFS в 1980-х гг., ее разработчики поняли, что многие связанные с сетью проблемы, которые необходимо решить с помощью протокола NFS, относятся и к другим сетевым службам. Они разработали более общую структуру для удаленного вызова процедур, известную как RPC (Remote Procedure Call) или SunRPC, и построили NFS на ее основе. Эта работа открыла возможности для приложений всех видов запускать процедуры на удаленных системах так, как если бы они выполнялись локально.

Система RPC от Sun была примитивной и несколько хакерской; сегодня существуют гораздо более совершенные системы для удовлетворения этой потребности.[1] Тем не менее в протоколе NFS по-прежнему используется технология RPC в стиле Sun в большей части его функциональности. Операции, которые читают и записывают файлы, монтируют файловые системы, получают доступ к метаданным файлов и разрешают доступ к файлам, все реализованы как удаленные вызовы процедур. Спецификация протокола NFS написана в общем виде, поэтому отдельный уровень RPC формально не требуется. Однако нам не известны какие-либо реализации NFS, которые отклонялись бы от исходной архитектуры в этом отношении.

[1] Как и бесконечно более ужасающие монстры, чем SunRPC, например, SOAP.

Транспортные протоколы

В версии NFS 2 применялся протокол UDP, поскольку он обеспечивал наилучшую скорость работы в локальных сетях 80-х годов. Несмотря на то что протокол NFS сам выполняет сборку пакетов и осуществляет контроль ошибок, ни в UDP, ни в NFS не реализованы алгоритмы управления перегрузкой, которые необходимы для достижения хорошей производительности в крупных IP-сетях.

Для решения этих проблем в большинстве UNIX-систем теперь разрешено использовать в качестве транспортного протокола в версии NFS 3 как UDP, так и TCP, а в версии NFS 4 — только TCP.[2] Первоначально эта возможность предназначалась для обеспечения работы NFS в сетях с маршрутизаторами и в Интернете. По мере удешевления оперативной памяти и роста производительности центральных процессоров и сетевых контроллеров первоначальное преимущество протокола UDP над протоколом TCP испарилось.

Состояние

Прежде чем начать работать с файловой системой NFS, клиент должен ее смонтировать, как если бы это была файловая система, находящаяся на локальном диске. Однако в версиях NFS 2 и 3 не сохраняется состояние, поэтому сервер не "знает", какие клиенты смонтировали ту или иную файловую систему. Вместо этого сервер вручает клиенту секретный ключ по факту успешного завершения операции монтирования. Этот ключ идентифицирует каталог монтирования на сервере NFS и дает клиенту возможность получить доступ к содержимому каталога. Ключ сохраняется при перезагрузке, чтобы сервер после сбоя смог вернуться в предыдущее состояние. Клиент может просто подождать, пока сервер перезагрузится, и повторить свой запрос.

С другой стороны, версия NFSv4 представляет собой протокол с сохранением состояния: и клиент, и сервер хранят информацию об открытых файлах и блокировках. При сбое сервера клиент облегчает процесс восстановления, посылая на сервер информацию о состоянии, предшествующем сбою. Восстанавливающийся сервер ожидает в течение заданного периода отсрочки, пока бывшие клиенты отчитаются о своем предыдущем состоянии, прежде чем приступить к выполнению новых операций и блокировок. Управления ключами, которое существовало в версиях V2 и V3, в версии NSFv4 больше нет.

Экспорт файловой системы

Говорят, что сервер "экспортирует" файловую систему (при этом поддерживается список экспортируемых каталогов), если он делает ее доступной для использования другими компьютерами. По определению, все серверы экспортируют хотя бы один каталог. Клиенты затем могут смонтировать эти экспортируемые каталоги и внести их в список файловых систем, хранящихся в их файле `fstab`.

🕮 Дополнительную информацию о файле `fstab` см. в разделе 20.10.

В версиях 2 и 3 каждый экспортируемый каталог рассматривается как независимая сущность. В версии 4 каждый сервер экспортирует одну иерархическую псевдофайловую систему, содержащую все свои экспортируемые каталоги. По существу, псевдофайловая система — это пространство имен файловой системы сервера, из которой удалено все, что не подлежит экспорту.

[2] С формальной точки зрения можно использовать любой транспортный протокол, реализующий алгоритм управления перегрузкой, но в настоящее время единственным разумным выбором является протокол TCP.

В качестве примера рассмотрим следующий список каталогов, в котором экспортируемые каталоги выделены полужирным шрифтом.

/www/domain1
/www/domain2
/www/domain3
/var/logs/httpd
/var/spool

В версии NFS 3 каждый экспортируемый каталог должен конфигурироваться отдельно. Клиентские системы должны выполнить три разных запроса на монтирование, чтобы получить доступ ко всем экспортируемым каталогам сервера.

В то же время в версии NFS 4 псевдофайловая система соединяет разрозненные части структуры каталогов и создает единое представление для клиентов NFS. Вместо запроса на монтирование каждого из каталогов **/www/domain1**, **/www/domain2** и **/var/logs/httpd**, клиент может просто смонтировать серверный псевдокорневой каталог и просмотреть всю иерархию.

Каталоги **/www/domain3** и **/var/spool**, не подлежащие экспорту, не появляются в этой иерархии. Кроме того, отдельные файлы, содержащиеся в каталогах **/**, **/var**, **/www** и **/var/logs**, не видны клиенту, потому что псевдофайловая часть иерархии состоит только из указанных каталогов. Таким образом, при использовании версии NFSv4 клиент видит экспортируемую файловую систему в следующем виде.

```
/
├─ var
│   └─ logs
│       └─ httpd
└─ www

    ├─ domain1
    └─ domain2
```

На сервере корень экспортированных файловых систем определяется в файле конфигурации **exports**, обычно хранящемся в каталоге **/etc**. Обычные клиенты NFSv4 не могут просматривать список точек монтирования на удаленном сервере. Вместо этого они просто монтируют псевдо-корневую файловую систему, а затем все экспортируемые каталоги становятся доступным через эту точку монтирования.

Так это выглядит с точки зрения спецификации RFC. Однако на практике ситуация несколько размыта. С одной стороны, реализация системы Solaris соответствовала этой спецификации. С другой стороны, разработчики операционной системы Linux сделали половинчатую попытку поддержки псевдофайловой системы в раннем коде NFSv4, но позже переработали ее, чтобы полностью поддерживать схему; сегодняшняя версия, похоже, учитывает цели RFC. Система FreeBSD не реализует псевдофайловую систему, как описано в спецификации RFC. Семантика экспорта FreeBSD по существу такая же, как и в версии 3; все подкаталоги внутри экспортируемых доступны для клиентов.

Блокировка файлов

Блокировка файлов (реализуемая системными вызовами **flock**, **lockf** или **fcntl**) в течение долгого времени была слабым звеном в системах UNIX. В локальных файловых системах этот механизм был воплощен далеко не идеально. В ранних версиях протокола NFS серверы не сохраняли состояние: им не известно, какие компьютеры работают с конкретным файлом. Однако эта информация необходима для использования блокировок. Что же делать?

Традиционное решение заключается в реализации механизма файловых блокировок отдельно от протокола NFS. В большинстве систем этой цели служат два демона: **lockd** и **statd**. К сожалению, по ряду причин они работают не оптимально, и блокировка файлов в протоколе NFS оставалась его слабым местом.

В версии NFSv4 демоны **lockd** и **statd** больше не используются для блокировки в основном протоколе, поскольку серверы сохраняют состояние. Это изменение значительно усложнило протокол, но устранило множество проблем, характерных для более ранних версий протокола NFS. К сожалению, по отдельности демоны **lockd** и **statd** все еще нужны для поддержки клиентов, использующих версии 2 и 3. Все рассматриваемые нами примеры операционных систем поставляются с ранними версиями протокола NFS, поэтому отдельные демоны по-прежнему запускаются по умолчанию.

Вопросы безопасности

Во многих аспектах версии 2 и 3 протокола NFS были типичными наследниками всех недостатков, касающихся вопросов безопасности в системах UNIX и Linux. Этот протокол изначально не предполагал никаких мер безопасности, и благодаря этому он был удобным. Версия NFSv4 устранила дефекты системы безопасности, которые были присущи предыдущим версиям, обеспечив сильную поддержку службам безопасности и внедрив более совершенную идентификацию пользователей.

Все версии протокола NFS не зависят от механизма, обеспечивающего безопасность работы в сети, и большинство серверов поддерживают разные режимы аутентификации. Некоторые из этих режимов перечислены ниже.

- `AUTH_NONE` — без аутентификации.
- `AUTH_SYS` — способ управления доступом для пользователей и групп, напоминающий стиль системы UNIX.
- `RPCSEC_GSS` — строгий режим аутентификации, предполагающий целостность и закрытость в дополнение к аутентификации.

Традиционно в большинстве организаций применяется режим `AUTH_SYS`, который использует идентификаторы групп и пользователей в системе UNIX. В этой схеме при запросе к серверу клиент просто посылает локальный идентификатор пользователя и группы. Сервер сравнивает значения этих идентификаторов со значениями из своего файла **/etc/passwd**[3] и определяет, следует ли предоставлять доступ данному пользователю. Таким образом, если два пользователя имеют одинаковые идентификаторы на двух разных клиентах, то они получат доступ ко всем файлам друг друга. Более того, пользователи, имеющие права администратора системы, могут с помощью команды **su** установить любой идентификатор пользователя по своему усмотрению; после этого сервер предоставит им доступ к соответствующим файлам.

Согласование файла **passwd** со всеми системами играет очень существенную роль в средах, использующих режим `AUTH_SYS`. Однако даже это — всего лишь иллюзия безопасности; любой мошеннический хост (или Windows-машина) может "аутентифицировать" своих пользователей по своему желанию и тем самым разрушить безопасность протокола NFS.

Для того чтобы предотвратить эти проблемы, в большинстве организаций используется более надежный механизм идентификации, такой как протокол Kerberos в сочетании со слоем NFS `RPSSEC_GSS`. Эта конфигурация требует от клиента и сервера совместного участия в работе механизма Kerberos. Механизм Kerberos аутентифицирует клиентов централизованно, тем самым предотвращая возможность самоидентифика-

[3] Или его эквивалента в виде сетевой базы данных, такой как NIS или LDAP.

ции, описанной выше. Кроме того, протокол Kerberos обеспечивает сильное шифрование и гарантирует целостность файлов, передаваемых по сети. Все системы, поддерживающие протокол NFS версии 4, должны реализовать режим RPCSEC_GSS, в то же время в версии 3 это требование не является строгим.

🕮 Дополнительную информацию о протоколе Kerberos см. в разделе 27.6.

Протокол NFSv4 использует в качестве транспортного протокола только TCP и обычно осуществляет связь через порт 2049. Поскольку в протоколе NFSv4 не используются другие порты, то чтобы открыть для него доступ из вне нужно в брандмауэре открыть доступ к порту 2049 протокола TCP. Как и в случае конфигурации всех списков управления доступом, кроме порта, следует указать адреса отправителя и получателя. Если ваша организация не планирует предоставлять услуги протокола NFS хостам, расположенным в Интернете, заблокируйте доступ к нему в брандмауэре или используйте локальный фильтр пакетов.

🕮 Дополнительную информацию о брандмауэрах см. в разделе 27.8.

Не рекомендуется использовать файловую службу в глобальных сетях с версиями протокола NFSv2 и 3 из-за длительной истории ошибок в протоколах RPC и отсутствия сильных механизмов безопасности. Администраторы серверов NFS версии 3 должны блокировать доступ к портам TCP и UDP 2049, а также порту 111 службы **portmap**.

Учитывая множество очевидных недостатков безопасности режима AUTH_SYS, мы настоятельно рекомендуем прекратить использование протокола NFSv3. Если у вас есть древние операционные системы, которые не могут быть обновлены до совместимости с NFSv4, по крайней мере следует применять фильтры пакетов для ограничения сетевого подключения.

Идентифицирующее отображение в версии 4

Прежде чем приступать к следующему обсуждению, должны предупредить: мы считаем, что все реализации безопасности AUTH_SYS более или менее уязвимы. Мы настоятельно рекомендуем аутентификацию Kerberos и RPCSEC_GSS; это единственный разумный выбор.

Как указывалось в главе 8, операционная система UNIX идентифицирует пользователей с помощью набора индивидуальных и групповых идентификаторов, записанных в локальном файле **passwd** или административной базе данных. С другой стороны, версия 4 протокола NFS представляет пользователей и группы в виде строковых идентификаторов, имеющих вид *пользователь@nfs-домен* или *группа@nfs-домен*. И клиенты, и серверы, работающие по протоколу NFS, запускают демон идентифицирующего отображения (identity mapping daemon), который преобразовывает значения идентификаторов пользователей UNIX в строки, соответствующие приведенному выше формату.

Когда клиент, использующий версию 4, выполняет операцию, возвращающую идентификаторы пользователей, например команду **ls -l**, выводящую листинг файлов, демон идентифицирующего отображения использует свой локальный файл **passwd** для преобразования пользовательских и групповых идентификаторов каждого файлового объекта в строку, например ben@admin.com. Затем отображение идентификаторов клиентов выполняет обратное действие, превращая строку ben@admin.com в локальные пользовательские и групповые идентификаторы, которые могут совпадать, а могут и не совпадать с идентификаторами сервера. Если строковое значение не соответствует ни одной локальной сущности, используется учетная запись анонимного пользователя.

В этот момент вызов удаленной файловой системы (**stat**) завершается и возвращает вызвавшей ее команде (в данном случае команде **ls**) значения пользовательского и груп-

пового идентификаторов. Однако, поскольку команда **ls** была выполнена с флагом **-l**, она должна вывести на экран текстовые имена, а не числа. Итак, команда **ls**, в свою очередь, переводит идентификаторы обратно в текстовые имена, используя библиотечные утилиты **getpwuid** и **getgrgid**. Эти утилиты еще раз проверяют файл **passwd** или эквивалентную ему базу данных. Какая долгая и запутанная процедура!

К сожалению, идентифицирующее отображение используется только при извлечении и записи атрибутов файлов, как правило, о праве владения. *Идентифицирующее отображение не играет никакой роли в процессе аутентификации и управлении доступом.* Оно лишь переводит идентификаторы в форму, принятую в протоколе RPC. Идентифицирующее отображение может оказаться полезнее, чем базовый протокол NFS, потому что оно позволяет выявлять конфликт между разрешениями на доступ к файлам и реальными разрешениями, выданными сервером NFS.

В качестве примера рассмотрим следующие команды клиента протокола NFSv4.

```
[ben@nfs-client]$ id ben
uid=1000(ben) gid=1000(ben) groups=1000(ben)

[ben@nfs-client]$ id john
uid=1010(john) gid=1010(john) groups=1010(john)

[ben@nfs-client]$ ls -ld ben
drwxr-xr-x  2   john   root   4096 May 27 16:42   ben

[ben@nfs-client]$ touch ben/file
[ben@nfs-client]$ ls -l ben/file
-rw-rw-r--  1   john   nfsnobody     0     May 27 17:07     ben/file
```

Мы видим, что пользователь `ben` имеет идентификатор 1000, а `john` — 1010. Рабочий каталог **ben**, экспортированный по протоколу NFS, имеет разрешение `775` и принадлежит пользователю `john`. Однако пользователь `ben` может создать файл в этом каталоге, даже если вывод команды **ls -l** означает, что у него нет прав на запись.

На сервере пользователь `john` имеет идентификатор 1000. Поскольку на клиентском компьютере идентификатор пользователя `john` равен 1010, идентифицирующее отображение выполняет преобразование идентификатора, как описано выше, и в результате пользователь `john` оказывается владельцем каталога. Однако демон идентифицирующего отображения не участвует в управлении доступом. При создании файла непосредственно на сервер посылается идентификатор пользователя `ben`, равный 1000, который интерпретируется как идентификатор пользователя `john`, поэтому права доступа предоставляются

Как узнать, какие операции используют идентифицирующее отображение, а какие нет? Это просто: как только пользовательский или групповой идентификатор используется в *вызовах API файловой системы* (например, как в **stat** или **chown**), он отображается. Если пользовательские или групповые идентификаторы *неявно* используются для управления доступом, они проходят через специальную систему аутентификации.

По этой причине принудительное согласование файлов **passwd** или использование LDAP важно для пользователей "безопасного" режима `AUTH_SYS`.

К сожалению для администраторов, демоны идентифицирующего отображения не стандартизованы, поэтому их процессы конфигурирования на разных системах могут быть разными. Специфика каждой системы описывается в разделе 21.5.

Учетные записи root и nobody

В общем случае пользователи должны иметь одинаковые привилегии в любой системе, к которой они получают доступ. Однако традиционно системные администраторы стара-

ются ограничивать возможности неконтролируемого применения прав привилегированного пользователя в файловых системах, смонтированных посредством NFS. По умолчанию сервер NFS перехватывает входящие запросы, посылаемые от имени пользователя с идентификатором 0, и "делает вид", будто они поступают от другого пользователя. Этот прием называется "поражением в правах" ("squashing root"). Таким образом, учетная запись **root** не отменяется, но уравнивается в правах с обычными учетными записями.

Для этого специально определена фиктивная учетная запись **nobody**, чтобы под нее "маскировался" пользователь **root**, работающий на сервере NFS. Традиционно ее идентификатор равен 65534 (16-разрядное значение числа –2, представленное в дополнительном коде)[4]. Значения UID и GID для пользователя **root** в файле **exports** можно изменить. С помощью опции `all_squash` можно заставить систему преобразовать все клиентские идентификаторы пользователей в одинаковый серверный идентификатор. В таком случае будут отменены все различия между пользователями и создан вариант открытой файловой системы, доступной всем.

Все эти предосторожности преследуют благородную цель, однако конечный результат далек от идеала. Даже будучи клиентом NFS, пользователь **root** может с помощью команды **su** "принять облик" любого пользователя, так что файлы никогда не бывают полностью защищены. Единственный эффект от смены идентификаторов заключается в том, что предотвращается доступ к файлам, которые принадлежат пользователю **root** и недоступны для чтения или записи остальным пользователям.

Производительность версии 4

Протокол NFSv4 был разработан для того, чтобы обеспечить высокую производительность в глобальных сетях. Большинство глобальных сетей имеет более высокую задержку и меньшую скорость передачи данных, чем локальные сети. Протокол NFS должен был решить эти проблемы с помощью следующих усовершенствований.

- Процедура протокола RPC под названием `COMPOUND` включает несколько файловых операций в один запрос, сокращая время ожидания при выполнении многочисленных сетевых запросов.
- Механизм делегирования поддерживает кеширование файлов на клиентской стороне. Клиенты могут сохранять контроль над локальными копиями файлов, включая их открытие для записи.

Эти функциональные возможности являются частью ядра протокола NFS и не требуют специального внимания со стороны системного администратора.

21.3. Серверная часть протокола NFS

Говорят, что сервер NFS "экспортирует" каталог, когда делает этот каталог доступным для использования другими компьютерами. Экспортированные каталоги представляются клиентам NFSv4 как единая иерархия файловой системы через псевдофайловую систему.

В версии NFSv3 процесс, используемый клиентами для монтирования файловой системы, отделен от процесса, используемого для доступа к файлам. Эти операции используют отдельные протоколы, а запросы обрабатываются разными демонами:

[4]На сервере NFS в системе Red Hat стандартное значение UID равно –2, тогда как в файле **passwd** учетной записи `nobody` присвоен идентификатор 99. Можно оставить все как есть, но добавить в файл **passwd** запись с идентификатором –2 или задать параметры `anonuid` и `anongid` равными 99. Это действительно все равно! В некоторых системах присутствует также учетная запись **nfsnobody**.

mountd — для запросов на монтирование и **nfsd** — для реальной файловой службы. В некоторых системах эти демоны называются **rpc.nfsd** и **rpc.mountd**, поскольку основаны на протоколе RPC, а значит, для их запуска нужен демон **portmap**. В этой главе для простоты мы будем пропускать префикс **rpc**.

В версии NFSv4 демон **mountd** не используется вообще. Однако, если не все ваши клиенты используют версию NFSv4, демон **mountd** следует запустить.

На сервере NFS демоны **mountd** и **nfsd** должны запускаться при загрузке системы и работать на протяжении всего времени ее функционирования. И в системе Linux и во FreeBSD эти демоны запускаются автоматически, если служба NFS была активизирована.

В протоколе NFS используется единая база данных для управления доступом, которая определяет, какие файловые системы должны быть экспортированы и какие клиенты могут их монтировать. Оперативная копия этой базы данных обычно хранится в файле **xtab**, а также во внутренних таблицах ядра. Файл **xtab** — это бинарный файл, с которым работает демон сервера.

Монтирование бинарного файла вручную — неблагодарная задача, поэтому в большинстве систем предполагается, что пользователь должен поддерживать текстовый файл (как правило, он называется **/etc/exports**), в котором указываются все экспортируемые системные каталоги и права доступа к ним. Система считывает этот текстовый файл при загрузке и автоматически создает файл **xtab**.

В большинстве систем каталог **/etc/exports** является каноническим списком, доступным для чтения человеком, в котором перечисляются все экспортируемые каталоги. В системе Linux его содержимое перечитывается демоном после выполнения команды **exportfs -a**, а в системе FreeBSD — после перезапуска сервера NFS. Следовательно, после внесения изменений в файл **/etc/exports**, необходимо выполнить команду **exportfs -a**, чтобы изменения вступили в силу в системе Linux, или выполнить команду **service nfsd restart** в системе FreeBSD. Если вы обслуживаете клиентов версии 3 в системе FreeBSD, перезапустите также демон **mountd** с помощью команды **service mountd reload**.

Протокол NFS работает на логическом уровне файловой системы. Любой каталог можно экспортировать; он не обязан быть точкой монтирования или корнем физической файловой системы. Однако с точки зрения безопасности протокол NFS различает границы между файловыми системами и требует, чтобы каждое устройство было смонтировано отдельно. Например, на компьютере, имеющем отдельный раздел **/chinchim/users**, можно было бы экспортировать каталог **/chinchim** без экспорта каталога **/chinchim/users**.

Клиентам обычно разрешается монтировать подкаталоги экспортируемых каталогов, если это необходимо, хотя протокол этого не требует. Например, если сервер экспортирует каталог **/chimchim/users**, то клиент может смонтировать только каталог **/chinchim/users/joe** и игнорировать остальную часть каталога **users**.

Файл exports в системе Linux



В системе Linux файл **exports** содержит список экспортированных каталогов в левом столбце и список связанных с ними параметров в правом. Список файловых систем отделен от списка клиентов пробелом, и после имени каждого клиента в скобках стоит список параметров, разделенных запятыми. Продолжение строки обозначается обратной косой чертой. Например, строка

```
/home    *.users.admin.com(rw)    172.17.0.0/24(ro)
```

позволяет смонтировать каталог **/home** для чтения и записи на всех машинах в домене users.admin.com, и только для чтения на всех машинах в сети 172.17.0.0/24 класса C. Если система в домене users.admin.com находится в сети 172.17.0.0/24, ее клиент получает доступ только для чтения. Действует правило наименьшей привилегии.

Файловые системы, перечисленные в файле **exports** без указания конкретных хостов, монтируются на *всех* компьютерах, что является значительной брешью в системе безопасности.

Такую же брешь можно создать, случайно неправильно расставив пробелы. Например, строка

```
/home   *.users.admin.com (rw)
```

предоставляет доступ для чтения и записи любому хосту, *за исключением* *.users.admin.com, которые по умолчанию имеют доступ только для чтения. Ой!

К сожалению, нет способа указать несколько спецификаций клиентов для одного набора параметров. Вы должны повторить параметры для всех желаемых клиентов. В табл. 21.1 перечислены типы спецификаций клиента, которые могут указываться в файле **exports**.

Таблица 21.1. Спецификации клиента в файле /etc/exports в системе Linux

Тип	Синтаксис	Описание
Имя хоста	*имя_хоста*	Отдельные хосты
Сетевая группа	*@имя_группы*	Сетевые группы NIS (используется редко)
Шаблонный символ	* и ?	Полностью определенные имена доменов (FQDN)[a] с шаблонными символами, причем символ "*" не соответствует точке
IPv4-сеть	*ip_адрес/маска*	Спецификации в стиле CIDR (например, 128.138.92.128/25)
IPv6-сеть	*ip_адрес/маска*	Адреса IPv6 с системе обозначений CIDR (2001:db8::/32)

[a] FQDN — Fully qualified domain names.

В табл. 21.2 показаны наиболее часто используемые параметры экспорта, используемые в системе Linux.

Параметр subtree_check (установлен по умолчанию) проверяет, что каждый файл, к которому обращается клиент, находится в экспортированном подкаталоге. Если вы отключите этот параметр, будет проверен только тот факт, что файл находится в экспортируемой файловой системе. Проверка поддеревьев может вызвать неожиданные проблемы, если запрашиваемый файл переименовывается в тот момент, когда он открыт на клиенте. Если вы ожидаете, что такие ситуации будут возникать часто, рассмотрите возможность установить параметр no_subtree_check.

Параметр async предписывает серверу NFS игнорировать спецификацию протокола и отвечать на запросы до их записи на диск. Это может привести к небольшому повышению производительности, но также может привести к повреждению данных, если сервер неожиданно перезапустится. Значение по умолчанию — это sync, т.е. полная синхронизация действий сервера.

Параметр replicas — это просто удобная возможность, которая помогает клиентам обнаруживать зеркала, если основной сервер вдруг выходит из строя. Фактическая репликация файловой системы должна быть выполнена автономно с помощью какого-то другого механизма, например **rsync** или DRBD (программное обеспечение репликации для Linux). Возможность репликации была добавлена в версию NFSv4.1.

Таблица 21.2. Основные параметры экспорта файловых систем в Linux

Опция	Описание
`ro`	Экспорт только для чтения
`rw`	Экспорт для чтения и записи (по умолчанию)
`rw=`*список*	Экспорт преимущественно для чтения; в *списке* перечисляются хосты, которым разрешено монтировать файловую систему для записи; остальные должны монтировать ее только для чтения
`root_squash`	Отображает ("поражает в правах") идентификаторы UID 0 и GID 0 в значения, заданные параметрами `anonuid` и `anongid`. Этот параметр задается по умолчанию
`no_root_squash`	Разрешает привилегированный доступ для всех. Опасно!
`all_squash`	Отображает все идентификаторы UID и GID в значения, установленные для анонимных пользователей.[a]
`anonuid=`*xxx*	Задает идентификатор UID для удаленных привилегированных пользователей, которых следует лишить привилегий
`anongid=`*xxx*	Задает идентификатор GID для удаленных привилегированных групп пользователей, которых следует лишить привилегий
`noaccess`	Блокирует доступ к данному каталогу и подкаталогам (используется при вложенном экспорте)
`wdelay`	Откладывает запись в ожидании объединения обновлений
`no_wdelay`	Немедленно записывает данные на диск
`async`	Заставляет сервер отвечать на запросы до того, как запись на диск будет выполнена в действительности
`nohide`	Выявляет файловые системы, смонтированные в дереве экспортированных файлов
`hide`	Противоположен по смыслу параметру `nohide`
`subtree_check`	Проверяет, находится ли запрашиваемый файл в экспортированном дереве
`no_subtree_check`	Проверяет, относится ли запрашиваемый файл к экспортированной файловой системе
`secure_locks`	Требует авторизации для всех запросов на блокировку
`insecure_locks`	Использует менее строгие критерии блокировки (поддерживает старых клиентов)
`sec=`*режим*	Задает список методов обеспечения безопасности для экспортируемого каталога. Допускаются значения `sys` (аутентификация в системе UNIX), `dh` (DES), `krb5` (аутентификация по протоколу Kerberos), `krb5i` (аутентификация и защита целостности по протоколу Kerberos), `krb5p` (аутентификация, также защита целостности и конфиденциальности по протоколу Kerberos) и `none` (анонимный доступ; не рекомендуется)
`fsid=`*num*	Задает псевдофайловую систему в версии 4 (обычно 0)
`pnfs`	Разрешает параллельные расширения NFS в версии V4.1 для обеспечения прямого доступа клиентов
`replicas=`*путь@хост*	Посылает клиентам список альтернативных сервером для этого экспорта

[a] Эта опция полезна для поддержки персональных компьютеров и других ненадежных однопользовательских хостов.

В ранних версиях реализации протокола NFSv4 в системе Linux требовалось, чтобы администраторы назначали корень псевдофайловой системы в файле **`/etc/exports`** с помощью флага `fsid=0`. Этого больше не требуется. Чтобы создать псевдофайловую

систему, как описано в спецификации RFC, просто перечислите экспортируемые каталоги как обычно, а из клиента NFSv4 смонтируйте каталог / на сервере. При этом все подкаталоги, находящиеся ниже точки монтирования, будут экспортироваться как файловые системы. Если вы укажете для экспорта опцию `fsid=0`, эта файловая система и все ее подкаталоги будут экспортированы для клиентов V4.

Файл `exports` в системе FreeBSD



В соответствии с давней традицией системы UNIX формат файла **`exports`**, используемый в системе FreeBSD, полностью отличается от формата Linux. Каждая строка в файле (за исключением строк, начинающихся с #, которые представляют собой комментарии) состоит из трех компонентов: списка каталогов для экспорта, параметров для применения к этому экспорту и набора хостов, к которым применяется экспорт. Как и в системе Linux, обратная косая черта означает продолжение строки.

```
/var/www -ro,alldirs www*.admin.com
```

Строка, приведенная выше, экспортирует каталог **`/var/www`** и все его подкаталоги в режиме только для чтения на все хосты, соответствующие шаблону `www*.admin.com`. Чтобы реализовать различные варианты монтирования для разных клиентов, просто скопируйте эту строку и укажите другие значения. Например, строка

```
/var/www -alldirs,sec=krb5p -network 2001:db8::/32
```

открывает доступ для чтения и записи всем хостам в указанной сети IPv6. Для аутентификации, проверки целостности и обеспечения конфиденциальности используется протокол Kerberos.

В системе FreeBSD экспортируемые каталоги должны находиться в пределах файловой системы сервера. При экспорте нескольких каталогов из одной и той же файловой системы для одного и того же набора клиентских хостов, они должны быть перечислены в одной и той же строке. Например,

```
/var/www1 /var/www2 -ro,alldirs www*.admin.com
```

Ошибка возникнет, если указать `www1` и `www2` в разных строках, но для одинакового набора хостов, если при этом каталоги `www1` и `www2` находятся в одной и той же файловой системе.

Чтобы подключить NFSv4, перед именем корня необходимо поставить префикс `V4:`, например,

```
V4: /exports -sec=krb5p,krb5i,krb5,sys -network *.admin.com
```

Допускается только один корневой путь V4. Однако он может быть указан несколько раз с различными вариантами для разных клиентов. Корень может появляться в любом месте файла **`exports`**.

Строка, содержащая префикс `V4:`, фактически не экспортирует никакие файловые системы. Она просто выбирает базовый каталог для NFSv4-клиентов для монтирования. Чтобы активировать ее, укажите экспортируемые каталоги в корневом каталоге:

```
/exports/www -network *.admin.com
```

Несмотря на то что для корня указана версия V4, сервер NFS в системе FreeBSD не реализует псевдофайловую систему, как описано в спецификации RFC. Когда назначается корень V4 и под этим корнем существует по крайней мере один экспортируемый каталог, клиент V4 может смонтировать корневой каталог и получить доступ ко всем

файлам и каталогам внутри него независимо от их опций экспортирования. На справочной странице **exports(5)** эта информация изложена неясно, и такая двусмысленность может быть весьма опасной. Не назначайте локальную корневую файловую систему сервера (/) в качестве корня V4; в противном случае вся корневая файловая система сервера будет доступна для клиентов.

При наличии корня V4, для клиентов V2 и V3 должны быть указаны другие пути для монтирования файлов, которые отличаются от тех, что используют клиенты V4. Например, рассмотрим следующий фрагмент экспортируемых каталогов:

```
/exports/www -network 10.0.0.0 -mask 255.255.255.0
V4: /exports -network 10.0.0.0 -mask 255.255.255.0
```

Здесь клиенты V2 или V3 в сети 10.0.0.0/24 могут смонтировать каталог **/exports/www**. Однако так как каталог **/exports** принадлежит корневой псевдофайловой системе, клиент V4 должен монтировать экспортируемый каталог как **/www**. В качестве альтернативы клиент V4 может смонтировать каталог / и обращаться к каталогу **www** под этой точкой монтирования.

Для обеспечения максимальной производительности при экспорте на большое количество клиентов используйте сетевые диапазоны. Для протокола IPv4 можно использовать форму записи CIDR или маску подсети. Для протокола IPv6 *необходимо* использовать CIDR; опция `-mask` не разрешена. Например:

```
/var/www -network 10.0.0.0 -mask 255.255.255.0
/var/www -network 10.0.0.0/24
/var/www -network 2001:db8::/32
```

В системе FreeBSD меньше опций для экспорта каталогов, чем в Linux. Их варианты перечислены в табл. 21.3.

Таблица 21.3. Основные варианты экспорта в системе FreeBSD

Вариант	Описание
`alldirs`	Позволяет монтировать разделы в любой точке файловой системы
`ro`	Экспорт только для чтения (чтение-запись по умолчанию)
`o`	Синоним для `ro`; экспортировать только для чтения
`maproot=`*xxx*	Имя пользователя или идентификатор UID для отображения удаленного пользователя `root`
`mapall=`*xxx*	Отображает всех пользователей клиента на указанного пользователя (например, `maproot`)
`sec=`*режим*	Задает допустимые режимы безопасности[a]

[a] Укажите несколько вариантов режимов безопасности в списке, разделенном запятыми, в порядке предпочтения. Возможными значениями являются `sys` (аутентификация UNIX, значение по умолчанию), `krb5` (аутентификация Kerberos), `krb5i` (аутентификация и проверка целостности Kerberos), `krb5p` (аутентификация Kerberos, проверка целостности и обеспечение конфиденциальности) и `none` (анонимный доступ, не рекомендуется).

Демон nfsd: обслуживание файлов

После того как демон **mountd** убедился в правильности клиентского запроса на монтирование, клиенту разрешается запрашивать различные операции над файлами. Эти запросы обрабатываются на сервере демоном **nfsd**[5]. Он не должен выполняться на ком-

[5] Демон **nfsd** всего лишь выполняет не предусматривающий возврата системный вызов сервера NFS, код которого встроен в ядро.

пьютере-клиенте NFS. Единственное исключение — когда клиент сам экспортирует файловые системы.

У демона **nfsd** не предусмотрен файл конфигурации. Все его параметры указываются в командной строке. Для запуска и остановки сервера **nfsd** следует использовать стандартные системные процедуры. В системе Linux, где запущен демон **systemd**, используется команда **systemctl**, а в FreeBSD — команда **service**. В табл. 21.4 указаны системные файлы и параметры, подлежащие изменению, если нужно изменить аргументы, передаваемые демону **nfsd**.

Чтобы изменения в конфигурации демона **nfsd** вступили в силу в системах семейства Linux, следует выполнить команду **systemctl restart nfs-config.service nfs-server.service**. В системе FreeBSD используются команды **service nfsd restart** и **service mountd restart**.

Параметр `-N` команды **nfsd** отключает указанную версию протокола NFS. Например, чтобы отключить версии протокола 2 и 3, добавьте в соответствующий файл конфигурации, указанный в табл. 21.4, опции `-N 2 -N 3` и перезапустите службу. Это хорошая идея, если вы уверены, что вам не нужна поддержка старых клиентов.

Таблица 21.4. Системные файлы конфигурации для настройки параметров демона nfsd

Система	Файл конфигурации	Параметр
Ubuntu	**/etc/default/nfs-kernel-server**	RPCNFSDCOUNT[a]
Red Hat	**/etc/sysconfig/nfs**	RPCNFSDARGS
FreeBSD	**/etc/rc.conf**	Параметры командной строки демона **nfsd**

[a] В некоторых версиях пакета **nfs-kernel-server** неправильно предлагалось изменить параметр RPCMOUNTDOPTS, чтобы задать некоторые параметры демона **nfsd**. Не дайте себя обмануть!

Демону **nfsd** передается числовой аргумент, определяющий, сколько экземпляров самого себя он должен создать посредством системного вызова **fork**. Правильный выбор числа процессов очень важен, но, к сожалению, это из области черной магии. Если это число будет слишком мало или слишком велико, скорость работы сервера NFS может снизиться.

Оптимальное количество выполняющихся демонов **nfsd** зависит от операционной системы и используемого аппаратного обеспечения. Если вы замечаете, что команда **ps** частенько стала отображать демон **nfsd** в состоянии `D` (непрерываемый сон) и при этом наблюдается простой некоторого количества центральных процессоров, попробуйте увеличить число потоков. Если при добавлении количества демонов **nfsd** происходит увеличение средней загрузки системы (о чем сообщит команда **uptime**), значит, вы слишком увлеклись и следует вернуться к прежнему состоянию.

Регулярно запускайте команду **nfsstat**, чтобы вовремя распознать проблемы производительности, которые могут быть связаны с количеством потоков **nfsd**. Более подробная информация о команде **nfsstat** представлена в разделе 21.6.



В системе FreeBSD параметры **-minthreads** и **-maxthreads** команды **nfsd** обеспечивают автоматическое управление количеством потоков в указанных границах. Для выяснения дополнительных настроек сервера NFS откройте справочную страницу **rc.conf** в системе FreeBSD и найдите описание параметров с префиксом `nfs_`.

21.4. Клиентская часть протокола NFS

Процессы монтирования сетевых и локальных файловых систем во многом схожи. Команда **mount** понимает запись вида *имя_хоста:каталог* как путь к каталогу, расположенному на указанном компьютере. Этому каталогу будет поставлен в соответствие каталог локальной файловой системы. По завершении монтирования доступ к сетевой файловой системе осуществляется традиционными средствами. Таким образом, команда **mount** и ее NFS-расширения — самое важное для системного администратора на NFS-клиенте.

Для того чтобы файловую систему NFS можно было монтировать, ее необходимо соответствующим образом экспортировать (об этом рассказывалось выше). Клиентская команда **showmount** позволяет клиенту проверить, правильно ли сервер экспортирует файловые системы.

```
$ showmount -e monk
Export list for monk:
/home/ben harp.atrust.com
```

Как следует из этого примера, каталог **/home/ben** на сервере `monk` экспортирован в клиентскую систему `harp.atrast.com`.

Если сетевая файловая система по какой-то причине не работает, возможно, вы просто забыли выполнить команду **exportfs -a** (в системе Linux) либо **service nfsd restart** или **service mounts reload** (в системе FreeBSD). Проверьте после этого вывод команды **showmount**.

Если каталог был правильно экспортирован на сервере, а команда **showmount** сообщает об ошибке или возвращает пустой список, внимательно проверьте, все ли необходимые процессы запущены на сервере (**portmap** и **nfsd**, а также **mountd**, **statd** и **lockd** для версии 3), разрешен ли в файлах **hosts.allow** и **hosts.deny** доступ к этим демонам и, вообще, та ли это клиентская система.

Информация о пути, отображаемая командой **showmount**, например **/home/ben**, как в предыдущем случае, является корректной только для версий протокола NFS 2 и 3. Серверы, работающие в рамках протокола NFS 4, экспортируют целостную и единообразную псевдофайловую систему. Традиционная концепция протокола NFS, относящаяся к разным точкам монтирования, в версии 4 не работает, поэтому команду **showmount** применять нельзя.

К сожалению, достойной альтернативы команде **showmount** в версии NFS 4 нет. На сервере команда **exportfs -v** демонстрирует существующие экспортированные файловые системы, но она работает только на в рамках локального компьютера. Если у вас нет прямого доступа к серверу, смонтируйте корень псевдофайловой системы сервера и пройдите по структуре каталогов вручную. Можно также смонтировать любой из подкаталогов экспортируемой файловой системы.

Реальное монтирование сетевой файловой системы для версий 2 и 3 осуществляется примерно такой командой.

```
$ sudo mount -o rw,hard,intr,bg server:/home/ben /nfs/ben
```

Для того чтобы сделать то же самое в версии 4 под управлением системы Linux, выполните следующую команду.

```
$ sudo mount -o rw,hard,intr,bg server:/ /nfs/ben
```

В данном случае указанные после опции **-o** флаги говорят о том, что файловая система монтируется в режиме чтения-записи (**rw**), файловые операции разрешается прерывать (**intr**), а повторные попытки монтирования должны выполняться в фоновом режиме (**bg**). Наиболее распространенные флаги команды **mount** в системе Linux приведены в табл. 21.5.

Таблица 21.5. Флаги команды `mount` в системе Linux

Флаг	Назначение
`rw`	Монтирование файловой системы для чтения-записи (она должна экспортироваться сервером в режиме чтения-записи)
`ro`	Монтирование файловой системы только для чтения
`bg`	Если смонтировать файловую систему не удается (сервер не отвечает), следует перевести операцию в фоновый режим и продолжить обработку других запросов на монтирование
`hard`	Если сервер отключился, операции, которые пытаются получить к нему доступ, блокируются до тех пор, пока сервер не включится вновь
`soft`	Если сервер отключился, операции, которые пытаются получить к нему доступ, завершаются выдачей сообщения об ошибке. Этот флаг полезно устанавливать для того, чтобы предотвратить зависание процессов в случае неудачного монтирования не очень важных файловых систем
`intr`	Позволяет прерывать с клавиатуры заблокированные операции (будут выдаваться сообщения об ошибке)
`nointr`	Не позволяет прерывать с клавиатуры заблокированные операции
`retrans=`*n*	Указывает, сколько раз нужно повторить запрос, прежде чем будет выдано сообщение об ошибке (для файловых систем, смонтированных с флагом `soft`)
`timeo=`*n*	Задает интервал тайм-аута для запросов (в десятых долях секунды)
`rsize=`*n*	Задает размер буфера чтения равным *n* байт
`wsize=`*n*	Задает размер буфера записи равным *n* байт
`sec=`*режим*	Задает режим безопасности
`nfsvers=`*n*	Задает версию протокола NFS
`proto=`*протокол*	Выбирает транспортный протокол; им должен быть протокол `tcp` для версии NVS 4

Клиентская сторона NFS обычно пытается автоматически согласовать подходящую версию протокола. Вы можете указать конкретную версию, передав параметры **`-o nfsvers=`***n*.

В системе FreeBSD команда **`mount`** — это оболочка, которая вызывает команду **`/sbin/mount_nfs`** для монтирования NFS. Эта оболочка устанавливает параметры NFS и осуществляет системный вызов **`nmount`**. Чтобы смонтировать файловую систему с сервера NFS версии 4 в системе FreeBSD, выполните команду:

```
$ sudo mount -t nfs -o nfsv4 server:/ /mnt
```

Если вы явно не укажете версию, оболочка **`mount`** автоматически согласует ее в порядке убывания. По сути можно использовать простую команду **`mount server:/ /mnt`**, потому что оболочка **`mount`** может определить по формату указанных параметров командной строки, что файловой системой является NFS.

Файловые системы, смонтированные с флагом `hard` (установка по умолчанию), могут вызывать зависание процессов при отключении сервера. Это особенно неприятно, когда такими процессами оказываются стандартные демоны. Вот почему не рекомендуется запускать ключевые системные программы через NFS.[6] В общем случае использо-

[6] Джефф Форис (Jeff Forys), один из наших рецензентов, заметил по этому поводу: "Большинство сетевых файловых систем должно монтироваться с флагами *hard*, *intr* и *bg*, поскольку они наилучшим образом соответствуют исходной концепции NFS (надежность и отсутствие информации о состоянии). Флаг *soft* — это мерзкий сатанинский трюк! Если пользователь пожелает прервать операцию монтирования, пусть он сделает это сам. В противном случае следует дождаться включения сервера, и все в конце концов нормализуется без какой бы то ни было потери данных".

вание флага intr позволяет сократить число проблем, связанных с NFS. Исправить некоторые недостатки монтирования позволяют средства автоматического монтирования, например программа **autofs** (описывается далее).

Размеры буферов чтения и записи устанавливаются на максимально возможном уровне, согласованном с клиентом и сервером. Рекомендуется устанавливать их между 1 КиБ и 1 МиБ.

Определить размер доступного дискового пространства на смонтированной сетевой файловой системе можно с помощью команды **df**, как если бы это была обычная локальная файловая система.

```
$ df /nfs/ben
Filesystem          1k-blocks    Used   Available   Use%    Mounted on
leopard:/home/ben   17212156   1694128   14643692    11%      /nfs/ben
```

Разделы NFS размонтируются командой **umount**. Если сетевая файловая система кем-то используется в момент размонтирования, будет выдано сообщение об ошибке, показанное ниже.

```
umount: /nfs/ben: device is busy[7]
```

Найдите с помощью команд **fuser** или **lsof** процессы, у которых есть открытые файлы в этой файловой системе, и уничтожьте эти процессы либо смените текущие каталоги в случае командных оболочек. Если ничего не помогает или сервер отключен, воспользуйтесь командой **umount -f** для принудительного размонтирования файловой системы.

Монтирование файловых систем NFS на этапе начальной загрузки

С помощью команды **mount** можно создавать лишь временные сетевые точки монтирования. Файловые системы, которые являются частью постоянной конфигурации, должны быть указаны в файле **/etc/fstab**, для того чтобы они автоматически монтировались на этапе начальной загрузки. С другой стороны, они могут обрабатываться утилитой автоматического монтирования, например программой **autofs**.

🕮 Дополнительную информацию о файле **fstab** см. в разделе 20.10.

Следующие элементы файла **fstab** предназначены для монтирования файловой системы **/home**, расположенной на компьютере **monk**.

```
# filesystem    mountpoint  fstype  flags                          dump  fsck
monk:/home      /nfs/home   nfs     rw,bg,intr,hard,nodev,nosuid   0     0
```

Выполнив команду **mount -a -t nfs**, можно сделать так, чтобы изменения вступили в силу немедленно (без перезагрузки).

Параметры монтирования файловой системы NFS задает поле flags в файле **fstab**; это те же самые параметры, которые необходимо указывать в команде **mount**.

Ограничения экспорта привилегированными портами

Клиентам NFS разрешается использовать любой TCP- или UDP-порт для подключения к серверу NFS. Однако некоторые серверы могут требовать, чтобы запросы поступали из привилегированного порта (номер которого меньше 1024). Другие серверы позволяют выбирать порт по своему усмотрению. Впрочем, применение привилегированных портов не приводит к реальному повышению безопасности системы.

[7] Устройство занято.

Тем не менее большинство клиентов протокола NFS следуют традиционному (и по-прежнему рекомендуемому) подходу: по умолчанию выбирается привилегированный порт, что позволяет избежать ненужных конфликтов. Для того чтобы разрешить запросы на монтирование, поступающие из непривилегированных портов, в системе Linux можно использовать параметр `insecure`.

21.5. Идентифицирующее отображение в протоколе NFS 4

Ранее мы уже изложили общие идеи, касающиеся идентифицирующего отображения в протоколе NFSv4. В этом разделе мы обсудим административные аспекты демона идентифицирующего отображения.

Все системы, участвующие в работе системы по протоколу NFSv4, должны иметь одинаковые домены NFS. В большинстве случаев в качестве домена NFS целесообразно использовать свой домен DNS. Например, естественно выбрать `admin.com` в качестве домена NFS для сервера `ulsah.admin.com`. Клиенты в поддоменах (например, `books.admin.com`) могут при желании использовать тот же самый домен (например, `admin.com`) для реализации взаимодействия в рамках протокола NFS.

К сожалению для администраторов, стандартной реализации отображения идентификаторов UID в протоколе NFSv4 не существует. В табл. 21.6 перечислены демоны идентифицирующего отображения в каждой из систем и указано местонахождение их конфигурационного файла.

Таблица 21.6. Демоны идентифицирующего отображения и их конфигурации

Система	Демон	Файл конфигурации	Справочная страница
Linux	**`/usr/sbin/rpc.idmapd`**	**`/etc/idmapid.conf`**	**`nfsidmap(5)`**
FreeBSD	**`/usr/sbin/nfsuserd`**	`nfsuserd_flags` в файле **`/etc/rc.conf`**	**`idmap(8)`**

Кроме указания доменов NFS, для реализации идентифицирующего отображения необходима дополнительная помощь системного администратора. Демон запускается в момент загрузки системы в том же самом сценарии, который управляет системой NFS. После внесения изменения в файл конфигурации необходимо вновь перезапустить демон. Параметры, определяющие, например, детализацию вывода в системный журнал и альтернативное управление анонимной учетной записью, обычно являются доступными.

21.6. Команда `nfsstat`: отображение статистики NFS

Команда **`nfsstat`** отображает различные статистические данные, накапливаемые в системе NFS. Команда **`nfsstat -s`** выдает статистику процессов сервера NFS, а команда **`nfsstat -c`** отображает информацию, касающуюся клиентских операций. По умолчанию команда **`nfsstat`** отображает статистику для всех версий протокола NFS.

```
$ nfsstat -c
Client rpc:
  calls  badcalls  retrans    badxid    timeout   wait  newscred   timers
  64235      1595        0         3       1592      0         0      886

Client nfs:
```

```
calls   badcalls   nclget    nslsleep
62613         3    62643           0
null    getattr   setattr  readlink  lookup  root      read
  0%        34%       0%       21%     30%    0%        2%
write   wrcache    create    remove  rename  link   symlink
   3%         0%        0%       0%      0%    0%        0%
mkdir    readdr     rmdir   fsstat
   0%         6%        0%       0%
```

Приведенные результаты получены на нормально функционирующем сервере NFS. Если более 3% вызовов терпят неудачу, то это говорит о наличии проблем на NFS-сервере или в сети. Как правило, причину можно выяснить, проверив значение параметра `badxid`. Если его значение близко к нулю, а количество тайм-аутов больше 3%, то пакеты, поступающие на сервер и отправляемые с него, теряются в сети. Решить эту проблему можно, уменьшив значение параметров монтирования `rsize` и `wsize` (размеры блоков для чтения и записи).

Если значение параметра `badxid` такое же большое, как и значение параметра `timeout`, то сервер отвечает, но слишком медленно. В этом случае необходимо либо заменить сервер, либо увеличить параметр `timeo` при монтировании.

Регулярное выполнение команд **`nfsstat`** и **`netstat`** и анализ выдаваемой ими информации помогут администратору выявлять возникающие в NFS проблемы раньше, чем с ними столкнутся пользователи.

21.7. Специализированные файловые серверы NFS

Быстрый и надежный файловый сервер — один из важнейших элементов вычислительной среды. Конечно, дешевле всего организовать файловый сервер на рабочей станции, воспользовавшись имеющимися жесткими дисками. Однако это не оптимальное решение с точки зрения производительности и удобства администрирования.

Уже много лет на рынке предлагаются специализированные файловые серверы NFS. У них есть ряд преимуществ по сравнению с "кустарными" системами.

- Они оптимизированы на выполнение операций с файлами и, как правило, обеспечивают наилучшую производительность при работе с системой NFS.
- По мере увеличения требований к хранилищу файлов можно легко наращивать мощности сервера, даже если придется обслуживать терабайтовые массивы данных и сотни пользователей.
- Они более надежны, чем обычные системы, благодаря усеченному набору программ, применению специализированных контроллеров, обеспечивающих избыточность и зеркальное дублированию информации на жестких дисках.
- Они обычно реализуют доступ к файлам для клиентов как UNIX, так и Windows, а иногда даже содержат встроенные веб-, FTP- и SFTP-серверы.
- Они обладают улучшенными средствами резервного копирования и контроля состояния, по сравнению с обычными UNIX-системами.

Одними из лучших по нашему мнению являются устройства NetApp. Диапазон предложений — от малых до очень крупных систем, и цены вполне приемлемы. Компания EMC занимает нишу устройств высшего класса. Она выпускает хорошие продукты, но их цены могут испугать кого угодно.

В среде AWS служба Elastic File System является масштабируемым сервером NFSv4.1, который экспортирует файловые системы в экземпляры EC2. Каждая файловая система

может поддерживать несколько потоков передачи данных по сети с гигабитной скоростью, в зависимости от размера файловой системы. Дополнительную информацию см. на сайте `aws.amazon.com/efs`.

21.8. Автоматическое монтирование

Индивидуальное монтирование файловых систем посредством упоминания их в файле **`/etc/fstab`** сопряжено в крупных сетях с рядом проблем. Во-первых, ведение файла **`fstab`** на нескольких сотнях компьютеров — весьма утомительная задача. Каждый из компьютеров может отличаться от других и требовать особого подхода. Во-вторых, если файловые системы монтируются с множества компьютеров, то в случае сбоя всего лишь одного из серверов наступает хаос, так как все команды, пытающиеся получить доступ к точкам монтирования этого сервера, зависают.

Решить эту проблему можно с помощью демона автоматического монтирования, который подключает файловые системы, когда к ним выполняется обращение, и отключает их, когда надобность в них отпадает. Этот процесс осуществляется незаметно для пользователей и позволяет свести к минимуму число активных точек монтирования. Демону можно также предоставить список реплицированных (содержащих идентичные данные) файловых систем, чтобы сеть могла функционировать в случае отказа основного сервера.

Как писал Эдвард Томаш Наперала (Edward Tomasz Napierała), разработчик инструмента для автоматического монтирования файловых системе в операционной системе FreeBSD, для решения проблемы требуется сотрудничество нескольких связанных частей программного обеспечения:

- драйвер **`autofs`** — резидентный драйвер файловой системы, который отслеживает запросы на монтирование файловых систем, приостанавливает вызывающую программу и вызывает инструмент автоматического монтирования для монтирования целевой файловой системы перед возвратом управления вызывающей программе;
- демоны **`automountd`** и **`autounmountd`**, которые считывают административную конфигурацию и фактически монтируют или размонтируют файловые системы;
- административная утилита **`automount`**.

Обычно инструмент автоматического монтирования работает прозрачно для пользователей. Вместо того чтобы зеркально дублировать в сети реальную файловую систему, программа автоматического монтирования воссоздает ее иерархию в соответствии со спецификациями, указанными в файле конфигурации. Когда пользователь ссылается на каталог виртуальной файловой системы с автоматическим монтированием, последний перехватывает эту ссылку, монтирует реальную файловую систему, к которой обращается пользователь, и передает ссылку дальше. На системах, поддерживающих драйвер **`autofs`**, файловая система NFS монтируется в файловой системе с автоматическим монтированием в обычном для системы UNIX режиме.

Идея автоматического монтирования была предложена компанией Sun. Имеющаяся в Linux программа автоматического монтирования имитирует работу одноименной утилиты компании Sun, хотя реализована независимо и обладает рядом отличительных особенностей. Начиная с версии 10.1 операционная система FreeBSD поддерживает новую реализацию, отказавшись от когда-то широко использовавшегося демона автоматического монтирования **`amd`**.

Разные реализации программы **automount** используют три вида файлов конфигурации, которые называются *таблицами* (maps): таблицы прямых назначений, таблицы косвенных назначений и главные таблицы.[8] Таблицы прямых и косвенных назначений содержат информацию о файловых системах, подлежащих автоматическому монтированию. В главной таблице указываются таблицы прямых и косвенных назначений, которые должна учесть программа **automount**. В каждый момент времени может быть активной только одна главная таблица; по умолчанию главная таблица хранится в каталоге **/etc/auto_master** в системе FreeBSD и **/etc/auto.master** — в системе Linux.

В большинстве систем программа **automount** представляет собой отдельную команду, которая считывает свои файлы конфигурации, настраивает все необходимые для программы **autofs** параметры и прекращает работу. Действительные ссылки на файловые системы, подлежащие автоматическому монтированию, обрабатываются (при участии программы **autofs**) другим процессом — демоном **automountd**. Этот демон выполняет свою работу незаметно и не требует дополнительной настройки.



В системах Linux этот демон называется **automountd**, а функция настройки выполняется в сценарии загрузки **/etc/init.d/autofs**. Подробная информация об этом приведена ниже. Далее мы будем называть команду настройки именем **automount**, а демон автоматического монтирования — **automountd**.

Если вы изменили главную таблицу или одну из таблиц прямых назначений, на которую она ссылается, то необходимо заново запустить программу **automount**, чтобы эти изменения вступили в силу. При использовании параметра **-v** команда **automount** демонстрирует изменения, внесенные в ее конфигурацию. Для достижения эффекта сухого запуска, который позволяет вам анализировать проблемы конфигурации и отладки, можно добавить параметр **-L**.

Кроме того, программа **automount** (**autounmountd** в системе FreeBSD) имеет аргумент **-t**, сообщающий, как долго (в секундах) файловая система, монтируемая автоматически, может оставаться неиспользуемой перед тем, как будет размонтирована. По умолчанию этот параметр равен 300 с (10 мин.). Поскольку точка монтирования NFS, принадлежащая сбойному серверу, может вызвать зависание программы, целесообразно удалять автоматически смонтированные файловые системы, которые больше не используются. Кроме того, не следует задавать тайм-аут слишком долгим.[9]

Таблицы косвенных назначений

Эти таблицы используются для автоматического монтирования нескольких файловых систем в общем каталоге. Однако путь к каталогу задается в главном файле, а не в самой таблице. Например, таблица назначений для файловых систем может иметь следующий вид.

```
users   harp:/harp/users
devel   -soft harp:/harp/devel
info    -ro harp:/harp/info
```

В первом столбце указывается имя подкаталога, в котором будет смонтирована файловая система. В следующих столбцах перечислены опции монтирования и приведен ис-

[8]Таблицы прямых назначений могут поддерживаться с помощью базы данных NIS или сервером каталогов LDAP, но это сложно.

[9]С другой стороны, монтирование файловой системы занимает определенное время. Система отвечает быстрее и плавнее, если файловые системы не монтируются непрерывно.

ходный путь к файловой системе. В данном примере (файл **/etc/auto.harp**) программе **automount** сообщается о том, что она может монтировать каталоги **/harp/users**, **/harp/devel** и **/harp/info** с компьютера **harp**, при этом каталог **info** монтируется только для чтения, а каталог **devel** — в режиме `soft`.

В рассматриваемой конфигурации подкаталоги на компьютере **harp** и на локальном компьютере будут идентичными, хотя это вовсе не обязательно.

Таблицы прямых назначений

В рассматриваемых таблицах указываются файловые системы, не имеющие общего префикса, такие как **/usr/src** и **/cs/tools**. Таблица прямых назначений (например, **/etc/auto.direct**), указывающая, что обе эти файловые системы должны монтироваться автоматически, может выглядеть следующим образом.

```
/usr/src    harp:/usr/src
/cs/tools   -ro monk:/cs/tools
```

Поскольку у этих файловых систем нет общего родительского каталога, их монтирование должно реализовываться по отдельности. Эта конфигурация требует дополнительных расходов, но ее преимуществом является то, что точка монтирования и структура каталогов всегда остаются доступными таким командам, как **ls**. Применяя команду **ls** к каталогу, содержащему таблицы косвенных назначений, пользователи могут запутаться, потому что программа **automount** не показывает подкаталоги, пока не получит доступ к их содержимому (команда **ls** не заглядывает внутрь каталогов, смонтированных автоматически, поэтому она не может вызвать их монтирование).

Главные таблицы

Главный файл содержит список таблиц прямых и косвенных назначений, которые должна учитывать программа **automount**. Для каждой таблицы косвенных назначений указывается корневой каталог, используемый для монтирования, определенного в таблице.

Главная таблица, использующая таблицы прямых и косвенных назначений, упомянутых в предыдущих примерах, может выглядеть следующим образом.

```
# Directory     Map
/harp           /etc/auto.harp -proto=tcp
/-              /etc/auto.direct
```

В первом столбце указывается имя локального каталога для таблицы косвенных назначений или специальный токен `/-` для таблицы прямых назначений. Во втором столбце указывается файл, в котором должна храниться таблица. Может существовать несколько таблиц разного типа. Если параметр монтирования указывается в конце строки, то он по умолчанию применяется ко всем точкам монтирования, указанным в таблице. Администраторы системы Linux всегда должны указывать флаг монтирования `-fstype=nfs4` для серверов, использующих четвертую версию протокола NFS.



В большинстве систем параметры, заданные по умолчанию для элементов главной таблицы, не смешиваются с параметрами, заданными для таблиц прямых и косвенных назначений. Если элемент главной таблицы имеет свой собственный список параметров, то значения, заданные по умолчанию, игнорируются. Тем не менее в системе Linux эти наборы параметров смешиваются. Если один и тот же параметр встречается в двух местах, то значение элемента главной таблицы заменяет значение, заданное по умолчанию.

Исполняемые таблицы

Если файл, содержащий таблицу косвенных назначений, является исполняемым, то он считается сценарием (или программой), динамически генерирующим информацию об автоматическом монтировании. Демон не читает таблицу в текстовом виде, а запускает файл на выполнение, передавая ему аргумент ("ключ"), где указывается, к какому подкаталогу пользователь пытается получить доступ. Сценарий отвечает за вывод соответствующей записи таблицы. Если переданный ключ неверен, сценарий завершается, ничего не отображая на экране.

Такая методика очень удобна и позволяет компенсировать многочисленные недостатки довольно странной системы конфигурирования программы **automounter**. По сути, она дает возможность создать единый для всей организации конфигурационный файл произвольного формата. Можно написать несложный сценарий, декодирующий глобальные конфигурационные параметры на каждом компьютере. В состав некоторых систем входит удобный сценарий **/etc/auto.net**, которому в качестве ключа передается имя компьютера, а он монтирует все экспортируемые файловые системы этого компьютера.

Поскольку автоматические сценарии запускаются динамически по мере необходимости, нет необходимости распространять главный файл конфигурации после каждого изменения или предварительно преобразовывать его в формат **automounter**; Фактически глобальный файл конфигурации может постоянно храниться на сервере NFS.

Видимость программы automount

Когда вы выводите содержимое родительского каталога файловой системы, монтируемой автоматически, этот каталог оказывается пустым независимо от того, сколько там было автоматически смонтировано файловых систем. Файловые системы, смонтированные автоматически, невозможно просмотреть с помощью графического пользовательского браузера.

Рассмотрим пример.

```
$ ls /portal
$ ls /portal/photos
art_class_2010     florissant_1003          rmnp03
blizzard2008       frozen_dead_guy_Oct2009  rmnp_030806
boston021130       greenville.021129        steamboat2006
```

Файловая система **photos** прекрасно работает и автоматически монтируется в каталоге **/portal**. Для того чтобы получить к ней доступ, необходимо указать ее полное имя. Однако вывод содержимого каталога **/portal** не указывает на ее существование. Если бы вы смонтировали эту систему с помощью команды **fstab** или **mount**, то она ничем бы не отличалась от других каталогов и была бы видимой в родительском каталоге.

Для того чтобы увидеть файловые системы, смонтированные автоматически, используются символические ссылки на точки автоматического монтирования. Например, если **/automounts/photos** — это ссылка на каталог **/portal/photos**, то команда **ls** позволяет увидеть среди содержимого каталога **/automounts**, что каталог **photos** был смонтирован автоматически. Ссылки на каталог **/authomounts/photos** по-прежнему проходят через механизм автоматического монтирования и работают правильно.

К сожалению, эти символические ссылки требуют сопровождения и могут потерять согласованность с реальными точками автоматического монтирования, если они периодически не обновляются с помощью некоего сценария.

Реплицированные файловые системы и программа automount

В некоторых случаях файловая система, предназначенная только для чтения, например **/usr/share**, может оказаться идентичной на нескольких серверах одновременно. В этом случае мы можем сообщить программе **automount** о нескольких потенциальных источниках для этой файловой системы. Демон самостоятельно выберет источник файловой системы, исходя из того, какой сервер окажется ближайшим по номеру в данной сети, по версии протокола NFS, а также по времени ответа на первоначальный запрос.

Несмотря на то что программа **automount** не видит файловую систему и не следит за ее использованием, реплицированные файловые системы должны предназначаться только для чтения (например, **/usr/share** или **/usr/local/X11**). Программа **automount** не имеет возможности синхронизировать записи между серверами, поэтому реплицированные файловые системы, предназначенные для чтения/записи, имеют небольшое практические значение.

При желании вы можете назначить свои приоритеты, указав, какая реплика должна выбираться первой. Приоритет задается небольшим целым числом, причем, чем больше число, тем ниже приоритет. Наиболее подходящим является приоритет по умолчанию, который равен 0.

Файл **auto.direct**, определяющий файловые системы **/usr/man** и **/cs/tools** как реплицированные, может выглядеть следующим образом.

```
/usr/man    -ro harp:/usr/share/man monk(1):/usr/man
/cs/tools   -ro leopard,monk:/cs/tools
```

Обратите внимание на то, что имена серверов могут указываться вместе, если пути к их источникам совпадают. Значение `(1)` после имени `monk` в первой строке задает приоритет сервера по отношению к файловой системе **/usr/man**. Отсутствие такого значения после имени `harp` означает, что этот сервер имеет неявный приоритет, равный нулю.

Автоматическое монтирование (V3; все, кроме Linux)

Вместо перечисления всех возможных монтируемых файловых систем в таблице косвенных или прямых назначений, вы можете сообщить программе **automount** немного информации о правилах именования и предоставить ей возможность разбираться в них самостоятельно. Главным обстоятельством при этом является тот факт, что демон **mountd**, применяемый к удаленному серверу, может посылать запросы о том, какие файловые системы были смонтированы на этом сервере. В четвертой версии протокола NFS экспорт всегда обозначается символом /, что исключает необходимость в такой функциональной возможности.

Существует несколько способов "автоматического конфигурирования автоматического монтирования". Самый простой из них относится к типу монтирования с флагом `-hosts`. Если указать флаг `-hosts` в качестве имени таблицы в файле главной таблицы, то программа **automount** отобразит файловые системы, экспортированные на указанные хосты, в заданный каталог автоматического монтирования.

```
/net -hosts -nosuid,soft
```

Например, если сервер `harp` экспортировал файловую систему **/usr/share/man**, то каталог должен стать доступным для демона автоматического монтирования с помощью пути **/net/harp/usr/share/man**.

Реализация флага `-hosts` не подразумевает перечисления всех возможных хостов, с которых можно экспортировать монтируемую файловую систему; это просто практи-

чески невозможно. Вместо этого команда ждет ссылки на имена конкретных подкаталогов, а затем монтирует экспортированные системы с указанного хоста.

Аналогичного, но более тонкого эффекта можно достичь с помощью шаблонных символов "`*`" и "`&`" в таблице косвенных назначений. Кроме того, существует множество макросов, используемых вместе с таблицами и именем текущего хоста, типом архитектуры и т.д. Подробности можно найти на справочной странице **automount**(1M).

Специфика системы Linux



Реализация программы **automount** в системе Linux немного отличается от реализации компании Sun. Основные отличия касаются имен команд и файлов. В системе Linux **automount** — это демон, который на самом деле монтирует и размонтирует удаленные файловые системы. Он выполняет работу, которую демон **automountd** делает в других системах, и обычно не требует запуска вручную.

По умолчанию главная таблица записана в файле **/etc/auto.master**. Ее формат и формат таблиц косвенных назначений описаны выше. Однако документацию о них найти трудно. Формат главной таблицы описан на странице **auto.master**(5), а формат таблицы косвенных назначений — на странице **autofs**(5). Будьте осторожны, не перепутайте их со страницей **autofs**(8), которая документирует команду **autofs**. (Как сказано на одной из справочных страниц: "Документация оставляет желать лучшего".) Для того чтобы изменения вступили в силу, выполните команду **/etc/init.d/autofs/reload**, являющуюся эквивалентом команды **automount** из системы Solaris.

Реализация системы Linux не поддерживает принятый в Solaris флаг `-hosts` для "автоматического конфигурирования автоматического монтирования".

21.9. Литература

Различные спецификации RFC, касающиеся системы NFS, перечислены в табл. 21.7.

Таблица 21.7. Документы RFC, связанные с NFS

RFC	Название	Автор	Дата
1094	Network File System Protocol Specification	Sun Mircosystems	Май 1989
1813	NFS Version 3 Protocol Specification	B. Callaghan et al.	Июнь 1995
2623	NFS Version 2 and Version 3 Security Issues	M.Eisler	Июнь 1999
2624	NFS Version 4 Design Considerations	S.Shepler	Июнь 1999
3530	NFS Version 4 Protocol	S.Shepler et al.	Апрель 2003
5661	NFS Version 4 Minor Version 1 Protocol	S.Shepler et al.	Январь 2010
7862	NFS Version 4 Minor Version 2 Protocol	T. Haynes	Ноябрь 2016

Глава 22

Файловая система SMB



В главе 21 описывается самая популярная система для совместного использования файлов в системах UNIX и Linux. Однако UNIX-системам также необходимо обеспечивать совместное использование файлов с такими системами, как Windows, которые не поддерживают NFS. Введите SMB!

В начале 1980-х гг. Барри Фейгенбаум (Barry Feigenbaum) создал протокол BAF для обеспечения совместного доступа к файлам и ресурсам. Перед выпуском название было изменено с инициалов автора на Server Message Block (SMB). Протокол был быстро подхвачен компанией Microsoft и сообществом пользователей персональных компьютеров, поскольку он давал доступ к файлам на удаленных системах, практически не отличающийся от доступа к локальным файлам.

В 1996 г. компания Microsoft выпустила версию под названием Common Internet File System (CIFS), в основном в качестве маркетингового упражнения.[1] Протокол CIFS внедрил (часто ошибочные) изменения в исходный протокол SMB. В результате компания Microsoft выпустила SMB 2.0 в 2006 г., а затем SMB 3.0 в 2012 г. Хотя в отрасли широко распространено обращение к файлам SMB как CIFS, правда в том, что CIFS давно устарел; работает только SMB. Если вы работаете в однородной среде UNIX и Linux, эта глава, вероятно, не для вас. Но если вам нужен способ коллективного использования файлов пользователями систем UNIX и Windows, читайте дальше.

[1]Компания Sun Microsystems также вступила в игру в 1996 г., предложив протокол WebNFS, и компания Microsoft увидела возможность продавать протокол SMB с более удобной реализацией и названием.

22.1. Samba: сервер SMB для UNIX

Samba — популярный программный пакет, доступный под лицензией GNU Public License, который реализует серверную часть протокола SMB на хостах UNIX и Linux. Он был первоначально создан Эндрю Триджеллом (Andrew Tridgell), который первым дизассемблировал код протокола SMB и опубликовал его в 1992 г. Здесь мы сосредоточимся на версии Samba 4.

Пакет Samba хорошо поддерживается и активно развивается, расширяя свои функциональные возможности. Он предлагает стабильный, промышленный способ совместного использования файлов пользователями систем UNIX и Windows. Настоящая красота Samba заключается в том, что вы устанавливаете только один пакет на стороне сервера; на стороне Windows специальное программное обеспечение не требуется.

В мире Windows файловая система или каталог, доступная по сети, называется *общей папкой* (share). Это звучит немного странно для пользователей UNIX, но мы следуем этому соглашению при обращении к файловой системе SMB.

Хотя мы рассматриваем только совместное использование файлов в этой главе, Samba может также реализовать множество других кросс-платформенных служб, включая

- аутентификацию и авторизацию;
- сетевую печать;
- преобразование имен;
- анонсирование службы (просмотр ресурсов файлового сервера и общих принтеров).

Samba также может выполнять основные функции контроллера Windows Active Directory. Тем не менее эта конфигурация недостаточно надежная; мы подозреваем, что контроллер AD, вероятно, лучше всего реализовать с помощью серверов Windows.

Однако, конечно, важно, чтобы ваши UNIX- и Linux-системы были добавлены в домен AD как клиенты. Это соглашение позволяет обмениваться идентификационной информацией и информацией об аутентификации на всех сайтах. Дополнительную информацию см. в главе 17.

Аналогично мы не рекомендуем Samba в качестве сервера печати. В этом случае, вероятно, лучше всего использовать пакет CUPS. Дополнительную информацию о печати в системах UNIX и Linux с помощью CUPS см. в главе 12.

Большинство функций Samba реализованы двумя демонами, **smbd** и **nmbd**. Демон **smbd** реализует обслуживание файлов и заданий на печать, а также аутентификацию и авторизацию. Демон **nmbd** обслуживает другие основные компоненты SMB: преобразование имен и анонсирование служб.

В отличие от системы NFS, для которой требуется поддержка на уровне ядра, пакет Samba не требует каких-либо изменений драйверов или ядра и работает полностью как пользовательский процесс. Он подключается к сокету, используемому для запросов SMB, и ожидает поступления запроса от клиента на доступ к ресурсу. После аутентификации запроса демон **smbd** создает экземпляр самого себя и запускает его от имени пользователя, приславшего запрос. В результате все обычные права доступа к файлу (включая групповые права) остаются ненарушенными. Единственная специальная функциональная возможность, которую добавляет демон **smbd**, — это служба блокировки файлов, которая предоставляет системам Windows семантику блокировки, к которой они привыкли.

Если вам по-прежнему интересно, зачем использовать SMB, а не, скажем, более интегрированную в систему UNIX удаленную файловую систему, такую как NFS, то ответ однозначен. Почти все операционные системы поддерживают SMB на определенном уровне. В табл. 22.1 приведены некоторые основные различия между SMB и NFS.

Таблица 22.1. Сравнение протоколов SMB и NFS

SMB	NFS
Серверы и процессы пользовательского пространства	Сервер ядра с потоками
Процессы для каждого пользователя	Один сервер (один процесс) для всех клиентов
Использует базовую операционную систему для контроля доступа	Имеет собственную систему контроля доступа
Механизмы монтирования: обычно индивидуальные пользователи	Механизмы монтирования: обычно системы
Довольно хорошая производительность	Наилучшая производительность

Система NFS более подробно рассматривается в главе 21.

22.2. Инсталляция и конфигурации пакета Samba

Пакет Samba может работать во всех рассмотренных нами операционных системах. Более того, в большинстве дистрибутивов Linux он установлен по умолчанию. Исправления, документацию и другие файлы можно загрузить с сайта `samba.org`. Обязательно следует убедиться в том, что используются новейшие из доступных для данной системы пакеты Samba, потому что дефекты предыдущих версий чреваты потерей данных и возникновением проблем с безопасностью.

Если пакет Samba еще не установлен в вашей системе, вы можете установить его в системе FreeBSD с помощью команды **`pkg install samba48`**. В системах семейства Linux загрузите пакет **`samba-common`** через используемый вами менеджер пакетов.

Пакет Samba настраивается в файле **`/etc/samba/smb.conf`** (**`/usr/local/etc/smb4.conf`** в системе FreeBSD). В этом файле указываются каталоги для совместного использования, их права доступа и общие рабочие параметры Samba. Пакеты Linux достаточно удобны тем, что в них предоставляется хорошо документированный файл конфигурации Samba, который является хорошей отправной точкой для новых настроек.

Samba поставляется с разумными настройками по умолчанию для своих параметров конфигурации, и для большинства применений нужен только небольшой файл конфигурации. Выполните команду **`testparm -v`** для вывода списка всех параметров конфигурации Samba и значений, которые им в настоящий момент назначены. Этот список включает ваши настройки из файла **`smb.conf`** или **`smb4.conf`**, а также любые значения по умолчанию, которые вы не переопределили. Обратите внимание: после запуска пакет Samba проверяет свой файл конфигурации каждые несколько секунд и загрузит любые внесенные в него изменения — перезагрузка не требуется!

Наиболее распространенное использование пакета Samba — обмен файлами с клиентами Windows. Доступ к этим ресурсам должен быть аутентифицирован через учетную запись пользователя одним из двух вариантов.

В первом варианте используются локальные учетные записи, для которых определяются идентификатор пользователя и пароль, которые никак не связаны с другими учетными записями пользователей (например, с их логином на вход в домен). Второй вари-

ант объединяет аутентификацию Active Directory и, таким образом, совмещает учетные данные пользователей.

Совместное использование файлов с локальной аутентификацией

Самый простой способ аутентификации пользователей, которые хотят получить доступ к ресурсам Samba, — создать для них локальную учетную запись на сервере UNIX или Linux. Поскольку пароли Windows работают совсем иначе, чем пароли UNIX, пакет Samba не может контролировать доступ к общим ресурсам SMB с помощью существующих паролей учетных записей пользователей UNIX. Следовательно, для использования локальных учетных записей необходимо хранить (и поддерживать) отдельный хеш пароля SMB для каждого пользователя.

Иногда, однако, эта простота перевешивает удобство для пользователя, но тем не менее эта система аутентификации действительно проста. Ниже приведен пример файла **smb.conf**, в котором используется эта система.

```
[global]
workgroup = ulsah
security = user
netbios name = freebsd-book
```

Параметр `security = user` предписывает пакету Samba использовать локальные учетные записи UNIX. Убедитесь, что имя рабочей группы (`workgroup`) установлено в соответствии с вашей средой. Обычно это домен Active Directory, если вы находитесь в среде Windows. Если вы используете другую среду, то можете опустить эту настройку. Пакет Samba имеет собственную команду **smbpasswd** для настройки хешей паролей в стиле Windows. Например, ниже мы добавляем пользователя `tobi` и устанавливаем для него пароль.

```
$ sudo smbpasswd -a tobi
New SMB password: <пароль>
Retype new SMB password: <пароль>
```

Учетная запись UNIX должна существовать до того, как вы попытаетесь установить для нее свой пароль Samba. Пользователи могут изменять свой пароль Samba, запустив команду **smbpasswd** без каких-либо параметров.

```
$ smbpasswd
New SMB password: <пароль>
Retype new SMB password: <пароль>
```

В этом примере изменяется пароль Samba текущего пользователя на сервере Samba. К сожалению, пользователи системы Windows должны сначала зарегистрироваться в системной оболочке на сервере, чтобы изменить пароль своего общего доступа к SMB. Возможность удаленного входа в систему должна настраиваться отдельно, скорее всего, с помощью протокола SSH.

Совместное использование файлов с помощью учетных записей, прошедших аутентификацию Active Directory

Простой процесс сохранения отдельной базы данных аутентификации для совместного использования файлов с помощью команды **smbpasswd** кажется архаичным в сегодняшнем гиперинтегрированном мире. В большинстве случаев необходимо, чтобы

пользователи проходили аутентификацию через некоторые формы централизованных систем управления полномочиями, такие как Active Directory или LDAP.

Последние годы привели к большим успехам в этой области для операционных систем UNIX и Linux. Необходимые компоненты, включая службы каталогов, демон **sssd**[2], а также файл **nsswitch.conf** и библиотеки PAM, описаны в главе 17. После развертывания этих компонентов легко настроить пакет Samba для их использования.

Ниже приведен пример файла конфигурации **smb.conf** для среды, в которой Active Directory выполняет аутентификацию пользователя (с помощью демона **sssd**).

```
[global]
    workgroup = ulsah
    realm = ulsah.example.com
    security = ads
    dedicated keytab file = FILE:/etc/samba/samba.keytab[3]
    kerberos method = dedicated keytab
```

В этом случае параметр `realm` должен быть таким же, как локальное имя домена Active Directory. Параметры `dedicated keytab file` и `kerberos method` позволяют Samba корректно работать с реализацией Kerberos Active Directory.

Настройка общих ресурсов

После того как вы настроили общие настройки и аутентификацию Samba, можете указать в файле **smb.conf**, какие каталоги должны использоваться совместно через протокол SMB. Каждому ресурсу совместного использования, который вы открываете для общего доступа, необходимо поставить в соответствие собственный раздел в файле конфигурации. Имя раздела становится именем общего ресурса, которое анонсируется для клиентов SMB.

Вот пример:

```
[bookshare]
   path = /storage/bookshare
   read only = no
```

Здесь клиенты SMB видят монтируемый ресурс с именем `\\sambaserver\bookshare`. Это дает доступ к дереву файлов, расположенному в каталоге `/storage/bookshare` на сервере.

Совместное использование рабочих каталогов пользователей

Вы можете автоматически преобразовывать рабочие каталоги пользователей в отдельные SMB-ресурсы с помощью специального названия раздела `[homes]` в файле **smb.conf**:

```
[homes]
comment = Home Directories
browseable = no
valid users = %S
read only = no
```

[2] В прошлом для интеграции Active Directory и пакета Samba использовался демон **winbind**. В наши дни предпочтительным является демон **sssd**.

[3] Этот файл **keytab** создается демоном **sssd**, если вы его настроили в соответствии с инструкциями из главы 17. Дополнительные сведения о файлах **keytab** в протоколе Samba см. на сайте **goo.gl/ZxCUKA** (глубокая ссылка внутри **wiki.samba.org**).

Например, эта конфигурация позволит пользователю `janderson` получить доступ к своему рабочему каталогу через путь `\\sambaserver\janderson` из любой системы Windows в сети.

На некоторых серверах разрешения по умолчанию для рабочих каталогов позволяют пользователям просматривать файлы друг друга. Поскольку в пакете Samba используются разрешения файлов UNIX для реализации контроля доступа, пользователи Windows, входящие через Samba, могут также читать домашние каталоги друг друга. Однако опыт показывает, что такое поведение приводит к путанице пользователей Windows и делает их уязвимыми.

Переменная `%S`, указанная как значение `valid users` в приведенном выше примере, заменяется на имя пользователя, связанного с каждым общим ресурсом; таким образом, она ограничивает доступ к владельцу домашнего каталога. Удалите эту строку, если такое поведение сервера SMB для вас нежелательно.

Пакет Samba использует специальный раздел файла конфигурации `[homes]` в качестве последнего средства. Если в рабочем каталоге конкретного пользователя есть явно определенный общий ресурс в файле конфигурации, параметры, установленные там, переопределяют значения, установленные в разделе `[homes]`.

Совместное использование каталогов проектов

🕮 Дополнительную информацию о списках ACL см. в разделе 5.6.

Пакет Samba может отображать списки управления доступом системы Windows (ACL) на традиционные права доступа к файлам в системах UNIX или ACL, если это поддерживает файловая система. Однако на практике мы обнаруживаем, что ACL слишком сложны для большинства пользователей.

Вместо использования списков ACL мы обычно настраиваем специальную общую папку для каждой группы пользователей, которая нуждается в коллективной рабочей области. Когда пользователь пытается подключить этот ресурс, пакет Samba проверяет, что заявитель находится в соответствующей группе UNIX, прежде чем разрешить доступ.

В приведенном ниже примере пользователь должен быть частью группы `eng` для подключения общей папки и доступа к ее файлам.

```
[eng]
comment = Общая папка для инженерного отдела
; Для доступа к этому ресурсу пользователь должен входить в группу eng.
; Пользователи должны подключаться со своими учетными записями, созданными
; на сервере Samba.
valid users = @eng
path = /home/eng

; Отключим NT ACL, так как мы их не используем.
nt acl support = no

; Убедитесь, что всем файлам в папке назначены соответствующие
; права доступа и что для этих каталогов установлен бит setgid
; (наследование группы).
create mask = 0660
directory mask = 2770
force directory mode = 2000
force group = eng
```

```
; Общие параметры для всех ресурсов.
browseable = no
read only = no
guest ok = no
```

Эта конфигурация не требует создания псевдопользователя, чтобы он мог действовать как владелец общей папки. Вам нужно просто создать группу в системе UNIX (здесь, `eng`) и включить в нее предполагаемых пользователей этого общего ресурса.

Пользователи будут подключать общий ресурс под своими учетными записями, но для облегчения совместной работы мы предпочли бы, чтобы любые файлы, созданные в рамках этого ресурса, принадлежали группе `eng`. Таким образом, другие члены команды могут получить доступ к вновь созданным файлам по умолчанию.

Первым шагом на пути к обеспечению такого поведения является применение параметра `force group` для принудительного использования текущего идентификатора группы UNIX `eng`, которая контролирует доступ к общей папке. Однако этого шага недостаточно, чтобы гарантировать, что все новые файлы и каталоги будут принадлежать к группы `eng`.

Как указывалось в разделе 5.5, установленный для каталога флаг `setgid` задает для всех новых файлов, созданных в этом каталоге, в качестве владельца группу этого каталога.[4] Чтобы гарантировать, что все вновь созданные файлы будут принадлежать группе `eng`, мы должны установить для корневой папки группу `eng`, а затем включить бит `setgid` для этого каталога, как показано ниже.

```
$ sudo chown root:eng /home/eng
$ sudo chmod u=rwx,g=rwxs,o= /home/eng
```

Эти меры достаточны для управления файлами, созданными в корневой папке. Однако для того, чтобы система работала со сложными иерархиями файлов, нам также необходимо обеспечить установку битов `setgid` для вновь созданных каталогов. Приведенная выше примерная конфигурация реализует это требование с помощью параметров `force directory mode` и `directory mask`.

22.3. Монтирование общих SMB-ресурсов

Монтирование общих SMB-ресурсов работает совсем не так, как это делается в других сетевых файловых системах. В частности, SMB-тома монтируются конкретным пользователем, а не самой системой.

Для выполнения монтирования SMB-ресурсов требуется локальное разрешение. Вам также нужен пароль для идентификации, который позволит удаленному SMB-серверу разрешить доступ к общему ресурсу. Типичная командная строка в системе Linux выглядит следующим образом.

```
$ sudo mount -t cifs -o username=joe //redmond/joes /home/joe/mnt
```

Ее эквивалент в системе FreeBSD имеет следующий вид.

```
$ sudo mount -t smbfs //joe@redmond/joes /home/joe/mnt
```

В системе Windows монтирования сетевых ресурсов выполняются для конкретного пользователя (отсюда и параметр **username=joe**), тогда как в системе UNIX они выпол-

[4] Или, по крайней мере, он делает это в системе Linux. В системе FreeBSD не устанавливается бит `setgid` для каталога; однако поведение по умолчанию заключается в унаследовании группы для новых файлов, так же, как в Linux это делается с включенным битом `setgid`. Впрочем, установка бита `setgid` в системе FreeBSD ничему не вредит.

няются как правило для всей системы в целом. Серверы Windows обычно не допускают, чтобы несколько разных пользователей могли получить доступ к смонтированному общему ресурсу Windows.

С точки зрения клиента UNIX все файлы в смонтированном каталоге, принадлежат пользователю, который его смонтировал. Если вы монтируете общий ресурс с правами администратора, то все файлы будут принадлежать пользователю `root`, а остальные пользователи, возможно, не смогут записывать файлы на сервере Windows.

Параметры монтирования **uid**, **gid**, **fmask** и **dmask** позволяют настроить эти параметры так, чтобы права собственности и биты разрешений были более точно согласованы с предполагаемой политикой доступа для этого ресурса. Дополнительную информацию об этих параметрах можно найти на страницах справочника для команды **mount.cifs** (Linux) или **mount_smbfs** (FreeBSD).

22.4. Просмотр файлов на общих SMB-ресурсах

В пакет Samba включена утилита командной строки, называемая **smbclient**, которая позволяет отображать список файлов конкретного сетевого ресурса без фактического их монтирования. В нем также реализован FTP-подобный интерфейс для интерактивного доступа. Эта функция может быть полезна при отладке или когда сценарию необходим доступ к общей папке.

Например, вот как можно вывести список доступных сетевых ресурсов для пользователя **dan** на сервере `hoarder`.

```
$ smbclient -L //hoarder -U dan
Enter dan's password: <password>
Domain=[WORKGROUP] OS=[Unix] Server=[Samba 3.6.21]

        Sharename   Type    Comment
        ---------   ----    -------
        Temp        Disk    Temp Storage
        Programs    Disk    Various Programs and Applications
        Docs        Disk    Shared Documents
        Backups     Disk    Backups of all sorts
```

Для того чтобы подключиться к общему ресурсу и передать файлы, уберите флаг **-L** и укажите имя этого ресурса.

```
$ smbclient //hoarder/Docs -U dan
```

После подключения введите команду **help** для получения списка доступных команд.

22.5. Обеспечение безопасности Samba-сервера

Важно помнить о влиянии безопасности на совместное использование файлов и других сетевых ресурсов. Чтобы обеспечить базовый уровень безопасности для типичной организации, необходимо сделать две вещи.

- Явно указать, какие клиенты могут получить доступ к общим ресурсам SMB-сервера. Эта часть конфигурации задается параметром **hosts allow** в файле **smb.conf**. Убедитесь, что он содержит только необходимые IP-адреса, диапазоны адресов или имена хостов.

В файл **smb.conf** можно также включить параметр настройки **hosts deny**, но учтите, что директива запрета на доступ к сетевым ресурсам имеет приоритет. Если вы укажете имя хоста или его адрес как в параметре **hosts deny**, так и в **hosts allow**, указанный хост не сможет получить доступ к ресурсу.

- Блокировать доступ к SMB-серверу за пределами сети вашей организации. В пакете Samba используется шифрование только для аутентификации с помощью пароля. В нем не используется шифрование для передачи данных. Почти во всех случаях вам следует блокировать доступ извне сети вашей организации, чтобы пользователи случайно не загружали файлы в открытом виде через Интернет.

  Блокировка обычно выполняется на уровне сетевого брандмауэра. Пакет Samba использует UDP-порты 137–139 и TCP-порты 137, 139 и 445.

С момента выпуска Samba версии 3 в викисистеме Samba по адресу `wiki.samba.org` доступна отличная документация по настройке системы безопасности.

22.6. Отладка Samba-сервера

Сервер Samba обычно не требует особого внимания. Если у вас возникла проблема, вы можете обратиться к двум основным источникам информации для отладки: команде **smbstatus** и средствам протоколирования Samba.

Запрос состояния Samba-сервера с помощью команды smbstatus

Команда **smbstatus** показывает активные соединения и заблокированные файлы; это первое место, в которое следует заглянуть, когда возникают проблемы. Выводимая информация особенно полезна для отслеживания проблем с блокировкой (например, "У какого пользователя есть эксклюзивный доступ к файлу **xyz** для чтения и записи?").

```
$ sudo smbstatus                    # Часть вывода сжата для ясности
Samba version 4.3.11-Ubuntu
PID     Username    Group       Machine
-------------------------------------------------------
6130    clay        atrust      192.168.20.48
23006   dan         atrust      192.168.20.25

Service   pid       machine     Connected at
-------------------------------------------------------
admin     6130   192.168.20.48   Wed Apr 12 07:25:15 2017
swdepot2  6130   192.168.20.48   Wed Apr 12 07:25:15 2017
clients   6130   192.168.20.48   Wed Apr 12 07:25:15 2017
clients   23006  192.168.20.25   Fri Apr 28 14:32:25 2017

Locked files:
Pid    Uid  DenyMode    R/W      Oplock   SharePath        Name
--------------------------------------------------------------------------
6130  1035  DENY_NONE   RDONLY   NONE     /atrust/admin    New Hire Proces...
6130  1035  DENY_ALL    RDONLY   NONE     /home/clay       .
23006 1009  DENY_NONE   RDONLY   NONE     /atrust/clients  Acme_Supply/Con...
```

В первом разделе вывода отображается список подключившихся пользователей. В столбце `Service` в следующем разделе показаны фактические сетевые ресурсы, которые они смонтировали. В последнем разделе, из которого мы удалили несколько столбцов для экономии места, перечислены любые активные блокировки файлов.

Если вы завершите работу демона **smbd**, связанного с определенным пользователем, то все блокировки этого пользователя исчезнут. Некоторые приложения обрабатывают эту ситуацию изящно и запрашивают блокировки заново. Другие просто зависают и требуют вмешательства со стороны Windows, чтобы закрыть неудачное приложение. Как бы драматично это ни звучало, никаких искажений данных в файлах в результате такой процедуры не возникает.

Будьте внимательны, когда Windows заявляет, что файлы были заблокированы другим приложением; это часто оказывается правильным предупреждением. Устраните проблему на стороне клиента, закрыв приложение-нарушитель, а не пытайтесь грубо снимать блокировки на сервере.

Настройка журнала Samba-сервера

Настройте параметры журнала в файле **smb.conf**.

```
[global]
# Параметр %m создает отдельный файл журнала для каждого клиента.
log file = /var/log/samba.log.%m
max log size = 10000

# Чтобы Samba-сервер выводил системный журнал исключительно средствами syslog,
# установите значение следующего параметра равным 'yes'.
syslog only = no

# Старайтесь минимизировать вывод Samba-сервера в системный журнал syslog.
# Вся информация должна записываться в файлы /var/log/samba/log.{smbd,nmbd}.
# Чтобы вывести в syslog более подробную информацию, увеличьте значение
# следующего параметра
syslog = 7
```

При увеличении значения параметра `syslog` в журнал будет выводиться больше информации. Для ведения журнала задействуются системные ресурсы, поэтому не устанавливайте слишком высокий уровень ведения журнала, если вы не собираетесь выполнять активную отладку.

В следующем примере показаны записи журнала, созданные в результате неудачной попыткой подключения.

```
[2017/04/30 08:44:47.510724, 2, pid=87498, effective(0,
   0), real(0, 0), class=auth] ../source3/auth/
   auth.c:315(auth_check_ntlm_password)
check_ntlm_password: Authentication for user [dan] -> [dan] FAILED
   with error NT_STATUS_WRONG_PASSWORD
[2017/04/30 08:44:47.510821, 3] ../source3/smbd/
   error.c:82(error_packet_set)
NT error packet at ../source3/smbd/sesssetup.c(937) cmd=115
   (SMBsesssetupX) NT_STATUS_LOGON_FAILURE
```

Удачная попытка подключения выглядит следующим образом.

```
[2017/04/30 08:45:30.425699, 5, pid=87502, effective(0,
   0), real(0, 0), class=auth] ../source3/auth/
```

```
   auth.c:292(auth_check_ntlm_password)
check_ntlm_password: PAM Account for user [dan] succeeded
   [2017/04/30 08:45:30.425864, 2, pid=87502, effective(0,
   0), real(0, 0), class=auth] ../source3/auth/
   auth.c:305(auth_check_ntlm_password)
check_ntlm_password: authentication for user [dan] -> [dan] -> [dan]
   succeeded
```

Команда **smbcontrol** удобна для изменения уровня отладки работающего Samba-сервера без внесения изменений в файл **smb.conf**. Например,

```
$ sudo smbcontrol smbd debug "4 auth:10"
```

Эта команда устанавливает для глобального уровня отладки значение 4 и устанавливает уровень отладки для связанных с аутентификацией вопросов до 10. Аргумент **smbd** указывает, что все демоны **smbd** в системе должны иметь установленные уровни отладки. При отладке конкретного установленного соединения используйте команду **smbstatus**, чтобы определить, какой демон **smbd** обрабатывает соединение, а затем передайте его идентификатор PID в **smbcontrol** для отладки только одного соединения.

При уровнях журнала более 100 вы начнете видеть зашифрованные пароли в журналах.

Управление наборами символов

Начиная с версии 3.0, Samba кодирует все имена файлов в кодировке UTF-8. Если ваш сервер работает с локальной кодировкой UTF-8, которую мы рекомендуем, то все отлично.[5] Если вы находитесь в Европе и по-прежнему используете на сервере один из языковых стандартов ISO 8859, то можете обнаружить, что созданные Samba-сервером имена файлов, которые включают акцентированные символы (например, ä, ö, ö, é или è), при выполнении команды **ls** отображаются неправильно. Решение состоит в том, чтобы Samba-сервер использовал кодировку вашего сервера:

```
unix charset = ISO8859-15
display charset = ISO8859-15
```

Убедитесь, что кодировка имен файлов правильно работает с самого начала. В противном случае файлы с неправильно закодированными именами накапливаются. Их последующее исправление — удивительно сложная задача.

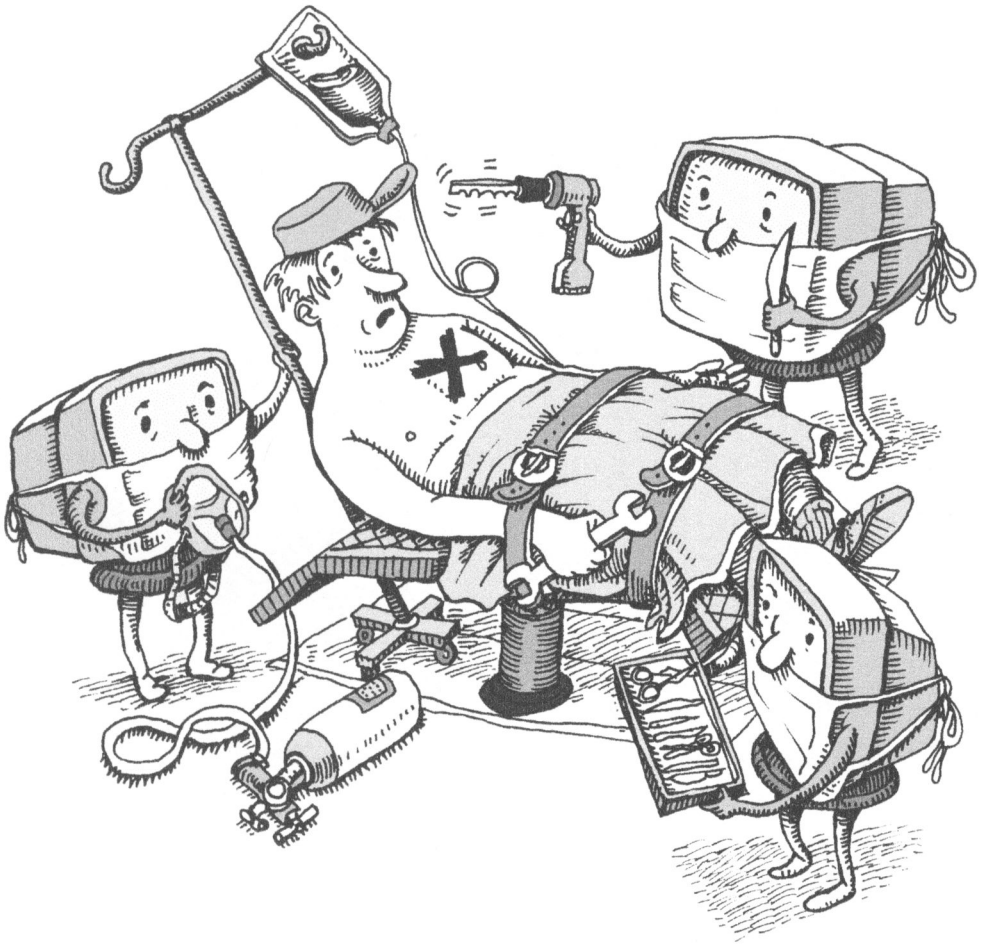
22.7. Литература

- Red Hat. *Red Hat Enterprise Linux System Administrator's Guide: File and Print Servers.* goo.gl/LPjNXa (глубокая ссылка на сайте access.redhat.com/documentation).
- Samba Project. *Samba Wiki Page.* wiki.samba.org. Эта вики-система обновляется относительно часто и является авторитетным источником информации, хотя ее части несколько дезорганизованы.

[5] Чтобы узнать, работает ли ваша система в режиме UTF-8, выполните команду **echo $LANG**.

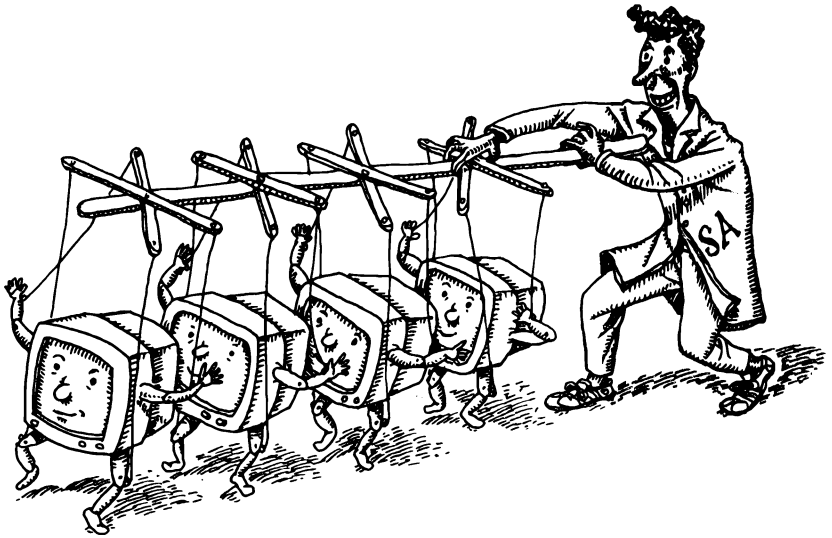
ЧАСТЬ IV

Эксплуатация



Глава 23

Управление конфигурацией



Многолетний принцип системного администрирования заключается в том, что изменения должны быть структурированными, автоматизированными и согласованными. Но это легче сказать, чем сделать, когда вы сталкиваетесь с разнородным парком систем и сетей, пребывающих в разных состояниях.

Программное обеспечение для управления конфигурацией автоматизирует управление операционными системами в сети. Администраторы составляют спецификации, описывающие, как серверы должны быть сконфигурированы, а программное обеспечение для управления конфигурацией затем воплощает в реальность соответствие с этими спецификациями. На практике широко используются версии управления конфигурациями с открытым исходным кодом. В этой главе мы расскажем о принципах управления конфигурацией и охарактеризуем основных игроков.

В качестве инструмента автоматизации управление конфигурацией тесно связано с методологией IT-операций DevOps, о которой мы более подробно расскажем в разделе 31.1. Люди зачастую не различают DevOps и управление конфигурацией, поэтому иногда эти термины используются как синонимы. Тем не менее они имеют разный смысл. В этой главе мы приведем несколько примеров, демонстрирующих, как управление конфигурацией воплощает несколько ключевых элементов методологии DevOps, в то же время не совпадая с ней.

Словосочетание "система управления конфигурацией" немного длинновато для чтения и записи, поэтому мы часто сокращаем этот термин до "системы CM (configuration management)" (или даже просто CM). (К сожалению, аббревиатура CMS (content management system) уже широко используется для обозначения "системы управления содержимым".)

23.1. Краткое введение в управление конфигурацией

🕮 Дополнительную информацию о сценариях оболочки см. в главе 7.

Традиционный подход к автоматизации системного администрирования основан на использовании сложного комплекса самостоятельно разработанных сценариев оболочки, дополненных специальными средствами безопасности на случай, если сценарии потерпят неудачу. Некоторое время эта схема работает хорошо, как и следовало ожидать. Однако со временем системы, управляемые таким образом, обычно вырождаются в хаотические обломки версий пакетов и конфигураций, которые невозможно достоверно воспроизвести. Такую схему иногда называют моделью системного администрирования снежинок, потому что не существует двух похожих друг на друга систем.

Более эффективный подход — управление конфигурацией. Он фиксирует желаемое состояние в виде кода. Затем изменения и обновления можно отслеживать с течением времени в системе контроля версий, создающей контрольный журнал и точку отсчета. Код также выступает в качестве неофициальной документации по структуре сети. Любой администратор или разработчик может прочитать код, чтобы определить, как настроена система.

Когда все серверы организации находятся под управлением конфигурации, система CM эффективно действует как база данных о ресурсах, а также как центр сетевого управления и контроля. Системы CM также предлагают функции "оркестровки", которые позволяют удаленно осуществлять изменения и выполнять специальные команды. Вы можете нацелиться на группы хостов, имена которых соответствуют определенным шаблонам или переменные конфигурации которых соответствуют заданному набору значений. Управляемые клиенты сообщают информацию о себе в центральную базу данных для анализа и мониторинга.

В большинстве случаев код управления конфигурацией использует декларативную, а не процедурную идиому. Вместо написания сценариев, которые сообщают системе, какие изменения нужно внести, вы описываете состояние, которого хотите достичь. Затем система управления конфигурацией использует собственную логику для настройки целевых систем по мере необходимости.

В конечном счете работа системы CM заключается в применении ряда спецификаций конфигурации, например операций, к отдельной машине. Операции различаются по степени детализации, но они обычно достаточно крупные, чтобы соответствовать элементам, которые могут чаще других отображаться в списке дел системного администратора: создать учетную запись пользователя, установить пакет программного обеспечения и т.д. Для полной настройки подсистемы, такой как база данных, может потребоваться от 5 до 20 операций. Полная конфигурация только что загруженной системы может повлечь за собой десятки или сотни операций.

23.2. Опасности управления конфигурацией

Управление конфигурацией — это значительное улучшение по сравнению с специальным подходом, но не волшебная палочка. Для администраторов имеет большее значение несколько острых проблем, о которых нужно знать заранее.

Хотя все основные системы CM используют аналогичные концептуальные модели, они описывают эти модели с помощью разных лексиконов. К сожалению, терминология, используемая конкретной системой CM, часто ориентируется на маркетинг, а не на максимальную ясность.

Результатом является отсутствие согласованности и стандартизации среди систем. Большинство администраторов будут сталкиваться с несколькими системами СМ на протяжении всей своей карьеры и будут развивать предпочтения, полученные на основе этого опыта. К сожалению, знание одной системы напрямую не переносится на другую.

По мере роста организации требуется инфраструктура, необходимая для поддержки своей системы управления конфигурацией. Для организации с несколькими тысячами серверов нужно несколько систем, предназначенных для выполнения рабочих нагрузок СМ. Для создания таких систем необходимы расходы на аппаратные ресурсы и постоянное обслуживание. Внедрение СМ-систем может быть одним из основных проектов.

Для того чтобы организация полностью использовала управление конфигурацией, необходим определенный уровень оперативной зрелости и строгости. Как только хост оказывается под управлением системы СМ, его нельзя изменять вручную, иначе система немедленно перейдет в состояние системы снежинок.[1]

Хотя некоторые системы СМ легче осваивать, чем другие, все они известны тем, что имеют крутую кривую обучения, особенно для администраторов, которым не хватает опыта в автоматизации. Если вы соответствуете этому описанию, подумайте о том, чтобы пройти практику в лаборатории виртуальных машин, оттачивая свои навыки, прежде чем заниматься производственной сетью.

23.3. Элементы управления конфигурацией

В этом разделе мы рассмотрим компоненты системы СМ и концепции, используемые для ее настройки на более высоком уровне детализации. Затем, начиная с раздела 23.4, мы изучим четыре из наиболее часто используемых систем СМ: Ansible, Salt, Puppet и Chef.

Вместо того чтобы принимать какую-либо терминологию конкретной системы СМ, мы используем самый ясный и наиболее понятный термин, который смогли найти для каждой концепции. В табл. 23.2 приводится соответствие между нашей лексикой и терминологией четырех систем СМ, перечисленных выше. Если вы уже знакомы с одной из этих систем СМ, рекомендуем обратиться к данной таблице при чтении приведенного ниже материала.

Операции и параметры

Мы уже рассматривали концепцию операций, которые представляют собой мелкомасштабные действия и проверки, используемые системой СМ для достижения определенного состояния. Каждая система СМ содержит большой набор поддерживаемых операций, который увеличивается с появлением каждой новой версии.

Вот несколько примеров операций, которые все системы СМ могут выполнять без предварительной подготовки:

- создавать или удалять учетную запись пользователя либо устанавливать ее атрибуты;
- копировать файлы в настраиваемую систему или из нее;

[1] Мы неоднократно сталкивались с ситуациями, в которых ленивые или торопливые администраторы вручную обновляли хост, управляемый конфигурацией, и делали свои модификации неизменяемыми, тем самым переопределяя ожидаемое состояние и не позволяя системе СМ применять будущие модификации. Это приводит к большой путанице, потому что коллеги администратора не смогут быстро понять, почему ожидаемая конфигурация не применяется к данному хосту. В одном случае это вызвало большой перерыв в обслуживании.

- синхронизировать содержимое каталога;
- визуализировать шаблон конфигурационного файла;
- добавлять новую строку в файле конфигурации;
- перезапускать службу;
- добавлять задание **cron** или таймер **systemd**;
- запускать произвольную команду системной оболочки;
- создавать новый экземпляр облачного сервера;
- создавать или удалять учетную запись базы данных;
- определять параметры работы базы данных;
- выполнять операции в репозитории Git.

Это всего лишь выборка; в большинстве систем СМ определены сотни операций, в том числе многие, которые выполняют потенциально сложные специальные операции, такие как настройка конкретных баз данных, среды выполнения или даже части оборудования.

Если операции кажутся подозрительно похожими на команды оболочки, ваша интуиция вас не подводит. Это сценарии, обычно написанные на языке реализации самой системы СМ и использующие стандартные инструменты и библиотеки системы. Во многих случаях они автоматически запускают стандартные команды оболочки как часть их реализации.

Так же, как в командах UNIX можно указать ряд параметров, большинству операций можно передать параметры. Например, операции управления пакетами передаются параметры, которые определяют имя пакета, версию и должен ли пакет быть установлен или удален.

Параметры меняются в зависимости от операции. Для удобства им обычно присваиваются стандартные значения, которые подходят для наиболее распространенных случаев использования.

Системы СМ позволяют использовать значения переменных (см. следующий раздел) для определения параметров. Они также могут сами определить значения параметров в соответствии со средой, в которой развернута система, например в сети, в которой инсталлирована система, в зависимости от того, присутствует ли конкретное свойство конфигурации или совпадает ли название хоста, на котором инсталлирована система, с данным регулярным выражением.

Корректная операция не должна зависеть от хоста или хостов, к которым в конечном итоге она может быть применена. Ее реализация должна быть относительно универсальной и не зависящей от операционной системы. Операции, привязанные к конкретным системам, выполняются на более высоком уровне иерархии управления конфигурацией.

Несмотря на то что системы СМ сосредоточены на декларативной конфигурации, операции должны в конечном счете выполняться, как и любая другая команда. Исполнение имеет начало и конец. Операция может завершиться успехом или неудачей. Она должна возвращать свое состояние вызывающей среде.

Однако операции отличаются от типичных команд UNIX несколькими важными аспектами.

- Большинство операций предназначены для многократного и безопасного применения. Заимствуя термин из линейной алгебры, это свойство иногда называют *идемпотентностью*.
- Операции знают, когда они изменяют фактическое состояние системы.

- Операции знают, когда *необходимо* изменить состояние системы. Если текущая конфигурация уже соответствует спецификации, операция завершается без каких-либо действий.
- Операции сообщают о своих результатах системе CM. Их данные отчета содержат больше информации, чем простой код завершения в стиле UNIX, и могут помочь в отладке.
- Операции должны быть кросс-платформенными. Обычно они определяют ограниченный набор функций, которые являются общими для всех поддерживаемых платформ, и интерпретируют запросы в соответствии с локальной системой.

Некоторые операции не могут быть сделаны идемпотентными без небольшой помощи администратора, который знает больше об этом контексте. Например, если операция запускает выполнение последовательности команд UNIX, то система CM не имеет прямого способа узнать, какой эффект она произвела на систему.

Кроме того, существует возможность писать собственные пользовательские операции. Это всего лишь сценарии, и система CM обычно обеспечивает гладкий способ интеграции пользовательских и стандартных операций.

Переменные

Переменные — это именованные значения, которые влияют на то, как конфигурации применяются к отдельным машинам. Они обычно устанавливают значения параметров и заполняют пробелы в шаблонах конфигурации.

Управление переменными в системах CM часто предоставляет много возможностей. Приведем несколько замечаний по этому поводу.

- Переменные обычно могут быть определены в нескольких местах и контекстах в базе конфигурации.
- Каждое определение имеет область видимости. Типы области видимости варьируются в зависимости от системы CM и могут охватывать одну машину, группу машин или определенный набор операций.
- Несколько областей могут быть активны в любом заданном контексте. Области могут быть вложенными, но чаще они просто взаимодействуют.
- Поскольку в нескольких областях могут определяться значения для одной и той же переменной, требуется некоторая форма разрешения конфликтов. В некоторых системах значения объединяются, но в большинстве используются правила приоритета или порядок выбора основного значения.

Переменные не ограничены скалярными значениями; массивы и хеши также являются приемлемыми значениями переменных во всех системах CM. Некоторые операции принимают значения нескалярных параметров напрямую, но такие значения чаще используются выше уровня отдельных операций. Например, элементы массива можно перебрать в цикле для применения одной и той же операции более одного раза с разными параметрами.

Факты

Системы CM исследуют каждый клиент конфигурации для определения описательных фактов, таких как IP-адрес основного сетевого интерфейса и тип операционной системы. Затем эта информация становится доступной из базы конфигурации через значения переменных. Как и в любой другой переменной, эти значения могут использоваться для определения значений параметров или для расширения шаблонов.

Для определения всех фактов, связанных с конкретной системой, может потребоваться некоторое время. Поэтому системы CM обычно кешируют факты, и не обязательно перестраивают кеш при каждом запуске. Если вы обнаружите, что в конкретном потоке конфигурации встречаются устаревшие данные конфигурации, вам может потребоваться явно аннулировать кеш.

Все системы CM позволяют целевым машинам добавлять значения в базу данных фактов либо путем включения статического файла деклараций, либо путем запуска специального кода на целевой машине. Эта функция полезна как для расширения типов информации, к которой можно получить доступ через базу данных фактов, так и для перемещения информации о статической конфигурации на клиентские компьютеры.

Рекомендации на стороне клиента могут быть особенно полезны для управления облачными и виртуальными серверами. Вы просто назначаете маркеры уровня облачности (например, теги EC2), когда создается экземпляр, а затем система управления конфигурацией может загрузить соответствующую конфигурацию по этим маркерам. Однако имейте в виду последствия этого подхода для безопасности: клиент контролирует факты, которые он сообщает, поэтому убедитесь, что скомпрометированный клиент не может использовать систему управления конфигурацией для получения дополнительных привилегий.

В зависимости от системы CM вы можете выйти за рамки своего локального контекста с помощью анализа окружения переменных или фактов. Помимо доступа к информации конфигурации для текущего хоста, вы также можете иметь доступ к данным для других хостов или даже для анализа состояния самой базы конфигурации. Это полезная функция для координации распределенной системы, такой как кластер серверов.

Обработчики изменений

Если вы изменили конфигурационный файл веб-сервера, вам лучше перезапустить веб-сервер. Это основная концепция обработчиков, которые являются операциями, выполняющимися в ответ на какое-то событие или ситуацию, а не относятся к базовой конфигурации.

В большинстве систем обработчик запускается, когда одна или несколько операций из заданного набора операций сообщают, что они изменили целевую систему. Обработчику ничего не говорится о точном характере изменения, но, поскольку связь между операциями и их обработчиками довольно специфична, дополнительная информация не требуется.

Привязки

Привязки завершают базовую конфигурационную модель, связывая конкретные наборы операций с конкретными хостами или группами хостов. Вы также можете связывать операции с динамическим набором клиентов, который определяется значением факта или переменной. Системы CM также могут определять группы хостов, просматривая информацию в локальной системе управления ресурсами или вызывая удаленную процедуру согласно интерфейсу API.

В дополнение к основной роли связывания, привязки в большинстве систем CM также действуют как переменные области. Эта функция позволяет настраивать поведение назначаемых операций, определяя или настраивая значения переменных для целевых клиентов. Данный хост может соответствовать критериям для множества разных привязок. Например, хост может находиться в определенной подсети, управляться определенным

отделом или выполнять явно назначенную роль (например, веб-сервер Apache). Система СМ учитывает все эти факторы и активирует операции, связанные с каждой привязкой.

После того как вы настроили привязки для хоста, вы можете вызвать механизм "настроить все" системы верхнего уровня вашей системы СМ, чтобы система СМ идентифицировала все операции, которые должны выполняться на целевом хосте, и выполнить их по порядку.

Пакеты и репозитории пакетов

Пакет представляет собой набор операций, которые выполняют определенную функцию, например установку, настройку и запуск веб-сервера. Системы СМ позволяют вам формировать пакеты в формате, подходящем для распространения или повторного использования. В большинстве случаев пакет определяется каталогом, а имя каталога определяет имя пакета.

Поставщики СМ поддерживают открытые репозитории, которые включают как официально одобренные, так и пользовательские пакеты. Вы можете использовать их "как есть" или изменить их в соответствии со своими потребностями. В большинстве систем СМ предоставляются собственные команды для взаимодействия с репозиториями.

Среды

Часто бывает полезно разделить клиенты, управляемые конфигурацией, на несколько "миров", таких как традиционные категории разработки, тестирования и производства. Крупные инсталляции могут создавать еще более тонкие различия для поддержки таких процессов, как постепенное ("ступенчатое") внедрение нового кода в производство.

Эти разные миры, "среды", известны как внутри, так и вне контекста управления конфигурацией. Кажется, это единственный термин, с которым согласны все системы управления конфигурацией.

При правильной реализации среды — это не просто группы клиентов. Это дополнительная ось изменения, которая может повлиять на несколько аспектов конфигурации. Например, среды разработки и производства могут включать веб-серверы и серверы баз данных, но детали того, как эти роли определены, могут различаться в разных средах.

Например, для базы данных и веб-сервера обычно используется один и тот же компьютер в среде разработки. Однако производственная среда обычно имеет несколько серверов каждого типа. Производственная среда также может определять типы серверов, которых нет в среде разработки, например те серверы, которые выполняют балансировку нагрузки или действуют как прокси-серверы в демилитаризованной зоне.

Система окружающей среды обычно рассматривается как своего рода конвейер для кода конфигурации. В качестве мысленного эксперимента можно себе представить, что фиксированные группы клиентов используют среды разработки, тестирования и производства. Поскольку данная база конфигурации проверена, она распространяется от одной среды к другой, обеспечивая должную проверку изменений до того, как они достигнут важнейших производственных систем.

В большинстве систем СМ разные среды — это разные версии одной и той же конфигурации. Если вы являетесь пользователем Git, считайте их тегами в репозитории Git: тег разработки указывает на самую последнюю версию базы конфигурации, а производственный тег может указывать на фиксацию состояния, которая была сделана несколько недель назад. Теги продвигаются вперед по мере того, как выпуски проходят через стадии тестирования и развертывания.

Различные среды могут предоставлять клиентам разные значения переменных. Например, учетные данные базы данных, используемые в разработке, скорее всего, будут отличаться от используемых в производственных системах, также как детали конфигурации сети и, возможно, имена пользователей и групп, которым разрешен доступ.

🕮 Дополнительную информацию о средах см. в главе 26.

Учет и регистрация клиентов

Поскольку системы CM определяют множество способов разделения клиентов на категории, общий парк машин под управлением конфигурации должен быть четко определен. Реестр управляемых хостов может храниться в обычном файле или в соответствующей реляционной базе данных. В некоторых случаях он может даже быть полностью динамичным.

Точный механизм, посредством которого код конфигурации распределяется, анализируется и выполняется, зависит от систем CM. Большинство систем на самом деле предоставляют несколько вариантов в этом отношении. Укажем несколько общих подходов.

- На каждом клиенте непрерывно работает демон, который загружает код конфигурации с назначенного сервера CM (или группы серверов).
- Центральный сервер CM передает данные конфигурации каждому клиенту. Этот процесс может выполняться по регулярному расписанию или может быть инициирован администраторами.
- Каждый управляемый хост запускает клиент, который периодически просыпается, считывает данные конфигурации из локального клона базы конфигурации и применяет соответствующую конфигурацию к себе. Центрального сервера конфигурации нет.

Конфигурационная информация является закрытой и часто включает в себя такие секреты, как пароли. Для защиты этих данных, во всех системах CM определяется способы аутентификации клиентов и серверов, а также алгоритмы шифрования конфиденциальной информации.

Чтобы добавить новый клиент в среду управления конфигурацией достаточно только инсталлировать соответствующее клиентское программное обеспечение. Если среда настроена на поддержку автоматической начальной загрузки, новый клиент может автоматически связаться с сервером конфигурации, пройти аутентификацию и инициировать процесс настройки. Механизмы инициализации, специфичные для операционной системы, обычно запускают эту цепочку событий при первом запуске клиента. Этот поток показан на рис. 23.1.



Рис. 23.1. Процесс инициализации нового клиента, управляемого CM

23.4. Сравнение популярных систем CM

В настоящее время четыре основных игрока владеют рынком общего управления конфигурацией в системах UNIX и Linux: Ansible, Salt, Puppet и Chef. В табл. 23.1 приведена общая информация об этих пакетах.

Таблица 23.1. Основные системы управления конфигурацией

	Языки и форматы				Демоны	
Система	**Веб-сайт**	**Реализация**	**Конфигурация**	**Шаблон**	**Сервер**	**Клиент**
Ansible	`ansible.com`	Python	YAML	Jinja	Нет	Нет
Salt	`saltstack.com`	Python	YAML	Jinja	По выбору	По выбору
Puppet	`puppet.com`	Ruby	Настраиваемая	ERB[a]	По выбору	По выбору
Chef	`chef.io`	Ruby	Ruby	ERB	По выбору	Да

[a] ERB (embedded Ruby) является основным синтаксисом для встраивания кода Ruby в шаблоны.

Все эти пакеты относительно новые. Самый старый, Puppet, дебютировал в 2005 г. Он по-прежнему претендует на самую большую долю на рынке, в значительной степени из-за своего раннего старта. Пакет Chef был выпущен в 2009 г., Salt — в 2011 г. и Ansible — в 2012 г.

Общая категория программного обеспечения для управления конфигурацией была впервые разработана в виде системы CFEngine Марка Берджесса (Mark Bergess) в 1993 г. CFEngine по-прежнему существует и продолжает активно развиваться, но большая часть ее пользовательской базы была завоевана более новыми системами. Для получения текущей информации см. `cfengine.com`.

Компания Microsoft имеет собственное решение CM в виде PowerShell Desired State Configuration. Хотя эта система относится к миру Windows и в первую очередь предназначена для настройки клиентов Windows, Microsoft также опубликовала расширения для настройки систем Linux. Стоит отметить, что все четыре системы, перечисленные в табл. 23.1, также могут настраивать клиентов Windows.

В ряде проектов основное внимание уделяется конкретным подобластям управления конфигурацией, в частности, внедрению новой системы (например, Cobbler) и развертыванию программного обеспечения (например, Fabric и Capistrano). Общее предложение этих систем заключается в том, что, более тщательно моделируя конкретную проблемную область, они могут обеспечить более простой и целенаправленный набор функций.

В зависимости от своих потребностей вы можете обнаружить (или не обнаружить), что эти специализированные системы обеспечивают разумную норму прибыли от ваших инвестиций в обучение. Общие системы управления конфигурацией, подобные описанным в табл. 23.1, не вполне подходят для всех возможных действий.

Системы, приведенные в табл. 23.1, работают практически с любым типом современных UNIX-совместимых клиентских машин, хотя всегда есть граница поддержки. Пакет Chef имеет скромное преимущество в совместимости и поддерживает даже операционную систему AIX.

🕮 Дополнительную информацию о контейнерах см. в главе 25.

Поддержка операционной системы на стороне сервера конфигурации (для тех систем, которые используют сервер конфигурации) более ограничена. Например, пакет Chef требует для своего сервера инсталляции операционных систем RHEL или Ubuntu.

Контейнерные версии сервера могут работать везде, поэтому их использование не является настолько серьезным препятствием, как это может показаться на первый взгляд.

Терминология

В табл. 23.2 показаны термины, используемые в каждой из рассматриваемых нами систем СМ для объектов, указанных в разделе "Элементы управления конфигурацией".

Таблица 23.2. Управление конфигурацией Rosetta Stone

Наш термин	Ansible	Salt	Puppet	Chef
Операция	Задача	Состояние	Ресурс	Ресурс
Тип операции	Модуль	Функция	Тип ресурса, провайдер	Провайдер
Список операций	Задачи	Состояния	Класс, манифест	Рецепт
Параметр	Параметр	Параметр	Свойство, атрибут	Атрибут
Привязка	Сценарий (книга)	Главный файл	Классификация, объявление	Список исполняемых рецептов
Серверный хост	Управление	Мастер	Мастер	Сервер
Клиентский хост	Хост	Миньон	Агент, узел	Узел
Клиентская группа	Группа	Группа узлов	Группа узлов	Роль
Переменная	Переменная	Переменная	Параметр, переменная	Атрибут
Факт	Факт	Гранула	Факт	Автоматический атрибут
Уведомление	Уведомление	Реквизит	Уведомление	Уведомления
Обработчик	Обработчик	Состояние	Подписка	Подписка
Пакет	Роль	Формула	Модуль	Книга рецептов
Репозиторий пакетов	Галактика	GitHub	Кузница	Супермаркет

Бизнес-модели

Все продукты, которые мы обсуждаем, являются пакетами модели freemium. Это означает, что базовые системы бесплатны и доступны по лицензии с открытым исходным кодом. Однако каждая такая система поддерживается со стороны отдельной организации, которая продает услуги по поддержке, консалтингу и дополнительные пакеты.

Теоретически у поставщиков есть потенциальная мотивация не раскрывать исходный код полезных функций, чтобы стимулировать дополнительные продажи. Однако в области управления конфигурацией этот эффект не был очевидным. Версия программного обеспечения с открытым исходным кодом является полнофункциональной и более чем достаточной для большинства организаций.

Дополнительные услуги в основном представляют интерес для крупных организаций. Если ваша организация попадает в эту категорию, вы можете оценить системы управления конфигурацией в отношении функциональности и ценообразования предложений с полным набором стеков. Основными преимуществами являются, как правило, поддержка, индивидуальная разработка, обучение, графические интерфейсы, отчеты и решения для мониторинга. В этой книге мы обсуждаем только основные, бесплатные версии.

Архитектурные параметры

Теоретически системы СМ не требуют серверов. Программное обеспечение может работать только на сконфигурированных компьютерах. Скопируйте базу конфигурации

на каждый целевой хост и просто запустите команду, чтобы сказать: "Я здесь, настройте меня в соответствии с этими спецификациями".

На практике приятно не беспокоиться о деталях получения информации о конфигурации, выталкиваемой клиентам и выполняемой ими. Системы CM всегда создают некоторые условия для централизованного управления, даже если главный компьютер определяется как "где бы вы ни вошли, вы имеете клон базы конфигурации".

Пакет Ansible не использует демоны вообще (кроме **sshd**), чем он и привлекает своей простотой. Выполнение конфигурации происходит, когда администратор (или задание **cron**) на сервере запускает команду **ansible-playbook**. Команда **ansible-playbook** выполняет соответствующие удаленные команды по протоколу SSH, не оставляя следов своего присутствия на клиентской машине после завершения конфигурации. Единственными требованиями для клиентских компьютеров являются их доступность по протоколу SSH и установленная версия Python 2.[2]

Пакеты Salt, Puppet и Chef включают в себя как серверные, так и клиентские демоны. В типичных сценариях развертывания демоны запускаются с обеих сторон отношений, и это среда, которую вы увидите в большинстве документации. Можно запускать каждый из этих пакетов без сервера, но эта конфигурация менее распространена.

Заманчиво предположить, что системы CM с демонами должны быть более тяжелыми и сложными, чем те, у кого их нет (т.е. Ansible). Однако это не обязательно так. В системах Salt и Puppet демоны являются стимуляторами и катализаторами. Они полезны, но необязательны, и они не изменяют фундаментальную архитектуру системы, хотя и позволяют использовать некоторые дополнительные функции. Если хотите, вы можете запускать эти системы без демонов и копировать базу конфигурации вручную.

У системы Salt даже есть SSH-режим, который работает аналогично Ansible. Учитывая это, зачем вам куча дополнительных демонов? По нескольким причинам.

- Эта схема работает быстрее. Разработчики пакета Ansible упорно трудятся, чтобы преодолеть ограничение производительности, налагаемое протоколом SSH и отсутствием на стороне клиента кеширования, но он по-прежнему заметно более медленный, чем Salt. Когда вы читаете книгу о системном администрировании, десять секунд кажутся незначительными. В разгар устранения сбоя эта задержка кажется бесконечной, особенно когда ее испытывают десятки или сотни клиентов.
- Некоторые функции не могут существовать без центральной координации. Например, система Salt позволяет клиентам уведомлять сервер конфигурации о событиях, таких как переполнение диска. Затем вы можете реагировать на эти события с помощью обычных средств управления конфигурацией. Наличие центральной точки подключения облегчает различные функции обмена данными между средами.
- Только демон на стороне сервера действительно является потенциальным источником административной сложности. Системы CM работают, чтобы заставить клиент загружать операцию онлайн, независимо от того, задействован ли демон.
- Наличие активных агентов как на клиенте, так и на сервере открывает множество архитектурных возможностей, недоступных в односторонних конфигурациях.

Что касается архитектуры, то система Chef отличается от остальных систем управления конфигурациями тем, что ее серверный демон является элементом верхнего уровня в концептуальной модели. Системы Salt и Puppet служат для конфигурирования данных непо-

[2] В зависимости от системы вам может понадобиться один или два дополнительных модуля на языке Python. Например, Fedora требует пакет **python-dnf**.

средственно из обычных текстовых файлов на диске; для внесения изменений вы должны просто отредактировать файл. Напротив, сервер Chef является непрозрачным и авторитетным источником информации о конфигурации. Изменения должны быть загружены на сервер с помощью команды **knife** или они не будут доступны для клиентов. (Тем не менее даже система Chef имеет режим работы без сервера в форме команды **chef-solo**.)

Мы перечислили выше преимущества серверных систем вовсе не для того, чтобы способствовать их повсеместному внедрению, а просто для того, чтобы обозначить основную линию раздела среди систем CM, которая проходит между системой Chef и всеми остальными. Системы Ansible, Salt и Puppet имеют одинаковую, умеренную степень сложности. Система Chef требует значительно больших инвестиций для обслуживания и освоения, особенно когда в смесь добавляется обширная линейка дополнительных модулей.

Из-за своей бессерверной модели система Ansible часто упоминается как "простой вариант" для управления конфигурацией. Но на самом деле основные архитектуры Salt and Puppet не менее доступны.[3] Не переводите их в категорию дополнительных вариантов.

Обратное также верно: система Ansible больше, чем просто приукрашенная стартовая система для неопытных системных администраторов. Это вполне приемлемый вариант для сложных организаций, хотя ее низкая производительность в этом контексте становится еще более очевидной.

Параметры языка

Системы Ansible и Salt написаны на языке Python. Системы Puppet и Chef написаны на языке Ruby. Но, кроме системы Chef, эта информация, вероятно, не так актуальна, чем может показаться на первый взгляд.

В типичной конфигурации систем Ansible или Salt код Python нигде не появляется. Обе эти системы в качестве основного языка конфигурации используют YAML (альтернативный синтаксис для выражения представления объектов JavaScript, или JSON). YAML — это просто структурированные данные, а не код, поэтому они не имеют собственного поведения, отличного от интерпретации данных, назначенных системой управления конфигурацией.

Вот простой пример кода системы Salt, в котором подключается и запускается служба SSH.

```
ssh.server.run_ssh:
   service:
      - name: sshd
      - running
      - enable: true
```

Чтобы сделать файлы YAML более динамически выразительными, системы Ansible и Salt дополняют их системой шаблонов Jinja2 в качестве препроцессора.[4] Система Jinja имеет свои корни в языке Python, но это не просто оболочка Python. При использовании она ведет себя как система шаблонов, а не настоящий язык программирования.

[3]Можно было бы привести убедительный аргумент в пользу того, что система Salt является самой простой системой среди всех, если не обращать внимания на ее передовые средства и несколько своеобразную документацию.

[4]Справедливости ради отметим, что система Salt на самом деле не зависит от формата и препроцессора, а также автоматически поддерживает несколько конвейеров ввода (включая исходный Python). Однако отклонение от проторенного пути Jinja и YAML означает отказ от документации и изоляцию от остального мира. Вероятно, это решение лучше всего отложить на будущее, пока вы не освоите систему Salt.

Даже система Salt, которая в большей степени полагается на Jinja, чем Ansible, предостерегает от помещения излишней логики в код Jinja.

Суть в том, что, если вы не пишете собственные типы операций или не используете явные вставки кода на Python, вы не столкнетесь с большим количеством кода на Python в пакетах Ansible и Salt.[5] (Расширение системы CM с помощью вашего собственного кода на самом деле может быть довольно легким и весьма полезным.)

В качестве своих основных систем конфигурации системы Puppet и Chef используют предметно-ориентированные языки на основе Ruby. Версия языка в системе Chef очень похожа на аналог управления конфигурацией на языке Rails из мира веб-разработки. Иначе говоря, он был расширен несколькими концепциями, предназначенными для упрощения управления конфигурацией, но по-прежнему является узнаваемым Ruby. Например:

```
service 'sshd' do
   supports :restart => true, :status => true
   action [:enable, :start]
end
```

Большинство задач управления конфигурацией могут быть решены без углубления в язык Ruby, но при необходимости вам доступна вся мощь этого языка. Вы оцените эту скрытую глубину еще больше, когда станете лучше разбираться в языке Ruby и системе Chef.

В отличие от этого, разработчики системы Puppet довольно много поработали над концептуальной независимостью от языка Ruby и использовали его только как уровень реализации. Хотя язык системы Puppet "под капотом" остается языком Ruby и допускает вставку кода на языке Ruby, он имеет свою собственную структуру, которая более сродни декларативной системе, такой как YAML, чем языку программирования.

```
service {
   "ssh":
     ensure => "running",
     enable => "true"
}
```

По нашему мнению, архитектура системы Puppet не предоставляет администраторам никаких преимуществ. Вместо того чтобы позволить вам использовать язык Ruby, система Puppet просто определяет свой собственный изолированный мир.

Варианты управления зависимостями

Независимо от того, как ваша система управления конфигурацией структурирует свои данные, рабочий список для данного клиента в конечном итоге сводится к набору операций для выполнения клиентом. Некоторые из этих операций зависят от порядка исполнения, а некоторые — нет. Например, рассмотрим следующие задачи системы Ansible для создания учетной записи пользователя `www`, которая может использоваться для владения файлами веб-приложения.

```
- name: Ensure that www group exists
  group: name=www state=present
```

[5] Джон Корбет (John Corbet), один из наших технических рецензентов, согласен с тем, что эти системы не демонстрируют много кода на языке Python... пока ситуация не станет ужасной. "В этот момент, — добавляет он, — знакомство с трассировкой Python и представлениями структуры данных очень помогает".

```
- name: Ensure that www user exists
  user: name=www group=www state=present createhome=no
```

Мы хотим, чтобы пользователь www имел собственную выделенную группу, также называемую www. Пользовательский модуль Ansible не создает группы автоматически, поэтому мы должны сделать это на отдельном этапе. И создание группы должно предшествовать созданию учетной записи www; было бы ошибкой указывать несуществующую группу при создании пользователя.

Система Ansible выполняет операции в том порядке, в котором они представлены в конфигурации, поэтому этот фрагмент конфигурации работает просто отлично. Система Chef тоже работает таким образом, отчасти потому, что гораздо сложнее изменить код, чем данные. Даже если бы это было необходимо, система Chef не могла бы надежно разделить ваш код на куски и создать сборки по своему усмотрению.

Напротив, системы Puppet и Salt позволяют явно объявлять зависимости. Например, в системе Salt эквивалентный набор состояний будет иметь следующий вид.

```
www-user:
   user.present:
      - name: www
      - gid: www
      - createhome: false
      - require:
        - www-group

www-group:
   group.present:
      - name: www
```

Здесь для создания драматического эффекта мы инвертировали порядок операций. Но из-за объявления require операции выполняются в правильном порядке независимо от того, как они отображаются в исходном файле. Следующая команда применяет описанную выше конфигурацию:

```
$ sudo salt test-system state.apply order-test
test-system:
----------
          ID: www-group
    Function: group.present
        Name: www
      Result: True
     Comment: Group www is present and up to date
     Started: 23:30:39.825839
    Duration: 3.183 ms
     Changes:
----------
          ID: www-user
    Function: user.present
        Name: www
      Result: True
     Comment: User www is present and up to date
     Started: 23:30:39.829218
    Duration: 27.435 ms
     Changes:

Summary for test-system
------------
```

```
Succeeded: 2
Failed: 0
------------
Total states run:     2
```

Параметр `require` может быть добавлен к любой операции ("состоянию" в системе Salt), чтобы гарантировать, что именованные предусловия выполняются до текущей операции. Система Salt определяет несколько типов отношений зависимости, и декларации могут появляться по обе стороны от отношения.

Система Puppet работает аналогично. Она также помогает облегчить сложности при объявлении зависимостей, выводя их автоматически в типичных ситуациях. Например, пользовательская конфигурация, которая называет определенную группу, автоматически становится зависимой от ресурса, который настраивает эту группу. Прекрасно!

Итак... Зачем явно объявлять свои зависимости, когда порядок настройки кажется естественным и легким? По-видимому, многие администраторы задавали этот вопрос, так как системы Salt и Puppet перешли к гибридной модели зависимостей, в которой порядок представления имеет значение. Тем не менее это всего лишь фактор внутри данного файла конфигурации; зависимости между файлами должны быть объявлены явно.

Основное преимущество декларирования зависимостей заключается в том, что он делает конфигурации более устойчивыми и явными. Система CM не обязана прерывать процесс настройки при первых признаках неисправности, поскольку знает, какие последующие операции могут быть затронуты сбоем. Она может прервать одну цепочку зависимостей, позволяя другим продолжать функционирование. Это приятно, но, на наш взгляд, все же не оправдывает дополнительной работы по объявлению зависимостей.

Теоретически система CM, которая знает информацию о зависимостях, может распараллеливать выполнение независимых цепей управления на конкретном хосте. Однако ни Salt, ни Puppet не пытаются совершить этот подвиг.

Общие комментарии по поводу систему Chef

Мы видели развертывание основных пакетов CM в организациях разных размеров, и все они проявляют тенденцию к росту энтропии. Советы по организации этих пакетов приведены в разделе 23.5. Тем не менее фундаментальное правило гласит: избегайте сложности, которая не приносит пользы вашей среде.

На практике это означает, что вам нужно четко понимать, применять систему Chef или нет. Система Chef оперирует крупными масштабами. Чтобы получить максимальную отдачу от нее, вы должны иметь:

- сотни или тысячи компьютеров в управлении конфигурацией;
- административный персонал с неравномерными привилегиями и опытом (система внутренних разрешений Chef и несколько интерфейсов здесь весьма полезны);
- специальную систему отчетов, жалоб и соблюдения нормативных требований;
- терпение, необходимое для обучения новых членов команды, не имеющих опыта работы с системой Chef. Конечно, вы можете свободно работать с системой Chef на отдельной машине. Никто не запрещает! Но вам все равно придется нести значительные накладные расходы на многие функции уровня предприятия, которые вы не используете. Они зашиты в архитектуру и документацию.

Нам нравится система Chef. Эта полная, надежная и масштабируемая система лучше, чем ее конкуренты. Но в глубине души мы признаем, что это просто еще одна система управления конфигурацией, которая выполняет те же основные функции, что и Ansible,

Salt и Puppet. Держите систему Chef на примете и не поддавайтесь соблазну внедрить ее только потому, что "она самая мощная" (или потому, что "она использует язык Ruby", если на то пошло).

Мы обнаружили, что привлечение новичков к работе с системой Chef может стать серьезной проблемой, особенно для тех, кто не имеет опыта управления конфигурацией. Эта система требует от мышления разработчика больше, чем другие системы. Здесь будет полезен предварительный опыт в области программирования.

Концепция порядка следования атрибутов в системе Chef является мощной, но также может оказаться источником разочарования. Своеобразная комбинация кулинарных терминов и интернет-мемов раздражает и не является информативной. Разрешение зависимостей между кулинарными книгами может оказаться сложным; иногда происходит разрыв зависимостей между старыми и новыми версиями, и тогда во всех ваших системах возникают проблемы, если вы забыли привязать все ваши зависимости к определенной версии.

Общие комментарии по поводу системы Puppet

Система Puppet — старейшая из четырех основных систем CM и единственная, имеющая большую инсталлированную базу. У нее много пользователей, множество встроенных модулей и бесплатный веб-интерфейс. Тем не менее она довольно быстро уступает свою долю рынка более поздним конкурентам.

В качестве прочного середнячка система Puppet находится под давлением с обоих концов рынка. Она известна узкими местами на стороне сервера, которые вызывают проблемы при управлении тысячами хостов, а за последние пару лет несколько крупных компаний, использовавших систему Puppet, отказались от нее (наиболее публичным был случай компании Lyft, которая перешла на Salt). В наши дни такие масштабные сценарии, похоже, лучше обрабатываются с помощью многоуровневой сети Chef или Salt.

В нише небольших компаний системы Ansible и Salt создают серьезную конкуренцию своими относительно низкими барьерами для входа. Как уже говорилось, система Puppet не является сложной. Однако она несет некоторый исторический багаж, который, как правило, мешает новичкам. Например, в ядро Puppet встроено относительно небольшое количество операций. Для дополнения набора базовых конфигураций большинству организаций придется искать модули, предоставленные пользователями.

По нашему субъективному впечатлению, система Puppet пережила несколько фальстартов на ранней стадии своего проектирования и развития. Хотя разработчики системы Puppet интенсивно работали над исправлением этих недостатков, история и обратная совместимость — неизбежные спутники текущего продукта.

Ситуацию усложнило то, что система Puppet превратила золотое сокровище, язык Ruby, в кусок угля, которым является язык конфигурации Puppet. Вероятно, это было разумное решение в 2005 г., когда будущее языка Ruby было неясным, а каркас Rails еще не появился на сцене и не завоевал популярность. В наши дни язык конфигурации Puppet просто кажется абсурдным.

Ни одна из этих проблем не является непреодолимым препятствием, но у системы Puppet, похоже, нет ясного и убедительного преимущества, которое могло бы уравновесить ее недостатки. По нашим данным, за последние несколько лет не появилось ни одного сравнительного обзора, рекомендующего систему Puppet.

Конечно, если вы унаследовали существующую инсталляцию системы Puppet, нет необходимости начинать искать ее немедленную замену. Система Puppet работает отлично; различия между этими системами в основном являются вопросом стиля и дополнительных преимуществ.

Общие комментарии по поводу систем Ansible и Salt

Ansible и Salt — это хорошие системы, и мы рекомендуем один из этих вариантов для большинства организаций.

Мы более подробно рассмотрим обе эти системы в разделах "Введение в систему Ansible" и "Введение в систему Salt". В каждом из этих разделов рассматривается синтаксис конфигурации системы и общий стиль повседневного использования.

Ansible и Salt выглядят обманчиво похожими внешне, главным образом потому, что они по умолчанию используют YAML и Jinja в качестве своих форматов. Однако под капотом они совершенно разные. Соответственно, мы отложим сравнение систем Ansible и Salt, пока не обсудим их более подробно. Однако, прежде чем изучить сами системы, кратко обсудим формат YAML.

Ода YAML

Как упоминалось ранее, формат YAML является просто альтернативным синтаксисом для формата JSON. Например, формат YAML для системы Ansible[6]

```
- name: Install cpdf on cloud servers
  hosts: cloud
  become: yes
  tasks:
    - name: Install OCAML packages
      package: name={{ item }} state=present
      with_items:
        - gmake
        - ocaml
        - ocaml-opam
```

так отображается в формат JSON:

```
[{
   "name": "Inst  all cpdf on cloud servers",
   "hosts": "cloud",
   "become": "yes",
   "tasks": [{
      "name": "Install OCAML packages",
      "package": {
         "name": "{{ item }}",
         "state": "present",
      },
      "with_items": [ "gmake", "ocaml", "ocaml-opam" ]
    }]
}]
```

В мире JSON списки обрамляются квадратными скобками, а хеши — фигурными. Двоеточие отделяет хэш-ключ от его значения. Эти разделители могут появляться и в

[6]Теоретически документ в формате YAML должен начинаться с трех тире в первой строке, и документация Ansible часто следует этому соглашению. Однако эта начальная строка документа YAML — обычный рудимент. Насколько нам известно, ее можно безопасно исключить во всех случаях.

формате YAML, но YAML, помимо этого, понимает отступы для указания структуры, как в языке Python. Синтаксис YAML отмечает элементы в списке префиксом в виде тире.

Найдите время, чтобы проверить, что вы действительно поняли, как приведенный выше пример YAML отображается в формат JSON, потому что системы Ansible и Salt на самом деле основаны на формате JSON. YAML — это всего лишь сокращение. Далее мы рассмотрим систему Ansible, поскольку ее версия YAML более своеобразна, но большинство общих утверждений относится и к системе Salt.[7]

Очевидно, версия YAML более читабельна, чем версия JSON. Проблема заключается не в формате YAML как таковом, а скорее в попытке выразить сложные системы управления конфигурацией в форме JSON.

Формат YAML хорош для представления простых структур данных, но это не инструмент, который хорошо масштабируется до произвольной сложности. Когда в модели появляются изъяны, они должны подвергаться множеству специальных исправлений.

В приведенном выше примере уже содержится такое исправление. Вы заметили это?

```
package: name={{ item }} state=present
```

Не обращайте внимания на часть `{{ item }}`; это просто расширение Jinja. Проблема кроется в синтаксисе *имя=значение*, который на самом деле является просто нестандартным сокращением для определения подхеша.

```
package:
   name: {{ item }}
   state: present
```

Или нет? На самом деле нет, потому что система Ansible не допускает хеш-значения, которые начинаются с расширения Jinja. Это выражение Jinja теперь должно включать в себя кавычки:

```
package:

   name: "{{ item }}"
   state: present
```

А что если операция описывается в виде аргумента "свободной формы"?

```
- name: Cry for help
  shell: echo "Please, sir, I just want the syntax to be consistent"
  args:
    warn: no
```

Визуально это выглядит не так уж плохо. Но подумайте о том, что на самом деле происходит: `shell` — это тип операции, а `warn` — параметр операции типа `shell`, так же как `state` — это параметр операции `package` в предыдущем примере. Итак, что там делает дополнительный словарь `args`?

Дело в том, что операции типа `shell` обычно передается в качестве основного аргумента сложная строка (для запуска команды оболочки), поэтому она была сделана специальным типом операции, которой передается именно строка, а не хеш. Словарь `args` на самом деле является свойством оболочки объекта задачи, а не операцией типа `shell`.

[7]Опять же, приверженцы системы Salt будут возражать, что Salt нельзя критиковать за формат YAML и язык Jinja, потому что у нее нет реальных зависимостей от этих систем. Вы можете использовать любую из нескольких альтернатив. Все верно. В то же время это очень похоже на ситуацию, когда вы не хотите нести ответственность за правительство страны, потому что вы за него не голосовали.

Его содержимое тайно загружается в операцию `shell` от вашего имени, чтобы выполнить всю работу.

Нет проблем; сохраняйте спокойствие и продолжайте. Но эта путаница проявляется уже в относительно простом примере.

Проблема не в этом конкретном сценарии. Это постоянная смесь ограничений, двусмысленностей и компромиссов, которые необходимы для принудительного преобразования данных конфигурации в формат JSON. Передается ли этот конкретный аргумент операции? Состоянию? Привязке? Поскольку иерархия JSON довольно большая, ответ редко бывает очевиден.

Посмотрите еще раз на раздел `Install cpdf on cloud servers` в приведенном выше примере. Очевидно ли, что ключевое слово `with_items` должно стоять на том же уровне, что и `package`, а не на том же уровне, что и `name` и `state` (которые на самом деле логически ниже `package`)? Возможно нет.

Основополагающий замысел формата YAML заслуживает похвалы: используйте существующий формат, который уже знают люди, и представляйте информацию о конфигурации как данные вместо кода. Тем не менее эти системы имеют синтаксические изъяны, которые, вероятно, не будут устранены в реальном языке программирования.[8]

23.5. Введение в систему Ansible

Система Ansible не имеет серверного демона и не устанавливает на клиентских компьютерах никакого программного обеспечения. Таким образом, на самом деле это всего лишь набор команд (в первую очередь, **`ansible-playbook`**, **`ansible-vault`** и **`ansible`**), которые необходимо инсталлировать в любой системе, из которой вы хотите управлять клиентами.

🕮 Дополнительную информацию о репозитории EPEL см. в разделе 7.5.

Стандартные пакеты уровня операционной системы широко доступны для Ansible, хотя имена пакетов варьируются от системы к системе. В системах RHEL и CentOS необходимо убедиться, что в серверных системах включен репозиторий EPEL. Как и в большинстве случаев, пакеты, специфичные для операционной системы, часто оказываются несколько устаревшими по отношению к основному коду. Если вы не планируете отказаться от управления пакетами, систему Ansible легко установить из репозитория GitHub (`ansible/ansible`) или с помощью команды **`pip`**.[9]

По умолчанию основным файлом конфигурации Ansible является файл **`/etc/ansible/ansible.cfg`**. (Как и для большинства устанавливаемых пакетов, в системе FreeBSD каталог **`ansible`** перемещен в каталог **`/usr/local/etc`**.) Файл **`ansible.cfg`** по умолчанию является коротким и простым. Единственное изменение, которое мы рекомендуем, — добавить следующие строки в его конец.[10]

---

[8] Несмотря на слабость формата YAML, используемого в системах управления конфигурацией, его спецификация на самом деле довольно длинная. Фактически она длиннее спецификации всего языка программирования Go.

[9] Программа **`pip`** — это менеджер пакетов для языка Python. Попробуйте выполнить команду **`pip install ansible`**, чтобы загрузить последнюю версию из репозитория PyPI (Python Package Index). Возможно, вам понадобится сначала установить программу **`pip`** с помощью вашей системы управления дистрибутивами пакетов.

[10] Любопытно, что в файле **`ansible.cfg`**, а также в нескольких других файлах конфигурации Ansible, используется формат **`.ini`**, а не YAML. Мы не знаем причины этого явления.

```
[ssh_connection]
pipelining = true
```

Эти строки включают конвейерную обработку — функцию SSH, которая значительно повышает производительность. Конвейерная обработка требует, чтобы команда **sudo** на клиентах не настраивалась на использование интерактивных терминалов; такое поведение программы установлено по умолчанию.

🕮 Запуск программы **sudo** без терминала управления описан в разделе 3.2.

Если данные конфигурации сохранены в файле **/etc/ansible**, то вы должны использовать программу **sudo** для внесения изменений. При этом вы привязываете себя к одному конкретному серверу. Кроме того, вы можете легко настроить Ansible для использования вашей собственной учетной записи. Сервер просто запускает оболочку **ssh** для доступа к другим системам, поэтому привилегии пользователя `root` не нужны, если вам не нужно запускать привилегированные команды на стороне сервера.

К счастью, система Ansible упрощает объединение системных и личных конфигураций. Не удаляйте общесистемную конфигурацию; просто затените ее, создав файл **~/.ansible.cfg**, и укажите в нем расположение вашего каталога ресурсов и ролей.

```
[defaults]
inventory = ./hosts
roles_path = ./roles
```

Параметр `inventory` — это список клиентских систем, а `roles` — это пакеты, абстрагирующие различные аспекты конфигурации клиента. В ближайшее время мы вернемся к обоим этим темам.

Здесь мы определяем оба местоположения как относительные пути, которые предполагают, что вы будете использовать команду **cd** для перехода в каталог, содержащий вашу копию базы конфигурации, и что вы будете следовать указанным соглашениям об именах. Если вы предпочитаете использовать абсолютные пути, то учтите, что система Ansible также понимает обозначение ~, используемое оболочкой для рабочих каталогов пользователей. (Система Ansible позволяет применять символ ~ почти везде.)

Пример использования системы Ansible

Прежде чем углубиться в детали, мы сначала рассмотрим небольшой пример, демонстрирующий несколько основных операций Ansible.

В следующем списке шагов мы сначала установим и настроим программу **sudo** в новой системе (так как она может потребоваться, например, при работе с системой FreeBSD, в которой по умолчанию программа **sudo** не установлена).

1. Установите пакет **sudo**.
2. Скопируйте стандартный файл **sudoers** с сервера и установите его локально.
3. Убедитесь, что файлу **sudoers** назначены соответствующие права доступа и владельцы.
4. Создайте группу UNIX с именем **sudo**.
5. Добавьте все учетные записи системных администраторов на локальном компьютере в группу **sudo**.

Эти шаги реализует приведенный ниже код Ansible. Поскольку этот код предназначен для иллюстрации нескольких свойств системы Ansible, его не следует считать характерным примером.

```
- name: Install sudo package
  package: name=sudo state=present

- name: Install sudoers file
  template:
    dest: "{{ sudoers_path }}"
    src: sudoers.j2
    owner: root
    group: wheel
    mode: 0640

- name: Create sudo group
  group: name=sudo state=present

- name: Get current list of usernames
  shell: "cut -d: -f1 /etc/passwd"
  register: userlist

- name: Add administrators to the sudo group
  user: name={{ item }} groups=sudo append=true
  with_items: "{{ admins }}"
  when: "{{ item in userlist.stdout_lines }}"
```

Операторы обрабатываются по порядку, так же, как и в сценарии оболочки.

Выражения, заключенные в двойные фигурные скобки (например, `"{{ admins }}"`), являются расширениями переменных. Система Ansible интерполирует факты аналогичным образом. Такая гибкость управления параметрами является общей характеристикой систем управления конфигурацией, и это одно из их основных преимуществ перед обычными сценариями. Вы определяете общую процедуру в одном месте, а параметры конфигурации — в другом. Затем система CM сводит воедино глобальную спецификацию и гарантирует, что соответствующие параметры будут применены к каждому целевому хосту.

Файл **`sudoers.j2`** является шаблоном Jinja2, который расширяется, чтобы стать файлом **`sudoers`** на целевой машине. Шаблон может состоять из статического текста или может иметь внутреннюю логику и собственные переменные.

Шаблоны обычно хранятся вместе с конфигурациями в том же репозитории Git, что позволяет совершать все сразу при использовании конфигураций. Нет необходимости поддерживать отдельный файловый сервер, из которого можно скопировать шаблоны. Система управления конфигурацией использует для инсталляции шаблонов существующий доступ к целевому хосту, поэтому управление учетными данными необходимо настраивать только один раз.

Мы срезали пару острых углов. Пользовательский модуль Ansible, используемый здесь для добавления учетных записей системных администраторов в UNIX-группу `sudo`, обычно гарантирует, что указанная учетная запись существует, и создает учетную запись, если ее ранее не было. В данном случае мы хотим воздействовать только на уже существующие учетные записи, поэтому вынуждены вручную проверять наличие каждой учетной записи, прежде чем разрешить пользователю изменять ее.[11]

[11] В типичном сценарии система управления конфигурацией отвечает за настройку учетных записей администраторов, а также доступ к программе **sudo**. В спецификациях конфигурации для обеих функций, скорее всего, будет указана ссылка на одну и ту же переменную *admins*, и поэтому конфликт невозможен, так что нет необходимости проверять каждое имя учетной записи.

В конфигурации выполняется команда оболочки **cut -d: -f1 /etc/passwd** для получения списка существующих учетных записей и присваивания его переменной `userlist`. По сути это похоже на строку сценария оболочки **sh**

```
userlist=$(cut -d: -f1 /etc/passwd)
```

Каждая учетная запись, указанная в переменной `admins` (`with_items: "{{ admins }}"`), обрабатывается отдельно. В свою очередь, имя текущей обрабатываемой учетной записи присваивается переменной `item`. (Имя переменой `item` — это просто соглашение Ansible, в конфигурации оно нигде не задается.) Для каждой учетной записи, присутствующей в выводе команды **cut** (директива `when`), вызывается директива `user`.

Мы не показали еще несколько дополнительных аспектов, привязывающих эту конфигурацию к определенному набору целевых хостов и сообщающих системе Ansible, чтобы изменения были внесены от имени пользователя `root`. Когда мы активируем эту привязку (путем выполнения команды **ansible-playbook example.yml**), система Ansible начинает работать и пытается настроить несколько целевых хостов одновременно. Если какая-либо операция не удалась, система Ansible сообщает об ошибке и перестает работать на том хосте, где была обнаружена ошибка. Другие хосты могут продолжать работу до тех пор, пока она вся не будет выполнена.

Настройка клиента

Для управления конфигурацией каждого клиента необходимо иметь три вещи:

- доступ к протоколу SSH;
- разрешение на доступ к **sudo**[12];
- интерпретатор языка Python 2.

Если клиент является облачным сервером Linux, он может быть доступен системе Ansible автоматически. Такие системы, как FreeBSD, в которых не установлены по умолчанию **sudo** или интерпретатор Python, могут нуждаться в немного более тонкой настройке, но вы можете выполнить часть работы в процессе начальной загрузки с помощью системы Ansible и операций `raw`, которая запускает команды удаленно, без привычной оболочки Python. Кроме того, вы можете просто написать собственный сценарий начальной загрузки.

При настройке клиентов Ansible необходимо сделать выбор из нескольких вариантов. Мы предлагаем разумный план выбора в разделе "Параметры доступа в системе Ansible", но пока давайте предположим, что вы создали на клиентской машине выделенного пользователя `ansible`, что соответствующий SSH-ключ находится в вашем стандартном наборе и что вы готовы ввести пароль **sudo** вручную.

Клиенты не представляют себя системе Ansible, поэтому вам нужно добавить их в список хостов Ansible. По умолчанию этот список хранится в отдельном файле с именем **/etc/ansible/hosts**. Одной из приятных особенностей системы Ansible является то, что вы можете заменить любой текстовый конфигурационный файл каталогом с тем же именем. Система Ansible затем объединяет содержимое файлов, содержащихся в каталоге. Эта функция полезна для структурирования вашей базы конфигурации, но также является способом включения динамической информации: если конкретный файл явля-

[12]Система Ansible фактически не требует доступа к программе **sudo**. Это необходимо, только если вы хотите выполнять привилегированные операции.

ется исполняемым, Ansible запускает его и перехватывает результаты вывода, а не читает файл напрямую. [13]

Эта возможность агрегации настолько полезна и так часто используется, что мы рекомендуем обходить этап обработки большинства файлов конфигурации и переходить непосредственно к каталогам. Например, мы можем определить клиент системы Ansible, добавив следующую строку в файл **/etc/ansible/hosts/static** (или в **~/hosts/static** в личной конфигурации):

```
new-client.example.com ansible_user=ansible
```



В системе FreeBSD интерпретатор Python устанавливается в необычное место, поэтому системе Ansible необходимо сообщить следующую информацию.

```
freebsd.example.com ansible_python_interpreter=/usr/local/bin/python
    ansible_user=ansible
```

Все это должно быть записано в одной строке файла **hosts**. (Ниже мы покажем лучший способ задать значения этих переменных, но данный метод является просто обобщением той же идеи.)

Чтобы проверить подключение к новому хосту, выполните операцию `setup`, которая возвращает каталог фактов клиента.

```
$ ansible new-client.example.com -m setup
new-client.example.com | SUCCESS => {
  "ansible_facts": {
    "ansible_all_ipv4_addresses": [
       "172.31.25.123"
    ],
    ...
<Более 200 строк пропущено>
```

Имя `setup` кажется нам неудачным, поскольку явная настройка не требуется. Если хотите, можно перейти непосредственно к действительным операциям конфигурации. Кроме того, вы можете запускать операцию `setup` столько раз, сколько будет необходимо для просмотра каталога фактов клиента.

Убедитесь, что эскалация привилегий с помощью программы **sudo** также работает правильно.

```
$ ansible new-client.example.com -a whoami --become --ask-become-pass
SUDO password: <password>
new-client.example.com | SUCCESS | rc=0 >>
root
```

Здесь операция `command`, которая запускает команды оболочки, задается по умолчанию. Мы могли бы явно написать **-m command** с эквивалентными результатами. Флаг **-a** вводит параметры операции; в данном случае это фактическая команда, которая должна быть выполнена.

Параметр `Become`, принятый для эскалации привилегий, имеет странное название в системе Ansible; с точки зрения системы вы *становитесь* (become) другим пользователем. По умолчанию этим другим пользователем является пользователь `root`, но вы можете указать другой вариант с помощью флага **-u**. К сожалению, вам нужно явно указать в командной строке системы Ansible специальный флаг, чтобы она запросила у вас

[13] На самом деле система Ansible делает еще больше. Она полностью игнорирует определенные типы файлов, например файлы **.ini**. Таким образом, вы можете не только добавлять их в сценарии, но и создавать файлы конфигурации для сценариев.

пароль **sudo** (это делается с помощью опции **--ask-become-pass**), причем это делается независимо от того, запрашивает удаленная система пароль или нет.

Группы клиентов

Группы также можно определить в каталоге **hosts**, хотя синтаксис при этом может стать неудобным.

```
client-four.example.com

[webservers]
client-one.example.com
client-two.example.com
[dbservers]
client-one.example.com
client-three.example.com
```

Если это выглядит не так уж плохо, то только потому, что мы обошли основные проблемные области. Формат **.ini** является текстовым, поэтому необходимы некоторые трюки, если вы хотите определить иерархические группы или добавить дополнения непосредственно в файл **hosts** (например, присвоить переменные для группы).

Однако эти функции на самом деле не так важны на практике. Обратите внимание: нам пришлось указать хост `client-four` в верхней части файла, потому что он не включен в какие-либо группы. Мы не можем просто добавить этот хост в конец файла **hosts**, поскольку это сделает хост `client-four` членом группы `dbservers`, даже если бы мы добавили пустую строку в качестве разделителя.

Это еще одна причина, почему каталоги конфигурации являются полезными. На практике мы, вероятно, захотим поместить каждое определение группы в отдельный файл. Система Ansible позволяет свободно смешивать имена клиентов и групп в командных строках и в базе конфигурации. Ни один из них никак специально не выделяется, и оба могут быть подвергнуты подстановке. Совместимость с регулярным выражением также доступна для обеих категорий имен; просто укажите в начале шаблона префикс ~. Существует также набор алгебр нотации для разных объединений когорт клиентов.

Например, в следующей команде используется выражение подстановки для выбора группы `webservers`. В ней применяется операция `ping` к каждому члену этой группы.

```
$ ansible 'web*' -m ping
client-one.example.com | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
client-two.example.com | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Присваивание переменных

Как мы видели ранее, значения переменных можно присваивать в файлах инвентаризации. Но это очень грубый прием; не делайте этого.

Каждый хост и группа могут иметь собственную коллекцию определений переменных в формате YAML. По умолчанию эти определения хранятся в каталогах **/etc/ansible/host_vars** и **/etc/ansible/group_vars** в файлах, названных по имени хо-

ста или группы. Если хотите, вы можете использовать суффикс **.yml**; система Ansible найдет соответствующие файлы в любом случае.

Как и в других конфигурациях Ansible, эти файлы могут быть преобразованы в каталоги, если вы хотите добавить дополнительную структуру или сценарии. Система Ansible выполняет обычную процедуру игнорирования файлов конфигурации, запуска сценариев и объединения всех результатов в окончательный пакет.

Система Ansible автоматически определяет группу `all`. Как и другие группы, группа `all` может иметь собственные групповые переменные. Например, если вы стандартизируете использование учетных записей клиентов с именем `ansible` для управления конфигурацией, это хороший повод, чтобы задать глобальную конфигурацию (например, в каталоге **group_vars/all/basics**):

```
ansible_user: ansible
```

Если для переменной существует несколько объявлений значения, система Ansible выбирает окончательное значение в соответствии с правилами приоритета, а не по порядку объявления. В настоящее время система Ansible имеет 14 различных категорий приоритетов, но важным моментом в этом случае является то, что переменные хоста перекрывают переменные группы.

Конфликты между перекрывающимися группами решаются случайным образом, что может привести к непоследовательному поведению и сложной отладке. Попытайтесь структурировать объявления переменных, чтобы не было возможности перекрытия.

Динамические и вычисляемые группы клиентов

Система группировки Ansible действительно проявляет себя, когда в действие вступают динамические сценарии. Например, сценарии динамической инвентаризации, используемые с поставщиками облачных вычислений, не просто выводят список всех доступных серверов. Они также разделяют и перемещают эти серверы в специальные группы в соответствии с метаданными из облака.

Например, служба EC2 компании Amazon позволяет назначать произвольные теги для каждого экземпляра. Вы можете назначить тег `web-server` каждому экземпляру, которому нужен стек NGINX, и тег `dbserver` — каждому экземпляру, которому нужна система PostgreSQL. В результате сценарий динамической инвентаризации **ec2.py** создаст группы с именами `tag_webserver` и `tag_dbserver`. Эти группы могут иметь собственные групповые переменные и могут быть названы в привязках ("плей-листах"), как и любая другая группа.

Ситуация становится более сложной, когда дело касается группировки клиентов по критериям, внутренним по отношению к системе Ansible, таким как значения фактов. Вы не можете сделать это напрямую. Вместо этого вы можете использовать целевые наборы сценариев (playbooks) для более широких групп (например, `all`) и применять условные выражения к отдельным операциям, которые, когда соответствующие условия не применяются, пропускают операцию.

Например, следующий набор инструкций гарантирует, что файл **/etc/rc.conf** содержит строку для настройки имени хоста на каждом клиенте системы FreeBSD.

```
- name: Set hostname at startup on FreeBSD systems
  hosts: all
  tasks:
     - lineinfile:
         dest: /etc/rc.conf
         line: hostname="{{ hostname }}"
```

```
      regexp: ^hostname
    when: ansible_os_family == "FreeBSD"
```

(Если последняя строка выглядит так, как будто она нуждается в двойных фигурных скобках, значит ваш инстинкт вас не подводит. На самом деле это всего лишь немного синтаксического сахара Ansible, позволяющего поддерживать конфигурацию в порядке. Спецификации `when` всегда являются выражениями Jinja, поэтому система Ansible окружает их содержимое двойными фигурными скобками автоматически. Эта функция полезна, но это всего лишь одно из довольно обширного списка отклонений в синтаксическом анализе формата YAML в системе Ansible.)

В этом примере к каждому хосту в инвентарном перечне применяется операция `lineinfile`. Однако благодаря спецификации `when` на самом деле эту операцию выполняют только хосты под управлением системы FreeBSD. Этот подход работает отлично, но он не превращает хосты FreeBSD в настоящую группу. Например, они не могут иметь нормальную запись **`group_vars`**, хотя вы можете имитировать этот эффект с помощью некоторых трюков.

Структурно предпочтительной, но немного более многословной альтернативой является использование операции `group_by`, которая выполняется локально и классифицирует хосты в соответствии с произвольным значением ключа, для которого вы назначаете шаблон:

```
- name: Group hosts by OS type
  hosts: all
  tasks:
    - group_by: key={{ ansible_os_family }}

- name: Set hostname at startup on FreeBSD systems
  hosts: FreeBSD
  tasks:
    - lineinfile:
        dest: /etc/rc.conf
        line: hostname="{{ hostname }}"
        regexp: ^hostname
```

Основной план аналогичен, но классификация происходит в отдельном сценарии (термин Ansible для того, что мы называем связыванием). Затем мы начинаем новый сценарий, чтобы указать другой набор целевых хостов, на этот раз используя группу FreeBSD, определенную для нас в первом сценарии.

Преимущество использования операции `group_by` заключается в том, что мы выполняем классификацию только один раз. После этого мы можем назначить любое количество задач из второго сценария, будучи уверенными в том, что мы нацелены только на предполагаемых клиентов.

Списки задач

В системе Ansible операции называются *задачами*, а набор задач, перечисленных в отдельном файле, называется *списком задач*. Как и остальные части конфигурации Ansible, за небольшим исключением, списки задач имеют формат YAML, поэтому файлы имеют суффикс **`.yml`**.

Связывание списков задач с конкретными хостами выполняется на объектах более высокого уровня, называемых *файлами сценариев* (playbooks), которые будут описаны чуть ниже. Теперь давайте сосредоточимся на самих операциях и не будем беспокоиться о том, как они применяются к конкретному хосту.

В качестве примера мы снова рассмотрим пример установки **sudo** с немного иным фокусом и реализацией. На этот раз мы создаем учетные записи администратора с нуля и назначаем каждой записи собственную группу UNIX с тем же именем. Затем мы создаем файл **sudoers**, в котором явно перечислены администраторы (вместо того, чтобы просто назначать привилегии группе **sudo** UNIX).[14]

Для управления этими операциями необходимы некоторые входные данные: в частности, расположение файла **sudoers**, а также имена и логины администраторов. Мы должны поместить эту информацию в отдельный файл переменной, скажем, **group_vars/all/admins.yml**.

```
sudoers_path: /etc/sudoers
admins:
   - { username: manny, fullname: Manny Calavera }
   - { username: moe, fullname: Moe Money }
```

Значение `admins` — это массив хешей; мы перебираем все элементы этого массива для создания списка всех учетных записей. Вот как выглядит полный список задач.

```
- name: Install sudo package
  package: name=sudo state=present

- name: Create personal groups for admins
  group: name={{ item.username }}
  with_items: "{{ admins }}"

- name: Create admin accounts
  user:
    name: "{{ item.username }}"
    comment: "{{ item.fullname }}"
    group: "{{ item.username }}"
    groups: wheel
  with_items: "{{ admins }}"

- name: Install sudoers file
  template:
    dest: "{{ sudoers_path }}"
    src: templates/sudoers.j2
    owner: root
    group: wheel
    mode: 0600
```

С точки зрения форматов YAML и JSON задачи образуют список. Каждое тире в левом поле начинает новую задачу, которая представлена хешем.

В этом примере каждая задача имеет поле `name`, которое описывает ее функцию на английском языке. Названия с формальной точки зрения необязательны, но если вы их не включите, то система Ansible будет очень мало сообщать о том, что она делает при запуске конфигурации (кроме вывода имен модулей: пакета, группы и т.д.).

Каждая задача должна иметь среди своих ключей имя точно одного операционного модуля. Значение этого ключа само по себе является хешем, в котором указаны параметры операции. Параметрам, которым явно не заданы значения, назначаются значения по умолчанию.

[14]Файл задачи или состояния обычно должен иметь четко определенный домен и четкую цель, поэтому данный пример является разновидностью беспорядка. Мы выбрали эти операции, чтобы проиллюстрировать некоторые общие моменты, а не как пример правильной базовой структуры конфигурации.

Обозначение

```
- name: Install sudo package
  package: name=sudo state=present
```

является расширением формата YAML в системе Ansible, которое по существу эквивалентно следующему коду:

```
- name: Install sudo package
  package:
    name:  sudo
    state: present
```

У таких операций, как `shell`, есть одна особенность — аргументы в свободном виде, но мы не будем переделывать этот код (см. раздел "Ода YAML").

Однострочный формат не только более компактен, но также позволяет устанавливать параметры, значения которых представляют собой выражения Jinja без кавычек, как видно из задачи, которая создает персональные группы для администраторов. В нормальном синтаксисе выражение Jinja не может появляться в начале значения, если все значение не указано в кавычках. Использование кавычек совершенно безвредно, но добавляет визуальный шум. Несмотря на внешний вид, кавычки не превращают значение в строку.

Теперь мы готовы описать некоторые из наиболее примечательных аспектов этого примера списка задач в следующих разделах.

Параметры состояния

В системе Ansible операционные модули часто могут выполнять несколько различных задач в зависимости от состояния, которое вы запрашиваете. Например, для модуля `package` состояние `state=present` инсталлирует пакет, `state=absent` удаляет пакет, а `state=latest` гарантирует, что пакет инсталлирован и обновлен. Операции часто ищут разные наборы параметров в зависимости от вызываемого состояния.

В некоторых случаях (например, когда модуль `service` с состоянием `state=restarted` перезапускает демон) эта модель немного отклоняется от того, что обычно понимают как "состояние", но в целом она работает хорошо. Состояние может быть пропущено (как показано здесь при создании группы `sudo`), и в этом случае оно принимает значение по умолчанию, обычно что-то положительное и расширяющее возможности, например `present`, `configured` или `running`.

Итерация

Ключевое слово `with_items` — это итерационная конструкция, которая повторяет задачу для каждого элемента, к которому она относится. Ниже приведена еще одна копия двух задач в нашем примере, использующих ключевое слово `with_items`.

```
- name: Create personal groups for admins
  group: name={{ item.username }}
  with_items: "{{ admins }}"

- name: Create admin accounts
  user:
     name: "{{ item.username }}"
     comment: "{{ item.fullname }}"
     group: "{{ item.username }}"
     groups: wheel
  with_items: "{{ admins }}"
```

Обратите внимание на то, что with_items является атрибутом задачи, а не операцией, выполняемой задачей. При каждом прохождении через цикл система Ansible устанавливает значение элемента item равным одному из элементов, взятому из параметра with_items. В нашем случае мы присвоили переменной admins список хешей, поэтому переменная item всегда является хешем. Обозначение item.username является сокращением выражения item['username']. Используйте тот вариант, который вы предпочитаете.

Каждая из этих задач перебирает все элементы массива admins по отдельности. За один проход создаются группы UNIX, а за другой — учетные записи пользователей. Хотя в системе Ansible определен механизм группировки задач (называемый блоком), в этой конструкции, к сожалению, не поддерживается ключевое слово with_items.

Если вам действительно нужен эффект одного цикла, в котором последовательно выполняется несколько задач, вы можете добиться этого, переместив тело цикла в отдельный файл и включив его в основной список задач.

```
- include: sudo-subtasks.yml
  with_items: "{{ admins }}"
```

Ключевое слово with_items — это не единственная разновидность цикла, доступная в системе Ansible. Существуют также формы циклов, предназначенные для итерации по хешам (называемые "словарями" в языке Python), спискам файлов и шаблонам подстановок.

Взаимодействие с Jinja

Документация Ansible не очень конкретна в отношении взаимодействия формата YAML и шаблонов Jinja, но важно понять детали. Как показывают конструкции, подобные with_items, Jinja — это не просто препроцессор, который применяется к файлу, прежде чем он будет передан механизму YAML (как в случае с системой Salt). Фактически система Ansible выполняет синтаксический разбор формата YAML, оставляя нетронутыми выражениями Jinja. Затем препроцессор Jinja разворачивает каждое строковое значение непосредственно перед его использованием. На каждой итерации параметры повторных операций вычисляются заново.

Препроцессор Jinja имеет собственные структуры управления, включая циклы и условные выражения. Однако они по своей сути несовместимы с архитектурой отложенных вычислений в системе Ansible и поэтому не разрешены в YAML-файлах системы Ansible (хотя их можно использовать в шаблонах). Простые конструкции, такие как when и with_items, — это не только декорации для эквивалентных шаблонов Jinja. Они представляют собой совершенно другой подход к структурированию конфигурации.

Визуализация шаблона

Система Ansible использует язык шаблонов Jinja2 как для добавления динамических функций в файлы YAML, так и для создания шаблонов для файлов конфигурации, установленных модулем template. В следующем примере мы используем шаблон для настройки файла **sudoers**. Ниже снова приведены определения переменных для справки.

```
sudoers_path: /etc/sudoers
admins:
  - { username: manny, fullname: Manny Calavera }
  - { username: moe, fullname: Moe Money }
```

Код задачи имеет следующий вид.

```
- name: Install sudoers file
  template:
```

```
    dest: "{{ sudoers_path }}"
    src: templates/sudoers.j2
    owner: root
    group: wheel
    mode: 0600
```

Файл **sudoers.j2** представляет собой сочетание простого текста и кода Jinja2 для динамических сущностей. Например, вот схематический пример, который дает привилегии "**sudo ALL**" каждому администратору.

```
Defaults env_keep += "HOME"

{% for admin in admins %}
{{ admin }} ALL=(ALL) ALL
{% endfor %}
```

Цикл **for**, заключенный в скобки **{% %}**, представляет собой синтаксис Jinja2. К сожалению, здесь нельзя использовать отступы, как в реальном языке программирования, потому что это приведет к тому, что они появятся в выводе шаблона.

Расширенная версия выглядит так.

```
Defaults env_keep += "HOME"

manny ALL=(ALL) ALL
moe ALL=(ALL) ALL
```

Обратите внимание на то, что переменные автоматически передаются в шаблоны. Их значения доступны для файлов конфигурации с точно такими же именами, которые используются для их определения; здесь нет ни префикса, ни дополнительной иерархии. Автоматически обнаруженные переменные фактов также находятся в пространстве имен верхнего уровня, но для предотвращения потенциальных конфликтов имен они начинаются с префикса `ansible_`.

Модуль Ansible для установки статических файлов называется `copy`. Однако вы можете обрабатывать все файлы конфигурации как шаблоны, даже если их содержимое изначально состоит из статического текста. Затем вы можете добавлять настройки без необходимости внесения изменений в код конфигурации; просто отредактируйте шаблон. Используйте модуль `copy` для двоичных и статических файлов, которые никогда не нуждаются в расширении, таких как открытые ключи.

Привязки: сценарии и файлы сценариев

Привязки — это механизм, посредством которого задачи ассоциируются с наборами клиентских машин. Объект привязки Ansible называется *сценарием* (play). Вот простой пример.

```
- name: Make sure NGINX is installed on web servers
  hosts: webservers
  tasks:
    - package: name=nginx state=present
```

Подобно тому, как несколько задач можно объединить для формирования списка задач, несколько сценариев образуют файл сценариев (playbook).

Как и в других системах, основными элементами привязки являются наборы хостов и задач. Однако система Ansible позволяет указать несколько дополнительных опций на уровне сценария. Они перечислены в табл. 23.3.

Таблица 23.3. Элементы сценария Ansible

Ключ	Формат	Что определяет
`name`	`string`	Имя для вывода на печать при выполнении сценария, можно опустить
`hosts`	`list, string`	Клиентские системы, на которых выполняются связанные задачи и роли
`vars`	`hash`	Значения переменных для установки в области видимости сценария
`vars_files`	`list`	Файлы для чтения значений переменных
`become*`	`strings`	Эскалация привилегий (например, **`sudo`**, см. ниже)
`tags`	`list`	Категории для выборочного исполнения (см. ниже)
`tasks`	`list`	Выполняемые операции; могут включать отдельные файлы
`handlers`	`list`	Операции, выполняемые в ответ на уведомление (`notify`)
`roles`	`list`	Привязки (роли) для вызова для этих хостов; см. ниже

Самыми важными в этой таблице являются элементы, связанные с переменными, причем не столько потому, что они появляются в самих сценариях, а потому, что они доступны практически везде, даже при выполнении операции `include`. Система Ansible может активировать один и тот же список задач или набор сценариев снова и снова с различными наборами значений переменных. Это очень похоже на определение функции (например, "создать учетную запись пользователя"), которая потом вызывается с различными наборами аргументов.

Система Ansible формализует эту систему с помощью реализации пакетов (называемых "ролями"), которые мы обсуждаем далее. Роль — это мощный инструмент, но по существу это всего лишь набор стандартизованных условностей, поэтому их легко понять.

Вот простой сценарий, демонстрирующий использование обработчиков.

```
- name: Update cow-clicker web app
  hosts: clickera,clickerb
  tasks:
    - name: rsync app files to /srv
      synchronize:
        mode: pull
        src: web-repo:~sites/cow-clicker
        dest: /srv/cow-clicker
        notify: restart nginx
  handlers:
    - name: restart nginx
      service: name=nginx state=restarted
```

Этот сценарий работает на хостах `clickera` и `clickerb`. Он копирует файлы из центрального (локального) репозитория, выполняя команду **`rsync`** (используя модуль `synchronize`), а затем перезагружает веб-сервер NGINX, если были сделаны какие-либо обновления.

Когда задача со спецификацией `notify` вносит изменения в систему, система Ansible запускает обработчик запрошенного имени. Сами обработчики — это просто задачи, но они объявлены в отдельном разделе сценария.

Основной единицей исполнения в системе Ansible являются файлы сценариев. Они запускаются с помощью команды **`ansible-playbook`**.

```
$ ansible-playbook global.yml --ask-sudo-pass
```

Система Ansible обрабатывает несколько хостов, выполняя задачу за задачей. Когда она читает файл сценариев, каждая задача выполняется параллельно на целевых хостах.

После того как каждый хост завершит задачу, система Ansible начинает следующую задачу. По умолчанию система Ansible запускает задачи одновременно на пяти хостах, но существует возможность установить другой предел с помощью флага **-f**.

При отладке часто бывает полезно включить аргумент **-vvvv** для увеличения объема вывода отладочной информации. Вы увидите точные команды, которые выполняются на удаленной системе, и реакцию системы на них.

Роли

Как мы писали в общих чертах, привязки (это наш термин) являются механизмом упаковки, определенным системой CM, который облегчает повторное и совместное использование фрагментов конфигурации.

В системе Ansible пакеты названы *ролями*, и они на самом деле не что иное, как структурированная система операций `include` и правил приоритета переменных. Они позволяют легко помещать определения переменных, списки задач и шаблоны, связанные с конфигурацией, в один каталог, что делает их доступными для повторного и совместного использования.

Каждая роль — это подкаталог каталога **roles**, который обычно находится на верхнем уровне базы конфигурации. Вы также можете добавить каталоги ролей для всей организации, установив переменную `role_path` в файле **ansible.cfg**, как было показано выше. Все известные каталоги ролей просматриваются всякий раз, когда вы включаете роль в файл сценариев.

Каталоги ролей могут иметь подкаталоги, показанные в табл. 23.4.

Таблица 23.4. Подкаталоги ролей Ansible

Подкаталог	Содержание
defaults	Значения по умолчанию для переопределяемых переменных
vars	Определения переменных (не переопределяемые, но могут ссылаться на переопределения)
tasks	Список задач (наборы операций)
handlers	Операции, которые реагируют на уведомления
files	Файлы данных (обычно используемые в качестве источника для операции `copy`)
templates	Шаблоны, которые должны обрабатываться препроцессором Jinja перед установкой
meta	Список запускаемых пакетов, необходимых для запуска конкретного пакета

Роли вызываются с помощью файлов сценариев и никак иначе. Система Ansible ищет файл **main.yml** в каждом из подкаталогов роли. Если он существует, содержимое автоматически включается в любой файл сценариев, который вызывает роль. Например, файл сценариев

```
- name: Set up cow-clicker app throughout East region
  hosts: web-servers-east
  roles:
    - cow-clicker
```

примерно эквивалентен следующему.

```
- name: Set up cow-clicker app throughout East region
  hosts: web-servers-east
  vars_files:
    - roles/cow-clicker/defaults/main.yml
    - roles/cow-clicker/vars/main.yml
  tasks:
```

```
    - include: roles/cow-clicker/tasks/main.yml
  handlers:
    - include: roles/cow-clicker/handlers/main.yml
```

Однако значения переменных из папки **default** не переопределяют значения, которые уже были установлены. Кроме того, система Ansible упрощает обращение к файлам из каталогов **files** и **templates**, а также включает все роли, упомянутые в качестве зависимостей в файле **meta/main.yml**.

Файлы, отличные от **main.yml**, игнорируются системой ролей, поэтому вы можете разбить конфигурацию на любые части и просто включить эти части в файл **main.yml**.

Система Ansible позволяет передавать набор значений переменных в конкретный экземпляр роли. При этом роль начинает действовать как своего рода параметризованная функция. Например, вы можете определить пакет, который используется для развертывания приложения Rails. Вы можете вызвать этот пакет несколько раз в файле сценариев, при каждом вызове предоставляя параметры для нового приложения.

```
- name: Install ULSAH Rails apps
  hosts: ulsah-server
  roles:
    - { role: rails_app, app_name: ulsah-reviews }
    - { role: rails_app, app_name: admin-com }
```

В этом примере роль `rails_app`, вероятно, будет зависеть от роли `nginx` или какого-либо другого веб-сервера, поэтому нет необходимости упоминать роль веб-сервера явно. Если вы хотите настроить установку веб-сервера, вы можете просто включить соответствующие значения переменных в вызов `rails_app`, и эти значения будут распространяться вниз.

Репозиторий открытой роли Ansible находится на сайте `galaxy.ansible.com`. Вы можете искать роли с помощью команды **ansible-galaxy**, но лучше использовать указанный веб-сайт. Он позволяет сортировать роли по рейтингу или количеству загрузок и легко переходить в репозиторий GitHub, где размещен фактический код для каждой роли. Для наиболее распространенных сценариев обычно доступно несколько ролей, поэтому стоит изучить код, чтобы определить, какая версия будет наилучшим образом удовлетворять ваши потребности.

После того как вы решились на реализацию роли, скопируйте файлы в каталог **roles**, выполнив команду **ansible-galaxy install**. Например, так.

```
$ ansible-galaxy install ANXS.postgresql
  - downloading role 'postgresql', owned by ANXS
  - downloading role from https://github.com/ANXS/postgresql/v1.4.0.tar.gz
  - extracting ANXS.postgresql to /etc/ansible/roles/ANXS.postgresql
  - ANXS.postgresql was installed successfully
```

Рекомендации по структурированию базы конфигурации

Большинство баз конфигурации организованы иерархически. Иначе говоря, различные части конфигурации загружаются в главный файл сценариев, который управляет глобальным состоянием. Тем не менее можно определить файлы сценариев для специальных задач, не связанных с глобальной схемой.

Старайтесь хранить списки задач и обработчиков за пределами файлов сценариев. Поместите их в отдельные файлы и интерполируйте с помощью операции `include`. Эта структура обеспечивает четкое разделение между привязками и поведением, а также вы-

равнивает приоритеты всех задач. Кроме того, хорошим стилем является полный отказ от самостоятельных списков задач и стандартизация ролей.

Иногда рекомендуется, чтобы один файл сценариев охватывал все задачи, относящиеся к каждой логически определенной группе хостов. Например, все роли и задачи, относящиеся к веб-серверам, должны быть включены в один файл сценариев **webserver.yml**. Такой подход позволяет избежать репликации групп хостов и обеспечивает четкий фокус управления для каждой группы хостов.

С другой стороны, следование этому правилу означает, что у вас нет прямого способа запуска части глобальной конфигурации даже для отладки. Система Ansible может выполнять только файлы сценариев; в ней нет простой команды, которая запускает определенный список задач на данном компьютере. Официальным решением этой проблемы является тегирование, которое отлично работает, но требует некоторой настройки. Вы можете включить поле `tags` в любую задачу или перед любой задачей, чтобы классифицировать ее. Чтобы указать подмножество тегов, которые вы хотите запустить, в командной строке используйте опцию **-t** команды **ansible-playbook**. В большинстве сценариев отладки следует также использовать параметр **-l**, чтобы ограничить выполнение указанным тестовым хостом.

Присваивайте теги на как можно более высоком уровне иерархии конфигурации. При нормальных обстоятельствах у вас не должно возникнуть соблазна назначать теги для отдельных задач. (Если вы это сделаете, это может быть признаком того, что конкретный список задач должен быть разделен на части.) Вместо этого добавьте теги в спецификацию `include` или `role`, которые включают в себя определенный список задач или ролей в конфигурации. После этого теги будут распространены на все включенные задачи.

Кроме того, вы можете просто с нуля создавать файлы сценариев, которые запускают части базы конфигурации на тестовом хосте. Настройка этих файлов сценариев — несложная работа, как и назначение тегов в целом.

Параметры доступа в системе Ansible

Система Ansible нуждается в доступе к службе SSH и программе **sudo** на каждой клиентской машине. Это кажется простым и естественным до тех пор, пока вы не учтете, что система управления конфигурацией содержит мастер-ключи для всей организации. Трудно обеспечить более высокую безопасность систем на базе демонов, чем безопасность учетной записи `root` на сервере конфигурации, но при продуманном планировании система Ansible может решить эту задачу.

Для простоты лучше всего использовать SSH-доступ через выделенную учетную запись, например `ansible`, которая имеет одно и то же имя на каждом клиенте. Эта учетная запись должна использовать простую оболочку и иметь минимальный файл конфигурации, имя которого начинается с точки.

На облачных серверах вы можете использовать стандартную учетную запись начальной загрузки (например, `ec2-user` на EC2) для управления со стороны Ansible. Просто убедитесь, что после первоначальной настройки учетная запись была правильно заблокирована и не разрешает, например, выполнение команды **su** для пользователя `root` без пароля.

У вас есть определенная гибкость в выборе фактической схемы системы безопасности. Но имейте в виду следующее.

- Системе Ansible требуется один мандат (пароль или закрытый ключ) для доступа к удаленной системе и другой — для повышения привилегий с помощью программы **sudo**. Правильные принципы безопасности предполагают, что это разные ман-

даты. Единый скомпрометированный мандат не должен предоставлять злоумышленнику доступ к учетной записи `root` целевой машины.[15]

- Если оба мандата хранятся в одном и том же месте с одинаковой формой защиты (шифрование, права доступа к файлам), они фактически представляют собой единый мандат.
- Мандаты можно повторно использовать на компьютерах, которые являются одноранговыми хостами (например, веб-серверами на ферме серверов), но не должно быть возможности использовать мандаты на одном сервере для доступа к более важному или даже совершенно другому серверу.
- Система Ansible имеет прозрачную поддержку зашифрованных данных с помощью команды **`ansible-vault`**, но только если данные содержатся в файле YAML или **`.ini`**.
- Администраторы могут запоминать всего несколько паролей.
- Неразумно требовать более одного пароля для данной операции.

Организации вырабатывают собственные правила, но мы предлагаем следующую систему в качестве надежного и удобного ориентира.

- Доступ к службе SSH контролируется парами ключей, которые используются только системой Ansible.
- Доступ к службе SSH по паролю запрещен в клиентских системах (путем установки параметра `PasswordAuthentication` равным `no` в файле **`/etc/ssh/sshd_config`**).
- Закрытые ключи SSH должны быть защищены парольной фразой (устанавливается с помощью команды **`ssh-keygen -p`**). Все личные ключи должны иметь одну и ту же парольную фразу.
- Закрытые ключи SSH должны храниться в известном месте на главной машине Ansible. Они не должны храниться в базе конфигурации, и администраторы не должны копировать их в другое место.
- Учетные записи на удаленных машинах для системы Ansible должны иметь случайные пароли UNIX, которые указываются в базе конфигурации в зашифрованном виде. Все они зашифрованы одной и той же парольной фразой, но она должна отличаться от парольной фразы, используемой для закрытых ключей SSH. Вам нужно будет добавить код Ansible, чтобы убедиться, что правильные пароли используются правильными клиентскими хостами.

В этой схеме оба набора мандатов шифруются, что делает их устойчивыми к простым нарушениям прав доступа к файлам. Этот слой косвенности также позволяет легко изменять основные фразы, не изменяя основные ключи.

Администраторы должны помнить только две фразы: парольную фразу, которая предоставляет доступ к секретным ключам SSH, и пароль хранилища Ansible, который позволяет системе расшифровывать пароли **`sudo`**, специфичные для хоста (а также любую другую конфиденциальную информацию, включенную в вашу конфигурационную базу).

[15] В некоторых организациях создаются клиентские учетные записи для системы Ansible с опцией `NOPASSWD` в файле **`sudoers`**, так что для этой учетной записи для запуска программы **`sudo`** не требуется пароль. Это ужасно небезопасная конфигурация. Если вы не можете заставить себя ввести пароль, то по крайней мере установите модуль агента PAM SSH и потребуйте переадресованный SSH-ключ для доступа к **`sudo`**. Дополнительную информацию о PAM см. в разделе 17.3.

Если вам нужны более специфичные права доступа администратора (что вполне вероятно), вы можете зашифровать несколько наборов мандатов с помощью разных парольных фраз. Если эти наборы имеют много общего (в отличие от непересекающихся), то каждому администратору не нужно будет помнить более двух парольных фраз.

В этой системе предполагается, что администраторы будут использовать команду **ssh-agent** для управления доступом к закрытым ключам. Все ключи могут быть активированы с помощью одной команды **ssh-add**, а пароль SSH необходимо вводить только один раз за сеанс. Для работы с системой, отличной от обычного мастера Ansible, администраторы могут использовать опцию SSH `ForwardAgent` для туннелирования ключей до машины, на которой выполняется работа. Вся другая информация о безопасности включена в саму конфигурационную базу.

🕮 Более подробную информацию о команде **ssh-agent** см. в разделе 27.7.

Команда **ssh-agent** и пересылка ключей являются настолько же безопасными, как и машины, на которых они запускаются. (На самом деле это немного не так: аналогично программе **sudo** с периодом отсрочки они настолько же безопасны, как и ваша личная учетная запись.) Однако риск смягчается ограничениями по времени и контексту. Используйте аргумент **-t** в командах **ssh-agent** или **ssh-add**, чтобы ограничить время действия активированных ключей и разорвать соединения, которые имеют доступ к переадресованным ключам, когда вы больше их не используете.

Если возможно, закрытые ключи никогда не должны развертываться в клиентских системах. Если клиентам нужен привилегированный доступ к контролируемым ресурсам (например, клонировать контролируемый репозиторий Git), используйте прокси-функции, встроенные в SSH и Ansible, или команду **ssh-agent** для временного доступа к закрытым ключам клиента без их копирования.

По какой-то причине система Ansible в настоящее время не может распознавать зашифрованные файлы в базе конфигурации и предлагать ввести парольную фразу для их дешифровки. Чтобы заставить систему сделать это, укажите аргумент **--ask-vault-pass** в командах **ansible-playbook** и **ansible**. Существует также опция **--vault-password-file** для неинтерактивного использования, но, конечно же, это снижает безопасность. Если вы решите использовать файл паролей, он должен быть доступен только для выделенной учетной записи системы Ansible.

23.6. Введение в систему Salt

Систему Salt часто называют SaltStack или Salt Open. Эти термины по существу взаимозаменяемы. Название ее поставщика — SaltStack, и поэтому название SaltStack используется как общий термин для обозначения полной линейки продуктов, включающей некоторые дополнительные корпоративные модули, которые мы не обсуждаем в этой книге. Тем не менее многие люди называют систему с открытым исходным кодом SaltStack.

Salt Open — это недавно появившееся название, которое обозначает только компоненты с открытым исходным кодом Salt. Но в настоящее время это название, похоже, не используется нигде вне сайта `saltstack.com`.

Система SaltStack поддерживает собственный репозиторий пакетов на сайте `repo.saltstack.com`, в котором размещаются обновленные пакеты для каждой системы управления пакетами Linux. Инструкции по добавлению репозитория к вашей конфигурации смотрите на веб-сайте. Некоторые дистрибутивы включают свободно распространяемые пакеты, но лучше всего перейти непосредственно к источнику.

Вам понадобится установить пакет **salt-master** на главном сервере конфигурации (мастер-сервер). Если у вас есть какие-либо договора с облачными провайдерами, также установите пакет **salt-cloud**. Он объединяет множество облачных провайдеров в стандартный интерфейс и упрощает процесс создания новых облачных серверов, управляемых системой Salt. Этот пакет по существу похож на собственные инструменты CLI для облачных провайдеров, но обрабатывает машины как на слоях Salt, так и на облаках. Новые машины автоматически загружаются, регистрируются и санкционируются. Удаленные машины удаляются из системы Salt, а также из облака провайдера.



Система SaltStack не поддерживает репозиторий пакетов для операционной системы FreeBSD, но это поддерживаемая в FreeBSD платформа. Веб-установщик распознает систему FreeBSD:

```
$ curl -L https://bootstrap.saltstack.com -o /tmp/saltboot
$ sudo sh /tmp/saltboot -P -M
```

По умолчанию веб-установщик устанавливает клиентское программное обеспечение, а также главный сервер. Если вы этого не хотите, укажите аргумент **-N** в команде **saltboot**.

🕮 Дополнительную информацию о контейнерах см. в главе 25.

Файлы конфигурации Salt размещены в каталоге **/etc/salt**, как на главном сервере, так и на клиентах (миньонах). Теоретически возможно запустить демон сервера от имени непривилегированного пользователя, но для этого требуется вручную задать права доступа для множества системных каталогов, с которыми будет взаимодействовать система Salt. Если вы хотите идти по этой дороге, вам, вероятно, лучше использовать контейнерную версию сервера или записать конфигурацию в предварительно сохраненном образе машины.

Система Salt имеет простую систему управления доступом, которую можно настроить, чтобы позволить непривилегированным пользователям инициировать операции Salt на миньонах. Тем не менее вы должны будете выполнить вручную настройку прав доступа, аналогичную той, которая требуется для работы от имени непривилегированного пользователя. Учитывая, что мастер имеет прямой доступ ко всем миньонам от имени суперпользователя `root`, мы считаем эту функцию довольно сомнительной с точки зрения безопасности. Если вы ее используете, внимательно следите за предоставляемыми правами доступа.

Система Salt поддерживает разделение между файлами конфигурации, в которых устанавливаются значения переменных (базовые элементы (pillars)) и файлами конфигурации, в которых определяются операции (состояния (states)). Различие проходит вплоть до верхнего уровня: вы должны установить разные местоположения для этих иерархий файлов конфигурации. Они обе по умолчанию находятся в каталоге **/srv**, чему соответствуют следующие записи в файле **/etc/salt/master**:

```
file_roots:
  base:
    - /srv/salt

pillar_roots:
   base:
     - /srv/pillar
```

Здесь `base` — это требуемая общая среда, поверх которой могут быть добавлены дополнительные среды (например, `development`). Определения переменных находятся в корне **/srv/pillar**, а все остальное находится в каталоге **/srv/salt**.

Обратите внимание на то, что сами пути являются элементами списка, так как они имеют префикс в виде тире. Вы можете включить несколько каталогов, что позволяет демону **salt-master** передавать объединенное представление перечисленных каталогов миньонам. Это полезная функция при создании большой базы конфигурации, так как она позволяет вам добавлять структуру, которую система Salt не смогла бы понять самостоятельно.

Как правило, база конфигурации управляется как единый репозиторий Git, который включает в себя подкаталоги **salt** и **pillar**. Это не подходит для стандартной схемы, принятой по умолчанию, поскольку означает, что каталог **/srv** будет корнем репозитория; рассмотрите возможность перемещения всего уровня в каталог **/srv/salt/salt** и **/srv/salt/pillar**.

Документация системы Salt не очень хорошо объясняет, почему базовые элементы и состояния должны быть полностью разделены, но на самом деле это различие является центральным для ее архитектуры. Демон **salt-master** не обращает ни малейшего внимания на файлы состояний; он просто делает их доступными для миньонов, которые несут ответственность за синтаксический разбор и выполнение этих файлов.

Базовые элементы — совсем другое дело. Они обрабатываются мастером и распространяются среди миньонов в виде единой унифицированной иерархии в формате JSON. Каждый миньон видит разные представления базовых элементов, но ни один из них не может видеть механизм реализации этих представлений.

Частично это мера безопасности: система Salt дает сильную гарантию, что миньоны не могут получить доступ к базовым элементам друг друга. Это также обеспечивает разделение источников данных, так как динамическое содержание базовых элементов всегда происходит от мастера. Это обеспечивает хорошую взаимодополняемость с зернами (термин системы Salt для фактов), которые происходят от миньонов.

🕮 Дополнительную информацию о сетевых брандмауэрах см. в разделе 27.7.

Коммуникационная шина системы Salt использует TCP-порты 4505 и 4506 на сервере. Убедитесь, что эти порты доступны через любые брандмауэры или фильтры пакетов, которые находятся между сервером и потенциальными клиентами. Сами клиенты не принимают сетевые подключения, поэтому этот шаг необходимо выполнить только один раз для сервера.

Впервые исследуя систему Salt, вы можете обнаружить, что она будет более информативной, если ее запустить с помощью команды **salt-master -l debug** в окне терминала, а не в качестве системной службы. Это заставляет демон **system-master** работать на переднем плане и выводить на экран информацию о действиях в коммуникационной шине системы Salt по мере их появления.

Настройка миньонов

Как и на главной стороне, у вас есть выбор собственных пакетов из репозитория SaltStack или универсального сценария начальной загрузки. Репозиторий вряд ли стоит загружать данными о миньонах, поэтому мы рекомендуем следующее.

```
$ curl -o /tmp/saltboot -sL https://bootstrap.saltstack.com
$ sudo sh /tmp/saltboot -P
```

Сценарий начальной загрузки работает в любой поддерживаемой системе. В системах без утилиты **curl** утилиты **wget** и **fetch** также работают нормально. Конкретные

сценарии инсталляции и исходный код представлены в репозитории `saltstack/salt-bootstrap` на сайте GitHub.[16]

🕮 Дополнительную информацию о службе DNS см. в главе 16.

По умолчанию демон **salt-minion** пытается зарегистрироваться на главной машине под именем `salt`. (Эта система "волшебного имени" была впервые популяризирована в системе Puppet.) Вы можете использовать систему DNS для правильного преобразования имени или явно задать главную машину в файле **/etc/salt/minion** (**/usr/local/etc/salt/minion** в системе FreeBSD):

```
master: salt.example.com
```

Перезагрузите демон **salt-minion** после изменения этого файла (обычно для этого используется команда **service salt_minion restart**; учтите, что здесь применяется подчеркивание, а не дефис).

Демон **salt-master** принимает регистрацию клиентов с любой случайной машины, которая имеет к нему доступ, но вы должны сами одобрить каждый клиент с помощью команды **salt-key** на главном сервере конфигурации до того, как он станет активным.

```
$ sudo salt-key -l unaccepted
Unaccepted Keys:
new-client.example.com

# Если все хорошо, одобрите все ожидающие ключи

$ sudo salt-key -yA
The following keys are going to be accepted:
Unaccepted Keys:
new-client.example.com
Key for minion new-client.example.com accepted.
```

Теперь вы можете проверить подключение со стороны сервера с помощью модуля `test`.

```
$ sudo salt new-client.example.com test.ping
new-client.example.com:
    True
```

В этом примере строка **new-client.example.com** напоминает имя хоста, но на самом деле это не так. Это обычный идентификатор Salt для машины, т.е. строка, которая по умолчанию соответствует имени хоста, но она может быть изменена по вашему желанию в файле клиента **/etc/salt/minion**.

```
master: salt.example.com
id: new-client.example.com
```

Идентификаторы и IP-адреса не имеют ничего общего друг с другом. Например, даже если строка **52.24.149.191** фактически соответствовала IP-адресу клиента, ее нельзя непосредственно указывать в качестве цели в параметрах команд системы Salt:[17]

```
$ sudo salt 52.24.149.191 test.ping
No minions matched the target. No command was sent, no jid was assigned.
ERROR: No return received
```

[16] Для производственных систем, запускаемых автоматически, необходимо минимизировать воздействие внешних событий, загрузив локально кешированную версию сценария загрузки. Инсталлируйте определенную версию клиента Salt из локального кеша или предварительно загрузите его в образ машины. Запустите загрузочный сценарий с параметром **-h**, чтобы просмотреть все параметры, которые он поддерживает.

[17] Разумеется, вы можете выполнить сопоставление на основе IP-адресов. Просто это должно быть сделано явно. Подробности см. ниже.

Привязка значения переменной к миньону

Как мы видели в разделе, посвященном настройке сервера, у системы Salt есть отдельные иерархии файловой системы для привязок состояний и привязок значений к переменным (базовые элементы). В корне каждой из этих иерархий каталогов находится файл **top.sls**, который связывает группы миньонов с файлами внутри дерева. У двух файлов **top.sls** используется один и тот же формат. (Расширение **.sls** — это стандартное расширение Salt для файлов YAML.)

В качестве примера приведем схему простой базы конфигурации Salt, в которой существуют корни **salt** и **pillar**.

```
$ tree /srv/salt
/srv/salt
├── salt
│   ├── top.sls
│   ├── hostname.sls
│   ├── bootstrap.sls
│   ├── sshd.sls
│   └── baseline.sls
└── pillar
    ├── top.sls
    ├── baseline.sls
    ├── webserver.sls
    └── freebsd.sls

2 directories, 9 files
```

Чтобы привязать переменные, определенные в файлах **pillar/baseline.sls** и **pillar/freebsd.sls**, к нашему клиенту, мы могли бы включить следующие строки в файл **pillar/top.sls**.

```
base:
  new-client.example.com:
    - baseline
    - freebsd
```

Как и в файле **master**, base — это обязательная общая среда, которая может быть переопределена в более сложных настройках. Подробнее об этом см. ниже в разделе "Среды".

Некоторые из тех же значений переменных можно определить в файлах **baseline.sls** и **freebsd.sls**. Для скалярных значений и массивов действующим источником является тот, который последним указан в файле **top.sls**.

Например, если миньон связывается с одним файлом переменных, который выглядит так:

```
admin-users:
  manny:
    uid: 724
  moe:
    uid: 740
```

и другим, который выглядит так:

```
admin-users:
  jack:
    uid: 1004
```

то система Salt объединяет две версии.

Базовые элементы, представленные миньонам, выглядят так:

```
admin-users:
   manny:
     uid: 724
   moe:
     uid: 740
   jack:
     uid: 1004
```

Сопоставление миньонов

В приведенном выше сценарии мы хотим применить файл **baseline.sls** ко всем без исключения клиентам, а файл **freebsd.sls** — ко всем клиентам, на которых работает операционная система FreeBSD. Вот как мы можем это сделать с помощью шаблонов выбора в файле **pillar/top.sls**.

```
base:
  '*.example.com':
    - baseline
  'G@os:FreeBSD':
    - freebsd
```

Звездочка соответствует всем идентификаторам клиентов в домене `example.com`. Можно было просто использовать здесь обозначение `'*'`, но мы хотели подчеркнуть, что это шаблон подстановки. Префикс `G@` запрашивает совпадение значений зерен. Проверяемое зерно называется `os`, а искомое значение — FreeBSD. Здесь также допускается использование шаблонов подстановки.

Было бы проще написать соответствующее выражение для FreeBSD так:

```
'os:FreeBSD':
  - match: grain
  - freebsd
```

Выбор зависит от вас, но символ `@` явно расширяет сложные выражения, содержащие круглые скобки и логические операции. В табл. 23.5 перечислены наиболее распространенные виды сопоставления, хотя некоторые из них были опущены.

Таблица 23.5. Типы сопоставления миньонов

Код	Цель	Тип	Сопоставление:	Пример
-[a]	ID	glob	glob	*.cloud.example.com
E	ID	regex	pcre	E@(nw|wc)-link-\d+
L	ID	list	list	L@hosta,hostb,hostc[б]
G	grain	glob	grain	G@domain:*.example.com
E	grain	regex	grain_pcre	E@virtual:(xen|VMWare)
I	pillar	glob	pillar	I@scaling_type:autoscale
J	pillar	regex	pillar_pcre	J@server-class:(web|database)
S	IP-адрес	CIDR-блок	ipcidr	S@52.24.9/20
-[в]	compound	compound	compound	not G@os_family:RedHat

[a] Это значение по умолчанию. Код соответствия не нужен (или определен).

[б] Обратите внимание на отсутствие пробелов; отдельные выражения не могут их содержать.

[в] Коды используются для обозначения отдельных терминов.

Если табл. 23.5 показалась вам сложной, не переживайте; это просто варианты. Реальные селекторы гораздо больше похожи на наши простые примеры.

Если вам интересно, с какими значениями зерен или базовых элементов можно выполнять сопоставление, их легко узнать. Просто используйте команду

```
$ sudo salt миньон grains.items
```

или

```
$ sudo salt миньон pillar.items
```

для получения полного списка.

Вы можете определить именованные группы в файле **/etc/salt/master**. Они называются *группами узлов* (*nodegroups*) и полезны для перемещения сложных селекторов групп из файлов **top.sls**. Тем не менее они не являются истинным механизмом группировки, а также способом именования шаблонов для повторного использования. В результате их поведение выглядит немного странным. Они могут быть определены только в терминах селекторов типа `compound` (а не, например, простым списком клиентов, если вы используете спецификацию `L@`), и вы должны использовать явное сопоставление с типом группы узлов с помощью директивы **match**. Глобальной сокращенной формы записи не существует.

Состояния системы Salt

Операции в системе Salt называются *состояниями* (states). Как и в системе Ansible, они определены в формате YAML, и на самом деле выглядят отдаленно похожими на задачи Ansible. Однако их внутреннее устройство совершенно разное. Вы можете включить серию определений состояний в файл **.sls**.

Состояния привязаны к конкретным миньонам в файле **top.sls**, который находится в корневой части ветви **salt** базы конфигурации. Этот файл выглядит и работает точно так же, как файл **top.sls** для привязок переменных; см. примеры, приведенные выше.

Взгляните на следующую Salt-версию примера, который мы использовали для иллюстрации системы Ansible: мы инсталлируем программу **sudo** и создаем соответствующую группу **sudo**, в которую мы добавляем администраторов, у которых должны быть привилегии **sudo**. Затем мы создаем группу учетных записей администратора, каждая из которых имеет собственную группу UNIX с тем же именем. Наконец, мы копируем файл **sudoers** из базы конфигурации.

Как это часто бывает, мы можем использовать точно такой же файл переменных для систем Salt, который мы использовали для Ansible:

```
sudoers_path: /etc/sudoers
admins:
  - { username: manny, fullname: Manny Calavera }
  - { username: moe, fullname: Moe Money }
```

Чтобы эти определения были доступны всем миньонам, мы помещаем их в базу конфигурации в файле **pillar/example.sls** и добавляем привязку в файл **top.sls**.

```
base:
  '*':
    - example
```

Вот Salt-версия этих операций.

```
install-sudo-package:
  pkg.installed:
```

```
    - name: sudo
    - refresh: true

create-sudo-group:
  group.present:
    - name: sudo

{% for admin in pillar.admins %}

create-group-{{ admin.username }}:
  group.present:
    - name: {{ admin.username }}

create-user-{{ admin.username }}:
  user.present:
    - name: {{ admin.username }}
    - gid: {{ admin.username }}
    - groups: [ wheel, sudo ]
    - fullname: {{ admin.fullname }}

{% endfor %}

install-sudoers-file:
  file.managed:
    - name: {{ pillar.sudoers_path }}
    - source: salt://files/sudoers
    - user: root
    - group: wheel
    - mode: '0600'
```

В этой версии показаны операции в их максимально канонической форме для более простого сравнения с эквивалентным списком задач Ansible, который был рассмотрен ранее. Мы можем внести несколько дополнительных изменений для улучшения примера, но сначала целесообразно рассмотреть эту более длинную версию.

Система Salt и препроцессор Jinja

Первое, что нужно отметить, это то, что данный файл содержит цикл Jinja, ограниченный символами `{%` и `%}`. Эти разделители похожи на `{{` и `}}`, за исключением того, что `{%` и `%}` не возвращают значения. Содержимое цикла интерполируется в файл YAML столько раз, сколько раз выполняется цикл.

🕮 Общую информацию о языке Python см. в разделе 7.5.

В препроцессоре Jinja используется синтаксис, подобный синтаксису языка Python. Однако в формате YAML уже предусмотрены отступы в файле **`.sls`**, поэтому в препроцессоре Jinja вынуждено определены маркеры завершения блоков, такие как `endfor`. В программах на языке Python блоки обычно определяются с помощью отступов.

В базовой схеме YAML системы Salt определена только элементарная конструкция итераций (см. комментарии по поводу директивы `name` в разделе "Параметры и имена"). Условные и надежные конструкции итерактивного выполнения должны предоставляться препроцессором Jinja или любым языком шаблонов, через который прогоняется файл **`.sls`**. (На самом деле система Salt тоже не анализирует формат YAML. Она просто пропускает конфигурационные файлы через выделенный конвейер и получает конечный вывод в формате JSON, который должен быть полностью литеральным.)

С одной стороны, этот подход ясен. Там нет концептуальной двусмысленности в отношении того, что происходит, и легко просмотреть расширенный файл `.sls`, чтобы убедиться, что он соответствует вашим целям. С другой стороны, это означает, что вы будете использовать препроцессор Jinja для обеспечения любой логики, необходимой для вашей конфигурации. Смешение кода шаблонов и формата YAML может стать затруднительным. Это немного похоже на написание логики веб-приложения с использованием только HTML-шаблонов.

Несколько эмпирических правил могут помочь сохранить конфигурацию Salt. Во-первых, у системы Salt есть пригодные для использования и четко определенные механизмы для реализации замещения значений переменных. Используйте их, чтобы сохранить как можно больше конфигурации в области данных, а не в коде.

Во многих примерах в документации Salt используются условные конструкции препроцессора Jinja, в то время как они не являются лучшим решением.[18] Следующий файл `.sls` позволяет установить веб-сервер Apache, который имеет разные имена пакетов в разных дистрибутивах:

```
# apache-pkg.sls
apache:
  pkg.installed:
    {% if grains['os'] == 'RedHat' %}
    - name: httpd
    {% elif grains['os'] == 'Ubuntu' %}
    - name: apache2
    {% endif %}
```

Этот вариант можно было бы более элегантно описать с помощью базового элемента (pillars):

```
# apache-pkg.sls
{{ pillar['apache-pkg'] }}:
   pkg.installed

# pillar/top.sls
base:
  '*':
    - defaults
  'G@os:Ubuntu':
    - ubuntu

# pillar/defaults.sls
apache-pkg: httpd

# pillar/ubuntu.sls
apache-pkg: apache2
```

Хотя замену одного файла четырьмя нельзя считать упрощением, теперь это расширяемая система без кода. В средах с несколькими операционными системами такие варианты встречаются часто, и все они могут быть рассмотрены в одном месте.

Если значение должно быть вычислено динамически, подумайте, нельзя ли поместить код в верхнюю часть файла `.sls` и просто запомнить его для последующего использования в переменной. Например, другой способ записи установки пакета Apache имел бы такой вид:

[18] Справедливости ради укажем, что эти примеры, как правило, предназначены для иллюстрации специфичных моментов и не стремятся к полной корректности.

```
{% set pkg_name = 'httpd' %}
{% if grains['os'] == 'Ubuntu' %}
  {% set pkg_name = 'apache2' %}
{% endif %}

{{ pkg_name }}:
   pkg.installed:
```

Это по крайней мере позволяет отделить логику Jinja от фактической конфигурации. Если вы вынуждены смешивать логику препроцессора Jinja с форматом YAML, подумайте, можете ли вы разбить некоторые сегменты YAML на отдельные файлы. Затем вы можете интерполировать эти сегменты по мере необходимости. И снова идея состоит в том, чтобы просто отделить код и YAML, а не чередовать их.

Для нетривиальных вычислений можно полностью отказаться от формата YAML и заменить его чистым языком Python или применить один из предметной-ориентированных языков, основанных на языке Python, который по умолчанию использует система Salt. Для получения дополнительной информации см. документацию Salt о рендеринге (renderers).

Идентификаторы состояний и зависимости

Вернемся к нашему примеру **sudo**. Напомним еще раз его первые два состояния.

```
install-sudo-package:
  pkg.installed:
    - name: sudo
    - refresh: true

create-sudo-group:
  group.present:
    - name: sudo
```

Легко видеть, что отдельные состояния являются элементами не списка (как в системе Ansible), а хеша. Хеш-ключ для каждого состояния — это произвольная строка, называемая *идентификатором*. Как обычно, идентификаторы должны быть уникальными, иначе возникнут коллизии.

Но подождите! Потенциальная область коллизий — это не только этот конкретный файл, но и вся клиентская конфигурация. Идентификаторы состояний должны быть глобально уникальными, потому что система Salt в конечном итоге собирает все данные вместе в один большой хеш.

Это довольно необычный хеш, потому что он сохраняет порядок ключей. В стандартном хеше при перечислении ключи появляются в случайном порядке. Система Salt работает именно так. В результате все зависимости между состояниями должны были быть объявлены явно. В настоящее время хеш сохраняет порядок представления по умолчанию, хотя это все равно можно переопределить, если объявлены явные зависимости (или если это поведение отключено в файле **master**).

Тем не менее есть еще одна хитрость. При отсутствии других ограничений порядок выполнения соответствует исходным файлам **.sls**. Однако система Salt по-прежнему полагает, что состояния не логически зависят друг от друга, если вы не утверждаете обратное. Если состояние не выполняется, система Salt отмечает ошибку, но затем продолжает и запускает следующее состояние.

Для того чтобы зависимое состояние не запускалось, если его предшественники завершились неудачей, можете объявить это явно. Например, так.

```
create-sudo-group:
  group.present:
    - name: sudo
    - require:
      - install-sudo-package
```

В этой конфигурации система Salt не будет пытаться создать группу `sudo`, если пакет `sudo` не будет успешно установлен.

Реквизиты также вступают в действие при упорядочении состояний из нескольких файлов. В отличие от системы Ansible, система Salt не интерполирует содержимое файла `include` в точке, где был обнаружен вызов. Она просто добавляет файл в список для чтения. Если несколько файлов пытаются включить один и тот же источник, в последней сборке все еще будет только одна копия источника, а порядок состояний может быть не таким, как вы ожидали. Выполнение по порядку гарантируется только в файле; если какие-либо состояния зависят от внешне определенных операций, они должны объявлять явные реквизиты.

Механизм реквизитов также используется для достижения эффекта, аналогичного уведомлениям в системе Ansible. Несколько альтернатив спецификации `require` являются синтаксически взаимозаменяемыми, но подразумевают тонкие оттенки поведения. Одна из них, `watch`, особенно полезна, потому что позволяет выполнять определенные действия, когда другое состояние вносит изменения в систему.

Например, в следующей конфигурации устанавливается часовой пояс системы и аргументы, которые должны быть переданы демону **`ntpd`** при запуске. Эта конфигурация всегда гарантирует, что демон **`ntpd`** будет запущен и настроен для запуска во время загрузки. Кроме того, она перезапускает **`ntpd`**, если обновляется либо системный часовой пояс, либо флаги **`ntpd`**.

```
set-timezone:
  timezone.system:
    - name: America/Los_Angeles

set-ntpd-opts:
  augeas.change:[19]
    - context: /files/etc/rc.conf
    - lens: shellvars.lns
    - changes:
      - set ntpd_flags '"-g"'[20]

ntpd:
  service.running:
    - enable: true
    - watch:
      - set-ntpd-opts
      - set-timezone
```

Функции состояния и выполнения

В файле **`.sls`** имена, которые указываются непосредственно под идентификаторами состояния, являются операциями, которые должны выполняться этими состояниями. В нашем примере есть две операции `pkg.installed` и `group.present`.

[19]Augeas — это инструмент, который понимает множество разных форматов файлов и облегчает автоматические изменения.

[20]Как видно из этой строки, использование кавычек в формате YAML может быть тонким делом.

В именах этих операций включены имя модуля и имя функции. Взятые вместе они примерно аналогичны имени модуля в системе Ansible, дополненным значением состояния. Например, в системе Ansible используется пакетный модуль со спецификацией `state = present` для установки пакетов, тогда как в системе Salt используется специальная функция `pkg.installed` в модуле `pkg`.

В системе Salt сделано многое, чтобы отделить функции для целевых систем ("функции выполнения") от тех, которые идемпотентно обеспечивают определенную конфигурацию ("функции состояния"). Функции состояния обычно вызывают связанные с ними функции выполнения, когда им необходимо внести изменения.

Общая идея заключается в том, что в файлах **`.sls`** указываются только функции состояния, а в командных строках должны указываться только функции выполнения. Система Salt слишком примитивно применяет эти правила, что иногда приводит к путанице.

Функции состояния и выполнения находятся в разных модулях Python, но связанные модули обычно имеют одно и то же имя. Например, есть как модуль состояния часового пояса, так и модуль выполнения часового пояса. Однако не может быть перекрытия имен функций между этими двумя модулями, поскольку это создавало бы двусмысленность. Конечным результатом является то, что для установки часового пояса из файла **`.sls`** вы должны использовать модуль `timezone.system`:

```
set-timezone:
  timezone.system:
    - name: America/Los_Angeles
```

Однако, чтобы установить часовую зону миньона из командной строки, необходимо использовать модуль `timezone.set_zone`:

```
$ sudo salt minion timezone.set_zone America/Los_Angeles
```

Если вы обратитесь к документации, то найдете описание двух половин модуля `timezone` в разных разделах руководства. Также не всегда ясно, как определить тип функции по ее поведению. Например, функция `git.config_set`, которая устанавливает параметры репозитория Git, является функцией состояния, а `state.apply`, которая идемпотентно задает конфигурации, является функцией выполнения.

В конечном счете вы просто должны знать функции и контексты, к которым они принадлежат. Если вам нужно вызвать функцию из "неправильного" контекста, что иногда необходимо, вы можете использовать функции адаптера `module.run` (запускает функцию выполнения из контекста состояния) и `state.single` (запускает функцию состояния из контекста выполнения). Например, адаптированные вызовы функций для задания часового пояса имеют следующий вид:

```
set-timezone:
  module.run:
    - name: timezone.set_zone
    - timezone: America/Los_Angeles
```

и

```
# salt minion state.single timezone.system name=America/Los_Angeles
```

Параметры и имена

Еще раз приведем первые два состояния нашего примера.

```
install-sudo-package:
  pkg.installed:
```

```
    - name: sudo
    - refresh: true

create-sudo-group:
  group.present:
    - name: sudo
```

После отступа под названием каждой операции (т.е. конструкцией *модуль.функция*) указывается ее список параметров. В системе Ansible параметры для операции образуют один большой хеш. Система Salt хочет, чтобы они были списком, причем каждая запись была предварена тире. Более конкретно, системе Salt нужен список хешей, хотя в каждом хеше обычно есть только один ключ.

Большинство списков параметров содержат параметр с именем `name`, который является стандартной меткой "того, что эта операция настраивает". В качестве альтернативы вы можете указать список целей в параметре с именем `names`. Например:

```
create-groups:
  group.present:
    - names:
      - sudo
      - rvm
```

Если вы задаете параметр `names`, то система Salt повторно запускает операцию несколько раз, подставляя один элемент из списка `names` вместо значения параметра `name` на каждом проходе. Это механический процесс, и сама операция не знает об итерации. Эта операция выполняется на этапе выполнения (а не синтаксического анализа) и напоминает конструкцию `with_items` в системе Ansible. Но поскольку операция расширения Jinja уже завершена, у системы нет возможности формировать значения других параметров на основе параметра `name`. Если вам нужно настроить несколько параметров, игнорируйте параметр `names` и просто повторяйте цикл Jinja.

В некоторых операциях могут использоваться сразу несколько аргументов. Например, операция `pkg.installed` может сразу передать несколько имен пакетов в базовый диспетчер пакетов операционной системы, что может быть полезно для повышения эффективности или распознавания зависимости. Поскольку система Salt скрывает итерацию на основе параметра `names`, в подобных операциях нужно использовать отдельное имя параметра для выполнения пакетных операций. Например, состояния

```
install-packages:
   pkg.installed:
      - names: [ sudo, curl ]
```

и

```
install-packages:
   pkg.installed:
      - pkgs: [ sudo, curl ]
```

инсталлируют программы **`sudo`** и **`curl`**. Первая версия делает это в двух отдельных операциях, а вторая — в одной.

Мы подчеркиваем этот, казалось бы, незначительный момент, потому что параметр `names` легко использовать ошибочно. Поскольку процесс механический, итерация на основе параметра `names` работает даже для тех операций, в которых не используется параметр `name`. При просмотре журнала системы Salt вы увидите, что несколько запусков были успешно выполнены, но целевая система почему-то по-прежнему не настроена должным образом, и нужно понять, что происходит.

Если вы не указываете явно параметр `name` для состояния, система Salt копирует в его поле идентификатор состояния. Вы можете использовать это поведение для упрощения определения состояний. Например,

```
create-sudo-group:
  group.present:
    - name: sudo
```

становится

```
sudo:
  group.present
```

и даже

```
sudo: group.present
```

Формат YAML не допускает использование хеш-ключей без значений, поэтому теперь, когда функция `group.present` больше не имеет перечисленных параметров, хеш-ключ должен стать простой строкой, а не хеш-ключом со списком параметров в качестве значения. Это хорошо! Система Salt проверяет это явно.

Лаконичный стиль более четкий, чем длинный. Отдельное поле идентификатора может теоретически служить комментарием или объяснением, но большинство идентификаторов, встречающихся в реальности, просто описывают поведение, которое уже очевидно. Если вы хотите добавить комментарии, сделайте это. У лаконичной формы есть потенциальная проблема: поскольку идентификаторы состояния должны быть глобально уникальными, короткие идентификаторы общих системных объектов становятся более уязвимыми для коллизий. Система Salt обнаруживает и сообщает о конфликтах, поэтому это скорее досадная, чем серьезная проблема. Но если вы пишете формулу Salt с намерением повторно использовать ее в нескольких конфигурационных базах или собираетесь поделиться ею с сообществом Salt, придерживайтесь идентификаторов, которые с меньшей вероятностью испытают коллизию.

Система Salt позволяет включать несколько операций в одно состояние. Поскольку две приведенные выше операции содержат общее поле `name`, мы можем объединить их в одно состояние без необходимости указывать в каких-либо состояниях поле `name` явно. Тем не менее есть еще одна ловушка YAML:

```
sudo:
  pkg.installed:
    - refresh: true
  group.present: []
```

Значение ключа `sudo` теперь должно быть хешем; причем к этому хешу не должна добавляться строка `group.present`. Соответственно, теперь мы должны рассматривать `group.present` как хеш-ключ и предоставлять явный список параметров в качестве значения, хотя этот список пуст. Это утверждение остается справедливым, даже если мы отбросим параметр `refresh` из `pkg.installed`:

```
sudo:
  pkg.installed: []
  group.present: []
```

Так же, как мы свернули эти два состояния, мы можем свернуть наши два состояния, которые управляют учетными записями пользователей. Таким образом, более аккуратная версия списка состояний имеет следующий вид.

```
sudo:
  pkg.installed: []
  group.present: []
```

```
{% for admin in pillar.admins %}
{{ admin.username }}:
   group.present: []
   user.present:
     - gid: {{ admin.username }}
     - groups: [ wheel, sudo ]
     - fullname: {{ admin.fullname }}
{% endfor %}
{{ pillar.sudoers_path }}:
   file.managed:
     - source: salt://files/sudoers
     - user: root
     - group: wheel
     - mode: '0600'
```

Привязка состояний к миньонам

О привязках состояний в системе Salt не так много можно сказать. Они работают точно так же, как привязки базовых элементов. В корневой иерархии состояния есть файл **top.sls**, в котором задаются соответствия групп миньонов файлам состояния. Приведем схематический пример.

```
base:
  '*':
    - bootstrap
    - sitebase
  'G@os:Ubuntu':
    - ubuntu
  'G@webserver':
    - nginx
    - webapps
```

В этой конфигурации ко всем хостам применяются состояния из файлов **bootstrap.sls** и **sitebase.sls** из корня иерархии состояний. В системах Ubuntu также существует файл **ubuntu.sls**, поэтому веб-серверы (т.е. миньоны с элементом верхнего уровня `webserver` в своих базах данных о зернах) запускают состояния для настройки NGINX и локальных веб-приложений.

Порядок в файле **top.sls** соответствует общему порядку выполнения на каждом миньоне. Но, как обычно, явная информация о зависимостях в состояниях переопределяет порядок, принятый по умолчанию.

Состояния высокого уровня

Система Salt относится к привязкам в файле **top.sls** как к состояниям высокого уровня миньона.[21] Вы активируете состояние высокого уровня, приказывая миньону запустить функцию `state.apply` без аргументов:

```
$ sudo salt миньон state.apply
```

[21] Существует потенциальная терминологическая путаница в том, что в системе Salt также используется термин "состояние высокого уровня" (highstate) для обозначения "разобранного и собранного JSON-дерева состояний", которое затем переходит в "состояние низкого уровня" (lowstate) — JSON-дерева, которое представляет собой входные данные низкого уровня для механизма выполнения.

Функция `state.highstate` эквивалентна функции `state.apply` без аргументов. Обе формы широко используются.

В частности, при отладке определений новых состояний вам может потребоваться, чтобы миньон запускал только один файл состояния. Это легко достигается с помощью функции `state.apply`:

```
$ sudo salt миньон state.apply файл_состояния
```

Не указывайте суффикс **`.sls`** в имени файла состояния; система Salt добавит его самостоятельно. Также имейте в виду, что путь к файлу состояния не имеет ничего общего с вашим текущим каталогом. Он всегда интерпретируется относительно корня состояния, как определено в файле конфигурации миньона. В любом случае эта команда не переопределяет состояние высокого уровня миньона; она просто запускает указанный файл состояния.

В команде **`salt`** можно указать несколько флагов для ориентации на различные группы миньонов, но проще всего запомнить флаг **`-C`** для типа `compound` и использовать одно из сокращений из табл. 23.5.

Например, чтобы перевести все миньоны Red Hat в состояние высокого уровня, выполните следующую команду:

```
$ sudo salt -C G@os:RedHat state.highstate
```

По умолчанию в типе сопоставления используется подстановка идентификатора, поэтому команда

```
$ sudo salt '*' state.highstate
```

представляет собой команду для "проверки всей конфигурации для сайта".

В соответствии с моделью выполнения Salt, ориентированной на миньоны, все параллельные процессы выполнения начинаются одновременно, а миньоны не сообщают об этом до тех пор, пока они не завершили свой процесс. Команда **`salt`** выводит результаты работы каждого миньона сразу же после их получения. Невозможно отображать промежуточные результаты во время выполнения файла состояния.

Если у вас много миньонов или сложная база конфигурации, результаты по умолчанию команды **`salt`** могут быть довольно объемными, потому что она сообщает о каждой операции, выполняемой каждым миньоном. Добавьте параметр **`--state-output=mixed`**, чтобы уменьшить этот вывод до одной строки для успешных операций и не вызывать никаких изменений. Параметр **`--state-verbose=false`** полностью подавляет вывод для операций без изменений, но система Salt по-прежнему выводит заголовок и сводку для каждого миньона.

Формулы Salt

В системе Salt пакеты (bundles) называются *формулами*. Подобно роли в системе Ansible, это всего лишь каталог файлов, хотя формулы Salt имеют внешнюю оболочку, которая также содержит некоторые метаданные и информацию о версии. В реальном использовании вам просто нужен внутренний каталог формул.

Каталоги формул входят в один из корней каталогов **`salt`**, определенных в файле **`master`**. При желании можно создать корень только для формул. Формулы иногда включают в себя пример базовых элементов, но вы сами должны их инсталлировать.

Система Salt ничего не делает для поддержки формул, за исключением того, что если вы указываете каталог в файле **`top.sls`** или используете операцию `include`, система Salt ищет файл **`init.yml`** в этом каталоге и интерпретирует его. Это соглашение обеспечивает понятный путь по умолчанию в формуле. Во многие формулы также включены автономные состояния, на которые вы можете ссылаться, указав как каталог, так и имя файла.

Ничто в системе Salt не может быть включено в конфигурацию более одного раза, это относится и к формулам. Вы можете сделать несколько запросов на включение, но все они будут объединены. В результате формулы не могут быть созданы несколько раз по образу и подобию ролей в системе Ansible.

В любом случае это не имеет значения, поскольку в системе Salt не определен никакой другой способ передачи параметров в формулу, кроме как путем ввода значений переменных в базовом элементе. (Выражения Jinja могут устанавливать значения переменных, но эти параметры существуют только в контексте текущего файла.) Чтобы смоделировать эффект вызова формулы повторно, вы можете предоставить данные базового элемента в виде списка или хеша, чтобы формула могла выполнять итерацию самостоятельно. Однако формула должна быть явно написана с учетом этой структуры. Вы не можете навязать это постфактум.

Центральным репозиторием Salt для предложений, внесенных сообществом, в настоящее время является только GitHub. Ищите формулы Salt по имени `salt-formulas`. Каждая формула представляет собой отдельный проект.

Среды

🕮 Дополнительную информацию о средах см. в разделе 26.1.

Система Salt предпринимает усилия в направлении явной поддержки сред (например, разделение сред разработок, тестирования и производства). К сожалению, возможности ее сред несколько своеобразны, и они не сопоставляются прямо с наиболее распространенными практическими случаями реального мира. Можно получить среду и работать с небольшим количеством определений и шаблонами Jinja, а потом обнаружить, что на практике многие организации просто вытаскивают и запускают отдельные главные серверы для каждой среды. Это хорошо соответствует стандартам безопасности и согласованности, которые требуют разделения сред на сетевом уровне.

Как мы видели ранее, в файле **`/etc/salt/master`** указываются различные места, где может храниться информация о конфигурации. Он также связывает среду с каждым набором путей.

```
file_roots:
  base:
    - /srv/salt

pillar_roots:
  base:
    - /srv/pillar
```

Здесь **`/srv/salt`** и **`/srv/pillar`** — это корневые каталоги состояния и базового элемента с именем `base`. Для простоты мы не будем упоминать ниже о базовых элементах; управление окружающей средой работает одинаково для обеих ветвей базы конфигурации.

В сайтах с более чем одной средой обычно добавляется дополнительный слой в иерархию каталога конфигурации, чтобы отразить этот факт.

```
file_roots:
  base:
    - /srv/base/salt
  development:
    - /srv/development/salt
  production:
    - /srv/production/salt
```

(Очевидно, что у этого примера нет тестовой среды. Не пытайтесь делать это дома!)

В среде может быть указано несколько корневых каталогов. Если их несколько, сервер прозрачно объединяет их содержимое. Однако каждая среда выполняет отдельное слияние, и конечные результаты остаются разделенными.

Внутри файлов **top.sls** (привязки, которые связывают миньоны с конкретными состояниями и файлами столбцов) ключи верхнего уровня всегда являются именами окружения. До сих пор мы видели только примеры, в которых использовалась базовая среда, но, конечно, в этом месте может использоваться любая подходящая среда. Например, так.

```
base:
  '*':
    - global
development:
  '*-dev':
    - webserver
    - database
production:
  '*web*-prod':
    - webserver
  '*db*-prod':
    - database
```

Точный импорт внешнего вида среды в файл **top.sls** зависит от того, как вы настроили систему Salt. Во всех случаях среды должны быть определены в основном файле; файлы **top** не могут создавать новые среды. Кроме того, файлы состояний должны исходить из контекста среды, в котором они упоминаются.

По умолчанию система Salt не связывает миньоны с какой-либо конкретной средой, а миньонам могут быть присвоены состояния из любой или всех сред в файле **top.sls**. В приведенном выше фрагменте, например, все миньоны запускают состояние **global.sls** из базовой среды. В зависимости от их идентификаторов отдельные миньоны могут также получать состояния из среды производства или разработки.[22]

Документация Salt поощряет этот способ настройки окружения, но у нас есть некоторые оговорки. Одна из потенциальных проблем заключается в том, что миньоны выводят элементы конфигурации из нескольких сред. Вы не можете отследить источники произвольной конфигурации миньона до одной конкретной среды в определенный момент времени, потому что у каждого миньона есть несколько родителей.

Это различие важно, потому что одна базовая среда должна совместно использоваться всеми другими средами. Какой она должна быть? Версией базовой среды для разработки? Производственной версией? Полностью отдельной и многоуровневой базой конфигурации? Когда именно вы должны перенести базовую среду на новую версию?

Кроме того, существуют дополнительные сложности, скрывающиеся за сценой. Каждая среда представляет собой полноценную иерархию конфигурации Salt, поэтому теоретически она может иметь собственный файл **top.sls**. Каждый из этих файлов **top.sls** теоретически может относиться к нескольким средам.

Сталкиваясь с такой ситуацией, система Salt пытается объединить все файлы **top** в одну сложную фрагментарную конфигурацию.[23] Среды могут требовать выполнения

[22]Когда вы устанавливаете идентификатор миньона в соответствии с шаблоном разработки или производства, вы функционально связываете его с соответствующей средой. Однако сама система Salt не делает явной ассоциации, по крайней мере не в этой конфигурации.

[23]Слияние происходит на уровне YAML, хотя было бы лучше надеяться, что несколько файлов **top** не будут пытаться назначать состояния одному и тому же шаблону соответствия в той же среде. Если они это сделают, некоторые состояния будут проигнорированы.

других состояний, которыми они не владеют, не контролируют и не знают о них ничего. Это было бы ужасно, если бы это не было так глупо.

Непонятно, какие сценарии использования эта архитектура пытается сделать возможными. Хотя слияние файлов **top** — это поведение, принятое по умолчанию, документация постоянно предостерегает вас от этого. Вместо этого вам рекомендуется для управления всеми средами назначить один файл **top.sls**, скорее всего, в спецификации `base`.

Однако, если вы это сделаете, вскоре станет очевидно, что между этим "внешним" файлом **top** и остальными средами существует некоторое организационное противоречие. Файл **top** является неотъемлемой частью конфигурации среды, поэтому состояния и файлы **top** обычно разрабатываются совместно; изменение одного часто требует изменений в другом. Создавая отдельный файл **top**, вы должны по существу разделить каждую среду на две части, которые необходимо синхронизировать вручную. Кроме того, главный файл **top** должен быть общим, синхронизированным и совместимым со всеми другими средами. Например, если вы организовываете тестовую среду для производства, вы должны убедиться, что главный файл **top.sls** настроен так, чтобы отражать правильные настройки для конкретной версии нового выпуска.

В качестве альтернативы можно подключить миньоны к заданной среде, установив значение переменной `environment` в файле **/etc/salt/minion** на миньоне или включив флаг `saltenv=`*среда* в командные строки системы Salt. В этом режиме миньон видит только файл **top.sls** своей назначенной среды. В этом файле **top** его представление также ограничено записями, которые отображаются в этой среде.

Например, машина, прикрепленная к среде разработки, может видеть файл **top.sls** из предыдущего примера в следующей сокращенной форме (если предположить, что файл **top.sls** был найден в корне дерева разработки).

```
development:
  '*-dev':
    - webserver
    - database
```

Этот режим работы немного ближе к традиционной концепции среды, чем принятый по умолчанию. Здесь не может возникнуть непреднамеренная перекрестная коммуникация между средами, что ограничивает возможность непреднамеренного поведения. Также преимуществом является то, что по мере продвижения конкретной версии базы конфигурации через цепочку окружения различные части файла **top.sls** автоматически применяются к клиентам.

Основным недостатком является то, что вы теряете способность учитывать части конфигурации, общие для более чем одной среды. Нет никакого встроенного способа "видеть" вне контекста текущей среды, поэтому элементы базовой конфигурации должны быть реплицированы в каждую среду.

Файл **top.sls** из предыдущего примера, переписанный для работы в контексте этого подхода, будет выглядеть примерно так.

```
development:
  '*':
    - global
  '*-dev':
    - webserver
    - database
production:
  '*':
    - global
```

```
  '*web*-prod':
    - webserver
  '*db*-prod':
    - database
```

Базовая среда сама по себе является рудиментарной, поэтому мы исключили ее из файла **top.sls** и скопировали прежнее содержимое этого ключа непосредственно в среды разработки и производства.

Имейте в виду, что мы сейчас работаем в мире, где каждое дерево среды имеет свой собственный файл **top.sls**. В этом примере мы предполагаем, что файл **top.sls** в обеих средах является одинаковым, поэтому одно и то же содержимое появится в обеих копиях **top.sls**.

Конечно, ручное воспроизведение элементов общей конфигурации внутри каждой среды подвержено ошибкам. Лучшим вариантом является определение общей конфигурации как макроса Jinja, чтобы он мог повторяться автоматически.

```
{% macro baseline() %}
  '*':
    - global
{% endmacro %}

development:
  {{ baseline() }}
  '*-dev':
    - webserver
    - database
production:
  {{ baseline() }}
  '*web*-prod':
    - webserver
  '*db*-prod':
    - database
```

В этом сценарии мы предполагаем, что все миньоны привязаны к определенным средам, поэтому теперь мы можем потенциально удалить индикаторы среды из идентификаторов миньонов. Тем не менее целесообразно сохранить их для безопасности.

Проблема в том, что миньоны контролируют свои собственные настройки среды. Например, если миньон в среде разработки был скомпрометирован, он мог бы объявить себя производственным сервером и потенциально получить доступ к ключам и конфигурациям, используемым в производственной среде.[24] (Это, пожалуй, одна из причин, по которой документация по системе Salt проявляет определенную осторожность в рекомендациях по закреплению окружающей среды.)

Настройка конфигурации конкретной среды с учетом как параметров среды, так и идентификаторов миньонов защищает от этой разновидности атак. Если миньон изменяет свой идентификатор, главный сервер больше не распознает его как одобренный клиент и игнорирует до тех пор, пока администратор не утвердит это изменение с помощью команды **salt-key**.

Если вы предпочитаете не использовать идентификаторы таким образом, альтернативой является использование данных базовых элементов для перекрестной проверки.

[24]Проблема заключается не в состоянии конфигурации, так как миньоны имеют свободный доступ ко всем файлам состояния. Она связана с данными базовых элементов, которые собираются на главной стороне и обычно должны быть защищены.

Независимо от того, что вы делаете, вы не можете просто отказаться от суффикса и превратить `'*-dev'` в `'*'`, потому что в общей части конфигурации уже используется суффикс `'*'` в качестве ключа. Повторяющиеся шаблоны в среде — это нарушение правил YAML.

При отладке среды вы обнаружите, что несколько функций выполнения оказываются особенно полезными. Функция `config.get` выводит значение, которое использует определенный миньон (или набор миньонов) для параметра конфигурации.

```
$ sudo salt new-client-dev config.get environment
new-client-dev:
  development
```

Здесь мы видим, что миньон с идентификатором `new-client-dev` был привязан к среде разработки, так же как и его идентификатор. Чтобы узнать, как выглядит конфигурация **`top.sls`** с точки зрения миньона, используйте функцию `state.show_top`.

```
$ sudo salt new-client-dev state.show_top
new-client-dev:
  ----------
  development:
    - global
    - webserver
    - database
```

На выходе отображаются только те состояния, которые активны и выбраны для целевого миньона. Другими словами, они являются состояниями, которые будут выполняться, если вы применяете функцию `state.highstate` к этому миньону.

Обратите внимание на то, что все отображаемые состояния поступают из среды разработки. Поскольку миньон закреплен, это будет происходить всегда.

Документация

Документация системы Salt (`docs.saltstack.com`), вероятно, заслуживает восхищения, но только после периода разочарования. Основная проблема заключается в том, что темы расположены на нескольких вложенных слоях, но заголовки в двух верхних слоях не обязательно указывают на то, что вы найдете на третьем слое. Например, в разделе "Архитектура" нет информации об архитектуре Salt (на самом деле в нем речь идет о многосерверных развертываниях).

Некоторые из наиболее полезных справочных материалов находятся в разделах, которые организованы как сценарии или учебные пособия. Последовательное чтение иногда может вызывать у системных администраторов полное разочарование: темы поднимаются циклически и закрываются без полного описания. Краткий момент ясности лишь подчеркивает тяжесть вашего состояния.

Подскажем некоторые ориентиры.

- Верхний раздел *Using Salt* представляет собой обзор концепций, а большая часть раздела *Configuration Management* помечена как учебное пособие. Из-за их форматов эти разделы выглядят как дополнительные материалы. Но это неправда; они в значительной степени являются основной документацией для материала, который они охватывают. Не пропустите их.
- Наилучшая справочная информация находится под ссылкой *System Reference* в разделе *Configuration Management*. Многие вещи здесь не важны при первом чтении, но разделы *Highstate data structure definitions, Requisites and other global state arguments* и *The top file* особенно полезно прочитать. (Раздел *Top file* также является авторитетной документацией по средам.)

- Документы, которые вы будете использовать наиболее часто, описывающие функции состояния и выполнения, скрыты под ссылкой *Salt Module Reference* и замаскированы среди 19 других типов модулей, которые в основном интересуют разработчиков модулей. Отметьте для себя разделы *Full list of builtin state modules* и *Full list of builtin execution modules*.

23.7. Сравнение систем Ansible и Salt

Нам нравятся как Ansible, так и Salt. Однако каждая из этих систем имеет некоторые особенности, и мы рекомендуем их для разных сред. В приведенных ниже разделах мы прокомментируем несколько факторов, которые следует учитывать при выборе между ними.

Гибкость развертывания и масштабируемость

Система Salt охватывает более широкий диапазон условий развертывания, чем Ansible. Она достаточно простая, так что вы можете использовать ее для управления одним сервером, но система Salt масштабируется без усилий и практически без ограничений. Если вы хотите изучить одну систему, которая охватывает максимально широкий диапазон вариантов использования, Salt — хороший выбор.

Частично это связано с тем, что архитектура системы Salt предъявляет относительно небольшие требования к главному серверу. Миньоны получают свои инструкции и не сообщают о них до тех пор, пока они не будут выполнены, при этом вся информация о состоянии будет сообщена сразу. Миньоны вызывают сервер для получения данных конфигурации, но, помимо выдачи данных о базовых элементах, сам сервер выполняет относительно небольшие вычисления.

Как только ваша организация перерастет размеры одного главного сервера системы Salt, вы можете преобразовать свою инфраструктуру в многоуровневую или реплицированную схему сервера. Мы не рассматриваем эти варианты в данной книге, но их легко настроить.

Крупные развертывания являются сравнительно слабым местом для системы Ansible. Он включает часть функций, которые помогут вам внедрить многоуровневые серверные системы, но переход к этой модели не так прозрачен, как в Salt.

Система Ansible на порядок медленнее, чем Salt, и из-за своей архитектуры она должна обрабатывать клиенты партиями. Однако большинство серверов могут обрабатывать гораздо больше, чем пять одновременных клиентов по умолчанию. Вы также можете изменить стратегию выполнения Ansible, чтобы клиенты не держались в строгом порядке друг с другом. Даже настроенная система Ansible не будет приближаться к скорости системы Salt, но она лучше, чем можно было бы наивно ожидать.

Встроенные модули и расширяемость

Разработчики программного обеспечения для управления конфигурацией иногда пытаются сравнивать количество типов операций, которые различные системы поддерживают автоматически. Однако эти сравнения трудно осуществить корректно из-за основополагающих структурных различий. Например, функции, которые распространяются по нескольким модулям в системе Ansible, могут быть сосредоточены в одном модуле в системе Salt Атомарная операция в одной системе может соответствовать нескольким операциям в другой.

В настоящее время системы Salt и Ansible примерно сопоставимы в этом отношении. В дополнение к обширным стандартным библиотекам, обе системы имеют структуру для внедрения модулей, написанных сообществом разработчиков, в ядро или легкодоступный дополнительный пакет. В любом случае, общее количество модулей не имеет значения почти так же, как охват систем и программного обеспечения, которые фактически используются в вашей организации. Все системы CM полностью управляют базовыми операциями, но по мере глубокого погружения в них вы будете обнаруживать все новые и сильно различающиеся возможности.

Вероятно, вы в конечном итоге захотите решить часть задач, для которых в вашей системе CM не существует готового решения. К счастью, системы Salt и Ansible легко расширяются с помощью собственного кода Python. Учтите эту расширяемость на ранней стадии и при необходимости используйте ее.

Безопасность

Как указано в разделе "Параметры доступа в системе Ansible", ее можно сделать практически сколь угодно безопасной. Единственный предел безопасности — ваша собственная готовность заново набирать пароли и справляться с бюрократией безопасности.

Система хранения данных Ansible позволяет хранить данные конфигурации в зашифрованном формате. Это на самом деле довольно большое достижение, означающее, что ни сервер Ansible, ни база конфигурации не должны быть особенно безопасными. (Модульная архитектура Salt, вероятно, позволяет легко добавить эту функцию, но она не входит в первоначальный комплект поставки.)

Напротив, система Salt может быть лишь настолько же безопасной, насколько безопасной является учетная запись `root` на главном сервере. Хотя сам главный демон прост в настройке, сервер, на котором он запускается, должен получать самую сильную защиту в вашей организации. В идеале мастер должен быть машиной или виртуальным сервером, специально предназначенным для этой задачи.

На практике администраторы ненавидят слишком навязчивые протоколы безопасности, также как и все остальные пользователи. Большинство реальных объектов Ansible имеют относительно слабую защиту. Подобно тому, как система Ansible может быть сколь угодно сильно защищенной, ее также можно сделать сколь угодно небезопасной. Даже если вы попытаетесь полностью защитить систему Ansible, возможно, вам не удастся сохранить этот подход, как только рост вашей организации достигнет того момента, когда управление конфигурацией может выполняться путем ввода команд в окне терминала администратором. Например, любое задание планировщика `cron` может требовать от администратора ввода пароля. Работа над этим ограничением неизбежно приводит к снижению безопасности до уровня учетной записи `root`.

Суть безопасности заключается в том, что система Ansible дает вам больше как полезных, так и вредных возможностей. Она допускает более гибкое управление уровнями безопасности, но это не обязательно означает, что она более безопасна. Любая система подходит для средней организации. Учитывайте ваши собственные потребности и ограничения при оценке этих систем.

Разное

Некоторые дополнительные сильные и слабые стороны систем Ansible и Salt приведены в табл. 23.6 и 23.7.

Таблица 23.6. Преимущества и недостатки системы Ansible

Преимущества	Недостатки
Требуется только SSH и Python; нет демонов	Очень медленно работает
Четкая и краткая документация	Требует мощного сервера; труднее масштабировать
Встроенные циклы и условные обозначения, минимальный Jinja	Множество файлов с одинаковыми именами
Отлично работает от имени непривилегированного пользователя	Специфический синтаксис YAML
Операции могут использовать выходные данные друг друга	Ясное и гибкое использование конфигурационных каталогов.
Высокая безопасность; возможность шифрования	Много разных областей видимости переменных
Роли могут создаваться несколько раз	Отсутствие демонов означает меньшее количество возможностей
Более крупная база пользователей, чем у системы Salt	Нет реальной поддержки сред

Таблица 23.7. Преимущества и недостатки системы Salt

Преимущества	Недостатки
Быстро работает	Зависит от Jinja
Проще, чем Ansible	Странно организованная документация
Гибкие, непротиворечивые привязки	Плохая поддержка непривилегированных пользователей
Интегрированная поддержка облачных серверов	Формулы не могут быть включены более одного раза
Краткий синтаксис конфигурации	Нет встроенного решения для шифрования
Многоуровневые серверные развертывания	Нет доступа к результатам операций
Контроль структурированных событий	Минимальная поддержка стандартных значений переменных
Журналы выполнения легко экспортируются	Требуются явные объявления зависимостей
Исключительно модульная	Исключительно модульная

23.8. Рекомендации

Если вы работаете над программным проектом, то можете обнаружить, что многие проблемы, связанные с системами управления конфигурацией, характерны для разработки программ. Среда разработки имеет много схожих характеристик: несколько платформ, несколько версий продукта, полученных из одной и той же кодовой базы, несколько типов сборок и конфигураций и развертывание с помощью последовательных этапов разработки, тестирования и производства.

Это сложные вопросы, а среды разработки — это всего лишь инструменты. Разработчики используют множество дополнительных элементов управления — руководящие принципы разработки, принципы проектирования, стандарты кодирования, внутреннюю документацию и четкие архитектурные границы, среди прочего — ограничение скольжения по направлению к энтропии.

К сожалению, администраторы часто блуждают по территории управления конфигурацией без надлежащего набора инструментов разработчика. На первый взгляд управле-

ние конфигурацией кажется обманчиво простым, как немного более общий и сложный способ подходить к рутинным задачам сценариев. Поставщики систем управления конфигурацией стараются усилить это впечатление. Их веб-сайты по легкости и изяществу можно сравнить с песнями сирен; на каждом из них есть учебное пособие, в котором показано, как развернуть веб-сервер, выполнив всего десять строк кода конфигурации.

На самом деле край пропасти может быть ближе, чем кажется, особенно когда несколько администраторов вносят вклад в одну и ту же конфигурационную базу с течением времени. Реальные характеристики даже для одного целевого сервера выполняются сотнями строк кода, разделенных на несколько разных функциональных ролей. Без координации легко превратить систему СМ в путаницу конфликтующего или параллельного кода.

Рекомендации зависят от системы управления конфигурацией и среды, но для большинства ситуаций применяется несколько правил.

- Держите базу конфигураций под управлением системы контроля версий. Это не столько полезная рекомендация, сколько базовое требование корректности СМ-системы. Система Git не только отслеживает изменения и историю изменений, но и решает многие механические проблемы, связанные с координацией проектов через административные границы.
- Базы конфигураций по своей сути иерархичны, по крайней мере в логическом смысле. Некоторые стандарты применяются во всех организациях, некоторые применимы к каждому серверу в определенном отделе или регионе, а некоторые относятся к конкретным хостам. Кроме того, вам, скорее всего, понадобится возможность в определенных случаях делать исключения. В зависимости от операций вашей организации вам также может потребоваться поддержка нескольких независимых иерархий.
- Планируйте всю эту структуру заранее и рассмотрите, как вы можете управлять сценариями, в которых разные группы управляют разными частями базы конфигурации. По крайней мере соглашения для классификации хостов (например, экземпляры EC2, хосты, подключенные к Интернету, серверы баз данных) должны быть скоординированы по всей организации и последовательно соблюдаться.
- Системы СМ позволяют хранить различные части базы конфигурации в разных каталогах или хранилищах. Однако эта структура обеспечивает небольшую фактическую выгоду и усложняет повседневную работу по настройке. Мы рекомендуем использовать одну большую интегрированную конфигурационную базу. Управление иерархией и координацию следует осуществлять на уровне Git.
- Конфиденциальные данные (ключи, пароли) не должны попадать в систему контроля версий, если они не зашифрованы даже в локальных репозиториях кода. Система Git, в частности, не предназначена для обеспечения безопасности. Ваша система СМ может иметь встроенные функции шифрования, но если нет, создайте свои собственные.
- Поскольку сервер конфигурации имеет доступ к учетной записи `root` на многих других хостах, он является одним из наиболее концентрированных источников риска для безопасности вашей организации. Разумно выделить специальный сервер для этой роли, который должен получить самую строгую защиту.
- Конфигурации должны выполняться без сообщений о побочных эффектах. Сценарии и команды оболочки обычно являются самыми крупными точками преткновения. Ознакомьтесь с документацией вашей системы СМ для получения

консультаций по этой теме, так как это одна из наиболее частых проблем, с которой сталкиваются пользователи.

- Не выполняйте тестирование на производственных серверах. Но выполняйте тесты! Легко развернуть тестовую систему в облаке или в среде Vagrant. Система Chef даже предлагает продуманную систему тестирования и разработки в виде среды Kitchen. Убедитесь, что ваша тестовая система соответствует вашим реальным системам и что в ней используются одни и те же образы машины и конфигурация сети.
- Ознакомьтесь с кодом дополнительных пакетов, которые вы получаете из открытых репозиториев. Дело не в том, что эти источники чем-то подозрительны; просто системы и соглашения сильно различаются. Во многих случаях вы обнаружите, что требуется несколько локальных настроек. Если вы можете обойти диспетчер пакетов CM и клонировать пакеты напрямую из репозитория Git, вы можете легко перейти на более поздние версии без потери настроек.
- Безжалостно разделяйте конфигурации. Каждый файл должен иметь четкую и единую цель. (Пользователи системы Ansible могут выбрать текстовый редактор, который хорошо сочетается с 50 различными файлами, все с именем `main.yml`.)
- Управляемые конфигурацией серверы должны управляться на 100%. Иначе говоря, не должно быть и намека на административную работу, которая была выполнена вручную и которую никто не знает, как реплицировать. Эта проблема возникает, прежде всего, при перемещении существующих серверов под управление конфигурацией.[25]
- Не позволяйте себе или вашей команде "временно отключать" систему CM на хосте или использовать ручной метод переопределения системы CM (например, установку неизменяемого атрибута в файле конфигурации, который обычно находится под контролем системы CM). Эти изменения неизбежно забываются, возникают беспорядки или сбои.
- Нетрудно открыть шлюз из существующих административных баз данных в систему управления конфигурацией, и это имеет большое значение. Системы CM предназначены именно для такого взаимодействия. Например, вы можете определить системных администраторов и их зоны деятельности в своей базе данных LDAP на всей территории организации и сделать эту информацию доступной в среде управления конфигурацией с помощью сценариев шлюза. В идеале каждая информация должна иметь один авторитетный источник.
- Системы CM отлично подходят для управления состоянием машин. Они не предназначены для согласованных действий, таких как операции развертывания программного обеспечения, хотя документация и даже некоторые примеры могут заставить вас поверить, что они есть. По нашему опыту, более подходящей является система непрерывного развертывания.
- В гибких облачных средах, где вычислительная мощность добавляется в ответ на запрос в реальном времени, время, которое требуется для загрузки нового хоста с помощью системы управления конфигурацией, может быть мучительно медлен-

[25]При преобразовании существующего сервера "снежинка" в управление конфигурацией может оказаться полезным клонировать исходную систему для использования в качестве основы для сравнения. Можно также выполнить несколько циклов управления конфигурацией и тестирования, чтобы учесть все особенности системы.

ным. Оптимизируйте его, включая пакеты и медленные элементы конфигурации в исходный образ машины, а не закачивая и устанавливая их во время загрузки.

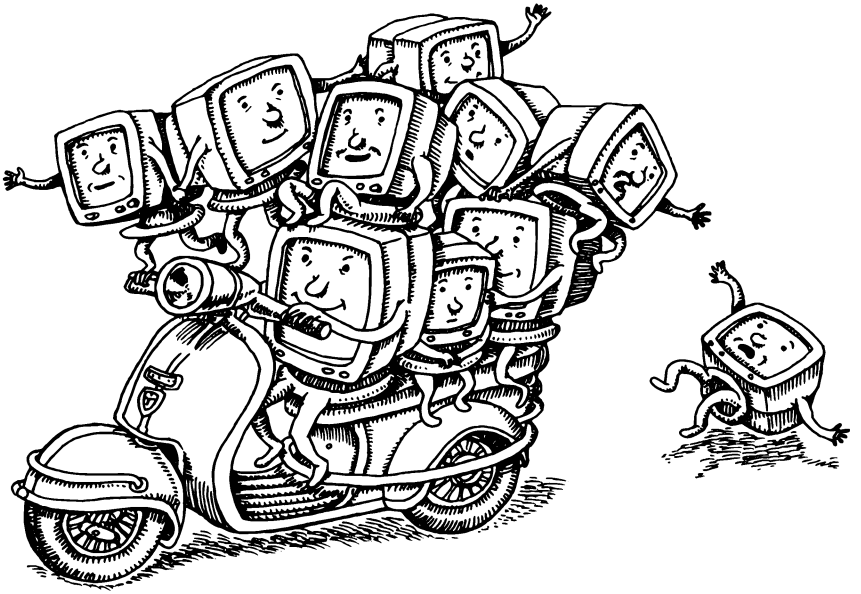
- Если вы используете настройку конфигурации для установки параметров конфигурации для приложения, убедитесь, что этот этап выполняется на ранней стадии процесса начальной загрузки, чтобы приложение стало работать быстрее. Для динамически масштабируемых хостов мы пытаемся ограничить время выполнения систем CM до менее 60 с.
- Как администратор, работающий с системой CM, выделите большую часть своего времени на создание кода CM, тестирование изменений в репрезентативном наборе систем, передачу обновлений в репозиторий и внесение поэтапных изменений в вашей организации. Чтобы быть наиболее эффективным, вы должны совершенствовать этот процесс, затрачивая время на изучение лучших рекомендаций и приемов для вашей системы.

23.9. Литература

- Cowie, Jon. *Customizing Chef: Getting the Most Out of Your Infrastructure Automation*. Sebastopol, CA: O'Reilly Media, 2014.
- Frank, Felix, and Martin Alfke. *Puppet 4 Essentials (2nd Edition)*. Birmingham, UK: Packt Publishing, 2015.
- Geerling, Jeff. *Ansible for DevOps: Server and configuration management for humans*. St. Louis, MO: Midwestern Mac, LLC, 2015. Эта книга ориентирована на основные противоречивые споры, но она включает в себя также некоторые полезные материалы об объединении системы Ansible с конкретными системами, такими как Vagrant, Docker и Jenkins.
- Hochstein, Lorin. *Ansible: Up and Running (2nd Edition)*. Sebastopol, CA: O'Reilly Media, 2017. Подобно книге *Ansible for DevOps*, эта книга охватывает как основы Ansible, так и взаимодействия с общими средами, такими как Vagrant и EC2. Некоторые основные моменты включают в себя более крупную примерную конфигурацию, главу о написании собственных модулей Ansible и советы по отладке.
- Morris, Kief. *Infrastructure as Code: Managing Servers in the Cloud*. Sebastopol, CA: O'Reilly Media, 2016. Эта книга содержит несколько специфических особенностей управления конфигурацией как таковой, но она помогает понять, как управление конфигурацией внедряется в более крупную схему DevOps и структурированное администрирование.
- Sebenik, Craig, and Thomas Hatch. *Salt Essentials*. Sebastopol, CA: O'Reilly Media, 2015. Это короткая и довольно схематичная книга, которая излагает основы системы Salt. Это не стиль книги, который мы обычно рекомендуем, но, учитывая необычность официальной документации, это потенциально полезная ссылка для тех, кто ищет "альтернативное мнение".
- Taylor, Mischa, and Seth Vargo. *Learning Chef: A Guide to Configuration Management and Automation*. Sebastopol, CA: O'Reilly Media, 2013.
- Uphill, Thomas, and John Arundel. *Puppet Cookbook (3rd Edition)*. Birmingham, UK: Packt Publishing, 2015.

Глава 24

Виртуализация



Виртуализация серверов позволяет одновременно запускать несколько экземпляров операционной системы на одном физическом оборудовании. Программное обеспечение виртуализации управляет ресурсами центрального процессора, памяти и устройств ввода-вывода, динамически распределяя их использование среди нескольких "гостевых" операционных систем и разрешая возникающие при этом конфликты. С точки зрения пользователя виртуальный сервер выглядит как полноценный физический сервер.

🕮 Дополнительную информацию о контейнерах см. в главе 25.

Эта концепция отделения аппаратных средств от операционной системы дает многочисленные выгоды. Виртуализированные серверы более гибкие, чем их физические аналоги. Они допускают перенос и могут управляться программно. Основное оборудование используется более эффективно, поскольку оно может обслуживать несколько гостевых операционных систем одновременно. И если этого недостаточно, технология виртуализации используется как в облачных вычислениях, так и в контейнерах.

Реализации виртуализации изменились с годами, но основные концепции не новы для отрасли. Фирма IBM (Big Blue) еще в конце 1960-х гг. использовала систему виртуальных машин на своих больших ЭВМ (мэйнфреймах), исследуя концепции разделения времени. Те же самые подходы использовались в период буйного расцвета мейнфреймов в 1970-х гг. до возникновения клиент-серверного бума 1980-х гг., когда сложность реализации виртуализации на основе архитектуры Intel x86 привела к короткому периоду относительного затишья.

Постоянно растущий размер серверных ферм вызвал интерес к виртуализации для современных систем. VMware и другие поставщики решили проблемы, связанные с процессорами x86, и упростили автоматическое предоставление операционных систем. В конечном итоге эти средства привели к росту систем управления виртуальными серверами, подключенными к Интернету и выделяющимися по мере необходимости. Все это создало инфраструктуру, которую мы теперь называем облачными вычислениями. Совсем недавно достижения в области виртуализации на уровне операционной системы открыли новую эру абстракции операционных систем в виде контейнеров.

В этой главе мы начнем с разъяснения терминов и понятий, необходимых для понимания виртуализации для систем UNIX и Linux. Затем мы опишем ведущие решения для виртуализации, используемые в наших примерах операционных систем.

24.1. Виртуальный жаргон

Терминология, используемая для описания виртуализации, несколько непрозрачна, в основном из-за того, как эволюционировала технология. Конкурирующие продавцы работали независимо друг от друга без использования стандартов, что приводило к появлению множества двусмысленных выражений и акронимов.[1]

Чтобы еще больше запутать проблему, сама "виртуализация" является перегруженным термином, который описывает больше, чем приведенный выше сценарий, в котором гостевые операционные системы работают в контексте виртуализованного оборудования. Виртуализация на уровне операционных систем, обычно называемая контейнеризацией, — это связанный, но другой набор средств, который стал столь же вездесущим, как и виртуализация серверов. Тем, кто не имеет практического опыта использования этих технологий, часто бывает трудно понять различия. Мы сравниваем эти два подхода далее в этом разделе.

Гипервизоры

Гипервизор (также известный как *монитор виртуальной машины*) представляет собой программный уровень, который посредничает между виртуальными машинами (VM) и базовым оборудованием, на котором они работают. Гипервизоры отвечают за совместное использование системных ресурсов среди гостевых операционных систем, которые изолированы друг от друга и которые получают доступ к оборудованию исключительно через гипервизор.

Гостевые операционные системы независимы, поэтому они не должны быть одинаковыми. Например, CentOS может работать вместе с FreeBSD и Windows. Примерами гипервизоров являются VMware ESX, XenServer и FreeBSD bhyve. Виртуальная машина на базе ядра Linux (KVM) преобразует ядро Linux в гипервизор.

Полная виртуализация

Первые гипервизоры полностью эмулировали базовое оборудование, определяя виртуальные подстановки для всех основных вычислительных ресурсов: жестких дисков, сетевых устройств, устройств управления прерываниями, материнских плат, BIOS и т.д. В этом режиме, называемом *полной виртуализацией*, гостевые операционные системы запускают-

[1]Здесь приходит в голову закон Конвея (Conway): "Организации, которые проектируют системы, вынуждены создавать проекты, которые являются копиями коммуникационных структур этих организаций".

ся без каких-либо изменений, что существенно снижает их скорость работы, потому что гипервизор должен постоянно выполнять преобразование между реальным оборудованием компьютера и виртуальным оборудованием гостевых операционных систем.

Имитация всего персонального компьютера — сложная задача. Большинство гипервизоров, которые предлагают полную виртуализацию, отделяют задачу поддержки нескольких сред (виртуализации) от задачи имитации оборудования в каждой среде (эмуляция).

Наиболее распространенным пакетом эмуляции, используемым в этих системах, является проект с открытым исходным кодом под названием QEMU. Более подробную информацию вы можете найти на сайте `qemu.org`, но в большинстве случаев эмулятор не требует пристального внимания со стороны администраторов.

Паравиртуализация

Паравиртуализация — это технология, используемая виртуальной платформой с открытым исходным кодом Xen, в которой модифицированная гостевая операционная система распознает свое виртуальное состояние и активно взаимодействует с гипервизором для организации доступа к аппаратному обеспечению. Этот подход значительно повышает производительность, однако для этого гостевые операционные системы должны быть сильно модифицированы, причем эти модификации зависят от конкретного гипервизора.

Аппаратная виртуализация

В 2004 и 2005 гг. компании Intel и AMD представили функции процессора (Intel VT и AMD-V), которые способствовали виртуализации на платформе x86. Эти расширения позволяли реализовать "аппаратную виртуализацию", также известную как "ускоренная виртуализация". В этой схеме процессор и контроллер памяти виртуализируются с помощью аппаратного обеспечения, хотя и под управлением гипервизора. Производительность такой системы очень хорошая, и гостевые операционные системы не должны знать, что они работают на виртуальном процессоре. В наши дни виртуализация с помощью аппаратного обеспечения является основным трендом.

Хотя центральный процессор является основной точкой соприкосновения между аппаратным обеспечением и гостевыми операционными системами, это всего лишь один из компонентов системы. Гипервизор все еще нужен в некотором роде для представления или эмуляции остальной части аппаратного обеспечения системы. Для этой задачи можно использовать полную виртуализацию или паравиртуализацию. В некоторых случаях используется сочетание подходов; это зависит от степени "осведомленности" гостевой системы о виртуализации.

Паравиртуализованные драйверы

Одно большое преимущество аппаратной виртуализации заключается в том, что она в значительной степени ограничивает необходимость поддержки паравиртуализации на уровне драйверов устройств. Все операционные системы позволяют добавлять дополнительные драйверы, поэтому настройка гостевой системы с паравиртуализованными дисками, видеокартами и сетевыми интерфейсами так же проста, как установка соответствующих драйверов. Драйверы знают "секретный протокол", который позволяет им использовать функции поддержки паравиртуализации гипервизора, а гостевая операционная система остается не в курсе дела.

Несколько досадных аспектов архитектуры персональных компьютеров, таких как контроллер прерываний и ресурсы BIOS, не попадают в категорию процессоров или драйверов устройств. В прошлом преобладающим был подход, который заключался в том, чтобы реализовать эти оставшиеся компоненты посредством полной вирту-

ализации. Например, режим HVM Xen (аппаратная виртуальная машина) объединяет поддержку расширений виртуализации на уровне процессора с копией эмулятора ПК QEMU. Режим PVHVM (ParaVirtualized HVM) добавляет к этой схеме паравиртуализированные драйверы в гостевых операционных системах, что значительно сокращает объем полной виртуализации, необходимый для поддержания работы системы. Тем не менее гипервизору по-прежнему нужна активная копия QEMU для каждой виртуальной машины, чтобы она могла покрывать возможности и цели, не затронутые паравиртуализованными драйверами.

Современная виртуализация

В самых последних версиях Xen и других гипервизоров более или менее устранена необходимость в эмуляции устаревшего оборудования. Вместо этого они полагаются на функции виртуализации на уровне процессора, паравиртуализованные драйверы гостевой операционной системы и несколько дополнительных разделов паравиртуализованного кода в гостевых ядрах. Xen называет этот режим PVH (ParaVirtualized Hardware), и он считается близким к идеальной комбинации, которая дает оптимальную производительность, но выдвигает минимально возможные требования к гостевым ядрам.

На практике или при чтении документации вы можете столкнуться с любым из вариантов виртуализации, описанных выше. Однако не стоит запоминать какую-либо конкретную терминологию или слишком беспокоиться о виртуализационных режимах. Границы между этими режимами являются нечеткими, и в гипервизоре обычно реализованы лучшие варианты для данной гостевой операционной системы. Если вы обновите свое программное обеспечение, то автоматически получите выгоду от последних улучшений. Единственной причиной выбора чего-либо, кроме режима работы по умолчанию, является поддержка старого оборудования или устаревших гипервизоров.

Типы гипервизоров

Во многих справочных материалах сделаны несколько сомнительные различия между гипервизорами "типа 1" и "типа 2". Гипервизор типа 1 работает непосредственно на аппаратном обеспечении без поддержки операционной системы, и по этой причине его иногда называют аппаратным или машинно-зависимым гипервизором. Гипервизоры типа 2 — это приложения для пользовательского пространства, которые работают поверх другой операционной системы общего назначения. Эти две модели показаны на рис. 24.1.

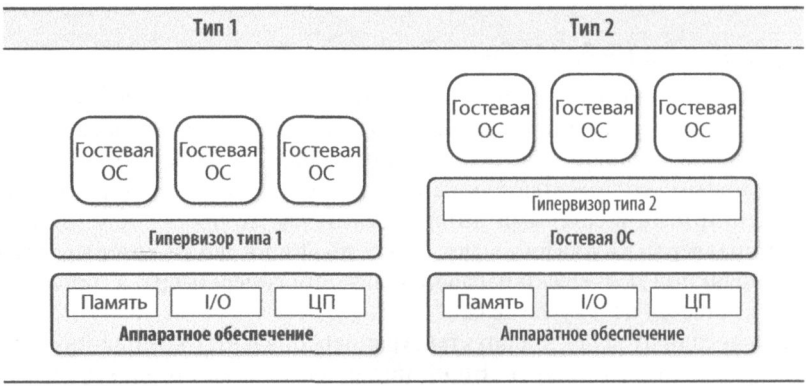


Рис. 24.1. Сравнение гипервизоров типа 1 и 2

Гипервизоры VMware ESXi и XenServer рассматриваются как тип 1, а FreeBSD bhyve — как тип 2. Аналогично пакеты виртуализации, ориентированные на рабочую станцию, такие как VirtualBox и VMware Workstation от компании Oracle, также относятся к типу 2.

Это правда, что системы типа 1 и 2 отличаются друг от друга, но разграничение не всегда такое четкое. Например, гипервизор KVM представляет собой модуль ядра Linux, который предоставляет виртуальным машинам прямой доступ к функциям виртуализации процессора. Дифференциация между типами гипервизоров имеет скорее академическое, чем практическое значение.

Динамическая миграция

Виртуальные машины могут перемещаться между гипервизорами, работающими на разных физических устройствах, в реальном времени, в некоторых случаях без перерывов в обслуживании или потери связи. Эта функция называется *динамической миграцией*. Ее секрет заключается в переносе памяти между исходными и целевыми хостами. Гипервизор копирует изменения из источника в пункт назначения, и как только память становится идентичной, миграция завершается. Динамическая миграция полезна для балансировки нагрузки в системах высокой доступности, аварийного восстановления, обслуживания серверов и общей гибкости системы.

Образы виртуальных машин

Виртуальные серверы создаются из образов, которые представляют собой шаблоны настроенных операционных систем, которые может загружать и выполнять гипервизор. Формат файла образа зависит от гипервизора. Большинство проектов гипервизоров поддерживают коллекцию образов, которые вы можете загрузить и использовать в качестве основы для собственных настроек. Вы также можете сделать мгновенный снимок виртуальной машины для создания образа с целью резервного копирования важных данных, либо для использования в качестве основы для создания дополнительных виртуальных машин.

Поскольку оборудование виртуальной машины, представленное гипервизором, стандартизировано, образы можно переносить с системы на систему, даже если их реальное оборудование различается. Образы специфичны для конкретного гипервизора, но существуют инструменты для переноса образов между гипервизорами.

Контейнеризация

Виртуализация на уровне операционной системы, или контейнеризация, — это другой подход к изоляции, в котором не используется гипервизор. Вместо этого используются функции ядра, которые позволяют изолировать процессы от остальной системы. Каждый контейнер процесса (container, или jail) имеет персональную корневую файловую систему и пространство имен процессов. Процессы, содержащиеся в контейнере, совместно используют ядро и другие службы хост-системы, но они не могут обращаться к файлам или ресурсам за пределами своих контейнеров. Эта архитектура проиллюстрирована на рис. 24.2.

🕮 Дополнительную информацию о контейнерах см. в главе 25.

Поскольку для контейнеризации не требуется виртуализация, накладные расходы на ресурсы для виртуализации на уровне операционной системы низки. Большинство реализаций предлагают почти естественную производительность. Однако этот тип вир-

туализации исключает использование нескольких операционных систем, поскольку ядро хоста используется всеми контейнерами.[2] Примерами реализации контейнеров являются контейнеры LXC и Docker в системе Linux, а также Jail в системе FreeBSD.

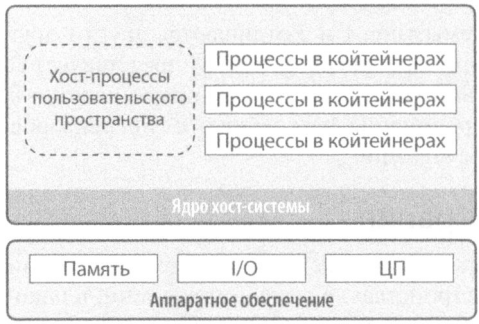


Рис. 24.2. Контейнеризация

Легко перепутать контейнеры с виртуальными машинами. Оба определяют переносимые изолированные среды исполнения, и оба выглядят и действуют как полноценные операционные системы с корневыми файловыми системами и запущенными процессами. Однако их реализация совершенно различна.

Истинная виртуальная машина имеет ядро операционной системы, процесс инициализации, драйверы для взаимодействия с оборудованием и полные атрибуты операционной системы UNIX. С другой стороны, контейнер является просто фасадом операционной системы. Он использует стратегии, описанные выше, чтобы дать отдельным процессам подходящую среду исполнения. В табл. 24.1 иллюстрируются некоторые практические различия между этими концепциями.

Таблица 24.1. Сравнение виртуальных машин и контейнеров

Виртуальная машина	Контейнер
Полнофункциональная операционная система, которая совместно использует базовое оборудование с помощью гипервизора	Отдельная группа процессов, управляемых совместно используемым ядром
Требуется полная процедура начальной загрузки для инициализации; начинает работу через 1-2 минуты	Процессы запускаются непосредственно ядром; не требуется начальная загрузка; начинает работу менее чем через 1 секунду
Долговременные	Часто заменяемые
Имеет один или несколько виртуальных дисков, выделенных гипервизором	Образ файловой системы представляет собой многоуровневую структуру, определенную контейнером
Размер образа измеряется в гигабайтах	Размер образа измеряется в мегабайтах
Не больше нескольких десятков на каждый физический хост	Большое количество на каждый виртуальный или физический хост
Полная изоляция гостевых операционных систем	Ядро операционной системы и службы совместно используются хостом
Несколько независимых операционных систем, работающих одновременно	В контейнере должно использоваться то же ядро, что и на хосте (версии операционных систем могут различаться)

[2]Это не совсем так. Уровень эмуляции Linux в системе FreeBSD допускает использование контейнеров Linux на хостах FreeBSD.

Обычно контейнеры используются в сочетании с виртуальными машинами. Виртуальные машины — лучший способ разделить физические серверы на управляемые части. Затем вы можете запускать приложения в контейнерах на виртуальных машинах для достижения оптимальной плотности системы (эту процедуру иногда называют "упаковкой контейнера" (bin packing). Архитектура контейнеров на виртуальной машине является стандартной для контейнеризованных приложений, которые должны запускаться в общедоступных облачных средах.

В остальной части этой главы мы сосредоточимся на настоящей виртуализации. Более подробно контейнеризация описана в главе 25.

24.2. Виртуализация с помощью системы Linux

Ведущими проектами с открытым кодом по виртуализации системы Linux являются две платформы: Xen и KVM. Платформа Xen теперь является частью проекта Linux Foundation и используется для реализации крупнейших открытых облачных платформ, в том числе Web Services компании Amazon и SoftLayer компании IBM. Платформа KVM — виртуальная машина, интегрированная в ядро системы Linux. Системы Xen и KVM продемонстрировали свою стабильность на примере многих промышленных инсталляций в крупных организациях.

Платформа Xen

Изначально Linux-ориентированная платформа Xen была разработана Яном Праттом (Ian Pratt) как исследовательский проект Кембриджского университета (University of Cambridge). Со временем она стала мощной платформой для виртуализации, которая благодаря производительности, безопасности и дешевизне оказалась способной конкурировать даже с коммерческими гигантами.

Накладные расходы, связанные с эксплуатацией паравиртуального гипервизора виртуальной машины Xen, составляют 0,1–3,5%, что намного меньше, чем у полностью виртуализованных платформ. Поскольку гипервизор Xen является программой с открытым исходным кодом, существует множество средств с разными уровнями поддержки ее функциональных возможностей. Исходный код платформы Xen можно загрузить на сайте `xenproject.org`. Кроме того, она входит во многие дистрибутивные пакеты Linux.

Xen — это автономный гипервизор, который функционирует непосредственно на физическом аппаратном обеспечении. Работающая виртуальная машина называется *доменом*. Всегда существует по крайней мере один домен, который называется *нулевым* (или `dom0`). Нулевой домен имеет полный доступ к аппаратному обеспечению, управляет другими доменами, в нем запускаются драйверы всех устройств. Непривилегированные домены называются `domU`.

В домене `dom0` обычно запускается дистрибутив Linux. Он выглядит, как и все остальные дистрибутивы системы Linux, но включает демоны, средства и библиотеки, которые дополняют архитектуру Xen и обеспечивают взаимодействие между доменами `domU`, `dom0` и гипервизором.

Гипервизор отвечает за распределение времени центрального процессора и управление памятью для всей системы в целом. Он также управляет всеми доменами, включая `dom0`. Однако следует заметить, что работа самого гипервизора контролируется и управляется из домена `dom0`. Как все запутанно!

Наиболее интересные части домена `dom0` для системы Linux перечислены в табл. 24.2.

В каждом конфигурационном файле гостевого домена, который находится в каталоге **/etc/xen** указываются виртуальные ресурсы, доступные для домена domU, включая дисковые устройства, процессор, память и сетевые интерфейсы. Каждый домен domU имеет отдельный файл конфигурации. Формат является гибким и дает администраторам хороший контроль над ограничениями, применяемыми к каждой гостевой системе. Если символическая ссылка на файл конфигурации domU добавляется в подкаталог **auto**, эта гостевая операционная система автоматически запускается во время начальной загрузки.

Таблица 24.2. Компоненты платформы Xen

Путь	Назначение
/etc/xen	Каталог основной конфигурации
auto	Конфигурационные файлы гостевой операционной системы для автоматического запуска во время начальной загрузки
scripts	Вспомогательные сценарии, создающие сетевые интерфейсы и т.д.
/var/log/xen	Файлы журналов платформы Xen
/usr/sbin/xl	Средство для управления гостевым доменом на платформе Xen

Инсталляция гостевой операционной системы на платформе Xen

Создание гостевого сервера и его запуск на платформе Xen состоит из нескольких этапов. Для упрощения этой процедуры мы рекомендуем использовать программу **virt-manager** (virt-manager.org). Программа **virt-manager** изначально была частью проекта системы Red Hat, но в настоящее время ее код стал открытым и доступным для большинства дистрибутивов Linux. В него входит утилита **virt-install**, запускаемая из командной строки для инсталляции операционной системы. Она может использовать различные источники для инсталляции ОС, например устройства сетевой файловой системы SMB или NFS, физические CD- или DVD-приводы или URL HTTP-ресурса.

Диски гостевых доменов обычно хранятся в виртуальных блочных устройствах (VBD) в домене dom0. VBD можно подключить к выделенному ресурсу, например физическому диску или логическому тому. Кроме того, это может быть файл LoopBack, также известный как VBD-файл, созданный с помощью команды **dd**. Выделенный диск или том обеспечивают более высокую производительность, но файлы — более гибкий инструмент, которым можно управлять с помощью обычных команд Linux (таких как **mv** и **cp**) в домене dom0. VBD-файлы — это разреженные файлы, которые увеличиваются по мере необходимости.

Если в системе нет проблем с производительностью, VBD с файловой поддержкой, как правило, является лучшим выбором. Существует простой процесс переноса VBD-файлов на выделенный диск на случай, если вы передумаете.

Например, инсталляция гостевого домена может быть выполнена с помощью следующей команды.

```
$ sudo virt-install -n chef -f /vm/chef.img -l http://example.com/myos
    -r 512 --nographics
```

Это типичный гостевой домен на платформе Xen с именем chef, дисковым устройством VBD, размещенным в каталоге **/vm/chef.img**, и средой инсталляции, полученной с помощью протокола HTTP. Этот экземпляр имеет 512 двоичных Мибибайт (MiB) оперативной памяти и не использует во время инсталляции графическую поддержку X Windows. Утилита **virt-install** загружает файлы, необходимые для начала инсталляции, а затем запускает процесс инсталлятора.

Вы увидите пустой экран и пройдете через все стандартные этапы инсталляции системы Linux в текстовом режиме, включая конфигурирование сети и выбор пакета. После инсталляции гостевой домен перезагружается и готов к использованию. Для отсоединения от гостевой консоли и возвращения в домен dom0 нажмите <Ctrl+]>.

Несмотря на то что в данном примере используется текстовая инсталляция, возможна также графическая поддержка этого процесса с помощью системы Virtual Network Computing (VNC).

Конфигурация домена хранится в файле **/etc/xen/chef**. Этот файл выглядит следующим образом.

```
name = "chef"
uuid = "a85e20f4-d11b-d4f7-1429-7339b1d0d051"
maxmem = 512
memory = 512
vcpus = 1
bootloader = "/usr/bin/pygrub"
on_poweroff = "destroy"
on_reboot = "restart"
on_crash = "restart"
vfb = [ ]
disk = [ "tap:aio:/vm/chef.dsk,xvda,w" ]
vif = [ "mac=00:16:3e:1e:57:79,bridge=xenbr0" ]
```

Как видим, по умолчанию сетевой интерфейс настроен на мостовой режим. В нашем случае устройство VBD представляет собой "блочный" файл, обеспечивающий более высокую производительность, чем стандартный LoopBack файл. Файл образа диска, допускающий запись, предоставлен гостевой ОС под именем **/dev/xvda**.

Утилита **xl** удобна для ежедневного управления виртуальными машинами, например для их запуска и остановки, подключения к их консолям и исследования текущего состояния. Ниже мы выводим список запущенных гостевых доменов, после чего подключаемся к консоли машины chef домена domU. Идентификаторы присваиваются в порядке возрастания по мере создания гостевых доменов. При перезагрузке хоста их нумерация сбрасывается.

```
$ sudo xl list
Name        ID   Mem(MiB)   VCPUs   State    Time(s)
Domain-0    0    2502       2       r-----   397.2
chef        19   512        1       -b----   12.8
$ sudo xl console 19
```

Для того чтобы изменить конфигурацию гостевого домена (например, подключить другой диск или перевести сеть из мостового режима в режим NAT), необходимо отредактировать гостевой конфигурационный файл в каталоге **/etc/xen** и перезагрузить гостевую систему.

Платформа KVM

KVM (Kernel-based Virtual Machine) — это средство полной виртуализации, включаемое по умолчанию в большинство дистрибутивов системы Linux. Необходимым условием его применения является наличие расширений центрального процессора Intel VT и AMD-V для поддержки виртуализации. В типичных сценариях установки для реализации полностью виртуальной системы аппаратного обеспечения используется QEMU. Хотя эта система изначально была предназначена для Linux, ее можно перенести в систему FreeBSD в виде загружаемого модуля ядра.

Поскольку в системе KVM по умолчанию реализован режим полной виртуализации, поддерживаемый на уровне аппаратного обеспечения центрального процессора, под ее управлением могут работать многие гостевые операционные системы, включая Windows. Существуют драйверы паравиртуализованной сетевой платы Ethernet, диска и графической карты для систем Linux, FreeBSD и Windows. Их использование не обязательно, но рекомендуется для повышения производительности.

На платформе KVM ядро операционной системы Linux функционирует как гипервизор; управление памятью и диспетчеризация выполняются с помощью ядра хоста, а гостевые машины представляют собой обычные процессы Linux. Этот уникальный подход к виртуализации сулит огромные преимущества. Например, сложность, порождаемая многоядерными процессорами, устраняется с помощью ядра системы, и для их поддержки не требуется вносить никаких изменений в гипервизор. Команды Linux, например **top**, **ps** и **kill**, управляют виртуальными машинами так, будто они являются обычными процессами. Интеграция с системой Linux является безупречной.

Инсталляция гостевой операционной системы на платформе KVM и ее использование

Несмотря на то что технологии, лежащие в основе платформ Xen и KVM, принципиально отличаются друг от друга, инструменты, инсталлирующие гостевые операционные системы и управляющие ими, похожи друг на друга. Как и на платформе Xen, для создания новых гостевых систем по технологии KVM можно использовать программу **virt-install**, а для управления ими — команду **virsh**.

Флаги, передаваемые программе **virt-install**, могут немного отличаться от флагов, используемых при инсталляции платформы Xen. Например, флаг **--hvm** означает, что для виртуализации гостевой системы должно использоваться аппаратное обеспечение, а не паравиртуализация. Кроме того, аргумент **--connect** гарантирует, что по умолчанию будет выбран правильный гипервизор, потому что утилита **virt-install** поддерживает несколько гипервизоров. В заключение, чтобы получить выгоду от возможностей платформы KVM, рекомендуется использовать аргумент **--accelerate**. Следовательно, пример полной команды для инсталлирования гостевого сервера Ubuntu с диска CD-ROM должен выглядеть следующим образом.

```
$ sudo virt-install --connect qemu:///system -n UbuntuYakkety
  -r 512 -f ~/ubuntu-Yakkety.img -s 12 -c /dev/dvd --os-type linux
  --accelerate --hvm --vnc
Would you like to enable graphics support? (yes or no)
```

Если дистрибутивный DVD-диск для системы Ubuntu вставлен в дисковод, эта команда запускает процесс инсталляции и сохраняет гостевую систему в файле **~/ubuntu-Yaketty.img**, позволяя увеличивать его размер до 12 Гбайт. Поскольку мы не указали ни флаг **--nographics**, ни флаг **--vnc**, утилита **virt-install** спрашивает, включать ли поддержку графики.

Утилита **virsh** запускает собственную оболочку, которая выполняет команды. Для того чтобы открыть эту оболочку, наберите команду **virsh --connect qemu:///system**. Следующая серия команд демонстрирует некоторые основные функциональные возможности утилиты **virsh**. Для того чтобы увидеть полный список этих команд, наберите команду **help** в оболочке или просмотрите справочную страницу.

```
$ sudo virsh --connect qemu:///system
virsh# list --all
   Id   Name                State
```

```
-----------------------------------------
   3   UbuntuYakkety      running
   7   CentOS             running
   -   Windows2016Server  shut off

virsh# start Windows2016Server
Domain WindowsServer started

virsh# shutdown CentOS
Domain CentOS is being shutdown

virsh# quit
```

24.3. Система FreeBSD bhyve

В версию FreeBSD 10.0 была включена относительно новая система виртуализации — bhyve. Она может запускать гостевые операционные системы BSD, Linux и даже Windows. Однако она работает с ограниченным набором аппаратных средств и у нее отсутствуют некоторые основные функции, существующие в других реализациях.

Учитывая большое количество платформ виртуализации на рынке, которые уже поддерживают систему FreeBSD, непонятно, почему усилия bhyve начались, когда они уже появились. Если вы не разрабатываете собственную платформу, требующую встроенной виртуализации FreeBSD, мы рекомендуем выбрать другое решение, пока этот проект не созреет.

24.4. Компания VMWare

Компания VMware является крупнейшим игроком в индустрии виртуализации и первым поставщиком, который разрабатывает технологии для виртуализации требовательной платформы x86. Компания VMware является коммерческим предприятием, но некоторые из ее продуктов бесплатны. Все они заслуживают внимания при выборе технологии виртуализации всего сайта.

Основным продуктом, представляющим интерес для администраторов UNIX и Linux, является ESXi[3], который представляет собой простой гипервизор для архитектуры Intel x86. Гипервизор ESXi является бесплатным, но некоторые полезные функции ограничены платными лицензиями.

В дополнение к гипервизору ESXi компания VMware предлагает несколько мощных, продвинутых продуктов, которые облегчают централизованное развертывание и управление виртуальными машинами. У них также есть самая зрелая технология динамической миграции, которую мы уже видели. Однако углубленное освещение полного набора продуктов VWware выходит за рамки этой главы.

24.5. Гипервизор VirtualBox

VirtualBox — это кросс-платформенный гипервизор второго типа. Он выполняет, вероятно, достаточно хорошую виртуализацию гостевых систем, используемых, как правило, частными лицами. Он популярен среди разработчиков и конечных пользовате-

[3]Аббревиатура ESXi расшифровывается как "Elasic Sky X, integrated". Даже не пытайтесь понять, что это значит.

лей, поскольку является бесплатным, простым в установке, простым в использовании и часто упрощает создание и управление тестовыми средами. Его производительность и аппаратная поддержка являются слабыми. Гипервизор VirtualBox обычно не подходит для виртуализации производственных систем.[4]

История гипервизора VirtualBox длинная и запутанная. Он начинался как коммерческий продукт компании Innotek GmbH, но был выпущен как проект с открытым кодом, прежде чем компания Innotek была приобретена компанией Sun Microsystems в 2008 г. После того как компания Oracle поглотила Sun в 2010 г., продукт был переименован в Oracle VM VirtualBox. Гипервизор VirtualBox существует и сегодня (доступен по лицензии Open Source GPLv2) и продолжает активно развиваться компанией Oracle.

Гипервизор VirtualBox работает в системах Linux, FreeBSD, Windows, macOS и Solaris. Компания Oracle не публикует и не поддерживает версию для хоста FreeBSD, но его можно установить из набора портов. Поддерживаемые гостевые операционные системы включают Windows, Linux и FreeBSD.

По умолчанию виртуальные машины контролируются с помощью графического интерфейса VirtualBox. Если вы заинтересованы в запуске виртуальных машин в системе, которая не допускает графический интерфейс, изучите гипервизор VBoxHeadless, инструмент командной строки для VirtualBox. Вы можете скачать программу VirtualBox и узнать больше о ней на сайте `virtualbox.org`.

24.6. Программа Packer

Packer (**`packer.io`**), широко признанный проект с открытым исходным кодом компании HashiCorp, является инструментом для создания образов виртуальной машины из файла спецификации. Он может создавать образы для различных платформ виртуализации и облачных платформ. Внедрение Packer в рабочий процесс позволяет вам не заботиться о платформе виртуализации. Вы можете легко создать свой настроенный образ для любой платформы, которую вы используете в определенный день.

Чтобы создать образ, программа Packer запускает экземпляр из исходного образа по вашему выбору. Затем он настраивает экземпляр, запуская сценарии или вызывая другие указанные вами этапы. Наконец, он сохраняет копию состояния виртуальной машины в качестве нового образа.

Этот процесс особенно полезен для поддержки "инфраструктуры как кода" для управления серверами. Вместо того чтобы вручную применять изменения к образам, вы изменяете шаблон, который описывает образ в абстрактных терминах. Затем вы записываете спецификацию в репозиторий, как традиционный исходный код. Этот метод обеспечивает превосходную прозрачность, повторяемость и обратимость. Это также создает четкий контрольный журнал.

Конфигурации Packer — это файлы JSON. Большинство администраторов согласны с тем, что JSON представляет собой плохой выбор формата, поскольку он, как известно, придирчив к кавычкам и запятым и не допускает комментариев. Если повезет, компания HashiCorp скоро преобразует Packer в свой улучшенный настраиваемый формат конфигурации, но до тех пор вам придется редактировать файлы в формате JSON.

[4]Веб-сайт VirtualBox утверждает, что это профессиональное решение, которое лицензируется для использования на предприятиях. На самом деле это может быть справедливо в отношении операционных систем Oracle, которые являются единственными заранее созданными виртуальными машинами.

В шаблоне "сборщики" определяют, как создать образ, а "поставщики" позволяют настроить и установить программное обеспечение для образа. Существуют сборщики, созданные компаниями AWS, GCP, DigitalOcean, VMware, VirtualBox и Vagrant и многими другими. Поставщики могут быть сценариями оболочки, рецептурными книгами Chef, ролями Ansible или другими инструментами управления конфигурацией.

В приведенном ниже шаблоне, **custom_ami.json**, показано использование сборщика `amazon-ebs` компании AWS и поставщика `shell`.

```
{
   "builders": [{
      "type": "amazon-ebs",
      "access_key": "AKIAIOSFODNN7EXAMPLE",
      "secret_key": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
      "region": "us-west-2",
      "source_ami": "ami-d440a6e7",
      "instance_type": "t2.medium",
      "ssh_username": "ubuntu",
      "ssh_timeout": "5m",
      "subnet_id": "subnet-ef67938a",
      "vpc_id": "vpc-516b8934",
      "associate_public_ip_address": true,
      "ami_virtualization_type": "hvm",
      "ami_description": "ULSAH AMI",
      "ami_name": "ULSAH5E",
      "tags": {
         "Name": "ULSAH5E Demo AMI"
      }
   }],
   "provisioners": [
      {
         "type": "shell",
         "source": "customize_ami.sh"
      }
   ]
}
```

Как любому инструменту командной строки нужны определенные параметры для запуска экземпляра, сборщику `amazon-ebs` нужны такие данные, как мандаты API, тип экземпляра, исходный интерфейс AMI, на котором основывается новый образ, и подсеть VPC, в которой должен располагаться экземпляр. Для выполнения этапа поставки программа Packer использует службу SSH, поэтому мы должны убедиться, что экземпляру назначен общедоступный IP-адрес.

В этом примере сборщиком является сценарий оболочки под названием `customize_ami.sh`. Программа Packer копирует сценарий в удаленную систему с помощью команды **scp** и запускает его. В этом сценарии нет ничего особенного; он может делать все, что вы обычно делаете. Например, он может добавлять новых пользователей, загружать и настраивать программное обеспечение или выполнять шаги по укреплению системы безопасности.

Чтобы создать интерфейс AMI, выполните команду **packer build**:

```
$ packer build custom_ami.json
```

Команда **packer build** отмечает каждый шаг процесса сборки на консоли. Сборщик `amazon-ebs` выполняет следующие этапы.

1. Автоматически создает пару ключей и группу безопасности.
2. Запускает экземпляр и ждет, пока он станет доступным в сети.

3. Использует утилиты **scp** и **ssh** для выполнения запрошенных этапов инициализации.
4. Создает AMI, вызывая интерфейс EC2 CreateImage API.
5. Очищает память, прекращая работу экземпляра.

Если все работает правильно, программа Packer выводит идентификатор AMI, как только он будет доступен для использования. Если во время сборки возникает проблема, программа Packer выводит сообщение с ошибкой пурпурного цвета и выходит из системы после очистки памяти.

При использовании аргумента **-debug** в команде **packer build** система будет приостанавливаться на каждом этапе, чтобы вы могли устранить проблемы. Вы также можете использовать сборщик `null` для исправления любых ошибок, чтобы не запускать экземпляр каждый раз при попытке запустить сборку.

24.7. Программа Vagrant

Программа Vagrant также разработана компанией HashiCorp. Она является оболочкой для платформ виртуализации, таких как VMware, VirtualBox и Docker. Однако сама по себе она не является платформой виртуализации.

Программа Vagrant упрощает поставку и настройку виртуальной среды. Ее задача — быстро и легко создать переносимые, предварительно сконфигурированные среды разработки, которые тесно связаны с производственными средами. Эта функциональная возможность позволяет разработчикам писать и тестировать код с минимальным вмешательством со стороны системных администраторов или группы эксплуатации.

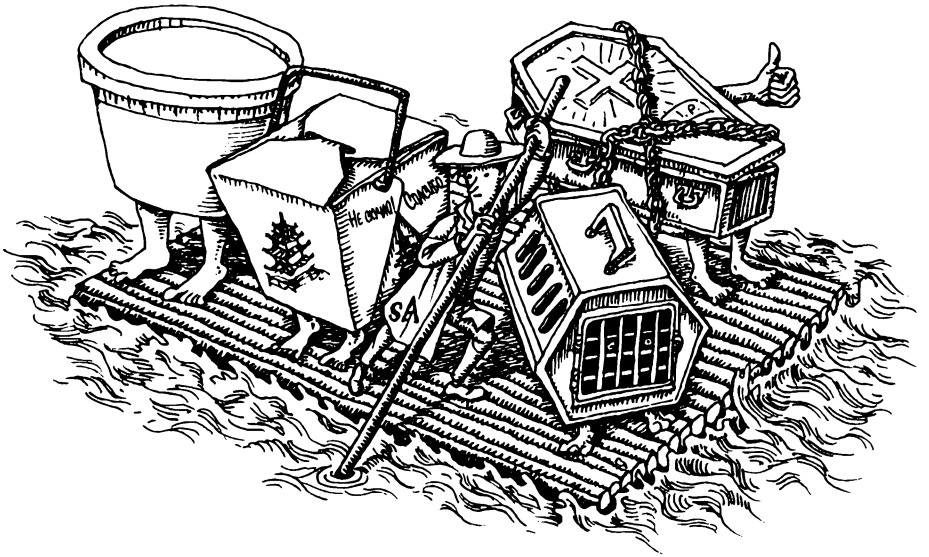
Можно (но не обязательно) использовать программу Vagrant в сочетании с программой Packer. Например, вы можете стандартизировать базовый образ, который вы используете для своих производственных платформ, с помощью программы Packer, а затем распространить сборку Vagrant этого образа среди разработчиков. Разработчики могут развернуть экземпляр образа на своем ноутбуке или облачном провайдере по своему выбору с любыми необходимыми настройками. Этот метод уравновешивает потребность в централизованном управлении производственными образами с потребностями разработчиков в доступе к аналогичной среде, которую они могут контролировать напрямую.

24.8. Литература

- Веб-сайт `virtualization.info` является отличным источником текущих новостей, тенденций и сплетен в области виртуализации и облачных вычислений.
- Hashimoto, Mitchell. *Vagrant: Up and Running: Create and Manage Virtualized Development Environments*. Sebastopol, CA: O'Reilly Media, 2013.
- Kusnetsky, Dan. *Virtualization: A Manager's Guide: Big Picture of the Who, What, and Where of Virtualization*. Sebastopol, CA: O'Reilly Media, 2011.
- Mackey, Tim, and J. K. Benedict. *XenServer Administration Handbook: Practical Recipes for Successful Deployments*. Sebastopol, CA: O'Reilly Media, 2016.
- Senthil, Nathan. *VirtualBox at Warp Speed: Virtualization with VirtualBox*. Seattle, WA: Amazon Digital Services, 2015.
- Troy, Ryan, and Matthew Helmke. *VMware Cookbook: A Real-World Guide to Effective VMware Use, 2nd Edition*. Sebastopol, CA: O'Reilly Media, 2012.

Глава 25

Контейнеры



Немногие технологии вызвали столько волнений и шумихи в последние годы, как скромный контейнер, взрыв популярности которого совпал с выпуском проекта с открытым исходным кодом Docker в 2013 г. Контейнеры представляют особый интерес для системных администраторов, поскольку они стандартизируют упаковку программного обеспечения, реализуя цель, которая долгое время считалась практически недостижимой.

Для того чтобы проиллюстрировать полезность контейнеров, рассмотрим типичное веб-приложение, разработанное на любом современном языке или с помощью каркаса. Для установки и запуска приложения необходимы следующие ингредиенты:

- код для приложения и его правильная конфигурация;
- десятки библиотек и других зависимостей, совместимых с конкретными версиями;
- интерпретатор (например, Python или Ruby) или среда выполнения (JRE) для выполнения кода, также связанные с конкретными версиями;
- локализации, такие как учетные записи пользователей, настройки среды и службы, предоставляемые операционной системой.

В типичной организации используются десятки и даже сотни таких приложений. Поддержание единообразия в каждой из этих областей при развертывании нескольких приложений является постоянной проблемой даже при использовании инструментов, обсуждаемых в главах 23 и 26. Несовместимые зависимости, требуемые отдельными приложениями, приводят к недогрузке систем, поскольку они не могут использоваться совместно. Кроме того, в организациях, где разработчики программного обеспечения и системные администраторы функционально разделены, необходима тщательная коор-

динация, потому что не всегда легко определить, кто несет ответственность за конкретные части операционной среды.

Образ контейнера упрощает дело, упаковывая приложение и его вспомогательные средства в стандартный переносимый файл. Любой хост с совместимой средой выполнения может создать контейнер, используя его образ в качестве шаблона. Десятки или сотни контейнеров могут работать одновременно без конфликтов. Как правило, размер образов составляет не более нескольких сотен мегабайтов, поэтому их можно копировать между системами. Эта легкость переноса приложений, возможно, является основной причиной популярности контейнеров.

В данной главе основное внимание уделяется платформе Docker. Имя Docker описывает механизм, сыгравший центральную роль в широком признании контейнеров, а платформа Docker — это то, с чем вы, скорее всего, столкнетесь как системный администратор. Компания Docker, Inc. предлагает несколько продуктов, связанных с контейнерами, но мы ограничиваем наше обсуждение основным контейнерным механизмом и менеджером кластеров Swarm.

Существует несколько жизнеспособных альтернативных контейнерных механизмов. Наиболее полным является механизм rkt для системы CoreOS. Он имеет более ясную модель процесса, чем платформа Docker, и более безопасную конфигурацию по умолчанию. Механизм rkt хорошо сочетается с системой оркестровки Kubernetes. Утилита **`systemd-nspawn`** из проекта **`systemd`** является вариантом облегченных контейнеров. Она имеет меньше возможностей, чем Docker или rkt, но в некоторых случаях это может быть полезным. Утилита rkt взаимодействует с **`systemd-nspawn`** при настройке пространств имен контейнеров.

25.1. Основные концепции

Быструю эволюцию контейнеров до высокой степени изящества можно объяснить скорее правильным согласованием разных механизмов, чем появлением какой-либо одной технологии. Контейнеры — это сочетание многочисленных существующих функций ядра, приемов работы с файловой системой и сетевых трюков. Контейнерный механизм — это программное обеспечение для управления, которое объединяет все это в одно целое.

По сути, контейнер представляет собой изолированную группу процессов, которые ограничены частной корневой файловой системой и пространством имен процессов. Содержащиеся процессы совместно используют ядро и другие службы хост-системы, но по умолчанию они не могут получить доступ к файлам или системным ресурсам за пределами своего контейнера. Приложения, которые запускаются внутри контейнера, не знают о своем контейнерном состоянии и не требуют модификации.

После того как вы прочитаете следующие разделы, вам должно стать ясно, что в контейнерах нет ничего загадочного. Фактически они основаны на некоторых средствах UNIX и Linux, которые существуют уже много лет. Описание различий между контейнерами и виртуальными машинами приведено в главе 24.

Поддержка ядра

В контейнерном механизме используется несколько средств ядра, которые необходимы для изоляции процессов.

- *Пространства имен* изолируют контейнерные процессы с точки зрения нескольких функций операционной системы, включая монтирование файловой системы,

управление процессами и сетевое взаимодействие. Например, пространство имен команды **mount** отображает процессы с помощью индивидуального представления иерархии файловой системы.[1] Контейнеры могут работать на разных уровнях интеграции с операционной системой хоста в зависимости от того, как эти пространства имен были настроены.

- *Контрольные группы* ограничивают использование системных ресурсов и определяют приоритетность одних процессов над другими. Контрольные группы препятствуют контейнерам, выходящим из-под контроля, использовать всю память и все доступное время центрального процессора.
- *Функциональные возможности* позволяют процессам выполнять определенные чувствительные операции с ядром и системные вызовы. Например, процесс может иметь возможность изменять права собственности на файл или устанавливать системное время.
- *Режим защищенных вычислений* ограничивает доступ к системным вызовам. Он обеспечивает более детальный контроль, чем функциональные возможности.

Разработка этих функций частично происходила в рамках проекта Linux Containers, LXC, который начался в компании Google в 2006 г. Проект LXC был основой Borg, внутренней платформы виртуализации Google. Система виртуализации LXC предоставляет исходные функции и инструменты, необходимые для создания и запуска Linux-контейнеров, но сделать это с помощью более чем 30 утилит командной строки и конфигурационных файлов довольно сложно. Первые несколько выпусков платформы Docker были по сути удобными для пользователя упаковками, которые упрощали использование системы LXC.

Теперь платформа Docker базируется на улучшенной и стандартизованной среде для запуска контейнеров **containerd**. Она также полагается на пространства имен Linux, контрольные группы и возможности изолировать контейнеры. Подробнее об этом можете прочесть на сайте `containerd.io`.

Образы

Образ контейнера похож на шаблон для контейнера. Образы используют точку монтирования объединенной файловой системы (union filesystem), обеспечивающую высокую производительность и переносимость. Это позволяет использовать перекрытия нескольких файловых систем для создания единой согласованной иерархии.[2] Образы контейнеров представляют собой объединенные файловые системы, которые организованы по образу и подобию корневой файловой системы типичного дистрибутива Linux.

Структура каталога и расположение бинарных файлов, библиотек и поддерживающих файлов соответствуют стандартным спецификациям иерархии файловой системы Linux. Для использования в качестве основы для образов контейнеров были разработаны специализированные дистрибутивы Linux.

Для создания контейнера на платформа Docker используется объединенная файловая система UnionFS, предназначенная только для чтения и записанная в образе, к которой добавляется уровень чтения-записи, обновляемый в контейнере. Когда контейнерные процессы изменяют файловую систему, их изменения прозрачно сохраняются на уровне

[1] Это аналогично системному вызову **chroot**, который необратимо устанавливает видимый процессу корневой каталог и тем самым отключает доступ к файлам и каталогам хоста, расположенным выше уровня **chroot**.

[2] См. статью "A brief history of union mounts" на сайте `LWN.net`, а также, например, `lwn.net/Articles/396020`.

чтения-записи. База остается неизменной. Это называется *стратегией копирования при записи*.

Во многих контейнерах могут использоваться одни и те же неизменные базовые уровни, что повышает эффективность хранения и сокращает время запуска. Эта схема изображена на рис. 25.1.

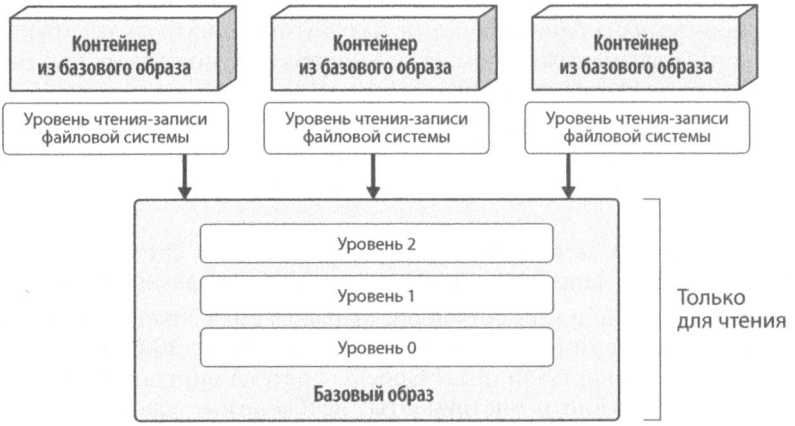


Рис. 25.1. Платформа Docker и объединенная файловая система

Сеть

По умолчанию для подключения контейнеров к сети используется сетевое пространство имен и мост внутри хоста. В этой конфигурации контейнеры имеют частные IP-адреса, недоступные за пределами хоста. При этом хост играет роль упрощенного IP-маршрутизатора и проксирует трафик между внешним миром и контейнерами. Эта архитектура позволяет администраторам контролировать, какие порты контейнера открыты для внешнего мира.

Также можно отказаться от схемы адресации частного контейнера и представить все контейнеры непосредственно в сети. Это называется *сетевым режимом хоста* и означает, что контейнер имеет неограниченный доступ к сетевому стеку хоста. В некоторых ситуациях это может быть полезным, но также представляет угрозу безопасности, поскольку контейнер не полностью изолирован.

Для получения дополнительной информации обратитесь к разделу "Docker".

25.2. Докер: механизм с открытым исходным кодом

Основной продукт компании Docker, Inc. — клиент-серверное приложение, которое создает контейнеры и управляет ими. Механизм контейнера Docker, написанный на языке Go, является в высокой степени модульным. Подключаемым хранилищем, сетевыми интерфейсами и другими функциями управляют отдельные проекты.

С компанией Docker, Inc. связаны определенные проблемы. Ее инструменты, как правило, разрабатываются настолько быстро, что новые версии иногда несовместимы с существующими развертываниями. Некоторые организации опасаются, что использование платформы Docker приведет к привязке к одному производителю. И, как и в случае с любой новой технологией, контейнеры порождают сложность и требуют подробного изучения.

Для того чтобы противостоять своим недоброжелателям, компания Docker, Inc. стала одним из основателей консорциума Open Container Initiative, целью которого является руководство развитием контейнерных технологий в здоровом конкурентном направлении, которое способствует стандартам и сотрудничеству. Вы можете об этом узнать больше на сайте `opencontainers.org`. Для того чтобы облегчить развитие сообщества среды выполнения Docker, в 2017 г. компания Docker основала проект Moby и внедрила в него основной репозиторий Docker Git (подробнее см. на сайте `mobyproject.org`).

Наше обсуждение платформы Docker основано на версии 1.13.1. Она поддерживает исключительно быстрые темпы развития, а текущие функции постоянно изменяются. Мы сосредоточимся на основах, но обязательно дополните наше описание справочным материалом с сайта `docs.docker.com`. Вы также можете поработать с песочницей на сайте `play-with-moby.com` и лабораторной средой Docker на сайте `labs.play-with-docker.com`.

Базовая архитектура

Приложение **`docker`** — это запускаемая команда, которая обрабатывает все задачи управления для системы Docker. Процесс **`dockerd`** — это резидентный процесс демона, который реализует операции с контейнерами и образами. Команда **`docker`** может запускаться в той же системе, что и **`dockerd`**, и связываться с ним через сокеты домена UNIX или взаимодействовать с **`dockerd`** с удаленного хоста с помощью протокола TCP. Эта архитектура изображена на рис. 25.2.

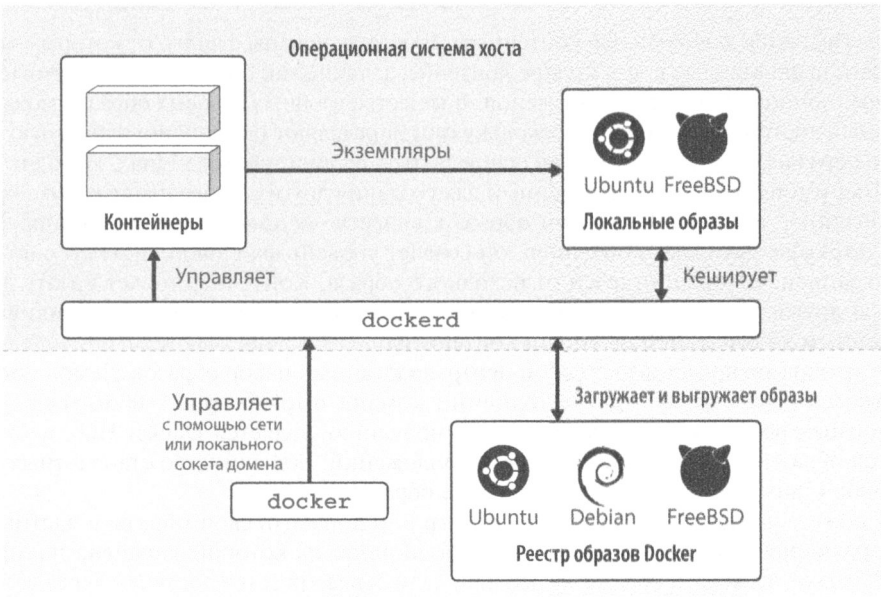


Рис. 25.2. Архитектура платформы Docker

Демон **`dockerd`** владеет всей инфраструктурой, необходимой для запуска контейнеров. Он создает виртуальную сеть и поддерживает каталог данных, в котором хранятся контейнеры и образы (по умолчанию в каталоге **`/var/lib/docker`**). Он отвечает также за создание контейнеров, вызывая соответствующие системные вызовы, настраивая

объединенные файловые системы и исполняющие процессы. Короче говоря, это программное обеспечение для управления контейнерами.

Администраторы взаимодействуют с демоном **dockerd** через командную строку, запуская подкоманды команды **docker**. Например, вы можете создать контейнер с помощью команды **docker run** или просмотреть информацию о сервере с помощью команды **docker info**. Некоторые часто используемые подкоманды приведены в табл. 25.1.

Таблица 25.1. Часто используемые подкоманды команды docker

Подкоманда	Предназначение
`docker info`	Отображает сводную информацию о демоне
`docker ps`	Отображает запущенные контейнеры
`docker version`	Отображает обширную информацию о версии сервера и клиента
`docker rm`	Удаляет контейнер
`docker rmi`	Удаляет образ
`docker images`	Отображает локальные образы
`docker inspect`	Отображает конфигурацию контейнера (данные в формате JSON)
`docker logs`	Отображает стандартный вывод из контейнера
`docker exec`	Выполняет команду в существующем контейнере
`docker run`	Запускает новый контейнер
`docker pull/push`	Загружает образы из удаленного реестра или загружает образы в удаленный реестр
`docker start/stop`	Запускает или останавливает существующий контейнер
`docker top`	Отображает состояние контейнерного процесса

Образ является *шаблоном* для контейнера. В него включены файлы, от которых зависят процессы, выполняемые в экземпляре контейнера, такие как библиотеки, двоичные файлы операционной системы и приложения. В качестве удобных базовых образов можно использовать дистрибутивы Linux, поскольку они определяют полноценную рабочую среду. Однако образ не обязательно должен основываться на дистрибутиве Linux. Исходным может быть пустой образ, предназначенный для создания других, более практичных образов.

Контейнер использует шаблон образа в качестве основы для выполнения. Когда демон **dockerd** запускает контейнер, он создает уровень файловой системы с возможностью записи, который отделен от исходного образа. Контейнер может читать любые файлы и другие метаданные, хранящиеся в образе, но любые записи ограничиваются собственным уровнем чтения-записи контейнера.

Реестр образов представляет собой централизованный набор образов. Демон **dockerd** связывается с реестрами при выполнении команд **docker pull** и **docker push**. Стандартным реестром, используемым по умолчанию, является Docker Hub, в котором хранятся образы для многих популярных приложений. Большинство стандартных дистрибутивов Linux также опубликованы в виде образов Docker.

Вы можете запустить собственный реестр или добавить свои образы в частные реестры, размещенные на Docker Hub. Любая система, на которой запущена платформа Docker, может извлекать образы из реестра, если сервер реестра доступен через сеть.

Инсталляция

Платформа Docker работает в операционных системах Linux, MacOS, Windows и FreeBSD, но Linux является флагманской системой. Поддержка FreeBSD считается экспериментальной. Посетите сайт `docker.com`, чтобы выбрать способ инсталляции, который наилучшим образом соответствует вашей среде.

Пользователи, принадлежащие группе `docker`, могут управлять демоном Docker через свой сокет, что фактически дает этим пользователям права пользователя `root`. Это значительный риск для безопасности, поэтому мы предлагаем использовать программу **sudo** для контроля доступа к командам **docker**, а не добавлять пользователей в группу `docker`. В приведенных ниже примерах мы запускаем команду **docker** в качестве пользователя `root`.

В процессе инсталляции демон Docker может и не запуститься сразу. Если он не стартовал, запустите его с помощью обычной системы **init**. Например, в операционной системе CentOS выполните команду **sudo systemctl start docker**.

Настройка клиента

Если вы подключаетесь к локальному демону **dockerd** и принадлежите к группе `docker` или имеете права **sudo**, то настройка клиента не требуется. Клиент **docker** по умолчанию подключается к демону **dockerd** через локальный сокет. Вы можете изменить стандартное поведение клиента через переменные среды.

Для того чтобы подключиться к удаленному демону **dockerd**, установите переменную окружения `DOCKER_HOST`. Обычный HTTP-порт для демона — 2375, а для версии с поддержкой TLS — 2376.

Например:

```
$ export DOCKER_HOST=tcp://10.0.0.10:2376
```

Всегда используйте TLS для связи с удаленными демонами. Если вы используете открытый протокол HTTP, с тем же успехом вы можете свободно раздавать права пользователя `root` всем, кто находится в вашей сети. Дополнительные сведения о конфигурации Docker TLS можно найти в разделе 25.3.

Мы также предлагаем включить функцию Docker Content Trust:

```
$ export DOCKER_CONTENT_TRUST=1
```

Эта функция проверяет целостность и автора образов Docker. Включение этой функции не позволяет клиентам извлекать образы, которым они не доверяют.

Если вы запускаете команду **docker** с помощью программы **sudo**, то вы можете запретить **sudo** очищать переменные среды с помощью флага **-E**. Вы также можете указать список сохраняемых переменных среды, установив значение переменной **env_keep** в файле **/etc/sudoers**. Например,

```
Defaults env_keep + = "DOCKER_CONTENT_TRUST"
```

Методики работы с контейнерами

Для того чтобы создать контейнер, необходим образ для использования в качестве шаблона. В образе есть все бинарные файлы файловой системы, необходимые для запуска программ. В новой инсталляции Docker образов нет. Чтобы загрузить образы из хранилища Docker Hub, используйте команду **docker pull**.[3]

```
# docker pull debian
Using default tag: latest
latest: Pulling from library/debian
f50f9524513f: Download complete
d8bd0657b25f: Download complete
Digest: sha256:e7d38b3517548a1c71e41bffe9c8ae6d6...
Status: Downloaded newer image for debian:latest
```

[3] Со списком доступных образов можно ознакомиться на странице `hub.docker.com`.

Шестнадцатеричные строки относятся к уровням объединенной файловой системы. Если один и тот же уровень используется более чем в одном образе, платформе Docker нужна только одна его копия. Мы не запрашивали определенный тег или версию образа Debian, поэтому Docker по умолчанию загрузил тег новейшего образа.

Список локальных загруженных образов можно вывести с помощью команды **docker image**.

```
# docker images
REPOSITORY    TAG         IMAGE ID        CREATED          SIZE
ubuntu        latest      07c86167cdc4    2 weeks ago      187.9 MB
ubuntu        wily        b5e09e0cd052    5 days ago       136.1 MB
ubuntu        trusty      97434d46f197    5 days ago       187.9 MB
ubuntu        15.04       d1b55fd07600    8 weeks ago      131.3 MB
centos        7           d0e7f81ca65c    2 weeks ago      196.6 MB
centos        latest      d0e7f81ca65c    2 weeks ago      196.6 MB
debian        jessie      f50f9524513f    3 weeks ago      125.1 MB
debian        latest      f50f9524513f    3 weeks ago      125.1 MB
```

На этом компьютере имеются образы для нескольких дистрибутивов Linux, включая только что загруженный образ Debian. Одному и тому же образу можно назначить несколько тегов. Обратите внимание на то, что `debian:jessie` и `debian:latest` имеют один и тот же идентификатор образа; это два разных имени для одного и того же образа.

Имея образ, очень легко запустить базовый контейнер:

```
# docker run debian /bin/echo "Hello World"
Hello World
```

Что сейчас произошло? Система Docker создала контейнер из базового образа Debian и выполнила команду **/bin/echo "Hello World"** внутри него.[4] Контейнер перестает работать, когда команда завершает работу: в данном случае сразу после завершения команды **echo**. Если образ Debian ранее не существовал локально, то демон попытается автоматически загрузить его перед запуском команды. Мы не указали тег, поэтому по умолчанию был использован новейший образ.

Для запуска интерактивной оболочки команде **docker run** нужно передать флаги **-i** и **-t**. Следующая команда запускает оболочку **bash** внутри контейнера и подключает к ней каналы ввода-вывода внешней оболочки. Мы также назначаем контейнеру имя хоста, что полезно для его идентификации в журналах. (В противном случае мы увидим в сообщениях журнала случайный идентификатор контейнера).

```
ben@host$ sudo docker run --hostname debian -it debian /bin/bash
root@debian:/# ls
bin   dev   home   lib64   mnt   proc   run   srv   tmp   var
boot  etc   lib    media   opt   root   sbin  sys   usr
root@debian:/# ps aux
USER   PID   %CPU   %MEM    VSZ    RSS   TTY   STAT   START   TIME  COMMAND
root     1    0.5    0.4  20236   1884   ?       Ss   19:02   0:00  /bin/bash
root     7    0.0    0.2  17492   1144   ?       R+   19:02   0:00  ps aux
root@debian:/# uname -r
3.10.0-327.10.1.el7.x86_64
root@debian:/# exit
exit
ben@host$ uname -r
3.10.0-327.10.1.el7.x86_64
```

[4] Это команда **echo** из проекта GNU. Не путайте ее с командой **echo**, встроенной в большинство оболочек UNIX, хотя обе команды делают одно и то же.

Все это удивительно похоже на организацию доступа к виртуальной машине. Существует полноценная корневая файловая система, но дерево процессов выглядит почти пустым. Команда **/bin/bash** имеет PID равный 1, потому что эту команду платформа Docker запустила в контейнере первой.

Результат команды **uname -r** одинаков как внутри, так и снаружи контейнера. Это условие выполняется всегда; мы указываем на это как напоминание о том, что ядро является общим.

Процессы в контейнерах не могут видеть другие процессы, запущенные в системе, из-за изолированного пространства PID. Тем не менее процессы на хосте могут видеть контейнерные процессы. Идентификатор PID процесса, видимый из контейнера, отличается от PID, который виден из хоста.

Для реальной работы необходимы долговечные контейнеры, которые работают в фоновом режиме и принимают подключения по сети. Следующая команда запускает в фоновом режиме (**-d**) контейнер с именем `nginx`, который создается из официального образа NGINX. Мы прокладываем туннель с порта 80 на хосте в тот же порт в контейнере:

```
# docker run -p 80:80 --hostname nginx --name nginx -d nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
fdd5d7827f33: Already exists
a3ed95caeb02: Pull complete
e04488adab39: Pull complete
2af76486f8b8: Pull complete
Digest: sha256:a234ab64f6893b9a13811f2c81b46cfac885cb141dcf4e275ed3
    ca18492ab4e4
Status: Downloaded newer image for nginx:latest
0cc36b0e61b5a8211432acf198c39f7b1df864a8132a2e696df55ed927d42c1d
```

У нас не было локального образа `nginx`, поэтому платформе Docker пришлось загрузить его из реестра. Как только образ был загружен, платформа Docker запустила контейнер и вывела его идентификатор — уникальную шестнадцатеричную строку, состоящую из 65 символов.

Команда **docker ps** отображает краткий отчет о запущенных контейнерах:

```
# docker ps
IMAGE   COMMAND                  STATUS          PORTS
nginx   "nginx -g 'daemon off"   Up 2 minutes    0.0.0.0:80->80/tcp
```

Мы не указали демону **docker**, что именно нужно запускать в контейнере, поэтому он по умолчанию использовал команду, которая была указана при создании образа. Вывод показывает, что этой командой является команда **nginx -g 'daemon off'**, которая запускает **nginx** как процесс переднего плана, а не как фоновый демон. В контейнере нет демона **init** для управления процессами, и если запустить сервер **nginx** как демон, контейнер будет запущен, но сразу же завершен, когда процесс **nginx** будет разветвлен и перейдет в фоновый режим.

Большинство демонов сервера поддерживают специальный флаг командной строки, который заставляет их запускаться на переднем плане. Если ваше программное обеспечение не может работать на переднем плане или если вам нужно запустить несколько процессов в контейнере, вы можете назначить систему управления процессами, такую как **supervisord**, в качестве упрощенной команды **init** для контейнера.

Если сервер NGINX работает в контейнере, а порт 80 хоста был отображен на порт 80 контейнера, то мы можем выполнить HTTP-запрос к контейнеру с помощью про-

граммы **curl**. По умолчанию сервер NGINX возвратит стандартную целевую страницу в формате HTML.

```
host$ curl localhost
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
...
```

Мы можем использовать команду **docker logs** для просмотра канала **STDOUT** контейнера, в который в нашем случае выводится журнал доступа к серверу NGINX. Единственным трафиком является наш запрос к программе **curl**:

```
# docker logs nginx
172.17.0.1 - - [24/Feb/2017:19:12:24 +0000] "GET / HTTP/1.1" 200 612
    "-" "curl/7.29.0" "-"
```

Мы также можем использовать команду **docker logs -f**, чтобы получить поток вывода контейнера в реальном времени, точно так же, как команда **tail -f** позволяет увидеть вывод из постоянно увеличивающегося файла журнала.

Команда **docker exec** создает новый процесс в существующем контейнере. Например, чтобы устранять неполадки, мы могли бы запустить интерактивную оболочку в контейнере:

```
# docker exec -ti nginx bash
root@nginx:/# apt-get update && apt-get -y install procps
root@nginx:/# ps ax
PID   TTY   STAT    TIME    COMMAND
1     ?     Ss      0:00    nginx: master process nginx -g daemon off;
7     ?     S       0:00    nginx: worker process
8     ?     Ss      0:00    bash
21    ?     R+      0:00    ps ax
```

Образы контейнеров настолько малы, насколько это возможно, и часто в них отсутствуют общие утилиты управления. В приведенной выше последовательности команд мы сначала обновили общий индексный файл пакетной системы, а затем инсталлировали из нее программу **ps**, которая является частью пакета **procps**.

В списке процессов показан главный демон **nginx**, рабочий демон **nginx** и оболочка **bash**. Когда мы выходим из оболочки, созданной с помощью команды **docker exec**, контейнер продолжает работать. Если процесс с идентификатором PID 1 завершит свою работу, когда оболочка еще активна, работа контейнера будет прекращена, и наша оболочка также завершит работу.

Мы можем остановить и запустить контейнер.

```
# docker stop nginx
nginx
# docker ps
IMAGE COMMAND     STATUS      PORTS
# docker start nginx
# docker ps
IMAGE COMMAND     STATUS      PORTS
nginx "nginx -g 'daemon off"  Up 2 minutes 0.0.0.0:80->80/tcp
```

Команда **docker start** запускает контейнер с теми же аргументами, которые были переданы при создании контейнера с помощью команды **docker run**.

Когда контейнеры прекращают работу, они остаются в системе в состоянии покоя. Вы можете увидеть список всех контейнеров, в том числе остановленных, с помощью

команды **docker ps -a**. Хранение ненужных старых контейнеров не представляет особой опасности, но считается плохим стилем и может привести к конфликтам при повторном использовании имен контейнеров.

Закончив работу с контейнером, мы можем остановить и удалить его:

```
# docker stop nginx && docker rm nginx
```

Команда **docker run --rm** запускает контейнер и автоматически удаляет его при выходе, но это работает только для контейнеров, в которых не запущены демоны с помощью флага **-d**.

Тома

Уровни файловой системы для большинства контейнеров состоят из статического кода приложения, библиотек и других системных или вспомогательных файлов. Уровень файловой системы чтения-записи позволяет контейнерам производить локальные модификации. Однако большая зависимость от объединенной файловой системы не является лучшим решением для приложений с интенсивным использованием данных, таких как базы данных. Для таких приложений в платформе Docker используется понятие тома.

Том (volume) представляет собой независимый, доступный для записи каталог в контейнере, который поддерживается отдельно от объединенной файловой системы. Если контейнер удален, то данные в томе сохраняются и могут быть доступны из хоста. Тома также могут быть распределены между несколькими контейнерами.

Мы добавляем том в контейнер с помощью аргумента **-v** команды **docker**:

```
# docker run -v /data --rm --hostname web --name web -d nginx
```

Если каталог **/data** уже существует в контейнере, все найденные там файлы копируются в том. Найти том на хосте можно с помощью команды **docker inspect**.

```
# docker inspect -f '{{ json .Mounts }}' web
...
"Mounts": [
    {
        "Name": "8f026ebb9c0cda27441fb7fd275c8e767685f260...f5fd1939823558",
        "Source": "/var/lib/docker/volumes/8f026ebb9c0cda...93823558/_data",
        "Destination": "/data",
        "Driver": "local",
        "Mode": "",
        "RW": true,
        "Propagation": ""
    }
]
```

Подкоманда **inspect** возвращает подробную информацию; мы применили фильтр, чтобы на экран выводились только смонтированные тома. Если контейнер прекратил работу или его нужно удалить, мы можем найти том данных в исходном каталоге на хосте. Имя больше похоже на идентификатор, но оно пригодится, если нам нужно будет идентифицировать том позже.

Для вывода списка доступных томов в системе на самом высоком уровне следует выполнить команду **docker volume ls**.

Платформа Docker также поддерживает механизм монтирования привязок, который монтирует тома на хосте и в контейнерах одновременно. Например, мы можем смонтировать привязку **/mnt/data** хоста на каталог **/data** в контейнере с помощью следующей команды:

```
# docker run -v /mnt/data:/data --rm --name web -d nginx
```

Когда контейнер записывает данные в каталог **/data**, изменения также отображаются в каталоге **/mnt/data** хоста.

Для томов, смонтированных описанным выше образом, система Docker не копирует существующие файлы из смонтированного каталога контейнера в том. Как и в случае с традиционным монтированием файловой системы, содержимое тома заменяет исходное содержимое смонтированного каталога в контейнере.

При запуске контейнеров в облаке мы предлагаем комбинировать монтирование привязок с хранением блоков, предлагаемым облачными провайдерами. Например, тома Elastic Block Storage службы AWS являются отличными резервными хранилищами для монтирования привязки Docker. У них есть встроенные средства для создания мгновенных снимков и они могут перемещаться между экземплярами EC2. Они также могут быть скопированы между хостами, что позволяет другим системам проще получать данные контейнера. Вы можете использовать собственные средства создания снимков EBS для создания простой системы резервного копирования.

Контейнеры данных

Одним из полезных проектных решений, которое возникло из реального опыта, является контейнер, предназначенный только для данных. Его цель — сохранить конфигурацию тома для других контейнеров, чтобы эти контейнеры можно было легко перезапустить и заменить.

Создайте контейнер данных, используя либо обычный том, либо том, смонтированный с привязкой на хосте. Контейнер данных при этом никогда не запускается.

```
# docker create -v /mnt/data:/data --name nginx-data nginx
```

Теперь вы можете использовать том контейнера данных в контейнере, где запускается сервер **nginx**:

```
# docker run --volumes-from nginx-data -p 80:80 --name web -d nginx
```

Контейнер **web** имеет доступ на чтение и запись к тому **/data**, который на самом деле является контейнером данных с именем **nginx-data**. Контейнер **web** может быть перезапущен, удален или заменен, но пока при его запуске указан параметр **--volumes-from**, файлы в каталоге **/data** будут оставаться неизменными.

По правде говоря, идея объединения сохраняемых данных с контейнерами немного не соответствует принятым стандартам. Контейнеры должны создаваться и удаляться незамедлительно в ответ на внешние события. Идеальным решением является наличие парка идентичных серверов, на которых запущен демон **dockerd**, причем контейнеры могут быть размещены на любом из серверов. Однако, когда вы добавляете сохраняемые тома данных, контейнер становится привязанным к определенному серверу. Как бы нам ни хотелось жить в идеальном мире, многим приложениям периодически нужны сохраняемые данные.

Сети Docker

Как обсуждалось выше в разделе "Сеть", существует несколько способов подключения контейнеров к сети. Во время установки система Docker создает три стандартных сетевых опции. Вывести их можно с помощью команды **docker network ls**.

```
# docker network ls
NETWORK ID      NAME      DRIVER
6514e7108508    bridge    bridge
```

```
1a72c1e4b230    none      null
e0f4e608c92c    host      host
```

В стандартном режиме моста (опция `bridge`) контейнеры находятся в частной сети с пространством имен, расположенным в пределах хоста. Мост соединяет сеть хоста с пространством имен контейнеров. Когда вы создаете контейнер и назначаете ему порт хоста с помощью команды **`docker run -p`**, система Docker создает правила **`iptables`**, которые маршрутизируют трафик от общего интерфейса хоста к интерфейсу контейнера в сети моста.

🕮 Дополнительную информацию о команде **`iptables`** см. в разделе 13.14.

При использовании опции `host` не используется отдельное пространство сетевых имен. Вместо этого в контейнере используется общий с хостом сетевой стек, включая все его интерфейсы. Порты, открытые в контейнере, также отображаются на интерфейсах хостах. Некоторые программы работают лучше в режиме `host`, но эта конфигурация также может привести к конфликтам портов и другим проблемам.

Сетевая опция `none` указывает, что система Docker не должна предпринимать никаких шагов для настройки сети. Она предназначена для сложных сценариев использования, которые имеют специальные требования к сети.

Для выбора опции сети контейнера передайте аргумент **`--net`** команде **`docker`** .

Пространства имен и мостовая сеть

Мост — это функция ядра Linux, соединяющая два сегмента сети. Во время инсталляции система Docker автоматически создает на хосте мост, называемый `docker0`. Система Docker выбирает пространство имен IP-адресов для дальней стороны моста, которое по ее расчетам вряд ли войдет в конфликт с любыми другими сетями, доступными из хоста. Каждому контейнеру предоставляется виртуальный сетевой интерфейс с пространством имен, который имеет IP-адрес в пределах мостового сетевого диапазона.

Алгоритм выбора адресов практичен, но не идеален. В вашей сети могут быть маршруты, которые не видны с хоста. Если произойдет коллизия, хост больше не сможет получить доступ к удаленной сети с перекрывающимся адресным пространством, но сможет получить доступ к локальным контейнерам. Если вы оказались в такой ситуации или вам нужно изменить адресное пространство моста по какой-либо другой причине, используйте аргумент **`--fixed-cidr`** демона **`dockerd`**.

Пространства имен в сети зависят от виртуальных интерфейсов, странных конструкций, создаваемых парами, где одна сторона находится в пространстве имен хоста, а другая — в контейнере. Таким образом, потоки данных, входящие на одном конце пары и выходящие на другом, соединяют контейнер с хостом. В большинстве случаев контейнер имеет только одну такую пару. Эту концепцию иллюстрирует рис. 25.3.

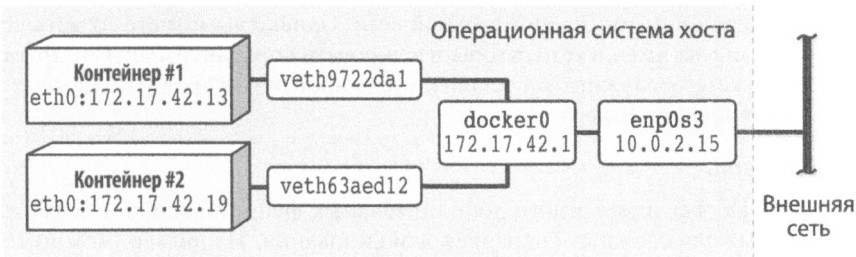


Рис. 25.3. Мостовая сеть платформы Docker

Одна половина каждой пары видна из сетевого стека хоста. Например, ниже представлены видимые интерфейсы на хосте CentOS с одним запущенным контейнером.

```
centos$ ip addr show
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
     state UP qlen 1000
     link/ether 08:00:27:c3:36:f0 brd ff:ff:ff:ff:ff:ff
     inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic enp0s3
         valid_lft 71368sec preferred_lft 71368sec
     inet6 fe80::a00:27ff:fec3:36f0/64 scope link
         valid_lft forever preferred_lft forever

3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
     state UP
     link/ether 02:42:d4:30:59:24 brd ff:ff:ff:ff:ff:ff
     inet 172.17.42.1/16 scope global docker0
         valid_lft forever preferred_lft forever
     inet6 fe80::42:d4ff:fe30:5924/64 scope link
         valid_lft forever preferred_lft forever
53: veth584a021@if52: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
     noqueue master docker0 state UP
     link/ether d6:39:a7:bd:bf:eb brd ff:ff:ff:ff:ff:ff link-netnsid 0
     inet6 fe80::d439:a7ff:febd:bfeb/64 scope link
         valid_lft forever preferred_lft forever
```

В этом листинге показан основной интерфейс хоста `enp0s3`, а также виртуальный мост Ethernet с интерфейсом `docker0`, в котором используется сеть 172.17.42.0/16. Интерфейс `veth` — это сторона хоста в виртуальной паре интерфейсов, соединяющей контейнер с мостовой сетью.

Сторона контейнера мостовой пары не видна с хоста без отображения низкоуровневых деталей сетевого стека. Эта невидимость является лишь побочным эффектом работы сетевых пространств имен. Однако мы можем обнаружить интерфейс, отобразив информацию самом контейнере.

```
# docker inspect -f '{{ json .NetworkSettings.Networks.bridge }}' nginx
   "bridge": {
      "Gateway": "172.17.42.1",
      "IPAddress": "172.17.42.13",
      "IPPrefixLen": 16,
      "MacAddress": "02:42:ac:11:00:03"
   }
```

IP-адрес контейнера — 172.17.42.13, а его стандартный шлюз — интерфейс моста `docker0`. (Эта мостовая пара изображена на рис. 25.3.) В конфигурации моста, принятой по умолчанию, все контейнеры могут связываться друг с другом, потому что все они находятся в одной и той же виртуальной сети. Однако вы можете создать дополнительные пространства имен в сети, чтобы изолировать контейнеры друг от друга. Таким образом, вы можете обслуживать несколько изолированных сред из одного и того же набора экземпляров контейнера.

Оверлейные сети

Платформа Docker имеет много дополнительных функциональных возможностей, предназначенных для сложных сценариев использования. Например, можно создавать пользовательские частные сети, которые автоматически связывают контейнеры. С помощью программного обеспечения для оверлейных сетей контейнеры, работающие

на отдельных хостах, могут маршрутизировать трафик друг к другу через частное сетевое адресное пространство. Технология Virtual eXtensible LAN (VXLAN), описанная в спецификации RFC7348, представляет собой одну систему, которая может быть объединена с контейнерами для реализации расширенных сетевых возможностей. За дополнительной информацией обратитесь к документации по сетям Docker.

Драйверы хранилищ

Системы UNIX и Linux предлагают несколько способов реализации объединенной файловой системы. Платформа Docker не зависит от этих способов реализации и пропускает все операции с файловой системой через выбранный вами драйвер хранилища.

Драйвер хранилища настроен как часть параметров запуска демона докеров. Ваш выбор механизма хранения имеет важные последствия для производительности и стабильности, особенно в производственных средах, поддерживающих многие контейнеры. Текущий набор драйверов приведен в табл. 25.2.

Таблица 25.2. Драйверы хранилищ Docker

Драйвер	Описание и комментарии
`aufs`	Повторная реализация оригинальной файловой системы UnionFS
	Оригинальный механизм хранения Docker
	По умолчанию предусмотрен в системах Debian и Ubuntu
	В настоящее время устарел, поскольку не является частью ядра основной линейки семейства Linux
`btrfs`	Использует файловую систему Btrfs с копированием при записи (см. раздел 20.13)
	Стабилен и включен в основное ядро Linux
	Использование на платформе Docker предусмотрено лишь в некоторых дистрибутивах и носит экспериментальный характер
`devicemapper`	По умолчанию предусмотрен в системе RHEL/CentOS 6
	Настоятельно рекомендуется режим прямого доступа LVM, но требуется настройка
	Имеет историю ошибок
	Требует изучения документации `devicemapper` для платформы Docker
`overlay`	Основан на файловой системе OverlayFS
	Считается заменой файловой системы AuFS
	Предусмотрен по умолчанию в системе CentOS 7, если загружен модуль ядра **`overlay`**
`vfs`	Не настоящая объединенная файловая система
	Медленный, но стабильный, подходит для некоторых производственных сред
	Хорошо подходит для доказательства концепции или в качестве тестового стенда
`zfs`	Использует файловую систему с поддержкой копирования при записи ZFS (см. раздел 20.12)
	По умолчанию предусмотрен для системы FreeBSD
	Считается экспериментальным в системе Linux

Драйвер VFS по существу делает невозможным использование объединенной файловой системы. Система Docker создает полную копию образа для каждого контейнера, что приводит к увеличению использования дискового пространства и времени запуска контейнера. Однако эта реализация проста и надежна. Если ваш сценарий использова-

ния включает долгоживущие контейнеры, виртуальная файловая система VFS является надежным выбором. Однако мы никогда не видели организаций, в которых бы использовались такие файловые системы в производственных приложениях.

Btrfs и ZFS также не являются истинными объединенными файловыми системами. Тем не менее они эффективно и надежно реализуют оверлеи, потому что изначально поддерживают клонирование файловой системы через копирование при записи. Поддержка платформы Docker для файловых систем Btrfs и ZFS в настоящее время ограничена несколькими конкретными дистрибутивами Linux (а также FreeBSD для ZFS), но это перспективные варианты. Чем меньше атрибутов файловой системы, характерных для контейнерной системы, тем лучше.

Выбор драйвера хранилища — это тонкая тема. Если вы или кто-то из вашей команды не знает одну из этих файловых систем в совершенстве, мы рекомендуем придерживаться файловой системы, которая предусмотрена в вашем дистрибутиве по умолчанию. Дополнительную информацию см. в документации о драйверах хранения на платформе Docker.

Изменение параметров настройки демона dockerd

Вам неизбежно придется изменить некоторые настройки демона **dockerd**. Параметры настройки включают механизм хранения, параметры DNS и базовый каталог, в котором хранятся образы и метаданные. Для того чтобы просмотреть полный список аргументов, запустите команду **dockerd -h**.

Вы можете увидеть текущую конфигурацию демона с помощью команды **docker info**.

```
centos# docker info
Containers: 6
   Running: 0
   Paused: 0
   Stopped: 6
Images: 9
Server Version: 1.10.3
Storage Driver: overlay
Backing Filesystem: xfs
Logging Driver: json-file
Plugins:
   Volume: local
   Network: bridge null host
Kernel Version: 3.10.0-327.10.1.el7.x86_64
Operating System: CentOS Linux 7 (Core)
OSType: linux
Architecture: x86_64
```

Это хороший способ проверки того, что все сделанные вами настройки вступили в силу.

🕮 Дополнительную информацию об изменении модульных файлов с помощью утилиты **systemctl** см. в главе 2.

Управление демонами, включая изменение параметров запуска на платформе Docker, аналогично управлению в системе **init**. Например, в дистрибутиве, использующем демон **systemd**, следующая команда редактирует модуль службы Docker для установки драйвера хранилища, не предусмотренного по умолчанию, набора DNS-серверов и специального адресного пространства для мостовой сети:

```
$ systemctl edit docker
[Service]
ExecStart=
ExecStart=/usr/bin/docker daemon -D --storage-driver overlay \
   --dns 8.8.8.8 --dns 8.8.4.4 --bip 172.18.0.0/19
```

Второе выражение `ExecStart=` не является ошибкой. Это выражение, характерное для утилиты **systemd**, сбрасывает настройки, принятые по умолчанию, гарантируя, что новое определение используется точно так, как показано. После изменения параметров нужно перезагрузить демон с помощью утилиты **systemctl** и проверить изменения.

```
centos$ sudo systemctl restart docker
centos$ sudo systemctl status docker
   docker.service
      Loaded: loaded (/etc/systemd/system/docker.service; static;
         vendor preset: disabled)
   Drop-In: /etc/systemd/system/docker.service.d
            |---- override.conf
   Active: active (running) since Wed 2016-03-09 23:14:56 UTC; 12s ago
Main PID: 4328 (docker)
   CGroup: /system.slice/docker.service
            |---- 4328 /usr/bin/docker daemon -D --storage-driver overlay
     --dns 8.8.8.8 --dns 8.8.4.4 --bip 172.18.0.0/19
...
```

В операционных системах, использующих систему инициализации **upstart**, параметры демона необходимо изменять в файле **/etc/default/docker**. Для более старых систем с инициализацией в стиле SysV используйте файл **/etc/sysconfig/docker**.

По умолчанию демон **dockerd** прослушивает сокет домена UNIX **/var/run/docker.sock**, который используется для подключения утилиты **docker**. Чтобы демон прослушивал защищенный сокет на основе TLS, используйте параметр **-H tcp://0.0.0.0:2376**. Более подробную информацию по настройке TLS см. в разделе 25.3.

Сборка образа

Вы можете заключить собственные приложения в контейнеры, создавая образы, содержащие код приложения. Процесс сборки начинается с базового образа. Добавьте свое приложение, внося любые изменения в качестве новых уровней и сохраняя образ в локальной базе образов. Затем вы можете создавать контейнеры из этого образа. Вы также можете занести свой образ в реестр, чтобы сделать его доступным для других систем, работающих с платформой Docker.

Каждый слой образа идентифицируется криптографическим хешем его содержимого. Хеш служит системой проверки, которая позволяет платформе Docker подтвердить, что никакое повреждение или злонамеренное вмешательство не изменило содержание образа.

Выбор базового образа

Перед созданием пользовательского образа выберите подходящую базу. Эмпирическое правило для базовых образов таково: чем меньше размер, тем лучше. База должна содержать только то, что необходимо для запуска вашего программного обеспечения, и ничего больше.

Многие официальные образы основаны на дистрибутиве под названием Alpine Linux, который занимает менее 5 Мбайт, но он может иметь библиотечную несовме-

стимость с некоторыми приложениями. Образ Ubuntu занимает больше 188 Мбайт, но по сравнению с типичной установкой сервера его можно считать довольно компактным. Возможно, вы сможете найти базовый образ, в котором уже настроены компоненты среды выполнения вашего приложения. Стандартные базовые образы существуют для наиболее распространенных языков, сред выполнения и платформ приложений.

Тщательно исследуйте свой базовый образ, прежде чем принимать окончательное решение. Изучите файл **Dockerfile** вашего базового образа (см. следующий раздел) и любые неочевидные зависимости, чтобы избежать сюрпризов. Базовые образы могут иметь неожиданные требования или включать уязвимые версии программного обеспечения. В некоторых случаях вам может потребоваться скопировать файл **Dockerfile** базового образа и перестроить его в соответствии со своими потребностями.

Когда демон **dockerd** загружает образ, он загружает только те уровни, которых у него еще нет. Если все ваши приложения используют одну и ту же базу, для загрузки демона требуется меньше данных, и при первом запуске контейнеры запускаются быстрее.

Сборка из файла Dockerfile

Dockerfile — это рецепт для сборки образа. Он содержит ряд инструкций и команд оболочки. Команда **docker build** считывает файл **Dockerfile**, последовательно выполняет инструкции в нем и фиксирует результат как образ. Проекты по разработке программного обеспечения, использующие файл **Dockerfile**, обычно хранят его в корневом каталоге хранилища Git, чтобы облегчить создание новых образов, содержащих это программное обеспечение.

Первая инструкция в файле **Dockerfile** указывает образ, который будет использоваться в качестве базы. Каждая последующая инструкция совершает переход на новый уровень, который в свою очередь используется в качестве базы для следующей команды. Каждый уровень включает только изменения предыдущего уровня. Общая файловая система объединяет уровни для создания корневой файловой системы контейнера.

Вот как выглядит файл **Dockerfile**, который создает официальный образ NGINX для системы Debian.[5]

```
FROM debian:jessie
MAINTAINER NGINX Docker Maintainers "docker-maint@nginx.com"
ENV NGINX_VERSION 1.10.3-1~jessie
RUN apt-get update \
   && apt-get install -y ca-certificates nginx=${NGINX_VERSION} \
   && rm -rf /var/lib/apt/lists/*
# forward request and error logs to docker log collector
RUN ln -sf /dev/stdout /var/log/nginx/access.log \
   && ln -sf /dev/stderr /var/log/nginx/error.log
EXPOSE 80 443
CMD ["nginx", "-g", "daemon off;"]
```

Сервер NGINX использует в качестве базы образ `debian:jessie`. После объявления контактов группы поддержки продукта (maintainer) в файле устанавливается переменная среды (`NGINX_VERSION`), которая становится доступной для всех последующих инструкций в файле **Dockerfile**, а также для любого процесса, который выполняется внутри контейнера после сборки и создания экземпляра образа. Большую часть работы по инсталляции сервера NGINX из репозитория пакетов выполняет первая инструкция `RUN`.

[5] Пример взят из проекта `github.com/nginxinc/docker-nginx` и немного упрощен.

По умолчанию сервер NGINX отправляет данные журнала в файл **/var/log/nginx/access.log**, но по соглашению, принятому для контейнеров, сообщения должны выводиться в поток STDOUT. Поэтому в последней команде RUN группа поддержки продукта создала символическую ссылку для перенаправления журнала доступа в поток STDOUT. Аналогично ошибки перенаправляются в поток контейнера .

Команда EXPOSE сообщает демону **dockerd**, какие порты прослушивает контейнер. Используемые порты могут быть переопределены во время запуска контейнера с помощью команды **docker run** с параметром **-p**.

Последняя инструкция в файле **Dockerfile** сервера NGINX — это команда, которую должен выполнить демон **dockerd** при запуске контейнера. В этом случае контейнер запускает двоичный файл **nginx** как процесс переднего плана.

Краткое описание инструкций из файла **Dockerfile** приведено в табл. 25.3. Справочное руководство на docs.docker.com является официальной документацией.

Таблица 25.3. Сокращенный список инструкций из файла Dockerfile

Инструкция	Описание
ADD	Копирует файлы с хоста сборки в образ[a]
ARG	Устанавливает переменные, которые можно использовать во время сборки, но не в конечном образе; не предназначена для закрытой информации
CMD	Устанавливает команды по умолчанию для выполнения в контейнере
COPY	Аналогична команде ADD, но только для файлов и каталогов
ENV	Устанавливает переменные среды, доступные для всех последующих инструкций сборки и контейнеров, порожденных этим образом
EXPOSE	Сообщает демону **dockerd** о сетевых портах, открытых в контейнере
FROM	Устанавливает базовый образ; эта инструкция должна быть первой
LABEL	Устанавливает теги образов, отображаемые с помощью команды **docker inspect**
RUN	Выполняет команды и сохраняет результат в образе
STOPSIGNAL	Задает сигнал, который отправляется в процесс для его остановки по команде **docker stop**; по умолчанию — SIGKILL
USER	Задает имя учетной записи для использования при запуске контейнера и выполнении любых последующих инструкций по сборке
VOLUME	Назначает том для хранения постоянных данных
WORKDIR	Задает стандартный рабочий каталог для последующих инструкций.

[a] Источником может быть файл, каталог, архив **tar** или удаленный ресурс, заданный URL.

Сборка производного образа из файла Dockerfile

Для создания производного образа сервера NGINX можно использовать очень простой файл **Dockerfile**, в котором добавлен пользовательский вариант файла **index.html**, заменяющего значение по умолчанию.

```
$ cat index.html
<!DOCTYPE html>
<title>ULSAH index.html file</title>
<p>A simple Docker image, brought to you by ULSAH.</p>
$ cat Dockerfile
FROM nginx
# Add a new index.html to the document root
ADD index.html /usr/share/nginx/html/
```

Если не принимать во внимание пользовательский вариант файла **index.html**, наш новый образ будет идентичным базовому. Вот как создается измененный образ.

```
# docker build -t nginx:ulsah .
Step 1 : FROM nginx
   ---> fd19524415dc
Step 2 : ADD index.html /usr/share/nginx/html/
   ---> c0c25eaf7415
Removing intermediate container 04cc3278fdb4
Successfully built c0c25eaf7415
```

Для создания образа с именем `nginx` и тегом `ulsah` используется команда **docker build** с параметрами **-t nginx:ulsah**, чтобы отличить его от официального образа сервера NGINX. Конечная точка сообщает команде **docker build**, где можно найти файл **Dockerfile** (в данном случае в текущем каталоге).

Теперь мы можем запустить образ и просмотреть работу нашего измененного файла **index.html**:

```
# docker run -p 80:80 --name nginx-ulsah -d nginx:ulsah
$ curl localhost
<!DOCTYPE html>
<title>ULSAH index.html file</title>
<p>A simple Docker image, brought to you by ULSAH.</p>
```

Мы можем убедиться, что наш образ указан среди локальных образов, выполнив команду **docker images**.

```
# docker images | grep ulsah
REPOSITORY      TAG       IMAGE ID          CREATED          SIZE
nginx           ulsah     c0c25eaf7415      3 minutes ago    134.6 MB
```

Чтобы удалить образы, выполните команду **docker rmi**. Вы не можете удалить образ, пока не остановите и не удалите все контейнеры, которые его используют.

```
# docker ps | grep nginx:ulsah
IMAGE          COMMAND                  STATUS          PORTS
nginx:ulsah    "nginx -g 'daemon off"   Up 37 seconds   0.0.0.0:80->80/tcp
# docker stop nginx-ulsah && docker rm nginx-ulsah
nginx-ulsah
nginx-ulsah
# docker rmi nginx:ulsah
```

Обе команды — **docker stop** и **docker rm** — повторяют название используемого контейнера, в результате чего строка "**nginx-ulsah**" выводится дважды.

Реестры

Реестр — это индекс образов платформы Docker, к которому демон **dockerd** может получить доступ через протокол HTTP. Когда запрашивается образ, не существующий на локальном диске, демон **dockerd** извлекает его из реестра. Образы загружаются в реестр с помощью команды **docker push**. Хотя операции с образами инициируются командой **docker**, только демон **dockerd** фактически взаимодействует с реестрами.

Docker Hub — это общедоступная служба реестра, поддерживаемая компанией Docker, Inc. Она содержит образы для многих дистрибутивов и проектов с открытым исходным кодом, включая все рассматриваемые нами примеры Linux-систем. Целостность этих официальных образов проверяется с помощью системы доверенного контента, гарантируя тем самым, что загружаемый образ предоставляется поставщиком, чье имя

находится на этикетке. Вы также можете публиковать собственные образы в реестре Docker Hub для других пользователей.

Любой пользователь может загружать общедоступные образы из реестра Docker Hub, но, имея подписку, вы также можете создавать частные репозитории. Получив платную учетную запись на сайте `hub.docker.com`, войдите в систему из командной строки с помощью команды **`docker login`** для доступа к персональному реестру, чтобы получить возможность загружать и выгружать собственные пользовательские образы. Вы также можете запускать сборку образов всякий раз, когда в репозитории GitHub рируется фиксация.

Служба Docker Hub — не единственный реестр, поддерживающий подписку. Существуют и другие: `quay.io`, Artifactory, Google Container Registry и Amazon EC2 Container Registry.

Docker Hub — это щедрый источник образов для крупной экосистемы. Кроме того, он постепенно становится реестром по умолчанию, если вам не нужно нечто более необычное.

Например, команда

```
# docker pull debian:jessie
```

сначала ищет локальную копию образа. Если образ не доступен локально, следующей остановкой является Docker Hub. Вы можете указать демону **`docker`** использовать другой реестр, включив имя хоста или URL-адрес в спецификацию образа:

```
# docker pull registry.admin.com/debian:jessie
```

Аналогично при создании образа для занесения в пользовательский реестр вы должны указать его URL-адрес и пройти аутентификацию перед загрузкой.

```
# docker tag debian:jessie registry.admin.com/debian:jessie .
...
# docker login https://registry.admin.com
Username: ben
Password: <password>
# docker push registry.admin.com/debian:jessie
```

Система Docker сохраняет данные регистрации в файле, хранящемся в вашем домашнем каталоге под названием **`.dockercfg`**, поэтому вам не нужно снова входить в систему при следующем взаимодействии с персональным реестром.

Исходя из соображений, связанных с производительностью или безопасностью, вы можете использовать собственный реестр образов. Проект реестра относится к открытому исходному коду (`github.com/docker/distribution`), а простой реестр легко запускается как контейнер.[6]

```
# docker run -d -p 5000:5000 --name registry registry:2
```

Служба реестра теперь запущена на порту 5000. Вы можете извлечь из него искомый образ, указав его имя.

```
# docker pull localhost:5000/debian:jessie
```

В реестре реализованы два метода аутентификации: `token` и `htpasswd`. Метод `token` делегирует аутентификацию внешнему провайдеру, что, вероятно, потребует специальных усилий с вашей стороны. Метод `htpasswd` проще и допускает базовую аутентификацию HTTP для доступа к реестру. Кроме того, вы можете настроить прокси-сервер (например, NGINX) для обработки аутентификации. Всегда запускайте реестр с поддержкой протокола TLS.

[6]Тег **`registry:2`** отличает реестр последнего поколения от предыдущей версии, которая реализует интерфейс API, несовместимый с текущими версиями платформы Docker.

Стандартная конфигурация персонального реестра не подходит для крупномасштабного развертывания. Для использования в производственной среде нужно учесть ряд требований, связанных с пространством для хранения, требованиями к аутентификации и авторизации, к стиранию образов и другими задачами обслуживания.

По мере расширения контейнерной среды ваш реестр будет затоплен новыми образами. Для пользователей, работающих в облаке, одним из возможных способов хранения всех этих данных являются объектные хранилища, такие как Amazon S3 или Google Cloud Storage. Реестр поддерживает обе службы.

Еще лучше перенаправить свои функции реестра в реестры, встроенные в выбранную вами облачную платформу, и одной проблемой станет меньше. Обе компании, Google и Amazon, предоставляют услуги управляемого реестра контейнеров. Вы платите за хранение и сетевой трафик, необходимый для загрузки и выгрузки образов.

25.3. Контейнеры на практике

Как только вы освоите общие принципы работы с контейнерами, вы обнаружите, что в контейнеризованном мире к административным функциями необходимо подходить по-другому. Например, как вы управляете файлами журналов для контейнерных приложений? Каковы требования безопасности? Как устранить ошибки?

В приведенном ниже списке содержатся эмпирические правила, которые помогут вам адаптироваться к жизни внутри контейнера.

- Если вашему приложению необходимо выполнить запланированное задание, не запускайте планировщик **cron** в контейнере. Для того чтобы создать расписание для короткоживущего контейнера, который запускает задание и завершает свою работу, используйте демон **cron** на хосте (или таймер **systemd**). Контейнеры предназначены для одноразового использвания.
- Вам необходимо войти в систему и проверить, что делает процесс? Не запускайте демона **sshd** в контейнере. Подключитесь к хосту с помощью команды **ssh**, а затем используйте команду **docker exec**, чтобы открыть интерактивную оболочку.
- По возможности настройте свое программное обеспечение так, чтобы оно получало информацию о конфигурации из переменных окружения. Вы можете передавать переменные окружения в контейнеры с помощью команды **docker run** и аргумента **-e** *ключ=значение*. Или установите сразу несколько переменных окружения из отдельного файла с помощью параметра **-env-file** *имя_файла*.
- Игнорируйте распространенное мнение, утверждающее, что в контейнере должен существовать только один процесс. Это вздор. Распределяйте процессы по отдельным контейнерам только тогда, когда это целесообразно. Например, обычно рекомендуется запускать приложение и его сервер базы данных в отдельных контейнерах, поскольку они разделены четкими архитектурными границами. Однако ничего страшного, если в контейнере есть несколько процессов. Руководствуйтесь здравым смыслом.
- Сосредоточьтесь на автоматическом создании контейнеров для вашей среды. Напишите сценарии для создания образов и их загрузки в реестры. Убедитесь, что процедуры развертывания программного обеспечения включают замену контейнеров, а не обновление их на месте.
- Избегайте обслуживания контейнеров. Если вы вручную загружаете контейнер, чтобы исправить что-то, выясните, в чем проблема, устраните ее на образе, а затем

замените контейнер. Немедленно обновите инструмент автоматизации, если это необходимо.

- Столкнулись с трудностями? Задавайте вопросы, используя список рассылки пользователей Docker Slack Community или IRC-канал `#docker` в сети `freenode`.

Все, что необходимо для работы приложения, должно быть доступно в его контейнере: файловая система, доступ к сети и функциональные средства ядра. Единственными процессами, которые запускаются в контейнере, являются те, которые запускаете вы. Для контейнеров нетипично запускать обычные системные службы, такие как **`cron`**, **`rsyslogd`** и **`sshd`**, хотя это, безусловно, возможно. Эти обязанности лучше всего оставить для хоста операционной системы. Если эти службы вам нужны именно в контейнере, еще раз исследуйте свою проблему и попытайтесь найти более удобный для контейнеров способ.

Ведение журнала

Для вывода диагностических сообщений в приложениях UNIX и Linux традиционно используется система Syslog (теперь демон **`rsyslogd`**). Система Syslog выполняет фильтрацию сообщений в журналах, а также их сортировку и маршрутизацию в удаленные системы. В некоторых приложениях система Syslog не используется, и вместо этого сообщения записываются непосредственно в файлы журнала.

Система Syslog не запускается в контейнерах. Вместо этого система Docker собирает диагностические сообщения с помощью специальных драйверов ведения журнала. Для этого контейнерные процессы должны записывать сообщения только в поток `STDOUT`, а ошибки — только в поток `STDERR`. Система Docker собирает эти сообщения и отправляет их в указанный пункт назначения.

Если ваше программное обеспечение поддерживает ведение журнала только с помощью файлов, примените тот же метод, что и в примере с сервером NGINX, приведенном выше: при сборке образа создайте символические ссылки из файлов журнала в потоки **`/dev/stdout`** и **`/dev/stderr`**.

Система Docker пересылает журнальные записи, которые она получает, выбираемому драйверу ведения журнала. В табл. 25.4 перечислены некоторые из наиболее распространенных и полезных драйверов ведения журнала.

Таблица 25.4. Драйверы ведения журнала в системе Docker

Драйвер	Предназначение
`json-file`	Заносит журнальные записи в формате JSON в каталог данных демона (по умолчанию)[a]
`syslog`	Заносит журнальные записи в место, указанное системой Syslog[б]
`journald`	Заносит записи в журнал **`systemd`**[a]
`gelf`	Заносит журнальные записи в расширенном формате Graylog
`awslogs`	Заносит журнальные записи в службу AWS CloudWatch
`gcplogs`	Заносит записи в службу Google Cloud Logging
`none`	Не ведет журналы

[a] Журнальные записи, хранящиеся таким образом, доступны для просмотра с помощью команды **`docker logs`**.

[б] Поддерживает протоколы UDP, TCP и TCP+TLS.

При использовании драйверов `json-file` или `journald` вы можете получить доступ к данным журнала из командной строки с помощью команды **`docker logs`** *`id_контейнера`*.

Стандартный драйвер ведения журнала для демона **dockerd** устанавливается с помощью параметра командной строки **--log-driver**. Вы также можете назначить драйвер ведения журнала во время запуска контейнера с помощью команды **docker run --logging-driver**. Некоторым драйверам можно передать дополнительные параметры. Например, параметр **--log-opt** *макс-размер* настраивает ротацию журнальных файлов для драйвера `json-file`. Используйте этот параметр, чтобы не заполнять диск журнальными файлами. Подробная информация содержится в документации по ведению журнала в системе Docker.

Советы по безопасности

Безопасность контейнеров зависит от процессов, выполняемых внутри контейнеров, которые не могут получить доступ к файлам, процессам и другим ресурсам вне их песочницы. Уязвимости, позволяющие злоумышленникам вырваться из контейнеров, известны как *атаки на прорыв* (breakout attack), — это серьезные, но редкие атаки. Код, лежащий в основе изоляции контейнеров, был включен в ядро Linux по крайней мере с 2008 г.; он зрелый и стабильный. Как и в случае с незащищенными или виртуализованными системами, небезопасные конфигурации являются гораздо более вероятным источником компрометации, чем уязвимости на изолирующем уровне.

Платформа Docker поддерживает интересный список известных программных уязвимостей, которые могут смягчаться путем контейнеризации (см. сайт `docs.docker.com/engine/security/non-events`).

Ограничьте доступ к демону

Прежде всего защитите демон **dockerd**. Поскольку **dockerd** обязательно работает с повышенными привилегиями, любой пользователь, имеющий доступ к этому демону, может легко получить полный привилегированный доступ к хосту.

Этот риск демонстрирует следующая последовательность команд.

```
$ id
uid=1001(ben) gid=1001(ben) groups=1001(ben),992(docker)
# docker run --rm -v /:/host -t -i debian bash
root@e51ae86c5f7b:/# cd /host
root@e51ae86c5f7b:/host# ls
bin   dev   home  lib64   mnt   proc   run    srv   test  usr
boot  etc   lib   media   opt   root   sbin   sys   tmp   var
```

Эти сообщения показывают, что любой пользователь в группе `docker` может подключить корневую файловую систему хоста к контейнеру и получить полный контроль над ее содержимым. Это всего лишь один из многих возможных способов повышения привилегий с помощью платформы Docker.

Если по умолчанию для связи с демоном вы используете сокет домена UNIX, добавьте только доверенных пользователей в группу `docker`, которая имеет доступ к сокету. Еще лучше контролировать доступ с помощью программы **sudo**.

Используйте TLS

Мы говорили об этом раньше и повторим еще раз: если демон **dockerd** должен быть доступен удаленно по сети (запущен с параметром **-H**), необходимо использовать протокол TLS для шифрования сетевых сообщений и взаимной аутентификации клиента и сервера.

🕮 Дополнительную информацию о протоколе TLS см. в разделе 27.6.

Настройка TLS предполагает наличие авторитетного органа, выдающего сертификаты для демона **dockerd** и клиентов. После инсталляции пары ключей и центра сертификации подключение протокола TLS для **docker** и **dockerd** просто сводится к указанию правильных аргументов командной строки. Основные параметры перечислены в табл. 25.5.

Таблица 25.5. Аргументы TLS, общие для программы `docker` и демона `dockerd`

Аргумент	Значение или аргумент
`--tlsverify`	Требуется аутентификация
`--tlscert`[a]	Путь к подписанному сертификату
`--tlskey`[a]	Путь к закрытому ключу
`--tlscacert`[a]	Путь к сертификату доверенного органа

[a] Необязательный аргумент. По умолчанию находится в каталоге `~/.docker/{cert,key,ca}.pem`.

Успешное использование протокола TLS зависит от зрелости процессов управления сертификатами. Выдача сертификатов, аннулирование и истечение срока их действия — вот некоторые из вопросов, которые требуют внимания. В основном — это бремя администратора, ответственного за безопасность.

Выполняйте процессы от имени непривилегированного пользователя

Процессы в контейнерах должны выполняться от имени непривилегированного пользователя, как это происходит в полнофункциональной операционной системе. Эта практика ограничивает способность злоумышленников организовывать атаки на прорыв. Когда вы создаете файл **Dockerfile**, используйте инструкцию `USER` для запуска будущих команд на образе под учетной записью с указанным именем пользователя.

Используйте корневую файловую систему в режиме только для чтения

Для того чтобы дополнительно ограничить использование контейнеров, вы можете задать команду **`docker run --read-only`**, тем самым ограничивая контейнер рамками корневой файловой системы, доступной только для чтения. Это решение хорошо подходит для служб без состояния, которым никогда и ничего не нужно записывать. Вы также можете смонтировать том для чтения-записи, который ваш процесс может изменять, но оставить корневую файловую систему только для чтения.

Ограничивайте возможности

🕮 Дополнительную информацию о возможностях Linux см. в разделе 3.3.

В ядре Linux определены 40 отдельных возможностей, которые могут быть назначены процессам. По умолчанию контейнеры Docker имеют большой набор возможностей из этого списка. Вы можете включить еще большее подмножество возможностей, запустив контейнер с флагом **`--privileged`**. Однако этот параметр отключает многие преимущества изоляции при использовании платформы Docker. Вы можете настроить определенные возможности, доступные для контейнерных процессов, с помощью аргументов **`--cap-add`** и **`--cap-drop`**:

```
# docker run --cap-drop SETUID --cap-drop SETGID debian:jessie
```

Вы также можете удалить все привилегии и добавить обратно те, которые вам нужны:

```
# docker run --cap-drop ALL --cap-add NET_RAW debian:jessie
```

Защищайте образы

Функция доверия в системе Docker позволяет проверять подлинность и целостность образов в реестре. Издатель образа подписывает его секретным ключом, а реестр проверяет его с помощью соответствующего открытого ключа. Этот процесс гарантирует, что образ был создан ожидаемым автором. Вы можете использовать доверительное содержимое для подписывания собственных образов или для проверки образов в удаленном реестре. Эта функция доступна в хранилище Docker Hub и в некоторых независимых реестрах, таких как Artifactory.

К сожалению, большая часть содержимого в хранилище Docker Hub является неподписанной и должна считаться ненадежной. На самом деле большинство образов в Docker Hub никогда не исправляются, не обновляются и не проверяются каким-либо образом.

Отсутствие надлежащей цепи доверия, ассоциированной со многими образами Docker, является показателем жалкого состояния безопасности в Интернете в целом. Обычно пакеты программного обеспечения зависят от сторонних библиотек, которые мало или вообще не заботятся о достоверности заложенного в них содержимого. В некоторых хранилищах программного обеспечения нет криптографических подписей вообще. Также часто можно встретить статьи, авторы которых активно поощряют отключение проверки. Ответственные системные администраторы должны очень подозрительно относиться к неизвестным и ненадежным репозиториям программного обеспечения.

Отладка и устранение неполадок

Контейнеры приносят с собой особенно отвратительное усложнение и без того мутных методов отладки. Когда приложение заключается в контейнер, симптомы этого явления становятся более сложными, а их исходные причины становятся более загадочными. Многие приложения могут работать без изменений внутри контейнера, но в некоторых ситуациях могут вести себя по-другому. Вы также можете столкнуться с ошибками внутри самой системы Docker. Этот раздел поможет вам ориентироваться в этих мутных водах.

Ошибки обычно проявляются в журнальных файлах, поэтому с них и нужно начать поиск проблемы. Используйте рекомендации из раздела "Ведение журнала", чтобы настроить ведение журнала для контейнеров и всегда просматривайте журналы при возникновении проблем. Если у вас возникли проблемы с запущенным контейнером, чтобы открыть его интерактивную оболочку, попробуйте выполнить команду

```
# docker exec -ti имя_контейнера bash
```

В этой оболочке вы можете попытаться воспроизвести проблему, проверить файловую систему на наличие ошибок и выявить проблемы в конфигурации. Если вы видите ошибки, связанные с демоном **dockerd**, или если у вас возникли проблемы с его запуском, выполните поиск в списке проблем по адресу `github.com/moby/moby`. Вы можете найти других коллег, которые столкнулись с такой же проблемой, и, наверняка, один из них сможет предложить потенциальное исправление или обходное решение.

Система Docker не стирает образы или контейнеры автоматически. Если пренебречь этим обстоятельством, такие остатки могут занять чрезмерный объем дискового пространства. Если рабочая нагрузка вашего контейнера предсказуема, настройте задание **cron** для удаления ненужного материала, выполнив в нем команды **docker system prune** и **docker image prune**.

С этой же проблемой связана еще одна досадная особенность — "заброшенные" тома, некогда прикрепленные к контейнеру, который был удален. Тома не зависят от контейнеров, поэтому любые файлы внутри них будут продолжать занимать дисковое пространство до тех пор, пока эти тома не будут уничтожены. Для удаления потерянных томов можно использовать следующие команды.

```
# docker volume ls -f dangling=true # Получаем список заброшенных томов
# docker volume rm $(docker volume ls -qf dangling=true) # Удаляем их
```

Базовые образы, от которых зависят созданные вами образы, могут иметь инструкцию **VOLUME** в своем файле **Dockerfile**. Если вы не обратите на это внимание, то можете получить переполнение диска после запуска нескольких контейнеров из этого образа. Для того чтобы вывести список томов, связанных с контейнером, выполните команду **docker inspect**:

```
# docker inspect -f '{{ .Volumes }}' имя_контейнера
```

25.4. Создание и управление контейнерными кластерами

Одним из величайших достижений контейнеризации является перспектива совместного размещения многих приложений на одном и том же хосте. Это позволяет избежать взаимозависимостей и конфликтов и тем самым обеспечить более эффективное использование серверов. Это привлекательная перспектива, но демон Docker отвечает только за отдельные контейнеры, а не за более широкий вопрос о том, как запускать множество контейнеров на распределенных хостах в конфигурации с высокой доступностью.

Все инструменты управления конфигурацией, такие как Chef, Puppet, Ansible и Salt, поддерживают систему Docker. Они могут гарантировать, что на хостах запускаются определенные наборы контейнеров с объявленными конфигурациями. Они также поддерживают создание образов, интерфейсы реестра, управление сетью и томами, а также другие задачи, связанные с контейнерами. Эти инструменты централизуют и стандартизуют конфигурацию контейнеров, но не решают проблему развертывания множества контейнеров в сети серверов. (Обратите внимание на то, что, хотя системы управления конфигурацией полезны для различных задач, связанных с контейнером, вам редко придется использовать управление конфигурацией внутри контейнеров.)

Для развертывания контейнеров в масштабах всей сети вам необходимо программное обеспечение для совместной работы контейнеров, также известное как *планирование контейнеров* или *программное обеспечение для управления контейнерами*. Для обработки большого количества контейнеров доступно множество открытого и коммерческого инструментария. Такие инструменты имеют решающее значение для запуска контейнеров в производственном масштабе.

Чтобы понять, как работают эти системы, подумайте о серверах в сети как о ферме вычислительных мощностей. Каждый хост фермы предлагает планировщик для процессора, памяти, дисков и сетевых ресурсов. Когда планировщик получает запрос на запуск контейнера (или набора контейнеров), он помещает контейнер на хост, у которого есть достаточные запасные ресурсы для удовлетворения потребностей контейнера. Поскольку планировщик знает, где были размещены контейнеры, он также может помочь в маршрутизации сетевых запросов на правильные хосты в кластере. Администраторы взаимодействуют с системой управления контейнерами, а не с каким-либо отдельным механизмом контейнера. Эта архитектура показана на рис. 25.4.



Рис. 25.4. Типичная архитектура планировщика контейнеров

Системы управления контейнерами предоставляют несколько полезных функций.

- Алгоритмы планирования выбирают лучший хост в свете запрошенных ресурсов задания и использования кластера. Например, задание с высокими требованиями к пропускной способности сети может быть размещено на хосте с сетевым интерфейсом 10 Гбит/с.
- Формальные интерфейсы API позволяют программам отправлять задания в кластер, открывая возможности для интеграции с внешними инструментами. Это позволяет легко использовать системы управления контейнерами в сочетании с системами CI/CD для упрощения развертывания программного обеспечения.
- Размещение контейнеров может удовлетворить потребности конфигураций с высокой доступностью. Например, может потребоваться запустить приложение на узлах кластера, расположенного в нескольких разных географических регионах.
- В систему встроен мониторинг работоспособности. Система может прекратить работу и перенести неисправные рабочие места, а также перенаправить задания из неисправных узлов кластера.
- Существует возможность легко добавлять или удалять вычислительную емкость. Если ваша вычислительная ферма не располагает достаточными ресурсами для удовлетворения спроса, вы можете просто добавить другой узел. Эта возможность особенно хорошо подходит для облачных сред.
- Система управления контейнером может взаимодействовать с балансировщиком нагрузки для маршрутизации сетевого трафика от внешних клиентов. Это средство устраняет сложный административный процесс ручной настройки доступа к сети в контейнеризованных приложениях.

Одной из наиболее сложных задач в распределенной контейнерной системе является сопоставление имен служб с контейнерами. Помните, что контейнеры обычно запускаются на непродолжительное время и им могут назначаться номера портов динамически. Как сопоставить удобное, постоянное имя службы с несколькими контейнерами, особенно когда узлы кластера и порты часто меняются? Эта проблема известна как обнаружение служб, а системы управления контейнерами имеют различные ее решения.

Все эти функции позволяют ознакомиться с базовым механизмом выполнения контейнера перед погружением в инструментарий оркестровки. Все системы управления контейнерами, о которых мы знаем, используют систему Docker в качестве стандартного механизма выполнения контейнеров, хотя некоторые системы также поддерживают другие механизмы.

Краткий обзор программного обеспечения для управления контейнерами

Несмотря на свою относительную молодость, инструменты управления контейнерами, которые мы обсуждаем ниже, являются зрелыми и могут считаться промышленными. Фактически многие из них уже используются на производстве в крупномасштабных технологических компаниях.

Большинство из них представляют собой программное обеспечение с открытым исходным кодом и имеют значительные сообщества пользователей. Основываясь на последних тенденциях, мы ожидаем существенного развития в этой области в ближайшие годы. В следующих разделах мы расскажем о функциональных возможностях и особенностях наиболее широко используемых систем. Мы также упомянем их точки интеграции и типичные сценарии использования.

Kubernetes

Название Kubernetes иногда сокращается до "k8s", потому что между первой буквой *k* и последней буквой *s* стоят восемь букв. Так называется проект, который стал лидером в области управления контейнерами. Он разработан компанией Google и был запущен частью из тех же разработчиков, которые работали над проектом Borg, менеджером внутреннего кластера Google. Kubernetes был выпущен в качестве проекта с открытым исходным кодом в 2014 г. и теперь имеет более тысячи активных участников. Он имеет большинство функциональных возможностей и самый быстрый цикл разработки любой системы, о которой мы знаем.

Программное обеспечение Kubernetes состоит из нескольких отдельных служб, которые объединяются для формирования кластера. К его основным строительным блокам относятся следующие компоненты.

- Сервер API для запросов оператора
- Планировщик для размещения задач
- Менеджер контроллеров для отслеживания состояния кластера
- Kubelet — агент, который работает на всех узлах кластера
- Служба cAdvisor для мониторинга показателей контейнеров
- Прокси-сервер для маршрутизации входящих запросов в соответствующий контейнер

Для обеспечения высокой доступности первые три элемента в этом списке выполняются на нескольких главных серверах (которые могут при необходимости выполнять двойные обязанности как узлы кластера). Процессы Kubelet и cAdvisor запускаются на каждом узле, обрабатывают запросы от менеджера контроллеров и сообщают статистику о состоянии своих задач.

В Kubernetes контейнеры размещаются в виде модуля, содержащего один или несколько контейнеров. Все контейнеры в модуле обязательно размещаются на одном узле кластера. Каждому модулю присваивается уникальный IP-адрес на уровне кластера, и он получает метку для целей идентификации и размещения.

Модули не предназначены для долговременного использования. Если узел прекращает работу, контроллер планирует перенос модуля на другой узел с новым IP-адресом. По этой причине нельзя использовать адрес модуля как долговременное имя.

Службы представляют собой коллекции связанных модулей с адресом, который никогда не изменяется. Если модуль внутри службы прекращает существование или не проходит проверку работоспособности, служба удаляет этот модуль из ротации. Можно также использовать встроенный DNS-сервер для назначения службам распознаваемых имен.

Система Kubernetes интегрировала поддержку для обнаружения служб, управления ключами, развертывания и автоматического масштабирования модулей. Она имеет подключаемые сетевые опции для поддержки оверлейных сетей контейнера. Она может поддерживать приложения с фиксацией состояния путем переноса томов между узлами кластера по мере необходимости. Ее инструмент командной строки под названием `kubectl` является одним из самых интуитивно понятных, с которыми мы когда-либо работали. Короче говоря, он имеет набор гораздо более сложных функций, чем те, что мы включили в этот короткий раздел.

Хотя система Kubernetes обладает самым активным и деятельным сообществом и оснащена самыми передовыми функциями, эти преимущества компенсируются крутой кривой обучения. Недавние версии упростили процесс освоения для новичков, но полноценное, тонко настроенное развертывание системы Kubernetes — занятие не для робких. Развертывание промышленных версий k8s связано с большой административной и оперативной нагрузкой.

На основе системы Kubernetes реализована служба Google Container Engine — одна из самых удобных для команд, которые хотят использовать контейнерные рабочие нагрузки без эксплуатационных издержек управления кластерами.

Mesos и Marathon

Mesos — проект совершенно другого рода. Он был задуман в Калифорнийском университете в Беркли около 2009 г. как универсальный менеджер кластера. Он быстро пробился в Twitter, где теперь работает на тысячах узлов. Сегодня Mesos является приоритетным проектом от организации Apache Foundation и может похвастаться большим количеством корпоративных пользователей.

Основными концептуальными объектами в Mesos являются мастера, агенты и каркасы. Мастер — это посредник между агентами и каркасами. Мастера ретранслируют предложения системных ресурсов от агентов к каркасам. Если у каркаса есть задача для выполнения, он выбирает предложение и отдает мастеру приказ выполнить задачу. Мастер отправляет данные задачи агенту.

Marathon — это каркас системы Mesos, который развертывает контейнеры и управляет ими. Он включает красивый пользовательский интерфейс для управления приложениями и простой интерфейс RESTful API. Чтобы запустить приложение, вы пишете определение запроса в формате JSON и отправляете его в Marathon через API или пользовательский интерфейс. Поскольку это внешний каркас структуры, его развертывание является гибким. Для удобства Marathon может работать на том же узле, что и мастер, или же запускаться извне.

Поддержка нескольких сосуществующих каркасов — самый главный отличительный признак системы Mesos. Apache Spark, инструмент обработки больших данных, и Apache Cassandra, база данных NoSQL, предлагают каркасы Mesos, что позволяет использовать агенты Mesos как узлы в кластере Spark или Cassandra. Каркас Chronos предназначен для планирования заданий, скорее, как версия `cron`, которая работает в кластере, а не а отдельной машине. Возможность запуска такого количества каркасов на одном и том же наборе узлов является удобной функциональной возможностью и помогает выработать единый и централизованный опыт для администраторов.

В отличие от системы Kubernetes, Mesos не поставляется с готовым набором функций. Например, балансировка нагрузки и маршрутизация трафика являются подключаемыми опциями, которые зависят от вашего выбора. Вы можете выбрать инструмент Marathon-lb, который реализует эту услугу, или же использовать другой. Например, мы успешно использовали инструменты Consul и HAProxy компании HashiCorp. Проектирование и внедрение точного решения остается в качестве упражнения для администратора.

Изучение систем Kubernetes и Mesos требуют определенных усилий. Для координации кластеров система Mesos и большинство ее каркасов полагаются на службу Apache Zookeeper. Ею довольно сложно управлять, к тому же стало известно о нескольких сложных случаях сбоев. Кроме того, кластер Mesos с высокой доступностью требует минимум трех узлов, что может быть обременительным для некоторых организаций.

Менеджер Docker Swarm

Чтобы не отставать, разработчики платформы Docker предложили Swarm, менеджер контейнерного кластера, встроенный непосредственно в систему Docker. Нынешняя реализация менеджера Swarm появилась в 2016 г. как ответ на растущую популярность Mesos, Kubernetes и других кластерных менеджеров, в которых использовались контейнеры Docker. Оркестровка контейнеров теперь является основным приоритетом компании Docker, Inc.

Запустить систему Swarm легче, чем Mesos или Kubernetes. Любой узел кластера, на котором запущена система Docker, может присоединиться к системе Swarm как рабочий узел, и любой рабочий узел также может быть менеджером. Нет необходимости запускать отдельные узлы в качестве мастеров.[7] Для запуска системы Swarm достаточно выполнить простую команды **`docker swarm init`**. Нет дополнительных процессов для управления и настройки, и нет состояния для отслеживания. Все работает прямо из коробки.

Для запуска служб в системе Swarm можно использовать знакомые команды **`docker`** (как в системе Kubernetes для работы с коллекциями контейнеров). Вы объявляете состояние, которое хотите достичь (например, "мое веб-приложение должно быть запущено на трех контейнерах"), а система Swarm планирует выполнение задач в кластере. Она автоматически обрабатывает состояния отказа и выполняет обновление без простоя. У системы Swarm есть встроенный балансировщик нагрузки, который автоматически настраивается при добавлении или удалении контейнеров. Платформа балансировки Swarm не такая полнофункциональная, как инструменты NGINX или HAProxy, но, с другой стороны, она не требует никакого внимания со стороны системных администраторов.

По умолчанию система Swarm обеспечивает безопасность системы Docker. Все соединения между узлами в системе Swarm зашифрованы по протоколу TLS, и со стороны администратора никакой настройки не требуется. Это большое преимущество системы Swarm перед конкурентами.

Контейнерная служба AWS EC2

Инфраструктура AWS предлагает ECS, службу управления контейнерами, предназначенную для экземпляров EC2 (естественных виртуальных серверов AWS). В манере, напоминающей многие службы Amazon, инфраструктура AWS сначала выпустила служ-

[7] Строго говоря, это верно и для системы Kubernetes и Mesos, но мы обнаружили, что обычной практикой в конфигурациях с высокой доступностью является отделение мастеров от агентов.

бу ECS с минимальной функциональностью, но со временем улучшила ее. Служба ECS стала прекрасным выбором для организаций, которые уже вложили средства в инфраструктуру AWS и хотят придерживаться режима E-Z.

ECS — это "почти управляемая" служба. Компоненты кластерного менеджера управляются инфраструктурой AWS. Пользователи запускают экземпляры EC2, на которых установлены система Docker и агент ECS. Агент подключается к центральному API ECS и регистрирует доступность его ресурсов. Чтобы выполнить задачу в вашем кластере ECS, отправьте определение задачи в формате JSON через API. Затем ECS назначит задачу на одном из ваших узлов кластера.

Поскольку служба является "почти управляемой", порог для входа низкий. Вы можете начать работу с ECS всего за несколько минут. Служба хорошо масштабируется по меньшей мере до сотен узлов и тысяч одновременных задач.

Служба ECS взаимодействует с другими службами AWS. Например, балансировка нагрузки между несколькими задачами вместе с обнаружением необходимых служб обрабатывается службой Application Load Balancer. Вы можете добавить ресурс в свой кластер ECS, воспользовавшись автоматическим масштабированием службы EC2. Она также интегрирована со службами Identity и Access Manager инфраструктуры AWS, что позволяет предоставлять разрешения для задач вашего контейнера с целью взаимодействия с другими службами.

Одной из наиболее проработанных частей ECS является встроенный реестр образов Docker. Вы можете загружать образы Docker в реестр EC2 Container Registry, где они хранятся и становятся доступными любому клиенту Docker, независимо от того, работает он со службой ECS или нет. Если вы используете контейнеры в инфраструктуре AWS, используйте реестр контейнеров в той же области, что и ваши экземпляры. Так вы достигнете гораздо большей надежности и производительности, чем с любым другим реестром.

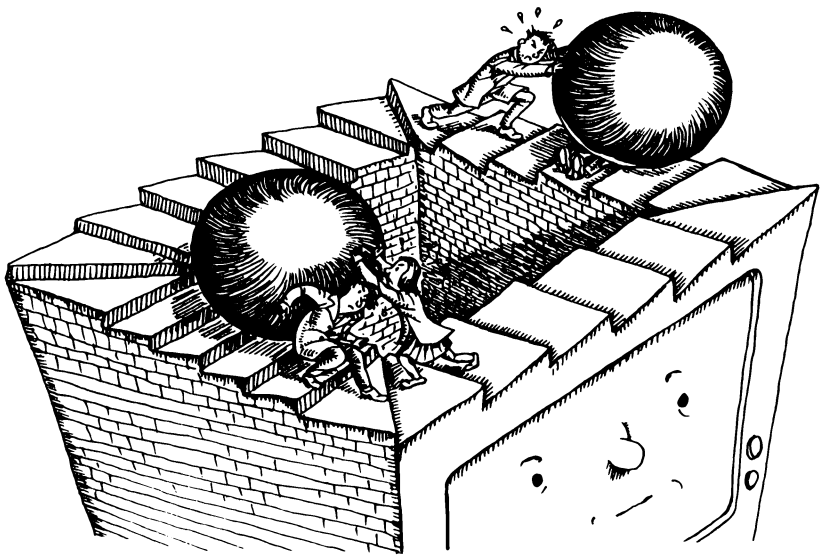
Пользовательский интерфейс ECS, хотя и функциональный, но ему присущи ограничения других интерфейсов AWS. Инструмент AWS CLI имеет полную поддержку API ECS. Для управления приложениями в службе ECS мы рекомендуем обратиться к сторонним инструментам с открытым исходным кодом, таким как Empire (`github.com/remind101/empire`) или Convox (`convox.com`).

25.5. Литература

- Docker, Inc. *Official Docker Documentation*. `docs.docker.com`. Платформа Docker имеет хорошую документацию. Она является исчерпывающей и, как правило, актуальной.
- Matthias, Kari., and Sean Kane. *Docker Up & Running*. Sebastopol, CA: O'Reilly Media, 2015. В этой книге основное внимание уделяется использованию контейнеров Docker в производственных средах.
- Mouat, Adrian. *Using Docker: Developing and Deploying software with Containers*. Sebastopol, CA: O'Reilly Media, 2016. Эта книга охватывает темы от базового до продвинутого и включает в себя множество примеров.
- Turnbull, James. *The Docker Book*. `www.dockerbook.com`.
- Блог Container Solutions на сайте `container-solutions.com/blog` содержит технические советы, рекомендации и интервью с экспертами по контейнерам.

Глава 26

Непрерывная интеграция и доставка



Еще примерно десять лет назад обновление программного обеспечения было трудным и долгим делом. В процессе выпуска обычно использовались специальные, кустарные сценарии, которые вызывались в загадочном порядке и сопровождались устаревшей и неполной документацией. Тестирование — если оно вообще существовало — выполнялось группой контроля качества, которая была далека от цикла разработки и часто становилась серьезным препятствием для поставки кода конечному потребителю. Администраторы, разработчики и руководители проектов были вынуждены планировать длительные процедуры на последних этапах выпуска обновлений для текущих пользователей. Перебои в обслуживании часто планировались за несколько недель.

Учитывая этот сомнительный контекст, неудивительно, что некоторые очень умные люди усердно работали над улучшением ситуации. В конце концов там, где одни видят только проблемы, другие видят возможность.

Одним из наиболее мудрых специалистов в этой области является Мартин Фаулер (Martin Fowler), оракул индустрии программного обеспечения и главный научный сотрудник влиятельной консалтинговой компании ThoughtWorks. В проницательной статье (`goo.gl/Y2lisI`) Фаулер описывает непрерывную интеграцию (Continuous Integration) как "практику разработки программного обеспечения, в которой члены команды часто интегрируют свою работу", тем самым устраняя одну из главных проблем в работе программного обеспечения: утомительную задачу согласования фрагментов кода, которые отдалились друг от друга в течение длительного периода независимой разработки. В на-

стоящее время практика непрерывной интеграции стала общепринятой среди разработчиков программного обеспечения.

Вершиной достижений в рамках этого подхода стала непрерывная доставка (Continuous Delivery), которая похожа на концепцию непрерывной интеграции, но направлена на достижение другой цели: надежное развертывание обновленного программного обеспечения в уже функционирующих системах. Непрерывная доставка подразумевает небольшие изменения в информационной инфраструктуре. Если что-то выходит из строя (т.е. появляется регресс), этот подход позволяет легко изолировать и решить проблему, потому что изменения между версиями малы. В крайних случаях некоторые организации стремятся разворачивать новый код для пользователей несколько раз в день. Ошибки и насущные проблемы безопасности можно решить в течение часов, а не дней или недель.

Сочетание концепций непрерывной интеграции и непрерывной доставки (далее обозначается как CI/CD) охватывает инструменты и процессы, необходимые для облегчения частого, постепенного обновления программного обеспечения и конфигурации.

🕮 Дополнительную информацию о методологии DevOps см. в разделе 31.1.

Концепция CI/CD является основой методологии DevOps. Это подход, объединяющий как специалистов, которые занимаются разработкой, так и специалистов, которые осуществляют эксплуатацию программного обеспечения. Это такой же бизнес-актив, как и технические инновации. После внедрения концепция CI/CD становится основой информационной политики организации, поскольку она определяет логику и структуру процессов выпуска, которые ранее были хаотичными.

Системные администраторы занимают центральное место в разработке, внедрении и постоянном обслуживании систем CI/CD. Администраторы устанавливают, настраивают и управляют инструментами, которые выполняют функцию CI/CD. Они несут ответственность за то, чтобы процессы сборки программного обеспечения были быстрыми и надежными.

Тестирование является важным элементом CI/CD. Администраторы не могут писать тесты (хотя иногда они это делают!), но они несут ответственность за настройку инфраструктуры и систем, на которых выполняются тесты. Возможно, самое главное, что именно системные администраторы отвечают за развертывание, т.е. компонент доставки CI/CD.

Эффективная система CI/CD реализуется не с помощью одиночного инструмента, а на основе комплексного программного обеспечения, которое работает в унисон, формируя связанную среду. Существуют разнообразные средства с открытым исходным кодом и коммерческие инструменты для координации различных элементов CI/CD. Эти инструменты координации полагаются на другие пакеты программного обеспечения для выполнения фактической работы (например, компиляция кода или настройка серверов в определенной конфигурации). Действительно, существует так много вариантов, что первоначальное знакомство с концепцией CI/CD может быть ошеломляющим. Кроме всего прочего, недавнее широкое распространение разнообразных инструментов в этой области свидетельствует о растущем значении CI/CD для отрасли.

В этой главе мы попытаемся сориентироваться в лабиринте концепций, терминологии и инструментов CI/CD. Мы освещаем основы конвейера CI/CD, различные типы тестирования и их релевантность концепции CI/CD, практику параллельной работы нескольких сред и некоторые из наиболее популярных инструментов с открытым исходным кодом. В конце главы мы рассмотрим пример конвейера CI/CD, в котором используются некоторые из самых популярных инструментов. Прочитав эту главу, вы начнете понимать принципы и методы, на которых основана мощная и гибкая система CI/CD.

26.1. Основные концепции

Многие термины, относящиеся к CI/CD, звучат одинаково и имеют перекрывающиеся значения. Итак, давайте сначала рассмотрим различия между непрерывной интеграцией, доставкой и развертыванием.

- *Непрерывная интеграция* (continuous integration — CI) — это процесс совместной работы с общей кодовой базой, слияние разрозненных изменений кода в систему контроля версий и автоматическое создание и тестирование сборок.
- *Непрерывная доставка* (continuous delivery — CD) — это процесс автоматического развертывания сборок в непроизводственных средах после завершения процесса непрерывной интеграции.
- *Непрерывное развертывание* (continuous deployment) завершает цикл путем развертывания в функционирующих системах, которые обслуживают реальных пользователей без какого-либо вмешательства оператора.

Непрерывное развертывание без какого-либо контроля со стороны людей может быть пугающим, но это именно так: идея состоит в том, чтобы уменьшить фактор страха путем развертывания программного обеспечения как можно чаще, тем самым устраняя все больше и больше проблем, пока команда не будет достаточно уверенной в результатах тестирования и инструментах, чтобы затем дать разрешение на автоматический выпуск.

Непрерывное развертывание не обязательно должно быть конечной целью всех организаций. В любой момент в конвейере могут возникнуть проблемы и риски. Если это так, вы все равно можете сделать так, чтобы каждый этап процесса был максимально простым для человека, который нажимает последнюю кнопку. Каждая организация должна устанавливать свои собственные границы.

Принципы и практика

Гибкость бизнеса является одним из ключевых преимуществ подхода CI/CD. Непрерывное развертывание облегчает выпуск проверенных функций на производстве в течение нескольких минут или часов вместо недель или месяцев. Поскольку каждое изменение собирается, тестируется и развертывается немедленно, разница между версиями становится намного меньше. Это снижает риск развертывания и помогает сузить круг причин возможных неполадок. Вместо того чтобы организовывать небольшое количество развертываний в год после внесения большого количества изменений, вы можете выпускать новый код несколько раз в неделю или даже в день.

Подход CI/CD стимулирует выпускать новый функционал больше и чаще. Эта цель достижима только тогда, когда разработчики пишут, отлаживают и фиксируют в общем хранилище небольшие фрагменты кода. Чтобы реализовать непрерывную интеграцию, разработчикам необходимо проводить фиксацию изменений кода не реже одного раза в день после проведения локальных тестов.

Для системных администраторов процессы CI/CD значительно сокращают время, затрачиваемое на подготовку и реализацию выпусков. Они также сокращают время отладки, если при развертывании возникают неизбежные проблемы. Немного на свете более приятных занятий, чем наблюдение за выпуском новой функции на производстве без вмешательства человека. В следующих разделах описаны некоторые основные правила, которые следует учитывать при разработке процессов CI/CD.

Использовать контроль версий

Весь код следует отслеживать в системе контроля версий. Мы рекомендуем систему Git, но есть много вариантов. Само собой разумеется, что большинство команд разработчиков программного обеспечения используют систему контроля версий.

В организациях, которые приняли на вооружение концепцию инфраструктуры как кода (см. раздел "Подход CI/CD на практике"), вы можете отслеживать инфраструктурный код вместе с вашими приложениями. Вы даже можете хранить документы и настройки конфигурации в системе контроля версий.

Убедитесь, что контроль версий является единственным источником истины. *Ничто* не должно управляться вручную или без фиксации записи.

Собирайте один раз, развертывайте часто

Конвейер CI/CD начинается со сборки. С этого момента результат сборки ("артефакт") используется для тестирования и развертывания. Единственный способ подтвердить, что конкретная сборка готова к производству, — пропустить данную сборку через все тесты. Разверните один и тот же артефакт по крайней мере в одной или двух средах, которые как можно лучше соответствуют целевой производственной платформе.

Сквозная автоматизация

Сборка, тестирование и развертывание кода без ручного вмешательства является ключом к надежным и воспроизводимым обновлениям. Даже если вы не планируете постоянно развертывать код на производстве, конечный шаг развертывания должен выполняться без вмешательства человека.

Собирайте каждую фиксацию в процессе интеграции

Интеграция объединяет изменения, сделанные несколькими разработчиками или командами разработчиков. Продукт представляет собой составную кодовую базу, которая включает в себя все обновления. Интеграция не должна случайным образом прерывать работу разработчиков и заставлять их помещать фрагменты кода в основную кодовую базу; это верный путь к катастрофе. Отдельные разработчики несут ответственность за управление собственным потоком разработки. Когда они будут готовы, они должны начать процесс интеграции. Интеграции должны происходить как можно чаще.

Интеграции выполняются через систему контроля версий исходного кода. Рабочий процесс может быть разным. Ответственность за слияние своей работы с основной веткой проекта (trunk) могут нести отдельные разработчики, или же назначенный наблюдатель за выпусками может одновременно интегрировать работу сразу нескольких разработчиков или команд. Процесс слияния может быть в значительной степени автоматизирован, но всегда существует возможность конфликта между двумя наборами изменений. Такая ситуация требует вмешательства человека.

Идея непрерывной интеграции заключается в том, что любая фиксация в интеграционной ветке системы контроля версий должна автоматически запускать процесс сборки. "Интеграционная ветка" важна, потому что контроль версий исходного кода служит нескольким целям. Помимо того, что он является средством сотрудничества и интеграции, он также полезен как система резервного копирования, как контрольная точка для незавершенного производства и как система, которая позволяет разработчикам работать с несколькими обновлениями, сохраняя при этом изменения, связанные с этими обновлениями, логически раздельными. Таким образом, только фиксация результата интеграции должна приводить к запуску процесса сборки.

Частая интеграция позволяет легко выявить причины неудачного построения сборки и выявить точные строки кода, которые вызвали проблему. Затем система контроля версий может определить личность ответственного разработчика. Но обратите внимание: неисправная сборка не должна стать причиной административного наказания или стать позором конкретного исполнителя. Наша цель состоит в том, чтобы снова запустить эту сборку. Поощряйте политику отказа от поиска виновных в своих командах.

Разделяйте ответственность

Если что-то пойдет не так, конвейер необходимо исправить. Никакой новый код не может быть внедрен до тех пор, пока предыдущая проблема не будет решена. Это эквивалентно остановке сборочной линии на производстве. Вся команда обязана исправить сборку, прежде чем возобновить работу по разработке.

Подход CI/CD не должен быть таинственной системой, которая работает в фоновом режиме и иногда отправляет электронную почту, если что-то идет не так, как ожидалось. Каждый член команды должен иметь доступ к интерфейсу CI/CD для просмотра панелей мониторинга и журналов. В некоторых организациях создаются юмористические виджеты, такие как светильники RGB, которые служат визуальным индикатором текущего состояния конвейера.

Собирайте быстро, исправляйте быстро

Подход CI/CD предназначен для обеспечения как можно более быстрой обратной связи, в идеале в течение нескольких минут после фиксации кода в системе контроля версий исходного кода. Этот быстрый отклик гарантирует, что разработчики обратят внимание на результат. Если сборка завершится неудачей, разработчики, скорее всего, смогут быстро решить проблему, потому что изменения, которые они только что внесли, еще свежи в их памяти. Медленные процессы сборки контрпродуктивны. Стремитесь устранять лишние и трудоемкие этапы. Убедитесь, что ваша система сборки имеет достаточно агентов и что агенты имеют достаточные системные ресурсы для быстрой сборки.

Аудит и проверка

Часть системы CI/CD включает подробную историю каждого выпуска программного обеспечения, включая его переход от разработки к производству. Эта возможность отслеживания может быть полезна для обеспечения развертывания только авторизованных сборок. Параметры и временны*е* рамки событий, связанные с каждой средой, могут быть неопровержимо подтверждены.

Среды

Приложения не работают изолированно. Они зависят от внешних ресурсов, таких как базы данных, прокси-серверы, сетевые файловые системы, записи DNS, удаленные интерфейсы HTTP API, другие приложения и внешние сетевые службы. Среда выполнения включает все эти ресурсы и все остальное, что необходимо для приложения, чтобы оно могло работать. Создание и поддержание такой среды является объектом значительного административного внимания.

Большинство организаций работают как минимум в трех средах, перечисленных здесь в порядке возрастания важности.

- Среда разработки для интеграции обновлений от нескольких разработчиков, тестирования изменений инфраструктуры и проверки на наличие очевидных сбоев. Эта среда используется главным образом техническим персоналом, а не бизнесме-

нами или конечными пользователями. В контексте CI/CD среда разработки может создаваться и разрушаться несколько раз в день.

- Промежуточная среда для ручного и автоматизированного тестирования и для дальнейшей проверки изменений и обновлений программного обеспечения. Некоторые организации называют ее тестовой средой. Тестировщики, владельцы продуктов и другие заинтересованные стороны бизнеса используют промежуточную среду для тестирования новых функций и исправления ошибок. Такие среды могут также применяться для тестирования незаконного проникновения и других проверок безопасности.
- Производственная среда для предоставления услуг конечным пользователям. Производственная среда обычно включает в себя обширные меры для обеспечения высокой производительности и безопасности. Сбой на производстве — это чрезвычайная ситуация, которая должна быть устранена немедленно общими усилиями.

Типичная система CI/CD последовательно продвигает программное обеспечение через каждую из этих сред, отфильтровывая ошибки и дефекты программного обеспечения на этом пути. Вы можете с уверенностью развернуть приложение в производственной среде, потому что знаете, что изменения уже были протестированы в двух других средах.

Паритет сред является предметом некоторой сложности для системных администраторов. Цель непроизводственных, или "нижних", сред заключается в подготовке и анализе изменений всех типов до перехода на стадию производства. Существенные различия между средами могут привести к непредвиденной несовместимости, которая в конечном итоге может вызвать ухудшение производительности, простои или даже разрушение данных.

Например, представьте, что среда разработки и промежуточного тестирования подверглась обновлению операционной системы, но в производственной среде все еще работает ее более старая версия. Пришло время для развертывания программного обеспечения. Новое программное обеспечение тщательно проверено на стадиях разработки и тестирования и, похоже, работает нормально. Однако во время развертывания в производственной среде становится очевидной неожиданная несовместимость, поскольку в старой версии определенной библиотеки отсутствует функциональная возможность, используемая в новом коде.

Этот сценарий довольно распространен, и это одна из причин, по которой системные администраторы должны проявлять бдительность в отношении синхронизации сред. Чем ближе производственная среда к среде более низкого уровня, тем выше ваши шансы на поддержание высокой доступности и доставки программного обеспечения.

Запуск нескольких сред на полную мощность может быть дорогостоящим и трудоемким. Поскольку производственная среда обслуживает гораздо больше пользователей, чем среды более низкого уровня, в этой среде обычно необходимо запускать больше дорогих систем. Наборы данных на производстве, как правило, крупнее, поэтому выделенное под него дисковое пространство и мощности сервера пропорционально увеличиваются.

Даже такой тип различий между средами может вызвать непредвиденные проблемы. Неправильная конфигурация балансировки нагрузки, которая не имеет значения в средах разработки и тестирования, может выявить дефект. Или запрос базы данных, который быстро запускается в средах разработки и тестирования, может оказаться намного медленнее при применении к данным производственного масштаба.

Совместимость производственных мощностей в средах более низкого уровня — сложная проблема. Стремитесь иметь хотя бы одну среду более низкого уровня, у которой есть избыточность там же, где она есть в производственной среде (например, несколько веб-серверов,

полностью реплицированные базы данных и соответствующие стратегии перехода на другой ресурс в случае отказа для любых кластерных систем). Это нормально для промежуточных серверов меньшего размера, хотя любые тесты, которые вы запускаете для проверки производительности, не будут отражать реальные производственные показатели.

Для достижения наилучших результатов наборы данных в средах более низкого уровня должны быть схожими по размеру и содержанию с объемами данных в производственной среде. Одна из стратегий — создавать по ночам снимки всех соответствующих производственных данных и копировать их в среду более низкого уровня. Для обеспечения соблюдения и обеспечения надлежащей гигиены безопасности конфиденциальные пользовательские данные должны быть сделаны анонимными до того, как они будут использованы таким образом. Для действительно массивных наборов данных, которые нецелесообразно копировать, импортируйте меньший, но все еще значимый образец.

Несмотря на все ваши усилия, среда более низкого уровня никогда не будет похожа на производственную среду. Некоторые параметры конфигурации (такие как учетные данные, URL-адреса, адреса и имена хостов) будут отличаться. Используйте управление конфигурацией для отслеживания этих элементов конфигурации между средами. Когда система CI/CD запускает развертывание, проконсультируйтесь с вашим авторитетным источником, чтобы найти соответствующую конфигурацию для этой среды, и убедитесь, что все среды развернуты одинаково.

Флаги функций

Флаг функции включает или отключает функцию приложения в зависимости от значения параметра конфигурации. Разработчики могут создавать поддержку флагов функций в своем программном обеспечении. Вы можете использовать флаги функций для включения определенных функций в определенных средах.

Например, вы можете включить некоторую функцию в тестовой среде, оставив ее отключенной в производственной среде до тех пор, пока она не будет полностью протестирована и готова для широкого использования.

В качестве примера рассмотрим приложение для электронной торговли с корзиной покупок. Руководство хочет запустить рекламную кампанию, требующую внесения в код некоторых изменений. Команда разработчиков может создать эту функцию и внедрить ее авансом во все три среды, но включить ее только в средах разработки и тестирования. Когда руководство будет готово проводить рекламные акции, для включения этой функции нужно просто изменить параметры конфигурации с низким уровнем риска, а не делать выпуск новой версии программного обеспечения. Если функция содержит ошибку, ее легко отключить без обновления программного обеспечения.

26.2. Конвейеры

Конвейер CI/CD представляет собой последовательность шагов, называемых этапами. Каждый этап — это по существу сценарий, который выполняет задачи, специфичные для вашего программного проекта.

На базовом уровне конвейер CI/CD выполняет следующие функции.

- Надежная сборка и упаковка программного обеспечения.
- Выполнение ряда автоматических тестов для поиска ошибок конфигурации.
- Развертывание кода в одной или нескольких средах, включая производственную.

На рис. 26.1 показаны этапы простого (но вполне зрелого) конвейера CI/CD.

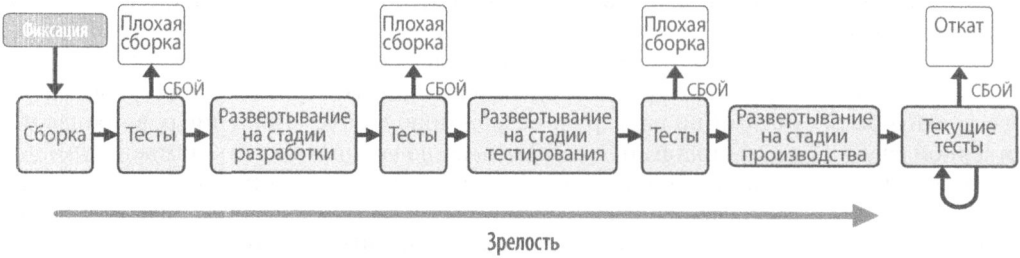


Рис. 26.1. Конвейер CI/CD

В следующих разделах эти три этапа описаны более подробно.

Процесс сборки

Сборка — это моментальный снимок текущего состояния программного проекта. Это, как правило, первый этап любого конвейера CI/CD, возможно, после этапа анализа кода, на котором происходит контролирование качества кода и поиск угроз безопасности. Этап сборки преобразует код в инсталлируемую часть программного обеспечения. Сборка может быть инициирована фиксацией в ветке интеграции репозитория кода либо выполняться по регулярному расписанию или по требованию.

Каждый запуск конвейера начинается со сборки, но не каждая сборка достигает производства. После того как произойдет тестирование, сборка станет "кандидатом на выпуск". Если кандидат на выпуск фактически развертывается для работы в производственной среде, он становится "выпуском". Если вы выполняете непрерывное развертывание, каждый кандидат на выпуск также является выпуском. Эти категории проиллюстрированы на рис. 26.2.



Рис. 26.2. Сборки, кандидаты на выпуск и выпуски

Точное содержание этапов процесса сборки зависят от языка и программного обеспечения. Для программ на языках C, C++ или Go процесс сборки представляет собой компиляцию, часто инициированную командой **`make`**, которая приводит к исполняемому двоичному коду. Для языков, которые не требуют компиляции, таких как Python или Ruby, этап сборки может включать упаковку проекта со всеми соответствующими зависимостями и ресурсами, включая библиотеки, образы, шаблоны и файлы разметки. Некоторые сборки могут включать только изменения конфигурации.

Результатом этапа сборки является "артефакт сборки". Характер этого артефакта зависит от программного обеспечения и конфигурации остальной части конвейера. В табл. 26.1 перечислены некоторые из распространенных типов артефактов. Независимо от формата, артефакт является основой для развертывания в остальной части конвейера.

Таблица 26.1. Распространенные типы артефактов сборки

Тип	Предназначение
Файл `.jar` или `.war`	Архив Java или архив веб-приложений Java
Статический двоичный файл	Статически скомпилированные программы, обычно на языках C или Go
Файл `.rpm` или `.deb`	Пакеты программного обеспечения для операционных систем Red Hat или Debian
Пакет `pip` или пакет `gem`	Упакованные приложения Python или Ruby
Образ контейнера	Приложения, выполняемые на платформе Docker
Образ машины	Виртуальные серверы, особенно для открытых или закрытых облаков
Файл `.exe`	Исполняемый файл Windows

Артефакты сборки сохраняются в хранилище артефактов. Тип репозитория зависит от типа артефакта. В своем простейшем случае репозиторий может быть каталогом на удаленном сервере, доступным через SFTP или NFS. Это также может быть репозиторий `yum` или APT, репозиторий образов Docker или хранилище объектов в облаке, такое как AWS S3. Репозиторий должен быть доступен для всех систем, которые должны загружать и инсталлировать артефакт во время развертывания.

Тестирование

На каждом этапе конвейера CI/CD выполняются тесты, чтобы выявлять ошибочный код и плохие сборки, чтобы код, который дошел до этапа производства, не имел дефектов (или по крайней мере был как можно более надежным). Основой этого процесса является тестирование. Это порождает доверие к тому, что выпуск готов к развертыванию.

Если сборка не проходит какой-либо тест, оставшиеся этапы конвейера не имеют смысла. Команда разработчиков должна определить, почему сборка не удалась, и решить основную проблему. Поскольку сборки создаются для каждой фиксации кода, легко изолировать проблему до последней фиксации. Чем меньше строк кода изменяется между сборками, тем легче изолировать проблему.

Неудачи не всегда связаны с ошибками программного обеспечения. Они могут возникать из-за проблем с сетью или ошибок инфраструктуры, требующих административного внимания. Если приложение зависит от внешних ресурсов, таких как сторонние API, сбои могут возникать во внешнем ресурсе. Одни тесты могут выполняться изолированно, но для других тестов требуется та же инфраструктура и данные, которые будут присутствовать в производственной версии.

Рассмотрите возможность добавления каждого из следующих типов тестов в конвейер CI/CD.

- *Анализ статического кода* позволяет выявить синтаксические ошибки, дублирования, нарушения правил кодирования, проблемы с безопасностью или чрезмерной сложностью. Эти проверки выполняются быстро и не требуют выполнения фактического кода.
- *Модульные тесты*, написанные теми же разработчиками, которые пишут код приложения, отражают представление разработчика о том, как должен функционировать

код. Цель состоит в проверке ввода и вывода каждого метода и функции (модуля) в коде. "Покрытие кода" — это (иногда вводящий в заблуждение) показатель, который описывает, какая часть кода подвергается модульному тестированию.[1]

- *Тесты интеграции* — это модульные тесты, которые выполняются после запуска приложения в его предполагаемой среде исполнения. Эти тесты запускают приложение с его базовыми каркасами и внешними зависимостями, такими как внешние API, базы данных, очереди и кеши.
- *Приемочные тесты* отражают точку зрения пользователя. Для веб-программ этот этап может включать дистанционное управление загрузкой страниц браузера с помощью таких инструментов, как Selenium. Для мобильного программного обеспечения артефакт сборки может поступать на ферму устройств, которая запускает приложение на разных мобильных устройствах. Различные браузеры и версии делают приемочные испытания сложными для создания, но в конце концов эти тесты имеют значимые результаты.
- *Тесты производительности* предназначены для поиска проблем производительности, связанных с последним кодом. Чтобы выявить узкие места, эти так называемые стресс-тесты должны выполнить приложение в идеальном клоне вашей производственной среды с реальным поведением трафика. Такие инструменты, как JMeter или Gatling, могут моделировать тысячи одновременных пользователей, взаимодействующих с приложением в запрограммированном шаблоне. Чтобы получить максимальную отдачу от тестирования производительности, убедитесь, что у вас установлены контрольные и графические средства. Эти инструменты проясняют как типичную производительность приложения, так и его поведение в новой сборке.
- *Инфраструктурные тесты* идут рука об руку с программной облачной инфраструктурой. Если вы создаете временную облачную инфраструктуру как часть конвейера CI/CD, вы можете написать тестовые примеры, чтобы проверить правильную конфигурацию и работу самой инфраструктуры. Успешно ли работает система управления конфигурацией? Выполняются ли только ожидаемые демоны? Один из интересных инструментов в этой области — Serverspec (`serverspec.org`).

🕮 Дополнительную информацию о системах мониторинга и графических системах см. в главе 28.

В зависимости от характеристик вашего проекта некоторые тесты могут оказаться важнее других. Например, программное обеспечение, которое реализует интерфейс REST API, не нуждается в приемочных тестах с применением браузера. Вместо этого вы, скорее всего, сосредоточитесь на тестах интеграции. С другой стороны, для программного обеспечения интернет-магазина обязательно требуется проверка в браузере всех важных пользовательских маршрутов (каталог, страницы продукта, корзина, оформление заказа). Рассмотрите потребности своего проекта и соответствующим образом выполните тестирование.

Рабочие процессы не обязательно должны быть линейными. На самом деле, поскольку одна из целей заключается в том, чтобы как можно быстрее получить обратную связь, рекомендуется проводить как можно больше тестов параллельно. Но имейте в виду, что одни тесты могут зависеть от результатов других тестов; некоторые могут потенциально мешать друг другу. (В идеале тесты не должны иметь перекрестных зависимостей.)

[1] Код, который трудно тестировать, скорее всего, имеет дефекты. У вашего кода может быть 85% покрытия кода (что довольно много по отраслевым стандартам), но если самый сложный код не протестирован, ошибки могут быть пропущены. Одно лишь покрытие кода не является адекватной мерой качества кода.

Избегайте соблазна игнорировать или недооценивать неудачные тесты. Иногда возникает привычка снисходительно относиться к некоторым причинам сбоя, считая их безвредными или неважными, и подавлять тест. Однако такой образ мыслей опасен и может привести к менее надежной системе тестирования. Имейте в виду золотое правило CI/CD: исправление неисправного конвейера является главным приоритетом.

Чтобы укрепить этот принцип, сделайте почти невозможным игнорирование неудачных тестов. Обязательно установите техническое требование с помощью программного обеспечения CI/CD: развертывание в производственной среде не может произойти, если есть какие-либо неудачные тесты.

Развертывание

Развертывание — это процесс установки программного обеспечения и подготовки его для использования в среде сервера. Специфика того, как это делается, зависит от набора технологий. Система развертывания должна понимать, как извлечь артефакт сборки (например, из репозитория пакетов или реестра образов контейнеров), как установить его на сервере и какие шаги настройки необходимы, если таковые имеются. Развертывание завершается, когда новая версия программного обеспечения запущена, а старая версия отключена.

Развертывание может быть очень простым, как, например, обновление некоторых файлов HTML на диске. При этом не требуется перезагрузка сервера или его дальнейшая настройка! Однако для большинства случаев развертывание включает по крайней мере установку пакета и перезапуск приложения. Комплексное развертывание крупномасштабного приложения в производственной среде может включать в себя установку кода на нескольких серверах при одновременном обработке трафика в реальном времени без приостановки обслуживания.

Системные администраторы играют важную роль в процессе развертывания. Обычно они отвечают за создание сценариев развертывания, мониторинг важных индикаторов работоспособности приложений во время развертывания и обеспечение соответствия потребностей инфраструктуры и конфигурации другим членам команды.

В следующем списке описано несколько возможных способов развертывания программного обеспечения.

- Запустите базовый сценарий оболочки, который вызывает программу **ssh** для входа в систему, загрузки и установки артефакта сборки, а затем перезапускает приложение. Эти типы сценариев обычно разрабатываются самостоятельно и не выходят за рамки небольшого количества систем.
- Используйте инструмент управления конфигурацией для организации процесса инсталляции с помощью управляемого набора систем. Эта стратегия более организована и масштабируема, чем использование сценариев оболочки. Большинство систем управления конфигурацией не предназначены специально для облегчения развертывания, хотя они могут использоваться для этой цели.
- Если артефакт сборки представляет собой образ контейнера, и приложение запускается на платформе управления контейнером, такой как Kubernetes, Docker Swarm или AWS ECS, развертывание может быть не более чем быстрым вызовом API для диспетчера контейнеров. Служба контейнера самостоятельно управляет остальной частью процесса развертывания (см. раздел "Контейнеры и подход CI/CD").
- Существует несколько проектов с открытым исходным кодом, которые позволяют стандартизировать и упростить развертывание. Capistrano (`capistranorb.com`) —

это инструмент развертывания на основе языка Ruby, который расширяет систему Rake Ruby для запуска команд на удаленных системах. Fabric (`fabfile.org`) — это аналогичный инструмент, написанный на языке Python. Эти инструменты, предназначенные для разработчиков, в основном служат для создания сценариев оболочки.

- Развертывание программного обеспечения — это хорошо изученная проблема для пользователей публичных облаков. Большинство облачных экосистем включают в себя как интегрированные, так и сторонние службы развертывания, которые могут использоваться из конвейера CI/CD, например Google Deployment Manager, AWS CodeDeploy и Heroku.

Примените технологию развертывания к арсеналу технологий вашей организации с учетом ее потребностей. Если у вас простая среда с несколькими серверами и небольшое количество приложений, то может быть уместным использование инструмента управления конфигурацией. В организациях с большим количеством серверов, распределенных между центрами обработки данных, требуется специализированное средство развертывания.

Термин *неизменяемое развертывание* (immutable deployment) означает принцип, согласно которому серверы никогда не должны изменяться после их инициализации. Чтобы развернуть новую версию, инструментарий CI/CD создает совершенно новые серверы с обновленным артефактом сборки, включенным в образ. В этой модели серверы считаются одноразовыми и временными. Эта стратегия основана на программируемой инфраструктуре, такой как открытое или закрытое облако, где экземпляры могут быть распределены через вызов интерфейса API. Некоторые из крупнейших пользователей открытого облака применяют неизменяемые варианты развертывания.

В разделе "Подход CI/CD на практике" мы рассмотрим пример неизменяемого развертывания, в котором используется инструмент Terraform от компании HashiCorp для создания и обновления инфраструктуры.

Методы развертывания с нулевым временем простоя

В некоторых организациях службы должны продолжать работать даже во время их обновления или повторного развертывания либо потому, что из-за сбоя возникает неприемлемый риск (в области здравоохранения или государственных услуг) или это может привести к значительным финансовым затратам (в области крупной электронной торговли или финансовых услуг). Обновление программного обеспечения в реальном времени без прерывания обслуживания — это высший пилотаж в сфере развертывания программного обеспечения и одновременно источник большого беспокойства и трудностей.

Одним из распространенных способов достижения выпуска с нулевым временем простоя является "сине-зеленое" развертывание. Концепция проста: запустите новое программное обеспечение в резервной системе (или наборе систем), выполните тесты, чтобы подтвердить его функциональность, а затем переключите трафик из реальной системы в резервную после завершения тестов.

Эта стратегия работает особенно хорошо, когда трафик проксируется через балансировщик нагрузки. Реальные системы обрабатывают все пользовательские соединения, пока готовятся резервные системы. В подходящий момент можно добавить в балансировщик нагрузки резервные системы и удалить предыдущие системы. Развертывание завершено, когда все старые системы находятся вне ротации и все транзакции, которые они обрабатывают, завершились.

🕮 Дополнительную информацию о балансировщиках нагрузки см. в разделе 19.2.

Плавное развертывание (rolling deployment) обновляет существующие системы поэтапно, изменяя программное обеспечение в одной системе за один раз. Каждая система удаляется из балансировщика нагрузки, обновляется, а затем добавляется обратно в ротацию для обработки пользовательского трафика. Такой тип развертывания может вызвать проблемы, если приложение не допускает существования двух разных версий одновременно.

Стратегии сине-зеленого и плавного развертывания можно совместить с подходом, который называется "канареечным", по аналогии с канарейкой в угольной шахте. Сначала вы направляете небольшой объем трафика в одну систему (или небольшой процент систем), которая запускает новую версию. Если у новой версии есть проблемы, вы ее отключаете и исправляете проблему, оказывая влияние только на нескольких пользователей. Конечно, канареечным системам нужна точная телеметрия и мониторинг, чтобы вы могли вовремя определить, не появились ли новые проблемы.

26.3. Jenkins: сервер автоматизации с открытым исходным кодом

Jenkins — это сервер автоматизации, написанный на языке Java. Это, безусловно, самое популярное программное обеспечение, используемое для реализации подхода CI/CD. Благодаря широкому внедрению и обширной экосистеме подключаемых модулей Jenkins хорошо подходит для различных вариантов использования.

Сервер Jenkins несложно запустить в контейнере Docker:

```
$ docker run -p 8080:8080 --name jenkins jenkinsci/jenkins
```

После запуска контейнера вы можете получить доступ к пользовательскому интерфейсу Jenkins в веб-браузере через порт 8080. Начальный пароль администратора будет указан в сообщениях, выводимых контейнером. На практике вам нужно немедленно изменить этот пароль!

Для изучения основ подойдет конфигурация с одним контейнером, но в производственных средах, скорее всего, потребуется более надежное решение. На странице загрузки сервера Jenkins (`jenkins.io/download`) есть инструкции по инсталляции, которые мы не будем здесь повторять. Обратитесь к этим документам для инсталляции сервера в операционных системах Linux и FreeBSD. Компания CloudBees, создатель сервера Jenkins, также предлагает версию с высокой доступностью под названием Jenkins Enterprise.

У сервера Jenkins есть подключаемые модули для каждой мыслимой задачи. Используйте подключаемые модули для выполнения сборок в разных типах агентов, отправки уведомлений, координации выпусков и выполнения запланированных заданий. Подключаемые модули взаимодействуют с инструментами с открытым исходным кодом и со всеми основными облачными платформами и внешними поставщиками SaaS. Именно подключаемые модули обеспечивают сверхспособности серверу Jenkins.

Большинство настроек Jenkins выполняются через веб-интерфейс, и, проявляя милосердие к вашему вниманию, мы не пытаемся описать все нюансы пользовательского интерфейса. Вместо этого мы представляем основы работы сервера Jenkins и некоторые из его наиболее важных особенностей.

Основные концепции сервера Jenkins

По сути, Jenkins — это координирующий сервер, который связывает ряд инструментов в цепочку, или, используя терминологию CI/CD, конвейер. Сервер Jenkins является организатором и посредником; вся фактическая работа зависит от внешних служб, та-

ких как репозитории исходного кода, компиляторы, инструменты сборки, средств тестирования и систем развертывания.

Задание (job) для сервера Jenkins, или проект, — это коллекция связанных этапов. Создание проекта — это задача первостепенной важности для новой инсталляции. Вы можете связать этапы проекта, чтобы они выполнялись последовательно или параллельно. Вы можете даже настроить условные этапы, которые делают разные вещи в зависимости от результатов предыдущих этапов.

Каждый проект должен быть подключен к репозиторию исходного кода. У сервера Jenkins есть собственная поддержка почти каждой системы контроля версий: Git, Subversion, Mercurial и даже таких древних систем, как CVS. Существуют также подключаемые модули интеграции для более высокоуровневых служб контроля версий, таких как GitHub, GitLab и BitBucket. Вам нужно будет предоставить серверу Jenkins соответствующие учетные данные, чтобы он мог загружать код из вашего репозитория.

"Контекст сборки" (build context) — это текущий рабочий каталог системы Jenkins, в котором выполняется сборка. Исходный код копируется в контекст сборки вместе с любыми поддерживающими файлами, которые необходимы для сборки.

Подключив сервер Jenkins к репозиторию контроля версий, вы сможете создать триггер сборки. Это сигнал для сервера Jenkins скопировать текущий исходный код и начать процесс сборки. Сервер Jenkins может опросить исходный репозиторий в поисках новых фиксаций, и когда он ее найдет — инициировать сборку. Он также может запускать сборку по расписанию или запускаться с помощью веб-триггера, который поддерживается репозиторием GitHub.

После настройки триггера создайте этапы сборки, т.е. конкретные задачи, которые будут создавать сборку. Этапы могут быть специфичными для кода или универсальными сценариями оболочки. Например, проекты Java обычно создаются с помощью инструмента Maven. Подключаемый модуль Jenkins поддерживает Maven напрямую, поэтому вы можете просто добавить этап сборки Maven. Для проекта, написанного на языке C, первый этап сборки может быть просто сценарием оболочки, который запускает команду **make**.

Остальные этапы сборки зависят от целей вашего проекта. Наиболее распространенные сборки включают этапы, которые инициируют задачи тестирования, обсуждаемые выше в разделе "Тестирование". Вам может понадобиться этап для создания произвольного артефакта сборки, такого как архив tar, пакет операционной системы или образ контейнера. Вы также можете включить этапы, которые инициируют уведомления администратора, предпринимают действия, связанные с развертыванием, или координируются с помощью внешних инструментов.

Для проекта CI/CD этапы сборки могут охватывать все этапы конвейера: создавать код, запускать тесты, загружать артефакт в репозиторий и запускать развертывание.

Каждый этап конвейера является всего лишь этапом сборки в рамках проекта Jenkins. Интерфейс Jenkins представляет собой обзор состояния каждого этапа, поэтому легко увидеть, что происходит в конвейере.

Организации, в которых есть много приложений, должны иметь отдельные проекты Jenkins для каждого приложения. Каждый проект будет иметь отдельный репозиторий кода и этапы сборки. Система Jenkins требует для запуска сборки в любом из своих проектов наличия всех инструментов и зависимостей. Например, если вы настроили проект на языке Java и проект на языке C, ваша система Jenkins должна иметь доступ как к инструменту Maven, так и к команде **make**.

Проекты могут зависеть от других проектов. Используйте это в своих интересах, структурируя проекты как общие, наследуемые шаблоны. Например, если у вас есть множество приложений, которые построены по-разному, но развернуты одинаково (например, как

контейнеры, запущенные на кластере серверов), вы можете создать общий проект "развертывания", который управляет общим этапом развертывания. Отдельные проекты приложений могут выполнять проект развертывания, тем самым устраняя шаг избыточной сборки.

Распределенные сборки

В тех средах, где поддерживаются десятки приложений, причем каждое со своими зависимостями и этапами сборки, легко непреднамеренно создать конфликты зависимостей и узкие места, поскольку сразу запускается слишком много конвейеров. Чтобы компенсировать это, сервер Jenkins позволяет вам перейти в распределенную архитектуру сборки. В этом режиме работы используется "мастер сборки", центральная система, которая отслеживает все проекты и их текущее состояние, а также "агенты сборки", которые выполняют фактические этапы сборки для проекта. Если вы часто используете сервер Jenkins, то довольно быстро перейдете к этой конфигурации.

Агенты сборки запускаются на хостах, которые отделены от мастера сборки. Мастер сервера Jenkins подключается к подчиненным системам (обычно через протокол SSH), чтобы запустить процесс агента и добавить метки, которые документируют возможности подчиненных систем. Например, вы можете отличить своих агентов, поддерживающих язык Java, от ваших же агентов с поддержкой языка C, применяя соответствующие метки. Для достижения наилучших результатов запускайте агенты в контейнерах, удаленных виртуальных машинах или временных облачных средах, которые масштабируются и возвращаются в исходное состояние по требованию. Если у вас есть кластер контейнеров, вы можете использовать подключаемые модули Jenkins для запуска агентов в кластере через систему управления контейнерами.

Конвейер как код

До сих пор мы описали процесс создания проектов Jenkins, объединяя отдельные этапы сборки в веб-интерфейсе. Это самый быстрый способ начать работу с сервером Jenkins, но с точки зрения инфраструктуры это немного непрозрачно. Код — в данном контексте им является содержимое каждого этапа сборки — управляется сервером Jenkins. Вы не можете проверять этапы сборки в репозитории кода с помощью графического инструмента, и если вы потеряете сервер Jenkins, у вас не будет простого способа его заменить; вам нужно будет восстановить свои проекты из недавней резервной копии.

Во второй версии сервера Jenkins введена новая основная функция, называемая Pipeline, которая обеспечивает первоклассную поддержку конвейеров CI/CD. В конвейере Jenkins этапы проекта кодируются в декларативном стиле, специфичном для предметно-ориентированного языка, который основан на языке программирования Groovy. Вы можете передать код Jenkins, называемый **`Jenkinsfile`**, вместе с кодом, связанным с конвейером.

Следующий код **`Jenkinsfile`** демонстрирует базовый цикл сборки — тестирования — развертывания.

```
pipeline {
   agent any
   stages {
      stage('Build') {
          steps {
             sh 'make'
          }
      }
      stage('Test') {
         steps {
```

```
                sh 'make test'
            }
        }
        stage('Deploy') {
            steps {
                sh 'deploy.sh'
            }
        }
    }
}
```

Выражение `agent any` инструктирует сервер Jenkins подготовить рабочую область для этого конвейера на любом доступном агенте сборки.[2] Этапы сборки, тестирования и развертывания соответствуют концептуальным этапам конвейера CI/CD. В нашем примере каждый этап имеет один шаг, который вызывает оболочку (**sh**) для выполнения команды.

На этапе развертывания выполняется собственный сценарий **`deploy.sh`**, который обрабатывает все развертывание, включая копирование артефакта сборки (сгенерированного на этапе `Build`) на набор серверов и перезапуск серверных процессов. На практике развертывание обычно делится на несколько этапов, чтобы обеспечить лучшую видимость и контроль над полным процессом.

26.4. Подход CI/CD на практике

Теперь мы перейдем к искусственному примеру, чтобы проиллюстрировать представленные нами концепции. Мы придумали простое приложение UlsahGo, намного более основательное, чем все, что вам может понадобиться для управления в реальном мире. Оно полностью автономное и не имеет зависимостей от других приложений.

Наш пример включает следующие элементы:

- веб-приложение UlsahGo с одной небольшой функцией;
- модульные тесты для приложения;
- образ виртуальной машины для облачной инфраструктуры DigitalOcean, которая содержит приложение;
- односерверную среду разработки, создаваемую по требованию;
- многосерверную среду с балансировкой нагрузки, создаваемую по требованию;
- конвейер CI/CD, который соединяет все эти части вместе.

В этом примере мы используем несколько популярных инструментов и служб:

- GitHub как репозиторий кода;
- виртуальные машины DigitalOcean и балансировщики нагрузки;
- упаковщик HashiCorp для обеспечения образа для облачной инфраструктуры DigitalOcean;
- инструмент Terraform от HashiCorp для создания среды развертывания;
- сервер Jenkins для управления конвейером CI/CD.

В ваших приложениях может использоваться другой набор технологий, но общие понятия похожи, независимо от арсенала используемых инструментов.

[2] Рабочая область — это синоним контекста сборки: место на локальном диске агента, в котором содержатся все файлы, необходимые для сборки, включая исходный код и зависимости. Каждая сборка имеет частное рабочее пространство.

На рис. 26.1 показаны первые несколько этапов примерного конвейера. На диаграмме показан опрос конвейеров GitHub в поисках новых фиксаций в проект UlsahGo. Когда фиксация найдена, сервер Jenkins запускает блок тестирования. Если тесты проходят, сервер Jenkins строит двоичный файл. Если двоичная сборка успешно завершена, конвейер продолжает создавать артефакт развертывания — образ машины DigitalOcean, который включает двоичный файл. Если какой-либо из этапов не работает, конвейер сообщает об ошибке.

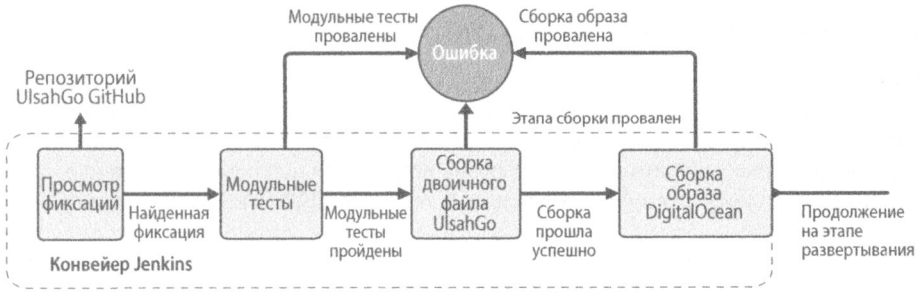


Рис. 26.3. Демонстрационный конвейер (часть первая)

Подробное описание этапов развертывания приводится ниже, но сначала мы должны рассмотреть начальные этапы.

Тривиальное веб-приложение UlsahGo

Приложение из нашего примера представляет собой веб-службу с одной функцией. Оно возвращает список, авторов, связанных с указанным изданием этой книги, в формате JSON. Например, следующий запрос отображает имена авторов указанного издания.

```
$ curl ulsahgo.admin.com/?edition=5
{
   "authors": [
      "Evi",
      "Garth",
      "Trent",
      "Ben",
      "Dan"
   ],
   "number": 5
}
```

При обработке этого запроса мы должны выполнить простую проверку корректности полученных данных, чтобы убедиться, что пользователи не слишком увлекаются, например запрашивая неправдоподобные номера изданий.

```
$ curl -vs ulsahgo.admin.com/?edition=6
< HTTP/1.1 404 Not Found
< Content-Type: application/json
{
   "error": "6th edition is invalid."
}
```

Наше приложение имеет также специальную точку входа для проверки работоспособности. Проверка работоспособности — это простой способ для мониторинга систем, чтобы спросить приложение: "Привет, все хорошо?"

```
$ curl ulsahgo.admin.com/healthy
{
```

```
    "healthy": "true"
}
```

Разработчики обычно тесно сотрудничают с администраторами в процессе создания этапов сборки и тестирования конвейера CI/CD. В данном случае, поскольку приложение написано на языке Go, мы можем использовать стандартные инструменты Go (**`go build`** и **`go test`**) в нашем конвейере.

Модульное тестирование UlsahGo

Модульные тесты — это первый тестовый набор для запуска, поскольку они работают на уровне исходного кода. Модульные тесты проверяют функции и методы приложения с максимально возможной детализацией. Для большинства языков программирования созданы тестовые каркасы, в которых реализована встроенная поддержка модульных тестов.

Давайте проанализируем один модульный тест для UlsahGo. Рассмотрим следующую функцию.

```
func ordinal(n int) string {
    suffix := "th"
    switch n {
    case 1:
        suffix = "st"
    case 2:
        suffix = "nd"
    case 3:
        suffix = "rd"
    }
    return strconv.Itoa(n) + suffix
}
```

Этой функции в качестве входного параметра передается целое число, а она определяет соответствующее порядковое выражение. Например, если функции передан аргумент, равный `1`, то она возвращает строку "`1st`". Программа UlsahGo использует эту функцию для форматирования текста в сообщении об ошибке для недопустимых номеров изданий книг.

Модульные тесты пытаются доказать, что при произвольных входных данных функция всегда возвращает ожидаемый результат. Вот модульный тест, который выполняет эту функцию.

```
func TestOrdinal(t *testing.T) {
    ord := ordinal(1)
    exp := "1st"
    if ord != exp {
        t.Error("expected %s, got %s", exp, ord)
    }
    ord = ordinal(10)
    exp = "10th"
    if ord != exp {
        t.Error("expected %s, got %s", exp, ord)
    }
}
```

Этот модульный тест запускает функцию для двух значений — 1 и 10 — и подтверждает, что фактический ответ соответствует ожиданию.[3] Мы можем запускать тесты через встроенную систему тестирования Go.

[3] В функции **`ordinal()`** реализовано три особых случая и один общий. При запуске полного набора модульных тестов должна быть выполнена каждая из возможных веток кода.

```
$ go test
PASS
ok github.com/bwhaley/ulsahgo      0.006s
```

Если какая-то часть приложения в будущем изменится — например, если в функцию `ordinal()` будут добавлены обновления — тесты сообщат о любом расхождении с ожидаемым выходным значением. За обновление модульных тестов отвечают разработчики, поскольку именно они вносят изменения в код. Опытные разработчики сразу пишут код, который легко тестировать. Они нацелены на полное покрытие тестами каждого метода и функции.

Знакомство с конвейером Jenkins Pipeline

Подготовив к поставке код и модульные тесты, сделайте первый шаг на пути освоения CI/CD — настройте проект на сервере Jenkins. Через процесс вас проведет графический пользовательский интерфейс. Ниже приводятся варианты, которые мы выбрали.

- Наш новый проект называется Pipeline. Он определен кодом, а не традиционным проектом в свободном стиле с этапами сборки, которые в основном определяются элементами пользовательского интерфейса.
- Мы хотим отслеживать наш конвейер вместе с репозиторием исходного кода в файле **`Jenkinsfile`**, поэтому мы выбираем для определения Pipeline пункт Pipeline script from SCM.
- Мы запускаем сборку путем опроса GitHub в поисках фиксации. Мы добавляем учетные данные, чтобы сервер Jenkins мог получить доступ к репозиторию UlsahGo и настроиться на опрос GitHub в поисках изменений каждые пять минут.

Начальная настройка занимает всего несколько секунд. В реальной жизни мы использовали бы веб-триггер GitHub, чтобы уведомить сервер Jenkins о том, что новая фиксация доступна, и избежать опроса, тем самым избавляя GitHub от ненужных обращений к его интерфейсу API.

С помощью этой настройки сервер Jenkins выполняет конвейер, описанный в файле **`Jenkinsfile`** в репозитории всякий раз, когда в GitHub появляется новая фиксация.

Теперь давайте рассмотрим структуру репозитория. В нашем проекте мы решили объединить CI/CD и код приложения в одном репозитории, при этом все файлы, связанные с CI/CD, хранятся в подкаталоге **`pipeline`**. Репозиторий UlsahGo представлен следующим образом.

```
$ tree ulsahgo
ulsahgo
├── pipeline
│   ├── Jenkinsfile
│   ├── packer
│   │   ├── provisioner.sh
│   │   ├── ulsahgo.json
│   │   └── ulsahgo.service
│   ├── production
│   │   ├── tf_prod.sh
│   │   └── ulsahgo.tf
│   └── testing
│       ├── tf_testing.sh
│       └── ulsahgo.tf
├── ulsahgo.go
└── ulsahgo_test.go
```

Интегрированная структура хорошо работает для небольшого проекта, подобного этому. Jenkins, Packer, Terraform и другие инструменты могут искать свои файлы конфигурации в подкаталоге конвейера. Изменение конвейера развертывания выполняется путем простого обновления репозитория. Для более сложных сред, в которых несколько проектов имеют общую инфраструктуру, рекомендуем использовать выделенный репозиторий инфраструктуры.

После настройки инфраструктуры проекта можно выполнить фиксацию в репозитории нашего первого файла **Jenkinsfile**. Первым шагом в любом конвейере является получение исходного кода. Вот полный сценарий конвейера **Jenkinsfile**, который делает именно это.

```
pipeline {
   agent any
   stages {
      stage('Checkout') {
         steps {
            checkout scm
         }
      }
   }
}
```

Строка `checkout scm` дает указание серверу Jenkins получить исходный код из системы управления конфигурацией программного обеспечения (это общий отраслевой термин для систем контроля версий).

После опроса репозитория GitHub сервером Jenkins и завершения этапа получения исходного кода мы можем перейти к настройке этапов тестирования и сборки. Наш проект Go не имеет внешних зависимостей. Единственным требованием для создания и тестирования нашего кода является двоичный исполняемый файл **go**. Мы уже установили систему Jenkins (с помощью команды **apt-get -y install golang-go**), поэтому нам нужно только добавить этапы тестирования и сборки в файл **Jenkinsfilse**:

```
stage('Unit tests') {
   steps {
      sh 'go test'
   }
}
stage('Build') {
   steps {
      sh 'go build'
   }
}
```

После того как мы зафиксируем изменения в хранилище, система Jenkins обнаружит новую фиксацию и запустит конвейер. Затем система Jenkins генерирует читабельную запись журнала, подтверждая, что она сделала это.

```
Mar 30, 2017 4:35:00 PM hudson.triggers.SCMTrigger$Runner run
INFO: SCM changes detected in UlsahGo. Triggering #4
Mar 30, 2017 4:35:11 PM org.jenkinsci.plugins.workflow.job.WorkflowRun
   finish
INFO: UlsahGo #4 completed: SUCCESS
```

В графическом интерфейсе Jenkins используется метеорологическая метафора, указывающая на работоспособность последних сборок. Значок солнца обозначает проект, который собран успешно, а значок пасмурного облака обозначает сбои. Вы можете

устранять сбои сборки, проверяя консольный вывод, который находится в деталях сборки. Он представляет собой содержание потока STDOUT, которое выводится на экран любой частью сборки.

Ниже приведен фрагмент результатов вывода команд **go test** и **go build** в ходе работы конвейера.

```
[Pipeline] stage
[Pipeline] { (Unit Tests)
[Pipeline] sh
[UlsahGo] Running shell script
+ go test
PASS
ok      _/var/jenkins_home/workspace/UlsahGo        0.006s
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Build)
[Pipeline] sh
[UlsahGo] Running shell script
+ go build
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Обычно определить причину неудачной сборки можно, просмотрев журнал. Ищите сообщения об ошибках, которые идентифицируют неудавшийся шаг. Вы также можете добавить собственные сообщения в журнал, чтобы предоставлять подсказки о состоянии системы, такие как значения переменных или содержимое сценария в данной точке выполнения. Использование отладочной печати — это старая традиция программистов.

Результатом нашей сборки является один двоичный файл **ulsahgo**, который содержит все наше приложение. (Это, кстати, одно из основных преимуществ программ на языке Go, и одна из причин, почему язык Go популярен у системных администраторов: с его помощью легко создавать статические двоичные файлы, которые выполняются в нескольких архитектурах и не имеют внешних зависимостей. Установка приложения на языке Go часто бывает простой и сводится к его копированию в нужный каталог системы.)

Создание образа DigitalOcean

Теперь, когда файл **ulsahgo** готов к поставке, мы создадим образ виртуальной машины для облака DigitalOcean. Мы начинаем с простого образа Ubuntu 16.04, устанавливаем последние обновления, а затем устанавливаем программу **ulsahgo**. Полученный образ становится артефактом развертывания для остальных этапов конвейера.

> 🕮 Если вы не знакомы с упаковщиком **packer**, который создает образ виртуальной машины, перед продолжением обратитесь к разделу 24.6.

Упаковщик **packer** считывает конфигурацию своего образа из шаблона, который имеет два основных раздела: 1) сборщики, которые взаимодействуют с удаленными интерфейсами API для создания машин и образов, и 2) вспомогательные устройства, которые запускают настраиваемые этапы настройки.

В шаблоне для нашего образа UlsahGo указан только один сборщик.

```
"builders": [{
   "type": "digitalocean",
   "api_token": "rj8FsrMI17vqTlB8qqBn9f7xQedJkkZJ7cqJcB1O5nmO6ihz",
   "region": "sfo2",
   "size": "512mb",
   "image": "ubuntu-16-04-x64",
   "snapshot_name": "ulsahgo-latest",
   "ssh_username": "root"
}]
```

Помимо других деталей, относящихся к конкретному провайдеру, сборщик сообщает упаковщику **packer**, какая платформа используется для создания образа и как аутентифицироваться в интерфейсе API. Для этого он выполняет следующие три подготовительных этапа.

```
"provisioners": [
   {
      "type": "file",
      "source": "ulsahgo",
      "destination": "/tmp/ulsahgo"
   },{
      "type": "file",
      "source": "pipeline/packer/ulsahgo.service",
      "destination": "/etc/systemd/system/ulsahgo.service"
   },{
      "type": "shell",
      "script": "pipeline/packer/provisioner.sh"
   }
]
```

🕮 Дополнительную информацию о модульных файлах **systemd** см. в разделе 2.7.

Первые два подготовительных этапа инициализации добавляют файлы к образу. Первый файл — это само приложение **ulsahgo**, которое загружается в каталог **/tmp** для последующего использования. Второй файл — это подключаемый файл **systemd**, который управляет службой.

На последнем подготовительном этапе запускается собственный сценарий оболочки в удаленной системе. Сценарий **provisioner.sh** обновляет систему, а затем настраивает приложение.

```
#!/usr/bin/env bash
app=ulsahgo

# Обновим ОС и добавим учетную запись пользователя
apt-get update && apt-get -y upgrade
/usr/sbin/useradd -s /usr/sbin/nologin $app

# Создадим рабочий каталог и скопируем в него приложение
mkdir /opt/$app && chown $app /opt/$app
cp /tmp/$app /opt/$app/$app
chown $app /opt/$app/$app && chmod 700 /opt/$app/$app

# Разрешим работу модуля systemd
systemctl enable $app
```

Кроме сценариев оболочки, утилита **packer** позволяет использовать на подготовительных этапах все популярные инструменты управления конфигурацией. Она может вызывать Puppet, Chef, Ansible или Salt, чтобы настроить ваши образы более структурированным и масштабируемым способом.

Наконец, мы можем добавить этап создания образа в наш файл **Jenkinsfile**.

```
stage('Build image') {
   steps {
      sh 'packer build pipeline/packer/ulsahgo.json > packer.txt'
      sh 'grep ID: packer.txt | grep -E -o \'[0-9]{8}\' > do_image.txt'
   }
}
```

На первом шаге вызывается программа **packer** и сохраняется результат ее вывода в файле **packer.txt** в рабочем каталоге сборки. В конце этого вывода содержится идентификатор нового образа.

```
==> digitalocean: Gracefully shutting down droplet...
==> digitalocean: Creating snapshot: ulsahgo-latest
==> digitalocean: Waiting for snapshot to complete...
==> digitalocean: Destroying droplet...
==> digitalocean: Deleting temporary ssh key...
Build 'digitalocean' finished.
==> Builds finished. The artifacts of successful builds are:
--> digitalocean: A snapshot was created: (ID: 23838540)
```

На втором шаге выполняется команда **grep**, с помощью которой извлекается идентификатор образа из последней строки файла **packer.txt** и сохраняется в новом файле, находящемся в контексте сборки. Поскольку образ является артефактом развертывания, на более поздних этапах конвейера мы должны ссылаться на него с помощью его идентификатора.

Обеспечение единой системы тестирования

Итак, у нас есть процесс непрерывного выполнения модульных тестов, создания приложения и образа виртуальной машины в качестве артефакта сборки. Остальные этапы сборки сосредоточены на этапе развертывания артефакта и его тестирования в реальных условиях. На рис. 26.4 представлено продолжение рис. 26.3.

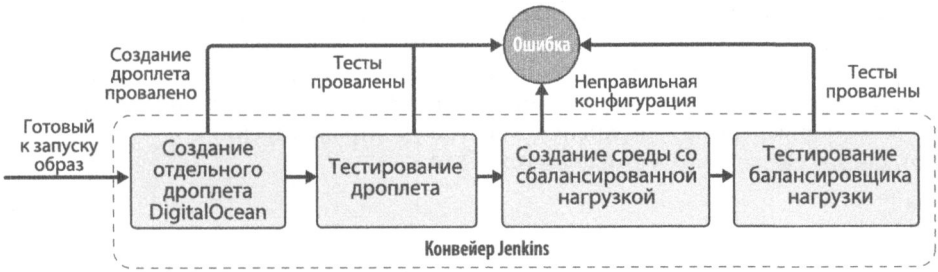


Рис. 26.4. Демонстрационный конвейер (часть вторая)

Для создания и управления инфраструктурой UlsahGo мы решили использовать утилиту **terraform**, еще один инструмент компании HashiCorp. Утилита **terraform** считывает свою конфигурацию из так называемых "планов" JSON-подобных файлов конфигурации, в которых описывается желаемая конфигурация инфраструктуры. Затем она создает облачные ресурсы, описанные в плане, путем создания соответствующей серии вызовов API. Утилита **terraform** поддерживает десятки облачных провайдеров и широкий спектр служб.

Следующая конфигурация **terraform**, заданная в файле **ulsahgo.tf**, подготавливает отдельный дроплет DigitalOcean, запускающий образ, который мы создали на предыдущем этапе конвейера.

```
variable "do_token" {}
variable "ssh_fingerprint" {}
variable "do_image" {}

provider "digitalocean" {
   token = "${var.do_token}"
}

resource "digitalocean_droplet" "ulsahgo-latest" {
   image = "${var.do_image}"
   name = "ulsahgo-latest"
   region = "sfo2"
   size = "512mb"
   ssh_keys = ["${var.ssh_fingerprint}"]
}
```

Большинство этих директив самоочевидны: используйте службу DigitalOcean в качестве провайдера, выполните аутентификацию с предоставленным токеном и создайте дроплет в области `sfo2` из указанного идентификатора образа.

В шаблоне Packer, представленном выше, мы непосредственно ввели в конфигурацию упаковщика такие параметры, как токен API. Одна (большая!) проблема с этим подходом заключается в том, что ключ API сохраняется в репозитории исходного кода, даже если он предположительно является секретным. Этот ключ дает доступ к интерфейсу API провайдера облачных вычислений и, следовательно, будет опасен в неправильных руках. Сохранение секретов в системе контроле версий — это антишаблон безопасности по причинам, которые мы более подробно описывали в разделе 7.8.

В этом примере мы вместо этого считываем параметры в виде переменных. Эти три переменные перечислены ниже.

- Токен DigitalOcean API.
- Отпечаток ключа SSH, которому будет разрешен доступ к дроплету.
- Идентификатор образа для новой системы, который мы записали на предыдущем этапе конвейера.

Система Jenkins может хранить такие секреты, как токен API, в своем "хранилище учетных данных" — зашифрованной области, предназначенной именно для таких конфиденциальных данных. Конвейер может считывать значения из хранилища учетных данных и сохранять их в виде переменных среды. Затем значения становятся доступными по всему конвейеру без сохранения в системе контроля версий.

Вот как мы сделали это в файле **`Jenkinsfile`**.

```
pipeline {
   environment {
      DO_TOKEN = credentials('do-token')
      SSH_FINGERPRINT = credentials('ssh-fingerprint')
   }
...
```

Напомним, что мы сохранили идентификатор образа машины DigitalOcean в файле **`do_image.txt`**, который находится в области сборки. Нам нужен этот идентификатор на нашем новом этапе конвейера, который создает фактический дроплет DigitalOcean. Код для нового этапа просто запускает сценарий из репозитория проекта.

```
stage('Create droplet') {
   steps {
      sh 'bash pipeline/testing/tf_testing.sh'
   }
}
```

Намного проще и удобнее хранить код сложных сценариев отдельно от остальной части конвейера, как мы и сделали. Файл **tf_testing.sh** содержит следующие строки.

```
cp do_image.txt pipeline/testing
cd pipeline/testing
terraform apply \
   -var do_image="$(<do_image.txt)" \
   -var do_token="${DO_TOKEN}" \
   -var ssh_fingerprint="${SSH_FINGERPRINT}"
terraform show terraform.tfstate \
   | grep ipv4_address | awk "{print $3}" > ../../do_ip.txt
```

Этот сценарий копирует файл с сохраненным идентификатором образа во временный каталог **pipeline/testing**, а затем запускает утилиту **terraform** из этого каталога. Утилита **terraform** ищет файлы в текущем каталоге с расширением **.tf**, поэтому нам не нужно явно указывать файл плана. (Это тот же самый файл **ulsahgo.tf**, который мы рассматривали выше.)

Необходимо сделать несколько пояснений.

- Переменные среды `DO_TOKEN` и `SSH_FINGERPRINT` доступны для любых команд оболочки в конвейере. Раздел `environment`, показанный выше, может появляться либо на уровне общего конвейера, либо на определенном этапе в зависимости от требуемой области видимости.
- Команда `$(<do_image.txt)` считывает содержимое идентификатора образа DigitalOcean из текстового файла, сохраненного на предыдущем этапе.
- Последняя строка сценария **tf_testing.sh** проверяет дроплет, созданный утилитой **terraform**, получает его IP-адрес и сохраняет адрес в текстовый файл для использования на следующем этапе. Файл **terraform.tfstate** представляет собой снимок состояния системы, сделанный утилитой **terraform**. Именно так утилита **terraform** отслеживает ресурсы.

Как и утилита **packer**, утилита **terraform** отправляет системные сообщения на страницу консоли Jenkins. Вот фрагменты из вывода команды **apply terraform**.

```
[Pipeline] { (Create droplet)
[Pipeline] sh
[UlsahGo] Running shell script
digitalocean_droplet.ulsahgo-latest: Creating...
   disk:                    "" => "<computed>"
   image:                   "" => "23888047"
   ipv4_address:            "" => "<computed>"
   ipv4_address_private:    "" => "<computed>"
   name:                    "" => "ulsahgo-latest"
   region:                  "" => "sfo2"
   resize_disk:             "" => "true"
   size:                    "" => "512mb"
   ssh_keys.#:              "" => "1"
   ssh_keys.0:              "" => "****"
   status:                  "" => "<computed>"
digitalocean_droplet.ulsahgo-latest: Still creating... (10s elapsed)
digitalocean_droplet.ulsahgo-latest: Still creating... (20s elapsed)
digitalocean_droplet.ulsahgo-latest: Still creating... (30s elapsed)
digitalocean_droplet.ulsahgo-latest: Creation complete (ID: 44486631)
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

Когда эта стадия завершается, дроплет включается и запускает приложение UlsahGo.

Тестирование дроплета

Теперь у нас появилась уверенность в том, что код является работоспособным, потому что он прошел этап модульного тестирования. Однако нам также необходимо убедиться, что он успешно работает как часть дроплета DigitalOcean. Тестирование на этом уровне считается формой теста интеграции. Мы хотим, чтобы тесты интеграции выполнялись каждый раз при создании нового образа, поэтому мы добавляем новый этап в файл **Jenkinsfile**.

```
stage('Test and destroy the droplet') {
   steps {
      sh '''#!/bin/bash -l
      curl -D - -v \$(<do_ip.txt):8000/?edition=5 | grep "HTTP/1.1 200"
      curl -D - -v \$(<do_ip.txt):8000/?edition=6 | grep "HTTP/1.1 404"
      terraform destroy -force
      '''
   }
}
```

Иногда тупой и тяжелый предмет является правильным инструментом для работы. Эта пара команд **curl** посылает запрос нашему приложению **ulshago** на удаленный порт дроплета 8000, который используется в программе **ulsahgo** по умолчанию. Мы проверяем, что запрос для пятого издания возвращает HTTP-код 200 (успех) и что запрос для шестого издания возвращает код HTTP 404 (сбой). Мы знаем, что ожидаем этих конкретных кодов состояния только из-за нашего знакомства с приложением.

По завершении тестов мы уничтожаем дроплет, потому что он больше не нужен. Дроплет создается, тестируется и уничтожается при каждом запуске конвейера.

Развертывание приложения UlsahGo на паре дроплетов и балансировщике нагрузки

Конечной задачей конвейера является развертывание приложения в нашей (макетной) производственной среде, которая состоит из двух дроплетов DigitalOcean и балансировщика нагрузки. Еще раз подчеркнем, что Terraform прекрасно справляется с задачей.

Мы можем повторно использовать конфигурацию из файла плана **terraform** с одним дроплетом. Нам все еще нужны одни и те же переменные и ресурс дроплета. На этот раз добавим второй ресурс дроплета.

```
resource "digitalocean_droplet" "ulsahgo-b" {
   name        =        "ulsahgo-b"
   size        =        "512mb"
   image       =        "${var.do_image}"
   ssh_keys    =        ["${var.ssh_fingerprint}"]
   region      =        "sfo2"
}
```

Мы также добавляем ресурс балансировщика нагрузки.

```
resource "digitalocean_loadbalancer" "public"
{
   name = "ulsahgo-lb"
   region = "sfo2"

   forwarding_rule {
      entry_port = 80
      entry_protocol = "http"
      target_port = 8000
      target_protocol = "http"
   }
```

```
  healthcheck {
    port = 8000
    protocol = "http"
    path = "/healthy"
  }

  droplet_ids = [
    "${digitalocean_droplet.ulsahgo-a.id}",
    "${digitalocean_droplet.ulsahgo-b.id}"
  ]
}
```

Балансировщик нагрузки прослушивает порт 80 и передает запросы каждому дроплету на порт 8000, который прослушивает программа **`ulsahgo`**. Мы указываем балансировщику нагрузки, чтобы он использовал конечную точку **`/healthy`** и подтвердил, что все копии службы работают. Балансировщик нагрузки добавляет дроплет в ротацию, если он получает код состояния 200, когда запрашивает эту конечную точку.

Теперь мы можем добавить конфигурацию для производственной среды в качестве нового этапа в конвейере.

```
stage('Create LB') {
  steps {
    sh 'bash pipeline/production/tf_prod.sh'
  }
}
```

Этап балансировки нагрузки более или менее идентичен стадии одиночного экземпляра. Даже внешний сценарий почти такой же, поэтому здесь мы опускаем его содержимое. Мы могли бы легко реорганизовать эти сценарии так, чтобы одна версия обрабатывала обе среды, но на данный момент мы сохранили сценарии отдельно.

Мы также можем добавить этап тестирования, на этот раз работающий с IP-адресом балансировщика нагрузки.

```
stage('Test load balancer') {
  steps {
    sh '''#!/bin/bash -l
      curl -D - -v -s \$(<do_lb_ip.txt)/?edition=5 | grep "HTTP/1.1 200"
      curl -D - -v -s \$(<do_lb_ip.txt)/?edition=6 | grep "HTTP/1.1 404"
    '''}
}
```

Команды **`curl`** аналогичны предыдущему набору команд, но они нацелены на порт 80, который прослушивает балансировщик нагрузки.

Выводы, сделанные из демонстрационного конвейера

Эта демонстрационная реализация CI/CD отражает несколько ключевых элементов реального мира.

- Первые два этапа (модульное тестирование и сборка) демонстрируют непрерывную интеграцию. Каждый раз, когда разработчик фиксирует код в хранилище, сервер Jenkins запускает модульные тесты и пытается построить проект.
- Третий этап (создание образа DigitalOcean в качестве артефакта сборки) является началом непрерывной доставки. Мы можем использовать один и тот же образ при развертывании в каждой среде.

- Развертывание одного дроплета считается средой разработки или тестирования.
- Заключительный этап развертывает программу `ulsahgo` в производственной среде с высокой доступностью, тем самым закрывая цикл на конвейере непрерывного развертывания.
- Если на каком-либо этапе конвейера происходит сбой, последующие этапы пропускаются. В этом случае необходимо анализировать консольный вывод, чтобы решить проблему.

Этот конвейер опирается на инструменты с открытым исходным кодом. Весь код развертывания покрывается всего несколькими текстовыми файлами, которые хранятся там же, где и исходный код приложения.

Проницательные читатели подумают о множестве улучшений, которые можно было бы сделать на этих этапах. Назовем только некоторые из них.

- Сине-зеленое развертывание для обеспечения отсутствия простоя на этапе производства.
- Уведомления о состоянии, рассылаемое по электронной почте или в систему чатов, для каждого этапа.
- Триггеры систем мониторинга, отмечающие, что произошло новое развертывание.
- Улучшение способа передачи данных, таких как идентификатор образа, между этапами.

Постоянное улучшение является неотъемлемой частью подхода CI/CD (и для системного администрирования в целом). Со временем цепочка дополнительных улучшений приводит к созданию высокоэффективной и автоматизированной системы доставки программного обеспечения.

26.5. Контейнеры и упрощение среды CI/CD

Большинство программных средств имеют внешние зависимости, такие как сторонние библиотеки, конкретный формат файловой системы, наличие определенных переменных среды и других локализаций. Конфликт между требуемыми зависимостями часто затрудняет запуск нескольких приложений на одной виртуальной машине.

Еще больше дело осложняет то, что для создания приложения требуются ресурсы, которых нет. Например, для процесса сборки может потребоваться компилятор и набор тестов, но эти дополнительные функции не нужны во время выполнения.

🕮 Дополнительную информацию о контейнерах см. в главе 25.

Контейнеры предлагают элегантное решение этих проблем. С точки зрения операций среда требует только возможности запуска контейнеров. Вы можете активировать любой данный контейнер в любой совместимой с контейнером системе без дополнительной настройки конфигурации, потому что все зависимости и локализации для приложения размещаются внутри его контейнера. Несколько контейнеров могут работать на одной и той же системе одновременно без конфликтов.

Вы можете использовать контейнеры для упрощения среды CI/CD несколькими способами:

- запуская саму систему CI/CD внутри контейнера;
- создавая приложения внутри контейнеров;
- используя образы контейнера в качестве артефактов сборки для развертывания.

Первый момент довольно очевиден: вы можете запускать свое программное обеспечение CI/CD в контейнерах (включая как сам мастер, так и любые агенты), тем самым избегая накладных расходов, связанных с поддержкой выделенной системы для работы инфраструктуры CI/CD.

В двух других сценариях требуется немного больше объяснений. Мы рассмотрим их более подробно в следующих разделах.

Контейнеры как среда сборки

Конфигурация конкретной среды, необходимой для создания приложения, специфична для проекта, а иногда и довольно сложна. Вместо того чтобы устанавливать все необходимые инструменты, зависимости, а также создавать программное обеспечение непосредственно ваших агентских системах CI/CD, вы можете создавать свое программное обеспечение в контейнерах и оставлять агенты CI/CD в чистом и универсальном состоянии. Процесс сборки становится мобильным и независимым от конкретного агента CI/CD. Рассмотрим типичное приложение, которое зависит от базы данных PostgreSQL и хранилища ключей-значений Redis. Чтобы создать и протестировать приложение в традиционном окружении, вам понадобятся отдельные серверы для каждого компонента: самого приложения, демона Redis и базы данных PostgreSQL. В крайнем случае вы можете запустить все эти компоненты в одной системе, но вы, вероятно, не станете использовать один и тот же сервер для сборки и тестирования другой службы, имеющей другие зависимости.

Вместо этого вы можете использовать краткосрочные контейнеры для каждого компонента. В одном контейнере можно создавать и запускать приложение. Он может подключаться к отдельным контейнерам (на том же или другом хосте) с базой данных PostgreSQL и демоном Redis. Как только процесс сборки будет завершен, контейнеры могут быть остановлены и удалены. Вы можете использовать один и тот же агент CI/CD для создания других приложений без риска возникновения конфликтов, которые имеют другие зависимости.

Образ контейнера, используемый для создания программного обеспечения, должен отличаться от образа контейнера, в котором оно запускается. Образ, в котором происходит процесс сборки приложения, обычно больше, чем образ, в котором приложение запускается, поскольку оно включает дополнительные компоненты, такие как компиляторы и средства тестирования.

Большинство современных инструментов CI/CD включают встроенную поддержку контейнеров. У системы Jenkins есть подключаемый модуль Docker, который прекрасно сочетается с конвейером. Также попробуйте Drone (`try.drone.io`) — платформу CI/CD, разработанную на основе контейнеров.

Контейнерные образы как артефакты сборки

Результатом сборки может быть образ контейнера, развертываемый с помощью системы оркестровки контейнеров. Контейнер — легкий и мобильный инструмент. Перемещение образов контейнеров между системами через реестр образов выполняется легко и быстро. В любом средстве CI/CD может использоваться стратегия создания контейнеров.

Основной рабочий процесс принимает следующий вид.

1. Создайте приложение в контейнере, предназначенном для сборки.
2. Создайте образ контейнера, который включает приложение и его зависимости.

3. Поместите образ в реестр.
4. Разверните этот образ в среде выполнения контейнера.

Как правило, лучше использовать платформу управления контейнерами, такую как Docker Swarm, Mesos/Marathon, Kubernetes или AWS EC2 для развертывания образов в производственной среде. На этапе развертывания вашего конвейера можно вызывать соответствующие интерфейсы API и позволить платформе обрабатывать детали. Эта процедура проиллюстрирована на рис. 26.5.

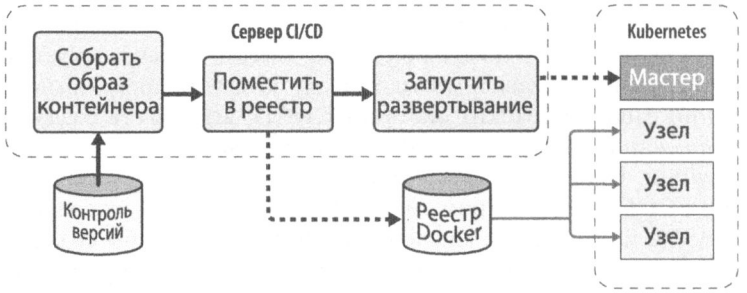


Рис. 26.5. Процесс развертывания на основе контейнера

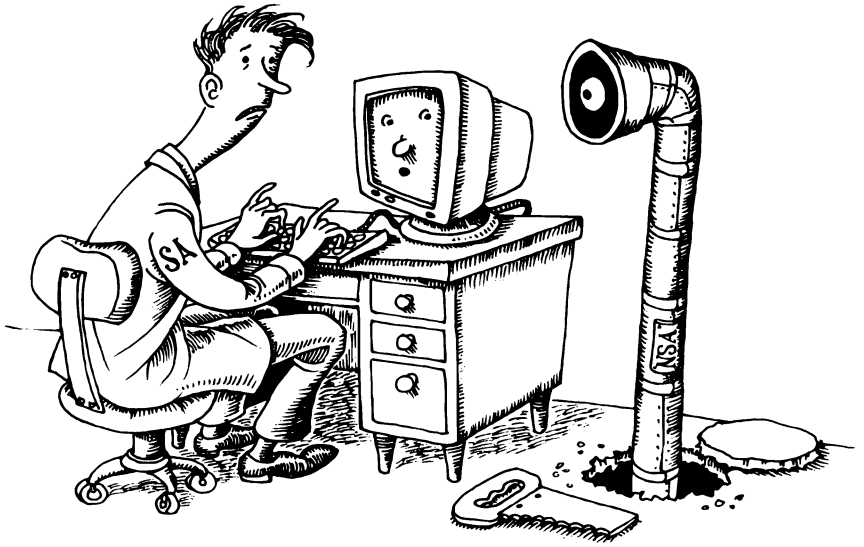
Мы обнаружили, что контейнеры отлично подходят для зрелых конвейеров CI/CD. Их чрезвычайно быстрый цикл упрощает развертывание нового кода и возврат к предыдущей версии в случае возникновения проблемы. Как виртуальные машины, так и системы управления конфигурацией работают на порядок медленнее.

26.6. Литература

- Beck, Kent, et al. *Manifesto for Agile Software Development*. `agilemanifesto.org`.
- Duvall, Paul M., Steve Matyas, and Andrew Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. Upper Saddle River, NJ: Addison-Wesley, 2007. (Поль М. Дюваль, Стивен М. Матиас, Эндрю Гловер, *Непрерывная интеграция: улучшение качества программного обеспечения и снижение риска*, пер. с англ., ИД "Вильямс", 2008 г.)
- Farcic, Viktor. *The DevOps 2.0 Toolkit: Automating the Continuous Deployment Pipeline with Containerized Microservices*. Seattle, WA: Amazon Digital Services LLC, 2016.
- Fowler, Martin. *Continuous Integration*. `goo.gl/Y2lisI` (`martinfowler.com`).
- Humble, Jez, and David Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Upper Saddle River, NJ: Addison-Wesley, 2010. (Джез Хамбл, Дейвид Фарли, *Непрерывное развертывание ПО: автоматизация процессов сборки, тестирования и внедрения новых версий программ*, пер. с англ., ИД "Вильямс", 2011 г.)
- Morris, Kief. *Infrastructure as Code: Managing Servers in the Cloud*. Sebastopol, CA: O'Reilly Media, 2016.
- `jenkinsci-docs@googlegroups.com. Jenkins User Handbook. jenkins.io/doc/book.`

Глава 27

Безопасность



Компьютерная безопасность находится в печальном состоянии. В отличие от прогресса, наблюдаемого практически во всех других областях вычислительной техники, недостатки безопасности становятся все более серьезными, а последствия неадекватных средств безопасности более опасными. Проблемы компьютерной безопасности напрямую угрожают обществам во всем мире.

Если у вас возникнет соблазн пропустить эту главу, позвольте нам возбудить ваше любопытство, напомнив о нескольких событиях компьютерной безопасности, которые произошли со времени последнего издания этой книги.

- Изощренный червь Stuxnet, обнаруженный в 2010 году, напал на ядерную программу Ирана, повредив центрифуги на заводе по обогащению урана.
- В 2013 году Эдвард Сноуден (Edward Snowden) разоблачил огромную машину слежки со стороны Агентства национальной безопасности США (National Security Agency — NSA), показав, что некоторые крупные интернет-компании были его соучастниками, позволяя правительству следить за гражданами США.
- Примерно в 2013 году наметился новый тип атаки, известный как выкуп (ransomware). Атакующие компрометируют целевую систему и шифруют ее данные, удерживая их в залоге. Жертвы должны заплатить выкуп за лечение и восстановление системы.
- В 2015 году было взломано Управление кадровой службы США (U.S. Office of Personnel Management), что поставило под угрозу конфиденциальную и приватную информацию о более чем 21 миллионе граждан США, многие из которых имели допуск к секретным документам.

- В 2017 году произошла беспрецедентная по масштабам атака с целью выкупа на системы Windows в более чем 150 странах. Атака использовала лазейку, обнаруженную Агентством национальной безопасности.

Ставки еще никогда не были так высоки. Мы думаем, что ситуация будет ухудшаться, а затем произойдет перелом к лучшему. Отчасти проблема заключается в том, что проблемы безопасности не являются чисто техническими. Вы не можете решить их, купив определенный продукт или услугу у третьей стороны. Достижение приемлемого уровня безопасности требует терпения, бдительности, знаний и настойчивости — не только от вас и других системных администраторов, но и от всего вашего сообщества пользователей и руководства.

Как системный администратор вы несете тяжелое бремя. Вы должны проводить мероприятия, которые защищают системы и сети вашей организации, обеспечивают их бдительный контроль и правильно обучают ваших пользователей и ваших сотрудников. Изучайте существующие технологии безопасности и работайте с экспертами для выявления и устранения уязвимостей в вашей организации. Вопросы безопасности должны быть частью каждого решения.

Запомните такую формулу:

$$Безопасность = \frac{1}{1{,}072 \times Удобство}$$

Чем безопаснее система, тем труднее пользователям работать в ней. Внедряйте меры безопасности, предложенные в этой главе, только после тщательного изучения последствий для пользователей.

Безопасна ли UNIX? Конечно нет. UNIX и Linux не являются безопасными, как и любая другая операционная система, которая обменивается данными в сети. Если вам нужна абсолютная, тотальная, непробиваемая система безопасности, то вам нужен значительный воздушный зазор между вашим компьютером и любым другим устройством.[1] Некоторые люди утверждают, что вам также необходимо разместить свой компьютер в специальной комнате, которая блокирует электромагнитное излучение (найдите в Интернете информацию о клетке Фарадея). Ну как, весело?

В этой главе рассматривается сложная область компьютерной безопасности: источники атак, основные способы защиты систем, инструменты для работы и источники дополнительной информации.

27.1. Элементы безопасности

Область информационной безопасности довольно широка, но ее лучше всего описывает "триада CIA":

- конфиденциальность (Confidentiality);
- целостность (Integrity);
- доступность (Availability).

Конфиденциальность касается секретности данных. Доступ к информации должен ограничиваться теми, кто уполномочен ее иметь. Аутентификация, контроль доступа

[1] Иногда даже воздушного зазора недостаточно. В статье 2014 года Генкин (Genkin), Шамир (Shamir) и Тромер (Tromer) описали метод извлечения ключей шифрования RSA из ноутбуков на основе анализа высоких частот, которые они излучают при расшифровке файла.

и шифрование — вот несколько составляющих частей конфиденциальности. Если хакер проникает в систему и крадет базу данных с контактной информацией о клиентах, конфиденциальность оказывается скомпрометированной.

Целостность относится к аутентичности информации. Технология обеспечения целостности данных гарантирует, что информация является корректной и не была изменена неавторизованными способами. Она также учитывает достоверность источников информации. Когда защищенный веб-сайт представляет подписанный сертификат TLS, он доказывает пользователю не только то, что передаваемая информация зашифровывается, но также проверяет подлинность источника доверенного центра сертификации (например, VeriSign или Equifax). Такие технологии, как PGP, также обеспечивают некоторую уверенность в целостности данных.

Доступность выражает мысль о том, что информация должна быть доступна авторизованным пользователям, когда они в ней нуждаются. В противном случае данные не имеют значения. Сбои, не вызванные злоумышленниками (например, возникшие вследствие ошибок системного администратора или перебоев в подаче электроэнергии), также относятся к категории проблем доступности. К сожалению, вопросы доступности часто игнорируются, пока не произойдет какая-нибудь неприятность.

Следуйте принципам CIA при проектировании, внедрении и обслуживании систем и сетей. Как говорится в старой поговорке о безопасности, "безопасность — это процесс".

27.2. Слабые места в системе защиты

В последующих разделах будут рассматриваться некоторые наиболее распространенные проблемы, связанные с безопасностью, и стандартные контрмеры, позволяющие избежать этих проблем. Но прежде чем погрузиться в детали, необходимо взглянуть на ситуацию в целом и определить основные источники неприятностей.

Социальная инженерия

Пользователи (и администраторы) системы часто являются ее слабым звеном в системе безопасности. Даже сейчас, когда образованность людей в области безопасности повысилась, беспечные пользователи легко распространяют конфиденциальную информацию. Никакая технология не может защитить сеть от неосторожных пользователей, поэтому системный администратор обязан информировать пользователей о существующих угрозах, и это обстоятельство должно стать частью системы безопасности.

Данная проблема может проявляться в разных формах. Хакеры звонят своим жертвам и представляются легальными пользователями, попавшими в трудное положение, пытаясь проникнуть в сеть. Иногда сами системные администраторы неосторожно размещают конфиденциальную информацию на открытых форумах, описывая решения сложных проблем. Физические взломы возникают, например, когда внешне легальные сотрудники отдела технического обслуживания пытаются перекоммутировать разводку сети в монтажном шкафу.

Термином *фишинг* (phishing) называют попытку сбора информации от пользователей с помощью обманных электронных сообщений, мгновенных сообщений и даже сообщений SMS, провоцирующих получателей выполнить опасное действие, например инсталлировать вредоносную программу. От фишинга (особенно так называемого "адресного фишинга") трудно защититься, поскольку сообщения часто содержат информацию, которая свидетельствует о знакомстве с жертвой и создает видимость аутентичности.

Социальная инженерия, т.е. использование человеческого фактора, остается мощным хакерским приемом, которому трудно противостоять. Ваша стратегия обеспечения безопасности должна предусматривать обучение новых сотрудников. Эффективно также распространять среди сотрудников информацию о телефонных атаках, физических проблемах, фишинге и правильном выборе паролей.

Может показаться, что для того, чтобы противостоять попыткам использовать человеческий фактор, можно попытаться самому спровоцировать такую атаку. Однако сначала необходимо получить разрешение от менеджера. Если у вас не будет такого разрешения, такие действия будут выглядеть подозрительными и похожими на внутренний шпионаж.

Многие организации считают необходимым сообщить своим пользователям, что администраторы ни при каких обстоятельствах не имеют права спрашивать у них пароли ни через электронные сообщения, ни через мгновенные сообщения, ни по телефону. О любой попытке узнать пароль с помощью этих средств следует немедленно сообщать в отдел информационных технологий.

Уязвимости в программах

За много лет в программном обеспечении (включая сторонние программы, как коммерческие, так и бесплатные) было выявлено несметное число ошибок, связанных с безопасностью. Используя незаметные программистские просчеты или контекстные зависимости, хакерам удавалось манипулировать системой по своему усмотрению.

Основной программной ошибкой, одной из самых опасных по своим последствиям, является переполнение буфера. Для хранения информации разработчики часто выделяют заранее определенный объем временной памяти, который называется *буфером*. Если программа не следит за размерами данных и контейнера, в котором они должны храниться, то память, смежная с выделенной областью, рискует оказаться перезаписанной. Умелые хакеры могут ввести тщательно скомпонованные данные так, чтобы они привели к краху программы или (в худшем случае) выполнили некий заданный код.

Переполнение буфера является частным случаем более широкого класса проблем, связанных с безопасностью программного обеспечения, который называется *уязвимостью проверки вводимых значений* (input validation vulnerability). Почти все программы принимают от пользователей вводимые данные (например, аргументы командной строки или данные формы HTML). Если программа обрабатывает эти данные без строгой проверки соответствующего формата и содержания, то могут возникнуть неприятности. Рассмотрим следующий простой пример.

В некотором роде операционные системы с открытым исходным кодом имеют иммунитет. Исходный код для Linux и FreeBSD доступен всем, и тысячи людей могут (и должны) тщательно анализировать каждую строку кода с точки зрения возможных угроз безопасности. Широко распространено мнение о том, что это соглашение обеспечивает лучшую безопасность, чем безопасность закрытых операционных систем, где ограниченное число людей имеют возможность изучить код в поисках слабых мест.

🕮 Дополнительную информацию о контейнерах см. в главе 25.

Что вы можете сделать как администратор, чтобы предотвратить такой тип атаки? Это зависит от приложения, но один очевидный подход заключается в уменьшении привилегий, выполняемых вашими приложениями, чтобы минимизировать влияние ошибок безопасности. Процесс, выполняющийся от имени непривилегированного пользователя, может нанести меньше урона, чем тот, который работает с правами суперпользователя. Параноидальный подход может включать в себя обязательную систему

управления доступом, такую как SELinux. Контейнеры с ограниченными возможностями также могут сыграть здесь свою роль.

Со временем сообщество разработчиков программ с открытым исходным кодом разработало стандартный процесс устранения уязвимостей программного обеспечения. Первоначальные отчеты должны поступать непосредственно разработчикам программного обеспечения, чтобы исправления для решения выявленной проблемы могли быть разработаны и выпущены до того, как хакеры разработают методы по ее использованию. Позднее детали проблемы, связанной с безопасностью, делаются публичными, чтобы администраторы узнали об этом, а проблема и исправления могли попасть под общественный контроль. По этой причине слежение за выпуском исправлений и бюллетеней по безопасности является важной частью работы большинства администраторов. К счастью, современные операционные системы стремятся упростить и легко автоматизировать обновления программного обеспечения.

Распределенные атаки типа "отказ в обслуживании" (DDoS)

Атака DDoS направлена на прерывание работы службы или снижение ее производительности, что делает службу недоступной для пользователей. Обычно это достигается путем наводнения сети трафиком, занимающим всю доступную пропускную способность сети или потребляющим значительные системные ресурсы.

Атаки DDoS могут быть финансово мотивированными (в этом случае злоумышленник хочет получить выкуп), политическими или ответными. Для проведения атаки злоумышленники устанавливают вредоносный код на незащищенные устройства за пределами сети жертвы. Этот код позволяет злоумышленникам удаленно управлять такими промежуточными системами, образуя "ботнет". В наиболее распространенном сценарии DDoS миньонам бот-сети поручается направить на жертву сетевой трафик.

В последние годы ботнеты были сформированы из подключенных к Интернету устройств, таких как IP-камеры, принтеры и даже средства слежения за детьми. У этих устройств практически нет средств безопасности, и владельцы обычно не знают, что их устройства были взломаны. Сложные инструменты управления бот-сетями можно купить в "темном" сегменте Интернета. Некоторые из них даже включают бесплатное обслуживание клиентов!

Осенью 2016 года ботнет Mirai нацелился на исследователя безопасности и блогера Брайана Кребса (Brain Krebs), обрушив на его сайт поток трафика на уровне 620 Гбит/с с десятков тысяч исходных IP-адресов. Естественно, его хостинг-провайдер любезно предложил ему куда-нибудь уйти. С тех пор код ботнета Mirai был открыт.

Большая часть ответственности за предотвращение и смягчение DDoS-атак приходится на уровень управления сетью. Для обнаружения атак и их отражения без прекращения работы легальных служб в режиме онлайн существует специальное программное и аппаратное обеспечение. Провайдеры открытых облачных систем и некоторые объекты совместного размещения серверов (co-location) уже оснащены этой технологией. Однако средства защиты не являются совершенными, и угрозы постоянно меняются.

Инсайдерская информация

Сотрудники, подрядчики и консультанты являются доверенными лицами организации и получают особые привилегии. Иногда эти привилегии используются во вред. Инсайдеры могут украсть или раскрыть данные, разрушить системы для финансовой выгоды или создать хаос по политическим мотивам.

Этот тип атаки часто бывает самым сложным для обнаружения. Большинство мер безопасности защищают от внешних угроз, поэтому они не эффективны против пользователей, которым был предоставлен доступ. Инсайдеры, как правило, не находятся под подозрением; только самые строгие организации систематически контролируют своих сотрудников.

Системные администраторы никогда не должны сознательно оставлять лазейки во внешнюю среду для собственного использования. Такие объекты слишком легко обнаруживаются и используются другими.

Ошибки конфигурации сети, системы или приложения

Программное обеспечение может быть настроено безопасно или небезопасно. Программное обеспечение должно быть полезным, а не раздражать пользователей, поэтому не стоит слишком часто использовать стандартные значения настроек. Хакеры часто получают доступ, используя функции, которые считаются полезными и удобными при менее коварных обстоятельствах: например, учетные записи без паролей, брандмауэры с чрезмерно слабыми правилами и незащищенные базы данных.

Типичным примером уязвимости конфигурации хоста является стандартная практика, позволяющая загружать системы Linux, не требуя пароля загрузчика. Загрузчик операционной системы GRUB можно настроить так, чтобы при инсталляции он требовал пароль, но администраторы почти никогда не используют эту возможность. Это упущение оставляет систему открытой для физической атаки. Тем не менее это также прекрасный пример необходимости сбалансировать безопасность с удобством использования. Требование пароля означает, что если система была непреднамеренно перезагружена (например, после отключения электропитания), администратор должен сам физически присутствовать, чтобы снова запустить машину в работу.

Один из самых важных шагов в обеспечении безопасности системы — просто убедиться, что вы случайно не открыли лазейку для хакеров. Проблемы в этой категории легче всего найти и исправить, хотя их потенциально много, и не всегда очевидно, что проверять. Средства сканирования портов и уязвимостей, рассмотренные в этой главе, могут помочь мотивированному администратору определить проблемы до того, как ими воспользуются.

27.3. Основные вопросы безопасности

Большинство операционных систем не поставляется с уже готовой защитой. Кроме того, настройка, которая осуществляется как во время инсталляции, так и после нее, изменяет профиль безопасности новых систем. Администраторы должны укреплять новые системы, интегрировать их в локальную среду и планировать их долговременную безопасную эксплуатацию.

Когда к вам придет проверка, вы должны доказать, что следовали стандартной методологии, особенно если эта методология соответствует общепринятым рекомендациям и передовым достижениям в вашей области промышленности. Рекомендации по выбору стандартной методологии см. в разделе 27.10.

На самом высоком уровне вы можете улучшить безопасность своей сети, имея в виду несколько эмпирических правил.

- Применять принцип наименьших привилегий, выделяя только минимальные привилегии, необходимые каждому объекту, человеку или роли. Этот принцип применяется к правилам брандмауэра, правам доступа пользователя, разрешениям на доступ к файлам и любой другой ситуации, когда используются средства контроля доступа.

- Меры безопасности уровня для обеспечения глубокой защиты. Например, не полагайтесь исключительно на внешний брандмауэр для защиты сети. В противном случае вы просто строите структуру, напоминающую карамельку: жесткая, хрустящая оболочка и мягкая внутренность.
- Минимизируйте поверхность атаки. Чем меньше интерфейсов, открытых систем, ненужных служб, а также неиспользуемых или недостаточно используемых систем, тем меньше вероятность возникновения уязвимостей и слабых мест с точки зрения безопасности.

Автоматизация является вашим лучшим союзником в войне за безопасность. Используйте управление конфигурацией и сценарии для создания воспроизводимых защищенных систем и приложений. Чем больше шагов безопасности вы автоматизируете, тем меньше места доступно для человеческой ошибки.

Обновления программного обеспечения

Оснащение операционной системы вновь разработанными программными исправлениями является первейшей обязанностью системного администратора. Большинство систем конфигурируется так, чтобы они имели связь с репозиторием пакетов производителя. В этом случае установка программного исправления сводится всего лишь к выполнению нескольких команд. В крупных сетях могут существовать локальные репозитории, являющиеся зеркалами репозитория производителя, так как они позволяют сэкономить сетевой трафик и ускорить установку обновлений.

При установке программных исправлений следует учитывать следующее.

- Нужно составить расписание инсталляции программных исправлений и строго его придерживаться. При составлении такого расписания необходимо учитывать его влияние на пользователей. Обычно достаточно выполнять ежемесячные обновления; регулярность важнее срочности. Нельзя сосредоточиваться только на защите от вредоносных программ и игнорировать другие обновления.
- Необходимо разработать план изменений, который учитывал бы влияние каждого набора исправлений, предусматривал соответствующие этапы тестирования после инсталляции системы и описывал возможности отказа от внесенных обновлений при возникновении проблем. Этот план следует обсудить со всеми заинтересованными сторонами.
- Необходимо понимать, какие заплаты подходят к данному окружению. Системные администраторы должны подписываться на специальные списки рассылки и блоги по вопросам безопасности, создаваемые производителями, а также принимать участие в работе форумов по вопросам безопасности, таких как Bugtraq.
- Точная инвентаризация приложений и операционных систем, используемых в вашей среде. Эта перепись помогает обеспечить полный охват. Используйте специальное программное обеспечение для отслеживания установленной базы.

Исправления программного обеспечения иногда сами создают новые проблемы в системе безопасности и слабые места. Однако большинство вредоносного кода обычно нацелено на более ранние уязвимости, которые широко известны. По статистике, намного лучше работать с системами, которые регулярно обновляются. Убедитесь, что это делается методично и последовательно.

Ненужные службы

Большинство систем поставляется с несколькими службами, которые запускаются по умолчанию. Отключите (и по возможности удалите) все ненужные службы, особенно если они реализованы в виде сетевых демонов. Для того чтобы обнаружить выполняемые службы, можно использовать команду **netstat**. Рассмотрим частичные результаты работы этой команды в системе FreeBSD.

```
freebsd$ netstat -an | grep LISTEN
tcp6    0    0    *.22       *.*       LISTEN
tcp6    0    0    *.2049     *.*       LISTEN
tcp6    0    0    *.989      *.*       LISTEN
tcp6    0    0    *.111      *.*       LISTEN
```



В системе Linux для этой цели используется новая команда **ss**, но команда **netstat** тоже хорошо справляется с этой задачей.

Для обнаружения служб, использующих конкретный порт, существует несколько приемов. Например, эту задачу можно решить с помощью команды **lsof**.

```
freebsd$ sudo lsof -i:22
COMMAND  PID  USER   FD TYPE SIZE/OFF  NODE NAME
sshd     701  root   3u IPv6 0t0        TCP *:ssh (LISTEN)
sshd     701  root   4u IPv4 0t0        TCP *:ssh (LISTEN)
sshd     815  root   3u IPv4 0t0        TCP 10.0.2.15:ssh-10.0.2.2:54834
     (ESTABLISHED)
sshd     817 vagrant 3u IPv4 0t0        TCP 10.0.2.15:ssh->10.0.2.2:54834
     (ESTABLISHED)
```

Выяснив идентификатор PID, с помощью команды **ps** можно идентифицировать конкретный процесс. Если данная служба не нужна, остановите ее и убедитесь, что она не будет заново стартовать во время загрузки системы. Если утилита **lsof** недоступна, используйте команды **fuser** или **netstat -p**.

🕮 Дополнительную информацию о процессах, выполняемых во время загрузки системы, см. в главе 2.

Ограничьте общий охват вашей системы. Чем меньше пакетов, тем меньше уязвимостей. В целом отрасль начинает решать эту проблему, уменьшая количество пакетов, включенных в стандартную установку. Некоторые специализированные дистрибутивы, такие как CoreOS, доводят это до крайности и вынуждают почти все пакеты работать в контейнере.

Удаленная регистрация событий

🕮 Дополнительную информацию о протоколе Syslog см. в главе 10.

Служба Syslog направляет регистрационную информацию в файлы, списки пользователей и другие хосты сети. Попробуйте настроить хост, отвечающий за безопасность, так, чтобы он действовал как центральная регистрационная машина, анализирующая полученные события и выполняющая соответствующие действия. Единый централизованный регистратор событий может получать сообщения от разных устройств и предупреждать администраторов о важных событиях. Удаленная регистрация также предотвращает перезапись или очистку регистрационных файлов во взломанных системах.

Большинство систем поставляется с конфигурацией, в котором служба Syslog включена по умолчанию, но для включения удаленной регистрации необходимо выполнить дополнительную настройку.

Резервные копии

Регулярное резервное копирование является важной частью системы безопасности любой сети. Оно входит в так называемую триаду CIA и относится к категории "доступность". Убедитесь, что все части системы регулярно копируются и что эта информация хранится в надежном месте (желательно за пределами сети организации). Если произойдет серьезный инцидент, связанный с безопасностью сети, вы сможете воспользоваться надежным источником информации и восстановить работу.

Резервное копирование само по себе также может создавать угрозу безопасности. Защитите свои резервные копии, ограничив и поставив под контроль доступ к файлам резервных копий, а также обеспечьте их шифрование.

Вирусы и черви

Системы UNIX и Linux имеют иммунитет от вирусов. Существует всего несколько вирусов (большинство из них носит чисто академический характер), которые могут заразить системы UNIX и Linux, но ни один из них не способен нанести серьезный урон, в отличие от среды Windows, где это стало частым явлением. Тем не менее этот факт не снижает озабоченности поставщиков антивирусных программ — разумеется, если вы покупаете их продукт по специальной сниженной цене.

Точная причина отсутствия вредоносных программ неясна. Некоторые утверждают, что система UNIX просто занимает слишком незначительную долю рынка настольных систем и поэтому не интересует авторов вирусов. Другие настаивают на том, что среда с контролем доступа, существующая в системе UNIX, ограничивает размер урона от самораспространяющихся червей или вирусов.

Последний аргумент заслуживает внимания. Поскольку в системе UNIX ограничен доступ по записи к исполняемым модулям на файловом уровне, непривилегированные пользователи не могут инфицировать остальную среду. Если код вируса был запущен не от имени суперпользователя, его вред будет существенно ограничен. Следовательно, не нужно использовать учетную запись `root` для повседневной деятельности. Комментарии по этому поводу см. в разделе 3.2.

🕮 Дополнительную информацию о сканировании содержимого электронных сообщений см. в главе 18.

Как ни странно, одна из причин внедрения антивирусного программного обеспечения на серверах UNIX — стремление защитить работающие под управлением Windows компьютеры, существующие в вашей сети, от Windows-ориентированных вирусов. Почтовый сервер может сканировать входящую электронную почту на вирусы, а файловый сервер в поисках "инфекции" может сканировать общие файлы. Однако это решение должно быть дополнительным по отношению к антивирусной защите настольных систем, а не основным.

Программа ClamAV, написанная Томашем Коймом (Tomasz Kojm), — это популярная свободно распространяемая антивирусная программа для систем UNIX и Linux. Она широко используется в качестве полноценного антивирусного программного обеспечения, которое хранит сигнатуры тысяч вирусов. Новейшую версию этой программы можно загрузить с сайта `clamav.net`.

Конечно, можно утверждать, что антивирусное программное обеспечение само по себе является контрпродуктивным. Его показатели обнаружения и профилактики являются посредственными, а стоимость лицензирования и управления является обременитель-

ной. Слишком часто антивирусное программное обеспечение нарушает другие аспекты системы, в результате чего возникают различные проблемы технической поддержки. Некоторые сбои даже были вызваны атаками на саму антивирусную инфраструктуру.

Недавние версии системы Microsoft Windows включают базовый антивирусный инструмент под названием Windows Defender. Это не самый быстрый способ обнаружения новых форм вредоносного программного обеспечения, но он эффективен и редко вмешивается в другие аспекты системы.

Руткиты

Умелые хакеры пытаются скрывать свои следы и избегать разоблачения. Часто они рассчитывают продолжить использование вашей системы для нелегального распространения программного обеспечения, зондирования других сетей или запуска атаки против других систем. Для того чтобы оставаться незамеченными, они часто используют так называемые "руткиты" ("rootkits"). Печально известная фирма Sony включала элементы, подобные руткиту, в программное обеспечение для защиты от копирования, помещаемое на миллионы музыкальных компакт-дисков.

Руткиты — это программы и заплатки, скрывающие важную системную информацию, например процесс, диск или сетевую активность. Они имеют много обличий и разную степень сложности — от простых замен приложения (например, взломанные версии утилит **ls** и **ps**) до модулей ядра, которые практически невозможно выявить.

Для эффективного мониторинга системы и выявления внедренных руткитов существует специальное программное обеспечение, например OSSEC. Разработаны также средства контроля целостности файлов, такие как AIDE для Linux, которые предупреждают о неожиданно измененных файлах. Кроме того, разработаны сценарии распознавания руткитов (такие как **chkrootkit**, `chkrootkit.org`), сканирующие систему в поисках известных руткитов.

Несмотря на существование программ, позволяющих системным администраторам удалять руткиты из взломанных систем, время, которое они вынуждены затрачивать на очистку систем, можно было бы сэкономить, сохранив данные, переформатировав диск и начав работу с нуля. Большинство современных руткитов знают о существовании программ для их удаления и пытаются оказывать этому сопротивление.

Фильтрация пакетов

Если вы подключаете систему к сети, имеющей доступ к Интернету, вы должны установить между системой и внешним миром маршрутизатор с возможностью фильтрации пакетов или межсетевой экран (брандмауэр). Фильтр пакетов должен передавать только трафик, предназначенный для тех служб, которые вы специально хотите предоставить пользователям из внешнего мира. Ограничение открытости ваших систем для внешнего мира — это первая линия защиты. Многие системы не обязательно должны быть доступны из открытого сегмента Интернета.

Кроме специализированных средств, типа межсетевого экрана, работающих на интернет-шлюзе, вы можете удвоить защиту с помощью пакетных фильтров, работающих на конкретном хосте, таких как **ipfw** для системы FreeBSD и **iptables** (или **ufw**) для систем Linux. Определите, какие службы запускаются на хосте, и открывайте порты только для этих служб. В некоторых случаях вы также можете определить, какие адреса отправителя пакетов разрешены для подключения к каждому порту. Для большинства систем требуется открыть всего несколько портов.

Если ваши системы находятся в облаке, вы можете использовать группы безопасности, а не физические брандмауэры. При разработке правил группы безопасности будьте как можно более внимательными к деталям. Подумайте о добавлении правил для фильтрации исходящих пакетов, чтобы ограничить возможность злоумышленника совершать исходящие подключения с ваших хостов.

🕮 Дополнительную информацию по этой теме см. в разделе 13.15, посвященном облачным платформам.

Пароли и многофакторная аутентификация

Мы — просто люди, подчиняющиеся простым правилам. Вот одно из них: у каждой учетной записи должен быть пароль, который нелегко угадать. Правила, устанавливающие определенную сложность пароля, могут создавать обычным пользователям некоторые неудобства, но они должны существовать по ряду причин. Легко угадываемые пароли являются одним из основных источников компрометации.

Одной из наших любимых тенденций последних лет является распространение поддержки систем многофакторной аутентификации (multifactor autentication — MFA). Эти схемы подтверждают вашу личность как тем, что вы знаете (пароль или кодовую фразу), так и тем, что у вас есть, например физическое устройство, обычно мобильный телефон. Почти любой интерфейс может быть защищен с помощью MFA — от учетных записей оболочки UNIX до банковских счетов. Поддержка MFA — это легкая и большая победа в области безопасности.

По разным причинам механизм MFA теперь является абсолютным минимальным требованием для любого интернет-портала, который предоставляет доступ к административным привилегиям. Он включает в себя VPN-туннели, SSH-доступ и административные интерфейсы для веб-приложений. Можно утверждать, что однофакторная (только с помощью пароля) аутентификация неприемлема для *любой* аутентификации пользователя, но вы *должны как минимум* обеспечить защиту всех административных интерфейсов с помощью механизма MFA. К счастью, сейчас доступны отличные облачные службы MFA, такие как Google Authenticator и Duo (`duo.com`).

Бдительность

Для того чтобы быть уверенным в безопасности системы, следите за ее состоянием, сетевыми соединениями, таблицей процессов. Делать это нужно регулярно (желательно каждый день). Проблема всегда начинается с малого, а затем нарастает, как снежный ком, так что, чем раньше будет обнаружена аномалия, тем меньшим окажется ущерб.

Возможно, вам будет полезно поработать с внешней фирмой для проведения всестороннего анализа уязвимостей. Эти проекты могут привлечь ваше внимание к вопросам, которые вы ранее не рассматривали. Как минимум, они устанавливают базовое понимание областей, в которых вы наиболее уязвимы. Такие исследования часто обнаруживают, что хакеры уже проникли в сеть клиента.

Тестирование приложений на проникновение

Приложения, которые подвергаются воздействию Интернета, нуждаются в собственных мерах предосторожности в дополнение к общей гигиене системы и сети. Поскольку сведения об уязвимостях и лазейках получили широкое распространение, рекомендуется хорошо протестировать все приложения, чтобы убедиться, что они разработаны с учетом правил безопасности и имеют соответствующие элементы управления.

Уровень безопасности определяется уязвимостью самого слабого звена в цепочке. Если у вас есть безопасная сетевая и системная инфраструктура, но приложение, работающее в этой инфраструктуре, обеспечивает доступ к конфиденциальным данным без пароля (например), вы выиграли битву, но проиграли войну.

Тестирование на проникновение является плохо определенной дисциплиной. Многие компании, которые рекламируют свои услуги по тестированию приложений на проникновение, в основном "мухлюют". Голливудские сцены с подростками, сидящими в подвалах без окон, заполненных терминалами эпохи 1980-х годов, не так уж и неточны. Поэтому будьте всегда на чеку!

К счастью, проект Open Web Application Security (OWASP) публикует информацию об общих уязвимостях, найденных в приложениях, и методах зондирования приложений для выявления этих проблем. Наша рекомендация: обратитесь к профессиональной компании, которая специализируется на тестировании приложений на проникновение, и выполняйте этот тест при первом запуске приложения, а также периодически на протяжении всего срока его службы. Убедитесь, что сотрудники компании придерживаются методологии OWASP.

27.4. Пароли и учетные записи пользователей

Кроме обеспечения привилегированного доступа со стороны Интернета с помощью многофакторной аутентификации важно безопасно выбирать и управлять паролями. В мире `sudo` личные пароли администраторов так же важны, как и пароли `root`. Более того: чем чаще используется пароль, тем больше возможностей для его компрометации с помощью методов, отличных от дешифрования методом прямого перебора.

С узкоспециальной точки зрения наиболее безопасный пароль заданной длины состоит из случайной последовательности букв, знаков препинания и цифр. Годы пропаганды и придирчивые формы проверки паролей на веб-сайтах убедили большинство людей, что это именно тот вид пароля, который они должны использовать. Но, конечно, они никогда так не поступают, если только не используют хранилище паролей для их запоминания. Случайные пароли просто непрактичны для фиксации в памяти при длинах, необходимых для противодействия атакам прямым перебором (12 символов или больше).

🕮 Дополнительную информацию о взломе паролей см. в разделе 27.5.

Поскольку безопасность паролей увеличивается экспоненциально с длиной, лучше всего использовать очень длинный пароль ("кодовую фразу"), который вряд ли появится в другом месте, но его легко запомнить. Для пущей надежности можно сделать преднамеренную опечатку или орфографическую ошибку, но общая идея заключается в том, чтобы использовать большую длину пароля и при этом не напрягать память.

Например, отличная кодовая фраза — "шесть гостей выпили отравленное вино Эви". (Или по крайней мере так было до тех пор, пока она не появилась в этой книге.) Это правда, несмотря на то, что пароль состоит в основном из простых, строчных, грамматически правильных слов и при этом слова логически связаны и грамматически упорядочены.

Другая основная концепция, которую должны учитывать все администраторы и пользователи, заключается в том, что ни одна кодовая фраза никогда не должна использоваться более чем для одной цели. Слишком часто происходят крупные взломы и имена пользователей с паролями оказываются раскрытыми. Если эти имена пользователей и пароли использовались в других местах, все эти учетные записи теперь подвергаются риску. Никогда не используйте один и тот же пароль для разных целей (например, в банковском личном кабинете и социальных сетях).

Изменение пароля

Изменение паролей суперпользователя `root` и администраторов должно происходить:

- не реже одного раза в шесть месяцев;
- каждый раз, когда кто-то, у кого есть доступ к ним, покидает вашу организацию;
- всякий раз, когда вы сомневаетесь в безопасности.

В прошлом считалось, что пароли должны меняться часто, чтобы не допускать скрытой компрометации. Однако обновления паролей имеют собственные риски и усложняют жизнь администраторов. Проникнув в сеть организации, компетентные хакеры устанавливают механизмы резервного копирования, поэтому изменения пароля менее полезны, чем могло бы показаться на первый взгляд.

По-прежнему рекомендуется регулярно выполнять запланированные изменения, но не злоупотреблять этим. Если вы действительно хотите повысить уровень безопасности, вам лучше думать не о качестве пароля.

Хранилища и депоненты паролей

Часто говорят, что пароли "никогда не должны записываться", но, возможно, точнее было бы сказать, что они никогда не должны оставаться доступными для людей, которые не имеют на это права. Как заметил эксперт по безопасности Брюс Шнайер (Bruce Schneier), клочок бумаги в кошельке администратора относительно безопасен по сравнению с большинством форм интернет-хранилища.

Хранилище паролей представляет собой часть программного обеспечения (или сочетание программного и аппаратного обеспечения), в котором пароли для вашей организации находятся в большей безопасности, чем после ответа на вопрос "Хотите, чтобы Windows запомнила этот пароль для вас?"

Хранилище паролей стало практически необходимым вследствие нескольких обстоятельств.

- Распространение паролей необходимо не просто для входа на компьютеры, но также для доступа к веб-страницам, настройки маршрутизаторов и брандмауэров и администрирования удаленных служб.
- Возрастает потребность в надежных паролях, поскольку компьютеры становятся такими быстрыми, что слабые пароли легко взламываются.
- Появились правила, согласно которым доступ к определенным данным должен прослеживаться вплоть до конкретного человека. Здесь общие учетные записи типа `root` уже не подходят.

Системы управления паролями стали более популярными в свете законодательства США, которое наложило дополнительные требования к таким секторам, как правительство, финансы и здравоохранение. В некоторых случаях это законодательство требует многофакторной аутентификации.

Хранилища паролей также являются отличным решением для компаний, предоставляющих услуги системного администрирования, которые должны безопасно и прозрачно управлять паролями не только к собственным машинам, но и к компьютерам своих клиентов.

Пароли хранятся в хранилищах паролей в зашифрованном виде. Как правило, каждый пользователь имеет отдельный пароль для доступа к хранилищу. (Если вы считали, что ваши парольные муки уже закончились, то теперь у вас есть еще больше паролей для отслеживания и беспокойства!)

Доступны многие реализации хранилищ паролей. Бесплатные хранилища для частных лиц (например, KeePass) локально хранят пароли, предоставляют доступ к базе данных паролей по принципу "все или ничего" и не требуют регистрации. Устройства, подходящие для огромных предприятий (например, CyberArk), могут стоить десятки тысяч долларов. Абонентная плата взимается либо в зависимости от количества пользователей, либо от количества хранящихся паролей.

Мы особенно любим хранилище паролей 1Password компании AgileBits (`1password.com`). Хранилище 1Password появилось из мира массового рынка, поэтому оно включает в себя тщательно отлаженные кросс-платформенные интерфейсы и взаимодействие с обычными веб-браузерами. Хранилище 1Password имеет "командный" уровень, расширяющий фундамент управления персональными паролями в области организационных секретов.

Еще одной превосходной системой является хранилище Secret Server компании Thycotic (`thycotic.com`). Эта система основана на браузере и была разработана с нуля для удовлетворения потребностей организаций. Она включает расширенные функции управления и аудита наряду с контролем доступа на основе ролей (см. раздел 3.4) и мелкомасштабных разрешений.

Одна из полезных функций, которую нужно искать в системе управления паролями, — это вариант "разбить стекло", названный так в честь устройств пожарной сигнализации отеля, на которых написано, что в случае чрезвычайной ситуации вы должны разбить стекло и нажать на большую красную кнопку. В данном случае "разбить стекло" означает получение пароля, к которому вы обычно не имеете доступа, при этом громкие сигналы тревоги пересылаются другим администраторам. Это хороший компромисс между расчетливым обменом паролями (нормальная передовая практика) и реалиями аварийного пожаротушения.

Плохое управление паролями приводит к общему ослаблению системы безопасности. По умолчанию допуск в систему определяется содержимым файлов **`/etc/passwd`** и **`/etc/shadow`** (или файла **`/etc/master.passwd`** в системе FreeBSD), поэтому данные файлы являются первой линией защиты системы от злоумышленников. Они должны быть тщательно сохранены и свободны от ошибок, угроз безопасности и исторического багажа.

🕮 Дополнительную информацию о файле **`passwd`** см. в разделе 8.2.

Система UNIX позволяет пользователям выбирать собственные пароли, и, хотя это очень удобно, данная возможность порождает много проблем, связанных с безопасностью. Комментарии в начале раздела 27.4 также относятся и к паролям пользователей.

Важно регулярно (желательно ежедневно) проверять, что каждая учетная запись имеет пароль. Записи в файле **`/etc/shadow`**, в которых описываются псевдопользователи, такие как демоны, которые владеют файлами, но не должны входить в систему, должны иметь звездочку или восклицательный знак в поле своего зашифрованного пароля. Эти символы не соответствуют никакому паролю и, таким образом, не позволяют использовать учетную запись для входа в систему.

В организациях, использующих централизованную схему аутентификации, такую как LDAP или Active Directory, применяется такая же логика. Требуйте использовать сложные пароли и блокируйте учетные записи после нескольких неудачных попыток входа в систему.

Устаревание паролей

В большинстве систем, использующих теневые пароли, можно реализовать механизм так называемого устаревания паролей, при котором пользователей заставляют перио-

дически менять пароли. На первый взгляд, это хорошая идея, однако ее практическая реализация влечет за собой определенные проблемы. Не всякому пользователю по душе периодически менять пароль, поскольку это сопряжено с запоминанием нового пароля. Обычно для пароля выбирается простое слово, которое легко вводится и запоминается, и, когда приходит время замены, многие пользователи, не желая себя утруждать, снова выбирают предыдущий пароль. Таким образом, дискредитируется сама идея. Модули PAM могут помочь выбрать сильные пароли, чтобы избежать описанной выше ситуации (см. раздел 17.3).



В системе Linux процессом устаревания паролей управляет программа **chage**. С ее помощью администраторы могут задавать минимальное и максимальное время, которое проходит до момента изменения пароля; дату истечения срока действия пароля; количество дней до наступления даты истечения срока действия пароля, когда следует заблаговременно предупредить пользователя; количество дней бездействия пользователя, после истечения которых учетная запись автоматически блокируются; и другие параметры. Приведенная ниже команда задает минимальное количество дней между изменениями пароля равным 2, а максимальное количество дней равным 90, дату истечения срока действия пароля равной 31 июля 2017 года, а также то, что пользователя следует предупредить об истечении срока действия пароля за 14 дней.

```
linux$ sudo chage -m 2 -M 90 -E 2017-07-31 -W 14 ben
```



В системе FreeBSD устареванием паролей управляет команда **pw**. Приведенная ниже команда задает срок действия пароля, равный 90 дням, и дату истечения срока действия пароля — 25 сентября 2017 года.

```
freebsd$ sudo pw user mod trent -p 2017-09-25 -e 90
```

Групповые и совместно используемые учетные записи

Опасно, если учетная запись используется несколькими людьми. Групповые учетные записи (например, **guest** и **demo**) представляют собой удобную лазейку для хакеров, поэтому они запрещены во многих сетях федеральными законами, такими как HIPAA. Не допускайте этого в своей сети. Однако технические средства не могут предотвратить совместное использование учетных записей разными пользователями путем сообщения паролей, поэтому в данном вопросе лучше всего вести разъяснительную работу.

Пользовательские оболочки

Теоретически можно установить в качестве оболочки для пользовательской учетной записи любую программу, включая пользовательский сценарий. На практике применение оболочек, отличающихся от стандартных, таких как **bash** и **tcsh**, весьма опасно. Еще опаснее беспарольные учетные записи, оболочкой которых является сценарий. Если у вас возникнет соблазн создать такую учетную запись, примените вместо пароля ключи SSH.

Привилегированные учетные записи

Единственная отличительная черта пользователя **root** состоит в том, что его UID равен 0. Поскольку в файле **/etc/passwd** может быть несколько записей с таким идентификатором, существует и несколько способов входа в систему в качестве суперпользователя **root**.

Один из способов, который хакеры, получив доступ к командной оболочке от имени пользователя **root**, широко применяют для открытия "черного хода", — редактирование файла **/etc/passwd**. Поскольку такие команды, как **who** и **w**, работают с регистрационным именем, хранящимся в файле **utmp**, а не с идентификатором пользователя, зарегистрировавшегося в командной оболочке, они не в состоянии разоблачить хакера, который выглядит как рядовой пользователь, хотя на самом деле зарегистрирован в системе в качестве суперпользователя с UID равным 0.

Не допускайте удаленные подключения суперпользователей, даже через стандартную учетную запись **root**. Для того чтобы это запретить, следует с помощью оболочки OpenSSH установить в файле **/etc/ssh/sshd_config** параметр конфигурации `PermitRootLogin` равным `No`.

Благодаря программе **sudo** (см. раздел 3.2) необходимость подключаться к системе в качестве суперпользователя **root**, даже с системной консоли, возникает редко.

27.5. Инструментальные средства защиты

Для решения задач, упоминавшихся в предыдущих разделах, можно использовать свободно распространяемые программы. Ниже будут рассмотрены наиболее интересные из них.

Команда nmap: сканирование сетевых портов

Эта команда является сканером сетевых портов. Ее основное назначение — проверка указанных компьютеров на предмет того, какие TCP- и UDP-порты на них прослушиваются серверными программами.[2] За большинством сетевых служб закреплены "известные" порты, поэтому такая информация позволяет узнать, какие программы выполняются на компьютере.

Запуск команды **nmap** — отличный способ узнать, как выглядит система в глазах того, кто пытается ее взломать. В качестве примера приведем отчет, выданный этой командой на самом обычном, относительно незащищенном компьютере.

```
ubuntu$ nmap -sT ubuntu.booklab.atrust.com

Starting Nmap 7.40 ( http://insecure.org ) at 2017-03-01 12:31 MST
Interesting ports on ubuntu.booklab.atrust.com (192.168.20.25):
Not shown: 1691 closed ports
PORT      STATE   SERVICE
25/tcp    open    smtp
80/tcp    open    http
111/tcp   open    rpcbind
139/tcp   open    netbios-ssn
445/tcp   open    microsoft-ds
3306/tcp  open    mysql

Nmap finished: 1 IP address (1 host up) scanned in 0.186 seconds
```

Аргумент **-sT** сообщает команде **nmap** о необходимости подключиться к каждому TCP-порту на указанном хосте.[3] Как только соединение устанавливается, команда **nmap**

[2]Как объяснялось в разделе 13.3, порт — это нумерованный канал взаимодействия. IP-адрес идентифицирует весь компьютер, а комбинация IP-адреса и номера порта определяет конкретный сервер или сетевое соединение.

[3]На самом деле по умолчанию проверяются только привилегированные (их номера меньше 1024) и "известные" порты. С помощью опции **-p** можно явно задать диапазон сканирования.

немедленно отключается, что не очень корректно, зато безболезненно для правильно написанного сетевого сервера.

Как следует из примера, на хосте ubuntu выполняются две службы, являющиеся традиционными источниками проблем для безопасности системы: **portmap** (rpcbind) и почтовый сервер (smtp). Это наиболее вероятные места для атаки хакеров на этапе предварительного сбора информации об атакуемом хосте.

Столбец STATE (состояние) содержит запись **open** (открыт) для портов, с которыми связаны серверы, **closed** (закрыт) для портов, с которыми не связан ни один сервер, запись **unfiltered** (нефильтруемый) для портов, пребывающих в неизвестном состоянии, и запись **filtered** (фильтруемый) для портов, доступ к которым невозможен из-за наличия брандмауэра. Утилита **nmap** классифицирует порты как **unfiltered** только при ACK-сканировании. Вот, например, результат запроса к более защищенному коммерческому веб-серверу secure.booklab.atrust.com.

```
ubuntu$ nmap -sT secure.booklab.atrust.com

Starting Nmap 7.40 ( http://insecure.org ) at 2017-03-01 12:42 MST
Interesting ports on secure.booklab.atrust.com (192.168.20.35):
Not shown: 1691 closed ports
PORT     STATE   SERVICE
25/tcp   open    smtp
80/tcp   open    http

Nmap finished: 1 IP address (1 host up) scanned in 0.143 seconds
```

В данном случае очевидно, что компьютер сконфигурирован на обработку электронной почты по протоколу SMTP и работу с сервером HTTP. Брандмауэр блокирует доступ к остальным портам.

Помимо стандартного опроса TCP- и UDP-портов, команда **nmap** способна проверять порты "по-хакерски", не устанавливая с ними реального соединения. В большинстве случаев посылаются пакеты, которые похожи на пакеты из уже имеющегося TCP-соединения (т.е. не пакеты установки соединения), а затем проверяются полученные в ответ диагностические пакеты. Таким способом можно обойти брандмауэр или программу мониторинга сети, которая контролирует появление сканеров портов. Если в вашей организации имеется брандмауэр (см. раздел 27.7), не поленитесь проверить его в разных режимах сканирования.

Команда **nmap** обладает магической способностью угадывать, какая операционная система установлена на удаленном хосте. Это делается путем анализа деталей реализации стека TCP/IP. Чтобы проверить работу команды **nmap** в этом режиме, воспользуйтесь параметром **-O**, например.

```
ubuntu$ sudo nmap -sV -O secure.booklab.atrust.com
Starting Nmap 7.40 ( http://insecure.org ) at 2009-11-
1 12:44 MST
Interesting ports on secure.booklab.atrust.com (192.168.20.35):
Not shown: 1691 closed ports
PORT     STATE   SERVICE    VERSION
25/tcp   open    smtp       Postfix smtpd
80/tcp   open    http       lighttpd 1.4.13
Device type: general purpose
Running: Linux 2.4.X|2.5.X|2.6.X
OS details: Linux 2.6.16 - 2.6.24

Nmap finished: 1 IP address (1 host up) scanned in 8.095 seconds
```

Эта возможность может оказаться весьма полезной при анализе состояния локальной сети. К сожалению, она же является рабочим инструментом хакеров, которые могут определить тип атаки на основании известных слабых мест конкретных операционных систем или серверов.

Не забывайте также, что большинство администраторов не приветствуют попыток сканирования сети и выяснения ее слабых мест, какими бы благородными ни были мотивы. Никогда не запускайте команду `nmap` в чужой сети без разрешения сетевого администратора.

Nessus: сетевой сканер нового поколения

В 1998 году Рено Дерезон (Renaud Deraison) выпустил интересный пакет Nessus, в котором реализуются многие функциональные возможности сканера. Он использует более 31000 надстроек для проверки локальных и удаленных дефектов. Несмотря на то что эта программа имеет закрытый код и является коммерческим продуктом, она свободно распространяется и для нее регулярно появляются новые надстройки. Это наиболее широко распространенный и полный из доступных сканеров.

Программа Nessus — это сетевой сканер, который ничему не доверяет. Вместо того чтобы предполагать, к примеру, будто все веб-серверы работают через порт 80, он ищет веб-серверы на любом порту, а затем анализирует их слабые места. Программа не доверяет даже номеру версии, о котором сообщает сам сервер, проверяя все известные слабые места, чтобы понять, насколько уязвим сервер.

Для первого запуска программы Nessus требуется потратить немало усилий (необходимо установить много пакетов, которые не входят в стандартный дистрибутив), но конечный результат того стоит. Система Nessus состоит из клиента и сервера. Сервер играет роль базы данных, а клиент отвечает за графический пользовательский интерфейс. Серверы и клиенты могут работать на платформах UNIX или Windows.

Одним из главных преимуществ программы Nessus является ее модульная структура, позволяющая сторонним разработчикам добавлять собственные модули проверки. Благодаря активности сообщества ее пользователей эта программа сможет быть полезной долгое время.

Metasploit: программа для выявления попыток проникновения

Проверка проникновения — это выявление попыток проникновения в компьютерную сеть с разрешения владельца с целью обнаружения слабых мест в ее системе безопасности. Metasploit — это программный пакет с открытым исходным кодом, написанный на языке Ruby, который автоматизирует данный процесс.

Программа Metasploit контролируется американской охранной компанией Rapid7, но ее проект GitHub содержит сотни участников. Metasploit включает базу данных сотен готовых программ по использованию известных уязвимых мест программного обеспечения. Те, у кого есть желание и умение, могут добавить в базу данных настраиваемые дополнительные модули.

Metasploit использует следующий основной рабочий процесс.

1. Сканирует удаленные системы, чтобы узнать информацию о них.
2. Выбирает и выполняет программы для использования уязвимых мест в соответствии с найденной информацией.
3. Если произошло проникновение, использует поставляемые программные средства для отключения скомпрометированной системы и переходит к другим хостам в удаленной сети.

4. Формирует отчеты для документирования результатов.
5. Очищает и возвращает все внесенные в удаленную систему изменения в исходное состояние.

Программа Metasploit имеет несколько интерфейсов: командную строку, веб-интерфейс и полноценный графический пользовательский интерфейс. Выберите формат, который вам больше всего нравится; все они имеют эквивалентную функциональность.

Узнайте больше на сайте `metasploit.com`.

Lynis: встроенный аудит безопасности

Если вам когда-либо приходилось искать дырки в стенах старого деревянного сарая, наверняка вы сначала проходили по его наружному периметру и искали большие, зияющие отверстия. Сетевые средства сканирования уязвимостей, такие как Nessus, дают вам подобное представление о профиле безопасности системы. Исследование стен изнутри сарая в солнечный и ясный день позволяет выявить очень малые отверстия в них. Поэтому чтобы получить такой же уровень проверки системы, вам нужно программное средство, наподобие Lynis, которое запускается в самой системе.

Несмотря на свое неудачное название, эта утилита проверки системы безопасности выполняет как разовые, так и плановые проверки состояния конфигурации системы, примененных заплаток и мер по усилению безопасности. Эта программа с открытым исходным кодом работает в системах Linux и FreeBSD и выполняет сотни автоматических проверок уязвимостей. Утилиту Lynis можно загрузить по адресу `cisofy.com/lynis`.

John the Ripper: средство для выявления слабых паролей

Один из способов пресечь использование плохих паролей заключается в том, чтобы самостоятельно взломать пароль и заставить пользователя выбрать другой. John the Ripper — это сложная программа, разработанная компанией Solar Designer, которая реализует разные алгоритмы взлома паролей в виде единого инструмента. Она заменила программу **`crack`**, которая была описана в предыдущем издании книги.

Несмотря на то что в большинстве систем используется файл теневых паролей, чтобы скрыть зашифрованные пароли от публики, целесообразно проверять, что пароли пользователей являются устойчивыми ко взлому.[4] Знание пароля пользователя может оказаться полезным, потому что люди предпочитают все время использовать один и тот же пароль. Единственный пароль может обеспечить доступ к другой системе, расшифровать файлы, хранящиеся в рабочем каталоге, и открыть доступ к финансовым счетам в сети веб. (Не стоит и говорить, что использовать пароль таким способом очень глупо. Однако никто не хочет запоминать десять разных паролей.)

Несмотря на свою внутреннюю сложность, программа John the Ripper весьма проста в использовании. Достаточно просто нацелить программу **`john`** на файл, подлежащий взлому, чаще всего **`/etc/shadow`**, и произойдет чудо.

```
$ sudo ./john /etc/shadow
Loaded 25 password hashes with 25 different salts (FreeBSD MD5 [32/32])
password (jsmith)
badpass (tjones)
```

В этом примере из теневого файла были считаны 25 уникальных паролей. После взлома пароля программа **`John`** выводит его на экран и сохраняет в файл **`john.pot`**.

[4] Особенно это относится к паролям системных администраторов, имеющих привилегии **`sudo`**.

Этот вывод содержит пароль в левом столбце и регистрационное имя пользователя в скобках в правом столбце. Для просмотра паролей после завершения работы программы `john` следует выполнить ту же самую команду с аргументом `-show`.

На момент выпуска русского издания книги самой новой версией программы John the Ripper была версия 1.9.0. Она доступна на сайте `openwall.com/john`. Поскольку вывод программы содержит взломанные пароли, его следует тщательно охранять и удалить сразу же после нахождения слабых паролей пользователей.

Как и для большинства других методов мониторинга безопасности, перед взломом паролей с помощью программы John the Ripper необходимо получить одобрение руководства.

Bro: программная система для распознавания вторжения в сеть

Система с открытым исходным кодом Bro предназначена для распознавания вторжений в сеть (network intrusion detection system — NIDS). Она выполняет мониторинг сети и следит за подозрительной активностью. Эта система была написана Верном Паксоном (Vern Paxson) и доступна на сайте `bro.org`.

Система Bro проверяет весь трафик, поступающий в сеть и исходящий из нее. Она может функционировать в пассивном режиме, генерируя предупреждения о подозрительной активности, или в активном режиме, в котором она пресекает враждебные действия. Оба режима, скорее всего, потребуют внесения изменений в конфигурацию вашей сети.

В отличие от других систем NIDS, система Bro отслеживает потоки трафика, а не просто сопоставляет образцы внутри отдельных пакетов. Это значит, что система Bro может распознавать подозрительную активность, основываясь на информации о том, кто с кем говорит, даже не сравнивая никаких строк или шаблонов. Например, система Bro может решать следующие задачи.

- Распознавать системы, использующие "мостики" ("stepping stones"), определяя корреляцию между входящим и исходящим трафиками.
- Распознавать сервер, имеющий инсталлированную лазейку, путем слежения за неожиданными исходящими соединениями, следующими сразу же после входящего соединения.
- Распознавать протоколы, выполняемые на нестандартных портах.
- Сообщать о правильно угаданных паролях (и игнорировать неправильные предположения).

Некоторые из этих функциональных возможностей требуют существенных системных ресурсов, но система Bro имеет поддержку кластеризации, облегчающую управление группами машин, распознающих вторжение.

Система Bro имеет сложный язык конфигурации, требующий значительного опыта кодирования. К сожалению, в системе не предусмотрены параметры стандартной конфигурации, которые можно было бы легко использовать новичкам. Для большинства систем требуется выполнить довольно приличный уровень настроек.

Система до некоторой степени поддерживается Группой сетевых исследований Международного института компьютерных наук (Networking Research Group of the International Computer Science Institute (ICSI)), но в большей степени она поддерживается сообществом ее пользователей. Если вы ищете готовую коммерческую систему NIDS, то, вероятно, будете разочарованы системой Bro. Однако она может делать то, что не доступно коммерческим системам NIDS, и может служить дополнением или заменой коммерческого продукта.

Snort: популярная программная система для распознавания проникновения в сеть

Система с открытым кодом Snort предназначена для предотвращения и распознавания несанкционированного проникновения в сеть; написана Марти Решем (Marty Roesch) и в настоящее время поддерживается коммерческой компанией Cisco (`snort.org`). Она де-факто стала стандартом для кустарных систем NIDS, а также основной для многих коммерческих реализаций систем NIDS с "управляемыми услугами".

Сама по себе система Snort распространяется как пакет с открытым кодом. Однако компания Cisco взимает плату за подписку на доступ к новейшим правилам распознавания.

Большое количество платформ, созданных сторонними производителями, включают в себя или экспортируют систему Snort, причем некоторые из этих проектов являются программами с открытым кодом. Ярким примером является проект Aanval (`aanval.com`), собирающий данные от многочисленных датчиков системы Snort на веб-консоль.

Система Snort перехватывает пакеты из сетевого кабеля и сравнивает их с наборами правил, которые называются *сигнатурами*. Когда система Snort распознает событие, которое представляет интерес, она может предупредить системного администратора или установить контакт с сетевым устройством и, помимо прочего, заблокировать нежелательный трафик.

Несмотря на то что система Bro намного мощнее, система Snort намного проще и легче конфигурируется, благодаря чему является удачным выбором для стартовой платформы NIDS.

OSSEC: система для распознавания вторжения в сеть на уровне хоста

Система OSSEC — это свободно распространяемое программное обеспечение, доступное в виде открытого исходного кода по лицензии GNU (General Public License). Она обеспечивает следующие функциональные возможности.

- Распознавание руткитов.
- Проверка целостности файловой системы.
- Анализ регистрационных файлов.
- Выдача сигналов по расписанию.
- Активные реакции.

Система OSSEC работает в заданной операционной системе и отслеживает ее активность. Она может выдавать сигналы или предпринимать действия, предусмотренные набором правил, которые устанавливает пользователь. Например, система OSSEC может выполнять мониторинг операционных систем, чтобы выявить добавление неавторизованных файлов и отправить по электронной почте уведомление, похожее на приведенное ниже.

```
Subject: OSSEC Notification - courtesy - Alert level 7
Date: Fri, 03 Feb 2017 14:53:04 -0700
From: OSSEC HIDS ossecm@courtesy.atrust.com
To: <courtesy-admin@atrust.com>

OSSEC HIDS Notification.
2017 Feb 03 14:52:52
```

```
Received From: courtesy->syscheck
Rule: 554 fired (level 7) -> "File added to the system."
Portion of the log(s):

New file
'/courtesy/httpd/barkingseal.com/html/wp-
content/uploads/2010/01/hbird.jpg'
added to the file system.

--END OF NOTIFICATION
```

Таким образом, система OSSEC работает круглосуточно, следя за операционной системой. Мы рекомендуем запускать OSSEC на каждом производственном компьютере.

Основные принципы системы OSSEC

Система OSSEC состоит из двух основных компонентов: менеджера (сервера) и агентов (клиентов). В сети нужен один менеджер, который необходимо инсталлировать первым. Менеджер хранит базу данных для проверки целостности файловой системы, регистрационные журналы, события, декодеры, основные варианты конфигурации, а также записи об аудите системы для всей сети. Менеджер может подключаться к любому агенту OSSEC, независимо от его операционной системы. Менеджер также может отслеживать работу определенных устройств, которые не имеют специального агента OSSEC.

Агенты работают в системах, которые подвергаются мониторингу, и сообщают информацию менеджеру. Они имеют маленькую зону обслуживания и оперируют небольшим объемом привилегий. Большую часть конфигурации агент получает от менеджера. Соединение между сервером и агентом зашифровано и аутентифицировано. Для каждого агента менеджер должен создать отдельный ключ аутентификации.

Система OSSEC классифицирует серьезность сигналов по уровням от 0 до 15; число 15 соответствует высшему уровню опасности.

Инсталляция системы OSSEC

Пакеты системы OSSEC для основных дистрибутивов операционных систем доступны на сайте `ossec.github.io`.

Сначала следует выполнить инсталляцию сервера OSSEC в предназначенной для этого операционной системе, а затем инсталляцию агента во всех остальных системах, которые будут предметом мониторинга. Сценарий инсталляции тоже задает несколько дополнительных вопросов, например, по какому адресу посылать уведомления и какие модули мониторинга следует подключить.

После завершения инсталляции запустите систему OSSEC с помощью следующей команды.

```
server$ sudo /var/ossec/bin/ossec-control start
```

Затем зарегистрируйте каждый агент у менеджера. Находясь на сервере, выполните следующую команду.

```
server$ sudo /var/ossec/bin/manage_agents
```

Вы увидите меню, которое будет выглядеть примерно так.

```
****************************************
* OSSEC HIDS v2.8 Agent manager.
* The following options are available:
****************************************
```

```
(A)dd an agent (A).
(E)xtract key for an agent (E).
(L)ist already added agents (L).
(R)emove an agent (R).
(Q)uit.
Choose your action: A,E,L,R or Q:
```

Выберите пункт **A**, чтобы добавить агент, а затем наберите имя и IP-адрес этого агента. Затем выберите пункт **E** и извлеките ключ агента. Вот как это выглядит.

```
Available agents:
  ID: 001, Name: linuxclient1, IP: 192.168.74.3
Provide the ID of the agent to extract the key (or '\q' to quit): 001
Agent key information for '001' is:
MDAyIGxpbnV4Y2xpZW50MSAxOTIuMTY4Ljc0LjMgZjk4YjMyYzlkMjg5MWJlMT
...
```

В заключение подключитесь к удаленной агентской системе и запустите в ней команду **manage_agents**.

```
agent$ sudo /var/ossec/bin/manage_agents
```

Находясь на компьютере клиента, вы увидите меню с другими пунктами.

```
****************************************
* OSSEC HIDS v2.8 Agent manager.
* The following options are available:
****************************************
   (I)mport key from the server (I).
   (Q)uit.
Choose your action: I or Q:
```

Выберите пункт **I**, а затем вырежьте и вставьте ключ, который был извлечен ранее. Добавив агент, заново запустите сервер OSSEC. Повторите процесс генерирования ключа, его извлечение и инсталляцию для каждого агента, с которым хотите установить связь.

Конфигурация системы OSSEC

После инсталляции и запуска системы OSSEC вам следует настроить ее так, чтобы она выдавала достаточный объем информации, но не слишком большой. Большая часть конфигурации хранится на сервере в файле **/var/ossec/etc/ossec.conf**. Это XML-подобный файл, который подробно прокомментирован и вполне понятен, хотя и содержит десятки параметров.

Чаще всего необходимо конфигурировать список файлов, которые следует игнорировать при проверке целостности (наличия изменений) файла. Например, если у вас есть приложение, записывающее свой регистрационный файл в каталог **/var/log/customapp.log**, вы можете добавить следующую строку в раздел <syscheck> этого файла.

```
<syscheck>
   <ignore>/var/log/customapp.log</ignore>
</syscheck>
```

После того как внесете эти изменения и заново запустите сервер OSSEC, система OSSEC прекратит выдавать сообщения при каждом изменении регистрационного файла. Многие параметры конфигурации системы OSSEC описаны на странице http://www.ossec.net/docs/manual/monitoring/index.html#configuration-options.

Для того чтобы запустить и настроить любую систему HIDS, требуется затратить много времени и усилий. Но через несколько недель вы отфильтруете шум, и система станет выдавать ценную информацию об изменяющихся условиях в вашем окружении.

Fail2Ban: система отражения атаки методом перебора

Fail2Ban — это сценарий на языке Python, который контролирует файлы журнала, такие как `/var/log/auth.log` и `/var/log/apache2/error.log`. Он ищет подозрительные события, такие как несколько неудачных попыток входа в систему или запросы к необычно длинным URL-адресам. Затем Fail2Ban принимает меры для устранения угрозы. Например, он может временно блокировать сетевой трафик с определенного IP-адреса или отправлять электронную почту в группу реагирования на инциденты. Узнайте больше на сайте `fail2ban.org`.

27.6. Основы криптографии

Большинство программ разработано с учетом безопасности, и это подразумевает использование криптографии в больших масштабах. Стандарты и правила безопасности подразумевают выбор криптографических алгоритмов и типа данных, которые должны быть защищены криптографически. Почти во всех современных сетевых протоколах используются криптографические методы для обеспечения безопасности. Короче говоря, криптография — это столп компьютерной безопасности, и системные администраторы сталкиваются с ней каждый день. Следовательно, желательно знать ее основы.

В криптографии применяются математические методы для защиты систем связи. Криптографический алгоритм, называемый *шифром*, представляет собой набор математических операций, предпринятых для защиты сообщения. Такие алгоритмы разрабатываются коллективами экспертов, представляющими академические, правительственные и исследовательские организации во всем мире. Принятие нового алгоритма — длительный и утомительный процесс. К тому времени, когда он добирается к массам, он уже тщательно проверен.

Шифрование — это процесс использования шифра для преобразования текстовых сообщений в нечитаемый зашифрованный текст. Расшифровка — это обратная сторона этого процесса. Криптографические сообщения (зашифрованный текст) имеют несколько полезных свойств.[5]

- *Конфиденциальность*: сообщение не может прочитать никто, кроме предполагаемых получателей.
- *Целостность*: невозможно изменить содержимое сообщения без обнаружения.
- *Невозможность отказа от авторства*: подлинность сообщения может быть проверена.

Другими словами, криптография позволяет обмениваться секретными сообщениями по незащищенным каналам связи, а также, в качестве дополнения, может подтвердить, подлинность сообщения и личность отправителя. Это очень ценное свойство.

Математически доказано, что сильные криптографические алгоритмы достаточно надежны. Однако программное обеспечение, реализующее эти алгоритмы, может иметь уязвимости, которые ставят под угрозу безопасность систем, защищающих криптогра-

[5] В некоторых шифрах реализовано только подмножество этих свойств. Поэтому часто несколько шифров используются вместе, формируя таким образом гибридную криптосистему.

фические секреты, что делает такие алгоритмы слабыми. Поэтому защита ваших секретов и выбор хорошо продуманного, легко обновляемого программного обеспечения для криптографии имеют первостепенное значение.

У криптографов есть традиционные имена для трех субъектов, которые участвуют в простом обмене сообщениями: Алиса и Боб, которые хотят общаться в конфиденциальном режиме, и Мэллори, злоумышленница, которая хочет выведать их секреты, нарушить их общение или выдать себя за другое лицо. Мы будем придерживаться этого соглашения.

В следующих подразделах представлены несколько криптографических примитивов, связанные с ними шифры и некоторые общие варианты использования каждого из них.

Криптография с симметричными ключами

Криптографию с симметричными ключами иногда называют обычной, или классической криптографией, потому что идеи, лежащие в ее основе, существуют уже давно. Приведем пример.

В этом случае Алиса и Боб должны иметь общий секретный ключ, который они будут использовать для шифрования и расшифровки сообщений. Перед этим они должны каким то образом тайно получить этот ключ, или обменяться им. Если они оба знают секретный ключ, они могут его использовать столько раз, сколько пожелают. В данном случае Мэллори сможет прочитать (или подделать) сообщение только если у нее также есть секретный ключ.

Симметричные ключи относительно эффективны с точки зрения использования центрального процессора и размера зашифрованных сообщений. В результате симметричная криптография часто используется в приложениях, где необходимы эффективное шифрование и расшифровка. Однако необходимость предварительного распространения общего секретного ключа является серьезным препятствием во многих случаях использования этого метода шифрования.

AES (Advanced Encryption Standard), стандарт расширенного шифрования Национального института стандартов и технологий США (National Institute of Standards and Technology — NIST), является, пожалуй, наиболее широко используемым алгоритмом с симметричными ключами. Также доступны варианты Twofish и его предшественника Blowfish, разработанные криптографом и экспертом по безопасности Брюсом Шнайером (Bruce Schneier). Эти алгоритмы играют определенную роль в обеспечении безопасности каждого сетевого протокола, включая SSH, TLS, IPsec VPN, PGP и многие другие.

Криптография с открытым ключом

Серьезным препятствием для использовании криптографии с симметричными ключами является необходимость в предварительном и безопасном получении сторонами этого самого секретного ключа. Единственный способ сделать это с полной безопасностью — встретиться лично без свидетелей, что является серьезным неудобством. На протяжении веков это требование ограничивало практическую полезность криптографии. Изобретение в 1970-х годах криптографии с открытым ключом, решающей эту проблему, было необычайным прорывом.

Схема работает следующим образом. Алиса генерирует пару ключей. Закрытый ключ остается секретным, но открытый ключ может быть широко известен. Боб также генерирует пару ключей и публикует свой открытый ключ. Когда Алиса хочет отправить Бобу сообщение, она шифрует его с помощью открытого ключа Боба. Боб, который имеет закрытый ключ, является единственным, кто может расшифровать сообщение.



Рис. 27.1. Отправка сообщения, зашифрованного с использованием криптографии с открытым ключом

При отправке сообщения Алиса также может подписать его своим секретным ключом. Тогда Боб сможет использовать подпись Алисы и ее открытый ключ для подтверждения подлинности этого сообщения. Данный процесс, упрощенный здесь для ясности, называется *цифровой подписью*. Он позволяет доказать, что именно Алиса, а не Мэллори, отправила полученное сообщение.

Первой публично доступной криптосистемой с открытым ключом был метод обмена ключами Диффи–Хеллмана–Меркла (Diffie–Hellmann–Merkle). Вскоре после этого известной командой, состоящей из Рона Ривеста (Ron Rivest), Ади Шамира (Adi Shamir) и Леонарда Адлемана (Leonard Adleman), была разработана криптосистема с открытым ключом под названием RSA. Эти методы и были положены в основу современной сетевой безопасности.

Шифры с открытым ключом, также называемые *асимметричными шифрами*, основываются на математической концепции односторонних функций с потайным входом (trapdoor function), значение которых легко вычислить, но восстановить алгоритм их работы сложно и дорого. Невысокая скорость работы асимметричных шифров обычно делает их непрактичными для шифрования больших объемов данных. Поэтому они часто сочетаются с симметричными шифрами, чтобы реализовать преимущества обоих. Открытые ключи применяются для установки сеанса связи и обмена секретным симметричным ключом, который далее используется для шифрования текущего сеанса обмена данными.

Инфраструктура с открытым ключом

Организация надежного, заслуживающего доверия способа регистрации и распространения открытых ключей — сложное дело. Если Алиса хочет отправить Бобу личное сообщение, она должна быть уверена, что имеющийся у нее открытый ключ Боба на самом деле принадлежит ему, а не Мэллори. Проверка подлинности открытых ключей в интернет-масштабе — грандиозная проблема.

Одной из основных концепций, реализованных в программе PGP, является так называемая *сеть доверия*. Это сеть организаций, которые доверяют друг другу в разной степени. Следуя косвенным цепочкам доверия вне вашей личной сети, вы можете установить, что открытый ключ заслуживает доверия с разумной степенью уверенности. К сожалению, интерес широкой общественности к участию в подписании ключей и культивированию сети криптодрузей был, скажем так, довольно далек от энтузиазма, о чем свидетельствует канувшая в лету популярность PGP.

В инфраструктуре с открытым ключом, используемой для реализации протокола TLS в Интернете, проблема доверия решена за счет привлечения третьей стороны, называемой Центром сертификации (Certificate Authority — CA), которая может пору-

читься за достоверность открытых ключей. Алиса и Боб могут не знать друг друга, но оба они доверяют CA и знают открытый ключ CA. Центр сертификации подписывает сертификаты для открытых ключей Алисы и Боба собственным закрытым ключом. Затем Алиса и Боб могут проверить поручительство CA, чтобы убедиться, что ключи являются подлинными.

Сертификаты крупных центров сертификации, таких как GeoTrust и VeriSign, поставляются с дистрибутивами операционных систем. Когда клиент начинает зашифрованный сеанс, он видит, что сертификат партнера был подписан авторитетным органом, уже указанным в надежном локальном хранилище клиента. Следовательно, клиент может доверять подписи Центра сертификации и открытому ключу партнера. Эта схема изображена на рис. 27.2.



Рис. 27.2. Инфраструктура с открытым ключом для Интернета

Органы сертификации взимают плату за подписание сертификатов, цена которых устанавливается в соответствии с их репутацией, рыночными условиями и различными характеристиками сертификата. Некоторые варианты, например так называемые *универсальные сертификаты для всех поддоменов* или *расширенные сертификаты подлинности* с более строгой проверкой фона для запрашивающего лица, являются более дорогими.

В этой системе Центр сертификации обладает полным доверием. Первоначально существовало всего несколько доверенных центров сертификации, но со временем их стало значительно больше. Современные настольные и мобильные операционные системы по умолчанию доверяют сотням сертификатов. Поэтому сами центры сертификации являются объектами высокой ценности для злоумышленников, которые хотели бы использовать их закрытый ключ для подписания сертификатов собственной разработки.

Если Центр сертификации взломан, вся система доверия оказывается скомпрометированной. Известно, что некоторые центры сертификации были скомпрометированы злоумышленниками, а в других широко обсуждаемых случаях стало известно, что центры сертификации пошли на сговор с правительствами. Мы рекомендуем читателям тщательно выбирать центры сертификации при покупке услуг по подписке сертификатов.

В 2016 году была запущена бесплатная служба Let's Encrypt, спонсируемая такими организациями, как Electronic Frontier Foundation, Mozilla Foundation, Cisco Systems, юридическая школа Stanford и Linux Foundation, которая выпускает сертификаты через автоматизированную систему. К концу 2016 года эта служба выдала более 24 миллионов сертификатов. Учитывая широко известные проблемы, связанные с коммерческими центрами сертификации, мы рекомендуем Let's Encrypt как "возможно безопасную" бесплатную альтернативу.

Вы можете легко организовать собственный центр сертификации с помощью протокола OpenSSL, импортировать сертификат в надежное хранилище в своей организации, а затем выдавать сертификаты от имени этого органа. Это обычная практика защиты служб во внутренней сети, где организация имеет полный контроль над хранилищем доверенных сертификатов. Более подробно данная информация изложена ниже.

Организации должны проявлять осторожность, принимая решение о внедрении собственных сертификатов на выпускаемых компанией машинах. Если у вас нет такой же строгой и проверенной системы безопасности, как у профессиональных центров сертификации, вы можете легко создать уязвимость в вашей среде. В результате, если вы работаете в организации, которая устанавливает собственный сертификат в надежном хранилище своих компьютеров, возникает подозрение, что ваша собственная безопасность может быть скомпрометирована.

Протокол защиты транспортного уровня TLS

В протоколе TLS (Transport Layer Security) используется криптография с открытым ключом и инфраструктура с открытыми ключами для защиты сообщений между хостами сети. Это преемник протокола SSL (Secure Sockets Layer, или уровень защищенных сокетов). Как правило, аббревиатуры SSL и TLS используются как синонимы, хотя протокол SSL устарел и объявлен вышедшим из употребления. Протокол TLS в сочетании с HTTP известен как HTTPS.

Протокол TLS работает на отдельном уровне, который служит оберткой TCP-соединения. Он обеспечивает только безопасность соединения и не участвует в транзакции HTTP. Благодаря этой аккуратной архитектуре протокол TLS может защищать не только HTTP, но и другие протоколы, такие как SMTP.

Как только клиент и сервер установили защищенное соединение через TLS, содержимое обмена, включая URL-адрес и все заголовки, защищено с помощью шифрования. Злоумышленник может определить только хост и порт, поскольку эти данные видны в инкапсулируемых пакетах протокола TCP. В модели OSI TLS находится где-то между уровнями 4 и 7.

Хотя типичным сценарием использования является одностороннее шифрование TLS, в котором клиент ратифицирует сервер, все чаще используется двухсторонний протокол TLS, иногда называемый *протоколом взаимной аутентификации*. В этой схеме клиент должен предоставить серверу сертификат, подтверждающий его собственную личность. Это, например, похоже на то, как клиенты Netflix (телеприставки и все остальное, которое передает видео из Netflix) аутентифицируются в интерфейсе API Netflix.

Последняя версия TLS имеет номер 1.2. Отключите все версии SSL вместе с TLS версии 1.0 из-за известных уязвимостей. Версия TLS 1.3 активно развивается и содержит существенные изменения, которые будут иметь значительные последствия для некоторых отраслей.[6]

Криптографические хеш-функции

Хеш-функция обрабатывает входные данные любой длины и генерирует небольшое значение фиксированной длины, которое каким-то образом получается из этих данных. Выходное значение называется по-разному: хеш-значение, просто хеш, сводка, дайд-

[6] Представитель сектора финансовых услуг попытался повлиять на техническое решение в списке рассылки разработчиков TLS, но опоздал на два года. Озабоченность была отвергнута (см. поток электронных сообщений на сайте `goo.gl/uAEwPN`).

жест, контрольная сумма или отпечаток пальца. Хеш-функции детерминированы, поэтому, если вы вычисляете некоторую хеш-функцию для заданного входного значения, вы всегда будет получать один и тот же результат.

Поскольку хеши имеют фиксированную длину, существует только конечное число возможных выходных значений. Например, 8-битный хеш имеет только 2^8 (то есть 256) возможных выходных значений. Поэтому для некоторых входных значений неминуемо генерируется одно и то же значение хеш-функции. Это событие называется коллизией. Более длинные значения хеша уменьшают частоту коллизий, но никогда не могут полностью их устранить.

В программном обеспечении используются сотни различных хеш-функций, но подмножество, известное как криптографические хеш-функции, представляет особый интерес для системных администраторов и математиков. В этом контексте слово "криптографическая" означает "действительно хорошая". Эти хеш-функции имеют почти все необходимые свойства, которые требуются от хеш-функции.

- *Запутанность*. Каждый бит хеш-значения зависит от каждого бита входных данных. В среднем изменение одного бита входных данных должно привести к изменению 50% битов хеш-значения.
- *Псевдослучайность*. Хеш-значения должны быть неотличимыми от случайных данных. Конечно, хеш-значения не являются случайными; они генерируются детерминированно и воспроизводятся по входным данным. Однако они все равно должны быть похожи на случайные данные: они не должны иметь обнаруживаемой внутренней структуры, явной зависимости от входных данных и должны проходить все известные статистические тесты на случайность.
- *Необратимость*. При заданном хеш-значении должно быть невозможно вычислить другое входное значение, которое сгенерирует то же самое значение хеш-функции.

Имея достаточно качественный алгоритм хеширования и достаточно длинное хеш-значение, мы можем быть уверены, что два входных значения, которые генерируют одно и то же значение хеш-функции, фактически являются идентичными. Конечно, это невозможно доказать теоретически, потому что все хеши имеют коллизии. Тем не менее это можно сделать для любого желаемого уровня статистического доказательства, увеличивая длину хеш-значения.

С помощью криптографических хешей проверяется целостность данных. Они могут подтвердить, что данный файл конфигурации или двоичный код утилиты не был изменен, или что сообщение, подписанное корреспондентом электронной почты, не было модифицировано в пути. Например, чтобы убедиться, что в системах FreeBSD и Linux используются идентичные файлы конфигурации **sshd_config**, мы можем использовать следующие команды.

```
freebsd$ sha256 /etc/ssh/sshd_config
SHA256 (/etc/ssh/sshd_config) = 3ef2d95099363d ... 8c14f63c5b9f741ea8d5

linux$ sha256sum /etc/ssh/sshd_config
3ef2d95099363d ... 8c14f63c5b9f741ea8d5 /etc/ssh/sshd_config
```

Здесь мы опустили часть хеш-значений. Как обычно для большинства случаев использования, выходные значения приведены в шестнадцатеричной системе счисления. Но имейте в виду, что фактические значения хеша — это всего лишь совокупности двоичных битов и эти данные могут быть представлены несколькими способами.

Существует множество криптографических алгоритмов хеширования, но единственными, рекомендуемыми для общего использования на данном этапе, являются семейства SHA-2 и SHA-3 (Secure Hash Algorithm), которые были выбраны институтом NIST после детального анализа.[7]

Для каждого из этих алгоритмов существуют варианты с разными длинами хеш-значений. Например, SHA3-512 представляет собой алгоритм SHA-3, сконфигурированный для генерации 512-битового хеш-значения. Алгоритм SHA без номера версии, например SHA-256, всегда относится к члену семейства SHA-2.

Другой распространенный криптографический алгоритм хеширования, MD5, по-прежнему широко поддерживается криптографическим программным обеспечением. Однако известно, что он уязвим для искусственных коллизий, когда несколько входных данных дают одно и то же значение хеш-функции. Хотя MD5 больше не считается безопасным для использования в криптографии, он по-прежнему является хорошо управляемой хеш-функцией и теоретически подходит для использования в приложениях с низкой степенью защиты. Но зачем такие сложности? Просто используйте SHA.

В проектах с открытым исходным кодом часто публикуются хеши файлов, которые распространяются среди сообщества пользователей. Например, в проекте OpenSSH распространяются сигнатуры PGP tar-архивов для проверки, которые вычисляются с помощью криптографических хеш-функций. Чтобы проверить подлинность и целостность полученных файлов, нужно вычислить их хеш-значение и сравнить с опубликованным значением хеша. Тем самым гарантируется, что вы получили полную и немодифицированную копию и без ошибок в битах.

Хеш-функции также используются в качестве компонента кодов аутентификации сообщений, или кодов MAC (message authentication code). Значение хеша внутри кода MAC подписывается закрытым ключом. В процесс проверки сначала верифицируется подлинность самого MAC (путем его расшифровки с помощью соответствующего открытого ключа) и целостности содержимого (путем вычисления хеша его содержимого). Схемы MAC часто играют важную роль в обеспечении безопасности веб-приложений.

Генерация случайных чисел

Криптографическим системам нужен источник случайных чисел, из которых можно генерировать ключи. Но для алгоритмов нехарактерно случайное и непредсказуемое поведение. Что делать?

Золотым стандартом для генерации случайных чисел являются данные, полученные из физически случайных процессов, таких как радиоактивный распад и радиочастотный шум от ядра галактики. Такие источники реально существуют: см. сайт `random.org` для доступа к некоторым фактическим случайным данным и получения объяснения того, как это происходит. Это интересно, но, к сожалению, не очень полезно для повседневной криптографии. Для генерации последовательностей случайных данных в традиционных генераторах псевдослучайных чисел используются методы, аналогичные методам хеш-функций. Однако этот процесс детерминирован. Если вы знаете внутреннее состояние генератора случайных чисел, вы можете точно воспроизвести последовательность его вывода. Следовательно, это плохой вариант для криптографии. Если вы генерируете случайный 2048-битовый ключ, вам требуется 2048 случайных битов, а не 128 бит состояния генератора чисел, на основе которых алгоритмически были получены эти 2048 бит.

[7] Алгоритм SHA-1 был скомпрометирован и больше не должен использоваться.

К счастью, разработчики ядра систем UNIX приложили немало усилий для фиксации тонких изменений в поведении системы и использования их в качестве источников случайных чисел. Для этой цели используются любые подходящие данные, начиная с временных меток получения сетевых пакетов, и заканчивая моментами времени возникновения аппаратных прерываний от таких слабо предсказуемых устройств, как дисковые накопители. Даже в среде виртуальных и облачных серверов все еще достаточно энтропии, чтобы сгенерировать достаточно случайные числа.

Данные со всех этих источников поступают во вторичный генератор псевдослучайных чисел, который гарантирует, что выходной поток случайных данных будет иметь корректные статистические свойства. Этот поток данных затем предоставляется для использования через драйвер устройства. В системах Linux и FreeBSD он представлен как **/dev/random** и **/dev/urandom**.

О случайных числах необходимо знать два факта.

- Все, что работает в пользовательском пространстве, не может конкурировать с качеством генератора случайных чисел ядра. Никогда не позволяйте криптографическому программному обеспечению генерировать собственные случайные данные; всегда убедитесь, что он использует случайные данные из файлов устройств **/dev/random** или **/dev/urandom**. Большинство программ делают это по умолчанию.
- Выбор файла **/dev/random** или **/dev/urandom** является предметом спора, и, к сожалению, аргументы носят слишком тонкий и математический характер, чтобы обобщить их здесь. Короткий вариант ответа заключается в том, что устройство **/dev/random** в системе Linux может вообще не генерировать данные, если ядро распознает, что система не накапливает достаточную энтропию. Либо изучите этот вопрос и сознательно выбирайте одну из сторон, либо просто используйте устройство **/dev/urandom** и не беспокойтесь об этой проблеме. Большинство экспертов, похоже, рекомендуют последний вариант. Пользователи системы FreeBSD освобождаются от проблемы выбора, поскольку файлы устройств **/dev/random** и **/dev/urandom** в ядре идентичны.

Выбор криптографического программного обеспечения

Есть веские основания очень подозрительно относиться ко всему программному обеспечению систем безопасности и особенно к пакетам, которые предоставляют криптографические услуги. По слухам, правительства больших и влиятельных государств пытались оказать влияние на разработчиков еще на этапе проектирования криптографических протоколов и алгоритмов. Можно предположить, что несколько хорошо финансируемых групп стремятся скомпрометировать любой криптографический проект, который имеет шансы на успех. Тем не менее мы доверяем программному обеспечению с открытым исходным кодом больше, чем закрытому. Такие проекты, как OpenSSL, имеют большую историю серьезных уязвимостей, но эти проблемы были обнаружены, смягчены их последствия и опубликованы результаты исправлений на профильном открытом форуме. История проекта и исходный код изучаются тысячами людей.

Никогда не применяйте доморощенные криптографические программы! Просто используйте готовые библиотеки правильно, хотя это и не всегда просто! Заказанные на стороне криптосистемы обречены на уязвимость.

Команда openssl

Команда **openssl** — это многоуровневый TLS-инструмент администратора. Вы можете использовать его для генерации пар открытого-закрытого ключа, шифрования и дешифрования файлов, изучения криптографических свойств удаленных систем, создания собственных центров сертификации, преобразования файловых форматов и множества других криптографических операций.

Подготовка ключей и сертификатов

Одной из наиболее распространенных административных функций команды **openssl** является подготовка сертификатов для их подписания в центре сертификации. Начнем с создания 2048-битового закрытого ключа:

```
$ openssl genrsa -out admin.com.key 2048
```

Используйте закрытый ключ для создания запроса на подпись сертификата. При этом команда **openssl** запросит метаданные в виде отличительных имен (Distinguished Name, DN) для включения в запрос. Эту информацию также можно указать в файле ответов, чтобы не вводить ее каждый раз вручную из командной строки, как показано ниже.

```
$ openssl req -new -sha256 -key admin.com.key -out admin.com.csr
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:Oregon
Locality Name (eg, city) []:Portland
Organization Name (eg, company) [Internet Widgits Pty Ltd]:ULSAH5E
Organizational Unit Name (eg, section) []:Crypto division
Common Name (e.g. server FQDN or YOUR name) []:server.admin.com
```

Отправьте содержимое файла **admin.com.csr** в центр сертификации, который выполнит процесс проверки, в результате которого будет подтверждено, что вы связаны с доменом, для которого получаете сертификат. Процесс верификации обычно выполняется путем отправки сообщения по электронной почте на один из адресов в этом домене. После этого вам возвращается подписанный сертификат. Затем вы можете использовать файл **admin.com.key** и сертификат, подписанный центром сертификации, в конфигурации вашего веб-сервера.

В большинство из этих полей можно ввести произвольный текст, но поле общего имени **Common Name** имеет важное значение. Оно должно соответствовать имени поддомена, который вы хотите обслуживать. Если, например, вы хотите работать через TLS с сайтом **www.admin.com**, укажите его доменное имя в поле **Common Name**. В этом поле вы можете указать несколько имен для одного сертификата или символ подстановки, который соответствует всем именам в поддомене: например ***.admin.com**.

После того как вы получите сертификат, вы можете изучить его свойства. Ниже приведены некоторые сведения об универсальном сертификате для поддомена ***.google.com**.

```
$ openssl x509 -noout -text -in google.com.pem
depth=2 /C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
...
       Signature Algorithm: sha256WithRSAEncryption
       Issuer: C=US, O=Google Inc, CN=Google Internet Authority G2
       Validity
          Not Before: Dec 15 13:48:27 2016 GMT
          Not After : Mar 9 13:35:00 2017 GMT
       Subject: C=US, ST=California, L=Mountain View, O=Google Inc,
   CN=*.google.com
```

Срок действия этого сертификата — с 15 декабря 2016 года по 9 марта 2017 года. Клиенты, которые подключаются к сайту после истечения этого временного интервала, будут видеть сообщение о том, что сертификат больше недействителен. Отслеживание срока действия сертификата и своевременное его обновление обычно является обязанностью системного администратора.

Отладка TLS-сеанса с сервером

Используйте команду **openssl s_client** для изучения подробностей TLS-соединения с удаленным сервером. Эта информация может оказаться весьма полезной при отладке веб-серверов, имеющих проблемы с сертификатами. Например, для проверки свойств TLS домена **google.com** необходимо выполнить приведенную ниже команду (ее вывод сокращен для краткости).

```
$ openssl s_client -connect google.com:443
---
New, TLSv1/SSLv3, Cipher is AES128-SHA
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
SSL-Session:
   Protocol        : TLSv1
   Cipher          : AES128-SHA
   Session-ID      : 4F72DC56EE4E80568F7E0EF9F59C8D7855C87F366B49BF1D9808...
   Session-ID-ctx  :
   Master-Key       : 095C6D8AF9B6B81E3E16BA05C0C9ACFACD72EF3335A32B86F3D3...
   Key-Arg          : None
   Start Time       : 1484163220
   Timeout          : 300 (sec)
   Verify return code : 0 (ok)
---
```

Вы можете использовать команду **openssl s_client**, чтобы проверить, какие версии протокола TLS поддерживаются сервером (см. также команду **openssl s_server**, которая запускает универсальный TLS-сервер). Это может пригодиться для тестирования и отладки клиентов.

PGP: довольно хорошая конфиденциальность

Пакет PGP (Pretty Good Privacy), написанный Филом Циммерманом (Phil Zimmermann), содержит криптографические утилиты, предназначенные в основном для обеспечения безопасности систем электронной почты. Он используется для шифрования данных, генерирования сигнатур и проверки источников происхождения файлов и сообщений.

🕮 Более подробная информация о защите электронной почты приведена в разделе 18.4.

Пакет PGP имеет интересную историю, которая включает в себя судебные иски, криминальные расследования и приватизацию фрагментов исходного пакета PGP. Недавно пакет PGP был подвергнут жесткой критике за то, что он раскрывал слишком много метаданных в своих наиболее распространенных режимах использования. Афиширование среди прочего длины сообщения, его получателей, а также использование незашифрованного хранилища для черновиков — все это слабые места, которые могут быть ис-

пользованы злоумышленниками, особенно субъектами национальных государственных структур, обладающими огромными ресурсами. Тем не менее, использование PGP попрежнему предпочтительнее, чем обмен открытыми сообщениями.

В настоящее время формат файла и протоколы PGP стандартизованы организацией IETF под именем OpenPGP, и уже существует несколько реализаций предложенного стандарта. Проект GNU обеспечивает превосходную, свободную и широко применяемую реализацию под названием GnuPG, которую можно найти по адресу `gnupg.org`. Для ясности мы будем называть эту систему просто PGP, даже если ее отдельные реализации имеют другие названия.

Вероятно, PGP — наиболее популярная криптографическая программа. К сожалению, для того чтобы работать с ней в системе UNIX/Linux, нужно быть специалистом в области криптографии. Пакет PGP может оказаться полезным, но мы не рекомендуем заставлять пользователей работать с ним из-за нетривиальности процесса обучения. Гораздо удобнее использовать Windows-версию PGP, чем команду **pgp** с ее многочисленными режимами работы.

Программные пакеты, распространяемые через Интернет, часто содержат файл сигнатуры PGP, подтверждающий их подлинность и целостность. Но проверить эти сигнатуры не так-то легко людям, не являющимся фанатиками PGP. Не потому, что процесс проверки сложен, а потому, что при использовании PGP истинная безопасность достигается только путем создания персональной коллекции открытых ключей тех людей, которые были проверены напрямую. Загрузка дистрибутива вместе с открытым ключом и файлом сигнатуры безопасна примерно в той же степени, что и загрузка самого дистрибутива без них.

Некоторые почтовые клиенты имеют надстройки, обеспечивающие простой графический пользовательский интерфейс для расшифровки входящих и шифрования исходящих сообщений. Пользователи браузера Google Chrome могут инсталлировать расширение, ориентированное на конечного пользователя, поддерживающее PGP для почты Gmail.

Kerberos: унифицированный подход к сетевой безопасности

Система Kerberos, разработанная в Массачусетсском технологическом институте (MIT), ориентирована на решение задач, связанных с обеспечением безопасности компьютерных сетей. Kerberos — это система аутентификации, предоставляющая "гарантию" того, что пользователи и служебные программы действительно являются теми, за кого себя выдают. Никаких дополнительных проверок и шифрования передаваемых данных не предусмотрено.

Используя симметричную и асимметричную криптографию, протокол Kerberos создает вложенные наборы идентификаторов, называемых *билетами* или *мандатами*. Мандаты передаются по сети с целью подтверждения личности пользователя и предоставления ему доступа к сетевым службам. В каждой организации, где используется Kerberos, должен выделяться хотя бы один физически защищенный компьютер, называемый *сервером аутентификации*, для запуска демона Kerberos. Этот демон выдает мандаты пользователям и служебным программам, требующим аутентификации, на основании предъявляемых ими "удостоверений", в частности паролей.

По сути, протокол Kerberos улучшает традиционную схему парольной защиты всего двумя способами: пароли никогда не передаются по сети в незашифрованном виде,

и пользователи избавляются от необходимости многократно вводить пароли. Все это позволяет создать благоприятную среду парольной защиты сетевых служб.

Сообщество пользователей Kerberos может похвастаться одним из самых толковых и оригинальных документов, посвященных криптосистемам, — "Designing an Authentication System: a Dialogue in Four Scene" (Проектирование системы аутентификации: диалог в четырех актах) Билла Брайанта (Bill Bryant). Его будет интересно почитать любому, кто интересуется темой криптографии. Документ можно найти по адресу: `web.mit.edu/kerberos/www/dialogue.html`.

Лучше использовать Kerberos, чем вообще отказаться от средств защиты. К сожалению, Kerberos нельзя назвать полностью безопасной системой, а ее инсталляция и запуск — не самое приятное занятие. В то же время она не заменяет собой другие меры безопасности, описанные в данной главе.

К сожалению (хотя, возможно, вполне предсказуемо), система Kerberos, распространяемая как часть службы Active Directory системы Windows, использует запатентованные, недокументированные расширения. Как следствие, она плохо взаимодействует с традиционной версией системы, в основу которой положен код MIT. К счастью, демон **sssd** позволяет системам UNIX и Linux взаимодействовать с версией системы Kerberos для службы Active Directory. Более подробную информацию о службе каталогов можно найти в разделе 17.3.

27.7. Система SSH

Система SSH (Secure SHell, или защищенная командная оболочка), разработанная Тату Юлёненом (Tatu Yl nen), является протоколом для удаленного входа в систему и обеспечения сетевых услуг в ненадежной сети. Возможности SSH включают в себя удаленное выполнение команд, доступ к командной оболочке, передачу файлов, переадресацию портов, услуги сетевого прокси и даже туннелирование VPN. Это незаменимый инструмент, настоящий швейцарский армейский нож для системных администраторов.

SSH — это протокол типа клиент-сервер, в котором используется криптография для аутентификации, а также обеспечения конфиденциальности и целостности сообщений, передаваемых между двумя хостами. Он поддерживает алгоритмическую гибкость и позволяет обновлять и заменять основные криптографические протоколы по мере развития отрасли. Система SSH документируется как группа связанных протоколов в RFC 4250–4256.

В этом разделе мы обсудим OpenSSH, реализацию SSH с открытым исходным кодом, которая включена и активирована по умолчанию почти в каждой версии UNIX и Linux. Мы также упомянем несколько альтернативных решений для предприимчивых и непредвзятых людей.

Основы OpenSSH

Набор программ OpenSSH был разработан в рамках проекта OpenBSD в 1999 году и с тех пор поддерживается этой организацией. Программный пакет состоит из нескольких команд:

- **ssh** — клиент;
- **sshd** — демон сервера;

- **ssh-keygen** — команда для генерации пар открытых-закрытых ключей;
- **ssh-add** и **ssh-agent** — утилиты для управления ключами аутентификации;
- **ssh-keyscan** — для извлечения открытых ключей с серверов;
- **sftp-server** — серверный процесс для передачи файлов через протокол SFTP;
- **sftp** и **scp** — утилиты для передачи файлов.

В наиболее распространенном сценарии использования клиент устанавливает соединение с сервером, аутентифицирует себя и впоследствии запускает оболочку для выполнения команд. Методы аутентификации согласовываются в соответствии со взаимной поддержкой и предпочтениями клиента и сервера. Многие пользователи могут входить в систему одновременно. Для каждого пользователя назначается псевдотерминал, вход и выход которого подключаются к удаленной системе.

Чтобы инициировать этот процесс, пользователь вызывает команду **ssh**, указывая в ее параметрах удаленный хост в качестве первого аргумента:

```
$ ssh server.admin.com
```

Команда **ssh** пытается установить TCP-соединение через порт 22, стандартный SSH-порт, назначенный реестром IANA. После установки соединения сервер отправляет свой открытый ключ для проверки. Если сервер еще не известен и не вызывает доверия, команда **ssh** запрашивает у пользователя подтверждение сервера, представляя хеш открытого ключа сервера, называемого *ключевым отпечатком*.

```
The authenticity of host 'server.admin.com' can't be established.
ECDSA key fingerprint is SHA256:quLdFoXBI6OpU6HwnUy/K50cR9UuU.
Are you sure you want to continue connecting (yes/no)?
```

Цель состоит в том, чтобы администратор сервера мог заранее передать пользователям ключ хоста. Затем пользователь сможет сравнить информацию, полученную им от администратора, с отпечатком сервера, выведенным при первом подключении. Если они совпадают, то подлинность хоста будет доказана.

Как только пользователь примет ключ, к файлу **~/.ssh/known_hosts** для будущего использования добавляется его отпечаток. Команда **ssh** не будет упоминать ключ сервера еще раз, если ключ не изменится. В противном случае **ssh** выведет предупреждающее сообщение о том, что личность сервера изменилась.

На практике эта процедура верификации сервера обычно игнорируется. Администраторы редко посылают ключ хоста пользователям, и пользователи слепо принимают ключ хоста без проверки. Этот процесс штамповки новых ключей хоста подвергает пользователей атакам "человек посредине". К счастью, этот процесс может быть автоматизирован и оптимизирован. Мы обсудим проблему проверки ключа хоста с помощью протокола SSHFP немного ниже.

После того как ключ хоста был принят, сервер указывает методы проверки подлинности, которые он поддерживает. Пакет программ OpenSSH реализует все методы, описанные в документах RFC SSH, включая простую аутентификацию по паролю UNIX, доверенные хосты, открытые ключи, GSSAPI для интеграции с протоколом Kerberos и гибкую схему запрос-ответ для поддержки подключаемых модулей аутентификации и одноразовых паролей. Из них наиболее часто используется аутентификация с открытым ключом, которая рекомендуется для большинства организаций. Она обеспечивает лучший баланс между безопасностью и удобством. Мы более подробно обсудим использование открытых ключей для аутентификации через SSH немного ниже.

Команды **ssh** и **sshd** могут быть настроены для разных потребностей и видов безопасности. Файлы конфигурации находятся в каталоге **/etc/ssh**, который является нехарактерно стандартным местом среди всех разновидностей UNIX и Linux. В табл. 27.1 перечислены файлы, находящиеся в этом каталоге.

Таблица 27.1. Файлы конфигурации в каталоге /etc/ssh

Файл	Разрешения	Содержимое
ssh_config	0644	Конфигурация клиента для всего сайта
sshd_config	0644	Конфигурация сервера
moduli	0644	Основные номера и генераторы для обмена ключами DH
***_key**	0600	Закрытые ключи для каждого алгоритма, поддерживаемого сервером
***_key.pub**	0644	Открытый ключ, соответствующий каждому закрытому ключу

Кроме каталога **/etc/ssh**, пакет программ OpenSSH использует каталог **~/.ssh** для хранения открытых и закрытых ключей, переопределения конфигурации каждого клиента и для нескольких других целей. Каталог **~/.ssh** игнорируется, если его разрешения не равны 0700.

Пакет OpenSSH имеет заслуживающий уважения, хотя и не безупречный послужной список уязвимых мест системы безопасности. Согласно базе данных CVE (cve.mitre.org), в ранних версиях было обнаружено несколько критических уязвимостей. Последняя из них была задокументирована в 2006 году. Часто появляются сообщения о случаях отказа в обслуживании и использования уязвимостей, но большинство из них считается неопасными. Тем не менее было бы разумно обновлять пакеты OpenSSH в рамках вашего регулярного графика обновлений.

Клиент ssh

Начало работы с командой **ssh** является простым, но ее сила и универсальность проявляются в многочисленных вариантах. С помощью конфигурации можно выбрать криптографические алгоритмы и шифры, создать удобные псевдонимы хостов, настроить переадресацию портов и многое другое.

Основной синтаксис:

ssh [*опции*] [*имя_пользователя*@]*хост* [*команда*]

Например, чтобы проверить занимаемое каталогом **/var/log** дисковое пространство, можно выполнить команду

```
$ ssh server.admin.com "df -h /var/log"
```

Если вы укажете *команду*, то утилита **ssh** аутентифицирует себя на хосте, выполнит эту команду и выйдет, не открывая интерактивную оболочку. Если вы не укажете *имя_пользователя*, то утилита **ssh** использует ваше локальное имя пользователя на удаленном хосте.

Утилита **ssh** считывает глобальные параметры конфигурации из файла **/etc/ssh/ssh_config** и обрабатывает дополнительные параметры и переопределения для каждого пользователя из файла **~/.ssh/config**. В табл. 27.2 перечислены некоторые из наиболее интересных параметров, которые вы можете указать в этих файлах. Мы обсудим эти варианты более подробно ниже в данной главе.

Таблица 27.2. Параметры конфигурации клиента `ssh`

Вариант	Значение	По умолчанию
`AddKeysToAgent`	Автоматически добавлять ключи к команде **`ssh-agent`**	`no`
`ConnectTimeout`	Время ожидания подключения в секундах	Варьируется [a]
`ControlMaster`	Разрешить мультиплексирование соединения	`no`
`DynamicForward`	Настройка прокси-сервера SOCKS4 или SOCKS5	-
`ForwardAgent`	Включить пересылку **`ssh-agent`**	`no`
`Host`	Маркер для нового псевдонима хоста	-
`IdentityFile`	Путь к закрытому ключу аутентификации	**`~/.ssh/id_rsa`**[6]
`Port`	Порт для подключения	`22`
`RequestTTY`	Укажите, требуется ли устройство TTY	`auto`
`ServerAliveInterval`	Выполнять периодическое зондирование подключения к серверу	`0` (отключено)
`StrictHostKeyChecking`	Требовать (`yes`) или игнорировать (`no`) ключи хоста	`ask`

[a] Значение по умолчанию зависит от стандартных значений протокола TCP, установленных в ядре, которые от системы к системе сильно различаются.

[6] Точное имя зависит от алгоритма аутентификации. По умолчанию все ключи начинаются с **`id_`**.

Когда утилита **`ssh`** собирает окончательную конфигурацию, аргументы командной строки имеют приоритет над элементами, указанными в файле **`~/.ssh/config`**. Глобальная конфигурация, установленная в файле **`/etc/ssh/ssh_config`**, имеет наименьший приоритет.

Утилита **`ssh`** отправляет текущее имя пользователя в качестве имени входа, если другое значение не указано. Вы можете указать другое имя пользователя с помощью флага **`-l`** или синтаксиса **`@`**:

```
$ ssh -l hsolo server.admin.com
$ ssh hsolo@server.admin.com
```

Параметры клиента, для которых не предусмотрены отдельные аргументы, передаваемые утилите **`ssh`**, все таки могут быть установлены из командной строки с помощью флага **`-o`**. Например, вы можете отключить проверку хоста на для сервера:

```
$ ssh -o StrictHostKeyChecking=no server.admin.com
```

Флаг **`-v`** выводит отладочные сообщения. Укажите его несколько раз (максимум три), чтобы увеличить степень детализации. Этот флаг неоценим при отладке проблем с аутентификацией.

Для удобства утилита **`ssh`** возвращает код завершения выполнения удаленной команды. Используйте эту возможность для выявления ошибочных ситуаций при вызове утилиты **`ssh`** из сценариев.

Чтобы ознакомиться с доступными параметрами и функциями, обратитесь к справочным страницам **`man ssh`** и **`man ssh_config`**. Для получения краткой информации запустите команду **`ssh -h`**.

Аутентификация с помощью открытого ключа

Протокол OpenSSH (и протокол SSH в целом) может использовать криптографию с открытым ключом для аутентификации пользователей на удаленных системах. Как пользователь, вы сначала должны создать пару открытый-закрытый ключ. После этого

вы должны каким-то образом передать свой открытый ключ администратору удаленного сервера, который добавит его на сервере в файл **~/.ssh/authorized_keys**. Затем вы можете войти на удаленный сервер, выполнив команду **ssh** с удаленным именем пользователя и своим соответствующим закрытым ключом.

```
$ ssh -i ~/.ssh/id_ecdsa hsolo@server.admin.dom
```

Для создания пары ключей используйте утилиту **ssh-keygen**. Вы можете указать, какой криптографический алгоритм использовать, а также длину ключа в битах и другие характеристики. Например, для создания пары ключей ECDSA с 384-битовым размером эллиптической кривой можно выполнить следующую команду.

```
$ ssh-keygen -t ecdsa -b 384
Generating public/private ecdsa key pair.
Enter file in which to save the key (/home/ben/.ssh/id_ecdsa): <return>
Enter passphrase (empty for no passphrase): <return>
Enter same passphrase again: <return>
Your identification has been saved in /home/ben/.ssh/id_ecdsa.
Your public key has been saved in /home/ben/.ssh/id_ecdsa.pub.
The key fingerprint is:
SHA256:VRh6raUfpn3YdtMm7GURbIoyfcp/npbwhsmvsdrlhK4 ben
```

Открытый ключ (**~/.ssh/id_ecdsa.pub**) и файлы с закрытым ключом (**~/.ssh/id_ecdsa**) являются текстовым файлами в формате ASCII с кодировкой base64. Никогда не делитесь личным ключом! Утилита **ssh-keygen** правильно устанавливает права доступа к файлам, содержащим открытый и закрытый ключи, как 0644 и 0600 соответственно. В этом примере используется алгоритм ECDSA, но также можно применять параметр **-t rsa** с длиной ключа 2048 или 4096 бит.

Утилита **ssh-keygen** запрашивает необязательную кодовую фразу для шифрования закрытого ключа. Если вы используете кодовую фразу, вы должны будете ввести ее для дешифрования ключа, прежде чем утилита **ssh** сможет его прочитать. Кодовая фраза улучшает безопасность, поскольку процесс аутентификации получает дополнительный этап проверки: вы должны иметь файл ключа и знать кодовую фразу, которая расшифровывает его, прежде чем вы сможете аутентифицироваться.

Мы предлагаем установить кодовую фразу для всех привилегированных учетных записей (т.е. тех, у кого есть привилегии на запуск **sudo**). Если вам необходимо использовать ключ без кодовой фразы для активизации автоматического процесса аутентификации, ограничьте полномочия соответствующей учетной записи сервера.

Если вы администратор сервера и вам нужно добавить открытый ключ для нового пользователя, выполните следующие действия.

1. Убедитесь, что у пользователя есть активная учетная запись с корректной системной оболочкой.
2. Получите копию открытого ключа пользователя от самого пользователя.
3. Создайте пользовательский каталог **.ssh** с разрешениями 0700.
4. Добавьте открытый ключ в файл *~пользователь*/**.ssh/authorized_keys** и установите разрешения для этого файла 0600.

Например, если открытый ключ пользователя **hsolo** был сохранен в файле **/tmp/hsolo.pub**, процесс будет выглядеть так.

```
$ grep hsolo /etc/passwd
hsolo: x: 503: 503: Han Solo:/home/hsolo:/bin/bash
```

```
$ mkdir -p ~hsolo/.ssh && chmod 0700 ~hsolo/.ssh
$ cat /tmp/hsolo.pub >> ~hsolo/.ssh/authorized_keys
$ chmod 0600 ~hsolo/.ssh/authorized_keys
```

Если вам нужно проделать эти действия более одного раза, то вполне разумно создать специальный сценарий для выполнения данной процедуры. Системы управления конфигурациями, такие как Ansible и Chef, очень хорошо справляются с этой задачей.

Демон ssh-agent

Демон **ssh-agent** кеширует дешифрованные закрытые ключи. Для этого вы должны загрузить свои личные ключи в агент, после чего утилита **ssh** сможет автоматически их использовать при подключении к новым серверам, что упрощает процесс подключения.

Используйте команду **ssh-add** для загрузки нового ключа. Если для ключа требуется кодовая фраза, вам будет предложено ее ввести. Чтобы отобразить текущие загруженные ключи, запустите команду **ssh-agent -l**.

```
$ ssh-add ~/.ssh/id_ecdsa
Enter passphrase for ~/.ssh/id_ecdsa: <кодовая фраза>
Identity added: ~/.ssh/id_ecdsa (~/.ssh/id_ecdsa)
$ ssh-add -l
384 SHA256: VRbIoyfcp /npbwhsmvsdrlhK4 ~/.ssh/id_ecdsa (ECDSA)
```

Вы можете одновременно активировать сразу несколько ключей. Для удаления ключа используется команда **ssh-add -d** *путь*, а для очистки всех загруженных ключей — команда **ssh-add -D**.

Как ни странно, чтобы удалить закрытый ключ из агента, открытый ключ должен находиться в том же каталоге и иметь то же имя файла, но с расширением **.pub**. Если открытый ключ недоступен, вы можете получить запутанное сообщение о том, что ключ не существует.

```
$ ssh-add -d ~ /.ssh/id_ecdsa
Bad key file /home/ben/.ssh/id_ecdsa: No such file or directory
```

Вы можете легко исправить эту проблему, извлекая открытый ключ с помощью утилиты **ssh-keygen** и сохранив его с ожидаемым именем файла. (Это извлечение возможно, потому что файл закрытого ключа содержит копию открытого ключа, а также закрытый ключ.)

```
$ key=/home/ben/.ssh/id_ecdsa
$ ssh-keygen -yf $key > $key.pub
Enter passphrase: <кодовая фраза>
```

Утилита **ssh-agent** еще более полезна, когда вы используете ее возможность пересылки ключей агентов, что делает загруженные ключи доступными для удаленных хостов, в то время как вы входите в систему через оболочку **ssh**. Вы можете использовать эту функцию для перехода с одного сервера на другой без копирования вашего закрытого ключа на удаленные системы (рис. 27.3).

Чтобы активизировать возможность пересылки ключа агента, добавьте параметр `ForwardAgent yes` в свой файл **~/.ssh.config** или используйте команду **ssh -A**.

Используйте пересылку ключей только на серверах, которым вы доверяете. Любой, кто контролирует сервер, на который выполняется пересылка, может действовать от вашего имени и получить доступ к удаленным системам. Он не сможет читать ваши личные ключи напрямую, но сможет использовать любые ключи, которые доступны через агент пересылки.



Рис. 27.3. Пересылка ключа агента с помощью утилиты ***ssh-agent***

Псевдонимы хостов в файле ~/.ssh/config

Вы, несомненно, столкнетесь с множеством различных конфигураций оболочки SSH, если будете взаимодействовать с большим количеством серверов или управлять ими. Чтобы упростить вашу жизнь, файл **~/.ssh/config** позволяет настраивать псевдонимы и переопределять параметры отдельных хостов.

Например, рассмотрим две системы. Первая — это веб-сервер с IP-адресом 54.84.253.153, где демон **sshd** прослушивает порт 2222. Допустим, что ваше имя пользователя на этом сервере — **han**, и у вас есть закрытый ключ для аутентификации. Другой сервер — debian.admin.com, где ваше имя пользователя — **hsolo**. Вы предпочитаете полностью отключать аутентификацию по паролю, но сервер Debian требует ее.

Чтобы подключиться к этим серверам из командной строки, вы можете использовать команды с такими параметрами.

```
$ ssh -l han -p 2222 -i /home/han/.ssh/id_ecdsa 54.84.253.153
$ ssh -l hsolo debian.admin.com
```

В данном случае в настройках вашего клиента лучше всего выключить аутентификацию по паролю (она включена по умолчанию), потому что каждый раз вводить в командной строке параметр **-o PasswordAuthentication=no** слишком обременительно.

Следующие настройки из файла **~/.ssh/config** задают псевдонимы для этих хостов и дополнительно позволяют отключать аутентификацию по паролю по умолчанию.

```
PasswordAuthentication no
Host web
   HostName 54.84.253.153
   User han
   IdentityFile /home/han/.ssh/id_ecdsa
   ForwardAgent yes
   Port 2222
Host debian
   Hostname debian.admin.com
   User hsolo
   PasswordAuthentication yes
```

Теперь вы можете использовать гораздо более простые команды **ssh web** и **ssh debian** для доступа к этим хостам. Клиент считывает псевдонимы и автоматически устанавливает параметры для каждой системы. Утилита **ssh** также понимает некоторые базовые шаблоны для сопоставления хостов. Например, так.

```
Host *
   ServerAliveInterval 30m
   ServerAliveCountMax 1
Host 172.20.*
   User luke
```

В этом примере утилита **ssh** разорвет соединение с любым из серверов, если оно не будет активно более 30 мин. Она также использует имя пользователя **luke** при подключении к хостам в сети 172.20/16.

Псевдонимы хостов — это очень мощное средство, особенно если они используются в сочетании с другими возможностями протокола OpenSSH.

Мультиплексирование соединения

`ControlMaster` — отличная возможность утилиты **ssh**, которая позволяет мультиплексировать соединения, что значительно повышает скорость работы протокола SSH по каналам глобальной сети. Если эта возможность включена, то первое подключение к хосту создает сокет, который затем можно использовать повторно. При последующих подключениях будет использоваться этот общий сокет, но для них требуется выполнить отдельный процесс аутентификации.

Для включения мультиплексирования задайте параметры `ControlMaster`, `ControlPath` и `ControlPersist` в директиве `Host` определения псевдонима, как показано ниже

```
Host web
   HostName 54.84.253.153
   User han
   Port 2222
   ControlMaster auto
   ControlPath ~/.ssh/cm_socket/%r@%h:%p
   ControlPersist 30m
```

Параметр `ControlMaster auto` активирует эту функцию, а `ControlPath` создает сокет в указанном месте. Параметры подстановки, которые могут использоваться в имени файла `ControlPath`, описаны на справочной странице **man ssh_config**. В данном случае файл называется в соответствии с именем пользователя удаленного хоста, его IP-адресом и номером порта. При подключении к этому хосту будет создан приведенный ниже сокет.

```
$ ls -l ~ /.ssh/cm_socket/
srw ------- 1  ben  ben  0  Jan  2  15:22  han@54.84.253.153:22
```

Такой шаблон гарантирует уникальное имя файла для каждого сокета. Параметр `ControlPersist` сохраняет сокет в течение указанного периода времени, даже если первое (главное) соединение будет разорвано.

Вся настройка займет у вас 30 с. После этого сделайте пожертвование в фонд OpenBSD, чтобы поблагодарить их за внедрение мультиплексирования и экономию вашего времени.

Проброс портов

Еще одной полезной вспомогательной функцией протокола SSH является его способность безопасно туннелировать TCP-соединения через зашифрованный канал, тем самым обеспечивая возможность подключения к удаленным хостам, которые считаются

небезопасными или защищены брандмауэрами. На рис. 27.4 показано типичное использование туннеля SSH и поясняется, как это работает.

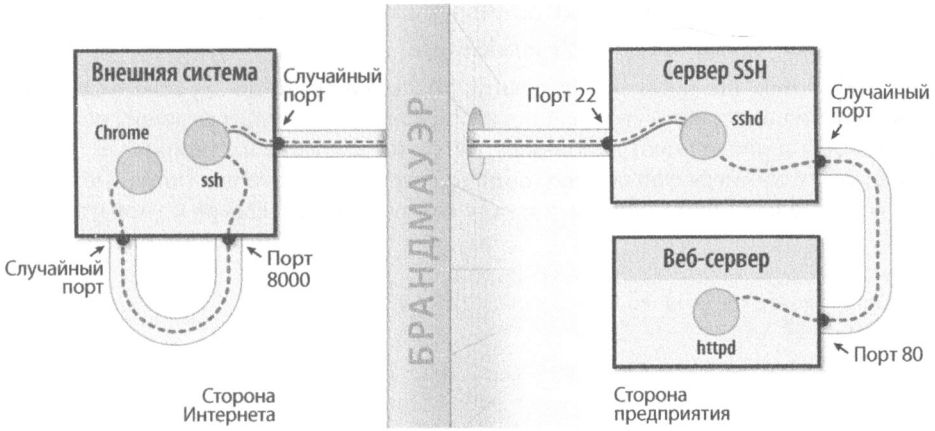


Рис. 27.4. SSH-туннель для протокола HTTP

В этом случае удаленный пользователь, допустим, Алиса, хочет установить HTTP-соединение с веб-сервером в сети предприятия. Доступ к этому хосту или порту 80 блокируется брандмауэром, но, имея доступ к SSH, Алиса может маршрутизировать соединение через SSH-сервер.

Чтобы настроить эту возможность, Алиса входит в систему на удаленном SSH-сервере с помощью утилиты **ssh**. При этом в командной строке **ssh** она указывает произвольный (но конкретный, в нашем случае 8000) локальный порт, данные с которого утилита **ssh** должна перенаправлять через защищенный туннель на порт 80 удаленного веб-сервера.

```
$ ssh -L 8000:webserver:80 server.admin.com
```

Все порты отправителя в этом примере помечены как случайные, так как программы сами выбирают произвольный порт, из которого можно инициировать соединения.

Чтобы получить доступ к веб-серверу предприятия, Алиса теперь может подключиться с помощью браузера к порту 8000 своего компьютера. Местный демон **sshd** примет это соединение и перенаправит трафик Алисы по существующему SSH-соединению к удаленному демону **sshd**. В свою очередь, удаленный демон **sshd** перенаправит данные Алисы на порт 80 веб-сервера.

Разумеется, туннели, подобные этим, могут стать преднамеренными или непреднамеренными лазейками. Используйте туннели с осторожностью, а также следите за несанкционированным использованием этой возможности пользователями. Вы можете отключить проброс портов в демоне **sshd**, задав параметр конфигурации `AllowTCPForwarding no`.

Демон sshd: сервер OpenSSH

Демон сервера OpenSSH, **sshd**, прослушивает порт 22 (по умолчанию) для соединений с клиентами. Его файл конфигурации, **/etc/ssh/sshd_config**, может похвастаться множеством опций, некоторые из них вам, возможно, потребуется настроить самостоятельно.

Демон **sshd** работает от имени пользователя **root**. Он создает непривилегированный дочерний процесс для каждого подключенного клиента с теми же правами, что и подключаемый пользователь. Если вы вносите изменения в файл **sshd_config**, то можете принудительно перезагрузить демон **sshd**, отправив сигнал **HUP** родительскому процессу.

```
$ sudo kill -HUP $(sudo cat /var/run/sshd.pid)
```

В системе Linux вы также можете запустить команду **sudo systemctl reload sshd**. Изменения вступают в силу для новых подключений. Существующие соединения не прерываются и продолжают использовать свои предыдущие настройки.

В следующем примере файла **sshd_config** содержит некоторые типичные параметры, настроенные ради поиска баланса между безопасностью сервера и удобством пользователя.

```
# Задайте inet для поддержки только протокола IPv4 или
# inet6 для поддержки только протокола IPv6
AddressFamily any

# Позволяет регистрироваться только указанным пользователям и группам.
# Довольно жесткие ограничения. После добавления/удаления пользователей
# требуется перезагрузка
AllowUsers foo bar hsolo
AllowGroups admins

# TCP-перенаправление очень удобно, но оно может быть небезопасно
AllowTcpForwarding yes

# Отображает сообщение перед аутентификацией пользователя.
# Очень важно в случае использования особых правовых норм или соблюдения
# оговоренных заранее требований
Banner /etc/banner

# Мы предпочитаем разрешить только аутентификацию с помощью открытого ключа
ChallengeResponseAuthentication no
PasswordAuthentication no
RSAAuthentication no
GSSAPIAuthentication no
HostbasedAuthentication no
PubkeyAuthentication yes

# Отключать неактивных клиентов через 5 мин.
ClientAliveInterval 300
ClientAliveCountMax 1

# Использовать сжатие всегда
Compression yes

# Не разрешать удаленным хостам пробрасывать порты
GatewayPorts no

# Фиксировать в журнале неудачные попытки регистрации
LogLevel VERBOSE

# Мы уменьшили заданное по умолчанию значение с 6 до 3
MaxAuthTries 3

# Не разрешать пользователю root напрямую подключаться к системе
# (заставляем использовать команду sudo)
PermitRootLogin no
```

```
# Запрещаем пользователям устанавливать свои параметры окружения
# в файле authorized_keys
PermitUserEnvironment no

# Использовать возможность "auth" для сообщений syslog
SyslogFacility AUTH

# Аннулировать сеанс связи в случае потери TCP-соединения
TCPKeepAlive no

# Не разрешать перенаправление данных системы X Window, если в вашей
# системе не используется эта графическая оболочка
X11Forwarding no
```

Мы рекомендуем вам явно указывать приемлемые шифры и алгоритмы обмена ключами. Мы не указываем здесь детали, потому что имена довольно длинные и в любом случае постоянно меняются. Следуйте инструкциям по настройке OpenSSH от компании Mozilla, которые можно найти на сайте `goo.gl/Xxgx7H` (глубокая ссылка на `wiki.mozilla.org`).

Проверка ключа хоста с помощью записи SSHFP

Напомним, что ключи хоста сервера SSH обычно игнорируются администраторами и пользователями сервера. Облачные экземпляры усугубляют проблему, потому что даже администратор не знает ключ хоста до входа в систему. К счастью, для решения этой проблемы была создана запись DNS, известная как SSHFP. Предпосылка заключается в том, что ключ сервера хранится как запись DNS. Когда клиент подключается к неизвестной системе, SSH просматривает запись SSHFP, чтобы проверить ключ сервера, а не просить пользователя проверить его.

Утилита **sshfp**, доступная из `github.com/xelerance/sshfp`, генерирует записи ресурсов DNS SSHFP либо путем сканирования удаленного сервера (флаг **-s**), либо путем разбора ранее принятого ключа из файла **known_hosts** (флаг **-k**; как и предыдущий флаг используется по умолчанию). Разумеется, любой из вариантов предполагает, что источник ключа правильный. Например, следующая команда создает BND-совместимую запись SSHFP для `server.admin.com`.

```
$ sshfp server.admin.com
server.admin.com IN SSHFP 1 1 94a26278ee713a37f6a78110f1ad9bd...
server.admin.com IN SSHFP 2 1 7cf72d02e3d3fa947712bc56fd0e0a3i...
```

Добавьте эти записи в файл зоны домена (не забывайте о коротких именах и `$ORIGIN`), перезагрузите зону и используйте утилиту **dig** для проверки ключа.

```
$ dig server.admin.com. IN SSHFP | grep SSHFP
; <<>> DiG 9.5.1-P2 <<>> server.admin.com. IN SSHFP
; server.admin.com. IN SSHFP
server.admin.com.  38400  IN  SSHFP 1 1 94a26278ee713a37f6a78110f...
server.admin.com.  38400  IN  SSHFP 2 1 7cf72d02e3d3fa947712bc56f...
```

По умолчанию утилита **ssh** не обращается к записям SSHFP. Добавьте параметр `VerifyHostKeyDNS` в файл **/etc/ssh/ssh_config**, чтобы включить эту проверку. Как и большинство опций клиента SSH, вы также можете передать параметры **-o VerifyHostKeyDNS=yes** в командной строке **ssh** при первом доступе к новой системе.

Вы можете автоматизировать этот процесс, создав запись SSHFP в сценариях инициализации сервера. Используйте динамический DNS или API вашего любимого провайдера DNS для создания записи.

Передача файлов

OpenSSH имеет две утилиты для передачи файлов: **scp** и **sftp**. На стороне сервера демон **sshd** запускает отдельный процесс под названием **sftp-server** для обработки передачи файлов. Протокол SFTP не имеет отношения к старому и небезопасному протоколу передачи файлов — FTP.

Вы можете использовать утилиту **scp** для копирования файлов из вашей системы на удаленный хост, с удаленного хоста в вашу систему или между удаленными хостами. Синтаксис напоминает команду **cp** с некоторыми дополнениями для обозначения хостов и имен пользователей.

```
$ scp ./file server.admin.com:
$ scp server.admin.com:file ./file
$ scp server1.admin.com:file server2.admincom:file
```

Утилита **sftp** имеет интерактивный характер и похожа на традиционный FTP-клиент. Существуют также графические интерфейсы SFTP для большинства настольных операционных систем.

Альтернативы для безопасного входа в систему

В большинстве систем и сетей для безопасного удаленного доступа используется пакет OpenSSH, но это не единственный выбор.

Dropbear — это реализация SSH, в которой особое внимание уделено сохранению компактности. Она компилируется в статически связанную бинарную версию размером 110 КиБ, идеально подходящую для маршрутизаторов потребительского класса и других встроенных устройств. Она обладает такими функциями (как и пакет OpenSSH), как аутентификация с помощью открытого ключа и проброс портов агента.

Сервер Teleport компании Gravitational — еще один альтернативный SSH-сервер, который предлагает несколько преимуществ. Его модель аутентификации основана на истечении срока действия сертификатов, что устраняет проблему распространения и настройки открытых ключей пользователей. Среди впечатляющих функций сервера Teleport — дополнительный журнал аудита для каждого подключения и отличная система совместной работы, которая дает возможность нескольким пользователям совместно использовать сеанс. По сравнению с системой OpenSSH, Teleport относительно новый и еще не проверенный сервер, но до сих пор в нем не было обнаружено никаких уязвимых мест. Мы предполагаем, что компания Gravitational сохранит стремительные темпы развития.

Оболочка Mosh, разработанная блестящей командой в Массачусетском технологическом институте, является заменой оболочки SSH. В отличие от SSH, оболочка Mosh работает с зашифрованными и аутентифицированными дейтаграммами UDP. Она предназначена для повышения скорости работы по WAN-соединениям и для роуминга. Например, вы можете возобновлять соединение, если вы переходите с одного IP-адреса на другой или если ваше соединение разрывается. Впервые выпущенная в 2012 году, оболочка Mosh имеет гораздо более короткую историю, чем OpenSSH, но за первые несколько лет в ней не было обнаружено никаких уязвимых мест с точки зрения безопасности. Как и Dropbear, она используется гораздо реже, чем OpenSSH.

27.8. Брандмауэры

Помимо защиты отдельных компьютеров, можно предпринять меры для обеспечения безопасности на сетевом уровне. Основным инструментом сетевой защиты является брандмауэр, физическое устройство или часть программного обеспечения, не допускающие нежелательные пакеты в системы или сети. Брандмауэры в настоящее время используются всюду. Их можно обнаружить как в настольных компьютерах и серверах, так и в маршрутизаторах и корпоративном сетевом оборудовании.

Брандмауэры, фильтрующие пакеты

Брандмауэр, фильтрующий пакеты, ограничивает виды трафика, проходящего через интернет-шлюз (или через внутренний шлюз, разделяющий домены в пределах организации), основываясь на информации, извлеченной из заголовков пакетов. Это напоминает прохождение таможни, когда вы на своем автомобиле пересекаете границу. Администратор указывает, какие целевые адреса, номера портов и типы протоколов являются допустимыми, а брандмауэр просто отбрасывает (в некоторых случаях также регистрирует) все пакеты, которые не соответствуют заданным критериям.

В системах семейства Linux фильтрация пакетов осуществляется с помощью утилиты **iptables** (и ее более удобного в использовании аналога **ufw**), а в системе FreeBSD — с помощью команды **ipfw**. Подробно они рассматриваются в разделе 13.14.

Несмотря на то что эти инструменты способны выполнять сложную фильтрацию и значительно повышают степень безопасности, в целом мы разочарованы использованием систем UNIX и Linux в качестве сетевых маршрутизаторов и особенно в качестве корпоративных маршрутизаторов, играющих роль брандмауэров. Сложность универсальных систем снижает их защищенность и надежность по сравнению со специальными устройствами. Для защиты корпоративных сетей лучше применять специальные брандмауэры, например устройства компании Check Point и Cisco.

Принципы фильтрации служб

Большинству "известных" служб назначается сетевой порт в файле **/etc/services** или его системно-зависимом эквиваленте. Демоны, которые реализуют соответствующие службы, прослушивают порты, ожидая поступления запросов на подключение со стороны удаленных компьютеров. В основном такие порты являются "привилегированными". Это значит, что их номера находятся в диапазоне от 1 до 1023 и что порты могут использоваться только процессами, работающими от имени пользователя **root**. Порты с номерами 1024 и выше являются непривилегированными.

Фильтрация на уровне служб основана на предположении, что клиент (компьютер, инициирующий соединение по протоколу TCP или UDP) будет использовать непривилегированный порт для установления соединения с привилегированным портом сервера. Например, если вы хотите разрешить только входящие HTTP-соединения для компьютера с адресом 192.108.21.200, то следует создать фильтр, который позволит направлять TCP-пакеты только на порт 80 и отправлять TCP-пакеты с этого адреса через любой порт.[8] Способ создания такого фильтра будет зависеть от типа используемого маршрутизатора.

[8] Порт 80 — это стандартный порт протокола HTTP, определенный в файле **/etc/services**.

В современных организациях, которые заботятся о безопасности, используется двухступенчатая фильтрация. Один фильтр в этой схеме подключен к Интернету как шлюз, а второй находится между этим шлюзом и остальной частью локальной сети. Идея состоит в том, чтобы сконцентрировать все входящие подключения со стороны Интернета только на компьютерах, расположенных между этими двумя фильтрами. Если эти промежуточные компьютеры административно отделены от остальной части сети, появляется возможность реализовать различные службы Интернета с меньшим риском. Частично защищенная сеть обычно называется "демилитаризованной зоной" (DMZ).

Наиболее безопасным способом использования фильтрации пакетов является настройка первоначальной конфигурации, которая не позволяет никаких входящих соединений. Затем можно постепенно ослабить фильтрацию, выяснив полезные службы, которые не работают, и перенеся все службы, доступные из Интернета, в демилитаризованую зону.

Брандмауэры, осуществляющие инспекцию пакетов с отслеживанием состояния соединений

В основе инспекции пакетов с отслеживанием состояния соединений лежит концепция, которую можно описать с помощью следующего примера. Представьте себе, что вы ищете террориста в переполненном аэропорту, внимательно прислушиваясь ко всем разговорам, происходящим вокруг вас, и понимая их содержание (на всех языках). Брандмауэры, осуществляющие инспекцию пакетов с отслеживанием состояния соединений (stateful inspection firewalls), проверяют весь трафик, проходящий через них, и сравнивают его с текущей сетевой активностью в ожидании события, которое "должно" произойти.

Например, если в пакетах, обмениваемых в рамках видеопоследовательности по протоколу H.323, указан порт, который впоследствии должен использоваться для обмена данными, то брандмауэр должен проверить, что обмен данными происходит именно через этот порт. Попытки удаленного хоста соединиться с другими портами заранее считаются фальшивыми и должны пресекаться.

Так что же на самом деле предлагают разработчики под видом инспекции пакетов с учетом состояния протокола. Их продукты либо осуществляют мониторинг ограниченного количества соединений или протоколов, либо распознают конкретный набор "неблагоприятных" ситуаций. В этом нет ничего плохого. Очевидно, что из любой технологии, способной распознавать аномалии трафика, можно извлечь пользу. Однако в данном случае важно помнить, что все эти обещания *преимущественно* являются лишь рекламным ходом.

Насколько безопасны брандмауэры

Фильтрация пакетов не должна быть основным средством защиты от хакеров. Это всего лишь один из компонентов, который заслуживает тщательного рассмотрения при создании многоуровневой системы безопасности. При наличии брандмауэра порой создается ложное впечатление, будто уровень безопасности является исключительно высоким. Если, доверившись брандмауэру, ослабить бдительность на других "рубежах", это *отрицательно* скажется на защищенности всей вашей сети.

Каждый компьютер необходимо защищать индивидуально и регулярно проверять с помощью таких средств, как Bro, Snort, Nmap, Nessus и OSSEC. Следует также обучать пользователей правилам безопасной работы в системе. В противном случае вы просто создадите внешне устойчивую, но крайне запутанную внутри систему.

В идеале локальные пользователи должны иметь возможность подключаться к любой службе Интернета, но доступ к локальным службам со стороны интернет-хостов должен быть ограничен демилитаризованной зоной. Например, к серверу архивов может быть разрешен SFTP-доступ, а к почтовому серверу, получающему входящую почту, — доступ только по протоколу SMTP.

Для того чтобы интернет-канал работал с максимальной эффективностью, советуем все же обратить основное внимание на удобство работы и доступность. В конце концов, сеть становится безопасной благодаря бдительности системного администратора, а не благодаря брандмауэру.

27.9. Виртуальные частные сети (VPN)

В простейшем виде виртуальная частная сеть (Virtual Private Networks — VPN) — это специальный тип сетевого соединения, эмулирующего непосредственное подключение удаленной сети, даже если в действительности она находится за тысячу километров и скрыта за множеством маршрутизаторов. Для повышения безопасности соединение не только подвергается аутентификации в том или ином виде (обычно с использованием общего "секретного ключа", например кодовой фразы), но еще и шифруется. Это называется *защищенным туннелем*.

Приведем пример ситуации, в которой сеть VPN оказывается очень кстати. Предположим, компания имеет офисы в нескольких городах: Чикаго, Боулдере и Майами. Если каждый офис подключен к локальному провайдеру Интернета, то посредством виртуальных частных сетей компания может прозрачным образом (и, главное, весьма безопасно) соединить все офисы через такую ненадежную сеть, как Интернет. Разумеется, такого же результата можно достичь и за счет покупки выделенных каналов связи, но это обойдется гораздо дороже.

В качестве еще одного примера можно взять компанию, служащие которой подключаются к сети предприятия из дому. Наличие виртуальных частных сетей позволит служащим пользоваться преимуществами своих высокоскоростных и недорогих домашних каналов в Интернет, и в то же время работать так, будто они непосредственно подключены к корпоративной сети.

Благодаря удобству и популярности такого подхода многие компании стали предлагать решения, связанные с сетью VPN. Функции поддержки VPN могут быть реализованы в маршрутизаторе, в операционной системе и даже в специализированном сетевом устройстве. Если позволяет бюджет, рассмотрите коммерческие предложения, в избытке имеющиеся на рынке.

Если же вы ограничены в средствах, обратитесь к пакету SSH, который позволяет организовать защищенные туннели путем проброса портов (раздел 27.7).

Туннели IPsec

Те, кого интересует надежное и недорогое решение в области VPN, соответствующее стандартам IETF, должны обратить внимание на протокол IPsec (Internet Protocol Security — механизм защиты протокола IP). Изначально он был реализован для протокола IPv6, но теперь доступен также и для IPv4. IPsec — это одобренная организацией IETF система аутентификации и шифрования соединений. Практически все серьезные поставщики VPN-систем оснащают свои продукты по крайней мере режимом совместимости с IPsec. Системы Linux и FreeBSD содержат поддержку технологии IPSec, встроенную в ядро.

Для аутентификации и шифрования механизм IPSec использует сильные криптографические алгоритмы. Аутентификация гарантирует, что пакеты приходят от легального отправителя и по дороге не были искажены, а шифрование предотвращает несанкционированный просмотр содержимого пакета.

В туннельном режиме система шифрует заголовок транспортного уровня, содержащий номера портов отправителя и получателя. К сожалению, этот механизм напрямую конфликтует со способом работы большинства брандмауэров. По этой причине в большинстве современных реализаций по умолчанию используют транспортный режим, в котором шифруется только содержимое пакетов (т.е. передаваемые данные).

Существует одна тонкость, связанная с туннелями IPsec и параметром MTU передаваемых пакетов. Важно убедиться в том, что пакет, зашифрованный средствами IPsec, не подвергается фрагментации при прохождении туннеля. Для этого может потребоваться снизить значение MTU для сетевых интерфейсов, расположенных перед туннелем (обычно подходит значение 1400 байт). Подробнее о параметре MTU рассказывалось в разделе 13.2.

Так ли уж нужны виртуальные частные сети

К сожалению, у виртуальных частных сетей есть и недостатки. Они позволяют организовать достаточно надежный туннель через ненадежный канал связи между двумя конечными точками, но они, как правило, не защищают сами конечные хосты. Например, если создать туннель VPN между корпоративной сетью и домашним компьютером президента компании, то можно ненароком открыть удобную лазейку для 15-летнего вундеркинда, по ночам наведывающегося в папин кабинет. Хорошо, если он использует эту возможность лишь для игр по сети с одноклассниками. А если нет?

Отсюда вывод: соединения, устанавливаемые через туннели VPN, следует рассматривать как внешние и предоставлять им дополнительные привилегии лишь в случае крайней необходимости, тщательно взвесив все за и против. Можно добавить в свод правил безопасности, принятый в организации, специальный пункт, касающийся работы в сети VPN.

27.10. Сертификаты и стандарты

Если содержание этой главы ошеломило вас, не беспокойтесь. Компьютерная безопасность — сложная и необъятная тема, которой посвящены бесчисленные книги, вебсайты и журналы. К счастью, для оценки и организации существующей информации было сделано немало. Существуют десятки стандартов и сертификаций, и разумный администратор должен их знать.

Сертификаты

Крупные корпорации часто нанимают постоянных сотрудников, работа которых заключается в защите информации. Для того чтобы поддерживать высокую степень компетентности в этой области, эти профессионалы посещают тренинги и получают сертификаты. Если вам придется работать с некоторыми из этих сертификатов, приготовьтесь запомнить массу аббревиатур.

Одним из наиболее широко признанных сертификатов является CISSP (Certified Information Systems Security Professional — сертифицированный специалист по безопасности информационных систем). Этот сертификат выдается организацией $(ISC)^2$, пол-

ное название которой звучит так: International Information Systems Security Certification Consortium, или Международный консорциум по сертификации безопасности информационных систем (попробуйте произнести это в десять раз быстрее!). Одной из главных особенностей сертификата CISSP является представление организации $(ISC)^2$ об "общей совокупности знаний" ("common body of knowledge — CBK), которая по существу представляет собой самые эффективные методы работы в области информационной безопасности. Комплекс CBK охватывает законодательство, криптографию, аутентификацию, физическую безопасность и многое другое. Это невероятно авторитетный сертификат в среде специалистов по безопасности.

Недостатком сертификата CISSP является излишняя широта охвата и, как следствие, недостаток глубины знаний. В комплекс CBK входит много тем, которые необходимо изучить за короткое время! Для того чтобы решить эту проблему, организация $(ISC)^2$ создала специальные программы CISSP, посвященные архитектуре, технике и управлению. Эти специализированные сертификаты придают глубину более общему сертификату CISSP.

Институт системного администрирования, сетей и безопасности (System Administration, Networking, and Security Institute — SANS Institute) в 1999 году создал набор сертификатов Global Information Assurance Certification, или Глобальная сертификация информационного обеспечения (GIAC). Три дюжины разных экзаменов охватывают всю область информационной безопасности тестами, разделенными на пять категорий. Сертификаты ранжируются по сложности — от умеренного уровня GISF с двумя экзаменами до экспертного уровня GSE с 23-часовым экзаменом. Экзамены на получение сертификата GSE печально известны как самые трудные в данной индустрии. Многие экзамены нацелены на технические вопросы и требуют довольно большого опыта работы.

В заключение упомянем мандат сертифицированного аудитора информационных систем (Certified Information Systems Auditor — CISA), представляющий собой сертификат специалиста по аудиту и процессам, связанным с информационной безопасностью. Курс обучения для получения этого сертификата сосредоточен на бизнес-процессах, процедурах, мониторинге и других вопросах управления. Некоторые специалисты считают сертификат CISA промежуточным и приемлемым для сотрудников, занимающихся безопасностью в организациях. Одним из преимуществ этого сертификата является то, что для его получения достаточно сдать только один экзамен.

Несмотря на то что сертификаты выдаются индивидуально, их признание в деловой среде невозможно отрицать. В настоящее время все больше компаний считают сертификат признаком эксперта. Многие компании предлагают таким сотрудникам более высокую зарплату и продвигают их по карьерной лестнице. Если вы решили получить сертификат, убедите вашу организацию, чтобы она оплатила ваше обучение.

Стандарты безопасности

Возрастающая роль информационных систем привела к появлению законов и указов правительства, регулирующих вопросы защиты конфиденциальной информации. Многие законодательные акты США, такие как HIPAA, FISMA[9], NERC CIP и Sarbanes-Oxley Act (SOX), содержат разделы, посвященные информационной безопасности. Несмотря на то что иногда удовлетворение этих требований связано с большими затратами, они привлекли внимание к важным аспектам технологии, которые ранее игнорировались.

[9]The Federal Information Security Management Act of 2002.

🕮 Дополнительную информацию о промышленных и законодательных стандартах, влияющих на информационные технологии, см. в разделе 31.7.

К сожалению, эти законодательные акты содержат множество юридических терминов, которые иногда трудно понять. Большинство из них не содержит никаких указаний на то, как удовлетворить сформулированные требования. По этой причине, для того чтобы помочь администраторам выполнить требования закона, были разработаны стандарты. Эти стандарты не являются законодательными актами, но следование этим стандартам обычно обеспечивает согласование с законами. Поскольку сразу всем стандартам соответствовать затруднительно, лучше всего сначала полностью реализовать какой-то один из них.

ISO 27001:2013

Стандарт ISO/IEC 27001 (бывший ISO 17799) является, вероятно, самым широко признанным в мире. Впервые он был введен в 1995 году как британский стандарт. Тогда он содержал 34 страницы и 11 разделов, которые охватывали широкий круг вопросов — от политических до вопросов физической безопасности и управления доступом. Каждый раздел имел цели, специфичные требования и рекомендации оптимальных практических решений. Этот документ стоит около 200 долл.

Требования не носят технический характер и могут быть выполнены любой организацией самым эффективным способом. С другой стороны, использование общих слов оставляло у читателя ощущение слишком большой гибкости. Критики жаловались на то, что недостаток специфики оставляет организации незащищенными перед атаками.

Тем не менее этот стандарт является одним из наиболее ценных документов в области информационной безопасности. Он создал мост над пропастью, пролегавшей между вопросами управления и техническими проблемами, и помог обеим сторонам минимизировать риск.

PCI DSS

Стандарт Payment Card Industry Data Security Standard (PCI DSS) имеет совершенно другую природу. Он возник вследствие возрастающей потребности повысить безопасность обработки платежных карт после ряда драматических событий. Например, в 2013 году правительство США обнаружило утечку информации о 160 миллионах платежных карт, выпущенных по лицензии Visa, включая JCPenney. Это было крупнейшим киберпреступлением в истории США. По некоторым оценкам ущерб от него составил более 300 млн долл.

Стандарт PCI DSS является результатом совместных усилий компаний Visa и MasterCard, хотя в данный момент он поддерживается только компанией Visa. В отличие от стандарта ISO 27001, он находится в свободном доступе. Этот стандарт полностью сфокусирован на защите данных о владельцах платежных карт и имеет 12 разделов, определяющих требования к этой защите.

Поскольку стандарт PCI DSS посвящен обработке платежных карт, он не является универсальным и не подходит для бизнеса, не имеющего дела с платежными картами. Однако компании, собирающие данные о платежных картах, обязаны строго следовать этому стандарту, чтобы избежать крупных штрафов и возможного уголовного преследования. Этот документ находится на сайте `pcisecuritystandards.org`.

Документы NIST 800

Сотрудники организации National Institute of Standards and Technology (NIST) разработали ряд документов под названием Special Publication (SP) 800, в которых изложили результаты своих исследований, рекомендации и другую информацию, посвященную компьютерной безопасности. Эти документы часто используются для оценки соответствия организаций, имеющих дело с правительственной информацией, требованиям федерального закона США об управлении информационной безопасностью (Federal Information Security Management Act). Это открытые и доступные стандарты. Они имеют превосходное содержание и широко применяются в промышленности.

Серия SP 800 содержит более 100 документов. Все они доступны на сайте `csrc.nist.gov/publications/PubsSPs.html`. Документы, с которых целесообразно начинать изучение стандартов, перечислены в табл. 27.3.

Таблица 27.3. Рекомендуемые публикации из серии NIST SP 800

Номер	Название
800-12	An Introduction to Computer Security: The NIST Handbook
800-14	Generally Accepted Principles and Practices for Securing Information Technology Systems
800-34 R1	Contingency Planning Guide for Information Technology Systems
800-39	Managing Risk from Information Systems: An Organizational Perspective
800-53 R4	Recommended Security Controls for Federal Information Systems and Organizations
800-123	Guide to General Server Security

Стандарт Common Criteria

Документ Common Criteria for Information Technology Security Evaluation (известный под названием "Common Criteria") — это стандарт, по которому оценивают уровень безопасности продукции информационных технологий. Эти рекомендации были установлены международным комитетом, состоящим из представителей разных компаний и отраслей промышленности. Для ознакомления с этим документом и сертифицированными продуктами посетите сайт `commoncriteriaportal.org`.

OWASP

Open Web Application Security Project (OWASP) — это некоммерческая всемирная организация, занимающаяся повышением безопасности прикладного программного обеспечения. Она хорошо известна благодаря своему списку "top 10", посвященному рискам веб-приложений и предназначенному для повышения бдительности разработчиков. Текущую версию этого списка и массу других материалов можно найти на сайте `owasp.org`.

CIS: Center for Internet Security

У центра CIS отличные ресурсы для администраторов. Возможно, наиболее ценными являются контрольные показатели CIS, сборник технических рекомендаций по настройке для обеспечения безопасности операционных систем. Вы можете найти тесты для каждой из рассмотренных в книге систем UNIX и Linux. В CIS также есть ориентиры для поставщиков облачных вычислений, мобильных устройств, программного обеспечения для настольных компьютеров, сетевых устройств и т.д. Подробнее читайте на сайте `cisecurity.org`.

27.11. Источники информации по вопросам обеспечения безопасности

В битве за безопасность системы половина успеха — знание современных разработок в этой области. Если система оказалась взломанной, это вряд ли стало следствием применения технологической новинки. Скорее всего, маленькая брешь в броне системы широко обсуждалась в какой-нибудь телеконференции или группе новостей.

Сервер SecurityFocus.com и списки рассылки BugTraq и OSS

Сервер `SecurityFocus.com` специализируется на сборе новостей и различной информации по вопросам безопасности. В новостях публикуются как общие обзоры, так и статьи, посвященные конкретным проблемам. Есть также обширная библиотека полезных документов, удобно отсортированных по тематике.

Программный архив сервера содержит инструментальные средства для множества операционных систем. Программы снабжаются аннотациями и пользовательскими рейтингами. Это наиболее глобальный и исчерпывающий из известных нам архивов подобного рода.

Список рассылки BugTraq — это форум для обсуждения проблем безопасности и путей их устранения. Для того чтобы подписаться на него, посетите страницу `securityfocus.com/archive`. Объем информации, рассылаемый по списку, может оказаться довольно значительным, и отношение "сигнал-шум" довольно плохое. На веб-сайте также хранится база данных с отчетами о рассмотренных на форуме BugTraq проблемах.

Список рассылки OSS (`openwall.com/lists/oss-security`) — превосходный источник новостей из мира разработчиков программного обеспечения с открытым исходным кодом, связанных с безопасностью.

Блог Брюса Шнайера

Блог Брюса Шнайера (Bruce Schneier), посвященный вопросам безопасности, является ценным и иногда увлекательным источником информации о компьютерной безопасности и криптографии. Кроме того, Шнайер является автором широко известных книг *Прикладная криптография* и *Secrets and Lies*. Информацию с его блога можно получать в виде ежемесячного бюллетеня под названием Grypto-Gram. Более подробную информацию вы найдете на сайте `schneier.com/crypro-gram.html`.

Отчет компании Verizon Data Breach Investigations

Этот ежегодный отчет заполнен статистикой о причинах и источниках нарушений защиты данных. В выпуске 2016 года на основе анализа 3141 инцидента было сделано предположение, что около 80% нарушений защиты данных являются финансово мотивированными. Шпионаж находится на далеком втором месте. В этой публикации также классифицируются атаки, наблюдаемые на практике.

Институт SANS

SANS (System Administration, Networking, and Security Institute — Институт системного и сетевого администрирования и проблем безопасности) — это профессиональная организация, которая спонсирует конференции и обучающие семинары, посвященные

вопросам безопасности, а также публикует различную информацию по данной тематике. Ее сайт `sans.org` является ценным информационным ресурсом, занимающим промежуточное положение между сайтами SecurityFocus и CERT; он не столь исчерпывающий, как первый, но и не такой скудный, как второй.

Организация SANS рассылает по электронной почте ряд еженедельных и ежемесячных бюллетеней, на которые можно подписаться на ее веб-сайте. Еженедельные NewsBites являются информативными, но ежемесячные обзоры похоже, содержат много шаблонной информации. Ни один из них не является прекрасным источником новостей из области безопасности.

Информационные ресурсы отдельных дистрибутивов

Проблемы, касающиеся безопасности, очень влияют на репутацию, поэтому поставщики операционных систем остро заинтересованы в том, чтобы помочь пользователям сделать свои системы безопасными. Многие крупные поставщики ведут собственные списки рассылки и публикуют в них бюллетени по вопросам безопасности. Эта же информация часто доступна и на веб-сайтах. Не редкость, когда защитные "заплаты" распространяются бесплатно, даже если поставщики обычно взимают плату за программную поддержку.

В Интернете есть веб-порталы, например `SecurityFocus.com`, на которых содержится информация, касающаяся конкретных производителей, и имеются ссылки на последние официальные пресс-релизы.



Разработчики системы Ubuntu поддерживают список рассылки, посвященный вопросам безопасности, по адресу:

`https://lists.ubuntu.com/mailman/listinfo/ubuntu-security-announce`

RHEL

Объявления о продуктах компании Red Hat, связанных с безопасностью, рассылаются по списку "Enterprise watch", который расположен по адресу:

`https://redhat.com/mailman/listinfo/enterprise-watch-list`



Несмотря на то что советы по безопасности системы CentOS почти всегда повторяют советы по безопасности системы Red Hat, вероятно, имеет смысл подписаться на ее рассылку по адресу:

`https://lists.centos.org/pipermail/centos-announce/`



Разработчики системы FreeBSD организовали активную группу по безопасности со списком рассылки, расположенным по адресу:

`https://lists.freebsd.org/mailman/listinfo/freebsd-security`

Другие списки рассылки и веб-сайты

Перечисленные выше контактные адреса — лишь небольшая часть интернет-ресурсов по вопросам безопасности. Учитывая объем имеющейся сегодня информации, а также скорость, с которой появляются и исчезают различные ресурсы, мы посчитали целесообразным указать несколько "метаресурсов".

Хорошей стартовой площадкой является веб-сайт `linuxsecurity.com`, на котором каждый день размещается несколько сообщений по вопросам безопасности системы Linux. Он также содержит постоянно обновляющуюся коллекцию советов по вопросам безопасности системы Linux, регистрирует происходящие события и поддерживает работу групп пользователей.

(IN)SECURE — это свободно распространяемый журнал, содержащий информацию о современных тенденциях, а также интервью с выдающимися специалистами в области безопасности. Правда, некоторые статьи в этом журнале следует читать с долей недоверия — убедитесь, что их авторы не занимаются беззастенчивой рекламой своей продукции.

Linux Weekle News — это симпатичный источник информации о ядре, безопасности, дистрибутивных пакетах и других вопросах. Раздел, посвященный проблемам безопасности, находится по адресу `lwn.net/security`.

27.12. Что нужно делать в случае атаки на сервер

Прежде всего, не паникуйте! Скорее всего, к моменту обнаружения взлома большая часть ущерба уже нанесена. Возможно, хакер "гулял" по системе несколько недель или даже месяцев. Вероятность того, что вам удалось выявить факт взлома, происшедшего всего час назад, близка к нулю.

В свете этого мудрая сова приказывает сделать глубокий вдох и начать тщательно продумывать стратегию устранения взлома. Старайтесь не вспугнуть злоумышленника, во всеуслышание заявляя о происшествии или выполняя действия, которые могут насторожить того, кто наблюдал за деятельностью организации на протяжении нескольких недель. Подсказка: в этот момент неплохо выполнить резервное копирование. Надеемся, это не удивит нарушителя?![10]

Стоит также напомнить самому себе о том, что согласно исследованиям в 60% инцидентов, связанных с нарушением безопасности, присутствовал "внутренний враг". Не изучив всех фактов, старайтесь не посвящать в проблему лишних людей, кем бы они ни были.

Приведем короткий план, который поможет администратору пережить кризис.

Этап 1: не паникуйте. Во многих случаях проблема обнаруживается только спустя какое-то время. Еще один-два часа или дня уже ничего не решают. Однако имеет значение реакция на событие — паническая или рациональная. Часто в результате возникшей паники ситуация только усугубляется уничтожением важной информации, которая позволяет выявить нарушителя.

Этап 2: определите адекватную реакцию. Никому не выгодно раздувать инцидент. Будьте благоразумны. Решите, кто из персонала будет участвовать в решении возникшей проблемы и какие ресурсы при этом должны быть задействованы. Остальные будут помогать осуществлять "вынос тела", когда все закончится.

Этап 3: соберите всю возможную информацию. Проверьте учетные и журнальные файлы. Попытайтесь определить, где первоначально произошел взлом. Выполните резервное копирование всех систем. Убедитесь в том, что резервные сменные накопители физически защищены от записи, если вы помещаете их в устройство для чтения.

Этап 4: оцените нанесенный ущерб. Определите, какая важная информация (если таковая имеется) "покинула" компанию, и выработайте подходящую стратегию смягчения возможных последствий. Оцените степень будущего риска.

[10] Если создание резервных копий не является "нормальным" действием в вашей организации, то обнаружение взлома окажется самой незначительной из ожидающих вас проблем.

Этап 5: выдерните шнур. Если это необходимо и возможно, отключите "взломанные" компьютеры от сети. Перекройте обнаруженные "дыры". Организация CERT предлагает инструкции по обнаружению взлома. Этот документ находится по адресу:

`cert.org/tech_tips/win-UNIX-system_compromise.html`

Этап 6: разработайте стратегию восстановления. Соберите толковых людей и обсудите план восстановления. Не заносите его в компьютер. Сосредоточьтесь на том, чтобы потушить "пожар" и свести ущерб к минимуму. Избегайте обвинений в чей-либо адрес или нагнетания обстановки. Не забывайте о психологическом ударе, который могут испытать пользователи. Пользователи по своей природе доверяют другим людям, поэтому вопиющее нарушение доверия часто заставляет многих людей замкнуться в себе.

Этап 7: сообщите о своем плане. Расскажите пользователям и управляющему персоналу о последствиях взлома, возможных будущих проблемах и предварительной стратегии восстановления. Будьте честны и откровенны. Инциденты, связанные с безопасностью, являются неотъемлемой частью современных сетей. Это не отражение ваших способностей как системного администратора или чего-то другого, чего можно было бы стыдиться. Открытое признание возникшей проблемы — 90% победы, если при этом вы демонстрируете, что у вас есть план выхода из ситуации.

Этап 8: воплотите план в жизнь. Вы знаете свои системы и сети лучше, чем кто бы то ни было. Доверьтесь плану и своему чутью. Посоветуйтесь с коллегами из других организаций (лучше всего с теми, кто вас очень хорошо знает), чтобы убедиться в правильности выбранного направления.

Этап 9: сообщите об инциденте в соответствующие органы. Если в инциденте участвовал "внешний игрок", сообщите об этом случае в организацию CERT на горячую линию (412)268-5800, либо по электронной почте **cert@cert.org**. Предоставьте им как можно больше информации.

Стандартная форма для отчета находится на веб-сайте `cert.org`. Ниже приведен список того, что желательно включить в отчет.

- Имена, модели "взломанных" компьютеров, а также установленные на них операционные системы.
- Список "заплат", которые были установлены в системе на момент инцидента.
- Список учетных записей, подвергшихся нападению.
- Имена и IP-адреса всех удаленных компьютеров, вовлеченных в инцидент.
- Контактная информация, если известны администраторы удаленных систем.
- Соответствующие журнальные записи и данные аудита.

Если есть подозрение, что вами обнаружена ранее неизвестная проблема в программном обеспечении, следует также сообщить об этом компании-разработчику.

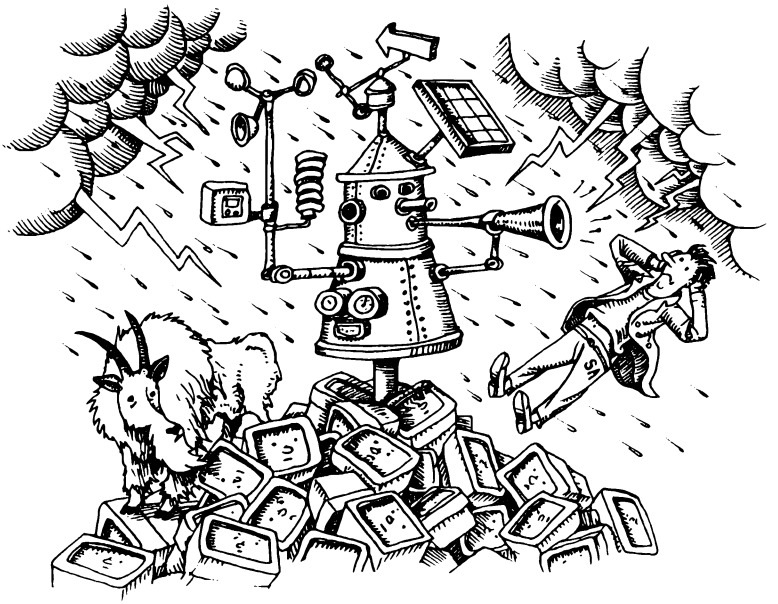
27.13. Литература

- Dykstra, Josiah. *Essential Cybersecurity Science: Build, Test, and Evaluate Secure Systems*. Sebastopol, CA: O'Reilly Media, 2016.
- Fraser, B., Editor. RFC2196: *Site Security Handbook*. rfc-editor.org, 1997.

- Garfinkel, Simson, Gene Spafford, and Alan Schwartz. *Practical UNIX and Internet Security (3rd Edition)*. Sebastopol, CA: O'Reilly Media, 2003.
- Kerby, Fred, et al. *SANS Intrusion Detection and Response FAQ*. SANS. 2009. `sans.org/resources/idfaq`
- Lyon, Gordon "Fyodor". *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Nmap Project, 2009. Книга о том, как использовать программу **nmap**, написанная ее создателем.
- Ristić, Ivan. *Bulletproof SSL and TLS: Understanding and Deploying SSL/TLS and PKI to Secure Servers and Web Applications*. London, UK: Feisty Duck, 2014.
- Schneier, Bruce. *Liars and Outliers: Enabling the Trust that Society Needs to Thrive*. New York, NY: Wiley, 2012.
- Thompson, Ken. *Reflections on Trusting Trust in ACM Turing Award Lectures: The First Twenty Years 1966-1985*. Reading, MA: ACM Press (Addison-Wesley), 1987.

Глава 28

Мониторинг



Стремление к постоянному мониторингу является отличительной характеристикой профессионального системного администратора. Неопытные системные администраторы часто оставляют системы неконтролируемыми и допускают обнаружение сбоев пользователями, когда разочарованный, сердитый сотрудник вызывает службу поддержки, поскольку не может выполнить поставленную задачу. Иногда группы администраторов создают платформу мониторинга, но отключают уведомления о сбоях в нерабочее время, чтобы лишний раз не расстраиваться. В обоих случаях начинается тушение пожаров и имитация бурной деятельности. Такие подходы негативно влияют на предприятие, усложняют усилия по восстановлению его работы и портят репутацию системных администраторов.

Профессиональные системные администраторы воспринимают мониторинг как свою религию. Каждая система, прежде чем начнет работать, добавляется на платформу мониторинга, а затем регулярно проверяется и настраивается с помощью многочисленных тестов. Показатели и тенденции оцениваются профилактически, чтобы выявить проблемы до того, как они повлияют на пользователей или поставят данные под угрозу.

Возможно, вы слышали о том, что системы трансляции потокового видео выдают телеметрическую информацию так часто, что скорее произойдет отключение службы, чем сбой мониторинга. Без мониторинга они бы в любом случае не знали, что происходит.

Методика упреждающего мониторинга (вместе со связанными с ней инструментами) делает вас супергероем. Вы лучше понимаете свое программное обеспечение и приложения, устраняете небольшие проблемы, прежде чем они начнут накапливаться в катастрофическую лавину сбоев, и более эффективно находите ошибки, слабые места и по-

вышаете производительность сложных систем. Мониторинг также улучшает качество вашей жизни, позволяя вам устранять большинство проблем в удобное для вас время, а не в три часа утра на Новый год.

28.1. Обзор мониторинга

Цель мониторинга — обеспечить правильное функционирование IT-инфраструктуры, а также организовать сбор данных для управления и планирования в доступной и понятной форме. Просто, не так ли? Но это абстрактное описание носит слишком общий характер.

Реальные системы мониторинга различаются по всем возможным измерениям, но все они имеют одну и ту же базовую структуру.

- Из систем и устройств, представляющих интерес, собираются исходные данные.
- Платформа мониторинга просматривает собранные данные и определяет, какие действия следует предпринять, как правило, применяя административно установленные правила.
- Информация об исходных данных и всех действиях, предпринятых системой мониторинга, поступают во внутренние службы, которые принимают соответствующие меры.

Реальные системы мониторинга варьируются от тривиально простых до чрезвычайно сложных.

Например, следующий сценарий на языке Perl включает все перечисленные выше элементы.

```
#!/usr/bin/env perl

$loadavg = (split /[\s,]+/, `uptime`)[10];

# Если загрузка больше 5, уведомим сисадмина
if ($loadavg > 5.0) {
   system 'mail -s "Загрузка сервера очень высокая." dan@admin.com < /dev/null'
}
```

Сценарий запускает команду **uptime** для оценки средней загрузки системы. Если одноминутная средняя загрузка больше 5, команда отправляет администратору сообщение. Данные, оценка, реакция.

В прошлом модные системы мониторинга состояли из коллекций таких сценариев, которые запускались планировщиком **cron** и управляли модемом для отправки сообщений на пейджеры системных администраторов. Сегодня у вас есть несколько вариантов на каждом этапе мониторинга.

Конечно, вы все равно можете писать индивидуальные сценарии мониторинга и запускать их из планировщика **cron**. Если это действительно все, что вам нужно, всеми силами стремитесь к максимальной простоте. Однако, если вы отвечаете не за один или два сервера, этого упрощенного подхода обычно недостаточно.

В следующих разделах более подробно рассматриваются этапы конвейерного подхода.

Инструментарий

Широкий спектр данных, которые могут оказаться полезными для вашей организации, включает показатели эффективности (время реакции, эффективность использования, ско-

рость передачи), показатели доступности (доступность и время безотказной работы), емкость, изменения состояния, записи в журнале и даже бизнес-показатели, такие как сумма среднестатистической корзины покупок или коэффициент переходов по кликам.

Поскольку все, что можно было бы сделать на компьютере, потенциально может быть связано с мониторингом, системы мониторинга обычно являются агностическими по отношению к источникам данных. Они часто приходят со встроенной поддержкой различных входных данных. Даже источники данных, которые не имеют прямой поддержки, обычно могут быть подключены с помощью нескольких строк кода адаптера или отдельного шлюза данных, такого как StatsD (см. раздел 28.4).

Если вам приходится собирать слишком большое количество данных, самой важной частью проектирования системы сбора данных становится выбор наиболее важных показателей и игнорирование ненужных. Избегайте сбора данных, которые не имеют четкой и полезной цели. Сбор данных переносит нагрузку как на систему мониторинга, так и на контролируемые объекты. Лишние данные также затмевают показатели, которые действительно важны, топя их в море шума.

К сожалению, часто бывает непросто отличить полезные данные от шлака. Вы должны постоянно пересматривать то, что контролируется, и переосмысливать, как эти данные будут использоваться в течение всего времени работы системы.

Типы данных

На самом высоком уровне данные мониторинга могут быть сгруппированы в три общие категории.

- *Показатели реального времени*, которые характеризуют рабочее состояние окружающей среды. Обычно это числа или логические значения. В целом система мониторинга должна сравнивать эти показатели с ожидаемыми значениями и генерировать предупреждение, если текущее значение выходит из предопределенного диапазона или превышает заданный порог.
- *События*, которые часто принимают форму записей в файле журнала или уведомлений, поступающих из подсистем. Эти события, иногда называемые *показателями на основе шаблонов*, могут указывать на то, что произошло изменение состояния, возникла тревога или выполнено определенное действие. События могут обрабатываться для формирования числовых показателей (например, сумма или скорость) или инициировать реакцию системы мониторинга напрямую.[1]
- *Совокупные и обобщенные исторические тенденции*, которые часто представляют собой временные ряды показателей в реальном времени. Они позволяют анализировать и визуализировать изменения, происходящие с течением времени.

Ввод и обработка

Большинство систем мониторинга вращаются вокруг центральной платформы мониторинга, которая принимает данные из контролируемых систем, выполняет соответ-

[1] Многие из данных, собранных программным обеспечением для мониторинга приложений, попадают в категорию "событие"; иногда они также имеют количественный характер. Взаимосвязи между событиями (например, "пользователь просмотрел страницу Настройки, но затем отменил ввод, не изменив ничего") часто помогают исследовать ситуацию. Платформы мониторинга общего назначения, как правило, не очень хороши для таких перекрестных ссылок, и это является одной из причин того, что мониторинг приложений является отдельной категорией.

ствующую обработку и применяет административные правила для определения того, что должно произойти в ответ.

Платформы первого поколения, такие как Nagios и Icinga, были сосредоточены на обнаружении возникающих проблем и реагировании. Эти системы в свое время были революционными и привели нас в современный мир мониторинга. Тем не менее со временем стало понятно, что все данные мониторинга являются данными временных рядов. Если значения не изменяются, вы не будете их контролировать.

Очевидно, что необходим подход, более сильно ориентированный на данные. Однако данные мониторинга обычно настолько объемные, что вы не можете просто сбрасывать все это в традиционную базу данных и позволять ей накапливаться. Это верный путь к снижению производительности и переполнению дисков.

Современный подход — организация мониторинга на основе хранилища данных, которое специализируется на обработке временных рядов. В течение начального периода хранятся все данные, но по мере возрастания их объема происходит их агрегирование с учетом ограничений, связанных с хранением. Например, хранилище может сберегать данные за час работы с точностью до одной секунды, за неделю — до одной минуты и за год — до часа.

Исторические данные полезны не только для презентаций, но и для сравнения. Например, превышает ли текущий показатель сетевой ошибки, равный 25%, его среднее значение за определенный период?

Уведомления

Создав систему мониторинга, внимательно изучите, что делать с результатами мониторинга. Основным приоритетом обычно является уведомление администраторов и разработчиков о проблеме, которая требует внимания.

Уведомления должны побуждать к действию. Структурируйте свою систему мониторинга, чтобы каждый, кто получит данное уведомление, потенциально мог что-то сделать в ответ, даже если его действие будет чем-то общим, например, "проверить позже, чтобы убедиться, что меры приняты". Уведомления, которые являются чисто информационными, персонал обычно игнорирует.

В большинстве случаев уведомления не должны ограничиваться электронной почтой, чтобы быть оптимально эффективными. Для критических проблем проще и эффективнее всего посылать SMS-уведомления (т.е. текстовые сообщения) на сотовые телефоны администраторов. Получатели могут устанавливать свои мелодии звонка и громкость телефона, чтобы проснуться посреди ночи, если это необходимо.

Уведомления также должны быть интегрированы с реализацией модели ChatOps вашей командой. Менее критические уведомления (например, состояние заданий, неудачные попытки регистрации в системе и информационные уведомления) можно отправлять в один или несколько чатов, чтобы заинтересованные стороны могли активно получать подмножества предупреждений, которые для них могут представлять интерес.

🕮 Дополнительную информацию о модели ChatOps см. в разделе 31.1.

Помимо этих основных каналов, существует масса других возможностей. Например, в центре обработки данных или в центре сетевых операций для индикации состояния может оказаться полезной светодиодная система освещения, меняющая цвета в зависимости от состояния системы. Другие варианты реагирования на ситуации, выявленные системами мониторинга, включают следующее.

- Автоматизированные действия, такие как сброс базы данных или ротация журналов.
- Вызов администратора по телефону.
- Отправка данных на доску объявлений для общего доступа.
- Хранение данных в базе данных временных рядов для последующего анализа.
- Ничего не делать, предоставив последующий обзор самой системе.

Контрольные панели и пользовательские интерфейсы

Помимо предупреждения о явно исключительных обстоятельствах, одной из основных целей мониторинга является представление состояния окружающей среды в более четком и понятном виде по сравнению с необработанными данными. Такие представления называются *контрольными панелями* (dashboards).

Контрольные панели разрабатываются администраторами или другими сторонами, интересующимися конкретными аспектами окружающей среды. Они используют различные методы для преобразования необработанных данных в информационную графику.

Во-первых, контрольные панели избирательны в том, что они представляют. Они концентрируются на наиболее важных показателях для данной предметной области, которые указывают на общую работоспособность или производительность. Во-вторых, они дают контекстные подсказки о значимости и происхождении данных, которые показаны на панели. Например, проблемные показатели и состояния обычно отображаются красным цветом, а основные показатели изображаются с помощью крупных шрифтов. Отношения между значениями показываются посредством группировки. В-третьих, контрольные панели отображают серию данных в виде диаграмм, что позволяет легко их оценивать с первого взгляда.

Конечно, большинство собранных данных никогда не появится на контрольной панели. Кроме того, желательно, чтобы ваша система мониторинга имела обобщенный интерфейс, который облегчает исследование и изменение схемы данных, позволяет делать произвольные запросы к базе данных и строить диаграммы произвольно определенных последовательностей данных "на лету".

28.2. Культура мониторинга

Эта глава в основном посвящена инструментам мониторинга, но не менее важной является культура мониторинга. Начиная изучение мониторинга, имейте в виду следующие принципы.

- Если кто-то зависит от системы или службы, ее необходимо контролировать. Необходим тотальный контроль. Ничто в среде, от которой зависит служба или пользователь, не может оставаться неконтролируемым.
- Если производственное устройство, система или служба предоставляют контролируемые атрибуты, эти атрибуты должны постоянно отслеживаться. Не пускайте дело на самотек, лишь изредка поглядывая на причудливые интерфейсы управления аппаратным обеспечением сервера. Так вы можете лишь через месяц узнать, что вентилятор вышел из строя.
- Необходимо контролировать все элементы системы с высокой доступностью. Было бы печально осознать, что основной сервер давно вышел из строя сразу после того, как резервный сервер перестал работать.

- Мониторинг является обязательным. В планах работы каждого системного администратора, разработчика, сотрудника эксплуатационного отдела, руководителя и менеджера проекта должны быть предусмотрены меры по мониторингу.
- Данные мониторинга (особенно исторические данные) полезны для всех. Сделайте эти данные легко доступными и видимыми, чтобы каждый мог использовать их в анализе основных причин, при планировании, управлении жизненным циклом и поисках возможностей для улучшения архитектуры. Приложите усилия и выделите ресурсы на создание и продвижение контрольных панелей мониторинга.
- Каждый должен реагировать на предупреждения. Мониторинг — это не просто выявление проблем. Все сотрудники, исполняющие технические роли, должны получать уведомления и работать совместно для решения проблем. Этот подход способствует добросовестному анализу первопричин, который наиболее подходит для устранения основной проблемы.
- Правильно реализованный мониторинг положительно влияет на качество жизни. Четкая схема мониторинга освобождает вас от бремени беспокойства о том, в каком состоянии находятся ваши системы, и дает другим возможность поддерживать вас. Без мониторинга и соответствующей документации вы по существу работаете в авральном режиме 24 × 7 × 365.
- Обучайте людей, предпринимающих ответные меры, чтобы исправлять предупреждения, а не просто подавлять их. Оцените ложноположительные или зашумленные предупреждения и настройте их так, чтобы они больше не генерировались ненадлежащим образом. Ложные предупреждения побуждают всех игнорировать систему мониторинга.

28.3. Платформы мониторинга

Если вы планируете контролировать несколько систем и более чем несколько показателей, стоит потратить некоторое время на развертывание платформы мониторинга полного обслуживания. Это системы общего назначения, которые собирают данные из нескольких источников, облегчают отображение и обобщение информации о состоянии систем и устанавливают стандартный способ определения действий и предупреждений.

Хорошей новостью является то, что существует множество вариантов. Плохая новость заключается в том, что одной совершенной платформы пока не существует. Выбрав один из доступных вариантов, рассмотрите следующие проблемы.

- *Гибкость сбора данных.* Все платформы могут получать данные из разных источников. Однако это не означает, что все платформы эквивалентны в этом отношении. Рассмотрите источники данных, которые вы хотите использовать. Вам нужно будет читать данные из базы данных SQL? Из записей DNS? От HTTP-соединения?
- *Качество пользовательского интерфейса.* Многие системы предлагают настраиваемые графические интерфейсы или веб-интерфейсы. Сегодня в большинстве хорошо продаваемых пакетов используются шаблоны JSON для представления данных. Пользовательский интерфейс — это не просто очередной маркетинговый ход; вам нужен интерфейс, который отображает информацию четко, просто и понятно. Вам нужны разные пользовательские интерфейсы для разных групп в вашей организации?
- *Стоимость.* Некоторые коммерческие пакеты управления имеют высокую цену. Многие корпорации считают престижным, что их сеть управляется высококаче-

ственной коммерческой системой. Если это не так важно для вашей организации, рассмотрите бесплатные варианты, такие как Zabbix, Sensu, Cacti и Icinga.

- *Автоматическое обнаружение.* Многие системы предлагают "исследовать" вашу сеть. Благодаря сочетанию широковещательных сообщений, запросов SNMP, поиска в таблице ARP и DNS-запросов, они идентифицируют все локальные хосты и устройства. Все реализации исследования, которые мы видели, работают очень хорошо, но в сложных системах и системах с большим количеством брандмауэров, их точность будет ниже.
- *Функции отчетности.* Многие продукты могут отправлять оповещения по электронной почте, взаимодействовать с моделью ChatOps, отправлять текстовые сообщения и автоматически генерировать билеты для популярных систем отслеживания неисправностей. Убедитесь, что выбранная вами платформа обеспечивает гибкую отчетность. Кто знает, с какими электронными устройствами вы столкнетесь через несколько лет?

Платформы реального времени с открытым исходным кодом

Хотя платформы, рассмотренные в этом разделе — Nagios, Icinga и Sensu Core, — делают всего понемножку, они известны своей способностью обрабатывать мгновенные (или пороговые) показатели.

У этих систем есть свои сторонники, но как инструменты мониторинга первого поколения они постепенно уступают системам временных рядов, о которых мы уже говорили выше. Начиная с нуля, лучше всего выбирать системы временных рядов.

Платформы Nagios и Icinga

Платформы Nagios и Icinga специализируются на выдаче уведомлений об ошибках в режиме реального времени. Несмотря на то что они не помогают вам определить, насколько увеличилось использование вашей полосы пропускания в течение последнего месяца, они могут выследить момент, когда ваш веб-сервер перестанет отвечать на запросы.

Платформы Nagios и Icinga первоначально были ветками одного дерева исходного кода, но современная версия Icinga 2 была полностью переписана. Тем не менее она по-прежнему совместима с платформой Nagios во многих отношениях.

Обе системы включают в себя множество сценариев для мониторинга всех видов и объемов, а также обширные возможности мониторинга SNMP. Возможно, их самая большая сила — модульная и тонкая система настройки, которая позволяет вам писать собственные сценарии для мониторинга любых мыслимых показателей.

Вы можете состряпать на скорую руку новые мониторы на языках Perl, PHP, Python или даже C, если страдаете излишним самомнением или мазохизмом. Многие стандартные методы уведомления уже реализованы в виде электронной почты, веб-отчетов, текстовых сообщений и т.д. Кроме того, как и при мониторинге дополнительных модулей, несложно создать собственные сценарии уведомлений и действий.

Платформы Nagios и Icinga работают хорошо для сетей, состоящих из менее чем тысячи хостов и устройств. Они легко настраиваются и расширяются и включают в себя такие мощные функции, как резервирование, удаленный мониторинг и эскалация уведомлений.

Если вы развертываете новую инфраструктуру мониторинга с нуля, мы рекомендуем Icinga 2, а не Nagios. Его кодовая база, как правило, более ясная, что хорошо способствует росту числа поклонников и поддержки сообщества. С функциональной точки зрения его

пользовательский интерфейс более простой и быстрый, и он способен автоматически обновлять служебные зависимости, которые могут быть необходимы в сложных средах.

Платформа Sensu

Sensu — это самодостаточная система мониторинга, которая доступна как в версии с открытым исходным кодом (Sensu Core), так и с платными, коммерчески поддерживаемыми надстройками. Она имеет ультрасовременный интерфейс и может запускать любые устаревшие дополнительные модули Nagios, Icinga или Zabbix.

Эта платформа была разработана для замены Nagios, поэтому совместимость с дополнительными модулями является одной из самых привлекательных функций. Система Sensu позволяет легко использовать уведомления Logstash и Slack, а процесс ее инсталляции очень простой.

Платформы временных рядов с открытым исходным кодом

Обнаружение и реагирование на текущие проблемы — всего лишь один из аспектов мониторинга. Часто не менее важно знать, как значения меняются со временем и как они соотносятся с другими значениями. Чтобы удовлетворить эти потребности, были разработаны четыре популярные платформы временных рядов: Graphite, Prometheus, InfluxDB и Munin.

В этих системах на первом плане и в центре внимания всей экосистемы мониторинга находится база данных. Они отличаются своей степенью полноты в качестве автономных систем мониторинга и в целом являются более модульными, чем традиционные системы, такие как Icinga. Возможно, вам понадобятся дополнительные компоненты для создания полной платформы мониторинга.

Платформа Graphite

Платформа Graphite была, возможно, авангардом нового поколения платформ для мониторинга временных рядов. По своей сути это гибкая база данных временных рядов с простым в использовании языком запросов. Причиной движения `#monitoringlove` и огромного влияния, которое платформа Graphite оказывает на пользовательские интерфейсы, является то, как она агрегирует и суммирует показатели. Она начала переход от ежеминутного мониторинга к посекундному.

Как вы можете догадаться из названия, платформа Graphite включает в себя функции графического отображения для веб-визуализации. Однако этот аспект пакета несколько затмил пакет Grafana. В наши дни система Graphite лучше известна благодаря другим ее компонентам — Carbon и Whisper, которые составляют основу системы управления данными.

Систему Graphite можно комбинировать с другими инструментами для создания масштабируемой, распределенной, кластерной среды мониторинга, способной поглощать и публиковать сотни тысяч показателей. На рис. 28.1 показана структурная схема такой реализации.

Платформа Prometheus

Нашей любимой платформой временного ряда сегодня является Prometheus. Это комплексная платформа, которая включает интегрированные компоненты сбора, трендов и оповещений. Ее компоненты ориентированы как на системных администраторов, так и на разработчиков, что делает ее отличным выбором для методики DevOps. Однако она не допускает кластеризацию, т.е. не подходит для систем, требующих высокой доступности.

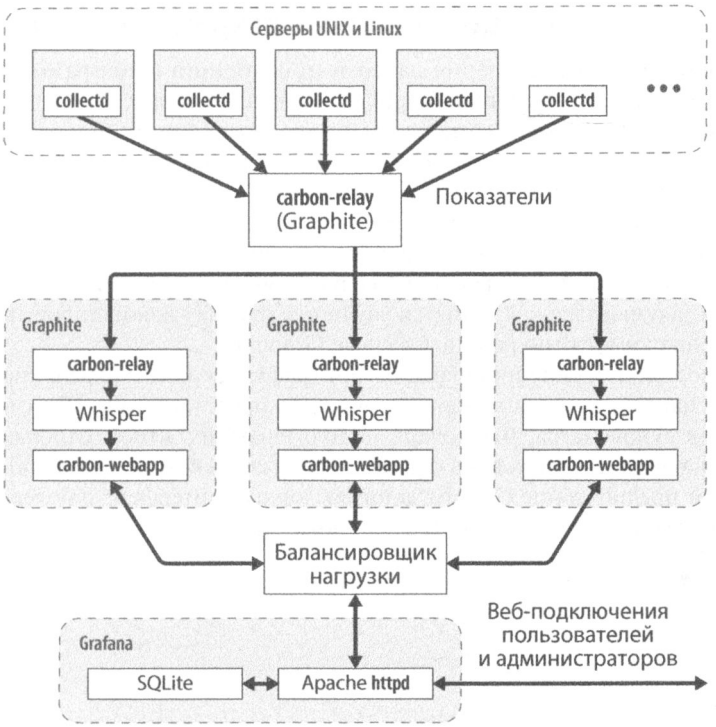


Рис. 28.1. Кластерная архитектура платформы Graphite

Платформа InfluxDB

InfluxDB — чрезвычайно удобная для разработчиков платформа мониторинга временных рядов, поддерживающая широкий спектр языков программирования. Подобно Graphite, платформа InfluxDB — это всего лишь механизм управления базой данных временных рядов. Чтобы сформировать полную систему мониторинга, включающую в себя такие функции, как оповещение, вам необходимо дополнить пакет с помощью внешних компонентов, таких как Grafana.

Функции управления данными InfluxDB намного богаче, чем перечисленные выше альтернативы. Однако дополнительные функции InfluxDB также добавляют некоторую нежелательную сложность для типичных установок.

Платформа InfluxDB имеет несколько проблемную историю ошибок и несовместимостей. Тем не менее текущая версия выглядит стабильной и, вероятно, является лучшей альтернативой Graphite, если вы ищете автономную систему управления данными.

Платформа Munin

Исторически платформа Munin пользовалась популярностью, особенно в Скандинавии. Он построена на интеллектуальной архитектуре, в которой дополнительные модули для сбора данных не только предоставляют данные, но и сообщают системе, как они должны быть представлены. Несмотря на то что Munin по-прежнему отлично подходит для использования, для новых развертываний следует рассматривать более современные решения, такие как Prometheus. В некоторых случаях Munin по-прежнему является полезным инструментом для мониторинга конкретных приложений; см. раздел 28.7.

Платформы визуализации данных с открытым исходным кодом

Два основных варианта создания панелей мониторинга и диаграмм — графические функции, встроенные в платформу Graphite и более новый пакет Grafana.

Платформа Graphite может извлекать данные для визуализации из хранилищ, отличных от Whisper (собственный компонент хранения данных в пакете Graphite), но это не всегда является простым решением.

В качестве пакета, не зависящего от базы данных, Grafana отлично справляется с внешними хранилищами данных, включая все перечисленное в предыдущем разделе. По последним подсчетам обеспечена поддержка 30 разных хранилищ. Первоначально пакет Grafana задумывался как попытка улучшить графику для платформы Graphite, так что с ним вполне комфортно работать в среде Graphite.

Обе платформы, Graphite и Grafana, имеют графический интерфейс, напоминающий контрольную панель, которая может генерировать представления, облегчающие анализ и удобные для руководства. Вы можете использовать их, чтобы отображать что-либо от низкоуровневых системных показателей до показателей бизнес-уровня. Пользователи обычно отдают предпочтение Grafana за превосходный интерфейс и более красивые графики. Пример контрольной панели Grafana показан на рис. 28.2.

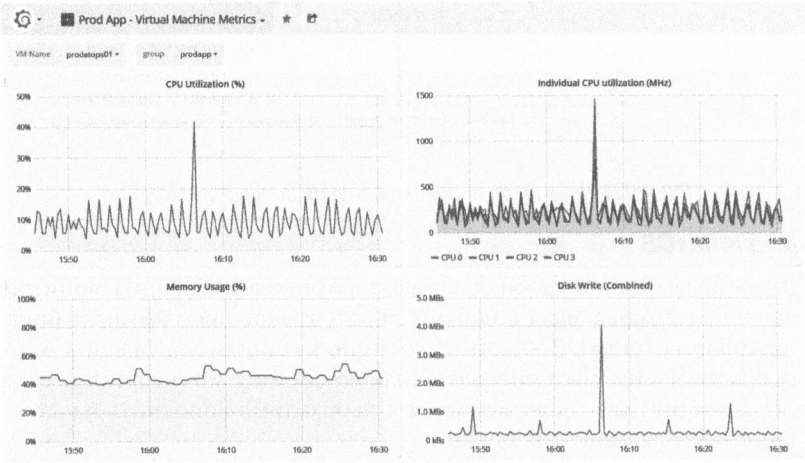


Рис. 28.2. Пример контрольной панели Grafana

Коммерческие платформы мониторинга

Сотни компаний продают программное обеспечение для мониторинга, а новые конкуренты выходят на рынок каждую неделю. Если вы ищете коммерческое решение, вы должны по крайней мере рассмотреть варианты, перечисленные в табл. 28.1.

Независимо от того, находятся ли их системы в облаке, на гипервизоре центра обработки данных или в системной стойке, большинству предприятий не следует создавать собственные средства мониторинга. Аутсорсинг дешевле и надежнее. Следовательно, если вам нужен набор инструментов для мониторинга общего набора приложений или серверов, рассмотрите платформы Datadog, Librato, SignalFx или Sysdig Cloud.

Таблица 28.1. Популярные платформы коммерческого мониторинга

Платформа	URL	Комментарии
Datadog	datadoghq.com	Облачная платформа мониторинга приложений Огромный список поддерживаемых систем, приложений и служб
Librato	librato.com	Система Plug and Play с готовыми дополнительными модулями с открытым исходным кодом
Monitus	monitus.net	Мониторинг платформы электронной коммерции
Pingdom	pingdom.com	Платформа мониторинга SaaS[a]
SignalFx	signalfx.com	Платформа SaaS с длинным списком облачных интеграций
SolarWinds	solarwinds.com	Платформа для сетевого мониторинга
Sysdig Cloud	sysdig.com	Специальность: мониторинг и оповещение системы Docker Простота корреляции событий между службами
Zenoss	zenoss.com	Невероятно сложная альтернатива Icinga

[a] Не требуется установка программного обеспечения. Хорошо подходит только для веб-приложений.

При исследовании платформ коммерческого мониторинга сначала обратите внимание на цену. Но не забудьте также изучить эксплуатационные детали.

- Насколько легко система мониторинга может быть внедрена в вашу систему управления конфигурацией?
- Как в системе мониторинга развертываются новые дополнительные модули или модули проверок для ваших хостов? Их выталкивают или вытягивают?
- Хорошо ли система мониторинга сочетается с существующей платформой уведомлений, если таковая имеется?
- Обеспечивает ли ваша среда тип внешнего подключения, необходимый для облачного мониторинга?

Это всего лишь часть вопросов, которые вы должны задавать при исследовании платформ. В конце концов, лучшая платформа для вашей организации — это та, которая легко настраивается, соответствует вашему бюджету и легко принимается вашими пользователями.

Размещенные платформы мониторинга

Если вы не заинтересованы в настройке и обслуживании собственных средств мониторинга сети, вы можете рассмотреть размещенное (облачное) решение. Существует множество бесплатных и коммерческих опций, но популярным является StatusCake (statuscake.com). Возможность внешнего провайдера видеть внутренние детали вашей сети ограничена, но размещенные варианты хорошо подходят для проверки работоспособности служб и веб-сайтов, ориентированных на широкую аудиторию пользователей.

Хостинг-провайдер системы мониторинга может также освободить вас от ограничений, накладываемых для обычных каналов в Интернет, которыми пользуются в вашей организации. Если для передачи уведомлений из внутренней системы мониторинга вы используете сеть своего хостинг-провайдера, как это в конечном итоге происходит в большинстве организаций, вы можете убедиться, что эта сеть сама контролируется и управляется, чтобы ее персонал смог оперативно отреагировать в случае возникновения проблем.

28.4. Сбор данных

В предыдущих разделах были рассмотрены различные пакеты, которые могут служить ядром системы мониторинга сети. Однако выбор и развертывание одной из этих систем — это только первая часть процесса настройки. Теперь вы должны убедиться, что данные и события, которые вы хотите отслеживать, доступны для центральной платформы мониторинга.

Детали этого инструментального процесса зависят от систем, которые вы хотите контролировать, и от философии вашей платформы мониторинга. Во многих случаях вам нужно написать несколько простых сценариев для преобразования информации о состоянии системы в форму, которую может понять ваша платформа мониторинга. Некоторые платформы, такие как Icinga, поставляются с широким набором дополнительных модулей, которые собирают стандартные показатели из типичных контролируемых систем. Другие, такие как Graphite и InfluxDB, не создают специальных возможностей для ввода данных вообще и должны быть дополнены интерфейсом, который выполняет эту роль.

В следующих разделах мы сначала рассмотрим статическую платформу для сбора данных общего назначения, а затем изучим некоторые инструменты и методы для управления типичными контролируемыми системами.

StatsD: протокол передачи общих данных

Демон StatsD был написан инженерами компании Etsy как способ отслеживать все и вся в пределах их собственной среды. Это прокси-сервер на основе протокола UDP, сбрасывающий любые данные, которые вы вводите в него, на платформу мониторинга для анализа, расчета и отображения. Преимущество StatD — его способность принимать и выполнять вычисления произвольных статистических показателей.

Демон StatsD компании Etsy был написан для среды Node.js. Но в наши дни имя "StatsD" относится скорее к протоколу, чем к любому из многих пакетов программного обеспечения, которые его реализуют. (Правда, даже версия Etsy не является оригинальной, в ее основу был положен проект с аналогичным названием на сайте Flickr.) Реализации StatsD были написаны на разных языках, но здесь мы сосредоточились на версии, выпущенной компанией Etsy.

Для работы демона StatsD требуется среда Node.js, поэтому перед установкой StatsD убедитесь, что Node.js была установлена и настроена соответствующим образом. Реализация Etsy не включена в большинство репозиториев пакетов поставщиков операционных систем, хотя другие версии StatsD часто в них есть. Поэтому убедитесь, что вы не путаете их. Проще всего клонировать версию Etsy прямо из GitHub:

```
$ git clone https://github.com/etsy/statsd
```

Демон StatsD невероятно модульный и может подавать входные данные на множество серверов сбора данных и клиентов. Давайте рассмотрим простой пример, в котором для сбора данных используется Graphite.

Чтобы обеспечить правильную связь Graphite и StatsD, вы должны изменить компонент хранения Carbon системы Graphite. Измените файл `/etc/carbon/storage-schemas.conf` и добавьте в него приведенный ниже раздел:

```
[stats]
pattern = ^stats.*
retentions = 10s:12h,1min:7d,10min:1y
```

Эта конфигурация сообщает компоненту Carbon хранить данные, полученные в течение 12 часов с интервалом в 10 с. Компонент Carbon суммирует устаревшие данные с интервалом в 1 мин. и сохраняет эту итоговую информацию еще 7 дней. Аналогично данные с 10-минутной детализацией сохраняются в течение всего года. В этих вариантах нет ничего волшебного; вам необходимо определить, что подходит для потребностей вашей организации в хранении и собираемых данных.

Точное определение того, что означает "суммировать" временные ряды, зависит от типа данных. Например, если вы подсчитываете сетевые ошибки, вы, вероятно, хотите суммировать, складывая значения. Если вы смотрите на показатели, представляющие загрузку системы или процент ее использования, вы, вероятно, захотите узнать их средние значения. Вам также может потребоваться указать подходящие способы обработки отсутствующих данных.

Эти политики указаны в файле **/etc/carbon/storage-aggregation.conf**. Если у вас еще нет рабочей инсталляции Graphite, вы можете использовать пример конфигурации Graphite в качестве отправной точки:

/usr/share/doc/graphite-carbon/examples/storage-aggregation.conf.example

Ниже приведены некоторые разумные стандартные значения для включения в файл **storage-aggregation.conf**.

```
[min]
pattern = \.lower$
xFilesFactor = 0.1
aggregationMethod = min

[max]
pattern = \.upper(_\d+)?$
xFilesFactor = 0.1
aggregationMethod = max

[sum]
pattern = \.sum$
xFilesFactor = 0
aggregationMethod = sum

[count]
pattern = \.count$
xFilesFactor = 0
aggregationMethod = sum

[count_legacy]
pattern = ^stats_counts.*
xFilesFactor = 0
aggregationMethod = sum

[default_average]
pattern = .*
xFilesFactor = 0.3
aggregationMethod = average
```

Обратите внимание: в каждом разделе конфигурации указано регулярное выражение `pattern`, с помощью которого сопоставляются имена рядов данных. Разделы считываются по порядку, и первый подходящий раздел становится управляющим спецификатором для каждой серии данных. Например, серия с именем `sample.count` будет соответ-

ствовать шаблону для раздела `[count]`. Значения будут свернуты путем суммирования точек данных (`aggregationMethod = sum`).

Параметр `xFilesFactor` определяет минимальное количество выборок, необходимых для значительного сокращения вашей метрики. Он выражается как число от 0 до 1, представляющее процент ненулевых значений, которые должны существовать на более детализированном уровне, чтобы уровень свертки имел ненулевое значение. Так, установка `xFilesFactor` для `[min]` и `[max]` в приведенном выше примере составляет 10%, поэтому даже одно значение данных будет соответствовать этому критерию, учитывая наши настройки в файле **`storage-schema.conf`**. По умолчанию это значение равно 50%. Если эти параметры заданы непродуманно, вы получите неточные или отсутствующие данные!

Мы можем отправить некоторые тестовые данные демону StatD с помощью системы Netcat (команда **`nc`**):

```
$ echo "sample.count:1|c" | nc -u -w0 statsd.admin.com 8125
```

Эта команда помещает значение **`1`** в качестве показателя подсчета (как указано в параметре **`c`**) в набор данных `sample.count`. Пакет поступает на порт 8125 на сайте `statsd.admin.com`; этот порт демон **`statsd`** прослушивает по умолчанию. Если наш набор данных отображается на контрольной панели Graphite, вы готовы собирать все виды данных мониторинга с помощью одного из многих клиентов StatD. На странице **`wiki StatsDGitHub`** (`github.com/etsy/statsd/wiki`) приведен список клиентов, которые могут общаться с помощью протокола StatsD. Или можете написать собственный! Протокол прост и его возможности бесконечны.

Сбор данных из вывода команды

Любую информацию, полученную из командной строки, можно отслеживать на вашей платформе мониторинга. Все, что нужно, — написать несколько строк сценарного кода, чтобы извлечь интересующие вас фрагменты данных и перевести их в формат, который может принять ваша платформа мониторинга.

Например, команда **`uptime`** выводит время непрерывной работы системы, количество зарегистрированных пользователей и среднюю загрузку системы за последние 1, 5 и 15 мин.

```
$ uptime
07:11:50 up 22 days, 10:13, 2 users, load average: 1.20, 1.41, 1.88
```

Человек может проанализировать этот вывод с первого взгляда и увидеть, что текущее среднее значение загрузки системы равно 1,20. Если вы хотите написать сценарий, чтобы регулярно проверять это значение или передавать его другому процессу мониторинга, нужно использовать команды обработки текста для выделения требуемого значения:

```
$ uptime | perl -anF'[\s,]+' -e 'print $F[10]'
1.20
```

Здесь мы используем язык Perl для разделения вывода везде, где есть последовательность пробелов и запятых, и выводим содержимое десятого поля (среднее значение загрузки за одну минуту). Готово! Хотя в большинстве предметных областей язык Perl вытесняется современными языками, такими как Python и Ruby, он все еще остается основным средством обработки текстовых строк. Вероятно, не стоит изучать Perl исключительно для этих целей, но его способность формулировать сложные текстовые преобразования в виде однострочных команд очень пригодится.

Мы можем легко развернуть этот однострочный сценарий в короткий код, который определяет среднюю загрузку сервера и отправляет ее демону StatD.

```
#!/usr/bin/env perl

use Net::Statsd;
use Sys::Hostname;

$Net::Statsd::HOST = 'statsd.admin.com';

$loadavg = (split /[\s,]+/, `uptime`)[10];
Net::Statsd::gauge(hostname . '.loadAverage' => $loadavg);
```

Сравните этот код сценария с однострочной тестовой командой на предыдущей странице и однострочным кодом обработки вывода команды **uptime**. Здесь язык Perl должен запускать команду **uptime** и обрабатывать ее вывод как строку, так что эта часть выглядит несколько иначе, чем ее однострочный эквивалент. (В однострочном варианте мы использовали режим автоматического разбиения строк в языке Perl.)

Вместо использования утилиты **nc** для сетевой передачи данных демону StatD мы применяем простую оболочку StatD, которую скачали из архива CPAN[2]. Это предпочтительный подход; библиотеки более надежны, чем специальные сценарии, и их использование проясняет назначение кода.

Многие команды могут генерировать более одного формата вывода. Проверьте справочную страницу команды, чтобы узнать, какие параметры доступны, прежде чем пытаться анализировать ее вывод. С одними форматами гораздо легче работать, чем с другими.

Ряд команд поддерживают выходной формат, который специально облегчает разбор. У других есть настраиваемые системы вывода, в которых можно запросить только необходимые поля. Еще один распространенный вариант — это флаг, который подавляет описательные строки заголовка на выходе.

28.5. Мониторинг сетей

Во многих организациях мониторинг состояния сети традиционно остается основной задачей, для решения которой приходится окунаться в более широкий мир разнообразных систем мониторинга и контрольных панелей. Поэтому его мы и рассмотрим более подробно. В последующих разделах также описывается мониторинг операционных систем, приложений, служб и систем безопасности.

Основным элементом сетевого мониторинга является тестовое зондирование сети с помощью эхо-запросов (network ping) с помощью так называемых ICMP-пакетов. Более подробно технические детали рассматриваются в разделе 13.12, вместе с командами **ping** и **ping6**, которые инициируют эхо-запросы из командной строки.

Концепция проста: вы отправляете пакет эхо-запроса другому хосту сети, а реализация протокола IP этого хоста возвращает вам пакет в ответ. Если вы получаете ответ на свой запрос, то знаете, что все сетевые шлюзы и устройства, которые находятся между вами и целевым хостом, работают. Вы также знаете, что целевой хост включен и что его ядро операционной системы запущено и работает. Однако, поскольку эхо-запрос обрабатывается стеком протокола TCP/IP, он не гарантирует получение информации о состоянии программного обеспечения более высокого уровня, которое может выполняться на целевом хосте.

[2]Полная архивная сеть языка Perl, `cpan.org`.

Эхо-запросы не создают больших нагрузок на сеть, поэтому их можно отправлять достаточно часто; скажем, каждые десять секунд. Продумайте свою стратегию эхо-запросов, чтобы она охватывала все важные шлюзы и сети. Имейте в виду, что если эхо-запрос не может пройти через шлюз, то никакая система мониторинга никогда не получит ответных данных, сигнализирующих о сбое в работе целевого хоста. Поэтому по крайней мере один набор эхо-запросов должен исходить из самого центрального узла мониторинга.

Сетевые шлюзы не обязаны отвечать на пакеты эхо-запросов, поэтому эхо-запросы могут быть отброшены перегруженным шлюзом. Даже в правильно функционирующей сети время от времени происходит потеря пакетов. Следовательно, не стоит тревожиться при первых признаках неприятностей. Имеет смысл собирать данные об эхо-запросах в виде двоичных записей событий (прошел–не прошел) и сводить их к совокупным показателям потери пакетов в процентах в течение более долгих периодов.

Вам также может показаться интересным измерить пропускную способность между двумя точками сети. Это можно сделать с помощью программы iPerf; см. раздел 13.13.

В большинстве сетевых устройств поддерживается простой протокол сетевого управления SNMP (Simple Network Management Protocol), представляющий собой промышленный стандарт для именования и сбора оперативных данных. Хотя SNMP ушел далеко за пределы своих сетевых корней, мы считаем его устаревшим для других целей, кроме базового сетевого мониторинга.

SNMP — довольно большая тема, поэтому мы откладываем дальнейшее обсуждение этого вопроса. Дополнительную информацию см. в разделе 28.9.

28.6. Мониторинг систем

Поскольку ядро управляет центральным процессором системы, памятью, вводом-выводом и периферийными устройствами, большая часть интересной информации о состоянии системного уровня, которую вы, возможно, захотите контролировать, находится где-то внутри ядра. Независимо от того, следите ли вы за конкретной системой в ручном режиме или настраиваете автоматическую платформу мониторинга, вам нужны правильные инструменты для извлечения и отображения информации о состоянии системы. В большинстве ядер определены формальные каналы, через которые такая информация экспортируется во внешний мир.

К сожалению, ядра похожи на другие типы программного обеспечения; средства выявления ошибок, инструменты и функции отладки часто запаздывают. Хотя в последнее время ситуация явно улучшилась, определение и понимание точного параметра, который вы хотите контролировать, может быть сложным, а иногда и невозможным делом.

Конкретное значение часто можно получить несколькими способами. Например, в случае усредненных значений загрузки системы вы можете считывать значения непосредственно из файла `/proc/loadavg` в системах Linux или с помощью команды `sysctl -n vm.loadavg` в системе FreeBSD. Средние значения загрузки системы также включаются в выходные данные команд `uptime`, `w`, `sar` и `top` (хотя для неинтерактивного использования команда `top` — неудачный выбор). Как правило, эти команды представляют собой самый простой и эффективный доступ к значениям, полученным непосредственно из ядра системы через `sysctl` или `/proc`.

🕮 Дополнительную информацию о файловой системе `/proc` см. в разделе 11.4.

Мониторинговые платформы, такие как Nagios и Icinga, включают богатый набор дополнительных модулей для мониторинга, разработанных сообществом, которые вы можете использовать для получения доступа к контролируемым элементам. Они также

часто являются просто сценариями, которые запускают команды и анализируют полученный результат, но уже протестированы и отлажены и часто работают на нескольких платформах. Если вы не можете найти дополнительный модуль, который дает интересующее вас значение, вы можете написать собственный.

Команды для мониторинга систем

В табл. 28.2 перечислены несколько команд, которые обычно используются при мониторинге. Многие из этих команд дают совершенно разные результаты в зависимости от указанных вами параметров командной строки, поэтому для уточнения информации обратитесь к справочным страницам **man**.

Таблица 28.2. Команды, которые возвращают часто контролируемые параметры

Команда	Доступная информация
df	Свободное и занятое дисковое пространство и индексные дескрипторы
du	Размеры каталогов
free	Объем свободной, занятой и виртуальной памяти (файла подкачки)
iostat	Производительность диска и пропускная способность
lsof	Открытые файлы и сетевые порты
mpstat	Использование каждого процессора на многопроцессорных системах
vmstat	Статистика процесса, процессора и памяти

Команда **sar** (system activity report) — это команда для извлечения данных из командной строки, так сказать, "швейцарский армейский нож" в области мониторинга деятельности систем. Эта команда имеет очень сложную историю, которая уходит корнями в систему System V UNIX, разработанную в 1980-х годах.[3] Основное внимание в этой команде было направлено на ее реализацию в самых разных системах, что повышает мобильность как сценариев, так и системных администраторов. К сожалению, она больше не поддерживается системах BSD.

Пример, приведенный ниже, запрашивает отчеты каждые две секунды в течение одной минуты (т.е. 30 отчетов). Аргумент **DEV** — это ключевое слово, а не заглушка для имени устройства или интерфейса.

```
$ sar -n DEV 2 30
17:50:43 IFACE rxpck/s txpck/s rxbyt/s txbyt/s rxcmp/s txcmp/s rxmcst/s
17:50:45 lc       3.61     3.61  263.40  263.40    0.00    0.00      0,00
17:50:45 eth0    18.56    11.86 1364.43 1494.33    0.00    0.00      0.52
17:50:45 eth1     0.00     0.00    0.00    0.00    0.00    0.00      0.00
```

Этот пример был выполнен на Linux-машине с двумя сетевыми интерфейсами. Выход включает в себя как мгновенные, так и средние значения использования интерфейса в единицах байтов и пакетов. Второй интерфейс (**eth1**) явно не используется.

Сборщик обобщенных системных данных collectd

По мере эволюции работы по системному администрированию она прошла путь от анализа данных в отдельных системах до управления большими наборами виртуализированных экземпляров. При этом используемые простые инструменты командной стро-

[3] Системных администраторов старой школы часто распознают по свободному владению командой **sar**.

ки начали создавать много сложностей в сфере мониторинга. Хотя написание сценариев для сбора и анализа параметров обеспечивает утилитарность и определенную гибкость, поддержание согласованности этой кодовой базы в нескольких системах быстро становится хлопотным делом. Современные инструменты, такие как **collectd**, **sysdig** и **dtrace**, предлагают более масштабируемый подход к сбору данных этого типа.

Сбор системной статистики должен быть непрерывным процессом, а решение UNIX для текущей задачи — создать демон для его обработки. Для этого рекомендуем использовать **collectd**, демон для сбора системной статистики.

Этот популярный и зрелый инструмент работает как в системе Linux, так и в системе FreeBSD. Как правило, демон **collectd** запускается в локальной системе, собирает показатели через определенные интервалы времени и сохраняет полученные значения. Вы также можете настроить демон **collectd** для запуска в режиме клиент-сервер, где один или несколько экземпляров **collectd** объединяют данные из группы других серверов.

Спецификация собираемых показателей и мест назначения, в которых они сохраняются, является гибкой; доступны более 100 дополнительных модулей, соответствующих самым разным потребностям. После запуска демона **collectd** его данные могут быть запрошены платформой (например, Icinga или Nagios) для мгновенного мониторинга или перенаправлены на платформу (например, Graphite или InfluxDB) для анализа временных рядов.

Пример файла конфигурации демона **collectd** показан ниже.

```
## /etc/collectd/collectd.conf

Hostname client1.admin.com
FQDNLookup false
Interval 30
LoadPlugin syslog
<Plugin syslog>
   LogLevel info
</Plugin>

LoadPlugin cpu
LoadPlugin df
LoadPlugin disk
LoadPlugin interface
LoadPlugin load
LoadPlugin memory
LoadPlugin processes
LoadPlugin rrdtool

<Plugin rrdtool>
   DataDir "/var/lib/collectd/rrd"
</Plugin>
```

Эта базовая конфигурация собирает множество интересных статистических данных системы каждые 30 с и записывает файлы данных, совместимые с RRDtool, в каталог **/var/lib/collectd/rrd**.

Утилиты sysdig и dtrace: трассировки выполнения

Программы **sysdig** (Linux) и **dtrace** (BSD) обеспечивают всесторонний анализ активности ядра и пользователя. Они включают компоненты, которые помещаются в самое ядро, отображая не только глубинные параметры ядра, но и системные вызовы

для каждого процесса и другие статистические данные о производительности. Эти утилиты иногда называют "Wireshark для ядра и процессов".

🕮 Дополнительную информацию о программе Wireshark см. в разделе 13.12.

Обе эти утилиты являются сложными. Однако они заслуживают внимания. Их изучение на досуге позволит вам освоить новые потрясающие возможности и обеспечить ваш высокий рейтинг в среде системных администраторов.

🕮 Дополнительную информацию о системе Docker и контейнерах см. в главе 25.

Инструмент `sysdig` является контейнером, поэтому он обеспечивает исключительную видимость в средах, где используются инструменты, такие как Docker и LXC. Программа `sysdig` распространяется как утилита с открытым исходным кодом, и ее можно сочетать с другими утилитами мониторинга, такими как Nagios или Icinga. Разработчики также предлагают коммерческую службу мониторинга (Sysdig Cloud), которая обладает полной возможностью мониторинга и оповещения.

28.7. Мониторинг приложений

На вершине пирамиды программного обеспечения находится святой Грааль — мониторинг приложений. Этот тип мониторинга определен довольно нечетко, но общая идея заключается в том, что он пытается проверить состояние и производительность отдельных частей программного обеспечения, а не систем или сетей в целом. Во многих случаях мониторинг приложений может достигать этих систем и следить за их функционированием.

Чтобы убедиться, что вы следите за правильными объектами, вам нужны бизнес-консультанты и разработчики, которые могут рассказать вам больше о своих интересах и проблемах. Например, если у вас есть веб-сайт, который работает на основе LAMP, вы, вероятно, захотите контролировать время загрузки страницы, выявить критические ошибки PHP, сохранить эту информацию в базе данных MySQL и отследить конкретные проблемы, такие как чрезмерные попытки подключения.

Хотя мониторинг этого уровня может быть сложным, в данной области он выглядит особенно привлекательным. Представьте себе мониторинг (и красочный вывод на контрольную панель Grafana) данных о количестве виджетов, проданных за последний час, или среднем периоде времени, в течение которого товар остается в корзине покупок. Если вы продемонстрируете разработчикам приложений и владельцам процессов этот уровень функциональности, то немедленно получите заказ на дополнительный мониторинг и даже можете получить помощь в его реализации. В конце концов, этот уровень мониторинга становится бесценным для бизнеса, и вы начинаете рассматриваться как чемпион по мониторингу, метрикам и анализу данных.

Мониторинг уровня приложений может дать дополнительное представление о других событиях в вашей среде. Например, если продажи виджета быстро снизятся, это может свидетельствовать о том, что одна из рекламных сетей недоступна.

Мониторинг системного журнала

В своей основной форме мониторинг системных журналов включает в себя просмотр файлов журналов в поисках интересных данных, которые вы хотите отслеживать, извлечение этих данных и представление их в виде, пригодном для анализа, отображения и оповещения. Поскольку сообщения в системном журнале состоят из текста в свобод-

ной форме, сложность реализация этого конвейера может варьироваться от тривиальной до высокой.

Журналы обычно лучше всего анализировать с помощью комплексной системы агрегации, предназначенной для этой цели. Такие системы рассматривались в разделе 10.6. Хотя эти системы ориентированы прежде всего на централизацию журнальных данных и упрощение поиска и просмотра, большинство систем агрегации также поддерживают функции порога, тревоги и отчетности.

Если вам нужен автоматический анализ системных журналов для нескольких конкретных целей и вы не хотите связываться с более общими решениями для управления журналами, мы рекомендуем несколько утилит меньшего масштаба — **logwatch** и OSSEC.

Программа **logwatch** — это гибкий, пакетно-ориентированный агрегатор журналов. Его основное предназначение — создавать ежедневные сводки событий, регистрируемых в журналах. Вы можете запускать программу **logwatch** чаще, чем один раз в день, но он не предназначен для мониторинга в реальном времени. Для этого вы, вероятно, захотите взглянуть на программу OSSEC, которая рассматривалась в разделе 27.5. Программа OSSEC рекламируется как средство обеспечения безопасности системы, но ее архитектура является достаточно общей, что также можно использовать для других видов мониторинга.

Supervisor + Munin: простой вариант для некоторых предметных областей

Универсальная платформа, такая как Icinga или Prometheus, может быть избыточной для ваших нужд или вашей среды. Что делать, если вы заинтересованы только в мониторинге одного конкретного процесса приложения и не хотите устанавливать полноценную платформу мониторинга? Рассмотрите возможность объединения программ Munin и Supervisor. Они просты в установке, требуют лишь небольшой настройки и хорошо работают вместе.

Программа Supervisor и ее серверный процесс **supervisord** помогают отслеживать процессы и генерировать события или уведомления, когда процессы заканчивают работу или вызывают исключение. Система похожа по духу на программу Upstart или на демон **systemd** в части управления процессами.

Как упоминалось выше, Munin является общей платформой мониторинга с особыми сильными сторонами в мониторинге приложений. Она написана на Perl и требует, чтобы агент, называемый узлом Munin, работал на всех системах, которые вы хотите контролировать. Настройка нового узла проста: установите пакет **munin-node** и отредактируйте файл **munin-node.conf**, чтобы указать в нем главную машину.

Программа Munin по умолчанию создает графики RRDtool на основе собранных данных, поэтому она является хорошим средством для получения некоторой графической информации без сложных приготовлений. Вместе с программой Munin распространяются более 300 подключаемых модулей и около 200 других доступны в виде библиотек. Вероятно, вы можете найти существующий подключаемый модуль, соответствующий вашим потребностям. Если нет, легко написать новый сценарий для пакета **munin-node**.

Коммерческие средства мониторинга приложений

Если ввести в поисковую систему Google слова “application monitoring tool” (“средство мониторинга приложений”), вы обнаружите множество страниц с коммерчески-

ми предложениями. Но сначала вам придется продраться через множество сообщений о мониторинге производительности приложений (APM).

Вы увидите много ссылок на методологию DevOps. Это не случайно: мониторинг приложений и APM являются ключевыми принципами DevOps. Они предоставляют количественные показатели, которые команды могут использовать для определения того, какие области стека будут наиболее полезны для усилий по повышению производительности и стабильности.

🕮 Дополнительную информацию о методологии DevOps см. в разделе 31.1.

Мы считаем, что компании New Relic (`newrelic.com`) и AppDynamics (`appdynamics.com`) являются лидерами в этой области. Возможности этих систем во многих отношениях перекрываются, но AppDynamics обычно нацеливается больше на решение "полного стека", в то время как New Relic больше занимается профилированием поведения внутри самого прикладного уровня.

Независимо от того, как вы контролируете свои приложения, крайне важно, чтобы команда разработчиков участвовала в этом процессе. Они помогут обеспечить мониторинг всех важных показателей. Тесное сотрудничество по мониторингу способствует взаимосвязи между командами и устраняет дублирование.

28.8. Мониторинг безопасности

Мониторинг безопасности — это отдельный мир. Эту область эксплуатационной практики иногда называют операциями по обеспечению безопасности, или SecOps.

Для мониторинга безопасности среды могут быть привлечены десятки инструментов с открытым исходным кодом, а также коммерческих инструментов и служб. Третьи стороны, иногда называемые *управляемыми поставщиками услуг безопасности* (managed security service providers — MSSP), предоставляют услуги сторонних поставщиков.[4] Несмотря на все эти варианты, нарушения безопасности все еще широко распространены и часто остаются незамеченными в течение нескольких месяцев или лет.

Возможно, самое важное, что нужно знать о мониторинге безопасности, — что недостаточно просто установить автоматизированный инструмент или службу. Вы *обязаны* внедрить комплексную программу обеспечения безопасности, которая включает, как минимум, стандарты для поведения пользователей, хранения данных и процедур реагирования на инциденты. Этой теме посвящена глава 27.

В вашей автоматизированной стратегии непрерывного мониторинга должны быть две функции по обеспечению безопасности: проверка целостности системы и обнаружение вторжений.

Проверка целостности системы

Проверка целостности системы (часто называемая мониторингом целостности файлов или FIM (file integrity monitoring)) — это сравнение текущего состояния системы с базовым состоянием. Чаще всего эта проверка сравнивает содержимое системных файлов (ядро, исполняемые команды, файлы конфигурации) с криптографически на-

[4]Всегда существует соблазн отдать операции по обеспечению безопасности на аутсорсинг. В этом случае обеспечение безопасности вашей среды становится чужой проблемой. Но подумайте об этом так: стоит ли платить кому-то, чтобы тот присматривал за вашим кошельком, лежащим на столе с 10 000 других кошельков на оживленной железнодорожной станции? Если да, то MSSP вам подходит!

дежной контрольной суммой, такой как SHA-512.[5] Если значение контрольной суммы файла в текущей системе отличается от значения базовой версии, системный администратор получает уведомление. Разумеется, необходимо учитывать регулярные мероприятия по техническому обслуживанию, такие как запланированные изменения, обновления и исправления; не все изменения являются подозрительными.

Наиболее распространенными платформами FIM являются Tripwire и OSSEC; последняя более подробно описывается в разделе 27.5. Версия AIDE для Linux также включает мониторинг целостности файлов, но, к сожалению, в версии для FreeBSD этот компонент отсутствует.

Чем проще — тем лучше. Отличная минимальная версия FIM — утилита **mtree** , которая входит в поставку системы FreeBSD и недавно была перенесена в систему Linux. Утилита **mtree** — простой способ отслеживать изменения состояния файла и его содержимого, который легко встраивается в ваши сценарии мониторинга.

Ниже приведен пример простейшего сценария, использующего утилиту **mtree**.

```
#!/bin/bash
if [ $# -eq 0 ]; then
   echo "mtree-check.sh [-bv]"
   echo "-b = create baseline"
   echo "-v = verify against baseline"
   exit
fi

## значение ключа
KEY=93948764681464

## каталог для хранения базового состояния
DIR=/usr/local/lib/mtree-check

if [ $1 = "-b" ]; then
   rm -rf $DIR/mtree_*
   cd $DIR
   mtree -c -K sha512 -s $KEY -p /sbin > mtree_sbin
fi

if [ $1 = "-v" ]; then
   cd $DIR
   mtree -s $KEY -p /sbin < mtree_sbin | \
      mail -s "`hostname` mtree integrity check" dan@admin.com
fi
```

С помощью флага **-b** этот сценарий создает и сохраняет базовое состояние. При повторном запуске с флагом **-v** он сравнивает текущее содержимое каталога **/sbin** с базовым состоянием.

Как и во многих аспектах системного администрирования, создание платформы FIM и управление ею с течением времени являются совершенно разными проблемами. Вам понадобится определенный процесс, чтобы поддерживать данные FIM и реагировать на предупреждения FIM. Мы предлагаем загружать информацию с платформы FIM в вашу инфраструктуру мониторинга и оповещения, чтобы она не отодвигалась в сторону и не игнорировалась.

[5]Приемлемые алгоритмы хеширования со временем меняются. Например, алгоритм MD5 больше не считается криптографически защищенным и в дальнейшем не должен использоваться.

Контроль обнаружения вторжений

Используются две распространенные формы систем обнаружения вторжений (intrusion detection systems — IDS): на основе хоста (host-based IDS — HIDS) и на основе сети (network-based IDS — NIDS). Системы NIDS исследуют трафик, проходящий через сеть, и пытаются идентифицировать неожиданные или подозрительные шаблоны его поведения. Наиболее распространенная система NIDS основана на системе Snort и подробно освещена в разделе 27.5.

Системы HIDS работают как набор процессов, выполняемых в каждой системе. Как правило, они ведут таблицы, в которых записана разная информация, в частности сетевые соединения, моменты изменения файлов и контрольные суммы, журналы демонов и приложений, факты использования повышенных привилегий и другие моменты, которые могут сигнализировать о работе средств, предназначенных для предоставления несанкционированного доступа к системе (так называемые руткиты). Системы HIDS не являются универсальным решением для обеспечения безопасности, но это ценный компонент комплексного подхода.

Двумя наиболее популярными платформами HIDS с открытым исходным кодом являются OSSEC (Open Source SECURE) и AIDE (Advanced Intrusion Detection Environment). По нашему опыту, OSSEC — лучший выбор. Хотя AIDE — хорошая платформа FIM в системе Linux, платформа OSSEC имеет более широкий набор функций. Ее можно даже использовать в режиме клиент-сервер, который поддерживают не-UNIX-клиенты, такие как Microsoft Windows и различные устройства сетевой инфраструктуры.

Подобно предупреждениям FIM, данные HIDS полезны, только если им уделяется внимание. HIDS — это не подсистема, работающая по принципу "установил и забыл". Вы обязаны интегрировать предупреждения HIDS и вашу общую систему мониторинга. Наиболее эффективная стратегия, которую мы нашли для решения этой проблемы, — автоматическое открытие билетов на оповещения HIDS в вашей системе выдачи билетов. Затем вы можете добавить контрольную проверку, которая предупреждает о любых необработанных билетах HIDS.

28.9. SNMP: простой протокол сетевого управления

Много лет назад сетевая индустрия решила, что было бы полезно создать стандартный протокол сбора данных мониторинга. В результате возник протокол Simple Network Management Protocol, также известный как SNMP.

Несмотря на свое название, протокол SNMP на самом деле довольно сложный. Он определяет иерархическое пространство имен данных управления и методы для чтения и записи этих данных на каждом сетевом устройстве. Протокол SNMP также определяет способ управления серверами и устройствами ("агентами") для отправки уведомлений о событиях ("ловушек") на станции управления.

Прежде чем мы углубимся в тайны протокола SNMP, следует отметить, что связанная с ним терминология — одна из самых неудачных в мире сетей. Во многих случаях стандартные названия концепций и объектов в протоколе SNMP активно уводят вас от понимания того, что происходит.

Тем не менее сам протокол прост; большая часть сложности SNMP лежит выше уровня протокола в соглашениях о построении пространства имен и ненужного вычурного словаря, который окружает SNMP как защитная оболочка. Если не слишком много думать о его внутреннем устройстве, протокол SNMP прост в использовании.

Протокол SNMP был разработан для реализации в специализированном сетевом оборудовании, таком как маршрутизаторы, в контексте которых он остается вполне приемлемым. Позднее SNMP расширился, включив мониторинг серверов и настольных систем, но его пригодность для этой цели всегда вызывала сомнения. Сегодня доступны гораздо более эффективные альтернативы (например, `collectd`, см. выше).

Мы предлагаем использовать SNMP в качестве протокола сбора данных низкого уровня для использования с устройствами специального назначения, которые не поддерживают ничего лишнего. Получайте данные из мира SNMP как можно быстрее и переходите к платформе мониторинга общего назначения для их хранения и обработки. SNMP может быть интересным объектом экскурсии, но вы не захотите там жить!

Организация протокола SNMP

Данные SNMP организованы в виде стандартизованной иерархии. Иерархия именования состоит из так называемых "баз управляющей информации" (Management Information Bases — MIB) — структурированных текстовых файлов, которые описывают данные, доступные через SNMP. Базы MIB содержат описания конкретных переменных, на которые ссылаются имена, называемые *объектными идентификаторами* (OID)[6]. Все текущие устройства, поддерживающие протокол SNMP, поддерживают структуру для MIB-II, определенную в спецификации RFC1213. Однако каждый производитель оборудования может расширить MIB, чтобы добавить больше данных и показателей, и часто пользуется этой возможностью.

Идентификаторы OID существуют в иерархическом пространстве имен, где узлы нумеруются, а не называются. Однако для удобства ссылок узлы также имеют обычные текстовые имена. Разделителем для компонентов пути является точка. Например, OID, который указывает на время безотказной работы устройства, — 1.3.6.1.2.1.1.3. Этот OID также может быть представлен в виде имени, понятного для человека (хотя и не всегда понятного без дополнительного описания):

```
iso.org.dod.internet.mgmt.mib-2.system.sysUpTime
```

В табл. 28.3 представлена выборка OID-узлов, которые могут быть интересны для мониторинга доступности сети.

Таблица 28.3. Избранные идентификаторы OID из базы MIB-II

OID[a]	Тип	Содержание
`system.sysDescr`	Строка	Системная информация: производитель, модель, тип операционной системы и т.д.
`interfaces.ifNumber`	Целое число	Количество сетевых интерфейсов
`interfaces.ifTable`	Таблица	Таблица инфобитов о каждом интерфейсе
`ip.ipForwarding`	Целое число	Равно 1, если система является шлюзом; в противном случае 2
`ip.ipAddrTable`	Таблица	Таблица данных IP-адресации (маски и т.д.)
`icmp.icmpInRedirects`	Целое число	Количество полученных переадресаций ICMP
`icmp.icmpInEchos`	Целое число	Количество полученных эхо-запросов
`tcp.tcpInErrs`	Целое число	Количество полученных ошибок протокола TCP

[a] Относительно `iso.org.dod.internet.mgmt.mib-2`.

В дополнение к универсально поддерживаемой базе MIB-II существуют базы MIB для различных типов аппаратных интерфейсов и протоколов, отдельных производителей оборудования, различных реализаций сервера **snmpd** и конкретных аппаратных продуктов.

База MIB — это всего лишь схема для управления данными об именах. Чтобы быть полезной, база MIB должна сопровождаться агентским кодом, который выполняет отображение между пространством имен SNMP и фактическим состоянием устройства.

Агент SNMP, работающий в системах UNIX, Linux или Windows, имеет встроенную поддержку баз MIB-II. Большинство из них могут быть расширены для поддержки дополнительных MIB и для взаимодействия со сценариями, которые выполняют фактическую работу по извлечению и хранению связанных данных MIB. Вы увидите много программного обеспечения, которое осталось от ушедшей эпохи, когда протокол SNMP был в моде. Но все это дым без огня; в настоящее время вы не должны запускать SNMP-агент в системе UNIX, за исключением, возможно, случая, когда он должен отвечать на основные запросы о настройке сети.

Операции протокола SNMP

Существуют только четыре основные операции SNMP: `get`, `get-next`, `set` и `trap`.

Операции `get` и `set` — это основные операции для чтения и записи данных на узел, определенный идентификатором OID. Операция `get-next` выполняет обход иерархии MIB и читает содержимое таблиц.

Операция `trap` — это незапрашиваемое асинхронное уведомление от сервера (агента) клиенту (менеджеру), которое сообщает о возникновении интересного события или ситуации. Определены несколько стандартных уведомлений, в том числе уведомления "Я только что включился", отчеты об отказе или восстановлении сетевого соединения, а также анонсы о различных проблемах маршрутизации и аутентификации. Механизм, с помощью которого определяются адресаты сообщений `trap`, зависит от реализации агента.

Поскольку сообщения SNMP могут изменять информацию о конфигурации, необходим какой-то механизм защиты. В простейшей версии защиты протокола SNMP используется концепция "общей строки" (community string), которая на самом деле представляет собой просто ужасно запутанное название пароля. Обычно есть одна общая строка доступа только для чтения и другая — для записи.[7] В наши дни имеет смысл установить инфраструктуру управления SNMPv3, позволяющую повысить систему защиты данных за счет авторизации и контроля доступа для отдельных пользователей.

Net-SNMP: средства для серверов

В системах Linux и FreeBSD наиболее распространенный пакет, реализующий протокол SNMP, называется Net-SNMP. Он включает в себя агент (**snmpd**), некоторые утилиты командной строки, сервер для приема уведомлений `trap` и даже библиотеку для разработки приложений, поддерживающих протокол SNMP.

В наши дни пакет Net-SNMP в первую очередь представляет интерес из-за его команд и библиотек, а не его агента. Он был перенесен на множество разных UNIX-подобных систем, поэтому действует как согласованная платформа, поверх которой можно писать сценарии. Поэтому в большинстве дистрибутивов агент Net-SNMP просто вынесен в отдельный пакет, что облегчает отдельную инсталляцию только команд и библиотек.



В системах Debian и Ubuntu пакеты Net-SNMP называются **snmp** и **snmpd**. Команды инсталлируются так: **apt-get install snmp**.

RHEL  В системах Red Hat и CentOS аналогичные пакеты называются **net-snmp** и **net-snmp-tools**. Команды инсталлируются так:**yum install net-snmp-tools**.



В системе Linux информация о конфигурации хранится в каталоге **/etc/snmp**; обратите внимание на файл **snmpd.conf** и каталог **snmp.d** в этом месте. Выполните команду **systemctl start snmpd**, чтобы запустить демон агента.



В системе FreeBSD все включено в один пакет: **pkg install net-snmp**. Информация о конфигурации хранится в каталоге в **/usr/local/etc/snmp**, который вам придется создать вручную. Вы можете запустить агент вручную вместе со службой **snmpd** или установить значение

```
snmpd_enable = "YES"
```

в файле **/etc/rc.conf**, чтобы запустить его во время загрузки системы.

Во всех системах, где необходимо запустить агент SNMP, вам необходимо убедиться, что порт 162 протокола UDP не заблокирован брандмауэром.

Команды, которые входят в пакет Net-SNMP, могут ознакомить вас с протоколом SNMP. Они также отлично подходят для однократных проверок определенных идентификаторов OID. Наиболее часто используемые средства перечислены в табл. 28.4.

Таблица 28.4. Средства командной строки в пакете Net-SNMP

Команда	Назначение
snmpdelta	Мониторинг изменений SNMP-переменных во времени
snmpdf	Мониторинг дискового пространства на удаленном хосте через протокол SNMP
snmpget	Возвращает значение SNMP-переменной от агента
snmpgetnext	Получает следующую переменную в последовательности
snmpset	Устанавливает SNMP-переменную на агенте
snmptable	Получает таблицу переменных SNMP
snmptranslate	Ищет и описывает идентификаторы OID в иерархии MIB
snmptrap	Генерирует предупреждение об уведомлении `trap`
snmpwalk	Выполняет обход базы MIB, начиная с определенного идентификатора OID

Для базовых проверок SNMP обычно используется некоторая комбинация команд **snmpget** и **snmpdelta**. Другие программы полезны, когда вы хотите найти новые идентификаторы OID для мониторинга с помощью вашего средства управления, принятого в вашей организации. Например, команда **snmpwalk** запускается с указанного идентификатором OID места (или по умолчанию с начала базы MIB) и повторно выполняет команды `get-next`, посылая вызовы агенту. Эта процедура выдает полный список доступных идентификаторов OID и связанных с ними значений.

Например, вот усеченный пример результатов применения команды **snmpwalk** к хосту **tuva** системы Linux. Общая строка — **"secret813community"**, а флаг **-v1** означает простую аутентификацию.

```
$ snmpwalk -c secret813community -v1 tuva
SNMPv2-MIB::sysDescr.0 = STRING: Linux tuva.atrust.com 2.6.9-11.ELsmp #1
SNMPv2-MIB::sysUpTime.0 = Timeticks: (1442) 0:00:14.42
SNMPv2-MIB::sysName.0 = STRING: tuva.atrust.com
```

```
IF-MIB::ifDescr.1 = STRING: lo
IF-MIB::ifDescr.2 = STRING: eth0
IF-MIB::ifDescr.3 = STRING: eth1
IF-MIB::ifType.1 = INTEGER: softwareLoopback(24)
IF-MIB::ifType.2 = INTEGER: ethernetCsmacd(6)
IF-MIB::ifType.3 = INTEGER: ethernetCsmacd(6)
IF-MIB::ifPhysAddress.1 = STRING:
IF-MIB::ifPhysAddress.2 = STRING: 0:11:43:d9:1e:f5
IF-MIB::ifPhysAddress.3 = STRING: 0:11:43:d9:1e:f6
IF-MIB::ifInOctets.1 = Counter32: 2605613514
IF-MIB::ifInOctets.2 = Counter32: 1543105654
IF-MIB::ifInOctets.3 = Counter32: 46312345
IF-MIB::ifInUcastPkts.1 = Counter32: 389536156
IF-MIB::ifInUcastPkts.2 = Counter32: 892959265
IF-MIB::ifInUcastPkts.3 = Counter32: 7712325
...
```

В этом примере общая информация о системе сопровождается статистикой сетевых интерфейсов хоста: **lo0**, **eth0** и **eth1**. В зависимости от баз MIB, поддерживаемых агентом, которым вы управляете, полный дамп может содержать до сотен строк. Фактически полная инсталляция в системе Ubuntu, настроенной для обслуживания каждой базы MIB, выводит более 12 000 строк!

Если вы просмотрите базы MIB[8] для последней версии Net-SNMP в системе Ubuntu, то увидите, что средний за пять минут показатель загрузки системы равен 1.3.6.1.4.1.2021.10.1.3.2. Если вам интересно увидеть пятиминутную загрузку системы для локального хоста, для которого задана общая строка **"public"**, вы должны выполнить команду:

```
$ snmpget -v 2c -c public localhost .1.3.6.1.4.1.2021.10.1.3.2

iso.3.6.1.4.1.2021.10.1.3.2 = STRING: "0.08"
```

Многие полезные модули SNMP, связанные с языками Perl, Ruby и Python, доступны из соответствующих репозиториев модулей этих языков. Хотя вы можете в своих сценариях непосредственно указывать команды пакета Net-SNMP, проще и легче использовать встроенные модули, которые уже написаны для вашего языка программирования.

28.10. Советы и рекомендации по мониторингу

На протяжении многих лет мы выработали несколько советов о том, как максимизировать эффективность мониторинга. Ниже изложены основные идеи.

- Избегайте выгорания системных администраторов, проводящих мониторинг. Убедитесь, что системные администраторы, получающие уведомления во внеурочное время, регулярно отдыхают. Эта цель лучше всего достигается с помощью системы ротации, в которой в течение одного дня или недели звонки адресуются команде из двух или более человек, а затем наступает очередь следующей команды. Неспособность прислушаться к этим советам приводит к появлению ворчливых системных администраторов, которые ненавидят свою работу.
- Определите, какие обстоятельства действительно требуют внимания 24 часа в сутки 7 дней в неделю, и убедитесь, что эта информация четко передается вашей группе мониторинга, дежурным командам, а также клиентам или бизнес-подраз-

[8] Зайдите на сайт `mibdepot.com` или установите пакет **snmp-mibs-downloader**.

делениям, работу которых вы поддерживаете. Сам факт, что вы контролируете что-то, не означает, что администраторы должны собираться в 3:30 утра, если некоторое значение выходит за рамки допустимого. Многие проблемы следует решать в обычное рабочее время.

- Устраняйте шум мониторинга. Если генерируются ложные срабатывания или уведомления для некритических служб, установите время, чтобы остановить и исправить их. В противном случае, как в истории о мальчике, который постоянно кричал о волках, все уведомления в конечном итоге не получат необходимого внимания.
- Создавайте рабочие журналы регистрации для всего. Любая обычная процедура перезапуска, сброса или исправления должна быть задокументирована в форме, которая позволяет системному администратору, который не знаком с данной системой, предпринять соответствующие действия. Если такой документации нет, проблемы станут хроническими, повысится вероятность ошибок и для борьбы с чрезвычайными ситуациями придется направлять дополнительный персонал. Для поддержки такого типа документации отлично подходит система Wiki.
- Контролируйте платформу мониторинга. Это станет очевидным после того, как вы пропустите критический сбой из-за выхода из строя платформы мониторинга. Учитесь на наших ошибках и убедитесь, что за вами тоже кто-то наблюдает.
- Пропустили сбой компонента из-за того, что он не контролировался? Добавьте его в систему мониторинга и в следующий раз вы вовремя распознаете проблему.
- Наконец, и, возможно, самое главное: ни сервер, ни служба не должны включаться в производственную цепочку без предварительного добавления в систему мониторинга. Исключений здесь быть не может!

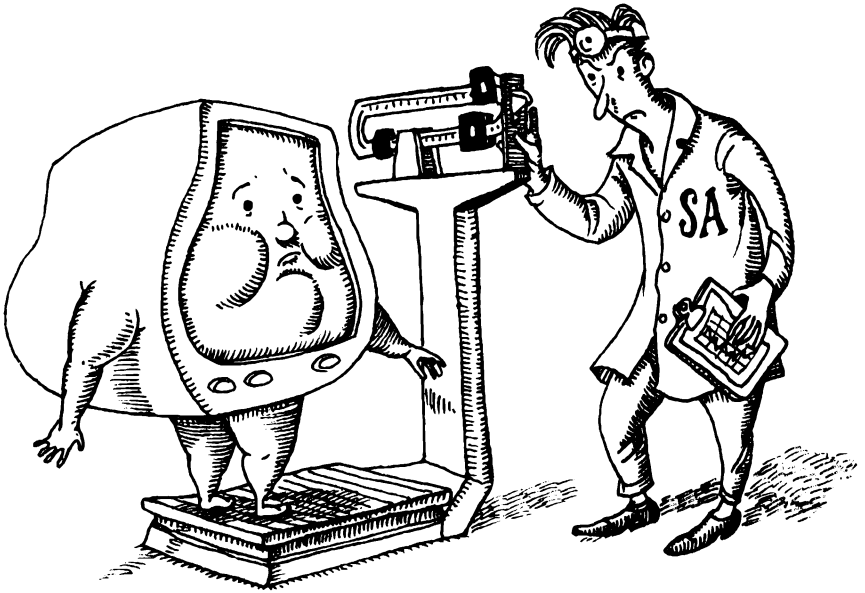
28.11. Литература

- Hecht, James. *Rethinking Monitoring for Container Operations*. Отличная информация о стратегии и философии мониторинга контейнеров. С этой книгой можно ознакомиться по адресу

  `http://thenewstack.io/monitoring-reset-containers/`
- Turnbull, James. *The Art of Monitoring*. Seattle, WA: Amazon Digital Services, 2016.
- Dixon, Jason. *Monitoring with Graphite: Tracking Dynamic Host and Application Metrics at Scale*. Sebastopol, CA: O'Reilly Media, 2017.

Глава 29

Анализ производительности



Анализ производительности и ее настройку часто относят к магической стороне системного администрирования. Конечно же, магии здесь никакой нет, а следует говорить о симбиозе науки и искусства. "Научная" часть включает тщательное проведение количественных измерений и применение научного подхода. Составляющая "искусства" связана с необходимостью сохранять баланс ресурсов на разумном уровне, поскольку оптимизация для одного приложения или пользователя может отрицательно повлиять на другие приложения или на других пользователей. Как это часто бывает в жизни, невозможно сделать счастливыми всех сразу.

В блогосфере бытует мнение, что сегодня проблемы производительности кардинально отличаются от аналогичных проблем предыдущих десятилетий. Но это не совсем так. Да, системы стали гораздо сложнее, но определяющие факторы производительности и высокоуровневые абстракции, используемые для ее измерения и управления, не меняются. К сожалению, повышение производительности систем сильно коррелирует со способностью соответствующего сообщества создавать новые приложения, которые поглощают все доступные ресурсы.

В последние годы дополнительную сложность создает многоуровневая абстракция, которая часто располагается между серверами и физической инфраструктурой облака. Часто невозможно точно знать, какое оборудование обеспечивает хранение или выполнение циклов центрального процессора на вашем сервере.

Магия и проблема облачных технологий — это два аспекта одного и того же вида. Несмотря на распространенное мнение, вы не можете игнорировать вопросы произво-

дительности только потому, что ваши серверы являются виртуальными. Фактически модели биллинга, используемые провайдерами облаков, создают еще более прямую связь между производительностью и затратами на сервер. Знать, как измерять и оценивать производительность, стало более важным, чем когда-либо.

В этой главе основное внимание уделяется производительности систем, которые используются в качестве серверов. Настольные системы (и ноутбуки) обычно не испытывают тех же проблем с производительностью, что и серверы. Ответ на вопрос о том, как повысить производительность на настольном компьютере, почти всегда сводится к совету обновить аппаратное обеспечение. Пользователям нравится этот ответ, потому что благодаря ему они чаще получают новые интересные системы.

29.1. Принципы настройки производительности

Одним из аспектов, отличающих UNIX и Linux от других популярных операционных систем, является объем данных, которые они предоставляют о своих внутренних операциях. Детальная информация доступна буквально по каждому уровню системы, благодаря чему администраторы могут настраивать различные параметры именно так, как нужно. Если установить источник проблем с производительностью никак не удается, всегда можно просмотреть исходный код. По этим причинам UNIX и Linux часто считаются наиболее подходящими операционными системами для пользователей, которых особенно беспокоит тема производительности.

Даже при этих условиях настройка производительности не является простой задачей. Как пользователи, так и администраторы часто думают, что если бы они владели нужными "магическими" приемами, их системы работали бы в два раза быстрее. Однако так бывает очень редко.

Одним из наиболее распространенных заблуждений является убеждение в том, что для настройки производительности нужно изменить значение переменных ядра, которые отвечают за управление системой страничного обмена и буферными пулами. В наши дни ядра основных дистрибутивов изначально настроены на достижение разумной (хотя и не оптимальной) производительности при различных уровнях загруженности. Если попытаться оптимизировать систему по какому-то конкретному показателю производительности (например, по степени использования буферов), весьма вероятно, что поведение системы ухудшится в отношении других показателей и режимов работы.

Наиболее сложные вопросы производительности часто связаны с приложениями, а не операционными системами. В этой главе будет рассказываться о том, как можно настроить производительность на уровне системы; возможность рассказать о настройке производительности на уровне приложений мы оставили другим. Как системному администратору, вам необходимо помнить о том, что разработчики — тоже люди. (Сколько раз вы говорили или думали, что проблема кроется в сети?) Так и разработчики приложений тоже часто предполагают, что все проблемы возникают в подсистемах, за которые отвечает кто-то другой.

При таком уровне сложности современных приложений некоторые проблемы можно решить только посредством сотрудничества их разработчиков, системных администраторов, проектировщиков серверов, администраторов баз данных, администраторов систем хранения данных и архитекторов сети. В этой главе мы поможем вам определить, какие данные следует предъявить этим специалистам, чтобы они смогли решить проблемы производительности (если в действительности эти проблемы лежат в их области).

Такой подход гораздо эффективнее, чем просто сказать: "Все выглядит прекрасно, это не моя проблема".

В любом случае никогда не доверяйте полностью тому, что пишут в Интернете. Какие только аргументы не приводятся в дискуссиях о производительности систем! При этом сторонники всевозможных теорий не обладают, как правило, ни знаниями, ни дисциплиной, ни временем, необходимыми для проведения достоверных экспериментов. Широкая поддержка пользователей еще ничего не значит, ибо каждое глупое предложение очень часто сопровождается хором восторженных комментариев: "Я увеличил размер кеш-буфера в десять раз, как он предложил, и система заработала *гораздо*, *гораздо* быстрее!" Да уж, конечно!

Ниже перечислены ряд правил, которыми следует пользоваться.

- Собирайте и анализируйте *хронологические* данные о работе системы. Если еще неделю назад производительность системы была нормальной, а сейчас все изменилось, то сравнение характеристик системы в двух состояниях позволит выявить источник проблемы. Всегда держите под рукой список оптимальных параметров производительности. Кроме того, первым делом просматривайте журнальные файлы, чтобы выяснить, не связана ли возникшая проблема с появлением какой-нибудь неполадки в оборудовании.
- В главе 28 рассказывалось о некоторых средствах анализа тенденций, которые также подходят для мониторинга производительности.
- Всегда настраивайте систему так, чтобы потом иметь возможность сравнить результаты с предыдущими показателями производительности системы.
- Не перегружайте умышленно свои системы или свою сеть. Ядро создает для каждого процесса иллюзию бесконечности ресурсов. Но как только в дело вступают все 100% ресурсов системы, операционной системе приходится работать очень напряженно. На поддержку иллюзии расходуется ощутимая доля самих ресурсов. В результате выполнение процессов замедляется.
- Как в физике элементарных частиц, чем больше данных вы собираете с помощью утилит мониторинга системы, тем большее влияние вы оказываете на исследуемую систему. Для наблюдения за системой лучше всего полагаться на простые инструменты, которые работают в фоновом режиме (например, **sar** или **vmstat**). Если они обнаружат что-то существенное, то для более глубокого анализа можете воспользоваться и другими инструментами.
- Вносите изменения шаг за шагом и документируйте их. Прежде чем вносить очередное изменение, измерьте показатели, запишите их и обдумайте результаты.
- Всегда продумывайте план отката на случай, если ваши изменения на самом деле ухудшат работу системы.

29.2. Способы повышения производительности

Ниже перечислены конкретные действия, которые можно выполнить, чтобы улучшить производительность.

- Убедитесь в том, что память доступна в достаточном объеме. В следующем разделе вы увидите, что объем памяти очень сильно влияет на производительность. Устройства памяти сегодня стоят довольно недорого, так что обычно оснастить по максимуму любой компьютер, от которого требуется высокая производитель-

ность, — не проблема. Если ваш сервер работает в облаке, то изменить объем памяти, выделенный его экземпляру, обычно очень легко (хотя часто он связан с другими выделяемыми ресурсами в полном профиле системы).

- Там, где это возможно, ликвидируйте зависимость систем хранения данных от механических устройств. В настоящее время широко доступны твердотельные диски (solid state disk drive — SSD), которые могут обеспечить быстрый рост производительности, поскольку для них не требуется физическое вращение дискоа для считывания битов информации. SSD-диски легко устанавливаются вместо "старых добрых" дисковых накопителей.
- Если UNIX- или Linux-система используется в качестве веб-сервера или сетевого сервера приложений, можно попробовать распределить трафик между несколькими компьютерами с помощью какого-нибудь (физического или виртуального) балансировщика нагрузки. Эти устройства распределяют нагрузку согласно одному из нескольких выбираемых пользователем алгоритмов типа "минимальное время отклика" или "циклический алгоритм обслуживания".

  Кроме того, балансировщики нагрузки добавляют полезную избыточность на случай перегрузки сервера. Они просто необходимы, если ваш сервер вынужден обрабатывать неожиданные всплески трафика.
- Тщательно проверяйте конфигурацию системы и отдельных приложений. Многие приложения могут резко увеличить производительность, если их правильно настроить (распределить данные между дисками, не осуществлять динамический поиск имен в DNS, запустить дополнительные экземпляры сервера и т.д.).
- Устраните проблемы, связанные с использованием ресурсов. Проблемы могут создаваться как "реальными заданиями" (одновременный запуск слишком большого количества серверов, неэффективные методы программирования, выполнение пакетов заданий с завышенным приоритетом, запуск ресурсоемких программ в неподходящее время), так и самой системой (ненужные демоны).
- Организуйте жесткие диски и файловые системы так, чтобы нагрузка на них была равномерной, и тем самым максимально повысьте пропускную способность каналов ввода-вывода. При наличии специфических приложений, таких как базы данных, можно попробовать использовать какую-нибудь модную многодисковую технологию вроде RAID для оптимизации процессов обмена данными. За рекомендациями следует обращаться к разработчику базы данных. Для систем Linux нужно удостовериться в том, что для диска выбран самый подходящий из доступных в Linux планировщик ввода-вывода (детали см. в разделе 29.6).

  Обратите внимание на то, что разные приложения и системы управления базами данных ведут себя по-разному при размещении на нескольких дисках. Технология RAID существует в нескольких вариантах, поэтому следует потратить время и выяснить, какой из них лучше всего подходит для данного конкретного случая (если вообще подходит).
- Проведите мониторинг сети и убедитесь в том, что она не перегружена трафиком и что количество ошибок в ней минимально. Очень много такой полезной информации о сети можно собрать с помощью команды **netstat** (FreeBSD) и **ss** (Linux).
- Определите ситуации, в которых система совершенно не соответствует предъявляемым к ней требованиям. Нельзя настроить производительность без учета таких ситуаций.

Эти меры перечислены в порядке убывания эффективности. Добавление памяти и распределение трафика между несколькими серверами может значительно повысить производительность. Степень эффективности остальных мер варьируется от значительной до нулевой.

Анализ и оптимизация структур данных и алгоритмов программ почти всегда ведет к существенному повышению производительности. Однако эти вопросы обычно находятся вне компетенции системного администратора.

29.3. Факторы, влияющие на производительность

Производительность системы во многом определяется базовыми характеристиками системных ресурсов и эффективностью их распределения и совместного использования. Определение термина *ресурс* весьма расплывчато. Оно может подразумевать даже такие компоненты, как кеш содержимого регистров центрального процессора и элементы таблицы адресов контроллера памяти. Однако в первом приближении серьезное влияние на производительность оказывают только четыре фактора:

- время использования центрального процессора и его захваченные циклы (см. ниже);
- память;
- пропускная способность дисковой подсистемы ввода-вывода;
- пропускная способность сетевой подсистемы ввода-вывода.

Если после запуска активных процессов остались свободные ресурсы, можно считать, что производительность системы является удовлетворительной.

Когда ресурсов не хватает, процессы становятся в очередь. Процесс, не имеющий немедленного доступа к необходимым ресурсам, должен ждать, ничего при этом не делая. Время, затрачиваемое на ожидание ресурсов, — один из основных показателей ухудшения производительности.

Самый простой с точки зрения учета ресурс — использование центрального процессора. В распоряжении системы всегда имеется примерно одна и та же часть его мощности. Теоретически эта величина составляет все 100% циклов центрального процессора, но "накладные расходы" и другие причины приводят к снижению этого показателя примерно до 95%. Для процесса, занимающего более 90% времени центрального процессора, ограничивающим фактором является его быстродействие. Такой процесс потребляет большую часть вычислительных мощностей системы.

Многие считают, что быстродействие центрального процессора — основной фактор, влияющий на общую производительность системы. При наличии неограниченных объемов всех остальных ресурсов или для определенных типов приложений (например, программ численного моделирования) быстродействие центрального процессора (или количество ядер) *действительно* играет роль. Но в повседневной жизни этот показатель значит относительно мало.

Настоящим узким местом является канал обмена данными с жестким диском. Жесткие диски представляют собой механические системы, поэтому на поиск нужного блока, выборку его содержимого и активизацию ожидающего его процесса уходят десятки миллисекунд. Задержки такого порядка отодвигают на второй план все остальные факторы ухудшения производительности. Каждое обращение к диску вызывает приостановку активного процесса "стоимостью" несколько миллионов инструкций централь-

ного процессора. Эту проблему можно решить, например, с помощью твердотельных дисковых накопителей.

Благодаря использованию виртуальной памяти скорость дискового обмена может быть непосредственно связана с объемом имеющейся памяти, если потребность в физической памяти превышает ее реальный объем. Ситуации, в которых физической памяти становится недостаточно, часто приводят к необходимости записывать содержимое ее страниц на диск, чтобы эти страницы можно было освободить и использовать для другой цели. Это означает, что обращение к памяти по своей скорости приближается к работе с диском. Избегайте таких ловушек, если характеристики производительности имеют для вас большое значение; побеспокойтесь о том, чтобы каждая система имела достаточно физической памяти.

Пропускная способность сети подвержена примерно тем же ограничениям, что и скорость дискового обмена. Ее ухудшение связано со всевозможными задержками. Но возникающие проблемы охватывают целые сообщества пользователей, а не отдельные компьютеры. Кроме того, сети восприимчивы к проблемам аппаратного обеспечения и перегрузкам серверов.

29.4. Захваченные циклы центрального процессора

Облачные технологии (и, как правило, виртуализация) обещают, что ваш сервер всегда будет иметь необходимые ему ресурсы. На самом деле большая часть этой щедрости — иллюзия. Даже в крупномасштабной виртуальной среде борьба за ресурсы может оказать заметное влияние на производительность виртуального сервера.

Центральный процессор — это наиболее часто используемый ресурс. Существует два способа, с помощью которых можно захватить циклы процессора на вашей виртуальной машине.

- Гипервизор, на котором работает ваша виртуальная машина, определяет квоту центрального процессора, основываясь на том, какую мощность вы заказали. Дефицит может быть устранен либо путем выделения большего количества ресурсов на уровне гипервизора, либо путем заказа более крупного экземпляра у поставщика облачных вычислений.
- Происходит превышение лимита, выделенного для физического оборудования, и для удовлетворения текущих потребностей всех экземпляров виртуальной машины физических циклов процессора оказывается недостаточно, хотя на все эти экземпляры могут распространяться квоты центрального процессора. В облачном провайдере исправление этой проблемы можно свести к перезапуску вашего экземпляра, чтобы он был переназначен на новое физическое оборудование. В вашем центре обработки данных решение может потребовать обновления среды виртуализации с большим количеством ресурсов.

Хотя захват процессора может произойти в любой операционной системе, работающей на виртуализованной платформе, система Linux позволяет выявить этот факт с помощью показателя **st** (stolen — захвачен) в командах **top**, **vmstat** и **mpstat**.

Вот пример использования команды **top**:

```
top - 18:36:42 up 3 days, 18:03, 1 user, load average: 3.40, 2.25, 2.08
Tasks: 218 total, 4 running, 217 sleeping, 0 stopped, 0 zombie
%Cpu: 41.6 us, 42.2 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.0 hi, 0.0 si, 16.2 st
```

В этом примере в 16,2% случаев система готова, но не может работать, потому что процессор отвлекается от виртуальной машины гипервизором. Время, потраченное на ожидание, непосредственно переводится в снижение пропускной способности. Внимательно следите за этим показателем на виртуальных серверах, чтобы гарантировать, что ваши рабочие нагрузки не будут непреднамеренно испытывать дефицит процессора.

29.5. Как анализировать проблемы производительности

В сложной системе нелегко выделить именно проблемы производительности. Системные администраторы часто получают невразумительные отчеты о проблемах, которые содержат эмоциональные описания конкретных случаев или сложных ситуаций (например, "веб-сервер застопорился из-за всех этих проклятых AJAX-вызовов..."). Вы, конечно, должны обратить внимание на эту информацию, но не воспринимайте ее как достоверную и надежную; проводите собственное исследование.

Строгая и прозрачная научная методика поможет вам сделать правильные выводы, которым ваши сотрудники и вы сами можете доверять. Научный подход позволит другим оценить ваши результаты, повысит доверие к вашей работе и увеличит вероятность того, что предлагаемые вами изменения смогут реально решить проблему.

Применение научного подхода не означает, что вы должны собирать все релевантные данные сами. Весьма полезной бывает и "внешняя" информация. Не стоит тратить часы на изучение вопросов, ответы на которые можно легко почерпнуть из разделов FAQ (Frequently Asked Questions — часто задаваемые вопросы).

Мы предлагаем вам выполнить следующие пять действий.

1. *Сформулируйте вопрос.* Поставьте конкретный вопрос в определенной функциональной области либо сформулируйте предварительное заключение или рекомендацию, которую вы обдумали. Будьте точны, говоря о технологиях, компонентах и альтернативах, которые вы предлагаете рассмотреть, и результатах, которые могут быть достигнуты.

2. *Соберите и классифицируйте факты.* Проведите систематический поиск в документации, базах знаний, известных вам изданиях, блогах, технических описаниях, на форумах и прочих информационных ресурсах с целью выявления внешних фактов, имеющих отношение к вашему вопросу. В собственных системах зафиксируйте телеметрические данные и (по возможности или необходимости) оснастите инструментальными средствами конкретные интересующие вас области в системе или отдельных приложениях.

3. *Критически оцените данные.* Просмотрите каждый источник данных на предмет релевантности и критически оцените его с точки зрения достоверности. Выделите ключевые данные и обратите внимание на качество источников информации.

4. *Подытожьте данные вербально и графически.* Представьте сведения, взятые из нескольких источников, в словесной (вербальной) и по возможности графической форме. Данные, кажущиеся сомнительными при выражении в числовой форме, могут оказаться убедительными в виде диаграммы.

5. *Сформулируйте заключение.* Представьте свои выводы в лаконичной форме (как ответ на поставленный вами же вопрос). Поставьте оценку, чтобы обозначить уровень силы или слабости доказательств, которые поддерживают ваши заключения.

29.6. Проверка производительности системы

Хватит общих советов — пора рассмотреть некоторые конкретные инструменты и "зоны интереса". Прежде чем переходить к измерениям, необходимо знать, что именно вы хотите получить.

Инвентаризуйте свое оборудование

Начните с оценки своего оборудования, особенно центрального процессора и ресурсов памяти. Инвентаризация поможет вам интерпретировать данные, представленные с помощью других инструментов, а также построить реалистичные ожидания касательно верхних границ производительности.



В системах Linux именно в файловой системе **/proc** стоит искать ответ на вопрос, какое с точки зрения вашей операционной системы вы используете оборудование (более детальную информацию об аппаратуре можно найти в каталоге **/sys**; см. раздел 11.3). Некоторые ключевые файлы перечислены в табл. 29.1. Общую информацию о файловой системе **/proc** можно найти в разделе 4.6.

Таблица 29.1. Источники информации об оборудовании в системах Linux

Файл	Содержимое
/proc/cpuinfo	Тип и описание центрального процессора
/proc/meminfo	Размер памяти и коэффициент загрузки
/proc/diskstats	Дисковые устройства и статистика их использования

Четыре строки в файле **/proc/cpuinfo** помогут вам идентифицировать центральный процессор в системе: `vendor_id`, `cpu family`, `model` и `model name`. Некоторые из этих значений непонятны, поэтому лучше всего узнать о них в интерактивной справочной системе.

Точная информация, содержащаяся в файле **/proc/cpuinfo**, зависит от системы и процессора. Рассмотрим типичный пример содержимого этого файла.

```
$ cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 15
model name      : Intel(R) Xeon(R) CPUE5310  @ 1.60GHz
stepping        : 11
cpu MHz         : 1600.003
cache size      : 4096 KB
physical id     : 0
cpu cores       : 2
siblings        : 2
...
```

Этот файл содержит по одной записи для каждого процессорного ядра, видимого операционной системой. Конкретные данные незначительно варьируются в зависимости от версии ядра. Значение поля `processor` уникально идентифицирует ядро. Значения поля `physical id` являются уникальными для каждого физического сокета на печат-

ной плате, а значения `core id` (не показано в приведенном выше примере) уникальны для ядра в физическом сокете. Ядра, которые поддерживают гиперпотоковый режим работы (дублирование контекстов ЦП без дублирования других характеристик обработки), идентифицируются значением `ht` в поле `flags` (также не показано в приведенном выше примере). Если в системе действительно реализован гиперпотоковый режим работы, поле `siblings` для каждого ядра показывает, сколько контекстов доступно на данном ядре.

Существует еще одна команда, которая позволяет получить информацию об аппаратных характеристиках вашего компьютера. Речь идет о команде **`dmidecode`**. Она выводит содержимое системной таблицы DMI (Desktop Management Interface — интерфейс управления настольными системами), также известной под именем SMBIOS. Самой полезной опцией этой команды является **`-t`** *тип* (допустимые типы представлены в табл. 29.2).

Таблица 29.2. Значения типов для команды `dmidecode -t`

Значение	Описание
1	Система
2	Материнская плата
3	Корпус
4	Процессор
7	Кеш-память
8	Разъемы портов
9	Системные разъемы
11	OEM-строки
12	Опции системной конфигурации
13	Язык BIOS
16	Массив физической памяти
17	Устройство памяти
19	Отображаемый адрес массива памяти
32	Загрузка системы
38	IPMI-устройство

В следующем примере демонстрируется получение типичной информации.

```
$ sudo dmidecode -t 4
# dmidecode 2.11
SMBIOS 2.2 present.

Handle 0x0004, DMI type 4, 32 bytes.
Processor Information
   Socket Designation: PGA 370
   Type: Central Processor
   Family: Celeron
   Manufacturer: GenuineIntel
   ID: 65 06 00 00 FF F9 83 01
   Signature: Type 0, Family 6, Model 6, Stepping 5
...
```

Биты информации о сетевой конфигурации разбросаны "по всей системе". Лучшим источником информации о IP- и MAC-адресам для каждого сконфигурированного интерфейса служат команды `ifconfig -a` (FreeBSD) и **`ip a`** (Linux).

Сбор данных о производительности

Программы анализа производительности в большинстве своем сообщают о том, что происходит в системе в данный момент времени. Но следует учесть, что структура и характер загрузки системы меняются в течение дня. Поэтому до принятия мер обязательно проведите всесторонний анализ данных, касающихся производительности. Истинная производительность выясняется только после длительного (месяц или даже больше) наблюдения за системой. Особенно важно собирать данные в периоды пиковой загруженности системы. Ограничения на использование ресурсов и ошибки в конфигурации системы часто выявляются только в таких условиях. Дополнительную информацию о сборе и анализе данных см. в главе 28.

Анализ использования центрального процессора

Обычно собирают три вида данных о центральном процессоре: общий показатель использования, показатели средней загруженности и потребление ресурсов отдельными процессами. Первая характеристика определяет, является ли быстродействие самого процессора узким местом. Показатели средней загруженности дают возможность оценить общую производительность системы. Данные, касающиеся отдельных процессов, позволяют выявить наиболее ресурсоемкие процессы.

Сводную информацию выдает команда **vmstat**. Ей передается два аргумента: время (в секундах), в течение которого необходимо наблюдать за системой для получения каждой строки выходной информации, и необходимое количество отчетов. Если не указать число отчетов, команда будет выполняться до тех пор, пока пользователь не нажмет комбинацию клавиш <Ctrl+C>. Рассмотрим пример.

```
$ vmstat 5 5
procs  -----------memory---------  -swap-  --io--  -system--  ----cpu----
 r b   swpd    free   buff  cache  si so    bi bo   in   cs  us sy id wa
 1 0    820 2606356 428776 487092   0  0  4741 65 1063 4857  25  1 73  0
 1 0    820 2570324 428812 510196   0  0  4613 11 1054 4732  25  1 74  0
 1 0    820 2539028 428852 535636   0  0  5099 13 1057 5219  90  1  9  0
 1 0    820 2472340 428920 581588   0  0  4536 10 1056 4686  87  3 10  0
 3 0    820 2440276 428960 605728   0  0  4818 21 1060 4943  20  3 77  0
```

Несмотря на то что содержимое этих столбцов может не совпадать в разных системах, статистические данные показателей использования центрального процессора практически не различаются среди платформ. Пользовательское время, системное время (время ядра), время простоя и время ожидания для каналов ввода-вывода (I/O) показаны в столбцах `us`, `sy`, `id` и `wa` соответственно. Большие числа в столбце `us` обычно означают активные вычисления, а в столбце `sy` они свидетельствуют о том, что процессы осуществляют очень много системных вызовов или выполняют операции ввода-вывода.

Выработанное нами за многие годы эмпирическое правило для серверов общего назначения, справедливое для большинства систем, гласит следующее: система должна тратить примерно 50% рабочего времени на обслуживание пользовательских запросов и столько же на системные запросы. Общее время простоя не должно быть нулевым. Если сервер выделяется под одно-единственное приложение, интенсивно использующее центральный процессор, то большая часть времени должна тратиться на обработку пользовательских запросов.

В столбце `cs` показано число переключений контекста за данный период, т.е. сколько раз ядро переключало процессы. В столбце `in` отображается число прерываний, ге-

нерируемых аппаратными устройствами или компонентами ядра. Слишком большие значения в этих столбцах свидетельствуют о неправильно работающем аппаратном устройстве. Остальные столбцы важны для анализа использования памяти и жесткого диска, о чем будет рассказываться ниже в этой главе.

Долговременные измерения средних показателей, характеризующих работу центрального процессора, позволяют определить, достаточно ли его ресурсов для нормальной работы системы. Если процессор регулярно часть времени простаивает, значит, есть запас по тактовой частоте. В этом случае увеличение быстродействия процессора ненамного улучшит общую производительность системы, хотя может и ускорить выполнение отдельных операций.

Как видно из показанного примера, центральный процессор постоянно переключается из режима интенсивного использования в режим полного бездействия и обратно. Поэтому необходимо наблюдать за показателями, соответствующими обоим режимам, и выводить среднее за определенный период времени. Чем меньше интервал наблюдения, тем ниже достоверность оценок.

В мультипроцессорных системах команды вроде **vmstat** сообщают средние значения по всем процессорам. В системе Linux существует команда **mpstat**, которая выдает отчет по каждому процессору. С помощью флага **-P** можно выбрать один конкретный процессор. Этой командой удобно пользоваться при отладке программ, поддерживающих симметричную многопроцессорную обработку. Интересно также узнать, насколько система эффективно (или неэффективно) работает с несколькими процессорами. Рассмотрим пример отображения состояния каждого из четырех процессоров.

```
linux$ mpstat -P ALL
08:13:38 PM CPU %user %nice %sys %iowait %irq %soft %idle intr/s
08:13:38 PM   0  1.02  0.00 0.49    1.29 0.04  0.38 96.79 473.93
08:13:38 PM   1  0.28  0.00 0.22    0.71 0.00  0.01 98.76 232.86
08:13:38 PM   2  0.42  0.00 0.36    1.32 0.00  0.05 97.84 293.85
08:13:38 PM   3  0.38  0.00 0.30    0.94 0.01  0.05 98.32 295.02
```

Центральный процессор однопользовательской рабочей станции обычно простаивает большую часть времени. Когда запрашивается прокрутка содержимого окна или обновляется веб-страница, центральный процессор справляется с этой операцией за считанные доли секунды. В такой ситуации долгосрочный средний показатель использования центрального процессора практически лишен смысла.

Еще одним статистическим показателем, полезным для оценки интенсивности использования системы, является показатель "средняя загруженность", который представляет собой среднее количество выполняемых процессов. Он дает достаточное представление о том, сколько процессов требуют свою долю процессорного времени. Узнать его значение можно с помощью команды **uptime**.

```
$ uptime
11:10am  up 34 days, 18:42, 5 users, load average: 0.95, 0.38, 0.31
```

Эта команда выдает три значения, которые соответствуют средней загруженности за пять, десять и пятнадцать минут. Как правило, чем выше средняя загруженность, тем важнее общая производительность системы. Если выполняется всего лишь один процесс, то он обычно ограничен одним ресурсом (центральным процессором или пропускной способностью дискового канала ввода-вывода). Пиковый спрос на этот ресурс и становится определяющим фактором производительности.

Когда в системе одновременно работает несколько процессов, нагрузка распределяется более равномерно. Если все работающие процессы потребляют ресурсы централь-

ного процессора, диска и памяти, то зависимость производительности системы от ограниченных возможностей какого-то одного ресурса снижается. В такой ситуации гораздо важнее следить за средними показателями загруженности, в частности за общим временем использования центрального процессора.

🕮 Дополнительную информацию о приоритетах см. в разделе 4.2.

Показатель средней загруженности отлично подходит для повседневного контроля системы. Если известен показатель работающей системы и он находится в диапазоне, характерном для перегрузки, значит, следует искать внешний источник проблемы (это может быть, к примеру, сеть). В противном случае нужно проверить процессы самой системы.

Еще один способ контроля над использованием центрального процессора — посмотреть, какую часть его времени отнимает каждый процесс с помощью команды `ps -aux`. Зачастую в интенсивно эксплуатируемой системе минимум 70% времени работы центрального процессора отнимается одним-двумя процессами. Запуск таких процессов в другое время суток или снижение их приоритетов позволит высвободить ресурсы ЦП для выполнения других процессов.

🕮 Подробнее об утилите `top` рассказывалось в разделе 4.4.

Прекрасной альтернативой команде `ps` является утилита `top`. Она выдает примерно ту же информацию, что и `ps`, но в "живом", постоянно обновляемом формате, позволяющем наблюдать за изменением состояния системы во времени.[1]

Управление памятью в системе

В UNIX- и Linux-системах память организована в виде блоков, называемых *страницами*, размер которых составляет 4 КиБ или больше. Когда процессы запрашивают у операционной системы память, ядро выделяет им виртуальные страницы. Каждой такой странице соответствует настоящее физическое хранилище, такое как оперативное запоминающее устройство или "резервное запоминающее устройство" на жестком диске. (Резервное запоминающее устройство обычно представляет собой просто пространство в области подкачки, но для страниц, которые содержат код исполняемой программы, резервное запоминающее устройство — это самый настоящий исполняемый файл. Аналогично резервное запоминающее устройство для некоторых файлов данных может представлять собой сами файлы.) Информация о связях между виртуальными и реальными страницами хранится в таблице страниц.

Ядро способно эффективно обслуживать запросы процессов, выделяя для них столько памяти, сколько им нужно. Это реализуется за счет дополнения области подкачки к реальной оперативной памяти. Поскольку для работы процессов требуется, чтобы их виртуальные страницы находились в реальной памяти, ядру постоянно приходится перемещать страницы между оперативной памятью и областью подкачки. Такие операции называются *страничным обменом* (paging).[2]

Ядро старается управлять памятью так, чтобы страницы, к которым недавно обращались, хранились в физической памяти, а менее активные страницы выгружались на диск. Этот алгоритм известен под названием LRU (least recently used — замещение

[1] Утилита `top` требует довольно много системных ресурсов, поэтому пользуйтесь ею в разумных пределах.

[2] Когда-то имела место иная операция, именуемая *подкачкой* (или *свопингом*), посредством которой все страницы некоторого процесса одновременно переписывались на диск. Теперь же во всех случаях обязательно используется страничный обмен (paging).

наименее используемых элементов), поскольку те страницы, к которым долго никто не обращался, перемещаются на диск.

Впрочем, отслеживать все обращения к страницам было бы слишком накладно для ядра, поэтому оно использует страничный кеш-буфер, анализ содержимого которого и позволяет принять решение о том, какие именно страницы оставить в памяти. Точный алгоритм этого механизма зависит от конкретной системы, но везде действует примерно одинаковый принцип. Такой вариант гораздо проще в реализации, чем LRU-система, а результаты дает почти такие же.

В случае нехватки памяти ядро пытается определить, какие страницы из "неактивного" списка давно не использовались. Если при последнем обращении такие страницы модифицировались процессом, ядро считает их "грязными"; они обязательно должны быть выгружены на диск, прежде чем занимаемая ими память будет повторно использована. Все остальные страницы реальной памяти считаются "чистыми" и могут быть назначены другому процессу.

Когда выполняется обращение к странице из "неактивного" списка, ядро возвращает ссылку на нее в таблицу страниц, обнуляет ее "возраст" и переводит страницу из "неактивного" списка в "активный". Если при этом изменяются адреса страниц реальной памяти, то страницы, выгруженные на диск, должны быть сначала прочитаны с диска, прежде чем их можно будет активизировать. Когда процесс обращается к неактивной странице, находящейся в памяти, происходит *программный сбой* (это так называемая "мягкая ошибка", или "soft fault"). В случае обращения к нерезидентной (выгруженной) странице имеет место *аппаратный сбой* (так называемая "жесткая ошибка", "hard fault"). Другими словами, при возникновении ошибки второго типа (в отличие от первого) требуется прочитать страницу с диска.

Ядро старается прогнозировать потребности приложений в памяти, поэтому операции выделения и выгрузки страниц не обязательно взаимосвязаны. Цель системы — обеспечить наличие достаточного объема свободной памяти, чтобы процессам не приходилось ждать выгрузки страниц всякий раз, когда им нужна свободная память. Если в периоды сильной загруженности системы страничный обмен резко возрастает, имеет смысл купить дополнительную память.



Можно изменить значение "коэффициента подкачки" (`/proc/sys/vm/swappiness`) и тем самым дать ядру подсказку о том, при каком проценте свободной памяти следует начинать активную выгрузку страниц. Если установить это значение равным нулю, то выгрузка страниц на диск начнется только когда закончится свободная память. При значении этого параметра 100, все страницы будут автоматически вытесняться на диск. По умолчанию для этого параметра устанавливается значение 60, т.е. выгрузка страниц начнется, когда процент использования основной памяти достигнет 40 (100–60=40). Если у вас возникает желание изменить этот параметр, значит, возможно, пришла пора увеличить объем оперативной памяти.

Если ядру не хватает как физической памяти, так и области подкачки, это означает, что виртуальная память исчерпана. В данной ситуации включается режим "принудительного уничтожения". Чтобы освободить память, системе приходится уничтожать целые процессы. И хотя выбираются наименее важные для системы процессы, все равно это очень неприятная процедура, ведь значительная часть ресурсов системы тратится не на полезную работу, а на управление памятью.

Анализ использования памяти

Интенсивность операций с памятью количественно представляется двумя показателями: общим объемом задействованной виртуальной памяти и текущим коэффициентом страничного обмена. Первый показатель характеризует общую потребность в памяти, а второй указывает на то, какая доля этой памяти активно используется. Задача администратора заключается в снижении интенсивности использования или увеличении объема памяти до тех пор, пока не будет достигнут приемлемый уровень страничного обмена. Страничный обмен — процесс неизбежный, поэтому не пытайтесь полностью избавиться от него.

Для того чтобы определить размер текущей области подкачки в системе Linux, выполните команду **swapon -s**.

```
linux$ swapon -s
Filename    Type        Size      Used  Priority
/dev/hdb1   partition   4096532   0     -1
/dev/hda2   partition   4096564   0     -2
```

При выполнении команды **swapon** значения выводятся в килобайтах. Значения размеров, выводимые этими командами, не включают содержимое памяти ядра, поэтому вам придется вычислить общий объем виртуальной памяти самостоятельно.

```
VM = объем оперативной памяти + используемый объем области подкачки
```

В системе FreeBSD вывод статистических показателей страничного обмена, полученных с помощью команды **vmstat**, выглядит так.

```
freebsd$ vmstat 5 5
procs  memory       page                    disk     faults
r b w  avm    fre   flt   re  pi  po  fr  sr da0 cd0   in     sy
0 0 0 412M  1.8G    97    0   1   0 200   6   0   0   51    359
2 0 0 412M  1.8G     1    0   0   0  0 7   0   0   0    5     27
2 0 0 412M  1.8G     0    0   0   0  0 7   0   0   0    4     25
1 0 0 412M  1.8G     0    0   0   0  0 6   0   0   0    4     25
0 0 0 412M  1.8G     0    0   0   0  0 7   0   0   0    6     26
```

Из этого примера удалена информация об использовании центрального процессора. Под заголовком `procs` показано количество процессов, готовых к немедленному выполнению, заблокированных в ожидании завершения операции ввода-вывода, а также готовых к выполнению, но вытесненных на диск. Если значение в столбце `w` когда-нибудь станет отличным от нуля, это будет означать, что объем системной памяти не соответствует текущей загрузке.

Столбцы, объединенные под общим заголовком `memory`, содержат данные об активной виртуальной памяти (`avm`) и свободной виртуальной памяти (`fre`). Столбцы, объединенные под общим заголовком `page`, содержат данные об страничном обмене. Во всех этих столбцах представлены средние значения: количество в секунду (табл. 29.3).

Таблица 29.3. Описание статистических показателей, выводимых командой `vmstat`

Столбец	Описание показателя
`flt`	Общее количество страничных сбоев
`re`	Количество восстановленных страниц, т.е. возвращенных из списка неиспользуемых
`pi`	Объем подкачанных страниц в килобайтах
`po`	Объем выгруженных страниц в килобайтах
`fr`	Объем списка неиспользуемых страниц в килобайтах
`sr`	Количество страниц, обработанных по алгоритму часов

В системах Linux статистические показатели, получаемые с помощью команды `vmstat`, могут иметь приведенный ниже вид.

```
linux$ vmstat 5 5
procs -------memory--------- -swap- ---io--  -system-  -----cpu------
 r b  swpd  free  buff  cache  si so   bi bo   in  cs  us sy id wa st
 5 0     0 66488 40328 597972   0  0  252 45 1042 278   3  4 93  1  0
 0 0     0 66364 40336 597972   0  0    0 37 1009 264   0  1 98  0  0
 0 0     0 66364 40344 597972   0  0    0  5 1011 252   1  1 98  0  0
 0 0     0 66364 40352 597972   0  0    0  3 1020 311   1  1 98  0  0
 0 0     0 66364 40360 597972   0  0    0 21 1067 507   1  3 96  0  0
```

В системе FreeBSD количество процессов, готовых к немедленному выполнению и заблокированных в ожидании ввода-вывода, выводится под заголовком `procs`. Статистические показатели страничного обмена ограничены двумя столбцами: `si` и `so`, которые представляют количество подкачанных и выгруженных страниц соответственно.

Кажущиеся несоответствия между столбцами большей частью иллюзорны. В одних столбцах указывается число страниц, в других — объем в килобайтах. Все значения являются округленными средними величинами. Более того, одни из них — средние скалярных величин, а другие — средние изменений.

С помощью полей `si` и `so` можно оценить интенсивность страничного обмена в системе. Операция загрузки (поле `si`) не обязательно означает, что из области подкачки восстанавливается страница. Это может быть исполняемый код, постранично загружаемый из файловой системы, или копия страницы, создаваемая в режиме дублирования при записи. Оба случая вполне типичны и не обязательно означают нехватку памяти. С другой стороны, операция выгрузки (поле `so`) всегда означает принудительное вытеснение страниц с данными ядром.

Система, в которой непрерывно происходят операции выгрузки, скорее всего, выиграет от добавления памяти. Если же это случается лишь время от времени и не вызывает жалоб со стороны пользователей, то можно не беспокоиться. В остальных случаях дальнейший анализ будет зависеть от того, что нужно сделать — оптимизировать систему для интерактивного режима (например, как рабочую станцию) или сконфигурировать ее для одновременной работы пользователей (например, как сервер приложений).

Анализ операций обмена с диском

Производительность жесткого диска можно контролировать с помощью команды **`iostat`**. Как и **`vmstat`**, эта команда поддерживает дополнительные аргументы, задающие интервал времени в секундах между моментами выдачи статистических данных за истекший период и число повторений. Команда **`iostat`** выдает также сведения об использовании центрального процессора. Приведем небольшой пример для системы Linux.

```
linux$ iostat
...
Device:  tps   kB_read/s   kB_wrtn/s    kB_read   kB_wrtn
sda     0.41        8.20        1.39     810865    137476
dm-0    0.39        7.87        1.27     778168    125776
dm-1    0.02        0.03        0.04       3220      3964
dm-2    0.01        0.23        0.06      22828      5652
```

Информация о каждом жестком диске размещается в столбцах `tps`, `kB_read/s`, `kB_wrtn/s`, `kB_read` и `kB_wrtn`: объем пересылок за секунду, количество килобайтов,

считываемых за секунду, число килобайтов, записываемых за секунду, общее количество считанных килобайтов и общее число записанных килобайтов.

Время, затрачиваемое на поиск блока, — основной фактор, влияющий на производительность жесткого диска. В первом приближении скорость вращения диска и быстродействие шины, к которой он подключен, не имеют особого значения. Современные диски могут пересылать сотни мегабайтов данных в секунду, если эти данные считываются из смежных секторов. Вместе с тем количество операций поиска составляет от 100 до 300 в секунду. Если после каждой операции поиска будет читаться один-единственный сектор, легко подсчитать, что в таком режиме задействуется менее 5% максимальной пропускной способности канала обмена данными с диском. Твердотельные диски имеют большое преимущество перед своими механическими предшественниками, поскольку их производительность не связана со скоростью вращения пластин и перемещениями головок.

Независимо от того, какой вид дисков вы используете — SSD или механические, для достижения максимальной производительности необходимо помещать файловые системы, которые применяются одновременно, на разные диски. Хотя тип шины и драйверы устройств влияют на степень эффективности, большинство компьютеров могут работать с несколькими дисками параллельно, что значительно повышает пропускную способность дисковой подсистемы. Например, данные и журнальные файлы веб-сервера имеет смысл размещать на разных дисках.

Особенно важно распределить область подкачки между несколькими дисками, если это возможно, поскольку страничный обмен обычно замедляет работу системы в целом. Многие системы позволяют создавать как выделенные разделы подкачки, так и файлы подкачки, записываемые в обычную файловую систему.

🕮 Подробнее о командах **lsof** и **fuser** см. в разделе 5.2.

Команда **lsof**, которая выводит список открытых файлов, и команда **fuser**, которая перечисляет процессы, использующие файловую систему, могут оказаться весьма полезными для выявления проблем, связанных со скоростью выполнения операций обмена с диском. Эти команды отображают взаимодействия между процессами и файловыми системами и могут указать на непредусмотренные вами взаимосвязи. Например, если некоторое приложение записывает свой журнальный файл на то же устройство, которое используется для ведения журналов регистрации базы данных, то в результате этот диск может стать "узким местом" системы.

Утилита `fio`: анализ производительности дисковой подсистемы

Утилита **fio** (`github.com/axboe/fio`) доступна как для системы Linux, так и для системы FreeBSD. Используйте ее для проверки производительности подсистемы хранения данных. Это особенно полезно в больших средах, где развертываются общие ресурсы хранения (такие как сеть хранения данных). Если вы оказываетесь в ситуации, когда скорость работы системы хранения данных является проблемой, часто бывает полезно определить следующие количественные показатели.

- Пропускная способность при выполнении операций ввода-вывода в секунду (IOPS) (чтение, запись и смесь).
- Средняя задержка (чтение и запись).
- Максимальная задержка (максимальные значения задержек при чтении и записи).

Как часть дистрибутива **fio** в подкаталог **examples** включены файлы конфигурации (**.fio**) для типичных тестов. Рассмотрим пример простого теста чтения-записи.

```
$ fio read-write.fio
ReadWriteTest: (g=0): rw=rw, bs=4K-4K/4K-4K/4K-4K, eng=posixaio, depth=1
fio-2.18
Starting 1 thread
Jobs: 1 (f=1): [M] [100.0% done] [110.3MB/112.1MB/0KB /s]
    [22.1K/28.4K/0 iops] [eta 00m:00s]
  read : io=1024.7MB, bw=91326KB/s, iops=20601, runt= 9213msec
    slat (usec): min=0, max=73, avg= 2.30, stdev= 0.23
    clat (usec): min=0, max=2214, avg=20.30, stdev=101.20
    lat (usec): min=5, max=2116, avg=22.21, stdev=101.21
    clat percentiles (usec):
    | 1.00th=[ 4], 5.00th=[ 6], 10.00th=[ 7], 20.00th=[ 7],
    | 30.00th=[ 6], 40.00th=[ 7], 50.00th=[ 7], 60.00th=[ 7],
    | 70.00th=[ 8], 80.00th=[ 8], 90.00th=[ 8], 95.00th=[ 10],
    | 99.00th=[ 668], 99.50th=[ 1096], 99.90th=[ 1208], 99.95th=[ 1208],
    | 99.99th=[ 1256]
    ...
    READ: io=1024.7MB, aggrb=91326KB/s, minb=91326B/s, maxb=91326KB/s,
      mint=10671msec, maxt=10671msec
    WRITE: io=1023.4MB, aggrb=98202KB/s, minb=98202KB/s, maxb=98202KB/s,
      mint=10671msec, maxt=10671msec
```

Как и у многих показателей, связанных с производительностью, у этих параметров нет общепринятых правильных значений. Лучше всего установить контрольный показатель, внести коррективы и выполнить повторные измерения.

Команда sar: сбор статистических данных и генерирование отчетов по ним

Команда **sar** — предназначенный для мониторинга производительности инструмент, который "проверен временем" (эта команда появилась еще во времена AT&T UNIX) на системах UNIX и Linux и все еще остается актуальным, несмотря на использование устаревшего синтаксиса командной строки.

На первый взгляд может показаться, что команда **sar** отображает ту же информацию, что и команды **vmstat** и **iostat**. Однако имеется одно очень важное отличие: **sar** "умеет" предоставлять отчеты как по старым (накопленным), так и текущим данным.

🕮 Пакет Linux, который содержит команду **sar**, называется **sysstat**.

Если не указаны конкретные параметры, команда **sar** предоставляет отчет о том, как центральный процессор использовался в течение того или иного дня каждые 10 мин., начиная с полуночи. Получение такого набора накопленных данных делает возможным сценарий **sal**, который является частью пакета **sar** и требует указания интервала времени, через который он должен запускаться из демона **cron**. Все данные, которые собирает команда **sar**, она сохраняет в бинарном формате в каталоге **/var/log**.

```
linux$ sar
Linux 4.4.0-66-generic (nuerbull) 03/19/17 _x86_64_ (4 CPU)
19:10:01  CPU  %user  %nice  %system  %iowait  %steal   %idle
19:12:01  all   0.02   0.00     0.01     0.00    0.00    99.97
19:14:01  all   0.01   0.00     0.01     0.00    0.00    99.98
```

Помимо информации о центральном процессоре, команда **sar** также может предоставлять отчеты по показателям дисковой и сетевой активности. Команда **sar -d** отображает сводку данных об активности диска за сегодняшний день, **sar -n DEV** — статистические данные о сетевом интерфейсе, а **sar -A** — всю доступную информацию.

Команда **sar** имеет некоторые ограничения и потому хорошо подходит только для быстрого получения кратких накопленных сведений. Тем, кто решил всерьез заняться мониторингом производительности, лучше установить специальную платформу с возможностями сбора коллекций данных и представления их в виде графиков, такую как платформа Grafana.

🕮 Подробнее о платформе Grafana см. в разделе 28.3.

Выбор планировщика ввода-вывода в системах Linux



В системах Linux используется алгоритм планирования подсистемы ввода-вывода, который выступает в роли "арбитра" между соревнующимися за дисковый ввод-вывод процессами. Он оптимизирует порядок и время запросов для обеспечения наилучшей возможной производительности подсистемы ввода-вывода для данного приложения или ситуации.

В текущую версию ядра Linux встроены три разных алгоритма планирования.

- Comletely Fair Queuing. Этот алгоритм используется по умолчанию и обычно является наиболее подходящим вариантом для серверов общего назначения. Он старается равномерно предоставлять доступ к подсистеме ввода-вывода. (Во всяком случае этот алгоритм заслуживает награды за маркетинг: кто скажет "нет" совершенно справедливому планировщику?)
- Deadline. Этот алгоритм "старается" сводить к минимуму время задержки для каждого запроса. Он изменяет порядок запросов для повышения производительности.
- NOOP. Этот алгоритм реализует простую очередь типа FIFO (First In, First Out — первым пришел, первым обслужен). Он предполагает, что запросы к подсистеме ввода-вывода уже были (или еще будут) оптимизированы или переупорядочены драйвером или устройством (каковым вполне может быть какой-нибудь контроллер со встроенной логикой). Он может быть наиболее подходящим вариантом в некоторых SAN-средах и для SSD-устройств (так как у этих устройств нет переменной задержки считывания данных).

Просмотреть или настроить алгоритм для конкретного устройства можно с помощью файла **/sys/block/***диск***/queue/scheduler**. Активный планировщик указывается в квадратных скобках.

```
$ cat /sys/block/sda/queue/scheduler
noop deadline [cfq]
$ sudo sh -c "echo noop > /sys/block/sda/queue/scheduler"
$ cat /sys/block/sda/queue/scheduler
[noop] deadline cfq
```

Определив, какой из этих алгоритмов планирования является наиболее подходящим для данной среды (что может потребовать проверки всех планировщиков), возможно, вам удастся улучшить производительность подсистемы ввода-вывода.

🕮 Дополнительную информацию о загрузчике GRUB см. в разделе 2.3.

К сожалению, алгоритм планирования не сохраняется при перезагрузке системы, если он был установлен таким образом. Вы можете установить его для всех устройств во время загрузки с помощью переменной ядра **elevator**=*алгоритм*. Эта конфигурация обычно устанавливается в файле конфигурации загрузчика GRUB **grub.conf**.

Программа perf: универсальный профилировщик системы Linux



Ядро Linux версии 2.6 и выше включает интерфейс `perf_events`, который обеспечивает доступ на уровне пользователя к потоку событий, связанных с производительностью ядра. Команда **perf** — это мощный интегрированный системный профилировщик, который считывает и анализирует информацию из этого потока. Можно профилировать все компоненты системы: аппаратное обеспечение, модули ядра, само ядро, совместно используемые библиотеки и приложения.

Чтобы начать работу с командой **perf**, вам нужно будет установить полный набор пакетов **linux-tools**.

```
$ sudo apt-get install linux-tools-common linux-tools-generic
  linux-tools-`uname -r`
```

После того как вы установили программное обеспечение, ознакомьтесь с его руководством на сайте `goo.gl/f88mt`, в котором приведены примеры и сценарии использования. (Это глубокая ссылка на сайт `perf.wiki.kernel.org`.)

Самый легкий путь начать работу не погружаясь в чтение документации — попробовать выполнить команду **perf top**, которая выводит на экран данные об использовании центрального процессора в стиле команды **top**. Конечно, простой пример, приведенный ниже, дает только самое поверхностное представление о команде **perf**.

```
$ sudo perf top
Samples: 161K of event 'cpu-clock', Event count (approx.): 21695432426
Overhead Shared Object              Symbol
4.63%    [kernel]                   [k] 0x00007fff8183d3b5
2.15%    [kernel]                   [k] finish_task_switch
2.04%    [kernel]                   [k] entry_SYSCALL_64_after_swapgs
2.03%    [kernel]                   [k] str2hashbuf_signed
2.00%    [kernel]                   [k] half_md4_transform
1.44%    find                       [.] 0x0000000000016a01
1.41%    [kernel]                   [k] ext4_htree_store_dirent
1.21%    libc-2.23.so               [.] strlen
1.19%    [kernel]                   [k] __d_lookup_rcu
1.14%    find                       [.] 0x0000000000169f0
1.12%    [kernel]                   [k] copy_user_generic_unrolled
1.06%    [kernel]                   [k] kfree
1.06%    [kernel]                   [k] _raw_spin_lock
1.03%    find                       [.] 0x00000000000169fa
1.01%    find                       [.] 0x0000000000016a05
0.86%    find                       [.] fts_read
0.73%    [kernel]                   [k] __kmalloc
0.71%    [kernel]                   [k] ext4_readdir
0.69%    libc-2.23.so               [.] malloc
0.65%    libc-2.23.so               [.] fcntl
0.64%    [kernel]                   [k] __ext4_check_dir_entry
```

В столбце Overhead показан процент времени, в течение которого процессор выполнял соответствующую функцию. В столбце Shared Object указан компонент (например, ядро), совместно используемая библиотека или процесс, в котором находится эта функция, а в столбце Symbol — имя функции (в тех случаях, когда информация о символах не была удалена).

29.7. Помогите! Мой сервер тормозит!

В предыдущих разделах речь шла в основном о мероприятиях, которые позволяют добиться средней производительности системы. Для ее повышения требуется корректировать параметры конфигурации или модернизировать аппаратное обеспечение системы.

Однако даже правильно сконфигурированные системы иногда работают медленнее обычного. К счастью, нерегулярные проблемы обычно легко диагностируются. В большинстве случаев они создаются "ненасытным" процессом, который потребляет так много ресурсов процессора, жесткого диска или сетевой подсистемы, что остальные процессы буквально останавливаются. Иногда процесс намеренно захватывает ресурсы, чтобы замедлить работу системы или сети. Это называется атакой типа "отказ в обслуживании".

Первый шаг в диагностировании проблемы — запуск команд **ps auxww** или **top** для выявления явно неуправляемых процессов. Любой процесс, отнимающий более 50% времени центрального процессора, можно с большой долей вероятности считать ненормальным. Если столь непомерную долю ресурсов центрального процессора не получает ни один процесс, посмотрите, сколько процессов получают минимум по 10%. Если их больше двух-трех (не считая самой команды **ps**), средняя загруженность системы, скорее всего, будет очень высокой. Такая ситуация сама по себе является причиной низкой производительности. Проверьте среднюю загруженность системы с помощью команды **uptime**, а затем выполните команду **vmstat** или **top**, чтобы узнать, простаивает ли когда-нибудь процессор.

Если конкуренции за право использования центрального процессора не наблюдается, выполните команду **vmstat**, чтобы посмотреть, какова интенсивность страничного обмена. Следует обращать внимание на все операции обмена с диском: множество операций выгрузки могут означать соперничество за память, а наличие дискового трафика в отсутствие страничного обмена говорит о том, что какой-то процесс монополизировал диск, непрерывно читая и записывая файлы.

Нельзя непосредственно сопоставить дисковые операции с выполняющими их процессами, но с помощью команды **ps** можно сузить круг "подозреваемых". Любой процесс, работающий с диском, должен отнимать какую-то часть времени ЦП. Как правило, всегда можно сделать обоснованное предположение о том, какой конкретно из активных процессов захватил ресурсы.[3] Для проверки своей теории на практике выполните команду **kill -STOP**.

Предположим, процесс-виновник найден. Что с ним делать дальше? Как правило, ничего. Некоторые операции действительно требуют много ресурсов и неизбежно за-

[3]Ранее признаками этого служили большой размер области данных процесса, размещенной в виртуальной памяти, либо неестественно большой объем занимаемой процессом физической памяти, но появление совместно используемых библиотек сделало эти показатели не столь полезными. Команда **ps** не умеет отделять общесистемные совместно используемые библиотеки от адресного пространства отдельных процессов. Кажется, что многие процессы занимают десятки мегабайтов физической памяти, хотя на самом деле это не так.

медляют работу системы, но это вовсе не означает, что они незаконны. Однако с помощью команды **renice** можно изменить приоритет процесса, для которого ограничивающим фактором является быстродействие центрального процессора.

Иногда правильная настройка приложения может привести к значительному снижению потребления ресурсов. Этот эффект особенно заметен в сетевых серверных программах, например в веб-приложениях.



В системе Linux существует удобный инструмент для работы с процессами, которые создают значительную нагрузку на диск, — команда **ionice**. Эта команда устанавливает класс планирования ввода-вывода процесса; по крайней мере один из доступных классов должен поддерживать числовые приоритеты ввода-вывода, которые также могут быть установлены с помощью команды **ionice**. Наиболее полезным вариантом для запоминания является команда **ionice -c 3 -p** *pid*, позволяющая указанному процессу выполнять операции ввода-вывода только в том случае, если на это не претендуют другие процессы.

Ядро позволяет процессу ограничивать объем потребляемой им самим физической памяти при помощи системного вызова **setrlimit**.[4] Эта возможность также доступна в системной оболочке в виде встроенной команды **ulimit** (**limits** в системе FreeBSD). Например, команда

```
$ ulimit -m 32000000
```

ограничивает использование физической памяти для всех последующих пользовательских команд тридцатью двумя мегабайтами. Ее действие примерно эквивалентно действию команды **renice** в отношении процессов, ограничивающим фактором для которых является объем памяти.

Если неуправляемый процесс не является источником снижения производительности, нужно проанализировать две другие возможные причины. Первая — перегрузка сети. Многие программы настолько тесно связаны с сетью, что иногда трудно понять, где речь идет о производительности системы, а где — о производительности сети.

В некоторых случаях проблемы, связанные с перегрузкой сети, выявить сложно, потому что они возникают и исчезают очень быстро. Например, если на всех компьютерах сети каждый день в определенное время с помощью демона **cron** запускается какая-то сетевая программа, то в сети может возникать кратковременный, но серьезный затор. Все компьютеры "зависнут" секунд на пять, после чего ситуация нормализуется.

Вторая причина — задержки, связанные с работой серверов. UNIX- и Linux-системы постоянно обращаются к удаленным серверам NFS, Kerberos, DNS и т.п. Если сервер не работает или какая-то иная проблема привела к тому, что связь с ним стала ненадежной, это ощущают на себе все клиентские системы.

Предположим, что в интенсивно эксплуатируемой системе какой-то процесс каждые несколько секунд вызывает библиотечную функцию **gethostent**. Если сбой в работе DNS заставляет эту функцию тратить на свое выполнение две секунды, будет заметно общее снижение производительности системы. На удивление, большое число проблем с производительностью серверов связано с неправильной конфигурацией прямого и обратного поиска в DNS.

---

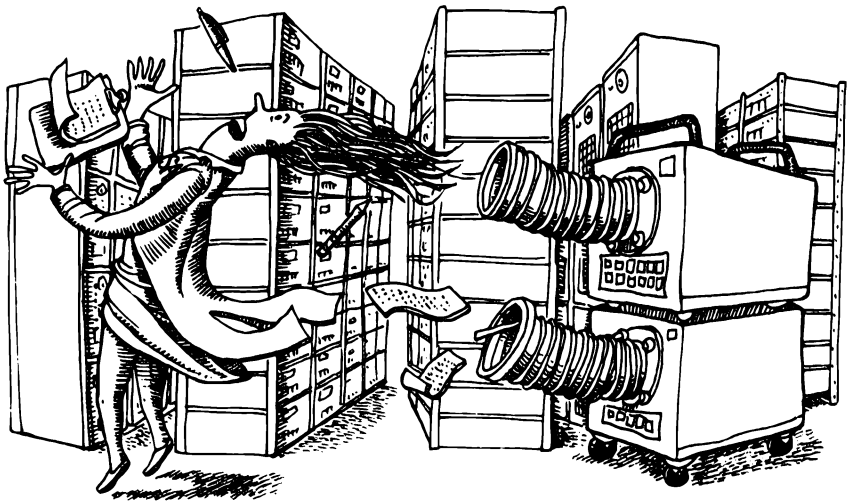
[4]Более тонкое управление ресурсами может быть достигнуто посредством CKRM-функций (Class-based Kernel Resource Management — управление ресурсами ядра на базе классов); см. `ckrm.sourceforge.net`.

29.8. Литература

- Drepper Ulrich. *What Every Programmer Should Know about Memory*. `lwn.net/Articles/250967`.
- Ezolt Phillip G. *Optimizing Linux Performance*. Upper Saddle River, NJ: Prentice Hall PTR, 2005.
- Gregg, Brendan. *Systems Performance: Enterprise and the Cloud*. Upper Saddle River, NJ: Prentice Hall PTR, 2013.
- Koziol, Prabhat, and Quincey Koziol. *High Performance Parallel I/O*. London: CRC Press, 2014.

Глава 30

Центры обработки данных



Надежность службы зависит от надежности работы центра обработки данных, который ее обеспечивает.[1] Для специалистов, имеющих практический опыт, это очевидно.

Сторонники облачных вычислений иногда, по-видимому, подразумевают, что облако может волшебным образом разбить цепи, соединяющие физический и виртуальный миры. Однако, хотя облачные провайдеры предлагают множество услуг, которые помогают повысить устойчивость и доступность, каждый облачный ресурс в конечном итоге живет где-то в физической реальности.

Понимание того, где на самом деле находятся ваши данные, является важной частью работы системного администратора. Если вы участвуете в выборе сторонних облачных провайдеров, оцените поставщиков и их возможности количественно. Вы также можете оказаться в положении, когда безопасность, суверенитет данных или политические проблемы диктуют необходимость создания и обслуживания собственного центра обработки данных.

Центр обработки данных состоит из следующих компонентов.

- Физически безопасное и защищенное место.
- Стойки с компьютерами, сетевым оборудованием и дисками.
- Система энергоснабжения (основная и резервная), достаточная для работы всех размещенных устройств.

[1] По крайней мере в первом приближении. Разумеется, можно распределить службу между несколькими центрами обработки данных, тем самым ограничив влияние сбоя в каком-либо из центров.

- Система охлаждения для поддержки требуемого температурного режима работы устройств.
- Сетевые соединения внутри центра и с внешним миром (корпоративная сеть, партнеры, поставщики оборудования, Интернет).
- Технический персонал, поддерживающий работу оборудования и инфраструктуры.

Некоторые аспекты центров обработки данных, такие как их физическое расположение, системы электропитания и охлаждения, традиционно разрабатывались и поддерживались техническим персоналом объекта. Тем не менее стремительные темпы развития IT-технологий и все более высокие требования к недопустимости простоев стимулируют тесное сотрудничество между сотрудниками IT-подразделений и техническим персоналом при планировании и эксплуатации центров обработки данных.

30.1. Стойки

Времена традиционных центров обработки данных с фальшполами, в которых кабели электропитания, система охлаждения, сетевые соединения и телефонные линии спрятаны под полом, прошли. Сможете ли вы обнаружить кабель, который теряется в подпольном лабиринте? Наш опыт подсказывает, что все это выглядит красиво только на картинке, а на самом деле комната с "классическим" фальшполом — это просто скрытая "крысиная нора". Сегодня фальшполы используются только для того, чтобы скрыть электрические кабели, распределить охлажденный воздух и *больше ни для чего.* Сетевые кабели (и медный, и оптоволоконный) должны проходить по верхним направляющим коробам.[2]

В специализированных центрах обработки данных размещение оборудования в стойках (а не, скажем, на столах или на полу) — единственный профессиональный выбор с точки зрения обслуживания. В лучших схемах расположения запоминающих устройств используются стойки, связанные друг с другом через верхние направляющие короба, по которым проходят кабели. Этот подход обеспечивает высокую технологичность без потери возможности для сопровождения.

Лучшие верхние направляющие короба для кабелей, насколько нам известно, производятся компанией Chatsworth Products (`chatsworth.com`). Использование стандартных 19-дюймовых монорельсовых телекоммуникационных стоек позволяет создавать блоки серверов, монтируемых на полках или в стойках. Две смежные 19-дюймовые телекоммуникационные стойки создают высокотехнологичную разновидность "традиционной" стойки для ситуаций, в которых вы вынуждены располагать в соседних стойках оборудование, ориентированное относительно передней и задней панелей. Компания Chatsworth поставляет стойки, короба для кабелей и всякие приспособления для их укладки и маркировки, а также оборудование, необходимое для их монтажа в вашем здании. Поскольку кабели располагаются в доступных коробах, их легко отслеживать и содержать в порядке.

30.2. Электропитание

Компьютерное оборудование требует стабильного и бесперебойного электропитания. Для этого центр обработки данных должен иметь следующие компоненты.

[2]Провода электропитания в настоящее время также проходят по наружным каналам.

- **Источники бесперебойного электропитания (UPS)** обеспечивают питание, когда обычный стационарный источник питания (например, коммерческая электросеть) становится недоступным. В зависимости от размера и мощности источники бесперебойного электропитания могут обеспечивать работу оборудования на протяжении от нескольких минут до нескольких часов. Они не могут самостоятельно поддерживать работу устройств в случае длительного отключения внешнего электропитания.
- **Автономные резервные электрогенераторы.** Если коммерческая сеть электропитания недоступна, автономные резервные электрогенераторы могут обеспечить долговременную работу оборудования. Генераторы обычно заправляются дизельным топливом, сжиженным или природным газом и могут поддерживать работу системы до тех пор, пока имеется топливо. Обычно принято хранить на месте запас топлива как минимум на 72 часа работы и организовать покупку топлива у нескольких поставщиков.
- **Резервные фазы электропитания.** В некоторых местах существует возможность подключить оборудование к нескольким фазам электропитания (возможно, от разных поставщиков).

🕮 Процедуры выключения и перезагрузки системы описаны в разделе 2.9.

Во всех случаях серверы и сетевая инфраструктура должны оснащаться источниками бесперебойного электропитания. Качественные устройства UPS имеют интерфейсы Ethernet или USB, позволяющие подключать их к компьютеру, который они обеспечивают электричеством, или к централизованной системе слежения, способной обеспечить быстрое реагирование. Такие соединения позволяют пересылать на компьютеры или операторам сообщения об отказе системы электропитания и требования выполнить аккуратное выключение компьютеров, пока не разрядились батареи.

Устройства UPS имеют разные размеры и мощности, но даже самое большое из них не может служить долговременным источником электропитания. Если ваше оборудование должно работать от автономного источника питания дольше, чем может выдержать UPS, вам нужно подключить к системе автономный резервный электрогенератор.

Существует широкий выбор резервных генераторов, мощность которых колеблется от 5 до 2500 кВт. Золотым стандартом является семейство генераторов компании Cummins Onan (`power.cummins.com`). Большинство организаций выбирает дизельный генератор. Если вы работаете в холодном климате, примите меры, чтобы генератор был заправлен "зимней соляркой", или замените ее авиационным топливом Jet A-1, чтобы предотвратить гелеобразование. Дизельное топливо является химически стабильным, но со временем в нем могут заводиться водоросли. Поэтому, если вы планируете хранить дизельное топливо долгое время, добавьте в него альгицид.

Генераторы и обслуживающая их инфраструктура — довольно дорогое удовольствие, но в некоторых ситуациях они могут сэкономить вам деньги. Если вы планируете установить резервный генератор, то ваше устройство UPS должно быть достаточно мощным, чтобы после выключения электропитания вы успели запустить резервный генератор.

Если вы используете устройства UPS или генераторы, чрезвычайно важно периодически их тестировать. Мы рекомендуем тестировать все компоненты резервной системы электропитания каждые полгода. Кроме того, вы (или производитель оборудования) должны выполнять регламентные работы на этом оборудовании как минимум раз в год.

Требования к электроснабжению стоек

Планирование электроснабжения центра обработки данных — одна из наиболее сложных задач. Как правило, возможность создать новый центр обработки данных или значительно перестроить существующий возникает каждые десять лет, поэтому, обдумывая систему электроснабжения, важно заглянуть далеко в будущее.

Большинство архитекторов для вычисления мощности, необходимой в перспективе для обеспечения работы центра обработки данных, склонны умножать его площадь в кв. футах на некий загадочный поправочный коэффициент. В реальных ситуациях этот подход не эффективен, потому что сам по себе размер центра обработки данных несет мало информации о типе оборудования, которое в нем будет работать. Мы рекомендуем использовать модель энергопотребления в расчете на стойку и не обращать внимания на площадь пола.

Традиционно центры обработки данных проектировались так, чтобы на каждую стойку приходилось от 1,5 до 3 кВт мощности. В настоящее время производители стали упаковывать серверы в стоечные корпуса типа 1U и создавать шасси лезвийных серверов, в которых размещаются более 20 лезвий, поэтому мощность, необходимая для питания всей стойки современного электронного оборудования, резко возросла.

Одно из возможных решений проблемы, связанной с плотностью упаковки оборудования и ростом при этом потребляемой мощности, — размещать в каждой стойке только необходимое количество серверов типа 1U, оставляя остальную часть стойки пустой. Несмотря на то что эта технология устраняет необходимость обеспечения дополнительной мощности для стойки, она приводит к чрезмерному растрачиванию места. Лучше разработать реалистичный проект обеспечения мощности, которая может понадобиться для каждой стойки, и подумать, где ее взять.

Требования к мощности у разного оборудования различные, поэтому трудно предсказать, какая мощность понадобится в будущем. Целесообразно создать систему с разными уровнями потребления мощности, в которой на каждом уровне все стойки получают одинаковую мощность. Эта схема полезна не только для удовлетворения потребностей текущих моделей, но и для планирования будущего использования. Ряд факторов, которые следует учитывать при определении уровней потребляемой мощности, перечислены в табл. 30.1.

Таблица 30.1. Оценки уровня мощности для стоек в центре обработки данных

Уровень мощности	кВт/стойка
Невероятно высокая плотность или специальные требования	40
Сверхвысокая плотность	25
Очень высокая плотность (например, лезвийные серверы)	20
Высокая плотность (например, серверы 1U)	16
Системы хранения данных	12
Сетевые коммутаторы	8
Обычная плотность	6

Определив уровни мощности, оцените потребности в стойках для каждого уровня. Затем на плане помещения поместите рядом друг с другом стойки, относящиеся к одному и тому же уровню. Такое разделение на зоны концентрирует стойки с высоким потреблением мощности и позволяет правильно распределить ресурсы охлаждения.

Киловольт-ампер или киловатт

Одна из основных причин непонимания между специалистами по информационным технологиям, техниками и инженерами-электриками заключается в том, что в каждой из этих групп используются разные единицы измерения мощности. Выходная мощность, которую может обеспечить устройство UPS, обычно измеряется в киловольт-амперах (кВА). Однако инженеры, обслуживающие компьютерное оборудование, и электрики, обеспечивающие работу центра обработки данных, обычно выражают мощность в ваттах (Вт) или киловаттах (кВт). Возможно, вы помните из курса физики, что ватт = = вольт × ампер. К сожалению, ваш школьный учитель забыл напомнить, что ватт — это векторная величина, в формулу которой для вычисления мощности переменного тока, кроме напряжения (вольт) и силы тока (апмер), входит еще так называемый "коэффициент мощности" (power factor).

Если вы разрабатываете конвейерную линию по разливу пива на пивоваренном заводе, в которой используется много двигателей переменного тока и другого тяжелого оборудования, то пропустите этот раздел и наймите квалифицированного инженера, который вычислит коэффициент мощности для вашей установки. При работе с современным компьютерным оборудованием вы можете схитрить и использовать константу. Для "приблизительного" преобразования кВА в кВт можно использовать следующие формулы.

$$кВА = кВт / 0{,}85$$

И наконец, оценивая суммарные мощности, необходимые для центра обработки данных (или для вычисления выходной мощности устройства UPS), следует измерить реальную потребляемую мощность, а не полагаться на значения, указанные производителем оборудования на этикетке устройства, на которой обычно указываются максимальные значения потребления.

🕮 Дополнительные советы по измерению потребляемой мощности см. в разделе 30.3.

Энергоэффективность

Энергоэффективность стала популярным показателем для оценки эксплуатации центров обработки данных. Промышленность стандартизировала простой коэффициент, известный как эффективность использования энергии (power usage effectiveness — PUE), выражающий общую эффективность центра.

$$PUE = \text{Общая мощность, потребляемая предприятием} / \text{Общая мощность, потребляемая IT-оборудованием}$$

Гипотетически идеальный центр обработки данных должен иметь показатель PUE, равный 1,0; иначе говоря, он будет потреблять ровно столько энергии, сколько необходимо для работы IT-оборудования, без накладных расходов. Конечно, эта цель недостижима на практике. Оборудование генерирует тепло, которое необходимо отводить, сотрудникам требуется освещение и другие условия для комфортной работы и т.д. Чем выше значение PUE, тем меньше энергоэффективность (и выше стоимость) центров обработки данных.

Современные центры обработки данных, которые являются достаточно энергоэффективными, обычно имеют коэффициент PUE, равный 1,4 или менее. Для справки, центры обработки данных еще десять лет назад обычно имели коэффициенты PUE в диапазоне 2,0–3,0. Компания Google, которая сделала акцент на энергоэффективность, регулярно публикует свои коэффициенты PUE, а с 2016 года она достигла в своих центрах обработки данных среднего значения PUE, равного 1,12.

Измерение

Вы можете судить только о том, что имеет количественный показатель, полученный в результате измерения. Если вы серьезно относитесь к энергоэффективности, важно понять, какие устройства потребляют больше всего энергии. Хотя коэффициент PUE дает общее представление об объеме энергии, потребляемой сверх IT-ресурсов, он очень мало говорит о энергоэффективности реальных серверов. (Фактически замена серверов на более энергоэффективные модели увеличит коэффициент PUE, а не уменьшит его.)

Администратору центра обработки данных следует выбрать компоненты, которые используют минимальное количество энергии. Одной из очевидных технологий является измерение потребления энергии на уровне отсека, стойки и устройства. Выберите или создайте центры обработки данных, которые могут легко предоставить наиболее важные данные.

Стоимость

Когда-то стоимость электроэнергии была более или менее одинаковой для центров обработки данных в разных местах. В наши дни индустрия облачных вычислений (Amazon, Google, Microsoft и др.) заставляет проектировщиков центров обработки данных учитывать их потенциальную экономическую эффективность в любом уголке мира. Одной из успешных стратегий было размещение крупных центров обработки данных вблизи источников недорогой электроэнергии, таких как гидроэлектростанции.

Принимая решение о том, следует ли запускать собственный центр обработки данных, обязательно учитывайте стоимость электроэнергии в вашей местности. Скорее всего, у крупных компаний будет естественное преимущество в стоимости таких (и других) услуг. Широкая доступность высокоскоростных оптических каналов передачи данных и их невысокая стоимость уже не ставят во главу угла поиск ближайшего центра обработки данных рядом с вашим местом работы.

Удаленное управление

Вы можете оказаться в ситуации, в которой необходимо регулярно включать и выключать сервер из-за ошибок в работе ядра или оборудования. Кроме того, возможно, в вашем центре обработки данных установлены серверы, которые работают под управлением другой операционной системы, а не Linux. В этом случае также часто возникает необходимость их регулярного включения и выключения. В любом случае следует рассмотреть возможность инсталляции системы, позволяющей выполнять эти операции в удаленном режиме.

Разумным решением является выбор продукции компании American Power Conversion (APC). Продукция MasterSwitch похожа на линейку устройств бесперебойного питания, за исключением того, что ею можно управлять с веб-браузера с помощью встроенного порта Ethernet.

30.3. Охлаждение и окружающая среда

Как и люди, компьютеры лучше и дольше работают в благоприятной среде. Поддержка безопасной температуры — основное условие их благополучной работы.

Американская ассоциация инженеров по отоплению, охлаждению и кондиционированию воздуха (American Society of Heating, Refrigerating and Air-conditioning Engineers — ASHRAE) традиционно рекомендует поддерживать температуру воздуха в центрах обработки данных (измеряемую на воздухозаборниках серверов) от 20 до 25 °С. Поддерживая попытки организаций сократить потребление электроэнергии, ассоциация ASHRAE выпустила в 2012 году новые рекомендации, согласно которым температуру воздуха следует поддерживать в диапазоне от 18 до 27 °С. Несмотря на то что этот диапазон кажется необычайно широким, он предполагает, что современное оборудование способно работать в широком диапазоне температур.

Оценка нагрузки на систему охлаждения

Поддержка требуемой температуры в помещении начинается с точной оценки нагрузки на систему охлаждения. Традиционные модели охлаждения в центрах обработки данных, взятые из учебников (даже из книг, написанных в 1990-х годах), могут на порядки отличаться от сегодняшних реалий, связанных с функционированием высокоупакованных шасси, содержащих лезвийные сервера. По этой причине мы рекомендуем перепроверять оценки нагрузки на систему охлаждения, полученные вашими инженерами по отоплению, охлаждению и кондиционированию воздуха (HVAC).

Вы должны проверить тепловую нагрузку, которую оказывают следующие компоненты.

- Крыша, стены и окна (от инженеров HVAC).
- Электронное оборудование.
- Осветительная арматура.
- Операторы (люди).

Из этих факторов только первый относится к компетенции инженеров HVAC. Конечно, они могут оценить и остальные факторы, но вы обязаны провести собственные вычисления. Оцените расхождения между вашими оценками и оценками, полученными инженерами HVAC, и дайте исчерпывающее объяснение.

Крыша, стены и окна

Крыша, стены и окна (не забывайте о солнечной нагрузке) вносят дополнительную нагрузку на систему охлаждения вашего помещения. Инженеры HVAC обычно имеют большой опыт работы с этими элементами и должны быть в состоянии дать вам хорошие оценки.

Электронное оборудование

Для того чтобы определить тепловую нагрузку, которую оказывают ваши серверы (и другое электронное оборудование), следует определить потребляемую серверами мощность. С практической точки зрения вся входящая электрическая энергия в конце концов превращается в тепловую.

Как и при планировании мощностей по электропитанию, непосредственное измерение потребляемой мощности, безусловно, является лучшим способом получить эту информацию. Вам может помочь дружелюбный сосед-электрик или же вы можете купить недорогой прибор и сделать это сами. Самым популярным устройством является Kill A Watt от компании P3 по цене около 20 долл., но он ограничен небольшими нагрузками (15 ампер), которые подключаются к стандартной розетке. Для больших нагрузок или нестандартных разъемов используйте такие амперметры, как Fluke 902 (также известный как "токовые клещи").

Большая часть оборудования характеризуется максимальной потребляемой мощностью, измеренной в ваттах. Однако типичное потребление, как правило, значительно меньше максимального.

Расход мощности можно преобразовать в стандартную единицу количества тепла BTUH (British Thermal Unit per hour — количество британских тепловых единиц в час), умножив его на коэффициент 3,412 BTUH/Вт. Например, если вы хотите построить центр обработки данных, в котором будет 25 серверов, потребляющих по 450 Вт каждый, то вычисления будут выглядеть следующим образом.

25 серверов × 450 Вт/сервер × 3,412 BTUH/Вт = 38 385 BTUH.

Осветительная арматура

Как и в случае электронного оборудования, оценить тепловую нагрузку от осветительного оборудования можно по расходу мощности. Как правило, в офисах в качестве осветительной арматуры используются 40-ваттные люминесцентные лампы. Если в вашем новом центре обработки данных шесть таких светильников (по 4 лампы в каждом), то вычисления будут выглядеть следующим образом.

6 ламп × 160 Вт/светильник × 3,412 BTUH/Вт = 3276 BTUH.

Операторы

Иногда людям необходимо войти в центр обработки данных, чтобы выполнить какие-то действия. Предположим, что каждый входящий человек создает тепловую нагрузку величиной 300 BTUH. Допустим, что одновременно в центр обработки данных могут войти четыре человека.

4 человека × 300 BTUH/чел = 1200 BTUH.

Общая тепловая нагрузка

Вычислив тепловую нагрузку для каждого компонента, просуммируйте их и определите общую тепловую нагрузку. В нашем примере мы предполагали, что инженер HVAC оценил тепловую нагрузку от крыши, стен и окон равной 20000 BTUH.

20000 BTUH для крыши, стен и окон

38385 BTUH для серверов и другого электронного оборудования

3276 BTUH для осветительной арматуры

1200 BTUH для операторов

62861 BTUH всего

Производительность систем охлаждения обычно измеряется в тоннах охлаждения. Для того чтобы преобразовать единицы BTUH в тонны охлаждения, необходимо поделить результат на 12000 BTUH/т. Кроме того, следует допустить по крайней мере 50%-ный коэффициент ухудшения, чтобы учесть ошибки и будущий рост системы.

62681 BTUH × 1 т / (12000 BTUH) × 1,5 = 7,86 тонн охлаждения

Проверьте, насколько ваши оценки расходятся с оценками инженеров HVAC.

Теплые и холодные отсеки

Вы можете резко уменьшить трудности, связанные с охлаждением центра обработки данных, проанализировав схему его устройства. Самой эффективной стратегией является разделение центра на теплые и холодные отсеки.

Помещения, которые имеют фальшпол и охлаждаются традиционными CRAC (computer room air conditioner — кондиционеры воздуха в компьютерных комнатах), часто спроектированы так, что холодный воздух проходит под полом, поднимается через отверстия в перфорированном полу, охлаждает оборудование, а затем уже в нагретом состоянии поднимается к потолку и всасывается отводящими воздуховодами. Обычно стойки и перфорированные плитки пола распределяются по центру обработки данных "случайно", и в результате обеспечивается относительно равномерное распределение температуры. Вследствие этого среда становится комфортной для человека, но не оптимальной для компьютеров.

Лучше было бы чередовать теплые и холодные отсеки, разделяя их стойками. В холодных отсеках находятся перфорированные плитки пола, а в теплых — нет. Стойки следует расположить так, чтобы оборудование втягивало воздух из холодного отсека, а отдавало — в теплый. Таким образом, стороны выпуска воздуха двух смежных стоек должны располагаться бок о бок. Эта схема изображена на рис. 30.1.

Эта схема оптимизирует поток воздуха так, чтобы воздуховоды серверов всегда втягивали холодный, а не теплый воздух, вышедший из системы охлаждения соседнего сервера. При правильном воплощении стратегия чередующихся отсеков позволяет создать заметно холодные и заметно теплые отсеки. Эффективность системы охлаждения можно оценить с помощью инфракрасного термометра (пирометра), который является незаменимым инструментом современного системного администратора. Это устройство стоит около 100 долл. (например, Fluke 62) и постоянно измеряет температуру любого предмета, на который вы его нацелите на расстоянии до 2 метров. Только не доставайте его в баре!

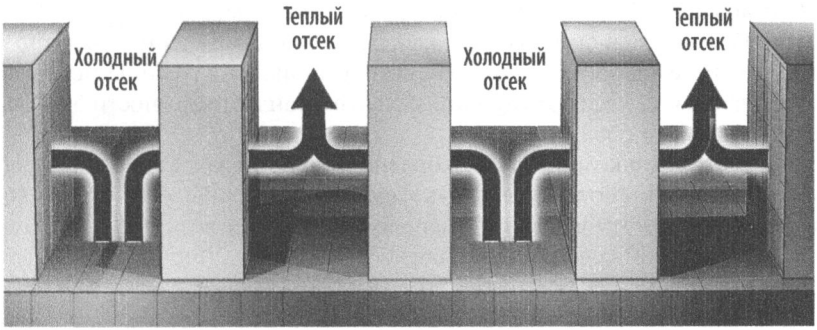


Рис. 30.1. Холодные и теплые отсеки с фальшполом

Если вам *нужно* развести кабеля под полом, прокладывайте кабели питания под холодными отсеками, а сетевые — под теплыми.

В помещениях без фальшпола могут использоваться междурядные охлаждающие устройства, например фирмы APC (`apc.com`). Эти небольшие плоские устройства помещаются между стойками. Принцип работы этой схемы показан на рис. 30.2.

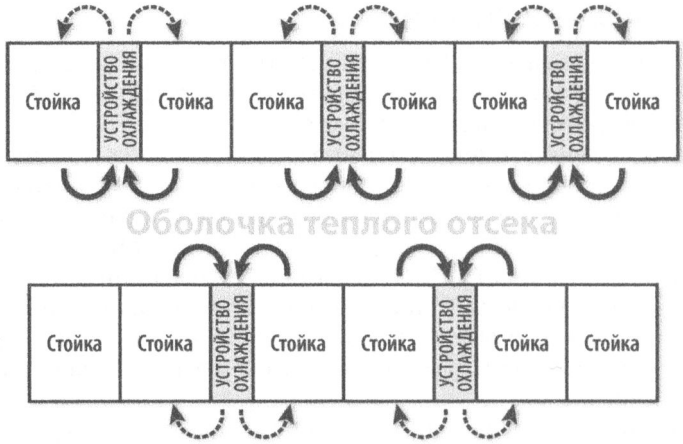


Рис. 30.2. Холодные и теплые отсеки с междурядными охлаждающими устройствами

Устройства CRAC и междурядные устройства охлаждения должны иметь возможность отводить тепло за пределы центра обработки данных. Это требование обычно удовлетворяется за счет циркуляции охлаждающей жидкости (например, охлажденной воды, хладагента Puron/R410A или R22), которая отводит тепло во внешнюю среду. Мы не показали на рис. 30.1 и 30.2 циркуляцию охлаждающей жидкости, чтобы не загромождать рисунки, но в большинстве проектов охлаждения они обязательно используются.

Влажность

В соответствии с рекомендациями 2012 ASHRAE влажность воздуха в центре обработки данных должна быть в пределах 8–60%. Если влажность воздуха слишком низкая, возникают проблемы из-за статического электричества. Проведенное недавно исследование показало, что существует небольшая эксплуатационная разница между 8% и предыдущим стандартом в 25%, поэтому минимальный стандарт влажности был скорректирован соответствующим образом.

Если влажность слишком высокая, конденсат воды может создать нежелательную проводимость на печатных платах, вызвать короткое замыкание и окисление контаков.

В зависимости от географического расположения может возникнуть необходимость увлажнения или осушения среды центра обработки данных, чтобы поддерживать требуемый уровень влажности.

Мониторинг окружающей среды

Если вы обеспечиваете работу чрезвычайно важного вычислительного оборудования, необходимо следить за температурой (и другими факторами окружающей среды, такими как шум и потребляемая мощность) в центре обработки данных, когда вы там отсутствуете. Вы будете очень огорчены, придя на работу в понедельник и обнаружив расплавленный пластик на полу центра обработки данных.

К счастью, существуют автоматические мониторы, которые могут следить за состоянием окружающей среды в ваше отсутствие. Мы сами используем и рекомендуем вам

продукцию семейства Sensaphone. Эти недорогие устройства для мониторинга следят за такими показателями, как температура, шум и потребляемая мощность, и звонят вам по телефону или отправляют сообщение на веб-сайт, если возникают проблемы.

30.4. Уровни надежности центров обработки данных

Исследованием вопросов, связанных с управлением центрами обработки данных, занимается промышленная группа Uptime Institute. Ее сотрудники разработали четырехуровневую систему классификации надежности центров обработки данных, которая представлена в табл. 30.2. В этой таблице обозначение *N* означает, что в распоряжении центра есть достаточно ресурсов (устройств UPS, генераторов), чтобы обеспечить нормальную работу. Обозначение *N*+1 означает, что у центра есть одна запасная единица оборудования; 2*N* — что у каждого устройства есть дублирующее оборудование.

Таблица 30.2. Система классификации, предложенная промышленной группой Uptime Institute

Уровень	Генератор	UPS	Источник электропитания	HVAC	Коэффициент доступности, %
1	Нет	*N*	Один	*N*	99,671
2	*N*	*N*+1 [a]	Один	*N*+1	99,741
3	*N*+1	*N*+1 [a]	Два (с возможностью переключения)	*N*+1	99,982
4	2*N*	2*N*	Два (работающих одновременно)	2*N*	99,995

[a] С запасными компонентами.

Центры, относящиеся к высшему уровню надежности, должны быть разделены на отсеки, чтобы сбой систем электропитания или охлаждения в одной из группы систем не приводил к отказу другой.

Коэффициент использования, равный 99,671%, на первый взгляд может выглядеть привлекательным, но это значит, что в течение года система будет находиться в нерабочем состоянии примерно 29 часов. Коэффициент использования, равный 99,995%, означает, что система может не работать 26 минут в год.

Разумеется, никакое количество резервных источников электропитания или устройств охлаждения не спасет систему, если она плохо управляется или неправильно спроектирована. Центр обработки данных — это основной структурный элемент, но его недостаточно для обеспечения работы всей организации с точки зрения конечного пользователя.

Стандарты работоспособности систем, разработанные группой Uptime Institute (включая сертификацию проектов, конструкций и этапов эксплуатации), можно найти на ее веб-сайте `uptimeinstitute.org`. В некоторых случаях организации используют концепцию этих уровней, не выплачивая значительных средств за сертификацию Uptime Institute. Важным является не сертификат в рамочке, а использование общей лексики и методологии оценки для сравнения центров данных.

30.5. Безопасность центров обработки данных

Возможно, это само собой разумеется, но физическая безопасность центра обработки данных по крайней мере так же важна, как и его экологические атрибуты. Удостоверьтесь,

что вы внимательно учли все возможные угрозы, как естественные (например, пожар, наводнение, землетрясение), так и искусственные (например, происки конкурентов и преступников). Многоуровневый подход к безопасности — лучший способ гарантировать, что одна ошибка или сбой не приведут к катастрофическому результату.

Местонахождение

По возможности центры обработки данных не должны располагаться в районах, подверженных лесным пожарам, торнадо, ураганам, землетрясениям или наводнениям. По аналогичным причинам рекомендуется избегать техногенных опасных зон, таких как аэропорты, автострады, нефтеперерабатывающие заводы и нефтебазы.

Поскольку центр обработки данных, который вы выбираете (или создаете), вероятно, будет вашим домом в течение длительного времени, стоит потратить некоторое время на изучение доступных данных о рисках при выборе места для его расположения. Геологическая служба США (`usgs.gov`) публикует статистические данные, такие как вероятность землетрясения, а организация Uptime Institute создает сложную карту рисков, связанных с выбором места для расположения центров обработки данных.

Периметр

Чтобы снизить риск целенаправленной атаки, центр обработки данных должен быть окружен ограждением, находящимся на расстоянии не менее 8 метров (25 футов) от здания со всех сторон. Доступ к внутренней части периметра ограждения должен контролироваться охранником или многофакторной системой пропусков (бейдж-системы доступа). Транспортные средства, которым разрешен въезд на территорию центра обработки данных, не должны находиться ближе 8 метров к зданию.

Непрерывный видеомониторинг должен охватывать 100% внешнего периметра, включая все ворота, подъездные пути доступа, автостоянки и крыши.

Здания не должны иметь вывесок. Никакая вывеска не должна указывать, кому принадлежит здание, или упоминать, что в нем находится центр обработки данных.

Доступ к объекту

Доступ к самому центру обработки данных должен контролироваться охранником и многофакторной системой пропусков, возможно, со снятием биометрических показателей. В идеальном случае уполномоченные стороны должны быть включены в систему контроля физического доступа до их первого посещения центра обработки данных. Если это невозможно, охранники на месте должны следить за процессом проверки, который включает подтверждение личности и санкционированных действий каждого сотрудника.

Одной из самых сложных ситуаций в обучении охранников является надлежащее обращение с посетителями, которые утверждают, что они пришли ремонтировать какую-то часть инфраструктуры, например кондиционер. Не делайте ошибку: если охранник не может подтвердить, что кто-то разрешил доступ или пригласил этого техника, такие посетители не должны получать пропуск.

Доступ к стойке

Большие центры обработки данных часто сдаются в аренду третьим сторонам. Этот подход экономически оправдан, но он несет дополнительную ответственность за организацию защиты каждой стойки (или отсека стоек). Это еще один случай, когда необходима многофакторная система контроля доступа (например, устройство чтения карт памяти и считыватель отпечатков пальцев), чтобы гарантировать, что только авторизованные стороны имеют доступ к вашему оборудованию. Каждая стойка также должна иметь индивидуальную систему видеонаблюдения.

30.6. Инструменты

Эффективный системный администратор — это хорошо оснащенный системный администратор. Иметь набор специальных инструментов важно для минимизации времени простоя в случае отказа оборудования. В табл. 30.3 перечислены некоторые инструменты, которые следует включать в такой набор.

Таблица 30.3. Инструментальный набор системного администратора

Универсальные инструменты	
Комплект шестигранных ключей (ключей Аллена)	Шариковый молоток, 150–200 грамм
Ножницы	Нож электрика или складной нож
Небольшой светодиодный фонарик	Крестообразная отвертка: №0, №1 и №2
Набор торцевых шестигранных ключей	Плоскогубцы и круглогубцы
Искатель скрытой проводки	Отвертка под шлиц: 1/8″, 3/16″ и 5/16″
Набор звездообразных ключей Torx	Миниатюрные часовые отвертки
Миниатюрный пинцет	
Специальные компьютерные инструменты	
Цифровой мультиметр (DMM)	Кабельная стяжка (и их аналоги на "липучках")
Инфракрасный термометр	Набор винтов для ремонта ПК
Щипцы RJ-45	Портативный сетевой анализатор/ноутбук
Терминаторы SCSI-устройств	Запасные коммутационные шнуры RJ-45 категорий 5 и 6A (как прямые, так и перекрещенные)
Запасной кабель питания	Запасные разъемы RJ-45 для обжима кабелей
Заземляющий браслет для статического электричества	Клещи для снятия изоляции (со встроенным инструментом для резки провода)
Разное	
Ватные палочки	Телескопический магнитный уловитель
Мобильный телефон	Комплект первой помощи, включающий ибупрофен и парацетамол
Изоляционная лента	Номера домашних и служебных телефонов сотрудников
Баллон со сжатым воздухом	Список контактов для срочной связи в случае опасной ситуации [a]
Зеркальце дантиста	Шесть банок пива (как минимум)

[a] С номерами контрактов на обслуживание.

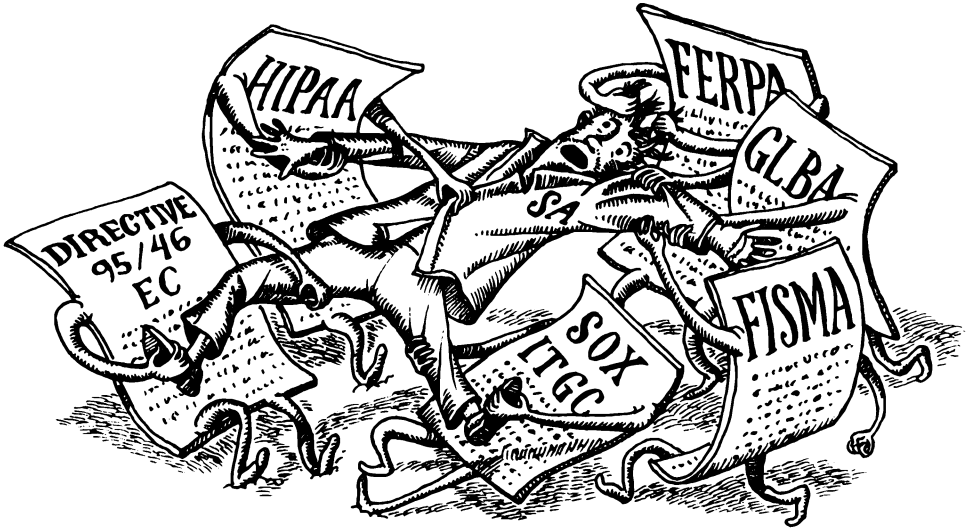
30.7. Литература

- ASHRAE Inc. *ASHRAE 2008 Environmental Guidelines for Datacom Equipment*. Atlanta, GA: ASHRAE, Inc., 2008.

  Разнообразную полезную информацию и стандарты, связанные с энергоэффективностью, можно найти на веб-сайте Центра экспертизы энергоэффективности в центрах обработки данных (Center of Expertise for Energy Efficiency in Data Centers) по адресу `datacenters.lbl.gov`.
- *Telecommunications Infrastructure Standard for Data Centers*. ANSI/TIA/EIA 942.

Глава 31

Методология, политика и стратегии



За последние четыре десятилетия роль информационных технологий в бизнесе и повседневной жизни резко изменилась. Трудно представить мир без мгновенного удовлетворения поиска в Интернете.

В течение большей части этого периода преобладающей философией IT-менеджмента было повышение стабильности за счет минимизации изменений. Во многих случаях сотни или тысячи пользователей зависели от одной системы. Если происходил сбой, оборудование часто должно было быть отправлено на ремонт, или требовалось время простоя для переустановки программного обеспечения и восстановления состояния. IT-команды жили в страхе, что что-то сломается и что они не смогут это исправить.

Стратегия минимизации изменений имеет нежелательные побочные эффекты. IT-отделы часто застревали в прошлом и не соответствовали потребностям бизнеса. Накапливался "технический долг" в виде систем и приложений, отчаянно нуждающихся в обновлении или замене, которых все боялись касаться, опасаясь что-то сломать. IT-сотрудники стали предметом шуток и наименее популярными людьми повсеместно — от залов для заседаний советов директоров до праздничных вечеринок.

К счастью, эти времена позади. Появление облачной инфраструктуры, виртуализации, средств автоматизации и широкополосной связи значительно сократило потребность в одиночных системах. Такие серверы были заменены армиями клонов, которые управляются как батальоны. В свою очередь, эти технические факторы позволили разработать философию обслуживания, известную как DevOps, которая по-

зволяет IT-организациям управлять изменениями, а не сопротивляться им. Имя DevOps — это слово-гибрид, состоящее из слов development (разработка) и operations (эксплуатация), — названий двух традиционных дисциплин, которые оно объединяет.

IT-организация — это больше, чем группа технических специалистов, которые настраивают точки доступа к Wi-Fi и компьютеры. Со стратегической точки зрения IT-подразделение — это группа людей и ролей, которые используют технологии для ускорения работы и достижения целей компании. Никогда не забывайте о золотом правиле системного администрирования: предприятиям нужно управлять IT-деятельностью, а не наоборот.

В этой главе мы обсудим нетехнические аспекты работы успешной IT-организации, которая использует методологию DevOps в качестве своей всеобъемлющей схемы. Большинство тем и идей, представленных в этой главе, не являются специфическими для какой-либо конкретной среды. Они применяются в равной степени к системному администратору с частичной занятостью или к большой группе профессионалов, работающих полный рабочий день. Подобно свежим овощам, они полезны, независимо от того, насколько большой обед вы готовите.

31.1. Великая единая теория: DevOps

Системное администрирование и другие эксплуатационные роли в сфере IT традиционно отделяются от таких областей, как разработка приложений и управление проектами. Теория заключалась в том, что разработчики приложений были специалистами, которые продвигали продукты с новыми функциями и улучшениями. Тем временем стабильная и устойчивая к изменениям эксплуатационная группа обеспечивала непрерывное управление производственной средой. Такое соглашение обычно создает колоссальный внутренний конфликт и в конечном итоге не отвечает потребностям бизнеса и его клиентов.



Рис. 31.1. С любезного разрешения Дэйва Рота (Dave Roth)

Подход DevOps объединяет разработчиков (программистов, прикладных аналитиков, владельцев приложений, менеджеров проектов) с IT-персоналом по эксплуатации (системными и сетевыми администраторами, контролерами безопасности, персоналом дата-центра, администраторами баз данных). Эта философия базируется на убеждении, что совместная работа членов единой команды разрушает барьеры, уменьшает частоту выяснения отношений и дает лучшие результаты. На рис. 31.2 приведены некоторые из основных концепций DevOps.



Рис. 31.2. Что такое DevOps

DevOps — относительно новая концепция в области управления IT-организациями. В начале 2000-х гг. произошли изменения в области разработки, которая перешла от каскадной схемы циклов выпусков к гибким подходам, которые отличались итеративной разработкой. Эта схема увеличила скорость, с которой могли быть созданы продукты, функции и исправления, но развертывание этих улучшений часто останавливалось, поскольку эксплуатационная сторона не была готова двигаться так быстро, как сторона разработки. Объединение разработчиков и эксплуатационных групп позволило всем работать в одном и том же темпе, и появилась методология DevOps.

DevOps — это CLAMS

Принципы методологии DevOps наиболее легко описываются аббревиатурой CLAMS: Culture (Культура), Lean (Рациональность), Automation (Автоматизация), Measurement (Измерения) и Sharing (Совместная работа).

Культура

В конечном счете, основной движущей силой любой успешной команды являются люди, поэтому культурные аспекты DevOps являются самыми важными. Хотя методология DevOps имеет собственный набор культурных советов и приемов, главная задача — заставить всех работать вместе и сосредоточиться на общей цели.

В методологии DevOps все работают вместе, чтобы поддерживать общий бизнес-драйвер (продукт, цель, сообщество и т.д.) на всех этапах его жизненного цикла. Достижение этой цели может в конечном итоге потребовать изменений в структуре отчетности (больше нет изолированных групп по разработке приложений), штатного расписания и даже должностных обязанностей. В наши дни хорошие системные адми-

нистраторы иногда пишут код (часто сценарии автоматизации или развертывания), а хорошие разработчики приложений регулярно проверяют показатели производительности инфраструктуры и управляют ими.

Перечислим некоторые типичные особенности культуры DevOps.

- И разработчики (Dev), и сотрудники отдела эксплуатации (Ops) работают непрерывно (24/7), одновременно ("сообщения получают все") и несут ответственность за всю среду. Это правило обладает прекрасным побочным эффектом, который состоит в том, что причины неполадок могут быть устранены в любом месте, где бы они не возникали.[1]
- Никакое приложение или служба не могут запускаться без автоматического тестирования и мониторинга как на уровне системы, так и на уровне приложений. Это правило фиксирует функциональность и создает контракт между Dev и Ops. Аналогично Dev и Ops должны заранее одобрять все запуски.
- Все производственные среды зеркально отражаются в одинаковых средах разработки. Это правило создает основу для тестирования и уменьшает количество производственных сбоев.
- Команды Dev регулярно проводят обзоры кода, на которые приглашаются члены команды Ops. Архитектура и функциональность кода больше не являются функциями команды Dev. Аналогично команда Ops проводит регулярные обзоры инфраструктуры, в которых участвуют члены команды Dev. Они должны знать и влиять на решения, касающиеся базовой инфраструктуры.
- Команды Dev и Ops проводят регулярные совместные совещания. В целом встречи должны быть сведены к минимуму, но совместные собрания служат полезным средством коммуникации.
- Члены команд Dev и Ops должны находиться в общем чате, посвященном обсуждению как стратегических (архитектура, направление, размер), так и эксплуатационных вопросов. Этот канал связи часто называют ChatOps, и для его поддержки доступны несколько замечательных платформ, в частности HipChat, Slack, MatterMost и Zulip.

Успешная культура DevOps настолько сближает команды Dev и Ops, что их области смешиваются, и при этом каждый член команды старается чувствовать себя комфортно. Оптимальный уровень перекрытия, вероятно, превышает уровень, на котором работают люди, не разделяющие принципы DevOps. Члены команды должны учиться правильно реагировать на запросы и учитывать отзывы о своей работе от коллег, которые могут быть формально обучены другим дисциплинам.

Рациональность

Самый простой способ объяснить рациональность DevOps — отметить, что если вы планируете периодические еженедельные встречи, чтобы обсудить план внедрения DevOps, значит вы уже потерпели неудачу.

DevOps — это взаимодействие и общение в реальном времени между людьми, процессами и системами. Используйте инструменты реального времени (например, ChatOps) для связи, где это возможно, и сосредоточьтесь на решении проблем по оче-

[1] Примерно первые шесть недель такая модель работы вызывает болезненные ощущения. Затем они внезапно прекращаются, поверьте нам.

реди. Всегда спрашивайте, "что мы можем сделать сегодня", чтобы добиться прогресса по проблеме. Избегайте искушения объять необъятное.

Автоматизация

Автоматизация — это наиболее общепризнанный аспект DevOps. Два золотых правила автоматизации гласят:

- если вам нужно выполнить задачу более двух раз, ее следует автоматизировать;
- не автоматизируйте то, чего вы не понимаете.

Автоматизация приносит много преимуществ.

- Она препятствует тому, чтобы сотрудники завязли в рутине. Интеллектуальная мощь и творческий потенциал персонала могут быть использованы для решения новых и более сложных задач.
- Она снижает риск человеческих ошибок.
- Она выражает инфраструктуру в виде кода, позволяя отслеживать версии и результаты.
- Она способствует эволюции, а также снижает риск. Если изменения потерпели неудачу, легко (ну, теоретически) выполнить автоматический откат.
- Она облегчает использование виртуальных или облачных ресурсов для обеспечения масштабирования и избыточности. Нужно больше ресурсов? Добавьте немного. Нужно меньше ресурсов? Удалите лишнее.

Важную роль в автоматизации играют инструменты. Такие системы, как Ansible, Salt, Puppet и Chef (см. главу 23), — это передний край и центр. Инструменты непрерывной интеграции, такие как Jenkins и Bamboo (см. раздел 26.3), помогают управлять повторяемыми или запускаемыми задачами. Низкоуровневые задачи инфраструктуры автоматизируют утилиты для упаковки и выпуска, такие как Packer и Terraform.

В зависимости от используемой среды вам может понадобиться один из этих инструментов, несколько или даже все. Новые инструменты и усовершенствования разрабатываются быстро, поэтому сосредоточьтесь на поиске инструмента, подходящего для конкретной функции или процесса, который вы автоматизируете, и не пытайтесь сначала выбирать инструмент, а затем выяснять проблемы, которые он решает. Переоценивать набор инструментов необходимо каждый год или два.

Ваша стратегия автоматизации должна включать по крайней мере следующие элементы.

- **Автоматическая настройка новых машин:** это не просто установка операционной системы. Она также включает в себя все дополнительное программное обеспечение и локальную конфигурацию, необходимые для запуска машины. Ваша организация неизбежно будет поддерживать более одного типа конфигурации, поэтому с самого начала включите в свои планы несколько типов компьютеров.
- **Автоматическое управление конфигурацией:** изменения конфигурации должны вводиться в базу конфигурации и автоматически применяться ко всем машинам одного и того же типа. Это правило помогает поддерживать целостность среды.
- **Автоматическое продвижение кода.** Распространение новых функций из среды разработки в тестовую среду и из тестовой среды в производственную должно быть автоматизировано. Само тестирование должно быть автоматизировано, с четкими критериями оценки и продвижения по службе.

- **Систематическое исправление и обновление существующих машин.** Когда станет ясна проблема с установкой используемой среды, вам нужен стандартизованный и простой способ развертывания обновлений для всех затронутых машин. Поскольку серверы могут быть не включены постоянно (даже если предполагается противоположное), при запуске обновления ваша схема внесения изменений должна правильно обрабатывать машины, которые находятся в выключенном состоянии.

Вы можете проверять обновления во время загрузки или по расписанию; дополнительную информацию см. в разделе 4.9.

Измерения

Возможность масштабирования виртуальной или облачной инфраструктуры (см. главу 9) подтолкнула мир приборов и измерений к новым высотам.

🕮 Дополнительную информацию о мониторинге см. в главе 28.

Сегодняшний золотой стандарт — это сбор субсекундных измерений во всех службах (бизнес, приложения, базы данных, подсистемы, серверы, сеть и т.д.). Эти усилия поддерживают некоторые инструменты DevOps, такие как Graphite, Grafana, ELK (стек Elasticsearch + + Logstash + Kibana), а также платформы мониторинга, такие как Icinga и Zenoss.

Однако иметь данные измерений и делать что-то полезное — это две разные вещи. Опытный отдел DevOps гарантирует, что метрики из окружающей среды будут доступными и признаваемыми всеми заинтересованными сторонами (как внутри, так и за пределами IT-организации). DevOps устанавливает номинальные цели для каждой метрики и преследует любые аномалии, чтобы определить их причину.

Совместная работа

В основе успеха методологии DevOps лежат совместная работа и совместное развитие возможностей. Персонал следует поощрять и стимулировать делиться своей работой как внутри организации (с помощью презентаций, лекций, материалов, показательных выступлений команд, учебных статей в вики-системе и т.д.), так и вне ее (используя семинары, публикации, конференции). Эти мероприятия усиливают взрывной эффект методологии за пределами локальной рабочей группы и помогают каждому учиться и расти.

Системное администрирование в мире DevOps

Системные администраторы всегда были главными и универсальными действующими лицами в мире IT, и это остается в силе под более широким зонтиком DevOps. Системный администратор контролирует системы и инфраструктуру, как правило, принимая на себя основную ответственность за следующие области.

- Создание, настройка, автоматизация и развертывание системной инфраструктуры.
- Обеспечение безопасности, исправления и обновления эксплуатационной системы и основных подсистем.
- Развертывание, поддержка и внедрение технологий DevOps для непрерывной интеграции, непрерывного развертывания, мониторинга, измерения, контейнеризации, виртуализации и инсталляции платформ ChatOps.

- Обучение других членов команды на основе передовой практики в области инфраструктуры и безопасности.
- Мониторинг и поддержка инфраструктуры (физической, виртуальной или облачной) для обеспечения соответствия требованиям производительности и доступности.
- Реагирование на пользовательские ресурсы или запросы на расширение.
- Устранение проблем с системами и инфраструктурой по мере их возникновения.
- Планирование будущего расширения систем, инфраструктуры и возможностей.
- Пропаганда кооперативных взаимодействий между членами команды.
- Управление различными внешними поставщиками (облако, совместное размещение, аварийное восстановление, сохранение данных, подключение, физическое оборудование, аппаратное обеспечение).
- Управление жизненным циклом компонентов инфраструктуры.
- Поддержание экстренного запаса ибупрофена, текилы и/или шоколада для совместного использования с другими членами команды в трудные времена.

Это всего лишь подмножество обязанностей, которые лежат на плечах успешного системного администратора. Он одновременно сержант-инструктор, опекун, спасатель и звено, которое обеспечивает бесперебойную работу.

Прежде всего, помните, что методология DevOps основана на преодолении нормальных рефлексов по защите своей территории. Если вы обнаружили, что воюете с другими членами команды, сделайте шаг назад и помните, что вы наиболее эффективны, если вас считают героем, который способствует общему успеху.

31.2. Системы управления билетами и задачами

В основе каждой действующей IT-группы лежит система управления билетами и задачами. Как и для всех элементов методологии DevOps, наличие одной системы билетов, которая охватывает все IT-дисциплины, имеет решающее значение. В частности, запросы на улучшение, управление выпуском и отслеживание ошибок программного обеспечения должны быть частью одной и той же системы.

Хорошая билетная система помогает сотрудникам избегать двух наиболее распространенных ловушек рабочего процесса:

- задач, которые ускользают от внимания, потому что все думают, что их кто-то решает;
- ресурсов, которые теряются в результате дублирования усилий, когда несколько человек или групп несогласованно работают над одной и той же проблемой.

Общие функции билетных систем

Билетная система принимает запросы через различные интерфейсы (наиболее распространенными являются электронная почта и Интернет) и отслеживают их от поступления до решения. Менеджеры могут назначать билеты группам или отдельным сотрудникам. Сотрудники могут запрашивать систему, чтобы увидеть очередь билетов, ожидающих выполнения, и, возможно, выполнить некоторые из них. Пользователи могут узнавать состояние своих запросов и выяснять, кто над ними работает. Менеджеры могут извлекать информацию высокого уровня, в частности:

- количество открытых билетов;
- среднее время закрытия билета;
- производительность сотрудников;
- процент невыполненных билетов;
- распределение рабочей нагрузки по времени, требуемому для выполнения.

История запросов, хранящаяся в билетной системе, становится историей проблем, связанных с вашей IT-инфраструктурой, а также решений этих проблем. Если эта история легко доступна для поиска, она станет неоценимым ресурсом для сотрудников системы.

Выполненные билеты могут быть предоставлены начинающим сотрудникам и стажерам, вставлены в систему часто задаваемых вопросов или включены в поисковую систему для последующего изучения. Новые сотрудники могут извлекать выгоду от получения копий выполненных билетов, поскольку эти билеты включают не только техническую информацию, но и примеры стиля общения с клиентами.

Как и все документы, исторические данные вашей билетной системы потенциально могут быть использованы против вашей организации в суде. Следуйте рекомендациям по сохранению документов, установленным юридическим отделом.

Большинство систем отслеживания запросов автоматически подтверждают новые запросы и присваивают им номер отслеживания, который участники могут использовать для отслеживания или выяснения состояния своего запроса. В автоматическом ответном сообщении должно быть четко указано, что это просто подтверждение. За ним должно незамедлительно следовать сообщение от реального человека, в котором объясняется план решения проблемы или выполнения запроса.

Владелец билета

Работа может быть совместной, но, по нашему опыту, ответственность меньше поддается распределению. У каждой задачи должен быть один, четко определенный владелец. Этот человек не должен быть руководителем или менеджером, а просто кем-то, желающим действовать в качестве координатора, — кем-то, кто хочет сказать: "Я беру на себя ответственность за то, чтобы эта задача была решена".

Важным побочным эффектом этого подхода является то, что благодаря ему становится ясным, кто и что реализовал или кто и какие изменения внес. Эта прозрачность очень важна, если вы хотите понять, почему что-то было сделано определенным образом или почему что-то неожиданно работает по-другому или больше не работает.

Если возникают проблемы, ответственного за задачу не следует приравнивать к козлу отпущения. Если ваша организация определяет ответственность как виновность, вы можете обнаружить, что количество доступных владельцев билетов быстро сократится. Ваша цель при назначении владельца билета — просто устранить двусмысленность в отношении того, кто должен решать каждую проблему. Не наказывайте сотрудников за просьбу о помощи.

С точки зрения клиента, хорошая система назначения — та, которая направляет проблемы человеку, имеющему большой объем знаний и способному быстро и полностью решить эти проблемы. Однако с управленческой точки зрения, задания иногда должны быть сложными, чтобы сотрудники продолжали расти и учиться в процессе выполнения своей работы. Ваша задача состоит в поиске такого баланса между сильны-

ми сторонами сотрудников и необходимостью решать сложные задачи, чтобы при этом были довольны и клиенты, и сотрудники.

Большие задачи могут быть любыми, вплоть до полномасштабных проектов разработки программного обеспечения. Эти задачи могут потребовать использования формальных инструментов управления проектами и разработки программного обеспечения. Мы не описываем эти инструменты здесь; тем не менее они важны и их не следует упускать из виду.

Иногда системные администраторы знают, что нужно выполнить определенную задачу, но они этого не делают, потому что задача им неприятна. Системный администратор, который укажет на эту забытую, неназначенную или непопулярную задачу, скорее всего, и получит эту задачу как свое задание. Эта ситуация создает конфликт интересов, потому что она побуждает системных администраторов сохранять молчание в таких ситуациях. Не допускайте, чтобы это происходило в вашей организации; дайте вашим системным администраторам возможность прояснить проблемы. Вы можете разрешить им открывать билеты, не назначая владельца или не связывая себя с проблемой, или можете создать псевдоним адреса электронной почты, по которому могут быть адресованы проблемы.

Восприятие пользователями билетных систем

Получение быстрого ответа от реального человека является критическим фактором, определяющим удовлетворенность клиентов, даже если персональный ответ содержит не больше информации, чем автоматизированный. В большинстве случаев гораздо важнее сообщить заявителю, что билет был просмотрен реальным человеком, чем устранить проблему немедленно. Пользователи понимают, что администраторы получают много запросов, и они готовы ждать вашего внимания настолько долго, насколько они сочтут это справедливым и разумным. Но они не хотят, чтобы их игнорировали.

Механизм, посредством которого пользователи отправляют билеты, влияет на их восприятие системы. Убедитесь, что вы понимаете культуру своей организации и предпочтения пользователей. Они хотят веб-интерфейс? Пользовательское приложение? Псевдоним адреса электронной почты? Может быть, они хотят звонить только по телефону!

Также важно, чтобы администраторы нашли время убедиться, что они правильно понимают запросы пользователей. Этот момент кажется очевидным, но вспомните последние пять раз, когда вы отправляли сообщение по электронной почте в службу поддержки клиентов или технической поддержки. Думаем, что было по крайней мере несколько случаев, когда ответ, казалось, не имел никакого отношения к вашему вопросу, — не потому, что эти компании были особенно некомпетентны, а потому, что подробно разбирать проблемы, указанные в билете, сложнее, чем кажется.

Как только вы прочитаете достаточно большую часть билета, чтобы понять, о чем спрашивает клиент, остальная часть билета покажется вам пустыми словами. Боритесь с этим! Клиенты злятся, когда узнают, что билет дошел до адресата, но запрос был неверно истолкован и его необходимо повторно отправить или перерегистрировать. Возникает замкнутый круг.

Билеты часто расплывчаты или неточны, потому что у заявителя нет технического опыта, необходимого для описания проблемы, как это было бы с системным администратором. Однако это не мешает пользователям делать собственные догадки о причинах неполадок. Иногда эти догадки совершенно правильны, а иногда вы должны сначала расшифровать билет, чтобы выяснить, что *думает* пользователь о проблеме, а затем проследить за ходом мысли пользователя, чтобы понять основную проблему.

Типовые билетные системы

В следующих таблицах приведены характеристики нескольких известных билетных систем. В табл. 31.1 указаны системы с открытым исходным кодом, а в табл. 31.2 — коммерческие системы.

В табл. 31.2 показаны некоторые коммерческие альтернативы для управления запросами. Поскольку веб-сайты для коммерческих предложений в основном содержат маркетинговую информацию, такие детали, как язык реализации и интерфейс, не указаны.

Таблица 31.1. Билетные системы с открытым исходным кодом

Имя	Ввод[а]	Язык	База данных[б]	Веб-сайт
Double Choco Latte	W	PHP	MP	`github.com/gnuedcl/dcl`
Mantis	WE	PHP	M	`mantisbt.org`
OTRS	WE	Perl	DMOP	`otrs.org`
RT: Request Tracker	WE	Perl	M	`bestpractical.com`
OSTicket	WE	PHP	M	`osticket.com`
Bugzilla	WE	Perl	MOP	`bugzilla.org`

[а] Типы ввода: W — веб, E — электронная почта.

[б] База данных: D — DB2, M — MySQL, O — Oracle, P — PostgreSQL.

Таблица 31.2. Коммерческие билетные системы

Имя	Масштаб	Веб-сайт
EMC Ionix (Infra)	Огромный	`infracorp.com/solutions`
HEAT	Средний	`ticomix.com`
Jira	Любой	`atlassian.com`
Remedy (теперь BMC)	Огромный	`remedy.com`
ServiceDesk	Огромный	`ca.com/us/service-desk.aspx`
ServiceNow	Любой	`servicenow.com`
Track-It!	Средний	`trackit.com`

Некоторые коммерческие предложения настолько сложны, что для их эксплуатации, настройки и поддержания работоспособности нужны один или два специально назначенных сотрудника. Другие (такие как Jira и ServiceNow) доступны в виде "программного обеспечения как услуги".

Диспетчеризация билетов

Во многих билетных системах, одна из которых широко известна, нерешенной осталась такая проблема: несколько сотрудников не могут эффективно разделять свое внимание между задачами, над которыми они работают прямо сейчас, и очередными запросами, особенно если запросы поступают по электронной почте в личный почтовый ящик. Мы экспериментировали с двумя решениями этой проблемы.

Наша первая попытка заключалась в том, чтобы назначить половину группы системных администраторов дневной смены на обслуживание запросов. Во время смены дежурный должен был попытаться ответить на как можно больше входящих запросов. Недостаток этого подхода заключался в том, что не все системные администраторы

умели правильно отвечать на все вопросы и исправлять все проблемы. Иногда ответы были неуместными, поскольку дежурный был новым и не был знаком с клиентами, их средой или конкретными контрактами на поддержку. В результате вышестоящие сотрудники вынуждены были следить за ситуацией и поэтому не могли сосредоточиться на собственной работе. В итоге качество обслуживания снизилось, и ничего не получилось.

После этого опыта мы создали роль диспетчера, на которую ежемесячно поочередно назначается кто-то из группы старших администраторов. Диспетчер проверяет билетную систему в поисках новых записей и назначает задания конкретным сотрудникам. При необходимости диспетчер связывается с пользователями для получения любой дополнительной информации, необходимой для установления приоритетности запросов. Диспетчер использует самодельную базу данных о навыках работы с персоналом, чтобы решить, кто в команде поддержки имеет соответствующие навыки и время для выполнения данного билета. Диспетчер также гарантирует своевременное выполнение запросов.

31.3. Поддержка локальной документации

Так же, как большинство людей понимают пользу для здоровья от физических упражнений и свежих овощей, каждый ценит хорошую документацию, но имеет туманное представление о том, насколько это важно. К сожалению, это не обязательно означает, что некто должен писать или обновлять документацию просто так. Почему мы должны об этом заботиться, правда?

- Документация уменьшает вероятность единственной точки отказа. Замечательно иметь инструменты, которые быстро развертывают рабочие станции, и распространять исправления с помощью одной команды, но эти инструменты почти бесполезны, если документации для них не существует, а эксперт находится в отпуске или уволился.
- Документация облегчает воспроизводимость. Если правила и процедуры не хранятся в локальной памяти, они вряд ли будут последовательно соблюдаться. Если администраторы не найдут информацию о том, как что-то сделать, они и не смогут это сделать.
- Документация экономит время. Не похоже, что вы экономите время, когда пишете ее, но, потратив несколько дней на повторное решение проблемы, которая раньше уже была решена, но решение было забыто, большинство администраторов убедятся в пользе документации.
- Последнее и самое главное: документация улучшает понятность системы и позволяет производить последующие модификации согласованным образом. Когда изменения производятся на основе только частичного понимания, они часто не вполне соответствуют архитектуре. Со временем энтропия увеличивается, и даже администраторы, которые работают в системе, видят в ней беспорядочную коллекцию приемов. Конечным результатом часто является желание отказаться от всего и начать с нуля.

Локальная документация должна храниться в четко определенном месте, таком как внутренняя вики-система или сторонняя служба, такая как Google Drive. После того как вы убедите своих администраторов документировать конфигурации и методы администрирования, важно также защитить эту документацию. Злоумышленник может на-

нести большой урон, нарушив документацию вашей организации. Убедитесь, что люди, которые нуждаются в документации, могут найти ее и прочитать (сделайте ее доступной для поиска), и каждый, кто поддерживает документацию, может ее изменить. В то же время соблюдайте баланс между доступностью и необходимостью защиты.

Инфраструктура как код

Другая важная форма документации называется "инфраструктура как код". Она может принимать различные формы, но чаще всего рассматривается в виде определений конфигурации (таких как модули Puppets или сценарии Ansible), которые затем могут храниться и отслеживаться в системе контроля версий, такой как Git. Система и ее изменения хорошо документированы в файлах конфигурации, и среда может быть построена и сопоставлена со стандартом на регулярной основе. Такой подход гарантирует, что документация и окружающая среда всегда соответствуют стандарту и актуальны, тем самым решая наиболее распространенную проблему традиционной документации. Дополнительную информацию см. в главе 23.

Стандарты документации

Если вы должны документировать элементы вручную, наш опыт показывает, что самый простой и эффективный способ ведения документации — стандартизация на основе коротких и легких документах. Вместо написания руководства по управлению системой для вашей организации напишите много одностраничных документов, каждый из которых охватывает одну тему.

Начните с большой картины, а затем разбивайте ее на куски, содержащие дополнительную информацию. Если вам нужно углубиться в подробности, напишите дополнительный документ на одну страницу, в котором основное внимание будет уделено особенно сложным шагам.

Такой подход имеет ряд преимуществ.

- Возможно, высшее руководство заинтересовано только в общей настройке вашей среды. Это все, что нужно, чтобы отвечать на вопросы сверху или вести управленческую дискуссию. Не раскрывайте слишком много деталей, иначе вы просто подтолкнете своего босса вмешаться в них.
- То же самое справедливо для клиентов.
- Новый сотрудник или кто-то, кто принимает новые обязанности в вашей организации, нуждается в обзоре инфраструктуры, чтобы стать продуктивным. Нехорошо хоронить таких людей под грудой информации.
- Эффективнее использовать нужный документ, чем просматривать большой.
- Вы можете индексировать страницы, чтобы их было легко найти. Чем меньше времени администраторам приходится тратить на поиск информации, тем лучше.
- Легче поддерживать текущую документацию, когда вы можете это сделать, обновив только одну страницу.

Этот последний момент особенно важен. Поддержание документации в актуальном состоянии является огромной проблемой; когда времени мало, документацией жертвуют в первую очередь. Мы обнаружили, что несколько конкретных подходов позволяют облегчить документирование.

Во-первых, укажите, что документация будет краткой, актуальной и неотшлифованной. Переходите к самой сути. Информация должна быть эффективной. Ничто так не отталкивает от документирования, как перспектива написания диссертации по теории проектирования. Попросите слишком много документации, и вы, возможно, не получите ничего. Рассмотрите возможность разработки простой формы или шаблона для ваших системных администраторов. Стандартная структура помогает избежать "воды" и ориентирует системных администраторов на изложение конкретной информации, а не общих рассуждений.

Во-вторых, внедряйте документацию в процессы. Комментарии в файлах конфигурации являются одними из лучших документов для всех. Они всегда уместны там, где они вам нужны, и их сохранение практически не требует времени. Большинство стандартных файлов конфигурации позволяют оставлять комментарии, и даже те, которые не особого хорошо прокомментированы, часто могут служить источником дополнительной информации.

Локально созданные инструменты могут требовать документацию как часть стандартной информации о конфигурации. Например, инструмент, который настраивает новый компьютер, может потребовать информацию о владельце компьютера, его местоположении, состоянии поддержки и финансовые данные, даже если эти факты не имеют прямого отношения к конфигурации программного обеспечения устройства.

Документация не должна создавать избыточности информации. Например, если вы поддерживаете общий список систем для всех компьютеров, не должно быть другого места, где эта информация обновляется вручную. Делать обновления в нескольких местах — это не только пустая трата времени, но и высокая вероятность того, что со временем проявятся несоответствия. Если эта информация требуется в других контекстах и файлах конфигурации, напишите сценарий, который получает ее от мастера конфигурации или обновляет. Если вы не можете полностью устранить избыточность, по крайней мере должно быть понятно, какой источник является авторитетным. Кроме того, напишите инструменты, чтобы уловить несоответствия, возможно, регулярно запуская их через планировщик `cron`.

🕮 Дополнительную информацию о планировщике `cron` см. в разделе 4.9.

Появление таких инструментов, как вики, блоги и другие простые системы управления знаниями, значительно облегчило отслеживание IT-документации. Создайте хранилище, где все ваши документы могут быть найдены и обновлены. Однако не забудьте сохранить его организованность. Одна страница вики-системы с 200 дочерними страницами в одном списке слишком громоздкая и сложная в использовании. Не забудьте включить функцию поиска, чтобы получить максимальную отдачу от вашей системы.

31.4. Разделение окружающей среды

Организации, которые создают и разворачивают собственное программное обеспечение, нуждаются в отдельных средах разработки, тестирования и производства, чтобы выпуски можно было передавать в общее пользование с помощью структурированного процесса.[2] Отдельные, но идентичные; убедитесь, что при обновлении систем разработки изменения распространяются также на тестовые и производственные среды. Разумеется, сами настройки конфигурации должны подчиняться тому же типу структу-

[2]Во многих случаях это утверждение относится также к организациям, в которых запущено готовое комплексное программное обеспечение, такое как ERP или финансовые системы.

рированного управления выпуском, что и код. "Изменения конфигурации" включают все: от исправления операционных систем до обновлений приложений и административных изменений.

Исторически сложилось так, что стандартная практика защищает производственную среду путем разделения ролей на протяжении всего процесса продвижения. Например, разработчики, обладающие правами администратора в среде разработки, — это не те люди, у которых есть привилегии администратора и продвижения в других средах. Существовало опасение, что недовольный разработчик, имеющий разрешения на продвижение кода, мог бы вставлять вредоносный код на стадии разработки, а затем продвигать его в производственную среду. Если одобрение дистрибутива и продвижение осуществляют разные люди, им необходимо будет сговориться или одновременно сделать ошибки, чтобы проблемы могли попасть в производственные системы.

К сожалению, ожидаемые выгоды от таких драконовских мер редко достигаются. У людей, продвигающих код, часто нет навыков или времени для пересмотра изменений кода на уровне, который фактически выявил бы злонамеренный фрагмент. Вместо того чтобы помогать, система создает ложное чувство защиты, вводит ненужные контрольно-пропускные пункты и тратит ресурсы.

В эпоху методологии DevOps эта проблема решается по-другому. Вместо отдельных ролей предпочтительным подходом является отслеживание всех изменений "как кода" в репозитории (например, Git), который имеет неизменяемый контрольный журнал. Любые нежелательные изменения можно проследить вплоть до конкретного человека, который их представил, поэтому строгое разделение ролей не нужно. Поскольку изменения конфигурации применяются автоматическим способом в каждой среде, идентичные изменения могут быть выполнены в более низких средах (например, разработки или тестирования) до того, как они будут продвинуты в производство, чтобы гарантировать, что непредвиденные последствия не проявятся. Если проблемы обнаружены, возвращение в исходное состояние происходит настолько же просто, как выявление проблемной фиксации и ее временный обход.

В идеальном мире ни разработчики, ни сотрудники отдела эксплуатации не должны иметь административных привилегий в производственной среде. Вместо этого все изменения должны выполняться с помощью автоматизированного отслеживаемого процесса, который имеет собственные привилегии. Несмотря на то что это достойная и весьма желанная цель, наш опыт заключается в том, что для большинства организаций это еще не реально. Работайте над этой утопической идеей, но не попадитесь в ловушку.

31.5. Восстановление после аварий

Работа организации зависит от функционирования информационной среды. Системный администратор отвечает не только за повседневные операции, но и за наличие плана действий на случай возникновения различных экстренных ситуаций, по крайней мере тех, которые можно предвидеть. Подготовка к таким масштабным проблемам влияет как на общий план работы, так и на способ выполнения повседневных операций. В этом разделе мы рассмотрим разные виды аварий, данные, которые необходимо восстановить, а также важные элементы планов по возобновлению работы.

Оценка рисков

Прежде чем завершить разработку плана восстановления после аварий, целесообразно оценить степень риска, чтобы понять, какими активами вы располагаете, каким ри-

скам подвергаетесь и как можно смягчить последствия аварии. Специальный документ NIST 800-30 детализирует обширный процесс оценки степени риска. Вы можете загрузить его с сайта `nist.gov`.

В ходе оценки степени риска необходимо составить список потенциальных угроз, от которых вы хотите защититься. Угрозы не являются одинаковыми, и вам, возможно, понадобится несколько различных планов, чтобы покрыть полный спектр возможностей. В качестве примера перечислим угрозы общего характера.

- Злонамеренные пользователи, как внутренние, так и внешние[3]
- Наводнения
- Пожары
- Землетрясения
- Ураганы и торнадо
- Магнитные бури и броски электропитания
- Перебои в электропитании, короткие и долгосрочные
- Чрезвычайно высокая температура или отказ охлаждающего оборудования
- Сбой в работе телекоммуникационного оборудования провайдера или облачной системы
- Отказы аппаратных средств ("зависшие" серверы, "сгоревшие" жесткие диски)
- Действия террористов
- Зомби апокалипсис
- Отказы сетевых устройств (маршрутизаторов, коммутаторов, кабелей).
- Случайные ошибки пользователей (удаленные или поврежденные файлы и базы данных, потерянная информация о конфигурации, забытые пароли и т.д.).

Для каждой потенциальной угрозы рассмотрите и запишите все возможные последствия.

Как только вы поймете угрозу, расставьте приоритеты служб в пределах своей информационной среды. Составьте таблицу, в которой перечисляются службы и их приоритеты. Например, компания, предлагающая "программное обеспечение как службу", может оценивать свой внешний веб-сайт как службу первостепенной важности, в то время как офис с простым информационным внешним веб-сайтом может не волноваться о том, что сайт не работает во время стихийных бедствий.

Планирование мероприятий по восстановлению

Все больше организаций проектируют свои критические системы так, чтобы можно было автоматически переключаться на резервные серверы в случае возникновения проблем. Это прекрасная идея, если простои в работе служб не допускаются. Тем не менее не становитесь жертвой веры в то, что зеркалирование ваших данных отменяет необходимость в их резервном копировании. Даже если ваши центры обработки дан-

[3]Причиной около половины нарушений в области безопасности являются действия внутренних пользователей. В большинстве организаций неправильные действия внутренних пользователей остаются наиболее вероятными причинами сбоев.

ных расположены достаточно далеко, вы вполне можете потерять в них все данные.[4] Не забудьте включить резервное копирование данных в свой план по борьбе с аварийными ситуациями.

🕮 Облачные вычисления описаны в главе 9.

Облачные вычисления — еще один ресурс для борьбы со стихийными бедствиями, который становится все более популярным. Благодаря таким службам, как EC2 компании Amazon, вы можете настроить удаленный сайт и запустить его за несколько минут без необходимости платить за специализированные аппаратные средства. Вы платите только за то, что используете.

В план по восстановлению после аварийных ситуаций должны быть включены перечисленные ниже разделы (составлено на основе стандарта аварийного восстановления NIST 800-34).

- *Введение* — цель и предмет документа.
- *Понятие операций* — описание системы, цели восстановления, классификация информации, порядок преемственности, обязанности.
- *Уведомление и активация* — процедуры уведомления, процедуры оценки повреждений, активация плана.
- *Восстановление* — последовательность событий и процедур, требуемых для восстановления работоспособности потерянных систем.
- *Возврат к нормальному функционированию* — параллельная обработка, тестирование восстановленной системы, возвращение к нормальному функционированию, отмена введенного аварийного плана.

Мы привыкли использовать сеть для общения и доступа к документам. Однако эти средства могут стать недоступными или оказаться под угрозой после инцидента. Поэтому сохраняйте локально все соответствующие контакты и процедуры. Вы должны знать, где можно получить свежие резервные копии своих данных и как использовать их независимо от сетевых данных.

Во всех сценариях восстановления вам понадобится доступ и к автономным, и к интерактивным копиям важных данных. Интерактивные копии по возможности должны быть сохранены на отдельном компьютере, на котором есть богатый набор инструментов, настроенная ключевая среда для системного администрирования, запущен собственный сервер имен, полный локальный файл `/etc/hosts`, подключение к принтеру и нет никаких зависимостей от ресурсов, расположенных на других компьютерах.

Ниже приведен список полезных данных для хранения в среде восстановления при аварийных ситуациях.

- Схема процедуры восстановления: кому звонить, что сказать
- Номера телефонов сервисного центра и клиентов
- Ключевые местные номера телефонов: полиция, пожарные, сотрудники, начальник
- Информация для входа в облачный сервис
- Набор резервных копий данных и график их создания
- Карты сети

[4] Злонамеренные хакеры и программы могут легко вывести из строя организацию, которая игнорирует необходимость создания автономных резервных копий, предназначенных только для чтения.

- Регистрационные номера программного обеспечения, лицензионные данные и пароли
- Копии инсталляционных пакетов программного обеспечения (могут храниться в формате ISO)
- Копии инструкций по эксплуатации ваших систем
- Контактная информация поставщика оборудования
- Административные пароли
- Данные о конфигурациях аппаратного и программного обеспечения: версии операционных систем, пакеты обновлений, таблицы разделов дисков и т.п.
- Инструкции о порядке запуска систем, работоспособность которых должна быть восстановлена в первую очередь

Подбор персонала на случай аварии

Нужно заблаговременно решить вопрос о том, кто будет справляться с ситуацией в случае, если произойдет авария. Следует составить план действий и записать номера телефонов тех, кому надлежит звонить в такой ситуации. Мы пользуемся небольшой ламинированной карточкой, на которой мелким шрифтом напечатаны все важные имена и номера телефонов: она очень удобна, потому что легко помещается в бумажник.

Лучше всего назначать ответственным одного из системных администраторов, а не IT-руководителя (обычно он плохо подходит для этой роли).

Ответственным в случае аварии должен быть кто-то, кто пользуется авторитетом и не боится принимать трудные решения при наличии минимального количества информации (наподобие решения отключить от сети весь отдел целиком). Способность принимать такие решения, уведомлять о них понятным образом и фактически руководить персоналом во время кризиса, пожалуй, важнее теоретических знаний о системном и сетевом управлении.

При составлении плана аварийных мероприятий обычно предполагается, что административный персонал будет на месте, когда произойдет авария, и сумеет справиться с ситуацией. К сожалению, люди иногда болеют, уходят на курсы повышения квалификации, уезжают в отпуск, а в тяжелые времена вообще могут даже становиться крайне недружелюбными. Поэтому стоит заранее продумать, где можно будет быстро найти дополнительную помощь. (Если система не слишком устойчива, а персонал неопытен, то недостаточное количество администраторов уже само по себе является аварийной ситуацией.)

Одним из решений может быть договоренность с какой-нибудь местной консультационной компанией или университетом, где всегда есть талантливые и готовые помочь администраторы. Конечно, если у них когда-нибудь возникнут проблемы, тогда вы тоже должны будете оказать им необходимую помощь. Но самое главное — это не работать на пределе: наймите достаточное количество администраторов и не требуйте, чтобы они работали по 12 часов в сутки.

Проблемы с безопасностью

Об обеспечении безопасности системы подробно рассказывалось в главе 27. Однако здесь тоже стоит коснуться этой темы, потому что вопросы безопасности вли-

яют на многие из выполняемых системным администратором задач. Ни один аспект стратегии управления организации не должен разрабатываться без учета безопасности.

В главе 27 по большей части рассказывалось о способах предотвращения проблем с безопасностью. Однако продумывание способов восстановления системы после происшедшего из-за связанных с безопасностью проблем инцидента является не менее важным.

Взлом веб-сайта относится к числу наиболее серьезных нарушений системы безопасности. Для системного администратора, работающего в компании, которая занимается предоставлением услуг веб-хостинга, такой инцидент может превратиться в настоящую катастрофу, особенно если речь идет о сайтах, использующих данные о пластиковых картах пользователей. При этом будет поступать лавина телефонных звонков от клиентов, средств массовой информации, VIP-клиентов компании, которые только что услышали о произошедшем взломе в новостях. Кто будет отвечать на эти звонки? Что он должен говорить? Кто будет главным? Кто что будет делать? Тем системным администраторам, которые работают в очень популярных компаниях, обязательно следует продумать такой сценарий, подготовить подходящие ответы и план действий и, возможно, даже провести тренировочную проверку, чтобы отработать детали.

Для сайтов, которые имеют дело с данными о платежных картах, всегда предусматриваются правила поведения в случае взлома с целью кражи информации. Поэтому системный администратор обязательно должен привлечь к планированию мероприятий на случай проблем с безопасностью сотрудников юридического отдела своей организации и позаботиться о наличии номеров телефонов и имен людей, которым следует звонить во время кризиса.

Когда в новостях объявляют, что такой-то веб-сайт был атакован хакерами, возникает ситуация, напоминающая аварию на дороге: все бросаются посмотреть, что же произошло, и в результате интернет-трафик сильно возрастает, нередко настолько сильно, что все, что администраторам наконец-то с таким трудом удалось восстановить, снова может оказаться под угрозой. Поэтому, если веб-сайт не рассчитан на 25-процентное (или более высокое) пиковое увеличение трафика, следует позаботиться о наличии устройств выравнивания нагрузки, которые будут просто направлять превышающие норму запросы на специальный сервер, а тот будет возвращать страницу со следующим сообщением: "Извините, сервер слишком загружен и поэтому в данный момент не может обработать ваш запрос". Разумеется, хорошо продуманный заранее план по возобновлению работы, включающий автоматическое масштабирование в облаке (см. главу 9), позволит избежать такой ситуации.

Разработайте подробное руководство по действиям в чрезвычайных ситуациях, чтобы исключить непродуктивную работу при устранении проблем с безопасностью. Более подробно об этом шла речь в разделе 27.12.

31.6. Инструкции и процедуры

Исчерпывающие инструкции и процедуры, касающиеся информационных технологий, служат основой для работы современных организаций. Инструкции устанавливают нормы для пользователей и администраторов и способствуют согласованной работе всех вовлеченных в нее сотрудников. Все чаще инструкции требуют подтверждения в форме подписи или другого доказательства, что пользователь согласился их соблюдать. Хотя это может показаться чрезмерным формализмом, такое подтверждение представляет собой действительно отличный способ защитить администраторов.

Хорошим основанием для разработки инструкций является стандарт ISO/IEC 27001:2013. Он сочетает общую стратегию в области информационных технологий с другими важными элементами, такими как информационная безопасность и роль отдела кадров. В следующих разделах мы обсудим стандарт ISO/IEC 27001:2013 и выдвинем на первый план некоторые из его самых важных и полезных элементов.

Различие между инструкциями и процедурами

Инструкции и процедуры — это разные категории, но их часто путают, и иногда они даже используются как синонимы, что вызывает путаницу. Для того чтобы правильно понимать их сущность, рассмотрим следующие аспекты.

- Инструкции — это документы, устанавливающие требования или правила. Требования обычно определяются на относительно высоком уровне. Примером инструкции можно считать требование, чтобы инкрементные резервные копии создавались ежедневно, а полные резервные копировании — каждую неделю.
- Процедуры — это документы, описывающие процесс выполнения требований или правил. Например, процедура, связанная с описанной выше инструкцией, могла бы быть сформулирована примерно так: "Инкрементные резервные копии выполняются с помощью программы Backup Exec, инсталлированной на сервере `backups01`..."

Это различие важно, потому что инструкции не должны изменяться очень часто. Они могут пересматриваться ежегодно и, возможно, в них поменяется один или два раздела. С другой стороны, процедуры уточняются непрерывно, поскольку постоянно изменяется архитектура, системы и конфигурации.

Некоторые стратегические решения диктуются программным обеспечением, которое вы используете, или инструкциями внешних групп, например интернет-провайдерами. Если необходимо защитить конфиденциальные данные пользователей, некоторые инструкции составляются в категоричной форме. Мы называем эти инструкции "не подлежащими обсуждению".

В частности, мы полагаем, что IP-адресами, именами хостов, идентификаторами пользователей, идентификаторами групп и именами пользователей необходимо управлять централизованно. Некоторые организации (транснациональные корпорации, например) являются слишком большими, чтобы осуществить эту политику, но если вы можете реализовать ее, то централизованное управление сделает все намного проще. Мы знаем компанию, которая реализует политику централизованного управления для 35 тысяч пользователей и 100 тысяч компьютеров. Таким образом, порог, после которого организация становится слишком крупной для централизованного управления, должен быть довольно высоким.

Другие важные проблемы имеют более широкую сферу влияния, чем ваша локальная группа системных администраторов.

- Борьба со взломами системы безопасности
- Управление экспортом файловой системы
- Критерии выбора паролей
- Удаление регистрационных записей
- Материал, защищенный авторским правом (например, MP3 и DVD)
- Программное пиратство

Лучшие практики применения инструкций

Инструкции могут быть ограничены несколькими рамками, но все они охватывают примерно одинаковую область. Ниже перечислены примеры инструкций, которые, как правило, включаются в стратегический набор инструкций, относящихся к информационным технологиям.

- Информационная политика безопасности
- Соглашения о возможности подключения посторонних организаций
- Политика управления активами
- Информационная система классификации данных
- Политика безопасности сотрудников
- Физическая политика безопасности
- Политика управления доступом
- Стандарты безопасности для разработки и обслуживания новых систем
- Политика управления инцидентами
- Управление непрерывностью бизнеса (аварийное восстановление)
- Стандарты хранения данных
- Защита конфиденциальности
- Политика соответствия установленным требованиям закона

Процедуры

Процедуры в форме контрольных списков или рецептов могут формализовать существующую практику. Они полезны и для новых системных администраторов, и для опытных людей. Еще лучше процедуры, которые содержат выполняемые сценарии или отражаются в инструментах управления конфигурацией, таких как Ansible, Puppet или Chef. В долгосрочной перспективе большинство процедур должны быть автоматизированы.

У стандартных процедур есть несколько преимуществ.

- Рутинные операции всегда выполняются единообразно
- Контрольные списки уменьшают вероятность ошибок или забытых операций
- По рецепту системный администратор работает быстрее
- Изменения самодокументируются
- Письменные процедуры обеспечивают измеримый стандарт корректности

Вот часть общих задач, для которых можно сформулировать процедуры.

- Подключение компьютера
- Добавление пользователя
- Конфигурирование локального компьютера
- Настройка процедур резервного копирования для нового компьютера
- Защита нового компьютера
- Удаление старого компьютера
- Повторный запуск сложных компонентов программного обеспечения

- Перезагрузка веб-серверов, которые не отвечают на запросы или "зависли"
- Обновление операционной системы
- Установка исправлений
- Установка пакета прикладных программ
- Обновление наиболее важных программ
- Резервное копирование и восстановление файлов
- Удаление старых резервных копий
- Аварийный останов системы (всех компьютеров; кроме самых важных; и т.д.)

Многие вопросы находятся в непосредственной связи политик и процедур. Например:

- Кто из сотрудников может иметь учетную запись в вашей сети?
- Что происходит, когда они увольняются?

Такие вопросы следует строго регламентировать, чтобы не стать жертвой известной уловки четырехлетних детей: "Мама не разрешила, надо спросить у папы!"

31.7. Соглашения о качестве оказываемых услуг

Для того чтобы отдел информационных технологий успешно справлялся со своими обязанностями, стараясь угождать пользователям и удовлетворяя потребности предприятия, все детали необходимо согласовать и зафиксировать в договоре о качестве оказываемых услуг. Хороший договор предусматривает соответствующий уровень обслуживания и является документом, на который можно ссылаться в проблемных ситуациях. (Однако помните, что отдел информационных технологий помогает, а не мешает пользователям!)

Когда что-то выходит из строя, пользователи хотят знать, когда это будет исправлено. Их не интересует, какой жесткий диск или генератор сломались и почему; оставьте эту информацию для своих административных отчетов.

С точки зрения пользователя, никакие новости не являются хорошими. Система или работает, или нет, и в последнем случае не имеет значения, почему. Максимальное удовольствие наши клиенты получают, если они даже не замечают, что мы существуем! Грустно, но факт.

Соглашение о качестве оказываемых услуг помогает помирить конечных пользователей и технический персонал. Хорошо написанное соглашение учитывает каждую проблему, упомянутую в следующих разделах.

Спектр услуг и их описание

В договоре о качестве оказываемых услуг описывается то, что организация может ожидать от отдела информационных технологий. Он должен быть написан в терминах, которые могут быть поняты нетехническими сотрудниками. Перечислим в качестве примера некоторые из этих служб.

- Электронная почта
- Чаты
- Интернет и веб-доступ
- Файловые серверы

- Бизнес-приложения
- Аутентификация

В договоре также должны быть определены стандарты, которых будет придерживаться отдел информационных технологий. Например, раздел о доступности услуг должен определять часы работы, согласованные окна обслуживания и время, в течение которого будет доступен технический персонал, чтобы обеспечить интерактивную поддержку. Одна организация могла бы решить, что регулярная поддержка должна быть доступна с 8:00 до 18:00 в будние дни, а чрезвычайная поддержка — круглосуточно. Другая организация могла бы решить, что нуждается в стандартной интерактивной поддержке, доступной всегда.

Представим список проблем, которые следует рассмотреть, документируя ваши стандарты.

- Время отклика
- Обслуживание (и время отклика) в течение выходных и неурочных часов
- Посещения на дому (поддержка для домашних компьютеров)
- Специальное (уникальное или патентованное) аппаратное обеспечение
- Политика модернизации (устаревшие аппаратные средства, программное обеспечение и т.д.)
- Поддерживаемые операционные системы
- Поддерживаемые облачные платформы
- Стандартные конфигурации
- Хранение данных
- Программное обеспечение специального назначения

Рассматривая стандарты услуг, имейте в виду, что многие пользователи захотят самостоятельно настроить свою среду (или даже свои системы), если не будет установлено программное обеспечение, чтобы это предотвратить. Стереотипный ответ на эти попытки — запретить всем пользователям осуществлять любые модификации. Это упрощает работу отдела информационных технологий, но не всегда является лучшей стратегией для всей организации.

Укажите эту проблему в своем соглашении о качестве оказываемых услуг и попытайтесь стандартизировать ее решение на нескольких определенных конфигурациях. Иначе ваше стремление к облегчению обслуживания и быстрому росту в рамках всей организации встретят серьезные препятствия. Поощряйте своих творческих сотрудников предлагать модификации, в которых они нуждаются, и будьте заботливы и щедры, учитывая их предложения в своих стандартных конфигурациях. Если вы не сделаете этого, то ваши пользователи будут упорно нарушать ваши правила.

Стратегии управления очередями

Пользователи должны знать не только, какие услуги им будут оказаны, но и схему приоритетов, используемую для управления очередью заданий. Схемы приоритетов всегда оставляют возможность для маневра, но все же попытайтесь спроектировать такую схему, которая охватывала бы большинство ситуаций, почти или вовсе не прибегая к исключениям. Некоторые факторы, связанные с приоритетом, перечислены ниже.

- Важность услуги для всей организации
- Влияние на безопасность (было ли нарушение правил безопасности?)
- Оплаченный или оговоренный уровень сервисного обслуживания
- Количество задействованных пользователей
- Важность любого релевантного крайнего срока
- Скандальность задействованных пользователей ("скрипучие колеса")
- Важность задействованных пользователей (это дело тонкое, но будем честными: у некоторых людей в вашей организации есть больше авторитета, чем у других).

Хотя на ваше ранжирование будут влиять все эти факторы, мы рекомендуем подходить к исключениям с простым сводом правил и долей здравого смысла. В основном, мы используем следующие приоритеты.

- Много людей не могут работать успешно
- Один человек не может работать успешно
- Запросы на усовершенствования

Если два или больше запросов имеют высший приоритет и они не могут выполняться параллельно, мы делаем выбор, основываясь на серьезности проблемы (например, неработающая электронная почта доставляет неудобства почти всем, тогда как временное отсутствие веб-службы могло бы помешать только нескольким людям). Очереди с более низкими приоритетами обычно обрабатываются по принципу FIFO.

Показатели соответствия

Соглашение о качестве оказываемых услуг должно определить, как организация измеряет ваш успех при выполнении условий соглашения. Цели и ориентиры позволяют работникам совместно стремиться к общему результату и могут заложить основу для сотрудничества во всей организации. Конечно, вы должны удостовериться, что у вас есть инструменты, позволяющие измерить согласованные показатели.

Как минимум, вы должны отследить следующие показатели вашей информационной инфраструктуры.

- Процент или количество проектов, законченных вовремя и в пределах бюджета
- Процент или количество выполненных пунктов соглашения о качестве оказываемых услуг
- Процент продолжительности работы системы (например, электронная почта доступна через службу Q1 в течение 99,92% времени)
- Процент или число билетов, проблема которых была удовлетворительно решена
- Среднее время решения проблемы, описанной в билете
- Процент или количество нарушений системы безопасности, обработанных согласно установленной процедуре

31.8. Соответствие законам и стандартам

IT-аудит и управление в настоящее время являются очень популярными. Инструкции и квазистандарты для спецификации, проведения измерений и сертификации соответствия законам породили множество аббревиатур: SOX, ITIL, COBIT и ISO 27001, и это

только часть из них. К сожалению, эта смесь букв оставляет у системных администраторов неприятный осадок, поэтому сейчас ощущается недостаток программного обеспечения, предназначенного для реализации всех средств управления, необходимых для обеспечения соответствия с действующим законодательством.

Некоторые из основных рекомендательных стандартов, руководящих принципов, промышленных концепций и законодательных требований, которые могут касаться системных администраторов, перечислены ниже. Законодательные требования являются в значительной степени специфичными для Соединенных Штатов Америки.

Обычно стандарты определяются типом организации или обрабатываемых данных. Организации, находящиеся вне юрисдикции США, должны сами выяснить, какие регуляторные правила распространяются на них.

- **CJIS (Информационные системы уголовного судопроизводства)** —стандарт, относящийся к организациям, которые отслеживают криминальную информацию и пользуются базами данных ФБР. Его требования могут быть найдены на странице `fbi.gov/hq/cjisd/cjis.htm`.
- **COBIT** — рекомендации для информационного управления, основанного на передовом промышленном опыте. Они разработаны совместно Ассоциацией аудита и управления информационными системами (Systems Audit and Control Association — ISACA) и Институтом информационного управления (IT Governance Instutute — ITGI); см. детали на сайте `isaca.org`. Задача рекомендаций COBIT состоит в том, чтобы "исследовать, развивать, предавать гласности и продвигать авторитетный, современный, международный набор общепринятых целей управления информационными технологиями для ежедневного использования менеджерами и аудиторами."

  Первая редакция этих рекомендаций была выпущена в 1996 году, сейчас действует версия 5.0, изданная в 2012 году. Последняя версия разрабатывалась под сильным влиянием требований закона Сарбейнза–Оксли (Sarbanes-Oxley). Она включает 37 целей высокого уровня, которые разделены на пять категорий: согласование, планирование и организация (Align, Plan, and Organize — APO), создание, приобретение и реализация (Build, Acquire, and Implement — BAI), поставка, обслуживание и поддержка (Deliver, Service, and Support — DSS), мониторинг, оценка и доступ (Monitor, Evaluate, and Access — MEA), а также оценка, управление и мониторинг (Evaluate, Direct, and Monitor — EDM).
- **COPPA (Children's Online Privacy Protection Act** — Закон о защите конфиденциальной информации о детях в Интернете); регулирует работу организаций, которые собирают или хранят информацию о детях, не достигших 13 лет. Для сбора определенной информации необходимо разрешение родителей; см. детали на сайте `ftc.gov`.
- **FERPA (Family Educational Rights and Privacy Act** — Закон о правах семьи на образование и неприкосновенность частной жизни); относится ко всем учреждениям, являющимся получателями федеральной помощи, которой управляет министерство образования. Этот закон защищает информацию о студентах и предоставляет студентам определенные права относительно их данных; см. детали на сайте `ed.gov`.
- **FISMA (Federal Information Security Management Act** — Закон об управлении информационной безопасностью в федеральном правительстве); относится ко всем правительственным учреждениям и подрядчикам правительственных учреждений. Это большой и довольно неопределенный набор требований, которые нацелены

на согласование со множеством публикаций об информационной безопасности, выпущенных институтом NIST, Национальным институтом по стандартизации и технологии (National Institute of Standards and Technology). Независимо от того, подпадает ваша организация под мандат FISMA или нет, документы NIST заслуживают внимания; см. `nist.gov`.

- **Концепция Safe Harbor (правило безопасной гавани) Федеральной комиссии по торговле США (FTC)** представляет собой мост между подходами США и Европейского Союза к законодательству, защищающему частную жизнь, и определяет способ, с помощью которого американские организации могут взаимодействовать с европейскими компаниями, чтобы продемонстрировать защиту своей информации; см. `export.gov/safeharbor`.
- **Закон Грэмма–Лича–Блайли (Gramm-Leach-Bliley Act — GLBA)** регулирует использование финансовыми учреждениями конфиденциальной информации потребителей. Если вы задавались вопросом, почему банки, выпускающие платежные карты, брокеры и страховщики забрасывали вас уведомлениями о конфиденциальности, то это следствие закона Грэмма–Лича–Блайли; см. детали на сайте `ftc.gov`. В настоящее время самая полная информация по закону Грэмма–Лича–Блайли находится в разделе Tips & Advice этого сайта. В качестве глубокой ссылки можно использовать сокращение `goo.gl/vv2011`.
- **Закон HIPAA (Health Insurance Portability and Accountability Act** — Закон об отчетности и безопасности медицинского страхования) относится к организациям, которые передают или хранят защищенную медицинскую информацию (иначе PHI). Это широкий стандарт, который был первоначально предназначен для борьбы с растратами, мошенничеством и злоупотреблениями в области здравоохранения и медицинского страхования, но теперь он используется для того, чтобы измерить качество и улучшить безопасность информации о здоровье; см. `hhs.gov/ocr/privacy/index.html`.
- **ISO 27001:2013 и ISO 27002:2013** — это рекомендательная (и информативная) коллекция передового опыта, связанного с безопасностью организаций, использующих информационные технологии; см. `iso.org`.
- **CIP (Critical Infrastructure Protection)** — семейство стандартов, разработанных корпорацией North American Electric Reliability Corporation (NERC), которые способствуют защите инфраструктурных систем, таких как электроснабжение, телефонные линии и финансовые сети, от стихийных бедствий и терроризма. В полном соответствии с учебной иллюстрацией ницшеанского понятия "жажды власти" оказывается, что большая часть экономики попадает в один из 17 секторов "критических инфраструктур и ключевых ресурсов" (CI/KR) и поэтому очень нуждается в стандартах CIP. Организации, попадающие в эти сектора, должны оценить свои системы и защищать их соответствующим образом; см. `nerc.com`.
- **Стандарт PCI DSS** (Payment Card Industry data Security Standard — Стандарт защиты информации в индустрии платежных карт) был создан консорциумом платежных брендов, включая American Express, Discover, MasterCard, JCB и Visa. Он охватывает вопросы управления данными о платежной карте и относится к любой организации, которая принимает платежи по пластиковой карте. Стандарт имеет два варианта: самооценку для небольших организаций и внешний аудит для организаций, которые обрабатывают много сделок; см. `pcisecuritystandards.org`.

- **Правила Red Flags Rules** Федеральной Комиссии по торговле США требуют, чтобы любой, кто расширяет кредит на потребителей (т.е. любая организация, которая отсылает счета), осуществлял формальную программу, предотвращающую и обнаруживающую "хищение персональных данных". Правила требуют, чтобы эмитенты кредитов разработали эвристические правила для того, чтобы идентифицировать подозрительные манипуляции со счетами. Для выяснения деталей наберите фразу "red flag" в поисковой строке на сайте `ftc.gov`.
- **Библиотека Information Technology Infrastructure Library (ITIL)** в 1990-х и 2000-х годах стала фактическим стандартом для организаций, ищущих полноценное решение для управления информационными услугами. Во многих крупных организациях была развернута формальная программа ITIL, дополненная назначениями менеджеров проекта для каждого процесса, менеджеров, управляющих менеджерами проектов, а также системой отчетности для менеджеров, управляющих менеджерами проектов. В большинстве случаев результаты не были благоприятными. Зацикленность на процессах в сочетании с разделением функций привел к неразрешимым IT-конфликтам. Эта бюрократическая цепочка создала возможности для небольшого количества стартапов, получивших возможность забрать долю рынка у хорошо зарекомендовавших себя компаний, в результате чего многие IT-специалисты остались без работы. Мы надеемся, что увидели последний из ITIL. Некоторые говорят, что DevOps — это методология против ITIL.
- **Раздел IT General Controls (ITGC) в законе Сарбейнза–Оксли** (SOX), последний по расположению, но не по важности, относится ко всем акционерным обществам и разработан, чтобы защитить акционеров от бухгалтерских ошибок и мошеннических методов; см. `sec.gov/rules/final/33-8124.htm`.

Некоторые из этих стандартов содержат хорошие рекомендации даже для организаций, которые не обязаны придерживаться их. Вероятно, стоит просмотреть некоторые из них, чтобы понять, содержат ли они какие-либо рекомендации, которые вы, возможно, захотите принять. Если у вас нет других ограничений, проверьте NERC CIP и NIST 800-53; они являются нашими фаворитами в отношении тщательности и применимости к широкому кругу ситуаций.

Национальный институт стандартов и технологии (NIST) издает множество стандартов, полезных для администраторов и технологов. Некоторые из популярных стандартов упомянуты ниже, но если вам скучно и вы ищете стандарты, можете зайти на его вебсайт. Вы не будете разочарованы.

Стандарт NIST 800-53 (Рекомендуемые средства управления системами безопасности для федеральных информационных систем и организаций) описывает, как оценить безопасность информационных систем. Если ваша организация разработала внутреннее приложение, которое хранит конфиденциальную информацию, этот стандарт может помочь вам удостовериться, что вы действительно защитили ее. Однако остерегайтесь: процедура согласования со стандартом NIST 800-53 не для слабонервных. Вы, вероятно, столкнетесь с документом объемом около 100 страниц со множеством мучительных деталей.[5]

Стандарт NIST 800-34 (Принципы планирования на случай непредвиденных ситуаций для информационных систем) является библией аварийного восстановления, разработанной организацией NIST. Он предназначен для правительственных учреждений, но

[5]Если вы планируете сотрудничество с правительственными учреждениями США, от вас могут потребовать соответствия стандарту NIST 800-53, хотите вы этого или нет...

любая организация может извлечь из него выгоду. Следование стандарту планирования NIST 800-34 требует времени, но это вынуждает вас ответить на такие важные вопросы, как: "Какие системы являются самыми важными?", "Сколько времени мы можем обойтись без этих систем?" и "Как мы собираемся восстанавливаться, если наш основной центр обработки данных потерян?"

31.9. Правовые вопросы

Правительство США и некоторые штаты издали законы о преступлениях в области компьютерной техники. На федеральном уровне с начала 1990-х годов существовало два закона, а теперь их стало больше.

- Федеральный Закон о конфиденциальности связи (Electronic Communications Privacy Act).
- Закон о компьютерном мошенничестве и компьютерных злоупотреблениях (Computer Fraud and Abuse Act).
- Закон о компьютерных кражах (No Electronic Theft Act).
- Закон об авторских правах (Digital Millenium Copyright Act).
- Закон о конфиденциальности электронной почты (The Email Privacy Act)
- Закон о кибербезопасности 2015 года (Cybersecurity Act of 2015).

Основными правовыми проблемами в настоящее время являются ответственность системных администраторов, сетевых операторов и облачных провайдеров; пиринговые сети для обмена файлами между пользователями, защита авторских прав и конфиденциальность. В разделе рассматриваются эти и другие юридические вопросы, связанные с системным администрированием.

Конфиденциальность

Частную жизнь всегда было трудно скрыть, но с развитием Интернета она подвергается еще большей опасности, чем когда-либо. Медицинская документация неоднократно раскрывалась с помощью плохо защищенных систем, украденных ноутбуков и магнитных лент с резервными копиями данных, которые находились не в положенном месте. Базы данных, содержащие номера пластиковых карт, всегда подвергались атакам. Веб-сайты, предлагающие антивирусное программное обеспечение, фактически сами устанавливают программы-шпионы. Поддельная электронная почта поступает почти ежедневно в виде фиктивных писем от вашего банка, в которых утверждается, что у вас возникли проблемы с вашим счетом и требуется, чтобы вы проверили свои учетные данные.[6]

Технические меры не могут защитить от этих видов атак, потому что они направлены на самую уязвимую часть вашего сайта: его пользователей. Поэтому лучшей его защитой является база хорошо обученных пользователей. В первом приближении можно сказать, что в никаком законно посланном сообщении электронной почты или на веб-сайте никогда не будет перечисленных ниже вещей.

[6]Обычно внимательное изучение электронной почты показывает, что данные будут отправлены какому-нибудь хакеру из Восточной Европы или Азии, а не в ваш банк. Этот тип атаки называется *фишингом*.

- Поздравлений с тем, что вы выиграли приз
- Запросов на "подтверждение" персональных данных учетной записи или паролей
- Требований о пересылке куда-либо фрагментов сообщения электронной почты
- Указаний на установку программного обеспечения, которое вы не искали и не запрашивали в явном виде
- Уведомлений об обнаруженных на вашем компьютере вирусах и других проблемах с системой безопасности

Пользователи, которые имеют основное представление об этих опасностях, чаще всего адекватно отреагируют, когда во всплывающем окне браузера появится сообщение, что они выиграли бесплатный MacBook и ссылка, на которой нужно щелкнуть.

Реализация политики безопасности

Файлы системного журнала могут легко доказать вам, что человек X сделал плохо человеку Y, но для суда это всего лишь слова. Защитите себя письменными заявлениями. Файлы системного журнала как правило содержат отметки времени, которые полезны, но их не всегда принимают в качестве доказательства, если на вашем компьютере системное время не синхронизировано с эталоном посредством протокола NTP (Network Time Protocol).

Лучше всего создать собственную политику безопасности и довести ее до сведения пользователей вашего сайта, чтобы затем можно было преследовать нарушителей в судебном порядке. Она должна содержать утверждение: "Несанкционированное использование вычислительных систем может повлечь не только нарушение организационной политики, но и нарушение законов штата и федеральных законов. Несанкционированное использование — это преступление, которое может повлечь уголовное наказание и гражданско-правовые санкции; оно будет преследоваться в правовом порядке в полном объеме".

Мы советуем показывать заставку, которая сообщит пользователям о ваших правилах контроля. Можно написать что-то вроде такого: "Действие может отслеживаться и фиксироваться в случае реального или предполагаемого нарушения правил безопасности".

Можно гарантировать, что пользователи увидят ваше уведомление, по крайней мере один раз, если вы включите его в файлы запуска, которые вы предоставляете новым пользователям. Если вы требуете, чтобы все попытки входа в систему через протокол SSH регистрировалось (а вы должны это требовать!), укажите соответствующие параметры конфигурации в файле `/etc/ssh/sshd_config`, чтобы при запуске клиент протокола SSH всегда показывал заставку.

Убедитесь, что, пользуясь своими учетными записями, пользователи соблюдают вашу письменную политику безопасности. Объясните, где они могут получить дополнительные копии документов политики и опубликуйте основные документы на соответствующей веб-странице. Также включите в нее информацию о том, что будет в случае несоблюдения этих правил (вплоть до удаления учетной записи и т.д.). Более важно, чтобы вы добросовестно уведомили пользователей об их обязанностях, а не просто составили юридически грамотное заявление.

Кроме заставки, целесообразно сделать так, чтобы пользователи в явном виде подписали соглашение о политике безопасности, прежде чем получить доступ к вашим системам. Это соглашение о приемлемом использовании должно быть разработано в сотрудничестве с вашим юридическим отделом. Если у вас нет подписанных соглашений

со служащими, соберите их. Сделайте подписание соглашения стандартной частью процесса приема на работу новых сотрудников.

Вы могли бы также периодически проводить семинары по вопросам безопасности. Это удобная возможность рассказать пользователям о важных проблемах, таких как фишинг, научить их правильно устанавливать программное обеспечение и пароли, а также кое-чему другому, что относится к вашей компетенции.

Контроль — это ответственность

Поставщики услуг (провайдеры Интернета, облачных вычислений и т.п.) обычно следуют правилам пользования сетью (appropriate use policy — AUP), которые диктуют стоящие над ними провайдеры и которые являются обязательными для их клиентов. Таким образом, вся ответственность за действия клиентов ложится на самих клиентов, а не на провайдеров. Целью такой стратегии является защита провайдеров от ответственности за рассылку спама и прочие незаконные действия, например хранение пользователями на своих компьютерах незаконного или защищенного авторскими правами материала. В каждом штате и каждой стране имеются свои законы на этот счет.

Ваши правила должны явно запрещать пользователям использовать ресурсы компании для незаконной деятельности. Однако на самом деле этого недостаточно — вы также должны дисциплинировать пользователей, обнаружив, что они осуществляют незаконные действия. Организации, которые знают о незаконных действиях, но не принимают меры для их пресечения, считаются соучастниками и могут преследоваться по закону. Нет ничего хуже несоблюдаемых или противоречащих друг другу правил, как с практической, так и юридической точки зрения.

Из-за риска стать соучастником незаконных действий пользователя некоторые организации ограничивают данные, которые они регистрируют, отрезок времени, в течение которого сохраняются файлы системного журнала, и объем информации о файлах системного журнала, хранящейся в виде резервных копий. Некоторые пакеты программ помогают выполнять эти правила, устанавливая уровни регистрации. Это позволяет системным администраторам устранять проблемы, не вторгаясь в частную жизнь пользователей. Тем не менее всегда следует знать, какой вид регистрации требуется по местным законам или по любым регулирующим стандартам, которые распространяются на вашу организацию.

Лицензии на программное обеспечение

Многие компании оплачивают меньшее количество копий программ, чем используют на самом деле. Если об этом становится известно, компания теряет гораздо больше, чем сэкономила на приобретении недостающего числа лицензий. Другие компании получают демонстрационную версию дорогого пакета и взламывают ее (меняют дату на компьютере, где-то находят и вводят лицензионный ключ и так далее), чтобы пакет продолжал работать по истечении демонстрационного срока. Как системный администратор должен реагировать на предложения нарушить лицензионное соглашение и установить нелицензионные копии программы на дополнительные компьютеры? Что он должен делать, если обнаруживается, что на обслуживаемых им компьютерах работает пиратское программное обеспечение? И как быть с условно-бесплатными программами, за которые так никогда и не заплатили?

Это очень сложный вопрос. К сожалению, начальство не всегда поддерживает администратора, предлагающего удалить нелицензионные копии программ либо оплатить их. А ведь часто именно системный администратор подписывает лицензионное соглашение, требующее удалить демонстрационные копии после определенной даты, тогда как решение их не удалять принимает руководитель.

Нам известно несколько случаев, когда непосредственный начальник системного администратора предлагал ему "не раскачивать лодку". Тогда администраторы написали докладную вышестоящему руководству, в которой указали количество лицензированных и используемых копий, а также процитировали лицензионное соглашение. В одном случае подход сработал, и уволен был непосредственный начальник системного администратора. Во втором случае уволиться пришлось системному администратору, потому что даже вышестоящее руководство отказалось действовать по закону. Что бы вы ни делали в такой ситуации, обязательно делайте это в письменном виде. Требуйте, чтобы ответы предоставлялись в письменном виде, а если не получится, документируйте полученные инструкции в виде коротких служебных записок и отправляйте их ответственному лицу.

31.10. Организации, конференции и другие ресурсы

Многие группы поддержки систем UNIX и Linux — и общего характера, и конкретных производителей — помогают устанавливать контакты с другими людьми, использующими то же программное обеспечение. В табл. 31.3 приведен короткий список таких организаций, хотя большинство национальных и региональных групп в этой таблице не упомянуто.

Таблица 31.3. Организации, ориентированные на системных администраторов систем UNIX и Linux

Название	Описание
FSF	Free Software Foundation — фонд свободного программного обеспечения, спонсор GNU
USENIX	Группа пользователей UNIX/LINUX, обсуждающих технические вопросы[a]
LOPSA	League of Professional System Administrators — лига профессиональных системных администраторов
SANS	Спонсор конференций по системному администрированию и безопасности
SAGE-AU	Австралийская гильдия системных администраторов; проводит ежегодные конференции в Австралии
Linux Foundation	Некоммерческий консорциум, ориентированный на стимулирование роста системы Linux
LinuxFest NorthWest	Новая и очень содержательная конференция

[a] Хорошо известная организация, создавшая специальную группу участников конференций LISA, которая была распущена в 2016 г.

Организация FSF (Free Software Foundation — фонд свободного программного обеспечения) является спонсором проекта GNU (GNU — аббревиатура от "GNU is Not Unix"; так называется проект по свободному распространению программного обеспечения). Под словосочетанием "свободное программное обеспечение" в названии этой организации подразумевается программное обеспечение, которое не имеет почти никаких

ограничений в использовании, а не бесплатное программное обеспечение. Также организация FSF является автором лицензии GPL, которая распространяется на большую часть систем UNIX и Linux.

Организация USENIX, представляющая собой союз пользователей Linux, UNIX и других операционных систем с открытым исходным кодом, каждый год проводит одну общую конференцию и несколько специализированных (небольших). Конференция Annual Technical Conference (ATC), на которой обсуждаются разнообразные темы, связанные с системами UNIX и Linux, представляет собой отличное место для укрепления связей с сообществом.

Лига профессиональных системных администраторов (League of Professional System Administrators — LOPSA) имеет сложную и запутанную историю. Изначально она была создана организацией USENIX и была предназначена для объединения системных организаторов в отдельную группу SAGE. К сожалению, отношения между организациями LOPSA и USENIX не сложились, и они стали существовать раздельно.

В настоящее время лига LOPSA организовывает учебные и образовательные программы по системному управлению сетями, такие как System Administrator Appreciation Day (День системного администратора) в последнюю пятницу июля. Традиционным подарком на этот праздник считается бутылка виски.

Институт SANS проводит конференции и семинары по вопросам безопасности, а также управляет отдельной учебной программой по сертификации Global Information Assurance Certification (GIAC). В рамках этой программы гильдия выдает сертификаты по разным направлениям, таким как системное администрирование, программирование, устранение сбоев и сетевая криминалистика (см. информацию на сайте `giac.org`).

Существует множество групп пользователей UNIX, Linux и других открытых систем. Одни из них связаны с организацией USENIX, а другие нет. Локальные группы обычно проводят регулярные встречи и рабочие семинары с местными или приезжими лекторами и часто организовывают банкеты до или после мероприятия. Это хороший способ поддерживать контакты с системными администраторами в своем регионе.

31.11. Литература

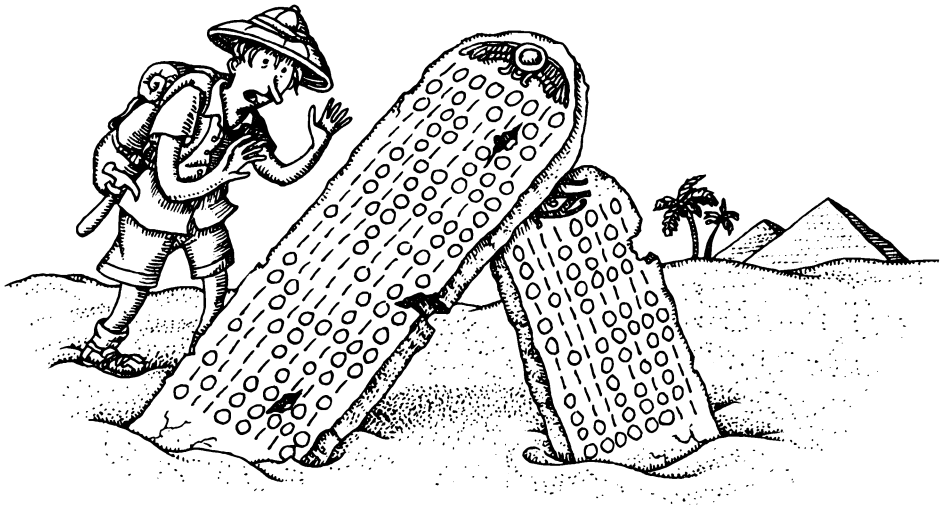
- Brooks, Frederick P., Jr. *The Mythical Man-Month: Essays on Software Engineering (2nd Edition)*. Reading, MA: Addison-Wesley, 1995.
- Kim, Gene, Kevin Behr, and George Spafford. *The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win (Revised Edition)*. Scottsdale, AZ: IT Revolution Press, 2014.
- Kim, Gene, et al. *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. Scottsdale, AZ: IT Revolution Press, 2016.
- Limoncelli, Thomas A. *Time Management for System Administrators*. Sebastopol, CA: O'Reilly Media, 2005.
- Machiavelli, Niccolo. *The Prince*. 1513. Доступен на сайте `gutenberg.org`.
- Morris, Kief. *Infrastructure as Code: Managing Servers in the Cloud*. Sebastopol, CA: O'Reilly Media, 2016. Эта книга представляет собой хорошо написанный и подробный обзор методологии DevOps, а также крупномасштабных инструментов для администрирования системы в облаке. Он включает в себя несколько специ-

фических особенностей управления конфигурацией как таковой, но это полезно для понимания того, как управление конфигурацией входит в более крупную схему DevOps и структурированное администрирование.

- Сайт `itl.nist.gov` является целевой страницей для лаборатории информационных технологий NIST и содержит множество информации о стандартах.
- Веб-сайт *Electronic Frontier Foundation*, `eff.org`, является отличным местом для поиска комментариев по актуальным вопросам конфиденциальности, криптографии и законодательства. Его всегда интересно читать.
- Институт SANS размещает коллекцию шаблонов политики безопасности по адресу `sans.org/resources/policies`.

Краткая история системного администрирования

Из записок доктора Питера Салуса (Peter H. Salus), историка, занимающегося вопросами развития технологий



В современную эпоху большинство людей имеют хотя бы минимальное представление о задачах, решаемых системными администраторами: они должны неустанно заботиться об удовлетворении потребностей пользователей и организаций, а также заниматься проектированием и реализацией устойчивой к ошибкам вычислительной среды, стараясь при этом поражать своих клиентов эффективными решениями. Несмотря на то что системные администраторы часто считаются низкооплачиваемыми и недооцененными, большинство пользователей могут хотя бы назвать имя своего местного системного администратора — причем во многих случаях гораздо быстрее, чем имя начальника их начальника.

Но так было не всегда. За последние 50 лет (и в течение более чем 30-летней истории этой книги) роль системного администратора постепенно эволюционировала вместе с операционными системами UNIX и Linux. Полное постижение предназначения системного администратора потребует понимания того, каким путем мы пришли к нынешнему состоянию, а также понимания ряда исторических факторов, которые сформировали наш ландшафт. Итак, окинем взглядом несколько чудесных десятилетий. Присоединяйтесь!

Рассвет компьютеризации: системные операторы (1952–1960)

Первый серийный коммерческий компьютер, IBM 701, был выпущен в 1952 году. До него все компьютеры выпускались в единичных экземплярах. В 1954 году модернизированная версия IBM 701 была анонсирована как IBM 704. Она имела оперативную память на магнитных сердечниках объемом 4 096 слов и включала три индексных регистра. Этот компьютер использовал 36-разрядные слова (в отличие от 18-разрядных слов у IBM 701) и выполнял арифметические операции с плавающей запятой. Он работал со скоростью 40 000 инструкций в секунду.

Однако компьютер IBM 704 был несовместим с IBM 701. Несмотря на то что его поставки должны были начаться к концу 1955 года, операторы (предшественники современных системных администраторов) восемнадцати существующих компьютеров IBM 701 уже нервничали: как им пережить эту "модернизацию" и на какие еще подводные камни они могут наткнуться?

Что касается самой компании IBM, то ее специалисты не имели решения проблемы модернизации и совместимости. В августе 1952 года для пользователей IBM 701 компания провела курсы повышения квалификации, но учебников не выпустила. Некоторые слушатели этих курсов продолжали неформально встречаться и обсуждать свой опыт использования системы. Компания IBM поощряла встречи операторов, поскольку совместное рассмотрение проблем помогало общими усилиями найти их решение. IBM финансировала встречи операторов и предоставила их участникам доступ к библиотеке, включающей 300 компьютерных программ. Эта группа по обмену информацией, именуемая SHARE, все еще существует (вот уже более 60-ти лет!)[1].

От узкой специализации к работе в режиме разделения времени (1961–1969)

Первые образцы компьютерного оборудования были достаточно громоздкими и чрезвычайно дорогими. Это наводило покупателей на мысль о том, что их компьютерные системы предназначены для решения одной-единственной конкретной задачи: только достаточно крупная и конкретная задача могла оправдать дороговизну и громоздкость такого компьютера.

Если компьютер был инструментом специального назначения (наподобие пилы), то персонал, который обслуживал компьютер, можно назвать операторами такой "пилы". К первым системным операторам относились скорее как к "тем, кто пилит древесину", чем как к "тем, кто обеспечивает все необходимое для строительства здания". Переход от системного оператора к системному администратору начался тогда, когда компьютеры превратились в универсальные (многоцелевые) инструменты. Основной причиной изменения точки зрения на компьютер стало то, что его начали использовать в режиме разделения времени.

Джон Маккарти (John McCarthy) начал подумывать о режиме разделения времени в средине 1950-х годов. Изначально это было только в Массачусетсском технологическом институте (MIT) в 1961-1962 годах, когда Джон Маккарти, Джек Деннис (Jack Dennis) и Фернандо Корбато (Fernando Corbato) начали серьезно говорить о том, что нужно разрешить "каждому пользователю компьютера вести себя так, будто он единственный его пользователь".

В 1964 году MIT, General Electric и Bell Labs объединили свои усилия по созданию проекта по построению претенциозной системы, работающей в режиме разделения времени. Эта система получила название Multics (Multiplexed Information и Computing Service). Пять лет спустя проект Multics превысил бюджет и безнадежно отстал от графика работ. В результате компания Bell Labs вышла из проекта.

Рождение UNIX (1969–1973)

Отказ компании Bell Labs от участия в проекте Multics оставил нескольких научных работников в Мюррей Хилл (штат Нью-Джерси) без работы. Трое из них — Кен Томпсон (Ken Thompson), Руд Кенедей (Rudd Canaday) и Деннис Ритчи (Dennis Ritchie) — заин-

[1] Несмотря на то что группа SHARE изначально спонсировалась производителями вычислительной техники, в настоящее время она является независимой организацией.

тересовались некоторыми аспектами проекта Multics, но не были в восторге от размера и сложности этой системы. Они не раз собирались вместе, чтобы обсудить принципы проектирования компьютерных систем. Компания Labs запустила систему Multics на своем компьютере GE-645, и Кен Томпсон продолжил работу над этим проектом "шутки ради." Дуг Макилрой (Doug McIlroy), руководитель этой группы, сказал: "Система Multics впервые заработала именно здесь. Вдохнуть в нее жизнь удалось трем нашим сотрудникам".

Летом 1969 года Томпсон на месяц стал холостяком, пока его жена, Бонни, взяв их годовалого сына, уехала повидать родственников на западное побережье. Томпсон вспоминал: "Я выделил по неделе на операционную систему, интерпретатор команд, редактор и ассемблер... И все эти компоненты были полностью переписаны в форме, которая придавала им вид операционной системы (с набором известных утилит, ассемблером, редактором, интерпретатором команд), которая практически полностью была самодостаточной и для своей работы больше не требовала GECOS[2]...По сути все это сделал один человек за один месяц".

Стив Борн (Steve Bourne), пришедший работать в Bell Labs в следующем году, так описывал "заброшенный" компьютер PDP-7, которым воспользовались Ритчи и Томпсон: "В PDP-7 были только ассемблер и загрузчик. На этом компьютере мог работать лишь один пользователь. Среда еще "отдавала сыростью", хотя части однопользовательской системы UNIX уже были "на подходе"... И вот были написаны ассемблер и зачаточное ядро операционной системы, которые еще требовали использования кросс-ассемблера для трансляции программы в системе GECOS и работы на машине PDP-7. Название UNICS (UNiplexed Information and Computing Service) для "новорожденной" операционной системы придумал заядлый остряк Питер Нейман (Peter Neumann) в 1970 году". Первоначально UNIX была однопользовательской системой, отсюда, по-видимому, и намек на "урезанный вариант Multics". И хотя некоторыми аспектами система UNICS/UNIX все же обязана своей предшественнице Multics, у них, по словам Денниса Ритчи, были "серьезные различия".

"Мы были немного подавлены большим менталитетом системы, — сказал он. — Кен хотел создать что-то простое. Вероятно, из-за того, что наши возможности были тогда довольно невелики. Мы могли получить доступ только к небольшим компьютерам, которые никак нельзя было сравнить с теми фантастическими аппаратными средствами, на которых работал Multics. Поэтому UNIX нельзя назвать ответным ударом, направленным против Multics... Операционная система Multics больше не существовала для нас, но нам нравилось ощущение интерактивной работы на компьютере, которое она предлагала пользователю. У Кена были некоторые идеи по созданию новой системы... И хотя Multics повлияла на подходы к реализации UNIX, это влияние не было определяющим".

"Игрушечная" система Кена и Денниса недолго оставалась "простой". К 1971 году в число ее пользовательских команд входили следующие: **as** (ассемблер), **cal** (простая утилита-календарь), **cat** (конкатенация и вывод), **chdir** (изменение рабочего каталога), **chmod** (изменение режима), **chown** (изменение владельца), **cmp** (сравнение двух файлов), **cp** (копирование файла), **date**, **dc** (калькулятор), **du** (отчет об использовании дискового пространства), **ed** (редактор) и еще два десятка других команд. Большинством из этих команд все пользуются и по сей день.

К февралю 1973 года можно было говорить о существовании 16 инсталляций UNIX, а также о двух больших нововведениях. Первое связано с "новым" языком программирования C, основанным на языке B, который сам представлял собой "урезанную" версию языка BCPL (Basic Combined Programming Language), созданного Мартином Ричардсом (Martin Richards). Второе нововведение — понятие канала.

[2]GECOS (General Electric Comprehensive Operating System) — операционная система, разработанная компанией General Electric в 1962 г.

Канал, попросту говоря, — это стандартный способ связи выходных данных одной программы с входными данными другой. Среда Dartmouth Time-Sharing System имела файлы связи, которые предвосхитили каналы, но их использование было гораздо более узким. Идея каналов (как обобщенного средства передачи данных) была предложена Дугом Макилроем, а реализована — Кеном Томпсоном благодаря настойчивости Макилроя. ("Это был один из немногочисленных случаев, когда я, по сути, осуществлял административное управление над UNIX", — сказал Дуг.)

«Легко сказать: `cat` — в `grep` — в... или `who` — в `cat` — в `grep` и так далее, — както заметил Макилрой. — Это легко сказать, и было ясно с самого начала, что именно это вы и хотели бы сделать. Но еще существуют все эти параметры... И время от времени я говорил: "Как бы нам сделать что-то вроде этого?" И вот в один прекрасный день я пришел с синтаксисом для командного языка, который развивался вместе с конвейерной пересылкой данных, и Кен сказал: "Я готов сделать это!"»

Применив множество вариантов, Томпсон обновил все UNIX-программы за одну ночь. Это было настоящее начало власти UNIX, возникшее не из отдельных программ, а из взаимосвязей между ними. Теперь у UNIX были собственные язык и основополагающие принципы:

- пишем программы, которые бы выполняли одну задачу, но выполняли ее хорошо;
- пишем программы, которые бы работали вместе;
- пишем программы, которые обрабатывали бы текстовые потоки как универсальный интерфейс.

Итак, можно было говорить о "рождении" операционной системы общего назначения с разделением времени, но пока лишь в "недрах" фирмы Bell Labs. ОС UNIX обещала простое и незаметное разделение компьютерных ресурсов между проектами, группами и организациями. Но прежде чем этот универсальный инструмент стал достоянием всего мира, он должен был вырваться из собственной колыбели и "размножиться".

UNIX становится знаменитой (1974–1990)

В октябре 1973 года международная научно-образовательная Ассоциация по вычислительной технике (Association for Computing Machinery — ACM) провела симпозиум на тему "Принципы построения операционных систем" (Symposium on Operating Systems Principles — SOSP) в аудитории Центра исследований Уотсона (T.J. Watson Research Center) компании IBM в Йорктаун Хайтс (Yorktown Heights), штат Нью-Йорк. Кен и Деннис в один прекрасный осенний день поехали в Долину Гудзона (Hudson Valley), чтобы представить на рассмотрение свой доклад. (Томпсон сделал настоящую презентацию.) В зале было около 200 человек, и доклад произвел фурор.

Шесть месяцев спустя количество инсталляций UNIX утроилось. Когда этот доклад был опубликован в июльском (1974) выпуске журнала *Communications of the ACM*, реакция читателей была просто ошеломляющей. Научно-исследовательские лаборатории и университеты увидели в разделяемых UNIX-системах потенциальное решение проблемы их постоянно возрастающих потребностей в компьютерных ресурсах.

В соответствии с положениями антимонопольного соглашения 1958 года, подписанного корпорацией AT&T (из недр которой выделилась фирма Bell Labs) с федеральным правительством, ее деятельность была весьма ограничена: она не имела право заниматься рекламой, продажей и сопровождением компьютерных продуктов. Компания Bell Labs должна была давать разрешение на использование другими своей технологии. Тем не менее система UNIX завоевала популярность в мире, особенно среди телефон-

ных компаний. Отвечая на многочисленные просьбы, Кен Томпсон стал распространять копии исходного кода UNIX, и, по легенде, каждый пакет включал его личную записку, подписанную "love, ken" (с любовью, Кен).

Одним из тех, кто получил копию системы от Кена, был профессор Роберт Фабри (Robert Fabry) Калифорнийского университета в Беркли. Так, к январю 1974 года зерно Berkeley UNIX попало в плодородную почву.

Другие ученые, работающие в области компьютерных наук, также проявили интерес к UNIX. В 1976 году Джон Лайонс (John Lions), преподаватель факультета компьютерных наук в Университете Нового Южного Уэльса в Австралии, опубликовал подробные комментарии относительно ядра версии V6. Эта работа стала первым серьезным пакетом документации по системе UNIX и помогла другим понять и развить работу Кена и Денниса.

Студенты Калифорнийского университета в Беркли поработали над версией UNIX, полученной из Bell Labs, и изменили ее так, чтобы она отвечала их требованиям. Первый ленточный дистрибутив программы Беркли (1st Berkeley Software Distribution — 1BSD) содержал систему Pascal для Unix и текстовый редактор `vi` для компьютера PDP-11. Этот дистрибутив подготовил аспирант Билл Джой (Bill Joy). Второй выпуск (2BSD) вышел в следующем году, а третий (3BSD) — в качестве первого дистрибутива программы Беркли для мини-компьютера VAX, выпускаемого корпорацией DEC, увидел свет в конце 1979 года.

В 1980 году профессор Роберт Фабри заключил контракт с управлением перспективных исследовательских программ в области обороны (Defense Advanced Research Project Agency — DARPA) на продолжение разработки системы UNIX. Результатом этого соглашения стало образование в Беркли исследовательской группы по компьютерным системам (Computer Systems Research Group — CSRG). В конце следующего года вышла четвертая версия системы — 4BSD. Она приобрела довольно широкую популярность, в основном потому, что это была единственная версия UNIX, которая работала на DEC VAX 11/750, весьма распространенной в то время компьютерной платформе. Еще одно крупное усовершенствование, отличавшее выпуск 4BSD, состояло в использовании сокетов TCP/IP, обобщенной сетевой абстракции, которая в будущем породила Интернет и сейчас используется многими современными операционными системами. К середине 1980-х годов в большинстве крупных университетов и научно-исследовательских институтов была установлена хотя бы одна система UNIX.

В феврале 1982 года Билл Джой основал компанию Sun Microsystems (сейчас это часть компании Oracle America) и положил в основу операционной системы SunOS изобретенную им систему 4.2BSD. В 1983 году по решению суда началась ликвидация корпорации AT&T, и одним непредвиденным побочным эффектом этой ликвидации было то, что AT&T отныне могла свободно продавать систему UNIX как программный продукт. В результате увидела свет версия AT&T UNIX System V — общеизвестная, хотя и несколько неудобная коммерческая реализация системы UNIX.

Теперь, когда Berkeley, AT&T, Sun и другие дистрибутивы UNIX стали доступны для широкого круга организаций, был заложен фундамент для общей компьютерной инфраструктуры, построенной на технологии UNIX. Систему, которую использовали в области астрономии для вычисления звездных расстояний, можно было успешно применять в математике для вычисления множеств Мандельброта. И та же система одновременно обеспечивала доставку электронной почты для всего университета.

Эра системных администраторов

Управление компьютерными системами общего назначения требовало другого уровня знаний, чем двадцать лет назад. Ушли в прошлое дни, когда системный оператор обслуживал единственную компьютерную систему, предназначенную для выполнения

специализированной задачи. Системный администратор в начале 1980-х годов стал настоящим "хозяином положения", который знает, как настроить систему UNIX, чтобы она удовлетворяла требованиям самых разных приложений и пользователей.

Поскольку система UNIX была весьма популярной в университетах и множество студентов горело желанием овладеть новейшими технологиями, именно университеты лидировали в создании организованных групп системных администраторов. Такие учебные заведения, как Университет Пердью, Университет штата Юта, Университет штата Колорадо, Университет штата Мэриленд и Университет штата Нью-Йорк (SUNY) в г. Буффало, стали "рассадниками" специалистов в области системного администрирования.

Кроме того, системные администраторы разработали собственные процессы, стандарты, инструкции и инструменты (например, **sudo**). Большинство из этих продуктов родилось из необходимости, поскольку без них системы работали нестабильно, что, естественно, вызывало недовольство пользователей.

Эви Немет (Evi Nemeth) приобрела известность как "мать системного администрирования" тем, что приглашала на работу в качестве системных администраторов студентов старших курсов Инженерного колледжа (Engineering College) при Университете штата Колорадо. Ее близкие связи с сотрудниками Калифорнийского университета в Беркли, Университета штата Юта и SUNY в г. Буффало позволили создать сообщество экспертов по вопросам системного администрирования, которые делились советами и инструментами. Ее команду часто называли *munchkins*, т.е. "переработчики информации" или "рабы Эви". Члены этой команды посещали различные конференции (в том числе и спонсируемые организацией USENIX) и работали там в качестве служебного персонала в обмен на возможность получать информацию, излагаемую участниками этих конференций.

Уже стало ясно, что системные администраторы должны были стать "мастерами на все руки". В 1980-х годах обычное утро системного администратора могло начаться с использования инструмента для накрутки провода для починки поврежденного переключателя на задней панели мини-компьютера VAX. Днем он мог заниматься очищением лазерного принтера первого поколения от рассыпавшегося тонера. Его обеденный перерыв мог быть потрачен на помощь какому-нибудь студенту отладить новый драйвер ядра, а вечерние часы могли быть заполнены записью информации на архивные ленты и уговорами пользователей почистить свои персональные каталоги, чтобы освободить пространство в файловой системе. Системный администратор был, без преувеличения, бескомпромиссным ангелом-хранителем, который должен был решить любую проблему.

1980-е годы можно было назвать временем ненадежного оборудования. Центральные процессоры были сделаны не из одной кремниевой микросхемы, а из нескольких сотен чипов, каждый из которых мог в любую минуту отказать. И именно системный администратор должен был быстро отыскать неисправный элемент и заменить его работающим. К сожалению, в то время почтовая служба Federal Express еще не работала, и поэтому элементы для замены нужно было находить самим, что зачастую было очень непросто.

Однажды наш любимый компьютер VAX 11/780 перестал работать, в результате чего весь университет остался без электронной почты. Мы знали, что недалеко от нас есть фирма, которая собирала компьютеры VAX, предназначенные "для научных исследований" для отправки в Советский Союз (то было время "холодной войны"). Практически без всякой надежды на успех мы заявились на склад с большой суммой наличных в кармане, и после часа переговоров мы все-таки получили необходимую печатную плату. Кое-кто отметил, что в то время в Боулдере (штат Колорадо) было легче достать наркотики, чем запчасти к VAX.

Документация по системному администрированию и обучение

По мере того как отдельные компьютерные специалисты стали считать себя системными администраторами — и причем стало ясно, что такая специализация может обеспечить достойную жизнь, — потребности в документации и соответствующем обучении ощутимо возросли. "Идя навстречу пожеланиям трудящихся", Тим О'Рейлли (Tim O'Reilly) и его команда (именуемая ранее *O'Reilly and Associates*, а теперь *O'Reilly Media*) начали публиковать документацию по системе UNIX, написанную простым языком и основанную на реальном опыте.

В качестве посредника межличностного взаимодействия ассоциация USENIX провела свою первую конференцию, посвященную системному администрированию, в 1987 году. Эта конференция (Large Installation System Administration — LISA) охватила, в основном, западное побережье. Три года спустя был учрежден институт SANS (SysAdmin, Audit, Network, Security) для удовлетворения потребностей специалистов восточного побережья. В настоящее время конференции LISA и SANS обслуживают всю территорию США и до сих пор на достаточно высоком уровне.

В 1989 году мы опубликовали первое издание этой книги, имевшей тогда название *UNIX System Administration Handbook*. Она была быстро раскуплена, в основном, из-за отсутствия альтернативы. В то время наш издатель был настолько далек от системы UNIX, что в их редакции все вхождения строки "etc" были заменены строкой "and so on", в результате чего появились такие имена файлов, как `/and so on/passwd`. Мы воспользовались создавшейся ситуацией, чтобы взять под полный контроль все содержимое книги от корки до корки, и сейчас, надо признать, наш издатель стал более осведомленным в вопросах UNIX. Наше 20-летнее сотрудничество с этим издателем дало пищу для других интересных историй, но мы их опустим, дабы не испортить наши вполне дружеские отношения.

UNIX при смерти. Рождение Linux (1991–1995)

К концу 1990 года казалось, что UNIX стремительно движется к мировому господству. Бесспорно, именно эту операционную систему выбирали как для ведения бизнеса (например, Taco Bell и McDonald's), так и для исследовательских и научных расчетов. Группа CSRG (Computer Systems Research Group — исследовательская группа по компьютерным системам) в Беркли, состоящая из Кирка Маккузика (Kirk McKusick), Майка Карелса (Mike Karels), Кейта Бостика (Keith Bostic) и многих других, как раз выпустила версию 4.3BSD-Reno, основанную на выпуске 4.3, в которую была добавлена поддержка для процессора CCI Power 6/32 (с кодовым именем "Tahoe").

Коммерческие выпуски UNIX (например, SunOS) также пользовались успехом, который, отчасти, им обеспечили появление Интернета и первые шаги в направлении электронной торговли. Оборудование персонального компьютера стало предметом широкого потребления. Оно уже было относительно надежным, недорогим и обеспечивало довольно высокую производительность. И хотя версии системы UNIX, запускаемые на персональных компьютерах, действительно существовали, все они были коммерческими, причем с закрытым исходным кодом. Назрела ситуация для появления UNIX для персональных компьютерах с открытым исходным кодом.

В 1991 году группа разработчиков, трудившихся над выпусками BSD, — Донн Сили (Donn Seeley), Майк Карелс, Билл Джолитц (Bill Jolitz) и Трент Р. Хейн (Trent R. Hein) — вместе с другими приверженцами BSD основали компанию Berkeley Software Design, Inc. (BSDI). Под руководством Роба Колстада (Rob Kolstad) компания BSDI предоставляла исполняемые файлы и исходный код для полностью функциональной коммерческой версии BSD UNIX на персональных компьютерах. Среди прочего, этот проект доказал,

что для создания массовых производственных сред можно использовать недорогие персональные компьютеры. Компания BSDI продемонстрировала взрывоподобный рост прибыли на заре развития Интернета, поскольку именно ее операционную систему выбирали первые провайдеры услуг Интернета (Internet service providers — ISP).

Пытаясь снова упрятать джинна, который выскочил из бутылки в 1973 году, корпорация AT&T в 1992 году начала судебный процесс против компании BSDI и членов правления университета Калифорнии (Regents of the University of California), заявив о копировании кода и воровстве производственных секретов. Юристам компании AT&T потребовалось более двух лет, чтобы идентифицировать проблемный код. В результате судебного разбирательства из кода BSD было удалено три файла (из более чем 18 000).

К сожалению, этот двухлетний период неопределенности оказал негативное воздействие на весь мир UNIX, операционную систему BSD и подобные ей не-BSD-версии. Многие компании перешли на использование Microsoft Windows, испугавшись, что они могут оказаться во власти компании AT&T, которая практически "задушила в объятиях" свое дитя. К тому времени, когда дым сражений развеялся, оказалось, что компании BSDI и CSRG "смертельно ранены". Эра BSD подходила к концу. Тем временем Линус Торвальдс (Linus Torvalds), студент колледжа в Хельсинки, наигравшись с Minix — свободной Unix-подобной микроядерной операционной системой, распространяемой по лицензии BSD, — начал писать собственную систему-клон UNIX[3]. К 1992 году появилось множество дистрибутивов Linux (включая SuSE и Yggdrasil Linux). В 1994 году мир узнал о создании систем Red Hat и Linux Pro.

Феноменальный успех Linux основан на многих факторах. Мощная поддержка всех, кому понравилась эта система, и обширный список программ из архива GNU сделали Linux неуязвимой системой. Она прекрасно работает в любых производственных средах, и поговаривают, что на основе Linux можно построить более надежную и более производительную систему, чем на основе любой другой операционной системы. Интересно также отметить, что отчасти своим успехом Linux обязана блестящей возможности, предоставленной ей действиями компании AT&T против BSDI и университета Калифорнии в Беркли. Тот неуместный судебный процесс вселил страх в сердца приверженцев UNIX прямо на заре электронной торговли и в начале эры Интернета.

Но не все ли равно теперь? Главное, что в результате всех этих перипетий осталась огромная потребность в системных администраторах. Багаж знаний и опыт, накопленный системными администраторами при обслуживании систем UNIX, полностью применимы к Linux, и большинство "UNIX-лоцманов" заботливо продолжали вести своих пользователей через бурные моря 1990-х. Возьмите себе на заметку: хороший системный администратор должен оставаться спокойным во время любого шторма.

Мир Windows (1996–1999)

В 1993 году компания Microsoft выпустила ОС Windows NT. Эта "серверная" версия Windows, которая имела популярный пользовательский интерфейс, имела значительный эффект, причем как раз тогда, когда корпорация AT&T старалась убедить всех в том, что она больше никому не позволит обманывать себя в лицензионных вопросах. Как следствие, многие организации приняли Windows в качестве предпочтительной платформы для совместных вычислений. Это был конец 1990-х. Вне всякого сомнения, платформа Microsoft прошла долгий путь развития, и для некоторых организаций это был наилучший выбор.

[3]ОС Minix была разработана Эндрю Таненбаумом (Andrew S. Tanenbaum), профессором Амстердамского свободного университета.

К сожалению, сначала администраторы UNIX, Linux и Windows использовали в своей работе конкурентные подходы, соперничая за "место под солнцем" путем противопоставления аргументов из рекламы пива: "отличный вкус" против "меньшего объема"[4]. Многие администраторы систем UNIX и Linux начали срочно изучать Windows, убежденные в том, что в противном случае им придется "положить зубы на полку". А на горизонте уже показалась Windows 2000. С приближением "миллениума" будущее UNIX выглядело все более мрачным.

Расцвет UNIX и Linux (2000–2009)

С приходом Интернета все старались понять, что "настоящее", а что — всего лишь мираж. Когда страсти "от новизны" немного улеглись, стало ясно, что многие организации с успешными техническими стратегиями использовали UNIX или Linux вместе с Windows, а не только что-то одно. Конкурентной войны больше не было.

Оценки экспертов, использующих методику определения суммарной стоимости владения (соотношения цены-качества, total cost of ownership — TCO), показали, что этот показатель для сервера Linux был значительно ниже аналогичного показателя для сервера Windows. После экономического кризиса 2008 г. показатель TCO стал еще важнее. Мир снова разворачивается лицом в версиям операционных систем UNIX и Linux с открытым кодом.

Системы UNIX и Linux в гипермасштабируемом облаке (2010– настоящее время)

Варианты Linux и UNIX для персональных компьютеров, такие как FreeBSD, продолжают расширять свою долю на рынке, при этом Linux — единственная операционная система, доля рынка которой растет на серверах. Кстати, текущая полномасштабная операционная система компании Apple под названием macOS, также является вариантом UNIX.[5]

Значительная часть недавнего роста рыночной доли операционных систем UNIX и Linux объясняется виртуализацией и облачными вычислениями. Более подробную информацию об этих технологиях см. в главах 9 и 24.

Возможность создания виртуальной инфраструктуры (и целых виртуальных центров обработки данных) путем использования вызовов API коренным образом изменила тенденции. Ушла в прошлое эпоха ручного управления физическими серверами. Масштабирование инфраструктуры больше не требует закупок новых серверов в коробках. Благодаря таким службам, как Google GCP, Amazon AWS и Microsoft Azure, наступила эпоха гипермасштабируемого облака. Стандартизация, инструменты и автоматизация — это не просто новинки, а неотъемлемые атрибуты каждой вычислительной среды.

Сегодня грамотное управление парками серверов требует обширных знаний и навыков. Системные администраторы должны быть дисциплинированными профессионалами. Они должны знать, как создавать и масштабировать инфраструктуру, как работать совместно со сверстниками в среде DevOps, как программировать простые сценарии автоматизации и мониторинга и как сохранять спокойствие, когда одновременно выходит из строя тысяча серверов.[6]

---

[4]Для ясности: в Windows действительно "меньше объема".

[5]Даже смартфоны iPhone компании Apple можно назвать кузенами UNIX, а операционная система Android компании Google основана на ядре Linux.

[6]Одно не изменилось: виски по-прежнему остается лучшим другом многих системных администраторов.

Завтрашний день UNIX и Linux

Куда мы направляемся дальше? Экономная модульная парадигма, которая так хорошо служила в сфере UNIX последние несколько десятилетий, также является одной из основополагающих технологий Интернета вещей (IoT). По данным Института Брукингса (Brookings Institution), к 2020 г. будет существовать 50 миллиардов небольших распределенных устройств, образующих Интернет вещей (см. `brook.gs/2bNwbya`).

Заманчиво думать об этих устройствах так же, как мы думали о несетевых бытовых приборах (например, тостерах или блендерах) прошлых лет: подключите их, используйте в течение нескольких лет и, если они сломаются, выбросьте их на свалку.[7] Им не нужно управление или услуги системного администратора, не так ли?

На самом деле ничто не могло быть дальше от истины. Многие из этих устройств обрабатывают конфиденциальные данные (например, звуковые, передаваемые из микрофона в вашей гостиной) или выполняют критически важные функции, такие как управление температурой вашего дома.

Некоторые из этих устройств запускают встроенное программное обеспечение, полученное из мира OSS. Но независимо от того, что находится внутри самих устройств, большинство из них отчитывается перед материнским кораблем, находящимся в облаке, которое работает, вы уже догадались, под управлением систем UNIX или Linux. На ранних, захватнических этапах развития рынка многие устройства уже были развернуты без особого внимания к вопросам безопасности или будущих экологических последствий.

Интернет вещей не ограничивается потребительским рынком. Современные коммерческие здания пронизаны сетевыми устройствами и датчиками, например для освещения, вентиляции и отопления, обеспечения физической безопасности и видеонаблюдения. Эти устройства часто появляются в сети без координации с IT-отделами или подразделениями информационной безопасности. Затем о них забывают без какого-либо плана постоянного управления, исправления или мониторинга.

Размер не имеет значения, когда дело доходит до сетевых систем. Системным администраторам необходимо защищать безопасность, производительность и доступность устройств Интернета вещей (и их поддерживающей инфраструктуры) независимо от размера, местоположения или функции.

Системные администраторы заботятся о целостности компьютерной инфраструктуры, решают сложнейшие проблемы эффективности и масштабируемости систем и снабжают квалифицированными рекомендациями в области компьютерных технологий как рядовых пользователей, так и руководителей организаций.

Мы системные администраторы! Без нас — никак!

Литература

- McKusick, Marshall Kirk, Keith Bostic, Michael J. Karels and John S. Quarterman. *The Design and Implementation of the 4.4BSD Operating System (2nd Edition)*. Reading, MA: Addison-Wesley, 1996.
- Salus, Peter H. *A Quarter Century of UNIX*. Reading, MA: Addison-Wesley, 1994.
- Salus, Peter H. *Casting the Net: From ARPANET to Internet and Beyond*. Reading, MA: Addison-Wesley, 1995.
- Salus, Peter H. *The Daemon, the Gnu, and the Penguin*. Marysville, WA: Reed Media Services, 2008. Эта книга также опубликована на сайте `www.groklaw.net`.

[7] Не делайте этого. Все должно подвергаться переработке.

Предметный указатель

B

BIND, пакет
  журнальная регистрация, 584
    каналы, 586
    категории сообщений, 586
  конфигурация, 561
  параметры конфигурации, 545
    allow-query, 549
    allow-transfer, 549
    also-notify, 547
    blackhole, 549
    directory, 546
    forward, 549
    forwarders, 549
    notify, 547
    recursion, 547
  сообщения об ошибках, 587
BIOS, 71

C

CUEFI, 71

D

DNS
  база данных
    директива
      $GENERATE, 530
      $INCLUDE, 530
      $ORIGIN, 530
      $TTL, 530; 532
    записи о ресурсах, 524; 531
    запись
      A, 537
      CNAME, 540
      KEY, 571
      MX, 539
      NS, 536
      PTR, 538
      SOA, 534
      SRV, 541
      TXT, 542
    класс записи, 533
    обновление, 565
  делегирование, 525
  динамические обновления, 554
  зона, 534
    порядковый номер, 535
    создание, 554
    цифровая подпись, 573
  кеширование, 526
  конфигурирование, 435
  некорректное делегирование, 522; 591
  обратное преобразование, 538
  передача зоны, 523
  прямое преобразование, 537
  сервер имен, 522
    авторитативный,
    авторитетный, 523; 536
    главный, 523; 554
    кэширующий, 523
    нерекурсивный, 524
    переадресующий, 549; 556
    подчиненный, 523; 555
    рекурсивный, 524
  сигнатуры транзакций, 570
  спецификация
    DNSSEC, 573
    TKEY, 570
    TSIG, 570
  файл подсказок, 554; 556

E

EFI, 71
Ethernet, 479
  инкапсуляция пакетов, 403
  параметр MTU, 404
  соединение сегментов, 483
  спецификации, 478
  топология, 479
  фрейм, 403
    стандарты, 404
  широковещательный домен, 480
  MAC-адрес, 405
Exim, почтовый агент, 657
  аутентификация, 667
  загрузка, 659
  конфигурация, 661
  макрос, 664
  маршрутизатор
    accept, 669
    dnslookup, 670

manualroute, 670
redirect, 670
регистрация, 673
список ACL, 664
транспортный механизм, 672
appendfile, 672
smtp, 672
утилита
exicyclog, 660
exigrep, 660
exilog, 660
exim_checkaccess, 660
exim_dbmbuild, 660
exim_dumpdb, 660
exim_fixdb, 660
exim_lock, 660
eximon, 660
eximstats, 660
exim_tidydb, 660
exinext, 660
exipick, 660
exiqgrep, 660
exiqsumm, 660
exiwhat, 660
фильтрация пользователей, 671

H

HTTP-запрос, 692
HTTP-метод
DELETE, 692
GET, 692
HEAD, 692
OPTIONS, 692
POST, 692
PUT, 692
HTTP-ответ, 692
HTTP-сообщение
заголовок, 693
тело, 693

M

Man-страница
ascii, 164
autofs(5), 832
autofs(8), 832
auto.master(5), 832
automount(1M), 831
capabilities(7), 120
chmod, 171
config(5), 370
config(8), 370
exports(5), 820
hier, 162
mount.cifs, 840
mount_smbfs, 840
proc, 140
procfs, 141
MBR, 72

N

NAT, 413
IP-маскирование, 441
NFS, 805
пользователь
nobody, 814
root, 814

P

PAM, 607
модуль
include, 609
optional, 609
required, 609
requisite, 609
sufficient, 609
Posrfix, почтовый агент
архитектура, 675
программа
cleanup, 676
local, 677
pickup, 676
pipe, 677
postdrop, 677
qmgr, 676
trivial-rewrite, 677
virtual, 677
таблица поиска, 679
утилита
mailq, 677
newalises, 677
postalias, 677
postcat, 677
postconf, 677
postfix, 677
postmap, 677
postsuper, 677
sendmail, 677

файл
main.cf, 678
master.cf, 678
аутентификация, 685
механизм SMTP AUTH, 686
виртуальные домены, 682
программа
local, 681
управление доступом, 683
шифрование, 685

R

Red Hat
сетевое конфигурирование, 436

S

Sendmail, почтовый агент, 630
база доступа, 646; 647
безопасность, 649
библиотека libmilter, 646
журнальная регистрация, 656
изменение корневого каталога, 653
макрос
DOMAIN, 639
FEATURE, 639
MAILER, 639
MAIL_HUB, 642
MASQUERADE_AS, 642
OSTYPE, 639
RELAY_DOMAIN, 647
RELAY_DOMAIN_FILE, 647
SMART_HOST, 642
направление почты в программу, 628; 651
направление почты в файл, 628; 651
опции безопасности, 652
права доступа, 651
производительность, 655
псевдонимы
хешированная база данных, 628
ретрансляция, 646
спам
дроссели, 646
черные списки, 646
специальные пользователи, 650
списки рассылки, 627
средство
blacklist_recipients, 647
local_lmtp, 651
relay_entire_domain, 647
relay_hosts_only, 647
smrsh, 651
утилита
smrsh, 652
vacation, 652
функциональная возможность
access_db, 640
always_add_domain, 640
ldap_routing, 641
redirect, 640
use_cw_file, 639
virtusertable, 641

T

TCP/IP
IP-адрес, 406; 409
альтернативный, 408
в стандарте IPv6, 415
выделение, 413
групповой, 408
классы, 409
маска подсети, 410
назначение компьютеру, 430
направленный, 408
подмена, 427
сетевой, 411
типы, 408
частный, 413
широковещательный, 408; 411
канальный уровень
параметр MTU, 404
пакет, 403
адресация, 405
порт, 407
сегмент, 403
сетевая модель, 400
фрейм, 403

U

UEFI, 69
URL-адрес, 690

W

Web-хостинг, 689

А

Автоматизация, 310
Авторизация, 310

Агент
пользовательский, 690
MSA, 615
Алгоритм
Comletely Fair Queuing, 1088
Deadline, 1088
LRU, 1082
NOOP, 1088
Анализатор пакетов, 492
Анализ статического кода, 963
Архитектура, 202
Атака
DDoS, 989
Аутентификатор, 667
Аутентификация, 995
многофакторная, 995

Б

Базовая аутентификация HTTP, 715
Балансирование
нагрузки, 527
Балансировщик нагрузки, 699
HAProxy, 724
NGINX, 723
Библиотека
libpcap, 454
Бит
выполнения, 168
дополнительный, 169
записи, 168
поиска, 168
режима, 167
чтения, 168
setgid, 129; 168
setuid, 129; 168
Брандмауэр, 1031
безопасность, 1032
фильтрация пакетов
на уровне служб, 1031; 1032

В

Веб-сервер
Apache httpd, 711
NGINX, 718
Веб-хостинг, 689
облачный, 708
статический, 710
Ветвление, 270
Виртуализация, 910
Виртуальная частная сеть, 429; 1033; 1034
Виртуальное частное облако VPC, 465
Виртуальный частный сервер, 308
Вирус, 993
Восстановление
облачных систем, 100
Выключение
облачных систем, 97
физических систем, 97
Выпуск, 201
Выражение
регулярное, 229

Г

Гипервизор
bhyve, 919
ESXi, 919
VirtualBox, 919
Группа
томов, 754
Группа безопасности, 468

Д

Дамп памяти, 131
Делегирование
некорректное, 591
Демон
инициализации, 80
службы, 91
управления системой, 79
acpid, 330
amd, 827
auditd, 326
automountd, 827; 828
autounmountd, 827
cupsd, 385; 388
devd, 357; 362
devfs, 362
dhcpd, 425
dhcrelay, 426
httpd, 325; 342
inetd, 88
init, 80; 130
journald, 82; 94
logind, 82
mountd, 816
mysqld, 325
named, 557
безопасность, 570

журнальная регистрация, 584
начальный каталог, 546
обработка запроса, 527
уровни отладки, 589
флаг -d, 589
networkd, 82
nfsd, 816
nginx, 91
nmbd, 834
routed, 511
rsyslogd, 330; 945
salt-master, 884
slapd, 599
slurpd, 599
smartd, 753
smbd, 834
ssh-agent, 1024
sshd, 96; 1019
sssd, 606
statd, 812
sysklogd, 334
syslog, 322
systemd, 70
udevd, 357
Дескриптор
файла, 223
Динамическая миграция, 913
Диспетчеризация билетов, 1116
Диспетчер печати, 383
Документация
серия RFC, 399
man-страницы, 55
Домен, 522
Доступность, 986
Драйвер
устройства, 165
хранилища, 937
md, 772
Драйвер устройства, 354
Дроплет, 317

Ж

Жесткий диск, 736
Журнал
авторизации, 324
systemd, 94; 326

З

Загрузка, 69
в однопользовательском режиме, 77
FreeBSD, 77
Загрузочный сектор, 72
Загрузчик
загрузчик, 72
первичный, 72
GRUB, 74
loader, 78
Запись
главная загрузочная, 758
управления доступом, 175
Запись DNS
базовая, 533
безопасности, 533
вспомогательная, 533
зонная, 533
Запрос на включение, 270
Захват, 245
Зеркальное отражение, 768
Зона, 522
Зона доступности, 307

И

Идентификатор
группы, 129
группы, GID, 278
объектный, 1066
пользователя, 129
текущий, 129
пользователя, UID, 278
процесса, 128; 159
родительского процесса, 128
UID, 106
Идентификация, 310
Имя
пользователя, 275
регистрационное, 275
Инкапсуляция, 403
Инсталляция
CentOS, 188
Debian, 188
FreeBSD, 188
Red Hat, 188
Ubuntu, 188
Инструмент DNSSEC
ldns, 581
Sparta, 582
Vantages, 583
Интернет
документация, 399
провайдеры, 398

число пользователей, 397
Интернет-шлюз, 467
Интерфейс обратной связи, 409; 420; 501
Инфраструктура
как код, 300
как услуга, 304

К

Канал
именованный, 166
STDIN, 227
STDOUT, 227
Каталог, 164
домашний, 279
корневой, 157
сетевых сценариев, 93
формул, 897
/bin, 160; 161
/bin/false, 116
/bin/nologin, 116
/bin/sh, 231
/boot, 161
/boot/grub, 75
cf/cf, 635
cf/feature, 635
cf/ostype, 635
/dev, 160; 161; 354
.do-not-touch, 233
/etc, 160; 161; 425
/etc/ansible/group_vars, 870
/etc/ansible/host_vars, 870
/etc/auto_master, 828
/etc/auto.master, 828
/etc/cups, 388
/etc/cups/lpoptions, 386
/etc/default/grub, 75
/etc/dhcp3, 425
/etc/logrotate.d, 346
/etc/security, 288
/etc/selinux, 124
/etc/skel, 288
/etc/ssh, 1021
/etc/systemd/system, 84
/etc/udev/rules.d, 359
/etc/xen, 915
/ets/sysconfig, 438
/home, 161
/lib, 160; 161
/lib/modules/, 371
/lib/systemd/system, 84
/lib/udev/rules.d, 359
/media, 161
/mnt, 161
/mnt/data, 934
/opt, 161
/proc, 140; 161
/proc/sys, 364
/proc/sys/net/ipv4/neigh, 442
/proc/sys/net/ipv6/neigh, 442
roles, 879
/root, 161
/run/systemd/system, 84
/sbin, 160; 161
/srv/pillar, 883
/srv/salt, 883
/srv/salt/pillar, 884
/srv/salt/salt, 884
/stand, 161
/sys, 357
/tmp, 160; 161; 757
/usr, 160; 161; 228
/usr/bin, 161
/usr/bin/env, 231
/usr/include, 161
/usr/lib, 161
/usr/lib/systemd/system, 84
/usr/local, 161; 209
/usr/local/etc, 209; 865
/usr/local/etc/skel, 288
/usr/sbin, 161
/usr/share, 161
/usr/share/man, 161
/usr/share/sendmail, 631
/usr/src, 161; 366
/usr/tmp, 161
/var, 160; 162; 757
/var/adm, 162; 323
/var/lib/dhclient, 426
/var/lib/dhcp, 426
/var/lib/docker, 927
/var/log, 162; 323
/var/log/journal/, 327
/var/log/syslog, 323
/var/spool, 162
/var/tmp, 162
/vm/chef.img, 916
Команда
фильтрации, 227
accept, 392

adduser, 222; 292
ansible, 865
ansible-playbook, 857; 865
ansible-vault, 865
arp, 422
basha, 288
cancel, 392
cat, 140
chef-solo, 858
chfn, 279
chgrp, 172
chmod, 167; 170
chown, 172; 627
chsh, 222; 280
control, 362
cp, 164
csh/tcsh, 288
cups-config, 391
cupsdisable, 391
cupsenable, 391
cut, 227
df, 781; 824
dmesg, 360
dmidecode, 1079
dnskeygen, 571
dnspktflow, 582
dnssec-keygen, 571; 577
dnssec-signzone, 578
docker, 927
donuts, 582
donutsd, 582
dpkg, 199
echo, 234
emacs, 288
ethtool, 440
exit, 108
find, 224
format, 750
fsck, 99; 158; 779
fuser, 159
gem, 261
git, 268
GNOME, 288
gpasswd, 284
grep, 229
GRUB
  boot, 76
  help, 76
  linux, 76
  reboot, 76
  search, 76
  usb, 76
halt, 97
hdparm, 752
head, 229
hostname, 431
ifconfig, 410; 432; 437; 444; 488
ifdown, 438
iostat, 1085
ip neigh show, 422
ip neight, 422
iwconfig, 488
iwlist, 488
journalctl, 94; 360
KDE, 288
kdestroy, 605
kill, 131; 133
killall, 134
kinit, 605
knife, 858
ldapsearch, 602
ldns-chaos, 582
ldns-compare-zones, 582
ldns-dpa, 582
ldns-key2ds, 582
ldns-keyfetcher, 582
ldns-keygen, 582
ldns-notify, 582
ldns-nsec3-hash, 582
ldns-read-zone, 582
ldns-revoke, 582
ldns-rrsig, 582
ldns-signzone, 582
ldns-update, 582
ldns-verify-zone, 582
ldns-walk, 582
ldns-zcat, 582
ldns-zsplit, 582
less, 229
ln, 164
logger, 343
lp, 392
lpadmin, 392
lpc, 392
lpinfo, 388; 391
lpmove, 392
lpoptions, 386; 391
lppasswd, 391
lpq, 392
lpr, 385; 392

lprm, 392
lpstat, 385; 392
ls, 163; 355
lsof, 1086
lvchange, 765
lvdisplay, 764
mail.local, 651
mail/mailx, 288
man, 55
mapper, 582
mdadm, 772
mii-tool, 440
mkdir, 224
mkfs, 779
mknod, 356
modprobe, 372
more, 140
mount, 158; 781; 822
  флаги монтирования, 818; 820
ndc
  инструкция dumpdb, 591
  инструкция notrace, 589
  инструкция trace, 589
ndp, 422
netstat, 437; 500; 826
newaliases, 628
newgrp, 284
newusers, 293
nfsstat, 825
nice, 139
nmap, 1000
  опция -O, 1001
nsupdate, 566
openssl, 1016
passwd, 115
ping, 447
ping6, 447
pkg
  audit, 210
  autoremove, 210
  backup, 210
  clean, 210
  delete, 210
  info, 210
  install, 210
  search, 210
  update, 210
  upgrade, 210
  which, 210
pkill, 134
printf, 234
ps, 135; 1082
reboot, 97
reject, 392
renice, 139
rm, 163; 164
rndc, 557
  инструкция freeze, 591
  инструкция reload, 591
rndc-confgen, 557
rollctl, 582
rollerd, 582
rollinit, 582
route, 420; 437
rpm, 198
rpmbuild, 198
salt-key, 901
scp, 1020
sed, 314
sftp, 1020
sftp-server, 1020
sg_format, 750
sh, 288
showmount, 822
shutdown, 97
smbcontrol, 843
smbpasswd, 836
smbstatus, 841
smrsh, 651
snmpdelta, 1068
snmpdf, 1068
snmpget, 1068
snmpgetnext, 1068
snmpset, 1068
snmptable, 1068
snmptrap, 1068
snmptranslate, 1068
snmpwalk, 1068
sort, 227
ssh, 1019
ssh-add, 1020
ssh-agent, 1020
ssh-keygen, 1020
ssh-keyscan, 1020
strace, 141
su, 108
swapon, 1084
systemctl, 84; 85
  daemon-systemctl, 85
  disable, 85

enable, 85
isolate, 85
kill, 85
list-unit-files, 85
reboot, 85
restart, 85
start, 85
status, 85
stop, 85
tail, 229
tee, 229
traceroute, 449
trustman, 583
udevadm, 358
UEFI
efibootmgr, 73
umount, 158; 782; 824
uname, 353
uniq, 228
uptime, 1044
useradd, 291
usermod, 281
validate, 583
vipw, 282; 286
virsh, 918
vi/vim, 288
vmstat, 1084
wc, 228
wpa_supplicant, 488
zfs, 788
zonesigner, 583
zpool, 788
Коммутатор, 483
Компонент, 202
Конвейер
CI/CD, 961
как код, 969
Контейнер, 924
данных, 934
менеджер
Docker Swarm, 953
Mesos, 952
образ, 925
AWS EC2, 953
Docker Hub, 942
том, 933
шаблон, 928
Контроль версий, 958
Контрольная панель, 1047
Конфиденциальность, 986
Концентратор, 483
Криптография
с открытым ключом, 1009
с симметричными ключами, 1009

Л

Липкая сессия, 726
Локализация, 187

М

Мандат, 120
Маршалинг, 74
Маршрутизатор, 484; 485; 668
Cisco, 512
XORP, 512
Маршрутизация, 419; 499
динамическая, 421
от источника, 427
перенаправление пакетов, 426
протоколы
состояния канала, 504
топологические, 504
стандартный маршрут, 501
статическая, 420
таблица, 419
Массив RAID, 755; 767
Менеджер
логических томов, 191; 754
udev, 359
Метасимволы, 52
Методология
DevOps, 956; 1109
Микросценарий, 216
Миньон, 884
Многопоточность, 128
Модуль, 83
ввода, 333
imfile, 333
imjournal, 333
imklog, 333
immark, 334
imtcp, 334
imuxsock, 333
imudp, 334
вывода, 333
omelasticsearch, 334
omfile, 334
omfwd, 334
omkafka, 334

ommysql, 334
модификации сообщений, 333
разбора, 454
состояние
bad, 86
disabled, 86
enabled, 86
indirect, 86
linked, 86
masked, 86
static, 86
PAM, 607
Мониторинг, 1044
безопасности, 1063
журнала, 1061
приложений, 1061
Мост, 935

Н

Накопитель, 754
USB, 783
Невозможность отказа от авторства, 1008
Непрерывная доставка, 957
Непрерывная интеграция, 957
Неэкранированная витая пара, 480

О

Облако, 299
гибридное, 302
публичное, 302
частное, 302
Облачная платформа, 301
AWS, 311
GCP, 315
Облачные вычисления, 299
Облачный провайдер
Amazon Web Services, 302
DigitalOcean, 302
Google Cloud Platform, 302
IBM Softlayer, 302
Microsoft Azure, 302
OpenStack, 302
Rackspace, 302
VMware vCloud Air, 302
Облачный сервер, 301
Оболочка
регистрационная, 280
Almquist, 222
ash, 222
bash, 222
dash, 222
ksh, 222
sh, 222
tcsh, 222
zsh, 222
Оператор
each, 257
exec, 240
for, 239
read, 240
while, 240
Оптическое волокно, 482
многомодовое, 482
одномодовое, 482
Организация
CAIDA, 414
IANA, 541
ICANN, 398; 413
IGF, 398
ISC, 423
ISO, 481
ISOC, 398
NIST, 1037
SANS, 1038
TIA, 480; 493
Осветительная арматура, 1100
Открытая система
Bro, 1004
OSSEC, 1005
Snort, 1005
Отпечаток ключа, 1020
Отчет
TPS, 218
Очередь
почтовая
/var/spool/clientmqueue, 633
/var/spool/mqueue, 633

П

Пакет
гигантский, 486
apt-mirror, 206
PGP, 1017
Samba, 834
smartmontool, 753
virtualenv, 263
Паника ядра, 379
FreeBSD, 382
Паравиртуализация, 911

Пароль, 995; 996
Передача зоны, 565
Переменная
  окружения, 226
  среды
  PAGER, 55
Перехват сигнала, 132
Платформа
  мониторинга
    с открытым исходным кодом, 1050
    Datadog, 1053
    Graphite, 1050
    Icinga, 1049
    InfluxDB, 1051
    Monitus, 1053
    Munin, 1051
    Nagios, 1049
    OSSEC, 1064
    Pingdom, 1053
    Prometeus, 1050
    Sensu, 1050
    SignalFx, 1053
    SolarWinds, 1053
    Sysdig Cloud, 1053
    Tripwire, 1064
    Zenoss, 1053
  DigitalOcean, 317
  Docker, 924
    реестр, 942
  Kubernetes, 951
  KVM, 917
  Xen, 915
Платформа как услуга, 305
  PaaS, 709
Подкачка, 784
Подключаемый модуль аутентификации,
  PAM, 296
Подписание
  двойное, 581
Показатель
  на основе шаблонов, 1045
  реального времени, 1045
Поле
  GECOS, 279
Пользователь
  nobody, 814
  root, 325; 999
    в NFS, 814
Правило, 359
Право
  владения, 289
  доступа, 289
Предварительная публикация, 581
Препроцессор
  m4, 634
Приоритет, 113
  процесса, 130
Программа
  мониторинга
    logwatch, 1062
    Superviser, 1062
  automount, 827
  aws, 311
  Berkeley DB, 628
  Bro, 1004
  chage, 999
  /etc/rc, 95
  gcloud, 315
  gem, 317
  John the Ripper, 1003
  lighttpd, 200
  Maildrop, 616
  Metasploit, 1002
  Nessus, 1002
  NetworkManager, 436
  OSSEC, 1005
  Packer, 920
  perf, 1089
  procmail, 616
  savelog, 346
  Snort, 1005
  sudo, 108
  Terraform, 469
  tugboat, 317
  Vagrant, 922
  vipw, 226
  virt-install, 918
  Wireshark, 452
  yum, 201
Программирование
  сценариев, 218
Программное обеспечение как услуга, 305
Проект, 315
  389 Directory Server, 600
  OpenLDAP, 599
Протокол
  мониторинга
    StatsD, 1054
  ARP, 400; 407; 422

BAF, 833
BGP, 504; 510
BOOTP, 424
CIDR, 410; 412
CIFS, 833
DHCP, 423; 424; 430
агент ретрансляции, 424
пакет ISC, 425
EIGRP, 504; 507
ESMTP, 616
HTTP, 689
HTTP/1.0, 695
HTTP 1.1, 696
HTTP/2, 690
ICMP, 400
директивы переадресации, 421; 426; 502
широковещательные пакеты, 427
IGMP, 408
IGRP, 507
IMAP, 617
IP, 400
фрагментация пакетов, 405
IPP, 384
IPSEC, 429; 1033
IPv6
адресация, 415
Kerberos, 119
LDAP, 296; 597; 641
OSPF, 507
PAM, 119
POP, 617
PXE, 189
RIP, 504; 506
SASL, 649
SMTP, 615; 619
SNMP, 1065
SSH, 1019
SSL, 649
TCP, 400
TLS, 654; 696; 1012
TSL, 429
UDP, 400
WEP, 490
Процесс
спонтанный, 79
dockerd, 927
nginx, 931
Псевдоустройство, 355

P

Рабочая область, 970
Развертывание, 965
неизменяемое, 966
непрерывное, 957
Раздел, 754
Разработчик, 270
Распознаватель
открытый, 569
Расшифровка, 1008
Регион, 306
Редактор
ed, 229
emacs, 223
vi, 223
Режим
многопользовательский, 80
однопользовательский, 80; 98
с загрузчиком GRUB, 100
Centos, 99
FreeBSD, 99
Red Hat, 99
сервера, 80
Роль, 878
Руткит, 994

C

Сборка, 962
распределенная, 969
Сервер
без сохранения состояния, 806
имен, 522
пограничный, 704
состояние, 806
с сохранением состояния, 806
Apache httpd, 699
CUPS, 384
DHCP, 424
DNS, 518
Dnsmasq, 424
H2O, 699
Jenkins, 967
NGINX, 699
OpenLDAP, 600
OpenSSH, 1027
SMB, 834
Teleport, 1030
Серия
BCP, 400
FYI, 400
STD, 400

Сетевая печать, 386
Сеть
беспроводная
стандарт
IEEE 802.11ac, 487
IEEE 802.11g, 487
IEEE 802.11n, 487
доверия, 1010
доставки содержимого, CDN, 704
локальная
виртуальная , 484
мостовая, 935
оверлейная, 936
отладка, 491
проектирование, 494
прокладка кабелей, 492
управление, 495
Сигнал, 131
BUS, 132
CONT, 132
HUP, 132
INT, 132
KILL, 132
QUIT, 132
SEGV, 132
STOP, 132
TERM, 132
TSTP, 132
USR1, 132
USR2, 132
WINCH, 132
Синтаксис
RainerScript, 338
Система
аутентификации
Kerberos, 603; 1018
билетная
Bugzilla, 1116
Double Choco Latte, 1116
EMC Ionix (Infra), 1116
HEAT, 1116
Jira, 1116
Mantis, 1116
OSTicket, 1116
OTRS, 1116
Remedy, 1116
Request Tracker, 1116
ServiceDesk, 1116
ServiceNow, 1116
Track-It!, 1116
контроля версий, 266
Git, 266
криптографическая
DKIM, 624
мандатного управления доступом, 122
управления конфигурацией, 847
обработчик изменений, 852
пакет, 853
переменная, 851
привязка, 852
среда, 853
факт, 851
Ansible, 855
Chef, 855
PowerShell Desired State Configuration, 855
Puppet, 855
Salt, 855
управления пакетами
APT, 203
AppArmor, 125
DNS, 518
Graylog, 348
Linux, 198
MAC, 122
Rsyslog, 330
SELinux, 123
Ubuntu, 199
загрузочная
PXE, 188
печати
CUPS, 384
регистрации
Syslog, 326; 329
управления пакетами
.deb, 198
pkg, 209
RPM, 198
TCP/IP, 397
yum, 198; 207
Системная прошивка, 70
BIOS, 71
UEFI, 71
Системны вызов
fcntl, 811
flock, 811
fork, 130
ioctl, 355
kill, 104
lockf, 811
settimeofday, 104

socket, 166
sync, 778
wait, 131
Скриплет, 216
Служба
Active Directory, 296
Red Hat Network, 201; 202
Событие, 1045
Сокет
локальный, 166
/dev/log, 326
/run/systemd/journal, 326
/run/systemd/journal/stdout, 326
/run/systemd/journal/syslog, 329
Сообщение
источник, 335
селектор, 335
auth.debug \;auth.!=warning, 336
auth.info \;auth.!err, 336
селектор
auth.=info, 336
auth.info, 336
уровень важности, 335
alert, 336
crit, 336
debug, 336
emerg, 336
err, 336
info, 336
notice, 336
warning, 336
Спам, 622
Специалист по эксплуатации, 270
Список
управления доступом, 175
NFSv4 ACL, 181
Среда
containerd, 925
Средство Syslog
auth, 335
authpriv, 335
cron, 335
daemon, 335
ftp, 335
kern, 335
local0-7, 335
lpr, 335
mail, 335
mark, 335
news, 335
syslog, 335
user, 335
Ссылка
жесткая, 164
мягкая, 164
символическая, 166
Стандарт
Common Criteria, 1037
Filesystem Hierarchy Standard, 162
ISO 27001, 1036
NIST 800-34, 1132
NIST 800-53, 1132
PCI DSS, 1036
SMART, 752
Стандартный канал
STDERR, 223
STDIN, 223
Стек
ELK, 347
Стойка, 1094
Страничный обмен, 1082
Строка
Exec, 90
Строка состояния, 692
Суперблок, 778
Суперпользователь, 106; 999
в NFS, 814
Схема четности, 768
Сценарий, 216; 876
оболочки, 70
deploy.sh, 970
/etc/rc.d/rc.local, 93
Fail2Ban, 1008
helloworld, 232
init, 70

T

Таблица
главная, 828
косвенных назначений, 828
прямых назначений, 827
разделов GUID, 759
DMI, 1079
Тепловая нагрузка, 1100
Тест
интеграции, 964
инфраструктурный, 964
модульный, 963
приемочный, 964

производительности, 964
Технология
SPF, 618
Тип экземпляра, 308
Том
логический, 754
Точка
беспроводного доступа, 488
Точка монтирования, 158
Туннель
защищенный, 1033
IPSEC, 1033

У

Уведомление, 1046
Унифицированный указатель ресурсов, 690
Управление
учетными данными, 297
доступом, 289
журналами, 321
Управляющий терминал, 130
Утилита
apachectl, 712
apt-get, 203; 206
automount, 827
/bin/mail, 614
curl, 694
debconf, 193
dig, 527
drill, 527
fio, 1086
gem, 261
getgrgid, 814
getpwuid, 814
gparted, 760
grub2-mkconfig, 75
grub-mkconfig, 75
gzip, 55
host, 527
iptables, 458
Kickstart, 190
ldapsearch, 601
logrotate, 345; 346
lsof, 159
make, 367
netcat, 690
newsyslog, 346
nslookup, 527
pip, 261
postconf, 679
prstat, 138
pwconv, 282
rkt, 924
rsync, 611
ruby, 263
sar, 1087
system-config-kickstart, 190
systemd-journal-gateway, 326
systemd-journal-remote, 326
systemd-journal-upload, 326
systemd-nspawn, 924
tcpdump, 452
telnet, 622; 690
test, 238
top, 138
update-grub, 75
Утилита sshfp, 1029
Учетная запись, 996
корневая, 106
root, 106

Ф

Файл
.bash_profile, 288
.config, 367
.cshrc, 288
.emacs, 288
.exrc/.vimrc, 288
.gconfpath, 288
.kde/, 288
.profile, 288
.xauth, 293
/boot/defaults/loader.conf, 78
/boot/loader.conf, 78
/dev/kmsg, 326
/dev/random, 1015
/dev/urandom, 1015
/efi/boot/bootx64.efi, 73
/efi/ubuntu/grubx64.efi, 73
/etc/ansible/ansible.cfg, 865
/etc/apt/sources.list, 203; 205
/etc/auto.master, 832
/etc/cups/cupsd.conf, 386
/etc/cups/mime.convs, 387
/etc/cups/mime.types, 387
/etc/defaults/config, 95
/etc/devd.conf, 363
/etc/devfs.conf, 363
/etc/fstab, 158; 824

/etc/group, 105; 279; 284
/etc/grub.d/40_custom, 75
/etc/gshadow, 284
/etc/hostname/, 438
/etc/hosts, 431; 520
/etc/login.conf, 283
/etc/login.defs, 293
/etc/logrotate.conf, 346
/etc/mail/access, 640
/etc/mail/alises, 625
/etc/mail/local-host-names, 641
/etc/mail/sendmail.cf, 631; 633
/etc/mail/service.switch, 631
/etc/mail/submit.cf, 633
/etc/master.passwd, 107; 118; 282; 998
/etc/named.conf, 568
  инструкция
    acl, 551
    controls, 557
    include, 545
    key, 551
    logging, 553; 585
    options, 545
    server, 552
    zone, 554; 556
    пример, 561
    список соответствия адресов, 544
    список управления доступом, 551; 568
/etc/network/interfaces, 438
/etc/newsyslog.conf, 347
/etc/nsswitch.conf, 275; 520; 607; 631
/etc/openldap/slapd.conf, 600
/etc/passwd, 105; 224; 274; 998; 999
  защита, 999
/etc/rc.conf, 95
/etc/rc.conf.local, 95
/etc/rc.d, 96
/etc/resolv.conf, 519
/etc/rsyslog.conf, 331; 332
/etc/rsyslog.d/55-apache.conf, 342
/etc/samba/smb.conf, 835
/etc/selinux/config, 124
/etc/services, 407; 1031
/etc/shadow, 107; 118
/etc/shadow, 998
/etc/shells, 628
/etc/ssh/sshd_config, 1000
/etc/sudoers, 109
/etc/sysconfig/network, 439
/etc/syslog.conf, 334
/etc/systemd/journald.conf, 327
/etc/systemd/journald.conf.d, 327
/etc/udev/udev.conf, 359
/proc/cmd, 140
/proc/cmdline, 140
/proc/cpuinf, 1078
/proc/cwd, 140
/proc/diskstats, 1078
/proc/environ, 140
/proc/exe, 140
/proc/fd, 140
/proc/loadavg, 1058
/proc/maps, 141
/proc/meminfo, 1078
/proc/root, 141
/proc/stat, 141
/proc/statm, 141
/tmp/bash-users, 224
/tmp/corefiles, 224
/usr/lib/systemd/system/nginx.service, 92
/usr/local/etc/smb4.conf, 835
/usr/local/etc/sudoers, 109
/var/log/auth.log, 334
/var/run/utmp, 1000
~/.bash_profile, 108
~/.bashrc, 108
aliases, 626; 627
auth.log, 339
cdrom/autoclose, 365
cdrom/autoeject, 365
dhclient-eth0.leases, 426
dhcpd.conf, 425
Dockerfile, 940
fs/file-max, 365
grub.cfg, 75
iptables-config, 93
kernel/ctrl-alt-del, 365
kernel/panic, 365
kernel/panic_on_oops, 365
kernel/printk_ratelimit, 365
kernel/printk_ratelimit_burst, 365
kernel/shmmax, 365
ks.cfg, 190
lastlog, 325
logrotate.conf, 345
meta/main.yml, 879
munin-node.conf, 1062
net/ip*/conf/default/rp_filter, 365
net/ip*/icmp_echo_ignore_all, 365

net/ip*/ip_forward, 365
net/ip*/ip_local_port_range, 365
net/ip*/tcp_syncookies, 365
network-scripts/ifcfg-eth0, 93
passwd, 626
resolv.conf, 435
sudoers, 109
sudoers.j2, 867
tcp_fin_timeout, 365
vm/overcommit_memory, 365
vm/overcommit_ratio, 365
wtmp, 325
wtmpx, 325
журнальный
apache2/*, 324
apt*, 324
auth.log, 324
boot.log, 324
cloud-init.log, 324
cron, cron/log, 324
daemon.log, 324
debug*, 324
dmesg, 324
dpkg.log, 324
faillog, 324
httpd/*, 324
kern.log, 324
lastlog, 324
mail*, 324
messages, 324
samba/*, 324
secure, 324
sulog, 324
syslog*, 324
wtmp, 324
xen/*, 324
Xorg.n.log, 324
yum.log, 324
модульный, 83
обычный, 164
паролей, 282
переключения, 631
сценария, 872
устройства, 165
блочного, 165
блочный, 355
символьного, 165
символьный, 355
ядра, 160
Файловая система, 157
9660, 156
объединенная, 937
сетевая, 805
автоматическое монтирование, 827
демонтирование, 824
монтирование, 816
devfs, 362
ext2, 777
ext4, 156; 777
FAT, 156
NTFS, 156
/proc, 1078
SMB, 834
безопасность, 840
монтирование, 839
отладка, 841
просмотр, 840
sysfs, 357
ZFS, 156
Фактор уступчивости, 139
Фильтр, 340
Флаг
setgid, 106
setuid, 106
Флеш-диск, 739
Формат
JSON, 707
Maildir, 616
mbox, 616
XML, 707
YAML, 863
Фрагмент, 195
Функция
getpwnam(), 274
getpwuid(), 274

X

Хеш-функция, 1012
Хранилищ
пакетов, 201
Хранилище
1Password, 998
блочное, 309
облачное, 309
объектное, 309
паролей, 997
эфемерное, 309
GitHub, 270
GitLab, 270

Ц

Целостность, 986
Цель, 87
Цепочка доверия, 573
Цифровая подпись, 1010

Ч

Червь, 993

Ш

Шифр, 1008
  асимметричный, 1010
  симметричный, 1009
  с открытым ключом, 1010
Шифрование, 1008

Э

Экземпляр, 308
Электронная почта
  агент
    доставки, 614; 616
    доступа, 614; 616
    пользовательский, 613; 614
    представления, 615
    транспортный, 614; 615
  почтовые псевдонимы, 625
  сообщение
    маршрутизация, 539
    структура, 617
    хранилище, 616
  стандарт MIME, 614
Электронное оборудование, 1099
Электропитание, 1094

Ю

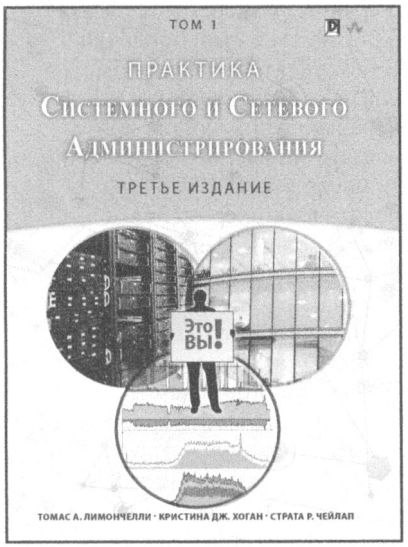
Юнит, 83

Я

Ядро, 351
  безопасность, 443
  FreeBSD, 353
  Linux, 353
Язык программирования
  Go, 706
  Java, 706
  JavaScript, 219
  Node.js, 706
  Perl, 219
  PHP, 219; 706
  Python, 219; 705
  Ruby, 219; 705
Якорь, 691

ПРАКТИКА СИСТЕМНОГО И СЕТЕВОГО АДМИНИСТРИРОВАНИЯ
ТОМ 1
ТРЕТЬЕ ИЗДАНИЕ

Томас А. Лимончелли
Кристина Дж. Хоган
Страта Р. Чейлап



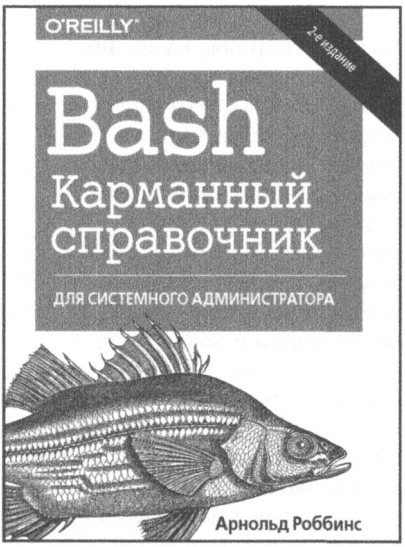
www.williamspublishing.com

В книге описана практика авторитетных системных администраторов, имеющих многолетний опыт работы. В ней изложены стратегические решения и концепции, поучительные истории и рекомендации, касающиеся системного и сетевого администрирования, в частности, вопросы автоматизации и централизации процессов инсталляции и поддержки, управления парком рабочих станций, серверами и службами, а также проблемы хранения данных и менеджмента (этические нормы, наем и увольнение сотрудников, управление временем и т.д.). Книга предназначена для системных администраторов всех уровней, которые стремятся освоить современные методы и стратегии. Она также будет полезной для менеджеров, управляющих работой системных администраторов.

ISBN 978-5-6040043-1-9 **в продаже**

BASH
КАРМАННЫЙ СПРАВОЧНИК
СИСТЕМНОГО АДМИНИСТРАТОРА
2-Е ИЗДАНИЕ

Арнольд Роббинс



Это второе, обновленное издание карманного справочника по оболочке Bash в краткой и удобной форме содержит самое основное, что требуется знать для работы с этой оболочкой. В нем читатель сможет быстро найти ответы на вопросы, нередко возникающие при написании сценариев оболочки, включая вызов оболочки, подстановки переменных, параметров, псевдонимов, предыстории выполнения команд, арифметических выражений, автозавершение вводимых команд, управление заданиями, применение функций, параметров оболочки и встроенных в нее команд. Книга адресована опытным пользователям, занимающимся разработкой программ и сценариев оболочки в Unix-подобных системах.

www.williamspublishing.com

ISBN 978-5-9909445-4-1 **в продаже**

Как автор, редактор и издатель, я никогда не придавал большого значения конкуренции — за исключением нескольких случаев. Это один из таких случаев. Unix System Administration Handbook — *это одна из немногих книг, на которые мы равняемся.* — **Тим О'Рейли, основатель компании O'Reily Media**

Это издание предназначено для тех, чьи системы работают в облаке или в виртуализированных центрах обработки данных; тех, чья административная работа в основном связана с написанием сценариев автоматизации и конфигурации; тех, кто тесно сотрудничает с разработчиками, сетевыми инженерами, сотрудниками по вопросам соблюдения требований и всеми другими рабочими "пчелами", которые обитают в современном "улье". — **Пол Викси (Paul Vixie), изобретатель и основатель компаний ISC и Farsight Security, лауреат премии Зал Славы Интернета (Internet Hall of Fame)**

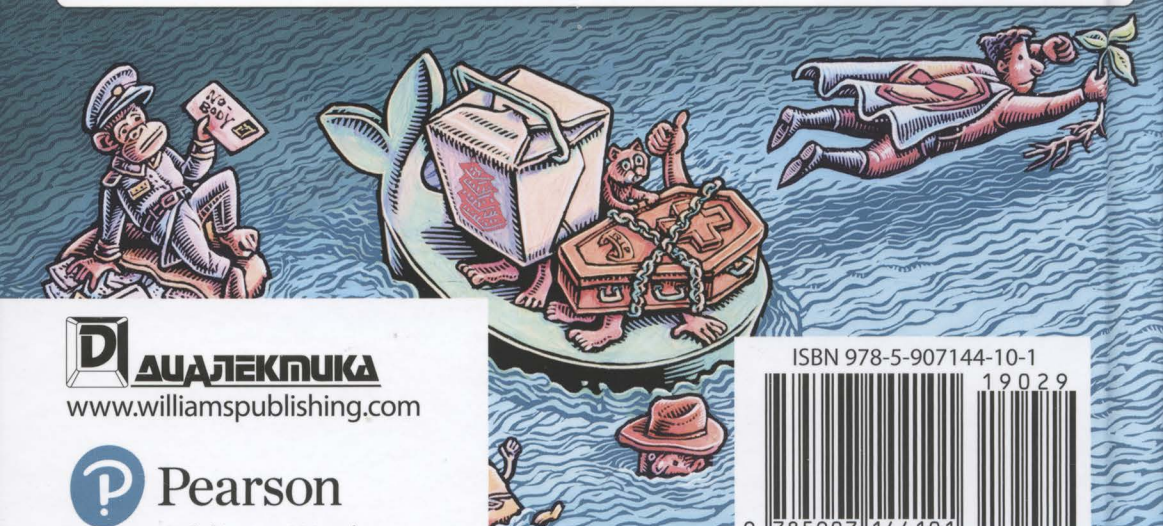
Это современное и полное руководство по инсталляции, настройке и обслуживанию любой системы UNIX или Linux, включая системы, предоставляющие базовую инфраструктуру Интернета и облачную инфраструктуру.

Обновленное с учетом новых дистрибутивов и облачных сред, это всестороннее руководство охватывает лучшие практики для всех аспектов системного администрирования, включая управление хранением данных, проектирование и администрирование сети, безопасность, веб-хостинг, автоматизацию, управление конфигурацией, анализ производительности, виртуализацию, DNS, безопасность и управление IT-организациями. Авторы — специалисты мирового класса — рассмотрели облачные платформы, методологию DevOps, непрерывное развертывание, контейнеризацию, мониторинг и многие другие важные темы.

Независимо от вашей роли в системах и сетях, построенных на UNIX или Linux, это понятное, хорошо написанное руководство поможет повысить эффективность и решить самые острые проблемы.

Эта книга увлекательна и полезна как справочник. Если вы используете системы UNIX и Linux, она должна стать вашей настольной книгой. В ней кратко и без лишних разглагольствований написано об истории этих систем. Она содержит точную информацию, которая излагается в яркой и запоминающейся форме.
— **Джейсон А. Наннелли (Jason A. Nunnelly)**

informit.com



ДИАЛЕКТИКА
www.williamspublishing.com

Pearson
Addison-Wesley

ISBN 978-5-907144-10-1
19029
9 785907 144101