Community Experience Distilled

Mastering Ubuntu Server

Get up to date with the finer points of Ubuntu Server using this comprehensive guide

Jay LaCroix

[PACKT] open source* PUBLISHING
community experience distilled

www.it-ebooks.info

Mastering Ubuntu Server

Get up to date with the finer points of Ubuntu Server using this comprehensive guide

Jay LaCroix

[PACKT] PUBLISHING open source community experience distilled

BIRMINGHAM - MUMBAI

www.it-ebooks.info

Mastering Ubuntu Server

Copyright © 2016 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: July 2016

Production reference: 1210716

Published by Packt Publishing Ltd.

Livery Place

35 Livery Street

Birmingham B3 2PB, UK.

ISBN 978-1-78528-452-6

www.packtpub.com

www.it-ebooks.info

Credits

Author
Jay LaCroix

Reviewers
David Diperna
Robert Stolk

Acquisition Editor
Prachi Bisht

Content Development Editor
Trusha Shriyan

Technical Editor
Vishal K. Mewada

Copy Editor
Madhusudan Uchil

Project Coordinator
Kinjal Bari

Proofreader
Safis Editing

Indexer
Hemangini Bari

Graphics
Kirk D'Penha

Production Coordinator
Shantanu N. Zagade

Cover Work
Shantanu N. Zagade

www.it-ebooks.info

About the Author

Jay LaCroix is an open-source enthusiast, specializing in Linux. He is passionate about Linux and its long-term adoption, and has been using Linux since 2002. Jay currently works as a DevOps Engineer for SolvIT Inc, and he holds a masters degree in Information Systems Technology Management from Capella University. During the course of his studies, he's obtained several certifications, including Linux+, LPIC-2, and SUSE Certified Linux Professional, among others. In addition to a great career working with Linux technologies, Jay also has a YouTube channel, available at `LearnLinux.tv`, where he posts instructional tutorial videos demonstrating how to utilize Linux and open-source technologies. When not practicing with technology, Jay enjoys photography, music, classic video-games and writing. He has also written *Linux Mint Essentials and Mastering Linux Network Administration*, also published by Packt Publishing.

www.it-ebooks.info

Acknowledgments

First and foremost, I would like to thank Randy Schapel at Mott Community College for conducting a Linux class back in 2002 that was responsible for creating a life-long addiction for me. During a time where Linux wasn't nearly as popular as it is today, that class spring-boarded my career and passion. In addition, I would also like to thank Packt Publishing for the wonderful opportunity I've been given, to share my knowledge with others. I never would've thought I'd be a published author, but here I am. It's hard to believe, but this is my third book, and writing it has been an incredibly enjoyable experience. I would like to thank Prachi Bisht for the opportunity of writing this book in particular, and Trusha Shriyan for supporting me as it was being written. In addition, I would also like to thank James Jones for introducing me to writing for Packt in the first place, back in 2014. I would also like to thank everyone else I've worked with at Packt, who have all been very supportive.

I would like to thank my best friend Krys for being there for me, Jim for his help and support, my two sons Alan and Johnny for being great kids, and my family on the Sherman side for not only being amazing, but also very supportive. I would like to thank the team at Security Inspection for being an amazing group of people to work with.

Last but certainly not least, I would like to thank my readers for their continued support as well as all my viewers and subscribers on my Youtube channel. It's a pleasure and an honor to create content to help you all advance your careers.

www.it-ebooks.info

About the Reviewers

David Diperna is a Linux system administrator who graduated from Oakland University with degrees in computer science and psychology. His work consists of multiple projects in different areas of Linux, but his main focus is on monitoring and logging solutions. Currently, he is working towards the CompTIA Linux+ and AWS sysops certifications and plans to complete his master's degree in computer science.

I'd like to thank Packt Publishing for the opportunity and Jay for his support.

Robert Stolk is an IT professional and has been working in the IT business for 18 years. He lives in the city of Rotterdam in the Netherlands.

He has experience as a Unix/Linux engineer, storage manager, technical application manager, and functional application manager in the area of ISPs, post office, (mobile) telecommunication, energy, hospitals, tax office, and banking.

He has fair knowledge of and uses the following Linux distributions at home and professionally: Ubuntu, Debian, RedHat, and CentOS.

Currently, he works for Olgreen in Almere, the Netherlands, as a technical specialist.

On his current project, he is the functional application manager of mobile applications.

www.it-ebooks.info

www.PacktPub.com

eBooks, discount offers, and more

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at customercare@packtpub.com for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

https://www2.packtpub.com/books/subscription/packtlib

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can search, access, and read Packt's entire library of books.

Why subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print, and bookmark content
- On demand and accessible via a web browser

www.it-ebooks.info

www.it-ebooks.info

Table of Contents

Preface v

Chapter 1: Deploying Ubuntu Server 1

- Setting up our lab 2
- Determining your server's role 3
- Obtaining Ubuntu Server 4
- Deciding between 32-bit and 64-bit installations 6
- Creating a bootable Ubuntu Server flash drive (Windows) 7
- Creating a bootable Ubuntu Server flash drive (Linux) 10
- Creating a bootable Ubuntu Server flash drive (Mac) 13
- Planning the partition layout 16
- Installing Ubuntu Server 18
- Installing Ubuntu Server (Raspberry Pi) 37
- Summary 39

Chapter 2: Managing Users 41

- Understanding when to use root 42
- Creating and removing users 43
- Understanding the /etc/passwd and /etc/shadow files 49
- Distributing default configuration files with /etc/skel 53
- Switching between users 54
- Managing groups 56
- Managing passwords and password policies 59
- Configuring administrator access with sudo 63
- Setting permissions on files and directories 67
- Summary 74

www.it-ebooks.info

Chapter 3: Managing Storage Volumes 75
Viewing disk usage 76
Adding additional storage volumes 80
Partitioning and formatting volumes 84
Mounting and unmounting storage volumes 87
Understanding the /etc/fstab file 89
Managing swap 94
Utilizing LVM volumes 97
Using symbolic and hard links 106
Summary 109
Chapter 4: Connecting to Networks 111
Setting the hostname 112
Managing network interfaces 114
Assigning static IP addresses 118
Understanding Linux name resolution 122
Understanding Network Manager 124
Getting started with OpenSSH 125
Getting started with SSH key management 129
Simplifying SSH connections with a ~/.ssh/config file 133
Summary 134
Chapter 5: Managing Software Packages 135
Understanding Linux package management 136
Installing and removing software 138
Searching for packages 142
Managing software repositories 144
Keeping your server up to date 149
Backing up and restoring packages 156
Making use of aptitude 157
Installing Snap packages 160
Summary 163
Chapter 6: Controlling and Monitoring Processes 165
Showing running processes with the ps command 166
Managing jobs 171
Killing misbehaving processes 174
Utilizing htop 176
Managing system processes 179
Monitoring memory usage 184
Scheduling tasks with Cron 188
Understanding load average 190
Summary 193

www.it-ebooks.info

Chapter 7: Managing Your Ubuntu Server Network 195
- Planning your IP address scheme 195
- Serving IP addresses with isc-dhcp-server 199
- Setting up name resolution (DNS) with bind 204
- Creating a secondary DNS server 211
- Setting up an Internet gateway 215
- Keeping your system clock in sync with NTP 217
- Summary 220

Chapter 8: Accessing and Sharing Files 221
- File server considerations 221
- Sharing files with Windows users using Samba 223
- Setting up NFS shares 229
- Transferring files with rsync 233
- Transferring files with SCP 238
- Mounting remote filesystems with SSHFS 241
- Summary 243

Chapter 9: Managing Databases 245
- Preparations for setting up a database server 246
- Installing MariaDB 248
- Taking a look at MariaDB configuration 251
- Understanding how MariaDB differs in Ubuntu 16.04 254
- Managing databases 255
- Setting up a slave DB server 262
- Summary 267

Chapter 10: Serving Web Content 269
- Installing and configuring Apache 269
- Installing additional Apache modules 276
- Securing Apache with SSL 278
- Setting up high availability with keepalived 285
- Installing and configuring ownCloud 290
- Summary 295

Chapter 11: Virtualizing Hosts and Applications 297
- Setting up a virtual machine server 298
- Creating virtual machines 306
- Bridging the virtual machine network 311
- Creating, running, and managing Docker containers 315
- Summary 323

www.it-ebooks.info

Chapter 12: Securing Your Server 325
- Lowering your attack surface 326
- Securing OpenSSH 330
- Installing and configuring Fail2ban 334
- MariaDB best practices 338
- Setting up a firewall 341
- Encrypting and decrypting disks with LUKS 344
- Locking down sudo 346
- Summary 347

Chapter 13: Troubleshooting Ubuntu Servers 349
- Evaluating the problem space 350
- Conducting a root-cause analysis 352
- Viewing system logs 354
- Tracing network issues 359
- Troubleshooting resource issues 364
- Diagnosing defective RAM 368
- Summary 372

Chapter 14: Preventing and Recovering from Disasters 373
- Preventing disasters 374
- Utilizing Git for configuration management 376
- Implementing a backup plan 383
- Creating system images with Clonezilla live 386
- Utilizing bootable recovery media 399
- Summary 403

Index 405

www.it-ebooks.info

Preface

Ubuntu is an exciting platform. You can find it everywhere—desktops, laptops, phones, and especially servers. The Server edition enables administrators to create efficient, flexible, and highly available servers that empower organizations with the power of open source. As Ubuntu administrators, we're in good company—according to W3Techs, Ubuntu is the most widely deployed distribution on the Web in regards to Linux. With the release of Ubuntu 16.04, this platform becomes even more exciting!

In this book, we will dive right into Ubuntu Server and learn all the concepts needed to manage our servers and configure them to perform all kinds of neat tasks, such as serving web pages, managing virtual machines, and sharing data with other users, among many other things. We'll start our journey right in the first chapter, where we'll walk through the installation of Ubuntu Server 16.04, which will serve as the foundation for the rest of the book. As we proceed through the journey, we'll look at managing users, connecting to networks, and controlling processes. Later, we'll implement important technologies such as DHCP, DNS, Apache, and MariaDB, among others. We'll even set up our own ownCloud server along the way.

Finally, the end of the book will cover various things we can do in order to troubleshoot issues, as well as prevent and recover from disasters.

What this book covers

Chapter 1, *Deploying Ubuntu Server*, covers the installation process for Ubuntu Server. This chapter will walk you through creating bootable media on various operating systems, planning the partition layout, and going through the installation process. Installation on the Raspberry Pi is also covered.

www.it-ebooks.info

Chapter 2, *Managing Users*, covers user management in full. Topics here include creating and removing users, password policies, the sudo command, group management, and switching from one user to another.

Chapter 3, *Managing Storage Volumes*, takes a look at storage volumes. You'll be shown how to view disk usage, format volumes, manage the /etc/fstab file, utilize LVM, and more. In addition, we'll look at managing swap and creating links.

Chapter 4, *Connecting to Networks*, takes a look at networking in Ubuntu, specifically how to connect to resources from other nodes. We'll look at assigning IP addresses, connecting to other nodes via OpenSSH, using Network Manager, and name resolution.

Chapter 5, *Managing Software Packages*, takes you through the process of searching for, installing, and managing packages. This will include a look at managing apt repositories, installing packages, and keeping your server up to date. This will even include a look at Snappy packages, which is a promising new package format for Ubuntu systems.

Chapter 6, *Controlling and Monitoring Processes*, teaches you how to understand what is running on the server, as well as how to stop misbehaving processes. This will include a look at htop, systemd, managing jobs, and understanding load average.

Chapter 7, *Managing Your Ubuntu Server Network*, revisits networking with more advanced concepts. In this chapter, you will learn more about the technologies that glue your network together, such as DHCP and DNS. In this chapter, you will set up your own DHCP and DNS server and install NTP.

Chapter 8, *Accessing and Sharing Files*, is all about sharing files with others. Concepts will include the setting up of Samba and NFS network shares and will even go over transferring files manually with rsync and scp.

Chapter 9, *Managing Databases*, takes you on a journey through setting up and managing databases via MariaDB. You will learn to install MariaDB, set up databases, and create a slave database server.

Chapter 10, *Serving Web Content*, takes a look at serving content with Apache. In addition, you will be shown how to secure Apache with an SSL certificate, manage modules, and set up keepalived. Setting up OwnCloud is also covered.

Chapter 11, *Virtualizing Hosts and Applications*, is all about virtualization. You will be walked through setting up your very own KVM installation, as well as utilizing Docker for containing individual applications.

www.it-ebooks.info

Chapter 12, *Securing Your Server*, takes a look at various things you can do in order to strengthen security on Ubuntu servers. Topics will include concepts such as lowering the attack surface, securing OpenSSH, and setting up a firewall, among others.

Chapter 13, *Troubleshooting Ubuntu Servers*, consists of topics relating to things we can do when our deployments don't go exactly according to plan. You will also investigate the problem space, view system logs, and trace network issues.

Chapter 14, *Preventing and Recovering from Disasters*, informs you of various strategies that can be used to prevent and recover from disasters. This will include a look at utilizing Git for configuration management, implementing a backup plan, creating hard disk images with Clonezilla, and more.

What you need for this book

This book covers Ubuntu Server in depth, and in order to follow along, you'll need to download Ubuntu Server 16.04 and have a machine or two to practice on. It really doesn't matter if you use virtual or physical machines, as the nature of the machine is irrelevant so long as it can run Ubuntu. Virtual machines have the added benefit of snapshots, which you can use to test and roll back modifications to the server.

In the case of virtual machines, it's recommended to use bridged mode for networking so that each VM can see each other easily over the network. It's a good idea to use multiple machines if you can, since several chapters will include networking concepts, such as connecting to other hosts via SSH, setting up a website, and accessing databases.

You'll need to create bootable media in order to install Ubuntu Server. On physical servers, this will require either a blank CD or empty flash drive. In the first chapter, the installation process will be covered and will include a walkthrough on creating bootable media. Ubuntu Server is available online at `http://www.ubuntu.com/server`.

Since Ubuntu Server doesn't include a graphical user interface by default, it can run on relatively modest hardware. As long as you have 512 MB of RAM and 8 GB of hard disk space available, you should be in good shape. Raspberry Pi users will need an available SD card, preferably 8 GB or larger. Raspberry Pi models 2 and 3 are supported by Ubuntu Server.

www.it-ebooks.info

Who this book is for

This book is for readers who already have some experience with Linux, though it doesn't necessarily have to be with Ubuntu. Preferably, you have basic Linux command line skills, such as changing directories, listing contents, and issuing commands as regular users or with root. Even if you don't have these skills, you should read this book—the opening chapters will cover many of these concepts.

In this book, we'll take a look at real-world situations in which we can deploy Ubuntu Server. This will include the installation process, serving web pages, setting up databases, and much more. Specifically, the goal here is to be productive. Each chapter will teach you a new and valuable concept, using practical examples that are relative to real organizations. Basically, we focus on getting things done, not primarily on theory. Although the theory that goes into Linux and its many distributions is certainly interesting, the goal here is to get you to the point where if a work colleague or client asks you to perform work on an Ubuntu-based server, you'll be in a good position to get the task done. Therefore, if your goal is to get up and running with Ubuntu Server and learn the concepts that really matter, this book is definitely for you.

Conventions

In this book, you will find a number of text styles that distinguish between different kinds of information. Here are some examples of these styles and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "MySQL and MariaDB create their log files in the `/var/log/mysql` directory."

A block of code is set as follows:

```
subnet 192.168.1.0 netmask 255.255.255.0 {
       range 192.168.1.100 192.168.1.240;
       option routers 192.168.1.1;
       option domain-name-servers 192.168.1.1;
```


www.it-ebooks.info

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
[Documents]
        path = /share/documents
        force user = myuser
        force group = users
        public = yes
        writable = no
```

New terms and **important words** are shown in bold. Words that you see on the screen, for example, in menus or dialog boxes, appear in the text like this: "Clicking the **Next** button moves you to the next screen."

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Any command line input or output is written as follows:

```
dpkg --get-selections > installed_packages.txt
```

Some commands need to be executed as the root user or with sudo. I'll mark these commands with the hash symbol (#) such as in the following example:

```
# apt-get install openssh-server
```

If you don't see a hash symbol preceding a command, then sudo or root privileges aren't necessary for you to run it. But if you do see this symbol, take note that you'll need root privileges. I won't always remind you of this beforehand. In most cases, it usually doesn't matter if you use sudo or switch to the root user, I'll leave that up to you. But if it does matter in a particular example, I will let you know.

www.it-ebooks.info

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of.

To send us general feedback, simply e-mail `feedback@packtpub.com`, and mention the book's title in the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the color images of this book

We also provide you with a PDF file that has color images of the screenshots/diagrams used in this book. The color images will help you better understand the changes in the output. You can download this file from `https://www.packtpub.com/sites/default/files/downloads/MasteringUbuntuServer_ColorImages.pdf`.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to `https://www.packtpub.com/books/content/support` and enter the name of the book in the search field. The required information will appear under the **Errata** section.

www.it-ebooks.info

Piracy

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors and our ability to bring you valuable content.

Questions

If you have a problem with any aspect of this book, you can contact us at `questions@packtpub.com`, and we will do our best to address the problem.

www.it-ebooks.info

www.it-ebooks.info

1
Deploying Ubuntu Server

Ubuntu Server is an extremely powerful distribution for servers and network appliances. Whether you're setting up a high-end database host or a small office file server, the flexible nature of Ubuntu Server will meet and surpass your needs. In this book, we'll walk through all the common use-cases to help you get the most out of this exciting platform. In this chapter in particular, I'll guide you through the process of deploying Ubuntu Server on your system. We'll start off with some discussion on best practices, and then we'll obtain the software and create our installation media. Next, I'll give you a step by step rundown of the entire installation procedure. By the end of this chapter, you'll have an Ubuntu Server installation of your own to use throughout the remainder of this book. I'll even show you the process of installing Ubuntu Server on a Raspberry Pi for good measure.

In particular, we will cover:

- Setting up our lab
- Determining your server's role
- Obtaining Ubuntu Server
- Deciding between 32-bit and 64-bit installations
- Creating a bootable Ubuntu Server flash drive (Windows)
- Creating a bootable Ubuntu Server flash drive (Linux)
- Creating a bootable Ubuntu Server flash drive (Mac)
- Planning the partition layout
- Installing Ubuntu Server
- Installing Ubuntu Server (Raspberry Pi)

www.it-ebooks.info

Setting up our lab

The first thing for us to do is to set up our environment. For the purposes of this book, it really doesn't matter what kind of hardware you use, if you even use physical hardware at all (virtual machines are also fine). If you have physical server hardware available to you, then by all means use it. Practicing on physical hardware is always preferred if you can, though not everyone has the money (or even the available space) to set up a complete lab. Nowadays, processing power and memory is cheaper than it's ever been, and the typical home PC is capable of running several virtual machines. Even better, software such as VirtualBox is free, so even if you have no budget at all, there are definitely options available to you. Perhaps you already have a server or two in your rack at work just waiting to be deployed, in which case, this will be even more exciting and fun. In the case of VirtualBox, it can be downloaded from here: `https://www.virtualbox.org`.

While I always prefer physical hardware, virtual machines have one major advantage. Not only are they easy to create and destroy at will, they are perfect for testing new or proposed solutions because you can easily take a snapshot of a VM and restore it if an experiment blows up in your face. In my case, it's common for me to create a virtual machine of a Linux distribution, fully update it, and then take a snapshot before making any changes. That way, I always have a base version of the OS to fall back on if I muck it up. Testing software roll-outs on virtual machines before graduating a solution to production is a good habit to get into anyway.

This book makes no assumptions regarding your environment, because the hardware one has access to differs from person to person. While the installation procedure for Ubuntu Server differs based on hardware (for example, perhaps your server doesn't have a physical optical drive, forcing you to use USB media instead of a CD), all other concepts contained throughout will be the same regardless.

With that said, once we run through the install procedure, feel free to install Ubuntu Server on as many devices or VMs as you can. If I can make any recommendation, it would be to utilize both physical and virtual servers if you can. Not because this book requires you to, but because it's a good habit to get into. If all else fails, cloud **Virtual Private Server** (**VPS**) providers such as Linode and Digital Ocean have Ubuntu Server available as an option for a small monthly fee. See the following URLs for more information on those providers:

- `https://www.digitalocean.com`
- `https://www.linode.com`

www.it-ebooks.info

These cloud providers are great if you wish to get a server up and running quickly or if your goal is to set up a server to be available in the cloud. With that said, all you need to do is have Ubuntu Server installed on something (basically anything at all) and you're ready to follow along.

Determining your server's role

While at this point your goal is most likely to set up an Ubuntu Server installation for the purposes of practicing the concepts contained within this book, it's also important to understand how a typical server roll-out is performed. Every server must have a purpose, also known as its **Role**. This role could be that of a database server, web server, and so on. Basically, it's the value the server adds to your network or your organization. Sometimes, servers may be implemented solely for the purpose of testing experimental code. And this is important too—having a test environment is a very common (and worthwhile) practice.

Once you understand the role your server plays within your organization, you can plan for its importance. Is the system mission critical? How would it affect your organization if for some reason this server malfunctioned? Depending on the answer to this question, you may only need to set up a single server for this task, or you may wish to plan for redundancy such that the server doesn't become a central point of failure. An example of this may be a DNS server, which would affect your colleague's ability to resolve local host names and access required resources. It may make sense to add a second DNS server to take over in the event that the primary server becomes unavailable.

Another item to consider is how confidential the data residing on a server is going to be for your environment. This directly relates to the installation procedure we're about to perform, because you will be asked whether or not you'd like to utilize **encryption**. The encryption that Ubuntu Server offers during installation is known as **encryption at rest**, which refers to the data stored within the storage volumes on that server. If your server is destined to store confidential data (accounting information, credit card numbers, employee or client records, and so on), you may want to consider making use of this option. Encrypting your hard disks is a really good idea to prevent miscreants with local access from stealing data. As long as the miscreant doesn't have your encryption key, they cannot steal this confidential information. However, it's worth mentioning that anyone with physical access can easily destroy data (encrypted or not), so definitely keep your server room locked!

www.it-ebooks.info

At this point in the book, I'm definitely not asking you to create a detailed flow chart or to maintain an IP roll-out spreadsheet, just to keep in mind some concepts that should always be part of the conversation when setting up a new server. It needs to have a reason to exist, it should be understood how critical and confidential the server's data will be, and the server should then be set up accordingly. Once you practice these concepts as well as the installation procedure, you can make up your own server roll-out plan to use within your organization going forward.

Obtaining Ubuntu Server

For starters, we'll need to get our hands on Ubuntu Server and then create bootable installation media in order to install it. How you do this largely depends on your hardware. Does your server have an optical drive? Is it able to boot from USB? Refer to the documentation for your server in order to find out. In most cases, it's a very good idea to create both a bootable CD and USB media of Ubuntu Server while you're at it. That way, regardless of how your server boots, you're most likely covered.

Unfortunately, the differing age of servers within a typical data center introduces some unpredictability when it comes to how to boot installation media. When I first started with servers, it was commonplace for all servers to contain a 3.5 inch floppy disk drive, and some of the better ones even contained a CD drive. Nowadays, servers typically contain neither and only ship with an optical drive if you ask for one nicely while placing your order. If a server does have an optical drive, it typically will go unused for an extended period of time and become faulty without anyone knowing until the next time someone goes to use it. Some servers boot from USB; others don't. In order to continue, check the documentation for your hardware and plan accordingly. Your server's capabilities will determine which kind of media you'll need to create.

Thankfully, we only need to download a single file in order to create our media, regardless of whether or not we plan on creating USB or CD media. To get started, navigate to `http://www.ubuntu.com/` in your browser, which will bring you to the main site for Ubuntu. On the top right, there will be a download button, and if you hover your cursor over that, you'll see an option for the Server version. Alternatively, you can simply Google Ubuntu Server and it should be the first result, or close. Just make sure the URL in the search result has a domain of `ubuntu.com` before you click on it. The file that you download will be an ISO file, ending with a `.iso` file extension. This is an image file that we will later use to create bootable media.

www.it-ebooks.info

When this book was written, Ubuntu Server 16.04 is the most recent version. Depending on when you're reading this, the next release, 16.10 (or newer) may also be available. So, on the Ubuntu website, you may have a choice between LTS and non-LTS. Ubuntu 16.04 is **Long-Term Support** (**LTS**) while 16.10 is not. What does this mean? Essentially, whether or not an Ubuntu release is an LTS release determines the length of time it will be supported with security updates. Non-LTS releases are supported with security updates for just nine months, while LTS releases are supported for five years. This makes LTS releases perfect for servers, since it's not typical for an administrator to upgrade the entire distribution frequently. Non-LTS releases can be very handy for testing code on a newer software stack, but that's the only reason I can think of to consider a non-LTS release on a server. When in doubt, stick with LTS.

Periodically throughout the life cycle of an LTS release of Ubuntu, several "point releases" are published. These are similar to service packs in comparison with Windows, as they contain current software and security updates built in. They also contain the latest hardware drivers. Point releases for Ubuntu Server 16.04 will be 16.04.1, 16.04.2, and so on. You should always download the latest version as newer versions support newer hardware.

If you're setting up a virtual machine, then the ISO file you download from the Ubuntu download site will be all you need. In that case, all you should need to do is create a VM, attach the ISO to the virtual optical drive, and boot it. From there, the installer should start, and you can proceed with the installation procedure contained within this chapter. Going over the process of booting an ISO image on a virtual machine differs from hypervisor to hypervisor, so going over the process on each would be beyond the scope of this book. Thankfully, the process is usually straight-forward and you can find the details within the documentation of your hypervisor or from performing a quick Google search.

As I mentioned before, I recommend creating both a bootable USB and bootable CD. This is due to the fact that you'll probably run into situations where you have a server that doesn't boot from USB, or perhaps your server doesn't have an optical drive and the USB key is your only option. In addition, the Ubuntu Server boot media also makes great recovery media if for some reason you need to rescue a server. To create a bootable CD, the process is typically just a matter of downloading the ISO file and then right-clicking on it. In the right-click menu, you should have an option to burn to disc or similar. This is true of Windows, as well as most graphical desktop environments of Linux when a disc burning application installed. If in doubt, Brasero is a good disc burning utility to download for Linux.

www.it-ebooks.info

The exact procedure differs from system to system, mainly because there is a vast amount of software combinations at play here. For example, I've seen many Windows systems where the right-click option to burn a CD was removed by an installed CD/DVD burning application. In that case, you'd have to first open your CD/DVD burning application and find the option to create media from a downloaded ISO file. As much as I would love to outline the complete process here, no two Windows PCs typically ship with the same CD/DVD burning application. The best rule of thumb is to try right-clicking on the file to see if the option is there, and if not, refer to the documentation for your application. Keep in mind that a "Data Disc" is not what you want, so make sure to look for the option to create media from an ISO image or your disc will be useless.

At the time of writing, the Ubuntu Server ISO image will fit on a standard 700 MB CD-R disc. However, the desktop flavors will require a writable DVD, and each release of Ubuntu is larger than previous ones, so at some point perhaps the server version will require a writable DVD as well. If the image size is over 700 MB, be sure to grab a blank DVD instead.

Creating a bootable USB is a bit more involved but easier to outline in steps, since each platform has a generally agreed upon method of creating them. In Linux, we can use `ddrescue`, or simply `dd`, while in Windows we can use the Universal USB installer from `http://www.pendrivelinux.com/`. In later section in this chapter, I'll outline the procedure of creating a bootable USB in Windows, Linux, and Mac.

Deciding between 32-bit and 64-bit installations

You might be wondering which version of Ubuntu Server (32-bit or 64-bit) you should download and install. After all, it does give you an option for both when you go to fetch the ISO image. So, which one should you go for? Nowadays, most (if not all) servers that are currently being sold are compatible with 64-bit operating systems. In fact, the majority of end-user PCs are also compatible with 64-bit operating systems. However, it's often the case that you may have older hardware lying around, so any existing servers you already have may or may not contain processors with 64-bit registers. Generally speaking, though, the 64-bit version is recommended, but the decision really comes down to whether or not you wish to run legacy 32-bit applications and what your hardware is able to support.

www.it-ebooks.info

Is a legacy application causing you to consider going with a 32-bit installation? It's often the case that legacy applications remain in use for quite some time in the enterprise, but thankfully 32-bit applications will run just fine (most of the time) in a 64-bit Linux installation, providing that the libraries required for the application have 32-bit versions available in Ubuntu's software repositories. We'll discuss software repositories and installing applications in more detail in *Chapter 5*, *Managing Software Packages*.

Often, the decision between 32-bit and 64-bit may come down to the misconception that 32-bit operating systems are unable to utilize more than 4 GB of RAM. In the Microsoft world, it's common to install 32-bit Windows on computers with less than 4 GB of RAM, and 64-bit Windows on computers with 4 GB or more. This is due to 32-bit versions of Windows being unable to utilize an address space of more than 4 GB. What many aren't aware of, however, is that this limitation is exclusive to Windows. Many 32-bit Linux distributions can access much more than 4 GB of RAM, by way of utilizing a **Physical Address Extension** (**PAE**), which is compiled into the kernel of most Linux distributions nowadays. Since Ubuntu 16.04 is able to utilize PAE, it can access more than 4 GB of RAM. If you prefer to use a 32-bit operating system and are concerned whether or not Ubuntu will be able to utilize all of your memory, you can forget your worries. It can support up to 64 GB!

If you're in doubt, choose the 64-bit version of Ubuntu on any computer or server you wish to install it on, unless you have a very specific use-case which makes the decision for you. You should consider the 32-bit version only when you need to run legacy applications or wish to install the distribution on a legacy server. With this discussion out of the way, we can move on to actually getting some Ubuntu Server installations going. Go ahead and download the Ubuntu Server ISO image from the Ubuntu website and then follow along with the relevant section for creating a bootable flash drive with your OS.

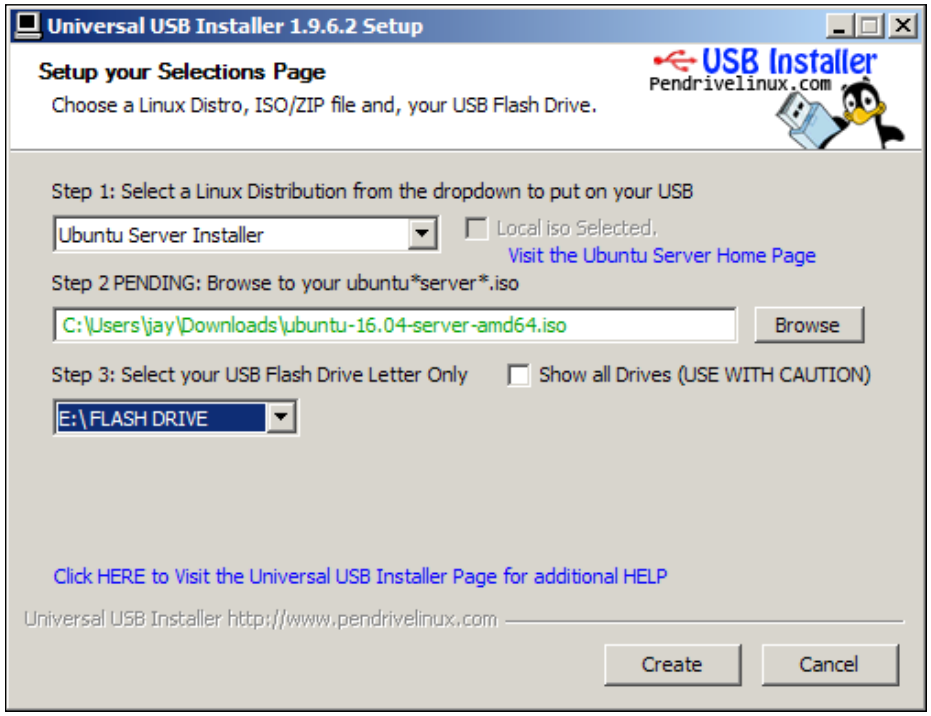
Creating a bootable Ubuntu Server flash drive (Windows)

In this section, I'll walk you through the procedure for creating a bootable flash drive for Ubuntu Server for those of you currently running Windows on your workstation:

1. To get started, grab a flash drive that's 1 GB or larger and insert it into your PC. You'll also need to download the Universal USB installer from the pendrivelinux website. You can find it at `http://www.pendrivelinux.com/universal-usb-installer-easy-as-1-2-3`. This executable doesn't actually install anything on your system, it will run from wherever you saved it.

www.it-ebooks.info

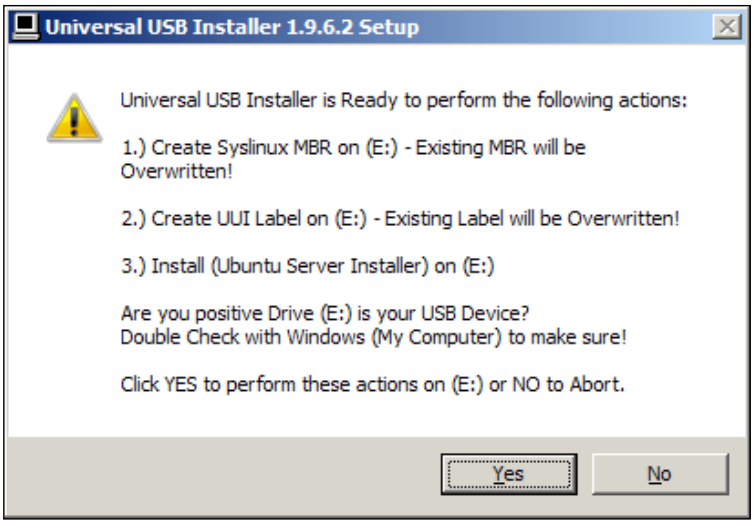
2. Once the application is open, click on the **Step 1** drop-down box and select **Ubuntu Server Installer** from the list of available Linux distributions. Next, click on **Browse** and select the Ubuntu Server ISO image you downloaded earlier. Then, select your flash drive in the **Step 3** drop-down box and then click on **Create**. When you've selected all your options, your application should look like this:



Utilizing the Universal USB installer to create a bootable flash drive in Windows

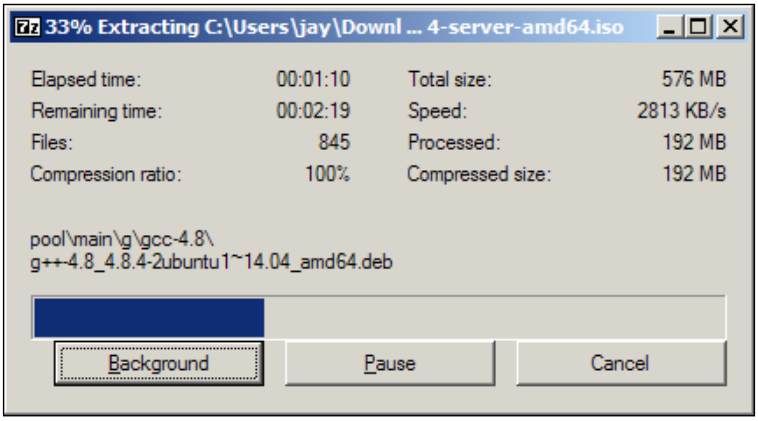
3. At this point, you'll see a confirmation dialog box appear, giving you one last chance to abort the process in case you selected an incorrect drive. Double-check that you've selected the proper drive and then click on **Yes**:

www.it-ebooks.info



Confirming your selections for creating a bootable flash drive

4. Next, you'll see a new dialog box open that will show you the progress as your flash drive is formatted and converted:



Flash drive creation in progress

This process will take about 5 minutes or so to complete, depending on the speed of your flash drive and your computer. Once it's finished, remove the flash drive from your PC and you should be good to go.

www.it-ebooks.info

It's a good idea to keep the Universal USB installer around even after you're finished using it, as it is a handy tool for creating bootable media for more than just Ubuntu Server. In fact, all of the distributions within the Ubuntu family are featured, as well as others such as OpenSUSE and Fedora. If nothing else, you can use it to recreate your server media when the next point release comes out so that you're always installing the latest version.

Creating a bootable Ubuntu Server flash drive (Linux)

On Linux systems, we can use either `ddrescue` or `dd` to create our bootable media. In the case of `ddrescue`, you'll first need to install the package on your system. If your distribution is Debian-based, you should be able to install it with the following command:

```
# apt-get install gddrescue
```

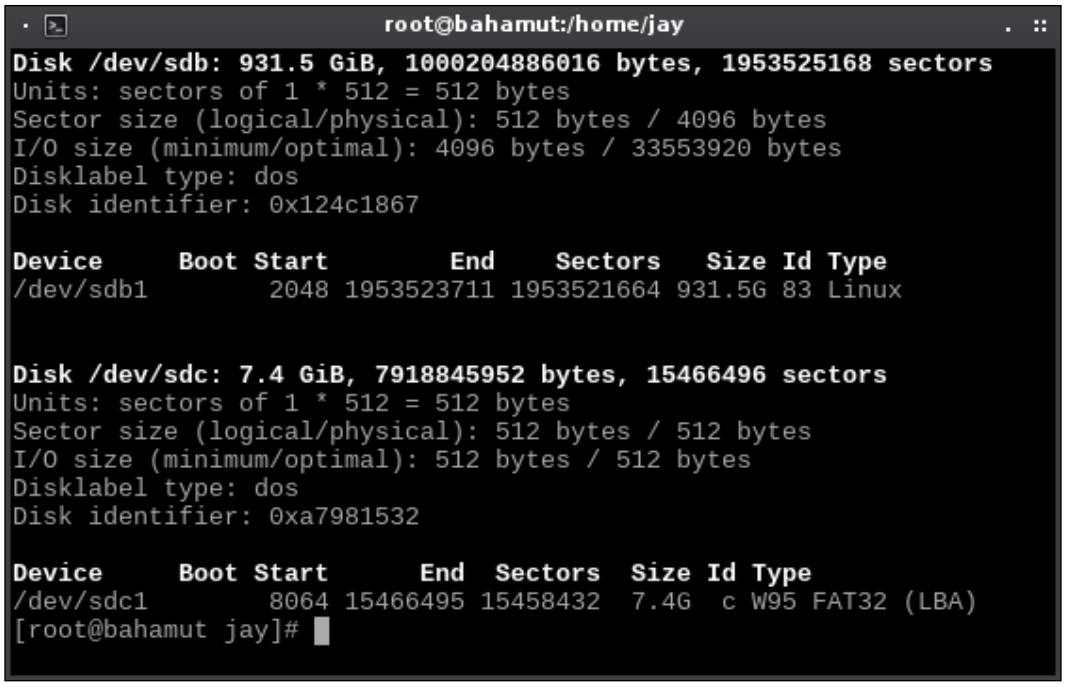
If your distribution is not Debian-based, use your distribution's package manager to install `gddrescue`. If this package is not available to you in your chosen platform, you can always fall back to the `dd` command, which I will also give you in this section.

With the `gddrescue` package installed, we can move on. Make sure the flash drive is inserted into your PC. Next, we'll need to run the following command in order to determine the name the system has provided for our flash drive:

```
# fdisk -l
```

From the output, you should be able to deduce which of the listed drives refers to your flash drive. In my case, `/dev/sdc` is mine. I know this because the partition `/dev/sdc1` is formatted as `FAT32`, and I don't have any other partition formatted this way:

www.it-ebooks.info



Output from the fdisk -l command

If you are at all unsure which drive is which, you may consider executing the `fdisk -l` command before and after inserting your flash drive and comparing the results. For the remaining examples in this section, I will use `/dev/sdc` for the commands. However, please make sure that you change this to match however your flash drive is named. If you format the wrong drive, you'll lose data (or worse, end up with a PC that won't start the next time you turn it on). Pay careful attention here!

Now that we know the device name for our flash drive, we can begin creating our media. If you have `ddrescue` available to you, use the following command:

```
# ddrescue -d -D --force <path and file name of the ISO> /dev/<device name>
```

For example, in my case, the command is the following:

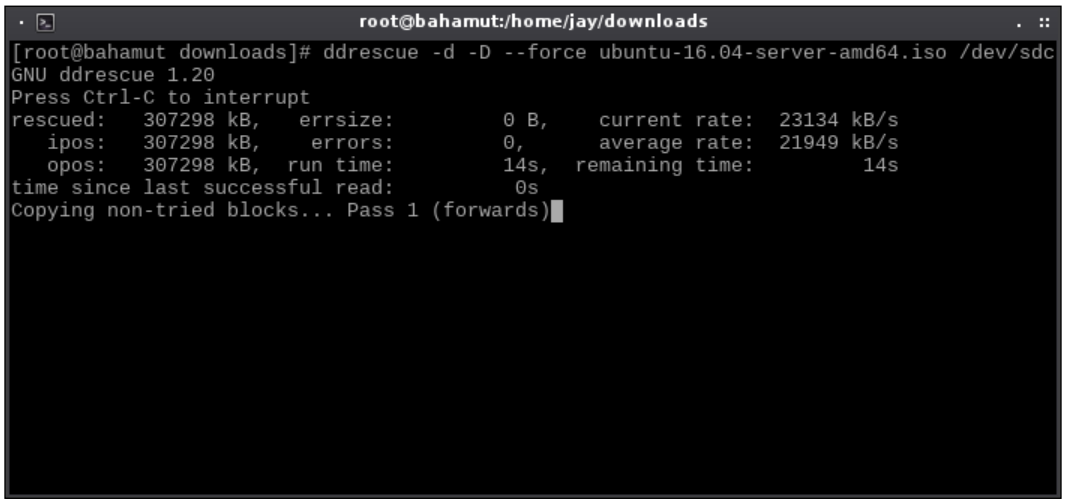
```
# ddrescue -d -D --force /home/user/downloads/ubuntu-16.04-server-amd64.iso /dev/sdc
```

Be sure to replace `/dev/sdc` with the device name of your flash drive and the name of the ISO with the filename and path of the Ubuntu Server ISO you downloaded earlier.

www.it-ebooks.info

With the example `ddrescue` command I provided, pay special attention to the fact that I did not identify a partition for the `dd` command. For example, I didn't use `/dev/sdc1`, I used simply `/dev/sdc`. This is important—the flash drive will not be bootable if you targeted a specific partition. After all, the ISO image is an image of an entire disk, not just a single partition.

As for the options I chose, I'm choosing direct disk access for input and output (the `-d` and `-D` flags), forcing our device to be overwritten (the `--force` flag), and using the full path and file name of our downloaded ISO and directing the output to `/dev/sdc`. Once you start the process, it shouldn't take very long to complete. When it's finished, you're ready to begin using it to install Ubuntu Server:



ddrescue in the process of creating a bootable flash drive

For those of you without `ddrescue` available, you can use `dd` instead:

```
# dd if=/home/user/downloads/ubuntu-16.04-server-amd64.iso of=/dev/sdc
bs=1M; sync
```

Similar to the `ddrescue` command, in the example for `dd` I'm using the path and filename of the Ubuntu Server ISO image as the input file (`if=`) and directing the output file (`of=`) to be that of my flash drive (`/dev/sdc`) and a block size of 1 MB (`bs=1M`). Unlike `ddrescue`, we won't see any fancy output with `dd`, but the process should work just fine. Once finished, we'll be ready to plan our partition layout and get our installation started.

www.it-ebooks.info

Creating a bootable Ubuntu Server flash drive (Mac)

The process for creating bootable USB media with Mac OS X is very similar to that of Linux, but there's a little preparation to do first. After you download the ISO, as mentioned earlier, we will need to convert the ISO file to the `IMG` format and then write the media.

First, let's convert the downloaded ISO image.

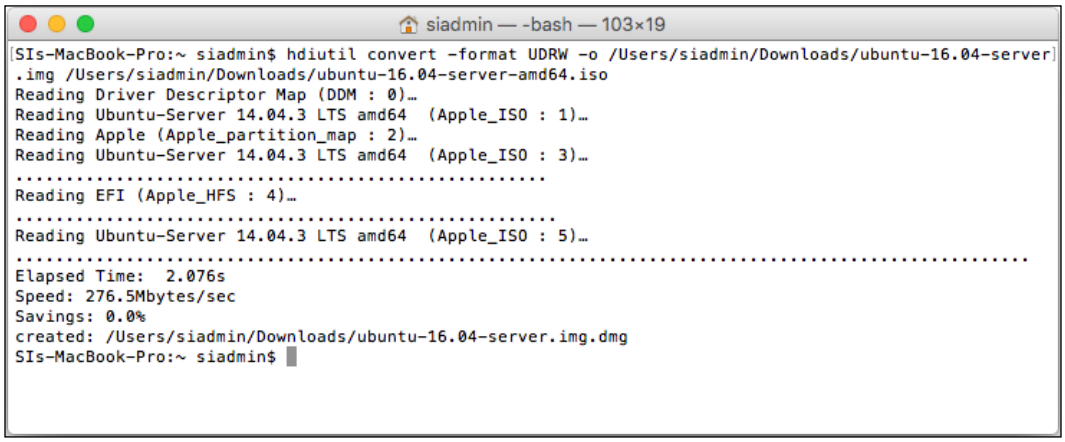
Open a terminal and then execute the following command:

```
hdiutil convert -format UDRW -o <path_to_save_IMG_file> <path_to_ISO_
image>
```

In my case, the command is as follows:

```
hdiutil convert -format UDRW -o /Users/jay/ubuntu_server.img
/Users/jay/Downloads/ubuntu-16.04-server-amd64.iso
```

All you should need to do is change the paths accordingly. First, we're choosing the path we want the created `IMG` file to be saved to, followed by the path to the Ubuntu ISO. The following screenshot shows an example run of this process:



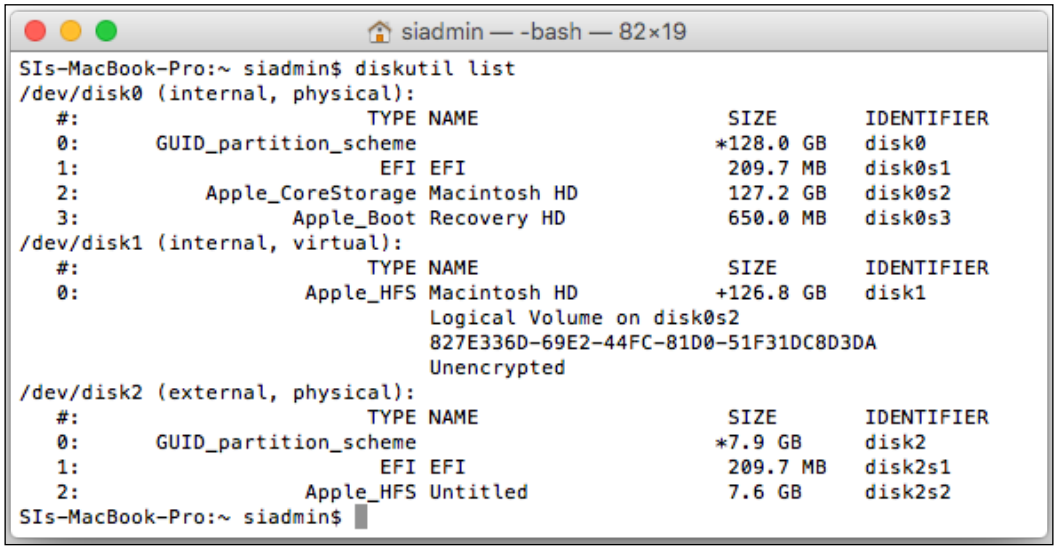
Converting the Ubuntu Server ISO to IMG format

With the conversion out of the way, we can run the following command to show information regarding the disks which are currently attached to the system. We need to do this so we know the device name OS X has given our USB drive, since we'll be referring to it later:

```
diskutil list
```


www.it-ebooks.info

In the example screenshot, you can see what the process looks like on the system I was using. In my case, I know my USB media is `/dev/disk2`, as it's the only thing attached to the Mac, which is roughly 8 GB in size. In your case, double-check the output of the `diskutil list` command and make a note of which entry is referring to your USB media. If in doubt, run the command before and after inserting your flash drive and compare the results:



Listing attached disks on a Mac system

Now that we know the device name of your USB media, we can continue with the final command and perform the actual creation of our Ubuntu Server media. But first, we should make sure that the flash drive is not mounted:

```
diskutil unmountDisk /dev/disk2
```

Be sure to change the device name to match the one you saw in the results of the `diskutil list` command you performed earlier.

The following command will create our actual media:

```
# sudo dd if=<path_to_IMG_file> of=<device_node_of_usb_media> bs=1m
```

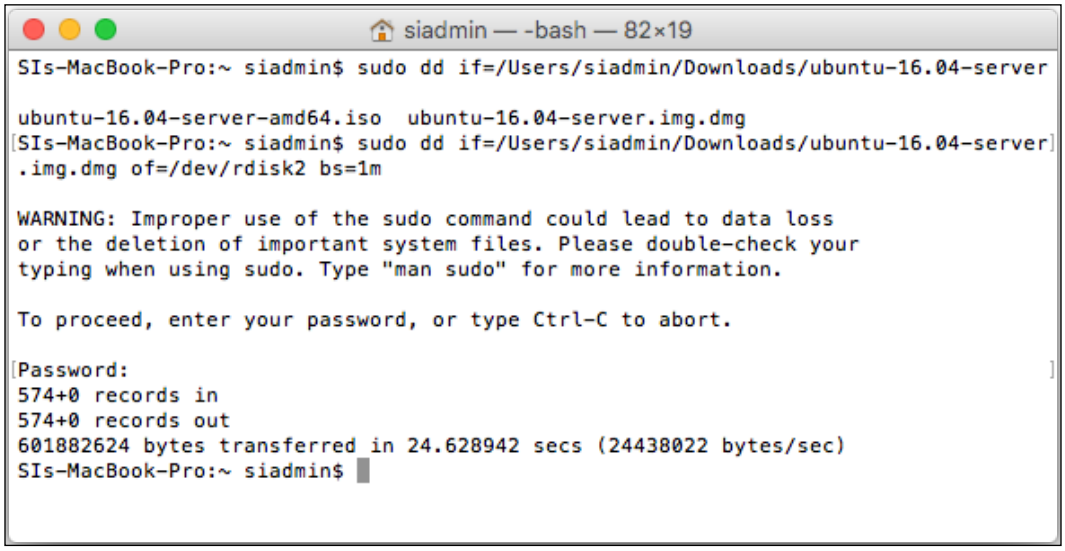

www.it-ebooks.info

For example, using the paths and device name on my system, the command would become the following:

```
# sudo dd if=/Users/jay/ubuntu_server.img.dmg of=/dev/rdisk2 bs=1m
```

Here, I'm using the `sudo` command, which means that I want to run the command using `root` privileges. From this point forward, I'll prefix commands with # if you need to run the commands with root privileges. It doesn't matter if you use `sudo` or switch to `root` whenever you see this mark. (Just keep in mind that commands prefixed with # needs to be executed either with `root` or `sudo`). Even though this section involves using a Mac, I thought it would be worth bringing up.

The next screenshot shows an example run of this `dd` command:



Writing the Ubuntu Server IMG file to a USB flash drive

www.it-ebooks.info

Now that we have our media, you can remove it from your Mac and continue on and get some installations going. Before we get into that, though, we should have a quick discussion regarding partitioning.

Planning the partition layout

One of the many joys of utilizing Linux for server platforms is how flexible the partitioning is. Partitioning your disk allows you to segregate your storage in order to dedicate specific storage allocations to individual applications. For example, you can dedicate a partition for the files that are shared by your Apache web server so changes to other partitions won't affect it. You can even dedicate a partition for your network file shares—the possibilities are endless. The names you create for the directories where your partitions are mounted are arbitrary; it really doesn't matter what you call them. The flexible nature of storage on Linux systems allows you to be creative with your partitioning as well as your directory naming.

With custom partitioning, you're able to do some very clever things. For example, with the right configuration you're able to wipe and reload your distribution while preserving user data, logs, and more. This works because Ubuntu Server allows you to carve up your storage any way you want during installation. If you already have a partition with data on it, you can choose to leave it as is so you can carry it forward into a new install. You simply set the directory path where it's mounted the same, restore your configuration files, and your apps will continue working as if nothing happened.

One very common example of custom partitioning in the real world is separating the `/home` directory into its own partition. Since this is where users typically store their files, you can set up your server such that a reload of the distribution won't disturb user files. When they log in after a server refresh, all their files will be right where they left them. You could even place the files shared by your Apache web server onto their own partition and preserve those too. The possibilities are endless.

www.it-ebooks.info

Another reason to utilize separate partitions may be to simply create boundaries or restrictions. If you have an application running on your server that is prone to filling up large amounts of storage, you can point that application to its own partition, limited by size. An example of where this might be useful is an application's log files. Log files are the bane of any administrator when it comes to storage. While helpful if you're trying to figure out why something crashed, logs can fill up a hard disk if you're not careful. In my experience, I've seen servers come to a screeching halt due to log files filling up all the available free space on a server where everything was on a single partition. The only boundary the application had was the entirety of the disk.

While there are certainly better ways of handling excessive logging (log rotating, disk quotas, and so on), a separate partition also would have helped. If the application's log directory was on its own partition, it would be able to fill up that partition, but not the entire drive. A full log partition will certainly cause issues, but it wouldn't have been able to affect the entire server. As an administrator, it's up to you to weigh the pros and the cons and develop a partitioning scheme that will best serve the needs of your organization.

Running an efficient server is a matter of efficiently managing resources, users, and security—and a good partitioning scheme is certainly part of that. Sometimes it's just a matter of making things easier on yourself so that you have less work to do should you need to reload your operating system. For the sake of following along with this book, it really doesn't matter how you install or partition Ubuntu Server. The trick is simply to get it installed—you can always practice partitioning later. After all, part of learning is setting up a platform, figuring out how to break it in epic ways and then fixing it up.

When we perform our installation in the next section, you'll have an opportunity to partition your system any way you'd like. However, you don't have to—having everything in one partition is fine too, depending on your needs.

www.it-ebooks.info

Installing Ubuntu Server

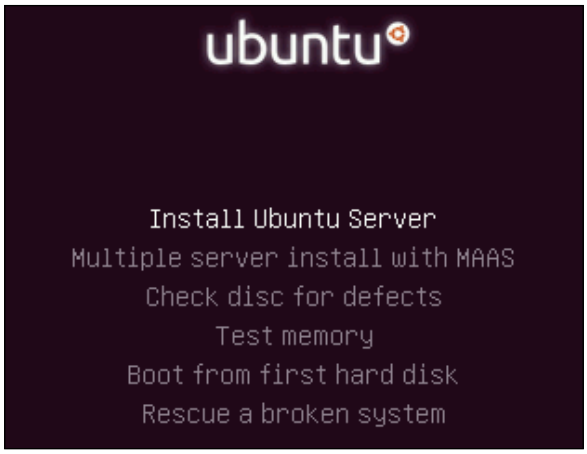
Now, regardless of your platform, we should be ready to get an installation or two of Ubuntu Server going. In the steps that follow, I'll walk you through the process.

1. To get started, all you should need to do is insert the media into your server or device and follow the onscreen instructions to open the boot menu. The key you press at the beginning of the POST process differs from one machine to another, but it's quite often *F10*, *F11*, or *F12*. Refer to your documentation if you are unsure, though most computers and servers tell you which key to press at the beginning. You'll probably miss this window of opportunity the first few times, and that's fine—to this day I still seem to need to restart the machine once or twice to hit the key in time. Once you successfully boot from the media, it should ask you to select your language. Use your arrow keys to change the selection, then press *Enter*:

Language selection at the beginning of the installation process

www.it-ebooks.info

2. Next, we'll be brought to the installation menu, where we'll have a few options. We can get right into the installation process by pressing *Enter* to choose the first option (**Install Ubuntu Server**), though a few of the other options may be useful to you:



Main menu of the Ubuntu installer

First, this menu also offers you an option to **Check disc for defects**. Although I'm sure you've done well in creating your media, all it takes is for a small corruption in the download or with the `dd` command and you would end up with invalid media. This is extremely rare—but it does happen from time to time. If you have the extra time, it may make sense to verify your media.

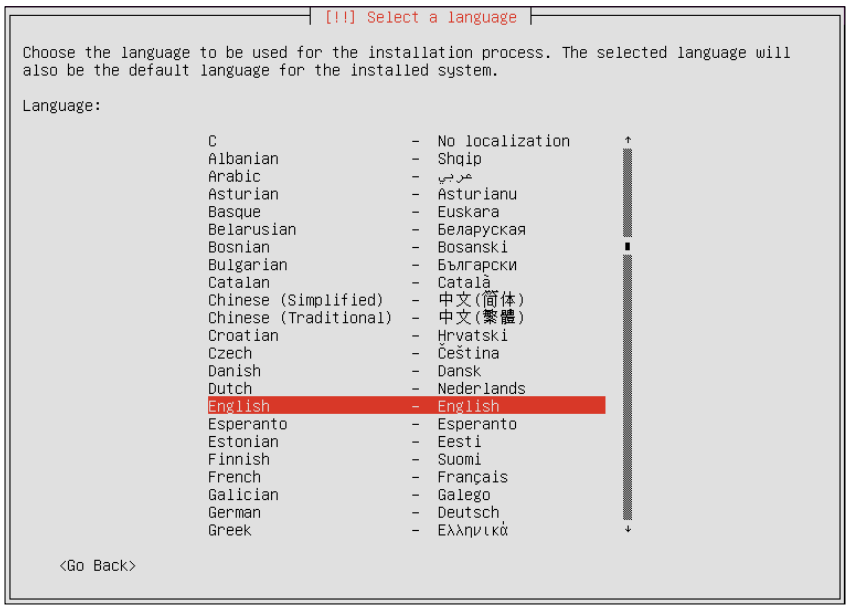
If boot media created from your PC is constantly corrupted or invalid, try creating the media from another machine and test your PC's memory. You can test memory right from the Ubuntu Server installation media.

www.it-ebooks.info

Second, this menu gives you the option to **Test memory**. This is an option I've found myself using far more often than I'd like to admit. Defective RAM on computers and servers is surprisingly common and can cause all kinds of strange things to happen. For one, defective RAM may cause your installation to fail. Worse, your installation could also succeed—and I say that's worse due to the fact you'd find out about problems later rather than right away. While doing a memory test can add considerable time to the installation process, I definitely recommend it, especially if the hardware is new and hasn't ever been tested at all. In fact, I make it a process to test the RAM of all my servers once or twice a year, so this media can be used as a memory tester any time you need one. The **Rescue a broken system** option is also useful to recover systems that, for some reason, are unable to boot at all.

From this point forward, we will progress through quite a few screens in order to customize our installation. I'll guide you along the way.

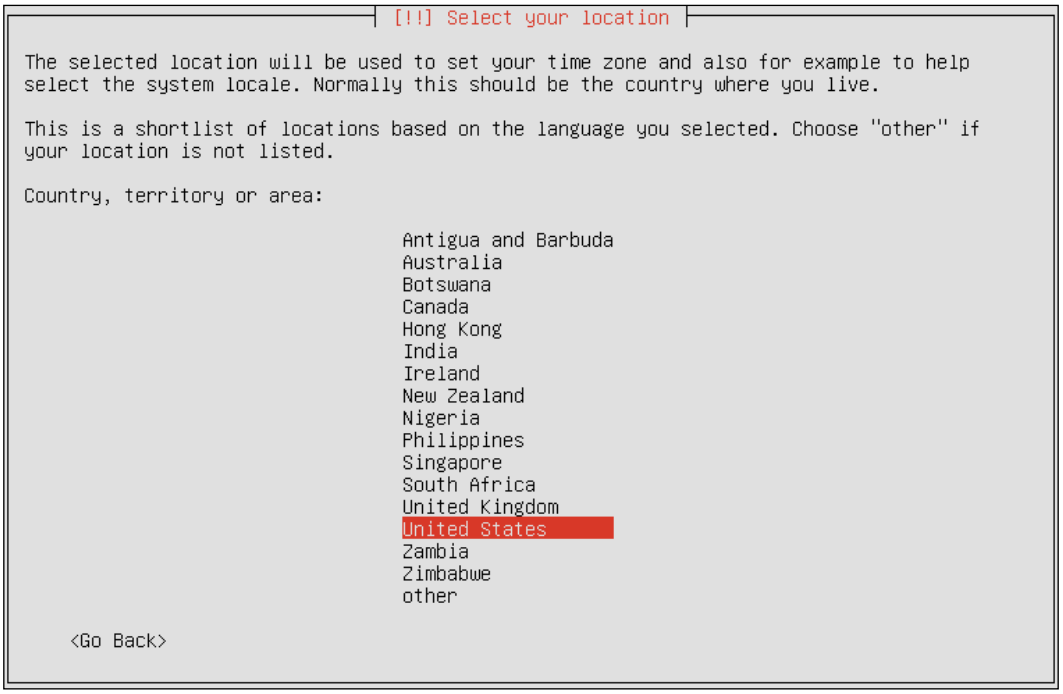
3. To navigate the installer, you simply use the arrow keys to move up and down to select different options, press *Enter* to confirm choices, and you press *Tab* to select between different buttons. The installer is pretty easy to navigate once you get the hang of it. Anyway, after you enter the actual installation process, the first screen will ask you to **Select a language**. Go ahead and do so. Please note that this selection refers to the language used for the installation process, not necessarily the language that will be used by default on your installation:



Language selection screen

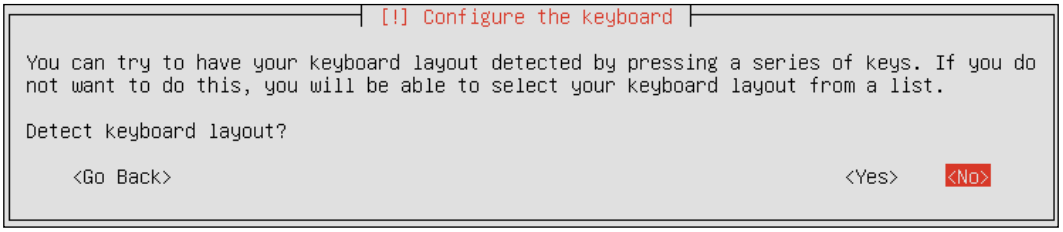
www.it-ebooks.info

4. On the **Select your location** screen, choose the area that is closest to you. As the installer notes, this will set your time zone and system locale:



Selecting the system locale and time zone

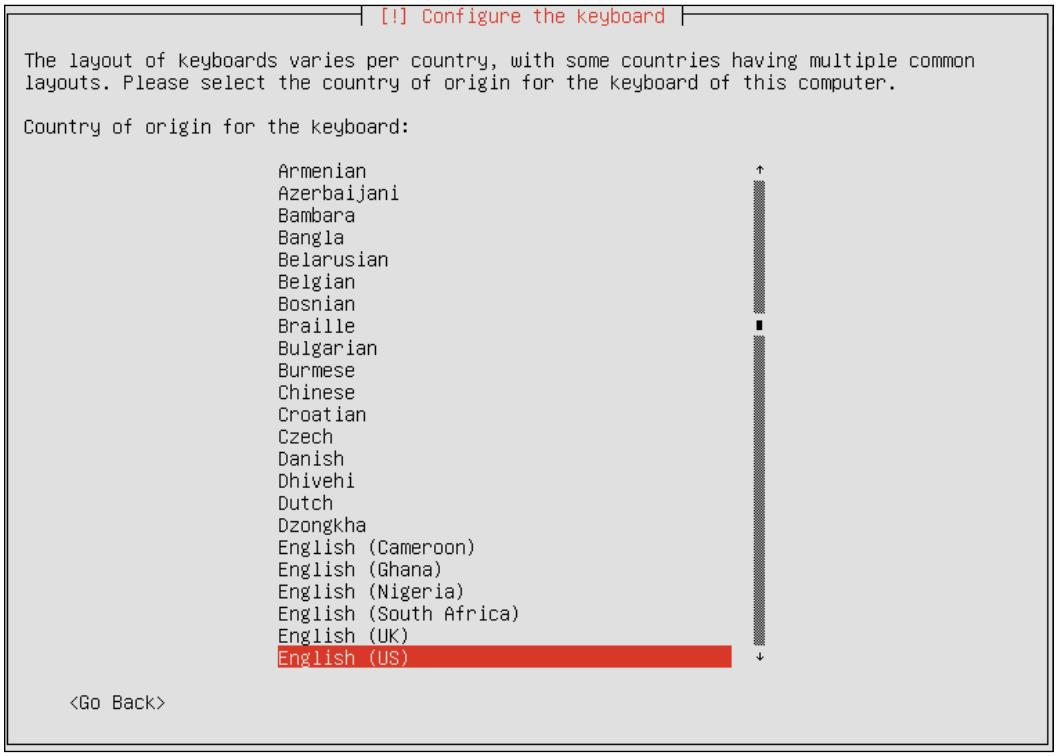
5. On the **Configure the keyboard** screen, you'll be given an option to have the installer automatically detect your keyboard. I recommend choosing **No** because it's just as easy (if not faster) to select it yourself. If you choose **No**, the next screen will ask you for your keyboard type.



Keyboard configuration during the Ubuntu Server installation procedure

www.it-ebooks.info

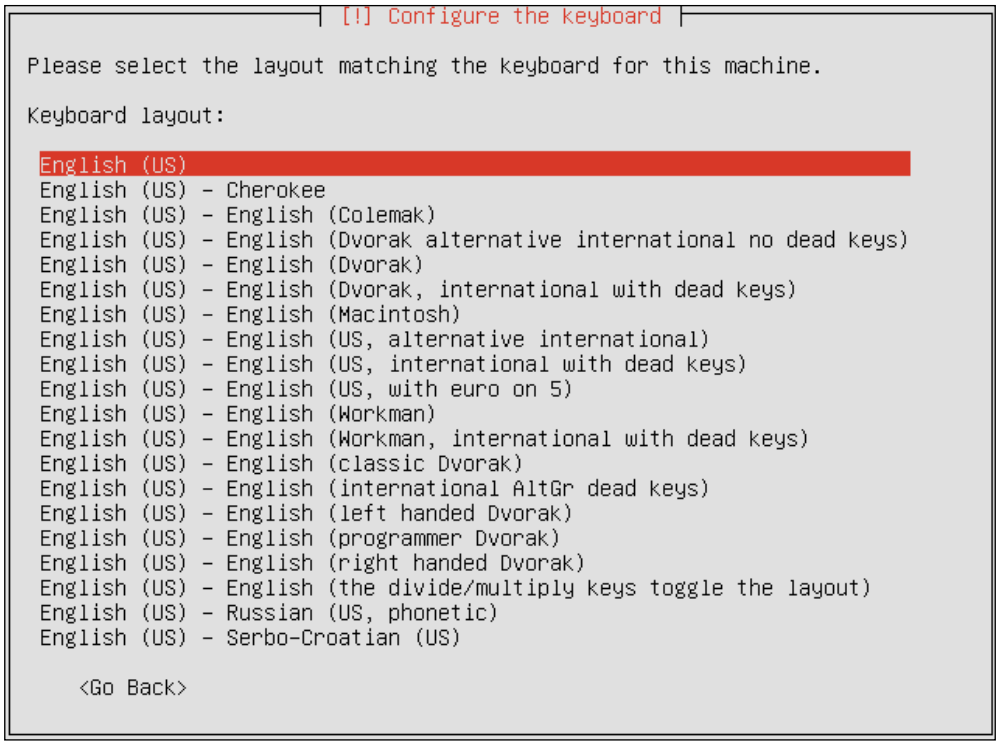
6. On the next screen, I chose **English (US),** but make sure you choose the country of origin that best represents your keyboard:



Selecting the country of origin for your keyboard

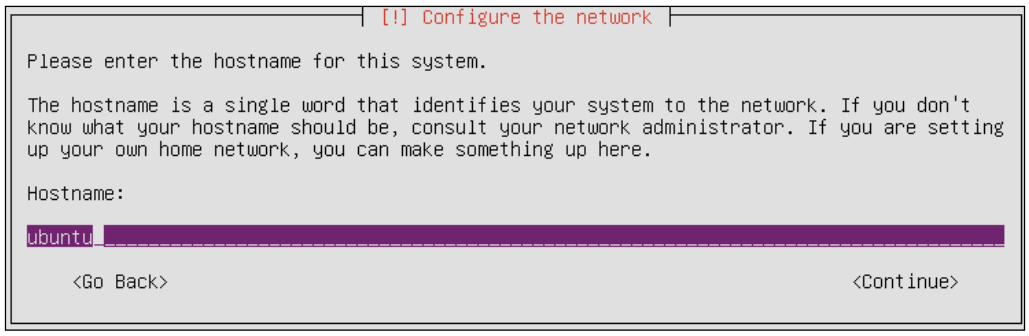
7. Next, you choose your actual keyboard layout. Once you've highlighted yours, press *Enter* to continue:

www.it-ebooks.info



Selecting the keyboard layout

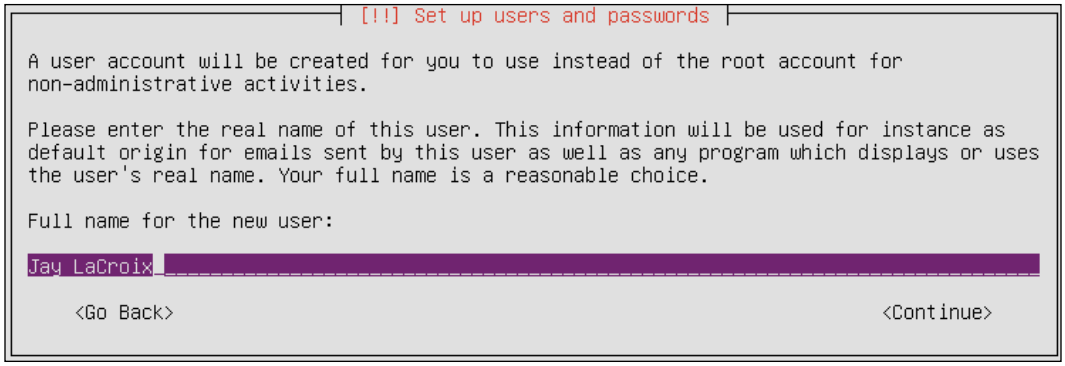
8. After several progress bars complete, you'll be brought to the **Configure the network** screen, where the installer will ask you to choose a **Hostname** for your server. The hostname is basically the name that will appear in your command prompt and on your network if you open up file shares to others. Enter whatever you'd like for this and then choose **Continue**. If you're indecisive, we can always change this later (and I'll show you how to do so in future chapters).



Entering in the host name for the new server

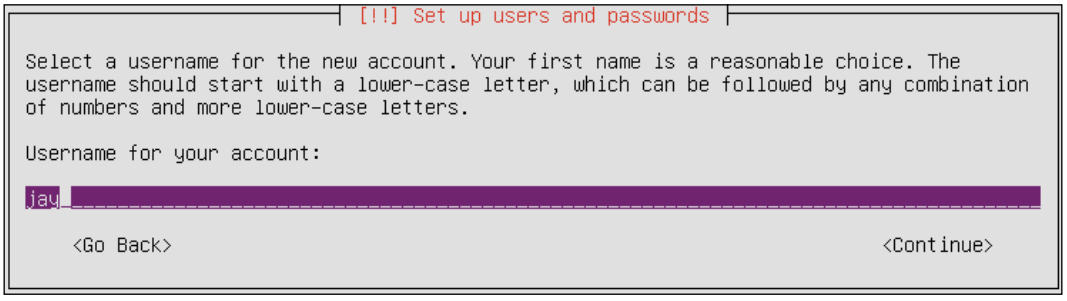
www.it-ebooks.info

9. On the **Set up users and passwords** screen, the installer will ask you for the **Full name for the new user** that will be created. Unlike Debian (on which Ubuntu Server is based), the `root` account is disabled by default. We can always enable this later, but with the way Ubuntu Server sets up your system, the user account you create during the installation process will be given `sudo` rights, which will allow that user to perform commands as `root` with the `sudo` command. We'll discuss `sudo` in more detail in the next chapter, so don't worry if you've never worked with this before. Anyway, put in the full name of the first user and continue on:



Creating a system user for Ubuntu Server

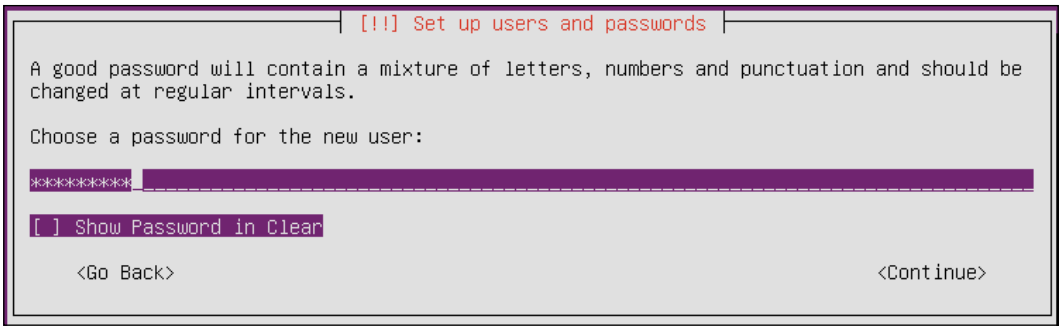
10. Next, the installer will ask you for the username for the `sudo` user. This will be the name you'll use to log in:



Selecting a user name

11. The next screen will ask you to enter and then confirm your password for our `sudo` user. Enter it both times, and we'll continue on:

www.it-ebooks.info



Entering in the password for the system user

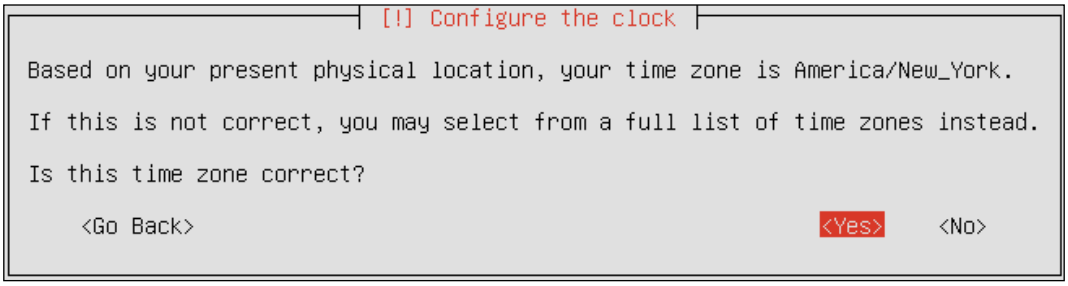
12. The final screen of the **Set up users and passwords** section will ask if you'd like to encrypt the home directory of your `sudo` user:

Choosing whether or not to encrypt the home directory for the system user

Although we'll talk more about encryption in *Chapter 12, Securing Your Server*, it's important to note that this is a bit different than doing full-disk encryption with software such as **Linux Unified Key Setup** (**LUKS**). Although there is a way you can go back and encrypt your home directory later, it's a relative pain. Therefore, if you'd like your home directory encrypted, you may as well do it now. Whether or not you do this is up to you, but if you do, it will make the contents of your home directory completely private should an unauthorized person get a hold of your server's hard disk. As a basic rule of thumb, if you're storing private data, encrypt. Otherwise, don't.

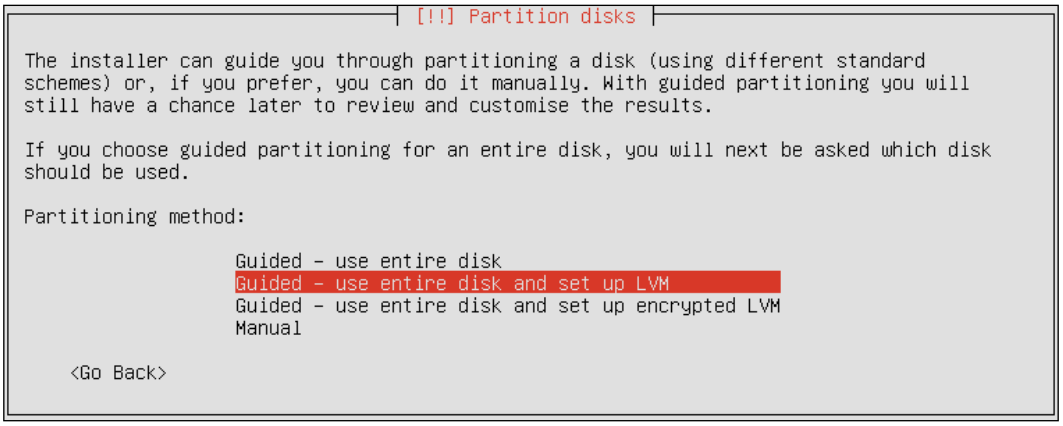
www.it-ebooks.info

13. On the next screen, the installer will recommend a time-zone based on your earlier choices. This is usually correct, but if it's not, you can select your time-zone from a list. With that out of the way, let's continue on.



Confirming time zone selections

14. Now comes the fun part, partitioning your system. As we discussed earlier in this chapter, creative partitioning can make your system easier to manage. Whatever you've decided in regards to partitioning, you'll implement that plan here.



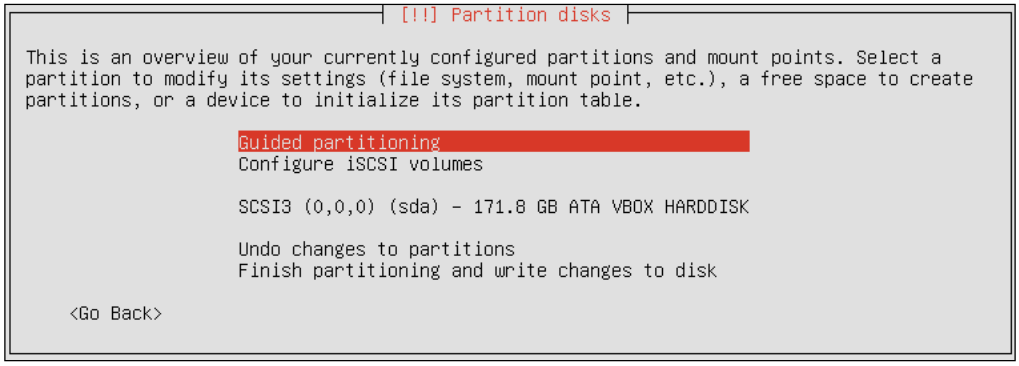
Choosing a partitioning method

15. If you want to get through the installer quickly and you don't plan on any partitioning, you can select **Guided – use entire disk** to just set up a single partition. If you do decide to use a single partition, you'll basically be skipping the next dozen or so steps, and you'll jump ahead to manual partitioning steps. In the next steps, we'll go through a bit of manual partitioning.

www.it-ebooks.info

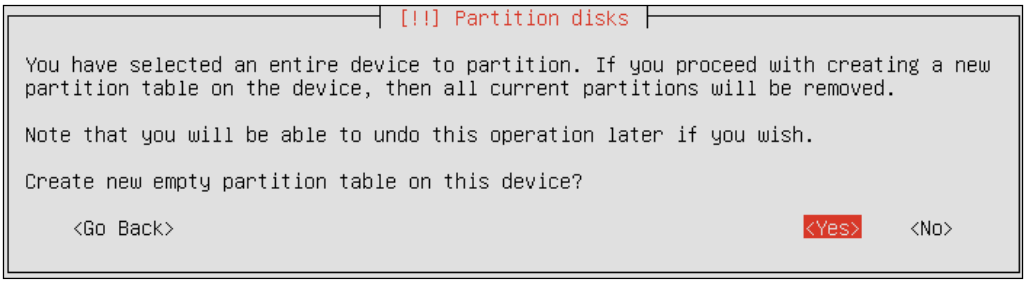
As an alternative to **Guided – use entire disk**, you can also choose **Guided – use entire disk and set up encrypted LVM**. Doing so will also allow you to skip the partitioning process but will encrypt your filesystem as well. This will help prevent people with physical access stealing information from the server if they boot it from live media. It's recommended that you do this if you plan on storing sensitive data.

16. This part of the installer is assuming you chose to manually partition your system (by choosing the **Manual** option). You should now see your hard disk on the list on this screen. In my case, my disk is listed as **SCSI3 (0,0,0) (sda) – 171.8 GB ATA VBOX HARDDISK** since I'm using a virtual machine to show the process. Yours will say something different, but it will be similar:



Main menu of the Ubuntu Server partitioner

17. If you've already installed an operating system on this disk before, you may see several partitions here. In my case, if I select the **VBOX HARDDISK,** the system will ask me whether or not I'd like to create an empty partition table. If the hard disk you're using has never been formatted before, you'll see the same thing. Feel free to choose **Yes** here to have a fresh partition table created if this screen appears:

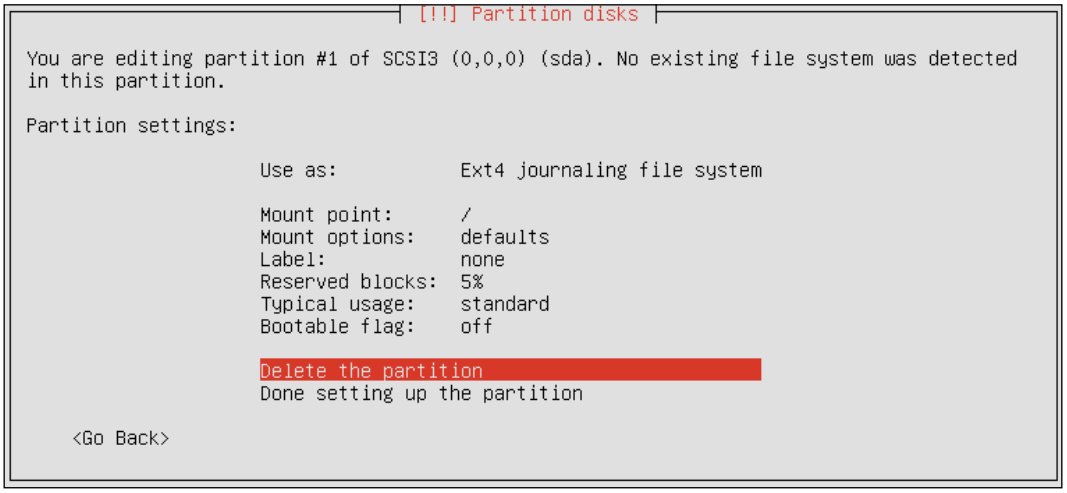


Creating a new partition table

www.it-ebooks.info

Assuming you already have partitions on your disk, you should delete them. It likely goes without saying, but you should always back up any important data on your machine that you care about before partitioning your disk.

18. To delete your current partitions, choose one by selecting it and pressing *Enter*, and you'll have an option to **Delete the partition** near the bottom of the list:



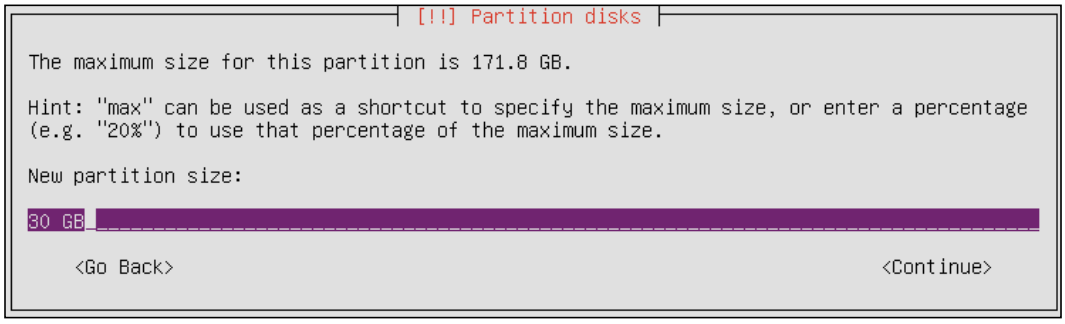
Deleting an existing partition

19. Now that you've deleted your existing partitions, you should only have empty space listed for your hard disk. Select this free space and choose to **Create a new partition**. To show how this process works, I'll walk you through setting up separate `/`, `/home`, and `swap` partitions. You don't have to follow this exact scheme; feel free to mix it up. You do need a `/` partition at a minimum, though:

Creating a new partition on our disk

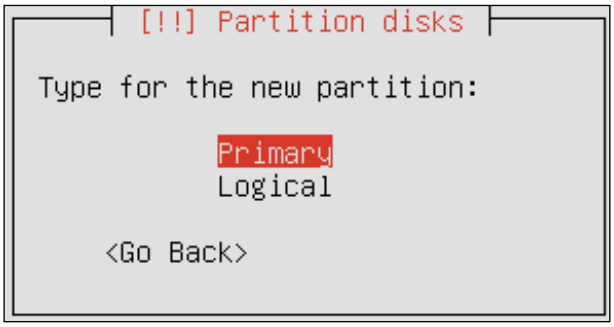
www.it-ebooks.info

20. After you choose to **Create a new partition**, you'll be prompted to enter the size for it. In my opinion, 30 GB is plenty for your `root` partition. Some opt for smaller, though I've found that eventually you'll find yourself having to keep a close eye on space after a year or two, and since hard disk space is cheap nowadays, 30 GB is reasonable, but the choice is yours.



Choosing the size of the new partition

21. You'll be prompted whether or not you'd like to create the partition as **Primary** or **Logical**. We'll always be choosing **Primary** each time we see this. Creating **Logical** volumes is a good idea only if you plan on creating more than four partitions.



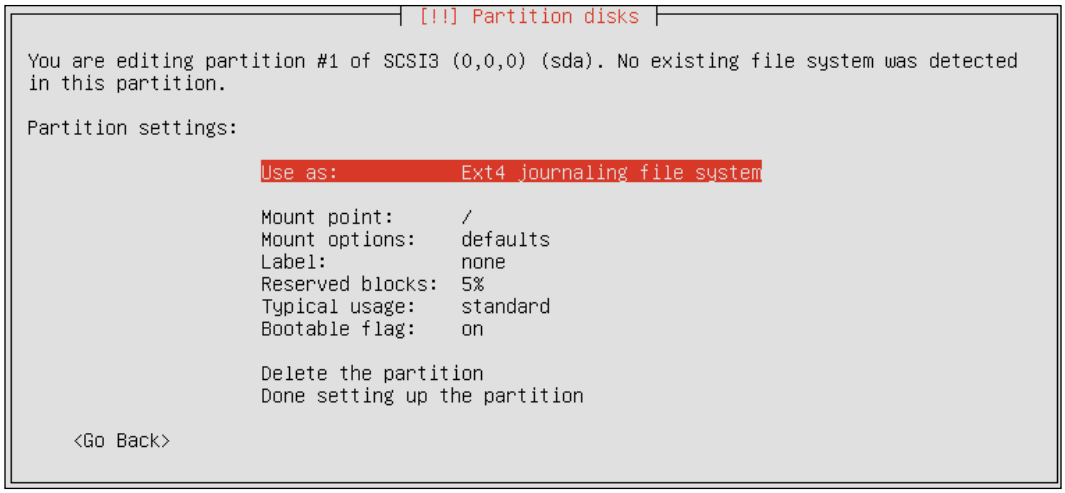
Choosing a partition type

www.it-ebooks.info

22. Next, you'll be asked whether or not you'd like the partition created at the **Beginning** or the **End** of the free space of your disk. Each time we see this, we'll choose **Beginning**:

Choosing to create the partition at the beginning or end of the free space

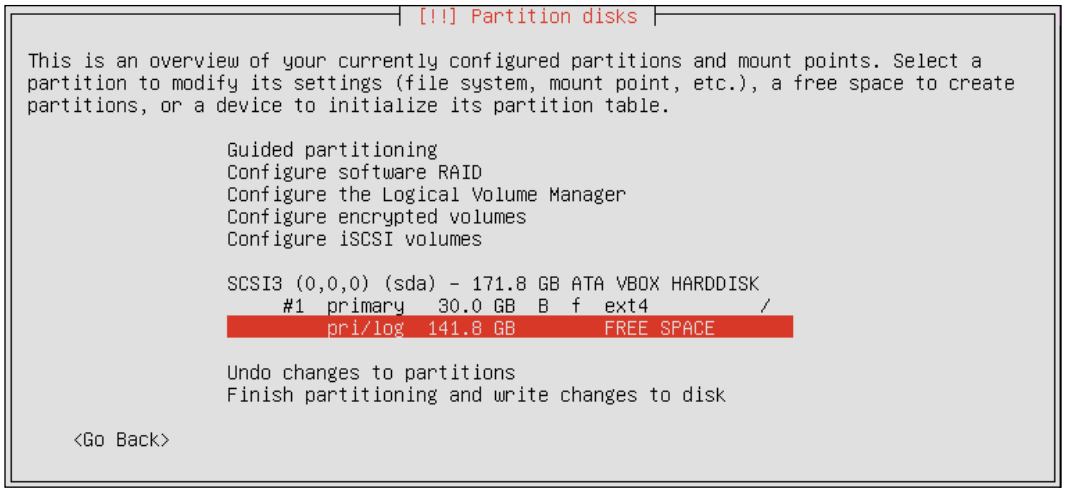
23. Next, we'll see a summary of our options, many with the defaults selected. In our case, make sure **Use as** is set to **Ext4 journaling file system**, the **Mount point** is set to **/**, and the **Bootable flag** is set to **on**. Compare your settings to the ones I've chosen in the following screenshot for reference. When finished, choose **Done setting up the partition**:



Creating the root partition for our installation

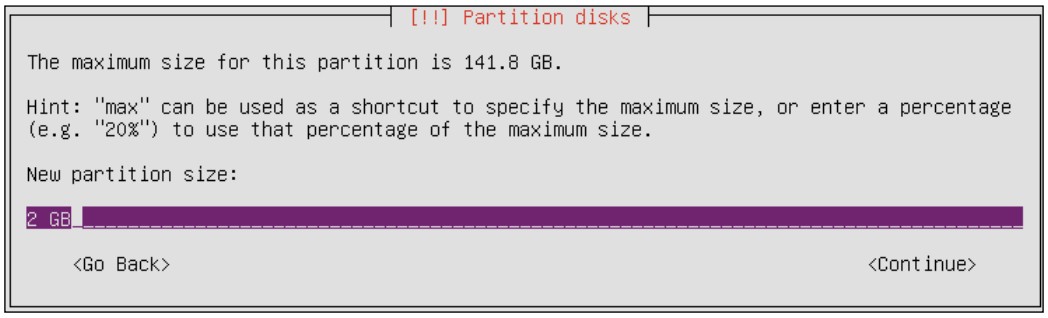
24. Next, we'll be taken back to the main **Partition disks** screen, where we should see the partition we created as well as some **FREE SPACE** remaining. Select **FREE SPACE**, and we'll set up our next partition:

www.it-ebooks.info



Preparing to create a second partition

25. In this case, we're setting up a partition for swap. The swap partition will be used if our server runs out of RAM, so hopefully we'll never need to use this partition much since the hard disk is slower than RAM. I think **2 GB** is healthy, but the choice is yours. Some will set this to 1.5 times the amount of RAM you have, which is also valid. If in doubt, choose a few gigabytes here, no less than 2:

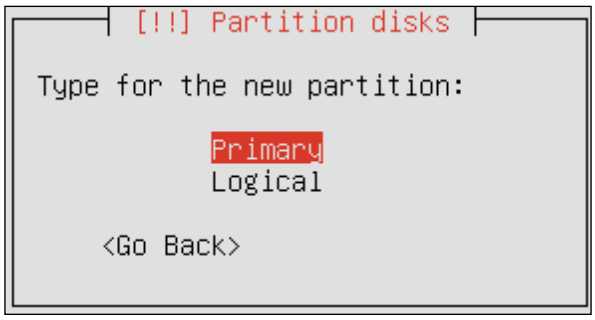


Selecting a size for our swap partition

www.it-ebooks.info

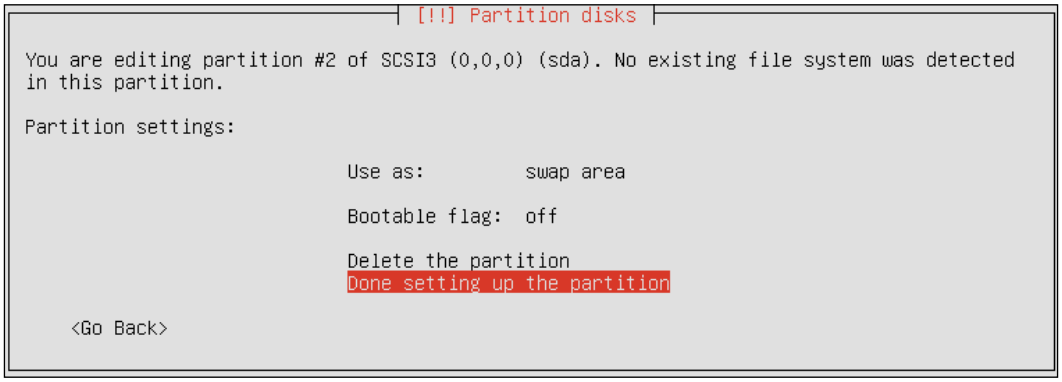
There is currently a lot of debate in the Linux community regarding whether or not swap is even necessary. Some administrators will even omit this altogether. While I won't get into that debate here, I really don't see the point in disregarding swap. After all, if you have even an average hard disk, how much of a difference would it really make if you save a few gigabytes? In my opinion, it doesn't hurt anything to create swap and it may even save you one day.

26. Again, we'll choose **Primary** for the type:



Choosing primary or logical for the swap partition

27. Ensure **Use as** is set to **swap area**, then select **Done setting up the partition** to continue:

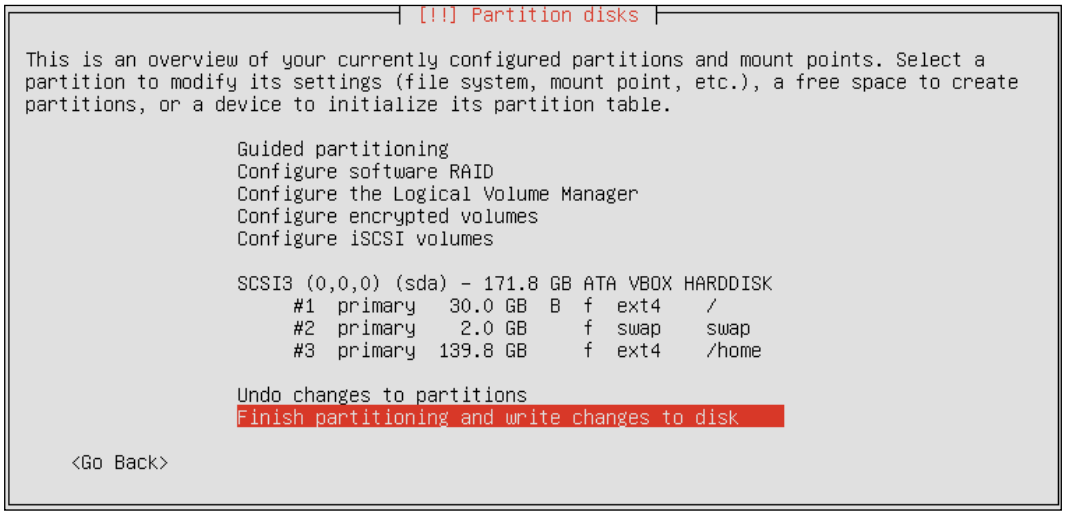


Finishing the creation of our swap partition

www.it-ebooks.info

With the swap partition created, go ahead and repeat the process to create a partition for `/home`. If you're following my partition scheme in this example, go ahead and allocate the remainder of the drive for `/home`. All the other options should be the same as before.

28. Once you have our three partitions created, select **Finish partitioning and write changes to disk** to continue on. Refer to the following screenshot for a comparison of all the options I've chosen during my installation:

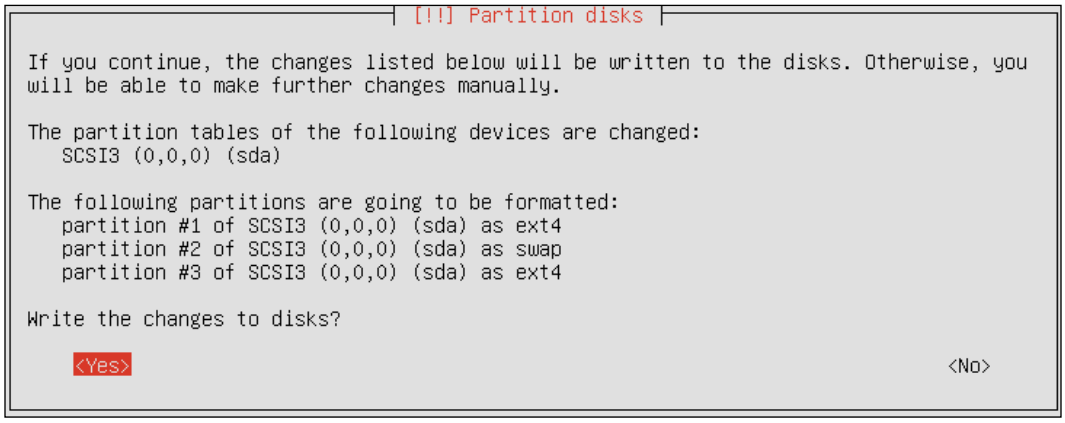


Previewing all of our partition selection so far

In case you're wondering why I partition this way, I think it makes more sense to create the partition most likely to need growth as the last one. In our case, I created `/home` last. This is so if I need to upgrade the storage later, all I would need to do is resize `/home` on the new disk. If I had created the `swap` partition last (as most Linux installers do), I would have to delete (or move) `swap` and then recreate it again afterwards. It's all about saving time in the long run. If I were creating a web server, I would've used the same style but instead of `/home`, I would've mounted the last partition in a directory like `/srv/www` instead.

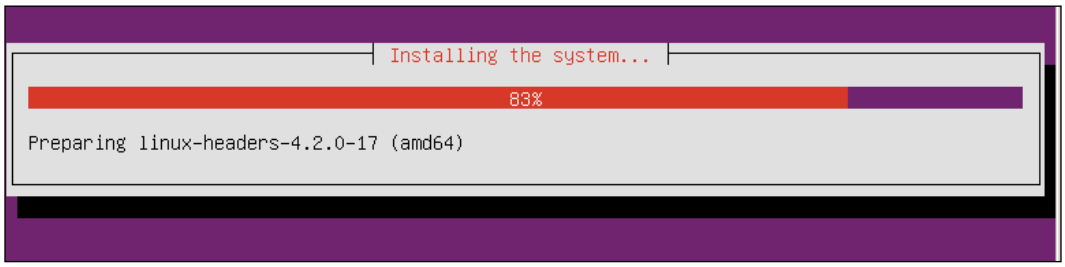
www.it-ebooks.info

29. Next, the following screen will confirm our selections one last time before the new partitions are written to disk:



Confirming our choices one last time

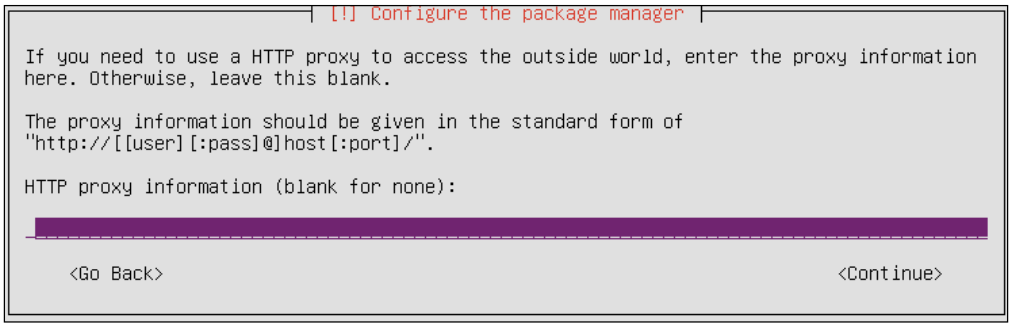
30. After the partitions are created, Ubuntu Server will be installed on your hard disk. This process should pass relatively quickly depending on the speed of your machine. You'll see a progress bar appear, detailing its progress:



Installing the base system

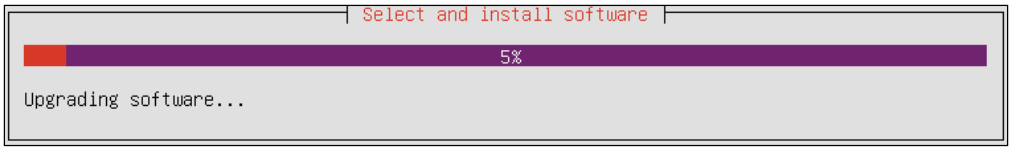
31. Once all the packages have been installed for the base system, the installer will ask you if you need a proxy to access the Internet. If you need one, enter it here. Otherwise, leave it blank and select **Continue**:

www.it-ebooks.info



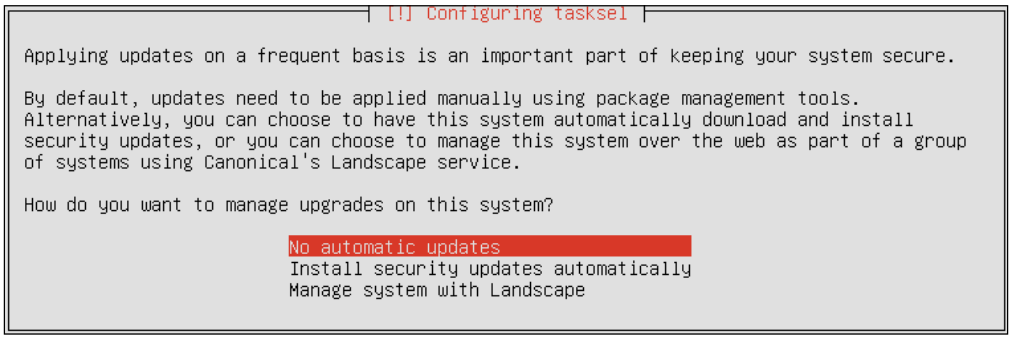
Proxy prompt during the Ubuntu Server installation

32. Another progress bar will scroll by, which will prepare a selection of additional software you can choose to install:



Preparing to install additional software

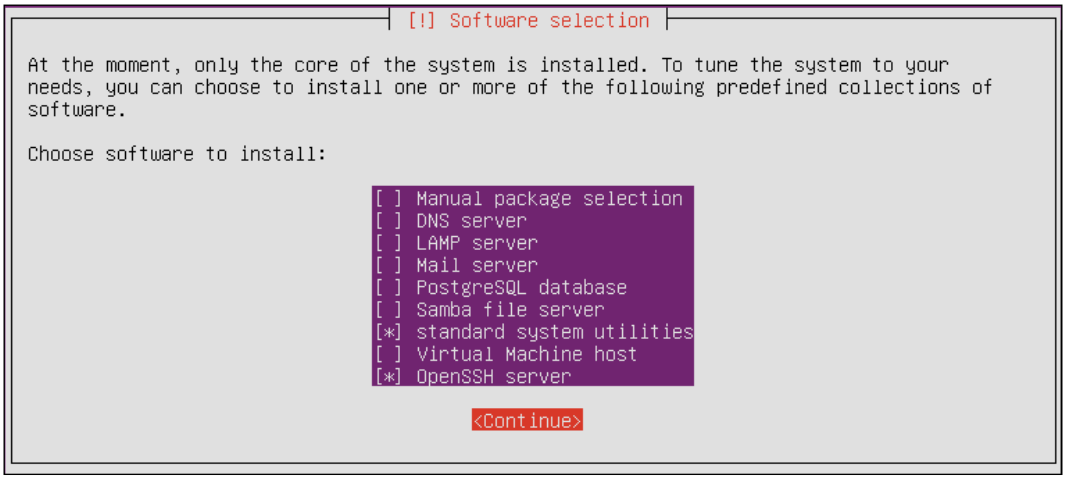
33. On the **Configuring tasks** screen, you'll be asked whether or not you'd like to enable automatic updates. Personally, I always choose **No automatic updates** here, because I like to handle this myself by writing a script. It's up to you, though. Be careful, because an updated application may become incompatible with custom configurations, which can be a disaster for production later. Security updates are definitely something you want to stay on top of, so if you don't enable automatic updates, make sure you at least come up with your own solution later. We'll discuss updating your system during *Chapter 5, Managing Software Packages,* so don't worry if you don't have an idea on how to handle updates just yet.



Deciding between automatic and non-automatic security updates

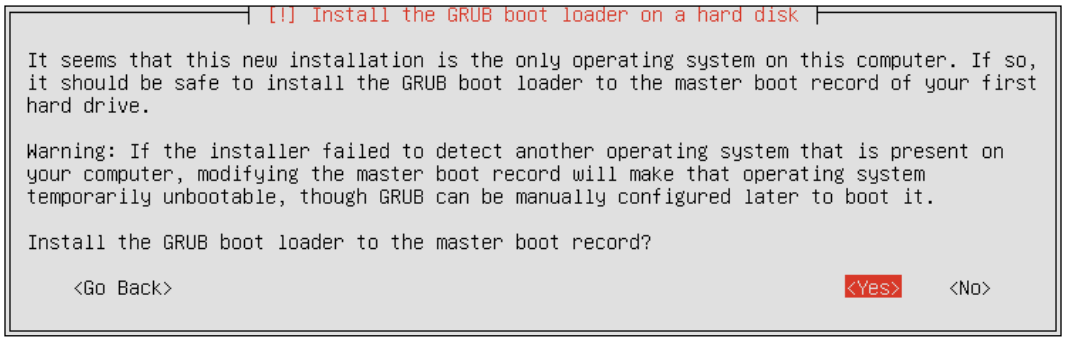
www.it-ebooks.info

34. On the next screen, you'll be given some additional software collections you can install if you want to. You'll have the option to set up an **OpenSSH server**, **Mail server**, **LAMP server**, and more. Most of these options we'll set up manually further on into the book, so I suggest selecting only **OpenSSH server** for now. Leave **standard system utilities** checked:



Selecting additional software

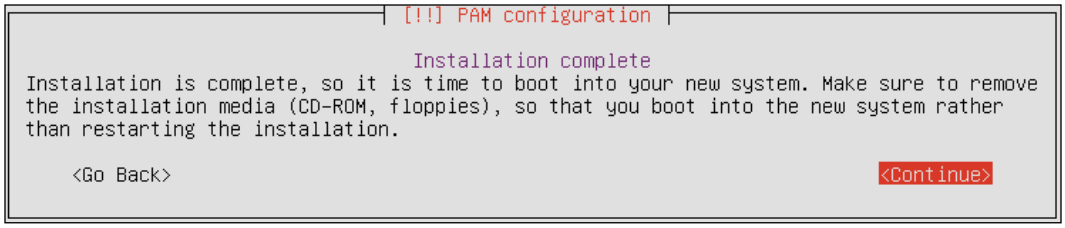
35. The next screen that appears will offer us the option to **Install the GRUB boot loader on the master boot loader**. You should always say **Yes** to this prompt, unless you have your own process to boot the system you wish to implement. For 99.99% of you, you'll definitely want to do this:



Confirming boot loader installation

www.it-ebooks.info

36. Finally, our installation is complete! At this point, you can remove your boot media and then select **Continue** at the final confirmation screen to reboot the system:



Final confirmation prompt during installation

At this point, your server will reboot and then Ubuntu Server should start right up. Although the installation process contained quite a few steps, it's really quick once you get used to it. To make things easier, later on in this book I'll go over the process of creating images you can use to deploy Ubuntu Server, so you'll be able to create your own pre-configured environments to simplify the process greatly.

Installing Ubuntu Server (Raspberry Pi)

The Raspberry Pi has become quite a valuable asset in the industry. These tiny computers, now with four cores and 1 GB of RAM, are extremely useful in running one-off tasks and even offering a great place to test code without disrupting existing infrastructure. In my lab, I have three Pis on my network, each one responsible for performing specific tasks or functions. It's a very useful platform.

Thankfully, Ubuntu Server is quite easy to install on a Raspberry Pi. In fact, it's installed in much the same way as Raspbian, which is the distribution most commonly used on the Pi. Ubuntu Server is available for Raspberry Pi models 2 and 3. All you'll need to do is download the SD card image and then copy it directly to your SD card. Once you've done that, you can boot the Pi and log in. The default username and password is `ubuntu` for both.

At the time of writing, the version of Ubuntu Server for the Raspberry Pi 3 is not officially supported but should work. The Pi 2 version is fully supported. The Pi 3 version is expected to be fully supported at some point.

www.it-ebooks.info

To install Ubuntu on the Pi, perform the following steps:

1. To get started, you'll first download the SD card image from here: `https://wiki.ubuntu.com/ARM/RaspberryPi` and check that it is compatible with your model of Raspberry Pi.
2. In order to write the image to an SD card, first extract it and then use `dd` to do the actual copying. This is very similar to the process we used to create bootable USB media. In fact, it is the same; the only difference is the source and target. From a Linux system, you can perform the process with the following command:

   ```
   # dd if=/path/to/downloaded/file of=/dev/sdX bs=1M
   ```

3. As before, replace `if=` and `of=` with the path to the downloaded and extracted file and the target respectively. If you are unsure about which device your SD card is on your system, execute:

   ```
   # fdisk -l
   ```

4. In my case, this process can take some time to complete. Often, the command prompt will return even before the syncing process is actually finished. This means you'll more than likely see the access indicator LED on your card reader continue to flash for a while before the process is truly complete. It goes without saying, but make sure you remove your SD card only once the LED has stopped flashing. If you'd rather have the command prompt NOT return until after the process is complete, add the sync option, like the following:

   ```
   # dd if=/path/to/downloaded/file of=/dev/sdX bs=1M; sync
   ```

5. So for example, on my system, the command will be:

   ```
   # dd if=/home/jay/downloads/ubuntu-server-16.04.img of=/dev/sdc
   bs=1M; sync
   ```

Depending on the speed of your card reader, this process will likely take at least 5 minutes to complete, if not longer. Once you're finished and you start up your Pi, you're likely to notice an issue straight away. If we execute:

```
df -h
```

We'll see that regardless of the size of our SD card, we'll have quite a bit of space that is not being utilized. This is because the SD card image will be the size of the SD card it was captured from, which will most likely not equal the size of yours. This will result in wasted space. What we'll need to do is resize the installation to match our SD card.

www.it-ebooks.info

First, from the Pi after it's started up, enter `fdisk` by executing the following command:

```
# fdisk /dev/mmcblk0
```

Next, we'll need to delete the second partition, by pressing *d* + *2*.

With the partition deleted (and while still within the `fdisk` console), we'll recreate the partition we deleted by pressing *n* + *p* + *2* and *Enter* (twice) +*w*.

With those commands executed, execute the `reboot` command to restart the Pi. However, we have one more command to execute. Once it starts backup, we can do the actual resizing with:

```
# resize2fs /dev/mmcblk0p2
```

After the resize procedure, you should be all set and able to utilize all the space of your SD card. You should also be able to use your Pi as a means of following along with the activities of this book. One last thing, though. If you execute this, you'll notice that your Pi doesn't have a swap partition:

```
free -m
```

This is normal, and while you can create a `swap` file to fill the role of a `swap` partition, I recommend you don't. SD cards are much too slow for `swap`, so you'd be better off without it. Since the Raspberry Pi 2 and Pi 3 only have about 1 GB of RAM, make sure whatever processes you plan on running fit within that restraint and you should be fine.

Summary

In this chapter, we covered the installation process in great detail. As with most Linux distributions, Ubuntu Server is scalable and able to be installed on a variety of server types. Physical servers, virtual machines, and even the Raspberry Pi have a version of Ubuntu Server available. The process of installation was covered step by step, and you should now have an Ubuntu Server of your own to configure as you wish. Also in this chapter, we covered partitioning, creating boot media, determining the role of your server, as well as some best practices.

In the next chapter, I'll show you how to manage users. You'll be able to create them, delete them, change them, and even manage password expiration and more. I'll also cover permissions so that you can determine what your users are allowed to do on your server. See you there!

www.it-ebooks.info

www.it-ebooks.info

2
Managing Users

As an administrator of an Ubuntu Server, users can be your greatest asset and also your biggest headache. During your career you'll add countless new users, manage their passwords, remove their accounts when they leave the company, and grant or remove access to resources across the file-system. Even on servers on which you're the only user, you'll still find yourself managing user accounts since even system processes run as users. To be successful at managing Linux servers, you'll also need to know how to manage permissions, create password policies, and limit who can execute administrative commands on the machine. In this chapter, we'll work through these concepts so that you'll have a clear idea of how to manage users and their resources. In particular, we will cover:

- Understanding when to use `root`
- Creating and removing users
- Understanding the `/etc/passwd` and `/etc/shadow` files
- Distributing default configuration files with `/etc/skel`
- Switching between users
- Managing groups
- Managing passwords and password policies
- Configuring administrator access with `sudo`
- Setting permissions on files and directories

www.it-ebooks.info

Understanding when to use root

In the last chapter, we set up our very own Ubuntu Server installation. During the installation process, we were instructed to create a user account as an administrator of the system. So, at this point, we should have two users on our server. We have the aforementioned administrative user, as well as `root`. We can certainly create additional user accounts with varying levels of access (and we will do so in this chapter), but before we get to that, some discussion is in order regarding the administrator account you created, as well as the `root` user that was created for you.

In regards to `root`, the `root` user account exists on all Linux distributions and is the most powerful user account on the planet. The `root` user account can be used to do anything, and I do mean anything. Want to use `root` to create files and directories virtually anywhere on the file-system? No problem. Want to use `root` to install software? Again, not a problem. The `root` account can even be used to destroy your entire installation with one typo or ill-conceived command. Even if you instruct `root` to delete all files on your entire hard disk, it won't hesitate to do so. It's always assumed on a Linux system that if you are using `root`, you are doing so because you know what you are doing. So there's often not so much as a confirmation prompt while executing any command as `root`. It will simply do as instructed, for better or worse.

It's for this reason that every Linux distribution I've ever used instructs, or at least highly recommends, you to create a standard user during the installation process. It's generally recommended that you not use the `root` account unless you absolutely have to. It's common in the Linux community for an administrator to have his or her own account and then switch to `root` whenever a task comes up that requires `root` privileges to complete. It's less likely to accidentally destroy your server with an accidental typo or bad command while you're not logged in as `root`. Don't get me wrong, some arrogant administrators will strictly use `root` at all times without any issue, but again, it's recommended to use `root` only when you have to. And besides, when it comes to arrogant administrators, don't be that person.

Most distributions ask you to create a `root` password during installation in order to protect that account. Ubuntu Server is a bit different in this regard, however. You weren't even asked what you wanted the `root` password to be. The reason for this is because Ubuntu defaults to locking out the `root` account altogether. This is very much unlike many other distributions. In fact, Debian (on which Ubuntu is based), even has you set a `root` password during installation. Ubuntu just decides to do things a little bit differently. There's nothing stopping you from enabling `root`, or switching to the `root` user after you log in. Being disabled by default just means the `root` account isn't as easily accessible as it normally would be. I'll cover how to enable this account later in this chapter, should you feel the need to do so.

www.it-ebooks.info

An exception to this rule is that most VPS providers, such as **Linode** or **DigitalOcean**, will enable the `root` account even on their Ubuntu servers. Typically the `root` password will be randomly generated and emailed to you.

Instead, Ubuntu (as well as its server version), recommends the use of `sudo` rather than using `root` outright. I'll go over how to manage `sudo` later on in this chapter, but for now, just keep in mind that the purpose of `sudo` is to enable you to use your user account to do things that normally only `root` would've been able to do. For example, you cannot issue a command such as the following to install a software package as a normal user:

```
apt-get install tmux
```

Instead, you'll receive an error:

```
E: Could not open lock file /var/lib/dpkg/lock - open (13: Permission
denied)
E: Unable to lock the administration directory (/var/lib/dpkg/), are you
root?
```

But if you prefix the command with `sudo` (assuming your user account has access to it), the command will work just fine:

```
# sudo apt-get install tmux
```

When you use `sudo`, you'll be asked for your user's password for confirmation, and then the command will execute. Understanding this should clarify the usefulness of the user account you created during installation. I referred to this user as an administrative account earlier, but all it really is is a user account that is able to utilize `sudo`. (Ubuntu Server automatically gives the first user account you create during installation access to `sudo`). The intent is that you'll use that account for administering the system, rather than `root`. When you create additional user accounts, they will not have access to `sudo` by default, unless you explicitly grant it to them.

Creating and removing users

Creating users in Ubuntu can be done with one of either of two commands: `adduser` and `useradd`. This can be a little confusing at first, because both of these commands do the same thing (in different ways) and are named very similarly. I'll go over the `useradd` command first and then I'll explain how `adduser` differs. You may even prefer the latter, but we'll get to that in a moment.

www.it-ebooks.info

First, here's an example of the `useradd` command in action:

```
# useradd -d /home/jdoe -m jdoe
```

As I mentioned in the front matter for this book, whenever I prefix commands with a pound symbol (#), I'm instructing you to execute the command as `root`. You can use the actual `root` account for these types of commands or you can simply prefix the command with `sudo`. In the latter case, the command would become:

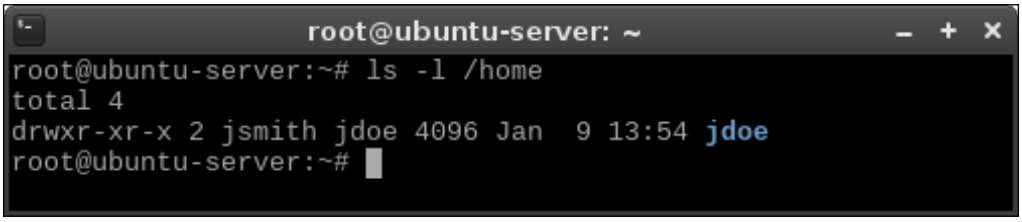
```
# sudo useradd -d /home/jdoe -m jdoe
```

I won't remind you of this henceforth, so just keep in mind commands prefixed with # need to be executed as `root` or with the prefix `sudo`.

In this example, I'm creating a user named `jdoe`. With the `-d` flag, I'm clarifying that I would like a `home` directory created for this user, and following the flag I called out `/home/jdoe` as the user's `home` directory. The `-m` flag tells the system that I would like the `home` directory created during the process; otherwise, I would've had to create the directory myself. Finally, I called out the username for my new user.

If you list the storage of `/home`, you should see a folder listed there for our new user:

```
ls -l /home
```



Listing the contents of /home after our first user was created

What about creating our user's password? We created a new user on our system, but we did not set a password. We weren't even asked for one—what gives? To create a password for the user, we can use the `passwd` command. The `passwd` command defaults to allowing you to change the password for the user you're currently logged in as, but it also allows you to set a password for any other user if you run it as `root` or with `sudo`. If you enter `passwd` by itself, the command will first ask you for your current password, then your new password, and then it will ask you to confirm your new password again. If you prefix the command with `sudo` and then specify a different user account, you can set the password for any user you wish. An example of the output of this process is as follows:

www.it-ebooks.info

```
root@ubuntu-server:~# passwd jdoe
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

You won't see any asterisks or any kind of output as you type a password while using the `passwd` command. This is normal. Although you won't see any visual indication of input, your input is being recognized.

Now we have a new user and we were able to set a password for that user. The `jdoe` user will now be able to access the system with the password we've chosen.

Earlier, I mentioned the `adduser` command as another way of creating a user. The difference (and convenience) of this command should become apparent immediately once you use it. Go ahead and give it a try; execute `adduser` along with a username for a user you wish to create. An example of a run of this process is as follows:

```
root@ubuntu-server:~# adduser dscully
Adding user `dscully' ...
Adding new group `dscully' (1006) ...
Adding new user `dscully' (1006) with group `dscully' ...
Creating home directory `/home/dscully' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for dscully
Enter the new value, or press ENTER for the default
    Full Name []: Dana Scully
    Room Number []: 405
    Work Phone []: 555-412-5555
    Home Phone []: 412-555-5555
    Other []: Trust no one
Is the information correct? [Y/n] y
```


www.it-ebooks.info

In the process above, I executed `adduser dscully` and then I was asked a series of questions regarding how I wanted the user to be created. I was asked for the password (twice), `Full Name`, `Room Number`, `Work Phone` and `Home Phone`. In the `Other` field, I entered the comment `Trust no one`, which is a great mindset to adopt while managing users. In the example output, I bolded sections where I was asked to type input. The latter prompts prior to the final confirmation were all optional. I didn't have to enter a `Full Name`, `Room Number`, and so on. I could've pressed *Enter* to skip those prompts if I wanted to. The only thing that's really required is the username and the password.

From the output, we can see that the `adduser` command performed quite a bit of work for us. The command defaulted to using `/home/dscully` as the `home` directory, the account was given the next available **User ID** (**UID**) and **Group ID** (**GID**) of `1006`, and it also copied files from `/etc/skel` into our new user's `home` directory. In fact, both the `adduser` and `useradd` commands copy files from `/etc/skel`, but `adduser` is more verbose regarding the actions it performs.

Don't worry if you don't understand `UID`, `GID`, and `/etc/skel` yet. We'll work through those concepts soon.

In a nutshell, the `adduser` command is much more convenient in the sense that it prompts you for various options while it creates the user without requiring you to memorize command line options. It also gives you detailed information about what it has done. At this point, you may be wondering why someone would want to use `useradd` at all, considering how much more convenient `adduser` seems to be. Unfortunately, `adduser` is not available on all distributions of Linux. It's best to familiarize yourself with `useradd` in case you find yourself on a Linux system that's not using Ubuntu.

It may be interesting for you to see what exactly the `adduser` command actually is. It's not even a binary program—it's a shell script. A shell script is simply a text file that can be executed as a program. In the case of `adduser`, it's a script written in Perl. Since it's not binary, you can even open it in a text editor in order to view all the magic code that it executes behind the scenes. However, make sure you don't open the file in a text editor with `root` privileges, so you don't accidentally save changes to the file and break it. The following command will open `adduser` in a text editor on an Ubuntu Server system:

```
nano /usr/sbin/adduser
```


www.it-ebooks.info

Use your up/down arrows as well as *Page Up* and *Page Down* keys to scroll through the file. When you're finished, press *Ctrl* + *X* on your keyboard to exit the text editor.

Those of you with keen eyes will likely notice that the `adduser` script is calling `useradd` to perform its actual work. So either way, you're using `useradd` either directly or indirectly.

Now that we know how to create users, it will be useful to understand how to remove them as well. After all, removing access is very important when a user no longer needs to access a system, as unmanaged accounts often become a security risk. To remove a user account, we'll use the `userdel` command.

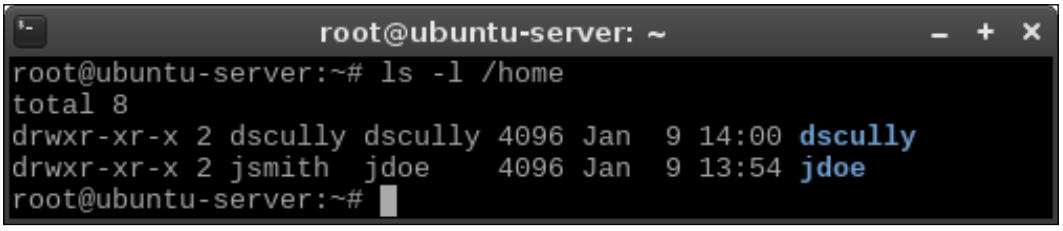
Before removing an account, though, there is one very important question you should ask yourself. Will you still need access to the user's files? Most companies have retention policies in place that detail what should happen to a user's data when he or she leaves the organization. Sometimes, these files are copied into an archive for long-term storage. Often, a manager, co-worker, or new hire will need access to the former user's files to continue working on a project where they left off. It's important to understand this policy ahead of managing users. If you don't have a policy in place that outlines retention requirements for files when users resign, you should probably work with your management and create one.

By default, the `userdel` command does not remove the contents of the user's `home` directory. If we use the following command to remove `dscully` from the system:

```
# userdel dscully
```

We'll see that `dscully`'s `home` directory still exists and is intact:

```
ls /home
```



The home directory for user dscully still exists, even though we removed the user

With the `home` directory for `dscully` still existing, we're able to move the contents of this directory anywhere we would like to. If we had a directory called `/store/file_archive`, for example, we can easily move the files there:

```
# mv /home/dscully /store/file_archive
```


www.it-ebooks.info

Of course, it's up to you to create the directory where your long-term storage will ultimately reside, but you get the idea.

If you weren't already aware, you can create a new directory with the `mkdir` command. You can create a directory within any other directory your logged-in user has access to. The following command will create the `file_archive` directory I mentioned in my example:

```
# mkdir -p /store/file_archive
```

The `-p` flag simply creates the parent directory if it didn't already exist.

If you do actually want to remove a user's `home` directory at the same time you removed an account, just add the `-r` parameter. This will eliminate the user and their data in one shot.

```
# userdel -r dscully
```

To remove the `home` directory for the user after the account was already removed (if you didn't use the `-r` parameter the first time), use the `rm -r` command to get rid of it, as you would any other directory:

```
# rm -r /home/dscully
```

It probably goes without saying, but the `rm` command can be extremely dangerous. If you're logged in as `root` or using `sudo` while using `rm`, you can easily destroy your entire installed system if you're not careful. For example, the following command (while seemingly innocent at first glance) will likely completely destroy your entire file system:

```
# rm -r / home/dscully
```

Notice the typo: I accidentally typed a space after the first forward slash. I literally accidentally told my system to remove the contents of the entire file system. If that command was executed, the server probably wouldn't even `boot` the next time we attempt to start it. All user and program data would be wiped out. If there was ever any single reason for us to be protective over the `root` account, the `rm` command is certainly it!

www.it-ebooks.info

Understanding the /etc/passwd and /etc/shadow files

Now that we know how to create (and delete) user accounts on our server, we are well on our way to being able to manage our users. But where exactly is this information stored? We know that users store their personal files in `/home`, but is there some kind of database somewhere that keeps track of which user accounts are on our system? Actually, user account information is stored in two special text files: `/etc/passwd` and `/etc/shadow`.

You can display the contents of each of those two files with the following commands. Take note that any user can look at the contents of `/etc/passwd`, while only `root` has access to `/etc/shadow`:

```
# cat /etc/passwd
# cat /etc/shadow
```

Go ahead and take a look at these two files (just don't make any changes), and I will help you understand them. First, let's go over the `/etc/passwd` file. What follows is example output from this file on my test server. For brevity, I limited the output to the last seven lines:

```
testuser:x:1000:1000::/home/testuser:0
testuser2:x:1006:1006:Test user,,,:/home/testuser2:/bin/bash
myuser:x:1002:1002::/home/myuser:
myuser2:x:1003:1003::/home/myuser2:
jdoe:x:1004:1004::/home/jdoe:
bsmith:x:1005:1005::/home/bsmith:/bin/bash
jdoe2:x:1007:1007::/home/jdoe2:
```

Each line within this file contains information for each user account on the system. Entries are split into columns, separated by a colon ":". The username is in the first column, so you can see that I've created users `testuser`, `testuser2`, `myuser`, and so on. The next column on each is simply an "`x`". I'll go over what that means a bit later. For now, let's skip to the third and fourth columns, which reference the `UID` and `GID` respectively. On a Linux system, user accounts and groups are actually referenced by their IDs. While it's easier for you and I to manage users by their names, usernames and group names are nothing more than a label placed on the `UID` and `GID` in order to help us identify with them easier. For example, it may be frustrating to try to remember that `jdoe` is `UID 1004` on our server each time we want to manage this account. Managing it by referring to the account as `jdoe` is easier for us humans, since we don't remember numbers as well as we do names. But to Linux, each time we reference user `jdoe`, we're actually just referencing `UID 1004`. When a user is created, the system (by default) automatically assigns the next available `UID` to the account.

www.it-ebooks.info

In my case, the `UID` of each user is the same as their `GID`. This is just a coincidence on my system and isn't always that way in practice. While I'll discuss creating groups later in this chapter, understand that creating groups works in a similar way as creating users, in the sense that the group is assigned the next available `GID` in much the same way as new user accounts are assigned the next available `UID`. When you create a user, the user's primary group is the same as their username (unless you request otherwise). For example, when I created `jdoe`, the system also automatically created a `jdoe` group as well. This is what you're actually seeing here – the `UID for` the user, as well as the `GID` for the user's primary group. Again, we'll get to groups in more detail later.

You probably also noticed that the `/etc/passwd` file on your system contains entries for many more users than the ones we've created ourselves. This is perfectly normal, as Linux uses user accounts for various processes and things that run in the background. You'll likely never interact with the default accounts at all, though you may someday create your own system user for a process to run as. For example, perhaps you'd create a data processor account for an automated data processing script to run under.

Anyway, back to our `/etc/passwd` file. The fifth column is designated for user info, most commonly the first and last name. In my example, the fifth field is blank for most except for `testuser2`. I created `testuser2` with the `adduser` command, which asked me for the first and last name, which I entered as `Test user`.
Here is that line again:

```
testuser2:x:1006:1006:Test user,,,:/home/testuser2:/bin/bash
```

In the sixth column, the `home` directory for our user is shown. In the case of `testuser2`, it's set as `/home/testuser2`. Finally, we designate the user's shell as `/bin/bash`. This refers to the default shell the user will use, which always defaults to `/bin/bash`. If we wanted the user to use a different shell, we can clarify that here (though shells other than `/bin/bash` are beyond the scope of this book). If we wanted, we could change the user's shell to something invalid to prevent them from logging in at all. This is useful for when a security issue requires us to disable an account quickly, or if we're in the mood to pull a prank on someone.

With that out of the way, let's take a look at the `/etc/shadow` file. We can use `cat` to display the contents as any other text file, but unlike `/etc/passwd`, we need `root` privileges in order to view it. So, go ahead and display the contents of this file, and I'll walk you through it:

```
# cat /etc/shadow
```


www.it-ebooks.info

For comparison, the last few lines of my `/etc/shadow` file are as follows:

```
myuser2:$6$maFOiNL.:16809:0:99999:7:::
jdoe:$6$TPxx8Z.:16809:0:99999:7:::
bsmith:$6$KoShUY.:16809:0:99999:7:::
testuser3:$6$QAGTNqR:16809:0:99999:7::
```

I've shortened the second column by quite a few characters in order to save space on this page. The second column in your output will appear much longer.

First, we have the username in the first column—no surprises there. Note that the output is not showing the `UID` for each user in this file. The system knows which username matches to which `UID` based on the `/etc/passwd` file, so there's no need to repeat that here. In the second column, we have what appears to be random *gobbledygook*. Actually, that's the most important part of this entire file. That's the actual hash for the user's password. If you recall, in the `/etc/passwd` file, each user listing had an "`x`" for the second column, and I mentioned I would explain it later. What the "`x`" refers to is the fact that the user's password is encrypted and simply not stored in `/etc/passwd` but is instead stored in `/etc/shadow`. After all, the `/etc/passwd` file is viewable by everyone, so it would compromise security quite a bit if anyone could just open up the file and see what everyone's password hashes are. In the days of old, you could actually store a user's password in `/etc/passwd`, but it's literally never done that way anymore. Whenever you create a user account on a modern Linux system, the user's password is encrypted (an "`x`" is placed in the second column of `/etc/passwd` for the user), and the actual password hash is stored in the second column of `/etc/shadow` to keep it away from prying eyes. Hopefully now the relationship between these two files has become apparent.

Remember earlier I mentioned that the root user account is locked out by default? Well, let's actually see that in action. Execute the following command to see `root`'s entry in `/etc/shadow`:

```
# cat /etc/shadow | grep root
```

On my system, I get the following output:

```
root:!:16402:0:99999:7:::
```


www.it-ebooks.info

You should notice right away that the `root` user account doesn't have a password hash at all. Instead, there's an exclamation point where the password hash would've been. In practice, placing an exclamation point here is one way to lock out an account, though the standard practice is to use an "`x`" in this field instead. But either way, we can still switch to the `root` account (which I'll show you how to do later on in this chapter). The restriction is that we can't directly log in as `root` from the shell or over the network. We have to log in to the system as a normal user account first.

With the discussion of password hashes out of the way, there are a few more fields within `/etc/shadow` entries that we should probably understand. Here's an example line from the file on my system to save you the trouble of flipping back:

```
jdoe:$6$TPxx8Z.:16809:0:99999:7:::
```

Continuing on with the third column, we can see the number of days since the UNIX Epoch that the password has last been changed. For those that don't know, the UNIX Epoch is January 1st, 1970. Therefore, we can read that column as the password having last been changed 16,809 days since the UNIX Epoch.

Personally, I like to use the following command to show more easily when the password was last changed:

`# passwd -S <username>`

By executing this command (it must be run as `root`), you can view information about any account on your system. The third column of this command's output gives you the actual date of the last password change for the user.

The fourth column tells us how many days are required to pass before the user will be able to change their password again. In the case of my previous sample, `testuser` can change the password anytime because the minimum number of days is set to `0`. We'll talk about how to set the minimum number of days later on in this chapter, but I'll give you a brief explanation of what this refers to. At first, it may seem silly to require a user to wait a certain number of days to be able to change his or her password. However, never underestimate the laziness of your users. It's quite common for a user, when required to change his or her password, to change their password several times to satisfy history requirements, to then just change it back to what it was originally. By setting a minimum number of days, you're forcing a waiting period in between password changes, making it less convenient for your users to cycle back through to their original password.

www.it-ebooks.info

The fifth column, as you can probably guess, is the maximum number of days that can pass between password changes. If you require your users to change their passwords every certain number of days, you'll see that in this column. By default, this is set to `99,999` days. It probably stands to reason that you'll move on to bigger and better things by the time that elapses, so it may as well be infinite.

Continuing with the sixth column, we have the number of days that will elapse before the user is warned that they will soon be required to change their password. In the seventh column, we set how many days can pass after the password expires, in which case the account will be disabled. In our example, this is not set. Finally, with the eighth column, we can see the number of days since the UNIX Epoch that will elapse before the account is disabled (in our case, there's nothing here, so there is no disabled day set). We'll go over setting these fields later, but for now hopefully you understand the contents of the `/etc/shadow` file better.

Distributing default configuration files with /etc/skel

In a typical organization, there are usually some defaults that are recommended for users in terms of files and configuration. For example, in a company that performs software development, there are likely recommended settings for text editors and version control systems. Files that are contained within `/etc/skel` are copied into the `home` directory for all new users when you create them (assuming you've chosen to create a `home` directory while setting up the user).

In fact, you can see this for yourself right now. If you execute the following command, you can view the contents of the `/etc/skel` directory:

```
ls -la /etc/skel
```

You probably already know how to list files within a directory, but I specifically called out the `-a` parameter because the files included in `/etc/skel` by default are hidden (their file names begin with a period). I threw the `-l` parameter solely because I like it better (it shows a long list, which I think is easier to read). Your output will likely include the following files:

```
.bash_logout
.bashrc
.profile
```


www.it-ebooks.info

Each time you create a new user and request a `home` directory to be created as well, these three files will be copied into their `home` directory, along with any other files you create here. You can verify this by listing the storage of the `home` directories for the users you've created so far. The `.bashrc` file in one user's `home` directory should be the same as any other, unless they've made changes to it.

Armed with this knowledge, it should be extremely easy to create default files for new users you create. For example, you could create a file named `welcome` with your favorite text editor and place it in `/etc/skel`. Perhaps you may create this file to contain helpful phone numbers and information for new hires in your company. The file would then be automatically copied to the new user's `home` directory when you create the account. The user, after logging in, would see this file in his or her `home` directory and see the information. More practically, if your company has specific editor settings you favor for writing code, you can include those files in `/etc/skel` as well to help ensure your users are compliant. In fact, you can include default configuration files for any application your company uses.

Go ahead and give it a try. Feel free to create some random text files and then create a new user afterwards, and you'll see that these files will propagate into the home directories of new user accounts you add to your system.

Switching between users

Now that we have several users on our system, we need to know how to switch between them. Of course, you can always just log in to the server as one of the users, but you can actually switch to any user account at any time providing you either know that user's password or have `root` access.

The command you will use to switch from one user to another is the `su` command. If you enter `su` with no options, it will assume that you want to switch to `root` and will ask you for your `root` password. As I mentioned earlier, Ubuntu locks out the `root` account by default, so you really don't have a `root` password. Unlocking the `root` account is actually really simple; all you have to do is create a `root` password. To do that, you can execute the following command as any user with `sudo` access:

```
sudo passwd
```


www.it-ebooks.info

The command will ask you to create and confirm your `root` password. From this point on, you will be able to use the `root` account as any other account. You can log in as `root`, switch to `root`; it's fully available now. You really don't have to unlock the `root` account in order to use it. You certainly can, but there are other ways to switch to `root` without unlocking it, and it's typically better to leave the `root` account locked unless you have a very specific reason to unlock it. Either of the following commands will allow you to switch to `root` from a user account that has `sudo` access:

```
sudo su
sudo -s
```

Now you'll be logged in as `root` and able to execute any command you want with no restrictions whatsoever. To return to your previous logged-in account, simply type `exit`. You can tell which user you're logged in as by the value at the beginning of your bash prompt.

But what if you want to switch to an account other than `root`? Of course, you can simply log out and then log in as that user. But you really don't have to do that. The following command will do the job, providing you know the password for the account.

```
su - <username>
```

The shell will ask for that user's password and then you'll be logged in as that user. Again, type `exit` when you're done using the account. That command is all well and good if you know the user's password, but you often won't. Typically in an enterprise, you'll create an account, force the user to change their password at first log in, and then you will no longer know that user's password. Since you have `root` and `sudo` access, you could always change their password and then log in as them. But they'll know something is amiss if their password suddenly stops working—you're not eavesdropping, are you?

Armed with `sudo` access, you can use `sudo` to change to any user you want to, even if you don't know their password. Just prefix our previous command with `sudo` and you'll only need to enter the password for your user account, instead of theirs:

```
sudo su - <username>
```


www.it-ebooks.info

Switching to another user account is often very helpful for support (especially while troubleshooting permissions). As an example, say that a user comes to you complaining that he or she cannot access the contents of a specific directory, or they are unable to run a command. In that case, you can log in to the server, switch to their user account, and try to reproduce their problem. That way, you can not only see their problem yourself, but you can also test out whether or not your fix has solved their issue before you report back to them.

Managing groups

Now that we understand how to create, manage, and switch between user accounts, we'll need to understand how to manage groups as well. The concept of groups in Linux is not very different from other platforms and pretty much serves the exact same purpose. With groups, you can more efficiently control a user's access to resources on your server. By assigning a group to a resource (a file, directory, and so on), you can allow and disallow access to users by simply adding them or removing them from the group.

The way this works in Linux is that every file or directory has both a user and a group that takes ownership of it. This is contrary to platforms such as Windows, which can have multiple groups assigned to a single resource. With Linux, it's just a one-to-one ownership: just one user and just one group assigned to each file or directory. If you list contents of a directory on a Linux system, you can see this for yourself:

```
ls -l
```

The following is sample output from a directory on one of my servers:

```
-rw-r--r-- 1 root bind  490 2013-04-15 22:05 named.conf
```

In this case, we can see that `root` owns the file and that the group bind is also assigned to it. Ignore the other fields for now; I'll explain them later when we get to the section of this chapter dedicated to permissions. For now, just keep in mind that one user and one group are assigned to each file or directory.

While each file or directory can only have one group assignment, any user account can be a member of any number of groups. The groups command will tell you what groups your currently logged-in user is currently a member of. If you add a username to the `groups` command, you'll see which groups that user is a member of. Go ahead and give the groups command a try with and without providing a username to get the idea.

www.it-ebooks.info

On the Ubuntu Server platform, you'll likely see that each of your user accounts is a member of just one group, a group that's named the same as your username. As I mentioned earlier, when you create a user account, you're also creating a group with the same name as the user. On some Linux distributions, though, a user's primary group will default to a group called `users` instead. If you were to execute the `groups` command as a user on the Ubuntu desktop platform, you would likely see additional groups. This is due to the fact that distributions of Linux that cater to being a server platform are often more stripped down and users on desktop platforms need access to more things such as printers, audio cards, and so no.

If you were curious as to which groups exist on your server, all you would need to do is `cat` the contents of the `/etc/group` file. Similar to the `/etc/passwd` file we covered earlier, the `/etc/group` file contains information regarding the groups that have been created on your system. Go ahead and take a look at this file on your system:

```
cat /etc/group
```

The following is sample output from this file on one of my servers:

```
kvm:x:36:qemu,jay
admins:x:901:ansible,jay
testuser:x:1000:
testuser2:x:1001:
myuser:x:1002:
```

I shortened this file a bit when I pasted it, as there are quite a few groups created by default on Linux systems. Toward the end of the file, we see some entries for users that we created earlier. These are the primary user groups I mentioned earlier. Like before, the columns in this file are separated by colons, though each line is only four columns long. In the first column, we have the name of the group. No surprise there. In the second, we are able to store a password for the group, but this is not used often. In the third column, we have the `GID`, which is similar in concept to the `UID` when we were discussing users. Finally, in the last column, we see a comma-separated list of each user that is a member of each of the groups. In this case, we're seeing that users `ansible` and `jay` are a member of admins and `qemu` and `jay` are members of the `kvm` group. The last several entries (the ones for our users) don't show any group memberships at all. Each user is indeed a member of their own group, so this is implied even though it doesn't explicitly call that out in this file. If you take a look at `/etc/passwd` entries for your users, you will see that their primary group (shown as the third column in the form of a `GID`) references a group contained in the `/etc/group` file.

www.it-ebooks.info

Creating new groups on your system is easy to do and is a great idea for categorizing your users and what they are able to do. Perhaps you can create an accounting group for your accountants, an admins group for those in your IT department, and a sales group for your sales people. The `groupadd` command allows you to create new groups. If you wanted to, you could just edit the `/etc/group` file and add a new line with your group information manually (and some even prefer this method), although, in my opinion, using `groupadd` saves you some work and ensures that group entries are created properly. Editing group and user files directly is typically frowned upon (and a typo can cause serious problems). Anyway, what follows is an example of creating a new group with the `groupadd` command:

```
# groupadd admins
```

If you take a look at the `/etc/group` file again after adding a new group, you'll see that a new line was created in the file and a `GID` was chosen for you (the first one that hadn't been used yet). Removing a group is just as easy. Just issue the `groupdel` command followed by the name of the group you wish to remove:

```
# groupdel admins
```

Next, we'll take a look at the `usermod` command, which will allow you to actually associate users with groups. The `usermod` command is more or less a Swiss Army knife; there are several things you can do with that command (adding a user to a group is just one thing it can do). If we wanted to add a user to our `admins` group, we would issue the following command:

```
# usermod -aG admins myuser
```

In that example, we're supplying the `-a` option, which means append, and immediately following that we're using `-G`, which means we would like to modify groups. I put the two options together with a single dash (`-aG`), but you could also issue them separately (`-a -G`) as well. The example I gave only adds the user to additional groups, it doesn't replace their primary group. If you wanted to change a user's primary group, you would use the `-g` option instead:

```
# usermod -g <group-name> <username>
```

Feel free to check out the manual pages for the `usermod` command, to see all the nifty things it allows you to do with your users. One additional example is changing a user's `home` directory. Suppose that one of your users has undergone a name change, so you'd like to change their username, as well as move their previous `home` directory (and their files) to a new one. The following commands will take care of that:

```
# usermod -d /home/jsmith jdoe -m
# usermod -l jsmith jdoe
```


www.it-ebooks.info

In that example, we're moving the `home` directory for `jdoe` to `/home/jdoe`, and then in the second example, we're changing the username from `jdoe` to `jsmith`.

If you wish to remove a user from a group, you can use the `gpasswd` command to do so. `gpasswd -d` will do the trick:

```
# gpasswd -d <username> <grouptoremove>
```

In fact, `gpasswd` can also be used in place of `usermod` to add a user to a group:

```
# gpasswd -a <username> <group>
```

So, now you know how to manage groups. With efficient management of groups, you'll be able to best manage the resources on your server. Of course, groups are relatively useless without some explanation of how to manage permissions (otherwise, nothing would actually be enforcing a member of a group to access a resource). Later on in this chapter, we'll cover permissions so you'll have a complete understanding of how to manage user access.

Managing passwords and password policies

In this chapter, we've already covered a bit of password management, since I've given you a few examples of the `passwd` command. If you recall, the `passwd` command allows us to change the password of the currently logged-in user. In addition, using `passwd` as `root` (and supplying a user name) allows us to change the password for any user account on our system. But that's not all this command can do.

One thing I've neglected to mention in regards to the `passwd` command is the fact that you can use it to lock and unlock a user's account. There are many reasons why you may want to do this. For instance, if a user is going on vacation or extended leave, perhaps you'd want to lock their account so that it cannot be used while they are away. After all, the fewer active accounts, the lower your attack surface. To lock an account, use the `-l` option:

```
# passwd -l <username>
```

And to unlock it, use the `-u` option:

```
# passwd -u <username>
```


www.it-ebooks.info

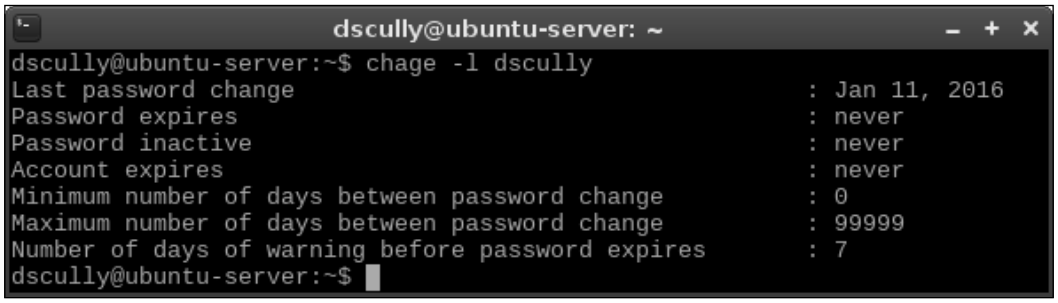
However, locking an account will not prevent the user from logging in if he or she has access to the server via SSH with utilizing public key authentication. In that case, you'd want to remove their ability to use SSH as well. One common way of doing this is to limit SSH access to users who are a member of a specific group. When you lock an account, simply remove them from the group. Don't worry so much about the SSH portion of this discussion if this is new to you. We will discuss securing your SSH server in *Chapter 12*, *Securing Your Server*. For now, just keep in mind that you can use the `passwd` to lock or unlock accounts, and if you utilize SSH, you'll want to lock your users out of that if you don't want them logging in.

However, there's more to password management than the `passwd` command, as we can also implement our own policies as well. Earlier, I mentioned that you can set an expiration date on a user's password (during our discussion on the `/etc/shadow` file). In this section, we'll go through how to actually do that. Specifically, the `chage` command (pronounced *Sage*) gives us this ability. We can use `chage` to not only alter the expiration period of a user's password, but it's also a more convenient way of viewing current expiration information than viewing the `/etc/shadow` file. With the `-l` option of `chage`, along with providing a username, we can see the relevant info:

```
chage -l <username>
```

Using `sudo` or `root` is not required to run `chage`. You're able to view expiration information for your own username without needing to escalate permissions. However, if you want to view information via `chage` for any user account other than your own, you will need to use `sudo`.

In the example that follows, we can see the output of this command from a sample user account:



Sample output from the chage command

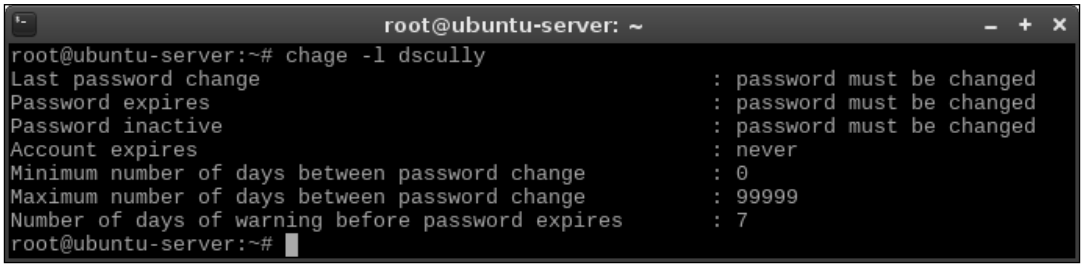
www.it-ebooks.info

In the output, we can see values for the date of expiration, maximum number of days between password changes, and so on. Basically, it's the same information stored in `/etc/shadow` but it's much easier to read.

If you would like to change this information, `chage` will again be the tool of choice. The first example I'll provide is a very common one. When creating user accounts, you'll certainly want them to change their password when they first log in. Unfortunately, not everyone will be keen to do so. The `chage` command allows you to force a password change for a user when he or she first logs in. Basically, you can set their number of days to expiry to `0` with the following example:

```
# chage -d 0 <username>
```

You can see the results of this command immediately if you run `chage -l` again against the user account you just modified:



The chage command listing a user that has a required password change set

To set a user account to require a password change after a certain period of days, the following example will do the trick:

```
# chage -M 90 dscully
```

In that example, I'm setting the user account to expire and require a password change in `90` days. When the impending date reaches 7 days before the password is to be changed, the user will see a warning message when they log in.

As I mentioned earlier, users will often do whatever they can to cheat password requirements and may try to change their password back to what it was originally after satisfying the initial required password change. You can set the minimum number of days with the `-m` flag, as you can see in the next example:

```
# chage -m 5 dscully
```


www.it-ebooks.info

The trick with setting a minimum password age is to set it so that it will be inconvenient for the user to change their password to the original one, but you still want a user to be able to change their password when they feel the need to (so don't set it too long, either). If a user wants to change their password before the minimum number of days elapses (for example, if your user feels that their account may have been compromised), they can always have you change it for them. However, if you make your password requirements too much of an inconvenience, it can also work against you.

Next, we should discuss setting a **Password Policy**. After all, forcing your users to change their passwords does little good if they change it to something simple, such as `abc123`. A password policy allows you to force requirements on your users for things such as length, complexity, and so on.

To configure options for password requirements, let's first install the required **Pluggable Authentication Module** (**PAM**):

```
# apt-get install libpam-cracklib
```

Next, let's take a look at the following file. Feel free to open it with a text editor, as we'll need to edit it:

```
/etc/pam.d/common-password
```

An extremely important tip while modifying configuration files related to authentication (such as password requirements, `sudo` access, SSH, and so on) is to always keep a root shell open at all times while you make changes, and in another shell, test those changes. Do not log out of your `root` window until you are 100% certain that your changes have been thoroughly tested. While testing a policy, make sure that your users can not only log in, but also your admins as well. Otherwise, you may remove your ability to log in to a server and make changes.

To enable a **History Requirement** for your passwords (meaning, the system remembers the last several passwords a user has used, preventing them from reusing them), we can add the following line to the file:

```
password        required                        pam_pwhistory.so
remember=99 use_authok
```


www.it-ebooks.info

In the example `config` line, I'm using `remember=99`, which (as you can probably guess) will cause our system to remember the last 99 passwords for each user and prevent them from using those passwords again. If you configured a minimum password age earlier, for example 5 days, it would take the user 495 days to cycle back to their original password if you take into account that the user changes his or her password once every 5 days, 99 times. That pretty much makes it impossible for the user to utilize their old passwords.

Another field worth mentioning within the `/etc/pam.d/common-password` file is the section that reads `difok=3`. This configuration details that at least three characters have to be different before the password is considered acceptable. Otherwise, the password would be deemed too similar to the old one and refused. You can change this value to whatever you like; the default is normally `5` but Ubuntu defaults it to `3` in their implementation of this `config` file. In addition, you'll also see obscure mentioned in the file as well, which prevents simple passwords from being used (such as common dictionary words and so on).

Setting a password policy is a great practice to increase the security of your server. However, it's also important to not get carried away. In order to strike a balance between security and user frustration, the challenge is always to create enough restrictions to increase your security, while trying to minimize the frustration of your users. Of course, the mere mention of the word "password" to a typical user is enough to frustrate them, so you can't please everyone. But in terms of overall system security, I'm sure your users will appreciate the fact that they can be reasonably sure that you as an administrator have taken the necessary precautions to keep their (and your company's) data safe. When it all comes down to it, use your best judgment.

Configuring administrator access with sudo

By now, we've already used `sudo` quite a few times in this book. At this point, you should already be aware of the fact that `sudo` allows you to execute commands as if you were logged in as `root`. However, we haven't had any formal discussion about it, yet nor have we discussed how to actually modify which of your user accounts are able to utilize `sudo`.

www.it-ebooks.info

On all Linux systems, you should protect your `root` account with a strong password and limit it to be used by as few people as possible. Using `sudo` is an alternative to using `root`, so you can give your administrators access to perform `root` tasks with `sudo` without actually giving them your `root` password. In fact, `sudo` allows you to be a bit more granular. Using `root` is basically all or nothing—if someone knows the `root` password and the `root` account is enabled, that person is not limited and can do whatever they want. With `sudo`, that can also be true, but you can actually restrict some users to use only particular commands with `sudo` and therefore limit the scope of what they are able to do. For example, you could give an admin access to install software updates but not allow them to reboot the server.

By default, members of the `sudo` group are able to use `sudo` without any restrictions. Basically, members of this group can do anything `root` can do (which is everything). During installation, the user account you created was made a member of `sudo`. To give additional users access to `sudo`, all you would need to do is add them to the `sudo` group:

```
# usermod -aG sudo <username>
```

Not all distributions utilize the `sudo` group by default, or even automatically install `sudo`. Other distributions require you to install `sudo` manually and may use another group (such as wheel) to govern access to `sudo`.

But again, that gives those users access to everything, and that may or may not be what you want. To actually configure `sudo`, we use the `visudo` command. This command assists you with editing `/etc/sudoers`, which is the configuration file that governs `sudo` access. Although you can edit `/etc/sudoers` yourself with a text editor, configuring `sudo` in that way is strongly discouraged. The `visudo` command checks to make sure your changes follow the correct syntax and helps prevent you from accidentally destroying the file. This is a very good thing, because if you did make some errors in the `/etc/sudoers` file, you may wind up in a situation where no one is able to gain administrative control over the server. And while there are ways to recover from such a mistake, it's certainly not a very pleasant situation to find yourself in! So, the takeaway here it to never edit the `/etc/sudoers` file directly; always use `visudo` to do so.

www.it-ebooks.info

The following output is what I receive when I make an error while using `visudo`:

```
>>> /etc/sudoers: syntax error near line 32 <<<
What now?
Options are:
  (e)dit sudoers file again
  e(x)it without saving changes to sudoers file
  (Q)uit and save changes to sudoers file (DANGER!)
```

Thankfully, I used `visudo` and didn't edit that file manually! Otherwise, if I had edited the file manually, I may not have known that I made an error. You shouldn't necessarily rely on `visudo` catching your mistakes, but it is a nice safety net, just in case.

When you run `visudo` from your shell, you are brought into a text editor with the `/etc/sudoers` file opened up. You can then make changes to the file and save it like you would any other text file. By default, Ubuntu Server opens up the nano text editor when you use `visudo`. With nano, you can save changes using *Ctrl* + *W*, and you can exit the text editor with *Ctrl* + *X*. However, you don't have to use nano if you prefer something else. If you fancy the vim text editor, you can use the following command to use `visudo` with vim:

```
# EDITOR=vim visudo
```

Anyway, so `visudo` allows you to make changes to who is able to access `sudo`. But how do you actually make these changes? Go ahead and scroll through the `/etc/sudoers` file that `visudo` opens and you should see a line similar to the following:

```
%sudo   ALL=(ALL:ALL) ALL
```

This is the line of configuration that enables `sudo` access to anyone who is a member of the `sudo` group. You can change the group name to any that you'd like, for example, perhaps you'd like to create a group called `admins` instead. If you do change this, make sure that you actually create that group and add yourself and your staff to be members of it before you log off; otherwise, it would be rather embarrassing if you found yourself locked out of administrator access to the server.

Of course, you don't have to enable access by group. You can actually call out a username instead. As an example of this, we also have the following line in the file:

```
root    ALL=(ALL:ALL) ALL
```


www.it-ebooks.info

Here, we're calling out a username (in this case, `root`), but the rest of the line is the same as the one I pointed out before. While you can certainly copy this line and paste it one or more times (substituting `root` for a different username), to grant access to others, using the group approach is really the best way. It's easier to add and remove users from a group (such as the `sudo` group), rather than use `visudo` each time.

So, at this point, you're probably wondering what the options on `/etc/sudoers` configuration lines actually mean. So far, both examples used `ALL=(ALL:ALL) All`. In order to fully understand `sudo`, understanding the other fields is extremely important, so let's go through them (using the `root` line again as an example).

The first `ALL` means that `root` is able to use `sudo` from any terminal. The second `ALL` means that `root` can use `sudo` to impersonate any other user. The third `ALL` means that `root` can impersonate any other group. Finally, the last `ALL` refers to what commands this user is able to do; in this case, any command he or she wishes.

To help drive this home, I'll give some additional examples. Here's a hypothetical example:

```
charlie    ALL=(ALL:ALL) /usr/sbin/reboot,/usr/sbin/shutdown
```

Here, we're allowing user `charlie` to execute the `reboot` and `shutdown` commands. If user `charlie` tries to do something else (such as install a package), he will receive an error message:

```
Sorry, user charlie is not allowed to execute '/usr/bin/apt-get install
tmux' as root on ubuntu-server.
```

However, if `charlie` wants to use the `reboot` or `shutdown` commands on the server, he will be able to do so because we explicitly called out those commands while setting up this user's `sudo` access. We can limit this further by changing the first `ALL` to a machine name, in this case, `ubuntu-server` to reference the host name of the server I'm using for my examples. I've also changed the command that `charlie` is allowed to run:

```
charlie    ubuntu-server=(ALL:ALL) /usr/bin/apt-get
```

Now, the `charlie` is only able to use `apt-get`. He can use `apt-get update`, `apt-get dist-upgrade`, and any other subset of `apt-get`. But if he tries to reboot the server, remove protected files, add users, or anything else we haven't explicitly set, he will be prevented from doing so.

www.it-ebooks.info

We have another problem, though. We're allowing `charlie` to impersonate other users. This may not be completely terrible given the context of installing packages (impersonating another user would be useless unless that user also has access to install packages), but it's bad form to allow this unless we really need to. In this case, we could just remove the `(ALL:ALL)` from the line altogether to prevent `charlie` from using the `-u` option of `sudo` to run commands as other users:

```
charlie    ubuntu-server= /usr/bin/apt-get
```

On the other hand, if we actually do want `charlie` to be able to impersonate other users (but only specific users), we can call out the username and group that `charlie` is allowed to act on behalf of by setting those values:

```
charlie    ubuntu-server=(dscully:admins) ALL
```

In that example, `charlie` is able to run commands on behalf of user `dscully` and group `admins`.

Of course, there is much more to `sudo` than what I've mentioned in this section. Entire books could be written about `sudo` (and have been), but 99% of what you will need for your daily management of this tool involves how to add access to users, while being specific about what each user is able to do. As a best practice, use groups when you can (for example, you could have an `apt` group, `reboot` group, and so on) and be as specific as you can regarding who is able to do what. This way, you're not only able to keep the `root` account private (or even better, disabled), but you also have more accountability on your servers.

Setting permissions on files and directories

In this section, all the user management we've done in this chapter so far all comes together. We've learned how to add accounts, manage accounts, and secure them but we haven't actually done any work regarding managing the resources as far as who is able to access them. In this section, I'll give you a brief overview of how permissions work in Ubuntu Server and then I'll provide some examples for customizing them.

I'm sure by now that you understand how to list the contents of a directory with the `ls` command. When it comes to viewing permissions, the `-l` flag is especially handy, as the output that the long listing provides allows us to view the permissions of an object:

```
ls -l
```


www.it-ebooks.info

The following are some example file listings:

```
-rw-rw-rw- 1 doctor doctor    5 Jan 11 12:52 welcome
-rw-r--r-- 1 root root     665 Feb 19  2014 profile
-rwxr-xr-x 1 dalek dalek      35125 Nov  7  2013 exterminate
```

In each line, we see several fields of information. The first column is our permission string for the object (for example, `-rw-r--r--`). We also see the link count for the object (second column), the user that owns the file (third), the group that owns the file (fourth), the size in bytes (fifth), the last date the file was modified (sixth), and finally the name of the file.

Keep in mind that depending on how your shell is configured, your output may look different and the fields may be in different places. For the sake of our discussion on permissions, what we're really concerned with is the permissions string, as well as the owning user and group. In this case, we can see that the first file (named `welcome`) is owned by a user named doctor. The second file is named `profile` and is owned by `root`. Finally, we have a file named `exterminate` owned by a user named `dalek`.

With these files, we have the permission strings of `-rw-rw-rw-`, `-rw-r--r--`, and `-rwxr-xr-x` respectively. If you haven't worked with permissions before, these may seem strange, but it's actually quite easy when you break them down. Each permission string can be broken down into four groups, as I'll show you in the following table:

Object type	User	Group	World
-	rw-	rw-	rw-
-	rw-	r--	r--
-	rwx	rwx	r-x

I've broken down each of the three example permission strings into four groups. Basically, I split them each at the first character and then again every third. The first section of the permission string is just one character. In each of these examples, it's just a single hyphen. This refers to what type the object is. Is it a directory? A file? A link? In our case, each of these permission strings are files, because the first position of the permission strings are all hyphens. If the object were a directory, the first character would've been a `d` instead of a `-`. If the object were a link, this field would've been `l` (lower-case L) instead.

www.it-ebooks.info

In the next group, we have three characters, `rw-`, `rw-`, and `rwx` respectively, in the second column of each object. This refers to the permissions that apply to the user that owns the file. For example, here is the first permission string again:

```
-rw-rw-rw- 1 doctor doctor    5 Jan 11 12:52 welcome
```

The third field shows us that `doctor` is the user that owns the file. Therefore, the second column of the permission string (`rw-`) applies specifically to the user `doctor`. Moving on, the third column of permissions is also three characters. In this case, `rw-` again. This section of the permissions string refers to the group that owns the file. In this case, the group is also named `doctor`, as you can see in column four of the string. Finally, the last portion of the permission string (`rw-` yet again, in this case) refers to world, also known as other. This basically refers to anyone else other than the owning user and owning group. Therefore, literally everyone else gets `rw-` permissions on the object.

Individually, `r` stands for read and `w` stands for write. Therefore, we can read the second column (`rw-`), indicating the user (`doctor`) has access to read and write to this file. The third column (`rw-` again) tells us the group `doctor` also has read and write access to this file. The fourth column of the permission string is the same, so anyone else would also have read and write permissions to the file as well.

The third permission string I gave as an example looks a bit different. Here it is again:

```
-rwxr-xr-x 1 dalek dalek      35125 Nov  7  2013 exterminate
```

Here, we see the `x` attribute set. The `x` attribute refers to the ability to execute the file as a script. So, with that in mind, we know that this file is actually a script and is executable by users, groups, and others. Given the filename of `exterminate`, this is rather suspicious and if it were a real file, we'd probably want to look into it.

If a permission is not set, it will simply be a single hyphen where there would normally be `r`, `w`, or `x`. This is the same as indicating that a permission is disabled. Here are some examples:

- `rwx`: Object has read, write, and execute permissions set for this field
- `r-x`: Object has read enabled, write disabled, and execute enabled for this field
- `r--`: Object has read enabled, write disabled, and execute disabled for this field
- `---`: Object has no permissions enabled for this field

www.it-ebooks.info

Bringing this all the way home, here are a few more permission strings:

```
-rw-r--r-- 1 sue accounting      35125 Nov  7  2013 budget.txt
drwxr-x--- 1 bob sales      35125 Nov  7  2013 annual_projects
```

For the first of these examples, we see that `sue` is the owning user of `budget.txt` and that this file is assigned a group of accounting. This object is readable and writable by `sue` and readable by everyone else (group and other). This is probably bad, considering this is a budget file and is probably confidential. We'll change it later.

The `annual_projects` object is a directory, which we can tell from the `d` in the first column. This directory is owned by user `bob` and group `sales`. However, since this is a directory, each of the permission bits have different meanings. In the following two tables, I'll outline the meanings of these bits for files and again for directories:

Files

Bit	Meaning
`r`	The file can be read
`w`	The file can be written to
`x`	The file can be executed as a program

Directories

Bit	Meaning
`r`	The contents of the directory can be viewed
`w`	Contents of the directory can be altered
`x`	The user or group can use `cd` to go inside the directory

As you can see, permissions are read differently depending on their context; whether they apply to a file or a directory. In the example of the `annual_projects` directory, `bob` has `rwx` permissions to the directory. This means that user `bob` can do everything (view the contents, add or remove contents, and use `cd` to move the current directory of his shell into the directory). In regards to a group, members of the `sales` group are able to view the contents of this directory and `cd` into it. However, no one in the `sales` group can add or remove items to or from the directory. On this object, other has no permissions set at all. This means that no one else can do anything at all with this object, not even view its contents.

www.it-ebooks.info

So, now we understand how to read permissions on files and directories. That's great, but how do we alter them? As I mentioned earlier, the `budget.txt` file is readable by everyone (other). This is not good because the file is confidential. To change permissions on an object, we will use the `chmod` command. This command allows us to alter the permissions of files and directories in a few different ways.

First, we can simply remove read access from `sue`'s budget file by removing the read bit from the other field. We can do that with the following example:

```
chmod o-r budget.txt
```

If we are currently not in the directory where the file resides, we need to give a full path:

```
chmod o-r /home/sue/budget.txt
```

But either way, you probably get the idea. With this example, we're removing the `r` bit from other (`o-r`). If we wanted to add this bit instead, we would simply use `+` instead of `-`. Here are some additional examples of `chmod` in action:

- `chmod u+rw <filename>`: The object gets `rw` added to the `user` column
- `chmod g+r <filename>`: The owning group is given read access
- `chmod o-rw <filename>`: Other is stripped of the `rw` bits

In addition, you can also use octal point values to manage and modify permissions. This is actually the most common method of altering permissions. I like to think of this as a scoring system. That's not what it is, but it makes it a lot easier to understand to think of each type of access as having its own value. Basically, each of the permission bits (`r`, `w`, and `x`) have their own octal equivalent, as follows:

- Read: `4`
- Write: `2`
- Execute: `1`

www.it-ebooks.info

With this style, there are only a few possibilities for numbers you can achieve when combining these octal values (each can only be used once). Therefore, we can get `0`, `1`, `2`, `4`, `5`, `6`, and `7` by adding (or not adding) these numbers in different combinations. Some of them you'll almost never see, such as an object having write access but not read. For the most part, you'll see `0`, `4`, `6`, and `7` used with `chmod` most often. For example, if we add `Read` and `Write`, we get `6`. If we add `Read` and `Execute`, we get `5`. If we add all three, we get `7`. If we add no permissions, we get `0`. We repeat this for each column (`User`, `Group`, and `Other`) to come up with a string of three numbers. Here are some examples:

- `600`: User has read and write (4+2). No other permissions are set. This is the same as `-rw-------`.
- `740`: User has read, write, and execute. Other has nothing. This is the same as `-rwxr-----`.
- `770`: Both user and group has full access (read, write, execute). Other has nothing. This is the same as `-rwxrwx---`.
- `777`: Everyone has everything. This is the same as `-rwxrwxrwx`.

Going back to `chmod`, we can use this numbering system in practice:

- `chmod 600 filename.txt`
- `chmod 740 filename.txt`
- `chmod 770 filename.txt`

Hopefully you get the idea. If you wanted to change the permissions of a directory, the `-R` option may be helpful to you. This makes the changes recursive, meaning that you'll not only make the changes to the directory, but all files and directories underneath it in one shot:

```
chmod 770 -R mydir
```

While using `-R` with `chmod` can save you some time, it can also cause trouble if you have a mix of directories and files underneath the directory you're changing permissions on. The previous example gives permissions `770` to `mydir` and all of its contents. If there are files inside, they are now given executable permissions, since `7` includes the execute bit (value of `1`). This may not be what you want. We can use the `find` command to differentiate these. While `find` is out of the scope of this chapter, it should be relatively simple to see what the following commands are doing and how they may be useful:

```
find /path/to/dir/ -type f -exec chmod 644 {} \;
find /path/to/dir/ -type d -exec chmod 755 {} \;
```


www.it-ebooks.info

Basically, in the first example, the `find` command is locating all files (`-type f`) in `/path/to/dir/` and everything it finds, it executes `chmod 644` against it. The second example is locating all directories in this same path and changing them all to permissions `755`. The `find` command isn't covered in detail here because it easily deserves a chapter of its own, but I'm including it here because hopefully these examples are useful and will be handy for you to include in your own list of useful commands.

Finally, we'll need to know how to change ownership of files and directories. It's often common that a particular user needs to gain access to an object, or perhaps we need to change the owning group as well. We can change user and group ownership of a file or directory with the `chown` command. As an example, if we wanted to change the owner of a file to `sue`, we could do the following:

```
chown sue myfile.txt
```

In the case of a directory, we can also use the `-R` flag to change ownership of the directory itself, as well as all the files and directories it may contain:

```
chown -R sue mydir
```

If we would like to change the group assignment to the object, we would follow the following syntax:

```
chown sue:sales myfile.txt
```

Notice the colon separating the user and the group. With that command, we established that we would like user `sue` and group `sales` to own this resource. Again, we could use `-R` if the object were a directory and we wanted to make the changes recursive.

Another command worth knowing is the `chgrp` command, which allows you to directly change group ownership on a file. To use it, you can execute the `chgrp` command along with the group you'd like to own the file, followed by the username. For example, our previous `chown` command can be simplified to the following, since we were only modifying the group assignment of that file:

```
# chown sales myfile.txt
```

Just like with the `chown` command, we can use the `-R` option with `chgrp` to make our changes recursively, in the case of a directory.

www.it-ebooks.info

Well, there you have it. You should now be able to manage permissions of the files and directories on your server. If you haven't worked through permissions on a Linux system before, it may take a few tries before you get the hang of it. The best thing for you to do is to practice. Create some files and directories (as well as users) and manage their permissions. Try to remove a user's access to a resource and then try to access that resource as that user anyway and see what errors you get. Fix those errors and work through more examples. With practice, you should be able to get a handle on this very quickly.

Summary

In the field, managing users and permissions is something you'll find yourself doing quite a bit. New users will join your organization, while others will leave, so this is something that will become ingrained in your mental tool-set. Even if you're the only person using your servers, you'll find yourself managing permissions for applications as well, given the fact that processes cannot function if they don't have access to their required resources. In this chapter, we took a lengthy dive into managing users, groups, and permissions. We worked through creating and removing users, assigning permissions, and managing administrative access with `sudo`. Practice these concepts on your server. When you get the hang of it, I'll see you in our next chapter, where we'll discuss all things related to storage.

www.it-ebooks.info

3
Managing Storage Volumes

When it comes to storage on our servers, it seems as though we can never get enough. While hard disks are growing in capacity every year, and high capacity disks are cheaper than ever, our servers gobble up available space quickly. As administrators of servers, we always do our best to order servers with ample storage, but business needs evolve over time, and no matter how well we plan, a successful business will always need more. While managing your servers, you'll likely find yourself adding additional storage at some point. But managing storage is more than just adding new disks every time your current one gets full. Planning ahead is also important, and technologies such as LVM will make your job much easier as long as you start using it as early as you possibly can.

In this chapter, I'll walk you through the concepts you'll need to know in order to manage storage and volumes on your server. This discussion will include:

- Viewing disk usage
- Adding additional storage volumes
- Partitioning and formatting volumes
- Mounting and unmounting storage volumes
- Understanding the `/etc/fstab` file
- Managing `swap`
- Utilizing LVM volumes
- Using symbolic and hard links

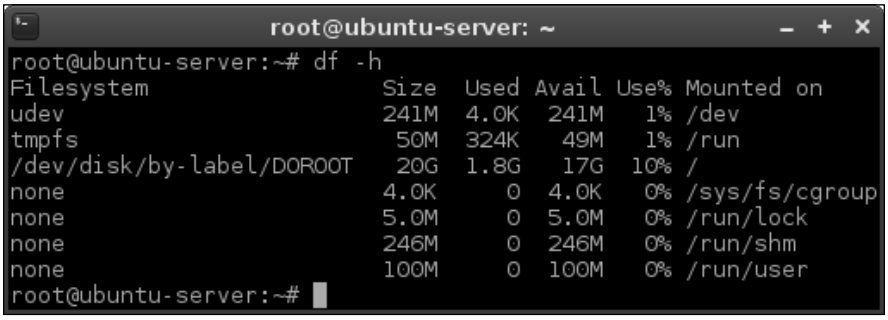
www.it-ebooks.info

Viewing disk usage

Keeping tabs on your storage is always important, as no one enjoys getting a call in the middle of the night that a server is having an issue, much less something that could've been avoided, such as a filesystem growing too close to being full. Managing storage on Linux systems is easy once you master the related tools, the most useful of which I'll go over in this section. In particular, we'll answer the question "what's eating all my free space?" and I'll provide you with some examples of how to find out.

First, the `df` command. This command is likely always going to be your starting point in situations where you don't already know which volume is becoming full. When executed, it gives you a high-level overview, so it's not necessarily useful when you want to figure out who or what in particular is hogging all your space. However, when you just want to list all your mounted volumes and see how much space is left on each, `df` fits the bill. By default, it shows you the information in bytes. However, I find it easier to use the `-h` option with `df` so that you'll see information that's a bit easier to read. Go ahead and give it a try:

```
df -h
```



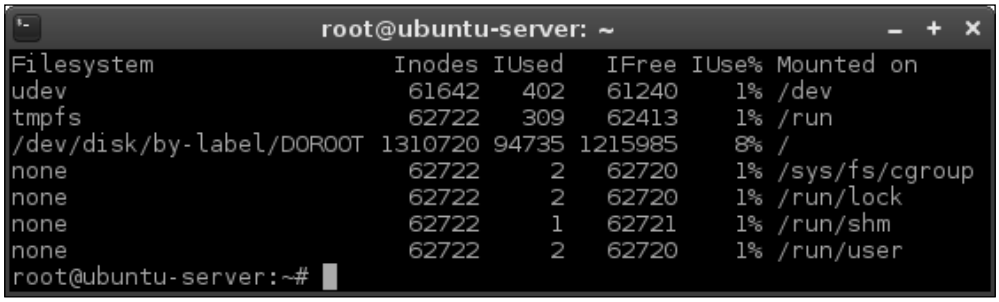
Output from the df -h command

The output will look different depending on the types of disks and mount points on your system. In the example I showed earlier, I took the information from a **Digital Ocean VPS**. In my case, you'll see that the root filesystem is located on a disk called `DOROOT`, and it is currently only utilizing 10% of the space allocated for it. As you can see from the example, `df` gives you some very useful information for starting your investigation. It shows you the `Filesystem`, its `Size`, how much space is `Used`, how much is available (`Avail`), how much percentage is being used (`Use%`), and the filesystem the device is `Mounted on`.

www.it-ebooks.info

While investigating disk utilization, it's also important to understand the concepts of **inodes**. While going deep into the concept of inodes is beyond the scope of this book, the basics are simple enough and can save you a lot of trouble, in particular in situations where an application is reporting that your disk is full but the `df -h` command shows plenty of free space is available. Think of the concept of an inode as a type of database object, containing metadata for the actual items you're storing. Information stored in inodes are details such as the owner of the file, permissions, last modified date, and type (whether it is a directory or a file).

The problem with inodes is that you can only have a limited number of them on any storage media. This number is usually extremely high and hard to reach. In the case of servers, though, where you're possibly dealing with hundreds of thousands of files, the inode limit can become a real problem. I'll show you some output from one of my servers to help illustrate this:



Output from the df -i command

The command I executed to see the output in the previous screenshot was `df -i`. As you can see, the `-i` option of `df` gives us information regarding inodes instead of the actual space used. In this example, my root filesystem has a total of `1310720` inodes available, of which `94735` are used and `1215985` are free. If you have a system that's reporting a full disk (though you see plenty of space is free when running `df -h`), it may actually be an issue with your volume running out of inodes. In this case, the problem would not be the size of the files on your disk, but rather the sheer number of files you're storing. In my experience, I've seen this happen from mail servers becoming bound (millions of stuck emails, with each email being a file), as well as unmaintained log directories. It may seem as though having to contend with an inode limit is unbecoming of a legendary platform such as Linux, though as I mentioned earlier, this limit is very hard to reach—unless something is very, very wrong.

www.it-ebooks.info

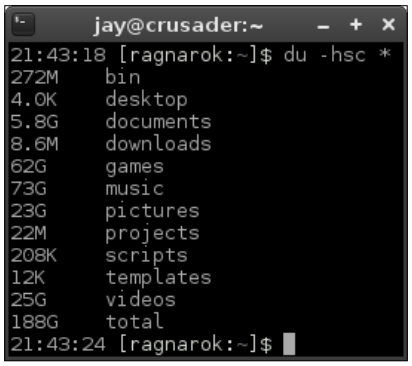
The next step in investigating what's gobbling up your disk space is finding out what is using it all up. At this stage, there are a multitude of tools you can use to investigate. The first I'll mention is the `du` command, which is able to show you how much space a directory is using. Using `du` against directories and subdirectories will help you narrow down the problem. Like `df`, we can also use the `-h` option with `du` to make our output easier to read. By default, `du` will scan the current working directory your shell is attached to and give you a list of each item within the directory, the total space each item consists of, as well as a summary at the end.

The `du` command is only able to scan directories that its calling user has permission to scan. If you run this as a non-root user, then you may not be getting the full picture. Also, the more files and subdirectories that are within your current working directory, the longer this command will take to execute. If you have an idea where the resource hog might be, try to `cd` into a directory further in the tree to narrow your search down and reduce the amount of time the command will take.

The output of `du -h` can often be more verbose than you actually need in order to pinpoint your culprit and can fill several screens. To simplify it, my favorite variation of this command is the following:

```
du -hsc *
```

Basically, you would run `du -hsc *` within a directory that's as close as possible to where you think the problem is. The `-h` option, as we know, gives us human readable output (essentially, giving us output in the form of megabytes, gigabytes, and so on). The `-s` option gives us a summary and `-c` provides us with a total amount of space used within our current working directory. The following screenshot shows this output from my desktop (yes, I have a lot of games and music):



Example output from du -hsc *

www.it-ebooks.info

As you can see, the information provided by `du -hsc *` is a nice, concise summary. From the output, we know which directories within our working directory are the largest. To further narrow down our storage woes, we could `cd` into any of those large directories and run the command again. After a few runs, we should be able to narrow down the largest files within these directories and make a decision on what we want to do with them. Perhaps we can clean unnecessary files or add another disk. Once we know what is using up our space, we can decide what we're going to do about it.

At this point in reading this book, you're probably under the impression that I have some sort of strange fixation on saving the best for last. You'd be right. I'd like to finish off this section by introducing you to one of my favorite applications, the **Ncurses Disk Usage Utility** (or more simply, the `ncdu` command). The `ncdu` command is one of those things that administrators who constantly find themselves dealing with disk space issues learn to love and appreciate. In one go, this command gives you not only a summary of what is eating up all your space, it also gives you an ability to traverse the results without having to run a command over and over while manually traversing your directory tree. You simply execute it once and then you can navigate the results and drill down as far as you need.

To use `ncdu`, you will need to install it as it doesn't come equipped on Ubuntu by default:

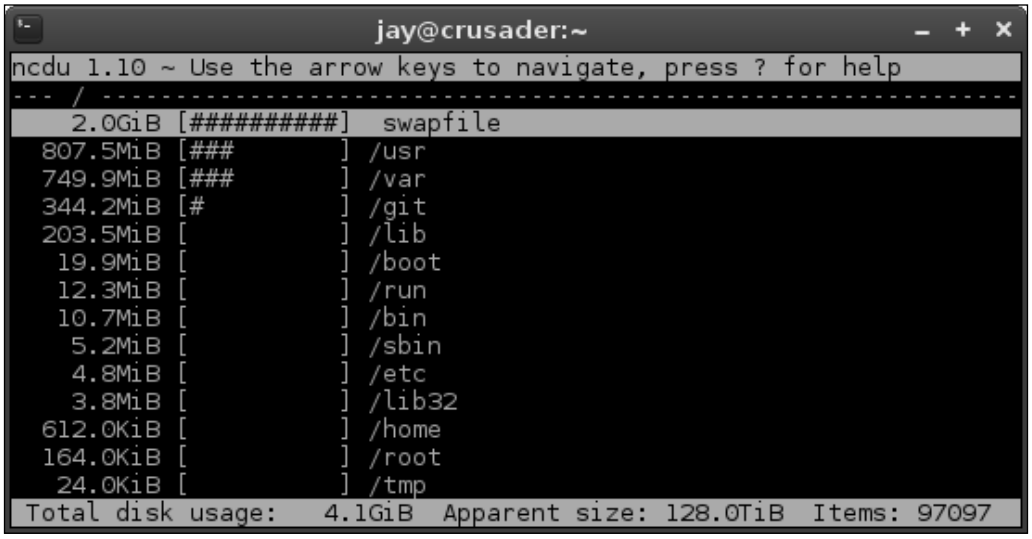
```
# apt-get install ncdu
```

Once installed, simply execute `ncdu` in your shell from any starting directory of your choosing. When done, simply press *q* on your keyboard to quit. Like `du`, `ncdu` is only able to scan directories that the calling user has access to. You may need to run it as `root` to get an accurate portrayal of your disk usage.

You may want to consider using the `-x` option with `ncdu`. This option will limit it to the current filesystem, meaning it won't scan network mounts or additional storage devices; it'll just focus on the device you started the scan on. This can save you from scanning areas that aren't related to your issue.

www.it-ebooks.info

When executed, `ncdu` will scan every directory from its starting point onward. When finished, it will give you a menu-driven layout allowing you to browse through your results:



ncdu in action

To operate `ncdu`, you move your selection (indicated with a long white highlight) with the up and down arrows on your keyboard. If you press *Enter* on a directory, `ncdu` switches to showing you the summary of that directory, and you can continue to drill down as far as you need. In fact, you can actually delete items and entire folders by pressing *d*. Therefore, `ncdu` not only allows you to find what is using up your space, it also allows you to take action as well!

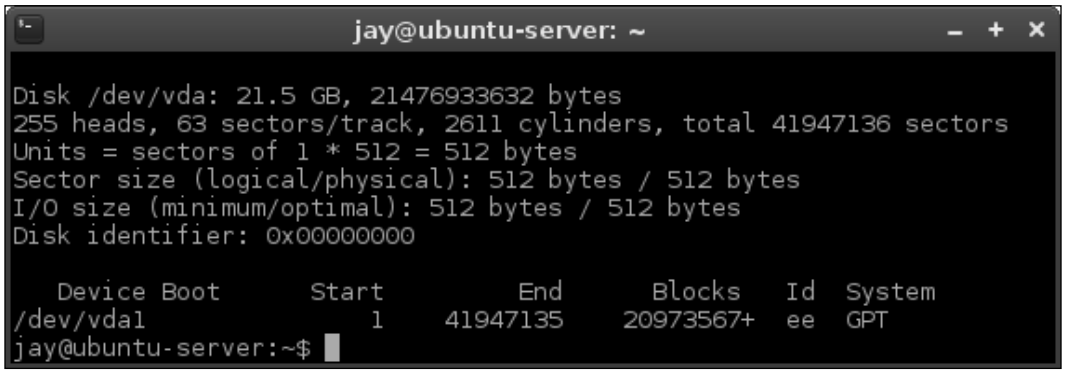
Adding additional storage volumes

At some point or another, you'll reach a situation where you'll need to add additional storage to your server. On physical servers, we can add additional hard disks, and on virtual or cloud servers, we can add additional virtual disks. Either way, in order to take advantage of the extra storage we'll need to determine the name of the device, format it, and mount it. In the case of LVM (which we'll discuss later in this chapter), we'll have the opportunity to expand an existing volume, often without a server reboot being necessary.

www.it-ebooks.info

When a new disk is attached to our server, it will be detected by the system and given a name. In most cases, the naming convention of `/dev/sda`, `/dev/sdb`, and so on will be used. In other cases (such as virtual disks), this will be different, such as `/dev/vda`, `/dev/xda`, and possibly others. The naming scheme usually ends with a letter, incrementing to the next letter with each additional disk. The `fdisk` command is normally used for creating and deleting partitions, but it will allow us to determine which device name our new disk received. Basically, the `fdisk -l` command will give you the info, but you'll need to run it as root or with `sudo`:

```
# sudo fdisk -l
```



Output of the fdisk -l command, showing a device of /dev/vda1

I always recommend running this command before and after attaching a new device. That way, it will be obvious which device name is the one that's new. Once you have the name of the device, we will be able to interact with it and set it up. There's an overall process to follow when adding a new device, though. When adding additional storage to your system, you should ask yourself the following questions:

- How much storage do you need? If you're adding a virtual disk, you can usually make it any size you want, as long as you have enough space remaining in the pool of your hypervisor.
- After you attached it, what device name did it receive? As I mentioned, run the `fdisk -l` command as `root` to find out. Another trick is to use the following variation of the tail command, with which the output will update automatically as you add the disk. Just start the command, attach the disk, and watch the output. When done, press *Ctrl* + *C* on your keyboard:

  ```
  # tail -f /var/log/dmesg
  ```


www.it-ebooks.info

- How do you want it formatted? At the time of writing, the `Ext4` filesystem is the most common. However, for different workloads, you may consider other options (such as `XFS`). When in doubt, use `Ext4`, but definitely read up on the other options to see if they may benefit your use case. `ZFS` is another option that's new in version 16.04 of Ubuntu, which you may also consider for additional volumes. We'll discuss formatting later in this chapter.

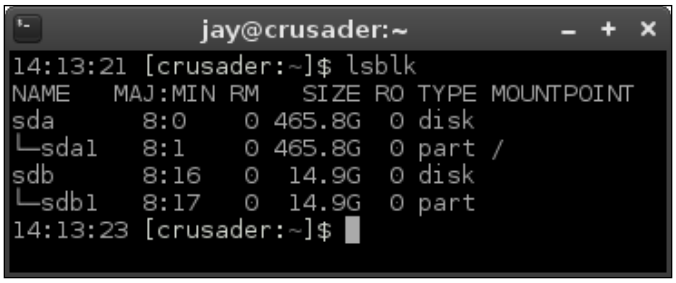
It may be common knowledge to you by now, but the word filesystem is a term that can have multiple meanings on a Linux system depending on its context, and may confuse newcomers. We use it primarily to refer to the entire file and directory structure (the Linux filesystem), but it's also used to refer to how a disk is formatted for use with the distribution (for example, the `Ext4` filesystem).

- Where do you want it mounted? The new disk needs to be accessible to the system and possibly users, so you would want to mount (attach) it to a directory on your filesystem where your users or your application will be able to use it. In the case of LVM, which we also discuss in this chapter, you're probably going to want to attach it to an existing storage group. You can come up with your own directory for use with the new volume. But later on in this chapter, I'll discuss a few common locations.

With regards to how much space you should add, you would want to research the needs of your application or organization and find a reasonable amount. In the case of physical disks, you don't really get a choice beyond deciding which disk to purchase. In the case of LVM, you're able to be more frugal, as you can add a small disk to meet your needs (you can always add more later). The main benefit of LVM is being able to grow a filesystem without a server reboot. For example, you can start with a 30 GB volume and then expand it in increments of 10 GB by adding additional 10 GB virtual disks. This method is certainly better than adding a 200 GB volume all at once when you're not completely sure all that space will ever be used. LVM can also be used on physical servers as well, but would most likely require a reboot anyway since you'd have to open the case and physically attach a hard drive.

The device name, as we discussed, is found with the `fdisk -l` command. You can also find the device name of your new disk with the `lsblk` command. One benefit of `lsblk` is that you don't need `root` privileges and the information it returns is simplified. Either works fine:

www.it-ebooks.info



The lsblk command in action, showing two disks with one partition each

On a typical server, the first disk (basically, the one that you installed Ubuntu Server on) will be given a device name of `/dev/sda`. Additional disks will be given the next available name, such as `/dev/sdb`, `/dev/sdc`, and so on. You'll also need to know the partition number as well. Device names for disk will also have numbers at the end, representing individual partitions. For example, the first partition of `/dev/sda` will be given `/dev/sda1`, while the second partition of `/dev/sdc` will be given `/dev/sdc2`. These numbers increment and are often easy to predict. As I mentioned before, your device naming convention may vary from server to server, especially if you're using a RAID controller or virtualization host such as VMWare or XenServer. If you haven't created a partition on your new disk yet, you will not see any partition numbers.

Next, you need to consider which filesystem to use. Ext4 is the most common filesystem type but many others exist, and new ones are constantly being created. At the time of writing, there are up and coming filesystems such as **B-tree file system** (**Btrfs**) being developed, while ZFS is not actually new but is new to Ubuntu (it's very common in the BSD community and has been around for a long time). Btrfs and ZFS are both not considered ready for stable use in Linux due to the fact that Btrfs is relatively new and ZFS is newly implemented in Ubuntu. At this point, it's believed that Ext4 won't be the default filesystem forever, as several are vying for the crown of becoming its successor. As I write this, though, I don't believe that Ext4 will be going away anytime soon.

An in-depth look at all the filesystem types is beyond the scope of this book due to the sheer number of them. One of the most common alternatives to Ext4 is XFS, which is a great filesystem if you plan on dealing with very large individual files or massive multi-terabyte volumes. XFS volumes are also really good for database servers due to their additional performance. In general, stick with Ext4 unless you have a very specific use case that gives you a reason to explore alternatives.

www.it-ebooks.info

Finally, the process of adding a new volume entails determining where you want to mount it and adding the device to the `/etc/fstab` file, which will help with automatically mounting it each time the server is booted (which is convenient but optional). We'll discuss the `/etc/fstab` file and mounting volumes in a later section. Basically, Linux volumes are typically mounted inside an existing directory on the filesystem, and from that point on, you'll see free space for that volume listed when you execute the `df -h` command we worked through earlier.

It's very typical to use the `/mnt` directory as a top-level place to mount additional drives. If you don't have an application that requires a disk to be mounted in a specific place, a subdirectory of `/mnt` is a reasonable choice. I've also seen administrators make their own top-level directory, such as `/store`, to house their drives. When a new volume is added, a new directory would be created underneath the top-level directory. For example, you could have a backup disk attached to `/mnt/backup` or a file archive mounted at `/store/archive`. The directory scheme you choose to use is entirely up to you.

I just mentioned mounting a backup disk at `/mnt/backup`, which those of you with more experience may be thinking is a terrible idea. As a rule of thumb, adding another internal disk is not technically considered a valid backup location. If the entire server were to catastrophically fail, all internal disks may be a victim and also fail. However, you're certainly not limited to adding internal disks. In a typical Linux network, you could also mount external disks (such as USB 3.0 external hard disks or network attached storage devices) as well. The method of adding disks is mostly the same regardless of which type of disk you're adding (local, external, network, and so on). Just as before, external devices will need to be formatted and given a location on which to be mounted. Mounting volumes will be discussed later on in this chapter.

Partitioning and formatting volumes

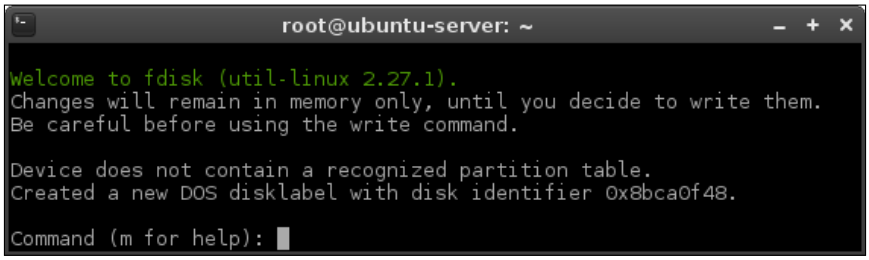
In order to utilize a disk, it must be partitioned and formatted. The `fdisk` command does much more than just show us what partitions are available, it allows us to manage them as well. In this section, I'll walk you through partitioning as well as formatting new volumes.

In order to begin the process of partitioning a disk, we'll use the `fdisk` command as root, using a device name as an option. For example, if you have a disk such as `/dev/sdb` that needs to be configured, you would execute the following:

```
# fdisk /dev/sdb
```


www.it-ebooks.info

Note that I didn't include a partition number here, as `fdisk` works with the disk directly (and we also have yet to create any partitions). In this section, I'm assuming you have a disk that has yet to be partitioned, or one you won't mind wiping out. When executed correctly, `fdisk` will show you an introductory message and give you a prompt:



Main prompt of fdisk

At this point, you can press *m* on your keyboard for a menu of possible commands you can execute. In this example, I'll walk you through the commands required to set up a new disk for the first time.

I'm sure it goes without saying, but I have to make sure you're aware of the destructive possibilities of `fdisk`. If you run `fdisk` against the wrong drive, irrecoverable data loss may result. It's common for an administrator to memorize utilities such as `fdisk` to the point where using it becomes muscle-memory. But always make sure that you take the time to make sure that you're running such commands against the appropriate disk.

Before we continue with creating a new partition, some discussion is required with regards to MBR and GPT partition tables. When creating partitions, you'll have the option to use an MBR partition table or a GPT partition table. GPT is the newer standard, while MBR has been around for quite some time and is probably what you've been using if you've ever created partitions in the past. By default, `fdisk` will create a partition table in the MBR format. But with MBR partition tables, you have some limitations to consider. First, MBR only allows you to create up to four primary partitions. In addition, it also limits you to using somewhere around 2 TB of a disk. If the capacity of your disk is 2 TB or less, this won't be an issue. However, disks larger than 2 TB are becoming more and more common. GPT doesn't have a 2 TB restriction, so if you have a very large disk, the decision between MBR and GPT has pretty much been made for you. In addition, GPT doesn't have a restriction of up to four primary partitions, as `fdisk` with a GPT partition table will allow you to create up to 128. It's certainly no wonder why GPT is fast becoming the new standard! It's only a matter of time before GPT becomes the default, so unless you have good reason not to, I recommend using it if you have a choice.

www.it-ebooks.info

When you first enter the `fdisk` prompt, you can press *m* to access the menu, where you'll see toward the bottom that you have a few options in regards to which partition style you'd like to use. If you make no selection and continue with the process of creating a partition, it will default to MBR. Instead, you can press *g* at the prompt to specifically create a GPT partition table, or to to switch back to MBR. Note that this will obviously wipe out the current partition table on the drive, so hopefully you weren't storing anything important on the disk.

Continuing on, after you've made your choice and created either an MBR or GPT partition table, we're ready to proceed. Next, at the `fdisk` prompt, type *n* to tell `fdisk` that you would like to create a new partition. Then, you'll be asked if you would like to create a primary or extended partition (if you've opted for MBR). With MBR, you would want to choose primary for the first partition and then you can use extended for creating additional partitions. If you've opted for GPT, this prompt won't appear, as it will create your partition as primary no matter what.

The next prompt that will come up will ask you for the partition number, defaulting to the next available number. Press *Enter* to accept the default. Afterwards, you'll be asked for the first sector for the partition to use (accept the default of 2048) and then the last sector to use. If you press *Enter* to accept the default last sector, your partition will consist of all the free space that was remaining. If you'd like to create multiple partitions, don't accept the default at this prompt. Instead, you can clarify the size of your new partition by giving it the number of megabytes or gigabytes to use. For example, you can enter `20G` here to create a partition of 20 GB.

At this point, you'll be returned to the `fdisk` prompt. To save your changes and exit `fdisk`, press *w* and then *Enter*. Now if you run the `fdisk -l` command as `root`, you should see the new partition you created. Here is some example output from my server, where I just partitioned an 8 GB virtual disk with a single partition:

```
Device  Boot  Start   End           Sectors       Size Id   Type
/dev/sdb1        2048  16777215 16775168   8G    83 Linux
```

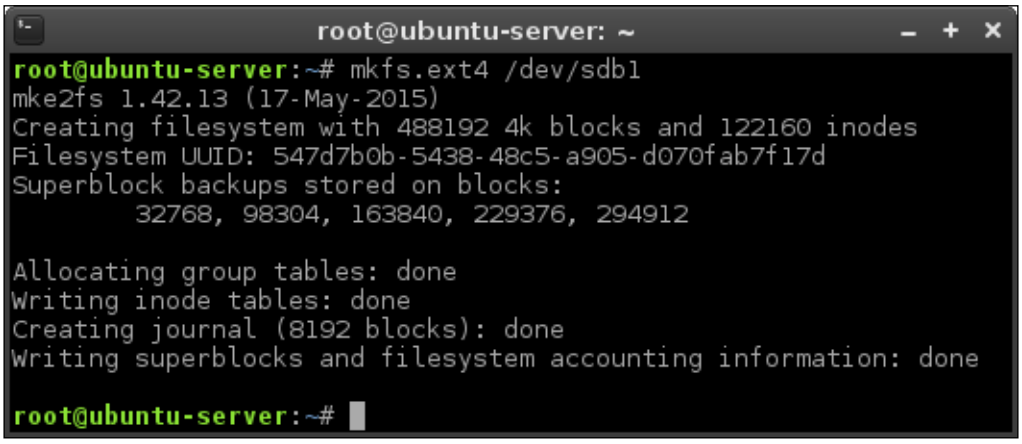
After you create your partition layout for your new disk and you're satisfied with it, you're ready to format it. If you've made a mistake or you want to redo your partition layout, you can do so by entering the `fdisk` prompt again and then pressing *g* to create a new GPT layout or *o* to create a new MBR layout. Then, continue through the steps again to partition your disk. Feel free to practice this a few times until you get the hang of the process.

Formatting is done with the `mkfs` command. To format a device, you execute `mkfs` with a period (`.`), followed by the type of filesystem you would like to format the target as. The following example will format `/dev/sdb1` as `Ext4`:

```
# mkfs.ext4 /dev/sdb1
```


www.it-ebooks.info

Your output will look similar to mine in the following screenshot:



Main prompt of fdisk

If you've opted for a filesystem type other than Ext4, you can use that in place of Ext4 when using `mkfs`. The following example creates an XFS filesystem instead:

```
# mfs.xfs /dev/sdb1
```

So, now that we've created one or more partitions and formatted them, we're ready to mount the newly created partition(s) on our server. In the next section, I'll walk you through mounting and unmounting storage volumes.

Mounting and unmounting storage volumes

Now that you've added a new storage volume to your server and have formatted it, you can mount the new device so that you can start using it. To do this, we use the mount command. This command allows you to attach a storage device (or even a network share) to a local directory on your server. This directory must be empty. The mount command, which we'll get to practice with an example very shortly, basically just requires you to designate a place (directory) for it to be mounted. But where should you mount the volume?

www.it-ebooks.info

Normally, there are two directories created by default in your Ubuntu Server installation that exist for the purposes of mounting volumes: `/mnt` and `/media`. While there is no hard rule as far as where media needs to be mounted, these two directories exist as part of the **Filesystem Hierarchy Standard** (**FHS**), which is a special specification defining the default directories and storage locations (as well as their intended purposes) on Linux systems. While an in-depth review of the FHS is beyond the scope of this book, it's worth bringing up because the purpose of the `/mnt` and `/media` directories are defined within this specification. The FHS defines `/mnt` as a mount point for a temporarily mounted filesystem, and `/media` as a mount point for removable media.

In plain English, this means that the intended purpose of `/mnt` is for storage volumes you generally keep mounted most of the time, such as additional hard drives, virtual hard disks, and network attached storage. The FHS document uses the term "**temporary**" when describing `/mnt`, but I'm not sure why that's the case since this is generally where things are mounted which your users and applications will use. Perhaps "temporary" just refers to the fact that the system doesn't require this mount in order to boot, but who knows. In regards to `/media`, the FHS is basically indicating that removable media (flash drives, CD-ROM media, external hard drives, and so on) are intended to be mounted here.

However, it's important to point out that where the FHS indicates you should mount your extra volumes is only a suggestion. No one is going to force you to follow it, and the fate of the world isn't dependent on your choice. With the mount command, you can literally mount your extra storage anywhere that isn't already mounted or full of files. You could even create the directory `/kittens` and mount your disks there and you won't suffer any consequences other than a few chuckles from your colleagues.

Often, organizations will come up with their own scheme for where to mount extra disks. Although I personally follow the FHS designation, I've seen companies use a custom common mount directory such as `/store`. Whatever scheme you use is up to you; the only suggestion I can make is to be as consistent as you can from one server to another, if only for the sake of sanity.

The mount command generally needs to be run as `root`. While there is a way around that (you can allow normal users to mount volumes, but we won't get into that just yet), it's usually the case that only root can or should be mounting volumes. As I mentioned, you'll need a place to mount these volumes, so when you've come up with such a place, you can mount a volume with a command similar to the following:

```
# mount /dev/sdb1 /mnt/vol1
```


www.it-ebooks.info

In that example, I'm mounting device `/dev/sdb1` to directory `/mnt/vol1`. Of course, you'll need to adjust the command to reference the device you want to mount and where you want to mount it. If you don't remember which devices exist on your server, you can list them with:

```
# fdisk -l
```

Normally, the `mount` command wants you to issue the `-t` option with a given type. In my case, the mount command would've been the following had I used the `-t` option, considering my disk is formatted with `ext4`:

```
# mount /dev/sdb1 -t ext4 /mnt/vol1
```

In that example, I used the `-t` option along with the type of filesystem I formatted the device with. In the first example, I didn't. This is because, in most cases, the mount command is able to determine which type of filesystem the device uses and adjust itself accordingly. Thus, most of the time, you won't need the `-t` option. In the past, you almost always needed it, but it's easier nowadays. The reason I bring this up is because if you ever see an error when trying to mount a filesystem that indicates an invalid filesystem type, you may have to specify this. Feel free to check the man pages for the mount command for more information regarding the different types of options you can use.

When you are finished using a volume, you can unmount it with the `umount` command:

```
# umount /mnt/vol1
```

The `umount` command, which also needs to be run as `root`, allows you to disconnect a storage device from your filesystem. In order for this command to be successful, the volume should not be in use. If it is, you may receive a device or resource busy error message.

The downside to manually mounting devices is that they will not automatically remount themselves the next time your server boots. In order to make the mount automatically come up when the system is started, you'll need to edit the `/etc/fstab` file, which I'll walk you through in the next section.

Understanding the /etc/fstab file

The `/etc/fstab` file is a very critical file on your Linux system. As I mentioned in the last section, you can edit this file to call out additional volumes you would like automatically mount at boot time. However, the main purpose of this file is also to mount your main filesystem as well, so if you make a mistake while editing it, your server will not boot. Definitely be careful.

www.it-ebooks.info

When your system boots, it looks at this file to determine where the root filesystem is. In addition, the location of your swap area is also read from this file and mounted at boot time as well. Your system will also read any other mount points listed in this file, one per line, and mounts them. Basically, just about any kind of storage you can think of can be added to this file and automatically mounted. Even network shares from Windows servers can be added here.

For an example, here is the content of `/etc/fstab` on one of my servers:

```
# / was on /dev/sda1 during installation
UUID=53ce034d-c06e-4bc3-b784    /     ext4    errors=remount-ro 0       1
# swap was on /dev/sda5 during installation
UUID=c9468709-008f-49f8-8b5e     none  swap    sw              0       0
```

I've shortened the `UUID` columns a bit in order to save space on this page, but all the relevant info is still present. When you install Ubuntu Server, the `/etc/fstab` file is created for you and populated with a line for each of the partitions you created during installation. On the server I used to grab the example `fstab` content, I have two volumes: one for the root filesystem and one for swap.

Each volume is designated with a **Universally Unique Identifier** (**UUID**) instead of the normal `/dev/sdaX` naming convention you are more than likely used to. In my output, you can see that some comments (lines beginning with #) were created by the installer to let me know which volume refers to which device. For example, `UUID c9468709-008f-49f8-8b5e` refers to my swap partition, `/dev/sda5` (according to the comment). The concept of a `UUID` has been around for a while, but there's nothing stopping you from replacing the `UUID` with the actual device names (`/dev/sda1` and `/dev/sda5` in my case). If you were to do that, the server would still boot and you probably wouldn't notice a difference (assuming you didn't make a typo).

Nowadays, UUIDs are preferred over common device names due to the fact that the names of devices can change depending on where you place them physically (which SATA port, USB port, and so on) or how you order them (in the case of virtual disks). Add to this the fact that removable media can be inserted or removed at any time and you have a situation where you don't really know what name each device is going to receive. For example, your external hard drive may be named `/dev/sdb1` on your system now, but it may not be the next time you mount it if something else you connect claims the name of `/dev/sdb1`. This is where the concept of UUIDs comes in handy. A UUID of a device will not change, no matter what order it's installed in. You can easily list the UUIDs of your volumes with the `blkid` command:

```
blkid
```


www.it-ebooks.info

The output will show you the UUID of each device attached to your system, and you can use this command any time you add new volumes to your server to list your UUIDs. This is also the first step in adding a new volume to your /etc/fstab file. While I did say that using UUIDs is not required, it's definitely recommended and can save you from trouble later.

Each line of an fstab entry is broken down into several columns, each separated by spaces. There isn't a set number of spaces necessary to separate each column; in most cases, spaces are only used to line up each column to make them easier to read. However, at least one space is required.

In the first column of the example fstab file, we have the UUID of each device. In the second column, we have the location we want the device to be mounted to. In the case of the root filesystem, this is /, which (as you know) is the beginning of the Linux filesystem. The second entry (for swap) has a mount point of none, which means that a mount point is not necessary for this device. In the third column, we have the filesystem type, the first being ext4, and the second being swap.

In the fourth column, we have a list of options for each mount separated by a comma. In this case, we only have one option for each of the example lines. With the root filesystem, we have an option of errors=remount-ro, which tells the system to remount the filesystem as read-only if an error occurs. Such an issue is rare, but will keep your system running in read-only mode if a problem occurs. The swap partition has a single option of sw. There are many other options that can be used here, so feel free to consult the man pages for a list. We will go over some of these options in this section.

The fifth and sixth columns refer to dump and pass respectively, which are 0 and 1 in the first example line and 0 and 0 in the last. Dump is almost always 0 and can be used with a backup utility to determine whether the filesystem should be backed up (0 for no, 1 for yes). In most cases, just leave this at 0 since this is rarely ever used by anything nowadays. The Pass field refers to the order in which fsck will check the file systems. The fsck utility scans hard disks for filesystem errors in the case of a system failure or a scheduled scan. The possible options for Pass are 0, 1, or 2. With 0, the partition is never checked with fsck. If set to 1, the partition is checked first. Partitions with a Pass of 2 are considered second priority and checked last. As a general rule of thumb, use 1 for your main filesystem and 2 for all others.

Now that we understand all the columns of a typical fstab entry, we can work through adding another volume to the fstab file. First, we need to know the UUID of the volume we would like to add (assuming it's a hard disk or virtual disk). Again, we do that with the blkid command:

```
# blkid
```


www.it-ebooks.info

Next, we need to know where we want to mount the volume. Go ahead and create the directory now, or use an existing directory if you wish. With this information, we can add a new entry to `fstab`. To do so, we create a new line after all the others. In my example server, an entry for `/dev/sdb1` will look like the following:

```
# Extra storage for user apps
UUID=b15ec9ce-7b59-45c6 /mnt/vol1  ext4  defaults 0 2
```

In my example, I created a comment line with a little note about what the extra volume will be used for. It's always a good idea to leave comments, so other administrators will have a clue regarding the purpose of the extra storage. Then, I created a new line with the `UUID` of the volume, the mount-point for the volume, the filesystem type, defaults option, and a dump/pass of `0` and `2`.

The defaults option I've not mentioned before. By using defaults as your mount option in `fstab`, your mount will be given several useful options in one shot, without having to list them individually. Among the options included with defaults are the following, which are worth an explanation:

- `rw`: Device will be mounted read/write
- `exec`: Allow files within this volume to be executed as programs
- `auto`: Automatically mount the device at boot time
- `nouser`: Only root is able to mount the filesystem
- `async`: Output to the device should be asynchronous

Depending on your needs, the options included with defaults may or may not be ideal. Instead, you can call the options out individually, separated by commas, choosing only the ones you need. For example, with regards to `rw`, you may not want users to be allowed to change content. In fact, I strongly recommend that you use `ro` (read-only) instead, unless your users have a very strong use case for needing to make changes to files. I've actually learned this the hard way, where I've experienced an entire volume getting completely wiped out (and no one admitted to clearing the contents). This volume included some very important company data. From that point on, I mandated `ro` being used for everything, with a separate `rw` mount created, with only a select few (very responsible) people having access to it.

The `exec` option may also not be ideal. For example, if your volume is intended for storing files and backups, you may not want scripts to be run from that location. By using the inverse of exec (`noexec`), you can prevent scripts from running to create a situation where users are able to store files on the volume but not execute programs that are stored there.

www.it-ebooks.info

Another option worth explanation is auto. The `auto` option basically tells your system to automatically mount that volume whenever the system boots or when you enter the following command:

```
# mount -a
```

When executed, `mount -a` will mount any entry in your `/etc/fstab` file that has the auto option set. If you've used defaults as an option for the mount, those will be mounted as well since defaults implies auto. This way, you can mount all filesystems that are supposed to be mounted without rebooting your server (this command will not disrupt anything that is already mounted).

The opposite of the auto option is `noauto`, which can be used instead. As you can probably guess by the name, an entry in `fstab` with the `noauto` option will not be automatically mounted and will not be mounted when you run `mount -a`. Instead, entries with this option will need to be mounted manually.

You may be wondering, then, what the point is of including an entry in `fstab` just to use `noauto`. To explain this better, here is an example `fstab` entry with `noauto` being used:

```
UUID=e51bcc9e-45dd-45c7 /mnt/ext_disk  ext4  rw,noauto 0 0
```

Here, let's say that I have an external disk that I only mount when I'm performing a backup. I wouldn't want this device mounted automatically at boot time (I may not always have it connected to the server), so I use the `noauto` option. But since I do have an entry for it in `/etc/fstab`, I can easily mount it any time with the following command:

```
# mount /mnt/ext_disk
```

Notice that I didn't have to include the device name, only where I want the device mounted. The mount command knows what device I'm referring to, since I have an entry for a device to be mounted at `/mnt/ext_disk`. This saves me from having to type the device name each time I want to mount the device. So, in addition to mounting devices at boot time, the `/etc/fstab` file also becomes a convenient place to declare devices that may be used on an on-demand basis but aren't always attached.

One final option I would like to cover before we move on is users. When used with a mount in `/etc/fstab`, this allows regular users (users other than `root`) to mount and unmount the filesystem. This way, `root` or `sudo` will not be necessary at all for a mount used with this option. Use this with care, but it can be useful if you have a device with non-critical data you don't mind your users having full control over when mounting and unmounting.

www.it-ebooks.info

While the concept of a text file controlling which devices are mounted on the system may seem odd at first, I think you'll appreciate being able to view a single file in order to find out everything that should be mounted and where it should be mounted. As long as all administrators add even on-demand devices to this file, it can be a convenient place to get an overview of the filesystems that are in use on the server.

Managing swap

As I mentioned earlier in this book, the concept of swap is one that has been a subject of great debate, with some administrators even omitting it outright. When your server is running smoothly, you should never need swap. If your server does start consuming a lot of swap, it's definitely cause for concern. Swap space is a special partition (or file) that is used for memory when your physical memory gets depleted. It's not uncommon for a handful of megabytes here and there to be consumed by swap, but when serious swap space is used, it's usually the sign of a struggling server.

Theoretically, this space should remain idle and (for the most part) unused. In a perfect world, your server will always run efficiently, processes will never misbehave, and users will never do anything stupid. But in the real world, swap space is here to catch you when you fall. While some believe that swap shouldn't be necessary given how much cheaper memory is nowadays, I counter with the fact that disk space is also cheap, so it shouldn't be a problem to attribute some of your disk to swap. If nothing else, it may save you from a reboot when things go awry.

There is an exception to this rule, though. When it comes to the Raspberry Pi, you should never have swap space if you can help it. Traditionally, the Raspberry Pi images do not include a swap partition. You could create one, but I recommend that you don't. The reason for this is because the Pi uses SD cards for storage, which are orders of magnitude slower than traditional hard disks. While the Raspberry Pi might seem fast enough with regards to IO when performing average tasks, swap would be so slow on these devices that it would just be more trouble than it's worth and end up working against you.

As you probably noticed from our earlier discussion, swap space is declared in our `/etc/fstab` file. In most cases, you would've chosen to create a swap partition during installation. You could, of course, add a swap partition later. In that case, you would add it just like you would any other partition, and you could use my output earlier in this chapter for a sample `fstab` line for swap. However, the `mount -a` command we discussed earlier won't activate swap, even after adding it to the file. To activate swap, we would use the `swapon` command:

```
# swapon -a
```


www.it-ebooks.info

When run, the `swapon -a` command will find your swap partition in `/etc/fstab`, mount it, and activate it for use. The inverse of this command is `swapoff -a`, which deactivates and unmounts your swap partition. It's rare that you'd need to disable swap, unless of course you were planning on deleting your swap partition, creating a larger one, and then mounting the new partition. If you find out that your server has an inadequate swap partition size, that may be a course of action you would take.

When you check your free memory (hint: execute `free -m`), you'll see swap listed whether you have it or not, but when swap is deactivated, you will see all zeros for the size totals.

In my experience with cloud servers, it's not uncommon that you won't have a swap partition at all. In the case of **Digital Ocean** (a popular cloud server provider), their default Ubuntu images do not include swap. Their lowest server tier for Ubuntu Server includes just 512 MB of RAM, and with no swap, you could easily encounter a situation where an application stops working when it hits the physical memory ceiling (as I've personally experienced). For situations where you aren't given swap by default, and you aren't able to add additional disks, you can create a swap file instead.

To do so, you'll first create the actual file to be used as swap. This can be stored anywhere, but `/swapfile` is typically ideal. You can use the `fallocate` command to create the actual file:

```
# fallocate -l 2G /swapfile
```

Here, I'm creating a 2 GB swap file, but feel free to make yours whatever size you want in order to fit your needs. Next, we need to prepare this file to be used as swap:

```
# mkswap /swapfile
```

Now, we have a handy-dandy swap file stored in our root filesystem. Next, we'll need to mount it. As always, it's recommended that we add this to our `/etc/fstab` file. What follows is an example entry:

```
/swapfile   none   swap   sw   0 0
```

From this point, we can activate our new swap file:

```
# swapon -a
```

But what if you wanted to create a swap partition on a physical device, such as a secondary hard or virtual disk? Remember our old friend `fdisk` we've used to create partitions before. The `fdisk` utility can also be used to create a `swap` partition as well. Assuming you have a disk that hasn't yet been partitioned or formatted, what follows is an example run of `fdisk` where I create a swap partition.

www.it-ebooks.info

First, at the `fdisk` prompt, I type `n` to create a new partition:

```
Command (m for help): n
```

Then, I'm asked for the partition type, where I'll press *p* for primary:

```
Partition type
   p   primary (0 primary, 0 extended, 4 free)
   e   extended (container for logical partitions)
```

When asked for the `Partition number`, I'll leave the default at `1`, and then press *Enter*:

```
Partition number (1-4, default 1):
```

I'll accept the defaults for `First sector` as well:

```
First sector (2048-16777215, default 2048):
```

For the last sector, you can press *Enter* to claim the entire drive (which would probably be a waste). In my case, I type `+2G` to instruct `fdisk` to create a 2 GB partition:

```
Last sector, +sectors or +size{K,M,G,T,P} (2048-16777215, default
16777215): +2G
```

Then, I receive confirmation that I've created a new partition:

```
Created a new partition 1 of type 'Linux' and of size 2 GiB.
```

So far, no surprises. We've been through this song and dance before. Next, comes the neat part. We can select a different type for our partition by entering *t* at the `fdisk` prompt:

```
Command (m for help): t
```

Next, we are asked for the type. We could list all the possibilities with `L`, but I'll give you a shortcut and let you know that `82` is the partition type for swap, so press *82* and then *Enter*:

```
Partition type (type L to list all types): 82
```

I then receive another confirmation:

```
Changed type of partition 'Linux' to 'Linux swap / Solaris'.
```

Finally, I type *w* to write the changes to disk:

```
Command (m for help): w
```


www.it-ebooks.info

Now, we have a swap partition. We can mount it on our system by finding its `UUID` (hint: use the `blkid` command as `root`) and then prepare this partition to be used as swap:

```
# mkswap /dev/sdd1
```

The output of the `mkswap` command will give us the `UUID`, which we can then add to our `/etc/fstab` to ensure it's used at boot time and mount it right now with `swapon -a` as `root`. For your convenience, an example `fstab` entry for swap is as follows:

```
UUID=8d773e13-01    none            swap     sw                0        0
```

While I certainly hope you won't need to resort to using swap, I know from experience that it's only a matter of time. Knowing how to add and activate swap when you need it is definitely a good practice, but for the most part, you should be fine as long as you created an adequate swap partition during installation. I always recommend a bare minimum of 2 GB on servers, but if you can swing 8 GB or even 16 GB of your disk for this purpose, that's even better.

Utilizing LVM volumes

The needs of your organization will change with time. While we as server administrators always do our best to configure resources with long-term growth in mind, budgets and changes in policy always seem to get in our way. LVM is something that I'm sure you'll come to appreciate. In fact, technologies such as LVM are one of those things that make Linux the champion when it comes to scalability and cloud deployments. With LVM, you are able to resize your filesystems online, without needing to reboot your server.

Take the following scenario for example. Say you have an application running on a production server, a server that's so important that downtime would cost your organization serious money. When the server was first set up, perhaps you gave the application's storage directory a 100 GB partition, thinking it would never need more than that. Now, with your business growing, it's not only using a lot of space—you're about to run out! What do you do? If the server was initially set up with LVM, you could add an additional storage volume, add it to your LVM pool, and grow your partition. All without rebooting your server! On the other hand, if you didn't use LVM, you're forced to find a maintenance window for your server and add more storage the old-fashioned way.

www.it-ebooks.info

With physical servers, you can install additional hard drives and keep them on standby without utilizing them to still gain the benefit of growing your filesystem online, even though your server isn't virtual.

It's for this reason that I must stress that you should always use LVM on storage volumes in virtual servers whenever possible. Let me repeat myself. You should *always* use LVM on storage volumes when you are setting up a virtual server! If you don't, this will eventually catch up with you when your available space starts to run out and you find yourself working over the weekend to add new disks. This will involve manually syncing data from one disk to another and then migrating your users to the new disk. This is not a fun experience, believe me.

When setting up a new server, you're given the option to use LVM during installation. If you can, it's not a bad idea. But it's much more important for your storage volumes to use LVM, and by those, I mean the volumes where your users and applications will store their data. LVM is a good choice for your Ubuntu Server's root filesystem, but not required. In order to get started with LVM, there are a few concepts that we'll need to understand, specifically **Volume Groups**, **Physical Volumes**, and **Logical Volumes**.

A volume group is a namespace given to all the physical and logical volumes on your system. Basically, a volume group is the highest name that encompasses your entire implementation of an LVM setup. Think of it as a kind of container that is able to contain disks. An example of this might be a volume group named `vg-accounting`. This volume group would be used for a location for the accounting department to store their files. It will encompass the physical volumes and logical volumes that will be in use by these users. It's important to note that you aren't limited to just a single volume group; you can have several, each with their own disks and volumes.

A physical volume is a physical or virtual hard disk that is a member of a volume group. For example, the hypothetical vg-accounting volume group may consist of three 100 GB hard disks, each called a physical volume. Keep in mind that these disks are still referred to as physical volumes, even when the disks are virtual. Basically, any block device that is owned by a volume group is a physical volume.

Finally, logical volumes are similar in concept to partitions. Logical volumes can take up a portion of, or an entire, disk, but unlike standard partitions, they may also span multiple disks. For example, a logical volume can include three 100 GB disks and be configured such that you would receive a cumulative total of 300 GB. When mounted, users will be able to store files there just as they would a normal partition on a standard disk. When the volume gets full, you can add an additional disk and then grow the partition to increase its size. Your users would see it as a single storage area, even though it may consist of multiple disks.

www.it-ebooks.info

The volume group can be named anything you'd like, but I always give mine names that begin with `vg-` and end with a name detailing its purpose. As I mentioned, you can have multiple volume groups. Therefore, you can have `vg-accounting`, `vg-sales`, and `vg-frontdesk` all on the same server. Then, you assign physical volumes to each. For example, you can add a 500 GB disk to your server and assign it to `vg-sales`. From that point on, the `vg-sales` volume group owns that disk. You're able to split up your physical volumes any way that makes sense to you. Then, you can create logical volumes utilizing these physical volumes, which is what your users will use.

I think it's always best to work through an example when it comes to learning a new concept, so I'll walk you through such a scenario. In my case, I just created a local Ubuntu Server VM on my machine via VirtualBox and then I added four additional disks after I installed the distribution. Virtualization is a good way to play around with learning LVM if you don't have a server available with multiple free physical disks.

To get started with LVM, you'll first need to install the required packages, which may or may not be present on your server. The following command will install a package that will download all the required packages as dependencies:

```
# apt-get install lvm2
```

Next, we'll need to take an inventory of the disks we have available to work with. You can list them with the `fdisk -l` command as `root`. In my case, I have `/dev/sdb`, `/dev/sdc`, `/dev/sdd`, and `/dev/sde` to work with. The names of your disks will be different depending on your hardware or virtualization platform, so make sure to adjust all the following commands accordingly. To begin, we'll need to configure each disk to be used with LVM, by setting up each one as a physical volume. The `pvcreate` command allows us to create physical volumes, so we'll need to run the `pvcreate` command against all of the drives we wish to use for this purpose. Since I have four, I'll use the following to set them up:

```
# pvcreate /dev/sdb
# pvcreate /dev/sdc
# pvcreate /dev/sdd
# pvcreate /dev/sde
```

And so on, for however many disks you plan on using.

www.it-ebooks.info

To confirm that you have followed the steps correctly, you can use the `pvdisplay` command as `root` to display the physical volumes you have available on your server:

```
jay@ubuntu:~$ sudo pvdisplay
  "/dev/sdd" is a new physical volume of "8.00 GiB"
  --- NEW Physical volume ---
  PV Name               /dev/sdd
  VG Name
  PV Size               8.00 GiB
  Allocatable           NO
  PE Size               0
  Total PE              0
  Free PE               0
  Allocated PE          0
  PV UUID               UmpzNh-bTi2-KvY1-oKPn-dMXW-Qy2a-VfWiGH
```

Output of the pvdisplay command on a sample server

Although we have some physical volumes to work with, none of them are assigned to a volume group. In fact, we haven't even created a volume group yet. We can now create our volume group with the `vgcreate` command, where we'll give our volume group a name and assign our first disk to it:

```
# vgcreate vg-test /dev/sdb
```

Here, I'm creating a volume group named `vg-test` and I'm assigning it one of the physical volumes I prepared earlier (`/dev/sdb`). Now that our volume group is created, we can use the `vgdisplay` command to view details about it, including the number of assigned disks (which should now be `1`):

```
# vgdisplay
```

```
jay@ubuntu:~$ sudo vgdisplay
  --- Volume group ---
  VG Name               vg-test
  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               8.00 GiB
  PE Size               4.00 MiB
  Total PE              2047
  Alloc PE / Size       0 / 0
  Free  PE / Size       2047 / 8.00 GiB
  VG UUID               v0NlwD-Pb1A-mK2x-vrph-z2J5-03nw-OR2vom
```

Output of the vgdisplay command on a sample server

www.it-ebooks.info

For all intents and purposes, LVM is now set up and we can begin using it. All we need to do at this point is create a logical volume and format it. Our volume group can contain all or a portion of the disk we've assigned to it. With the following command, I'll create a logical volume of 2 GB, out of the 8 GB disk I added to the volume group:

```
# lvcreate -n myvol1 -L 2g vg-test
```

The command may look complicated, but it's not. In this example, I'm giving my logical volume a name of `myvol1` with the `-n` option. Since I only want to give it 2 GB of space, I use the `-L` option and then `2g` to represent 2 GB. Finally, I give the name of the volume group that this logical volume will be assigned to. You can run `lvdisplay` to see information regarding this volume:

```
jay@ubuntu:~$ sudo lvdisplay
  --- Logical volume ---
  LV Path                /dev/vg-test/myvol1
  LV Name                myvol1
  VG Name                vg-test
  LV UUID                BSVPkC-L9io-FnfZ-6MmC-3lKk-l6eV-lTtI0S
  LV Write Access        read/write
  LV Creation host, time ubuntu, 2016-05-13 12:54:29 -0400
  LV Status              available
  # open                 0
  LV Size                2.00 GiB
  Current LE             512
  Segments               1
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           252:0
```

Output of the lvdisplay command on a sample server

The next step is for us to format our logical volume so that it can be used. But first, we'll need to find the full name for our logical volume:

```
ls /dev/mapper
```

The `/dev/mapper` directory is where our volume groups will be listed. There should be only one listed here, and its name should be the same as you named your volume group in the previous step (so it should be obvious which one represents your logical volume). Record this path and name. Now that we know the name, we can format it as we would any other volume. In my case, the one I created is now located at `/dev/mapper/vg-- test-myvol1`:

```
# mkfs.ext4 /dev/mapper/vg--test-myvol1
```


www.it-ebooks.info

And now this device can be mounted as any other hard disk. I'll mount mine at `/mnt/lvm/myvol1`, but you can use any directory or name you wish:

```
# mount /dev/mapper/vg--test-myvol1 /mnt/lvm/myvol1
```

We now have an LVM configuration containing just a single disk, so this isn't very useful. The 2 GB I've given it will not likely last very long. In my case, I only used 2 GB of the physical volume I assigned to this volume group, so there is some remaining space we can use. With the following command, I can resize my logical volume to take up the remainder of the physical volume:

```
# lvextend -n /dev/mapper/vg--test-myvol1 -l 100%FREE
```

```
jay@ubuntu:~$ sudo lvextend -n /dev/mapper/vg--test-myvol1 -l 100%FREE
  Size of logical volume vg-test/myvol1 changed from 2.00 GiB (512 extents) to
6.00 GiB (1535 extents).
  Logical volume myvol1 successfully resized.
```

Extending a logical volume

Now my logical volume is using the entire physical volume I assigned to it. Be careful, though, because if I had multiple physical volumes assigned, that command would've claimed all the space on those as well, giving the logical volume a size that is the total of all the space it has available, across all its disks. You may not always want to do this, but since I only had one physical volume anyway, I don't mind. If you check your mounted disks with the `df -h` command, you should see this volume is mounted:

```
df -h
```

Unfortunately, it's not showing the extra space we've given the volume. The output of `df` is still showing the size the volume was before. That's because although we have a larger logical volume, and it has all the space assigned to it, we didn't actually resize the `ext4` filesystem that resides on this logical volume. To do that, we will use the `resize2fs` command:

```
# resize2fs /dev/mapper/vg--test-myvol1
```

Now you should see the added space as usable when you execute `df -h`. The coolest part is that we resized an entire filesystem without having to restart the server. In this scenario, if our users have got to the point where they have utilized the majority of their free space, we would be able to give them more space without disrupting their work.

www.it-ebooks.info

However, you may have additional physical volumes that have yet to be assigned to a volume group. In my example, I created four and have only used one. We can add additional physical volumes to our volume group with the `vgextend` command. In my case, I'll run this against the three remaining drives. If you have additional physical volumes, feel free to add yours with the same commands I use, but substitute my device names with yours:

```
# vgextend vg-test /dev/sdc
# vgextend vg-test /dev/sdd
# vgextend vg-test /dev/sde
```

You should see a confirmation similar to the following:

```
Volume group "vg-test" successfully extended
```

When you run `pvdisplay`, you should see the additional physical volumes attached (shown under `Cur PV` in the output). Also in the output, you can see that the amount of free space is greater than what the volume group is actually using. You'll see how much space you're using under `Alloc PE/Size` and how much space you have remaining listed under `Free PE/Size`. Now, we have some options. We could give all the extra space to our logical volume and extend it as we did before. However, I think it's better to withhold some of the space from our users. That way, if our users do use up all our available space again, we have an emergency reserve of space we could use in a pinch if we needed to while we figure out the long-term solution. The following example command will add an additional 10 GB to the logical volume:

```
# lvextend -L+10g /dev/vg-test/myvol1
```

And finally, make the free space available to the filesystem:

```
# resize2fs /dev/mapper/vg--test-myvol1
```

With very large volumes, the resize may take some time to complete. If you don't see the additional space right away, you may see it gradually increase every few seconds until all the new space is completely allocated.

As you can see, LVM is very useful in managing storage on your server. It gives you the ability to scale your server's storage as the needs of your organizations and users evolve. However, what if I told you that being able to resize your storage on command isn't the only benefit of LVM? In fact, LVM also allows you to perform snapshots as well.

www.it-ebooks.info

LVM snapshots allow you to capture a logical volume at certain point in time and preserve it. After you create a snapshot, you can mount it as you would any other logical volume and even revert your volume group to the snapshot in case something fails. In practice, this is useful if you want to test some potentially risky changes to files stored within a volume, but want the insurance that if something goes wrong, you can always undo your changes and go back to how things were. LVM snapshots allow you to do just that. LVM snapshots require you to have some unallocated space in your volume group.

However, LVM snapshots are definitely not a viable form of backup. For the most part, these snapshots are best when used as a very temporary holding area when running tests or testing out experimental software. If you recall, you were offered the option to create an LVM configuration during installation of Ubuntu Server, so therefore you can use snapshots to test how security updates will affect your server if you used LVM for your root filesystem. If the new updates start to cause problems, you can always revert back. When you're done testing, you should merge or remove your snapshot.

So, why did I refer to LVM snapshots as a temporary solution and not a backup? First, backups aren't secure if they are stored on the same server that's being backed up. It's always important to save backups off the server at least, preferably off-site. But what's worse is that if your snapshot starts to use up all available space in your volume group, it can get corrupted and stop working. Therefore, this is a feature you would use just as a means of testing something and then reverting back or deleting the snapshot when you're done experimenting.

When you create a snapshot with LVM, what happens is a new logical volume is created that is a clone of the original. Initially, no space is consumed by this snapshot. But as you run your server and manipulate files in your volume group, the original blocks are copied to the snapshot as you change them, to preserve the original logical volume. If you don't keep an eye on usage, you may lose data if you aren't careful and the logical volume will fill up.

To show this in an example, the following command will create a snapshot (called `mysnapshot`) of the `myvol1` logical volume:

```
# lvcreate -s -n mysnapshot -L 4g vg-test/myvol1
```

You should see the following output:

```
Logical volume "mysnapshot" created.
```


www.it-ebooks.info

With that example, we're using the `lvcreate` command, with the `-s` option (snapshot) and the `-n` option, where we declare a name of `mysnapshot`. We're also using the `-L` option to designate a maximum size for the snapshot, which I set to 4 GB in this case. Finally, I give it the volume group and logical volume name, separated by a forward slash (`/`). From here, we can use the `lvs` command to monitor its size.

Since we're creating a new logical volume when we create a snapshot, we can mount it as we would a normal logical volume. This is extremely useful if we want to pull a single file without having to restore the entire thing. If we would like to remove the snapshot, we can do so with the `lvconvert` command:

```
# lvconvert --merge vg-test/mysnapshot
```

The output will look similar to the following:

```
Merging of volume mysnapshot started.
myvol1: Merged: 100.0%
```

It's important to note, however, that unlike being able to resize a logical volume online, we cannot merge a snapshot while it is in use. If you do, the changes will take affect the next time it is mounted. Therefore, you can either unmount the logical volume before merging, or ummount and remount after merging. Afterwards, you'll see that the snapshot is removed the next time you run the `lvs` command.

Finally, you may be curious about how to remove a logical volume or volume group. For these purposes, you would use the `lvremove` or `vgremove` commands. It goes without saying that these commands are destructive, but are useful in a situation where you want to delete a logical volume or volume group. To remove a logical volume, the following syntax will do the trick:

```
# lvremove vg-test/myvol1
```

Basically, all you're doing is giving the `lvremove` command the name of your volume group, a forward slash, and then the name of the logical volume within that group that you would like to remove. To remove the entire volume group, the following command and syntax should be fairly self-explanatory:

```
# vgremove vg-test
```


www.it-ebooks.info

Hopefully you're convinced by now how awesome LVM is. It allows you flexibility over your server's storage that other platforms can only even dream of. The flexibility of LVM is one of the many reasons why Linux excels in the cloud market. These concepts can be difficult to grasp at first if you haven't worked with LVM before. But thanks to virtualization, playing around with LVM is easy. I recommend you practice creating, modifying, and destroying volume groups and logical volumes until you get the hang of it. If the concepts aren't clear now, they will be with practice.

Using symbolic and hard links

With Linux, we can link files to other files, which gives us quite a bit of flexibility with how we can manage our data. Symbolic and hard links are very similar, but to explain them, you'll first need to understand the concept of inodes.

We already discussed inodes earlier in this chapter. But as a refresher, an inode is a data object that contains metadata regarding files within your filesystem. Inodes are represented by an integer number, which you can view with the `-i` option of the `ls` command. On my system, I created two files: `file1` and `file2`. These files are inodes `543815` and `558287` respectively. You can see this output in the following screenshot where I run the `ls -i` command. This information will come in handy shortly.

Output of ls -i

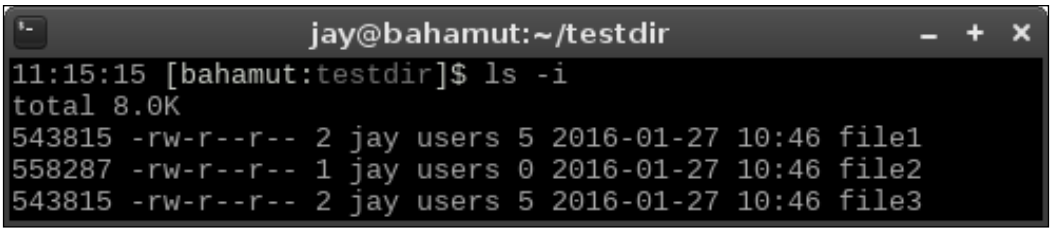
There are two types of links in Linux: **Symbolic Links** and **Hard Links**. This concept is similar in purpose to shortcuts created in graphical user interfaces. Almost all graphical operating systems have a means of creating a shortcut icon that points to another file or application. I'm sure you've seen shortcut icons to applications on Windows or Mac OS X systems. Even Linux has this same functionality in most desktop environments that are available. On a Linux server, you typically don't have a graphical environment, but you can still link files to one another using symbolic or hard links. And while links approach the concept of shortcuts differently, they pretty much serve the same purpose. Basically, a link allows us to reference a file somewhere else on our filesystem.

www.it-ebooks.info

For a practical example, let's create a hard link. In my case, I have a couple of files in a test directory, so I can create a link to any of them. To create a link, we'll use the `ln` command:

```
ln file1 file3
```

Here, I'm creating a hard link (`file3`) that is linked to `file1`. To play around with this, go ahead and create a link to a file on your system. If we use `ls` again with the `-i` option, we'll see something interesting:



Output of ls -i after creating a link

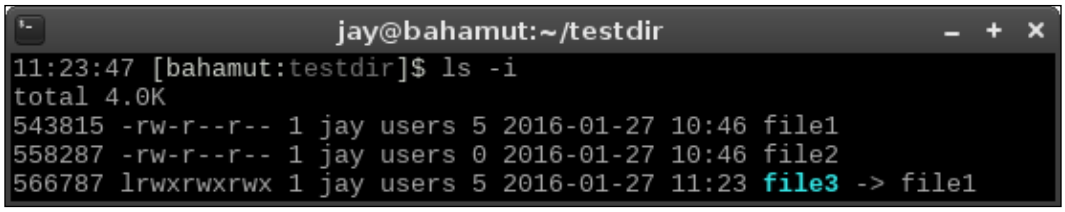
If you look closely at the output, both `file1` and `file3` have the same inode number. Essentially, they're the same file. With this hard link created, we can move `file3` into another location on the filesystem and it will still be a link to `file1`. Hard links have a couple of limitations, however. First, you cannot create a hard link to a directory, only a file. Second, this link cannot be moved to a different filesystem. That makes sense, considering each filesystem has its own inodes. Inode `543815` on my system would of course not point to the same file on another system, if this inode number even exists at all.

To overcome these limitations, we can consider using a **Symbolic Link** instead. Symbolic links (also known as **Soft Links** or **Symlinks**) can not only be moved around between filesystems (as these do not share the same inode number as the original file), we can also create a symbolic link to a directory as well. To illustrate this, let's create a symbolic link. In my case, I'll delete `file3` and recreate it as a symbolic link. We'll again use the `ln` command:

```
rm file3
ln -s file1 file3
```


www.it-ebooks.info

With the `-s` option of `ln`, I'm creating a symbolic link. First, I deleted the original hard link with the `rm` command (which doesn't disturb the original) and then created a symbolic link, also named `file3`. If we use `ls -i` again, we'll see that `file3` does not have the same inode number as `file1`:



Output of ls -i after creating a symbolic link

That's not the only thing that's different from the output, though. Notice that the file name field of the `ls` command shows that `file3` is redirecting to `file1`. At this point, the main difference from a hard link should become apparent. A symbolic link is not a clone of the original file; it's simply a pointer to the original file. Any commands you execute against `file3` are actually being run against the target that the link is pointing to.

In practice, symbolic links are incredibly useful when it comes to server administration. One example of this is with regards to configuration files. As you most likely know, system-wide configuration files for an application on a Linux server are stored in the `/etc` directory. I find that it's a best practice to use `git` for managing these configuration files, which we'll actually discuss in *Chapter 14*, *Preventing and Recovering from Disasters*. In a Linux server, you may create a central location for all the configuration files that matter to your server. For example, I usually create the `/git` directory. Inside, I create several `git` repositories for configurations that need to be under version control. For example, I may have `/git/bind` to include files for a DNS server. However, the bind daemon doesn't look in `/git/bind` for its data files, it looks (by default) in `/etc/bind`. Therefore, I can create a symbolic link (`/etc/bind`) that actually points to `/git/bind`. The bind daemon won't know the difference and I can use a common directory to house the configuration files for daemons that are under version control. Of course, this method may not be practical for a DNS server that sees a heavy amount of frequent changes (such as **Dynamic DNS**), but for a small server, this does the trick, and it works to illustrate one of the many uses for symbolic links.

www.it-ebooks.info

However, it's important not to go crazy and create a great number of symbolic links all over the filesystem. This certainly won't be a problem for you if you are the only administrator on the server, but if you resign and someone takes your place, it will be a headache for them to figure out all of your symbolic links and map where they lead to. You can certainly create documentation for your symbolic links, but then you'd have to keep track of them and constantly update documentation. My recommendation is to only create symbolic links when there are no other options, or if doing so benefits your organization and streamlines your file layout.

Getting back to the subject of symbolic links vs hard links, you're probably wondering which one you should use and when to use it. The main benefit of a hard link is that you can move either file (the link or the original) to anywhere on the same filesystem and the link will not break. This is not true of symbolic links, however. If you move the original file, the symbolic link will be pointing to a file that no longer exists at that location. Hard links are basically a mirror image of the original file (since they point to the same inode), so both will have the same file size and content. A symbolic link is merely a pointer—nothing more, nothing less.

However, even though I just spoke about several benefits of hard links, I actually recommend symbolic links for most use cases. They can cross filesystems, can be linked to directories, and are easier to determine from the output where they lead. If you move hard links around, you may forget where they originally were located or which file actually points to which. Sure, with a few commands you can find them and map them easily. But overall, symbolic links are more convenient in the long run. As long as you're mindful of recreating your symbolic link whenever you move the original file (and you use them only when you need to), you shouldn't have an issue.

Summary

Efficiently managing the storage of your servers will ensure that things continue to run smoothly, as a full filesystem is a sure fire way for everything to grind to a halt. Thankfully, Linux servers feature a very expansive tool set for managing your storage, some of which are a source of envy for other platforms. As Linux server administrators, we benefit from technologies such as LVM, and utilities such as `ncdu`, as well as many others. In this chapter, we explored these tools and how to manage our storage. We covered how to format, mount, and unmount volumes, as well as managing LVM, monitoring disk usage, and creating links. We also discussed swap, as well as managing our `/etc/fstab` file.

In our next chapter, we'll work through connecting to networks. We'll configure our server's host-name, work through examples of connecting to other servers via OpenSSH, and take a look at IP addressing.

www.it-ebooks.info

www.it-ebooks.info

4
Connecting to Networks

Linux networks are taking the industry by storm. Many big-name companies use Linux in their data centers, to the point where most people use Linux nowadays, whether they realize it or not (either directly or indirectly). The scalability of Linux in the data center lends itself very well to networking. This flexibility allows Linux to not only be useful for large server deployments, but also allows it to power routers and network services.

So far in this book, we've worked with a single Ubuntu Server instance. Here, we begin a two-part look at networking in Linux. In this chapter, we'll discuss connecting to other nodes and networks. We'll resume this exploration in *Chapter 7*, *Managing Your Ubuntu Server Network*, where we'll work on some foundational concepts that power much of the things we'll need to set up our Linux network. In this chapter, we'll take a look at:

- Setting the hostname
- Managing network interfaces
- Assigning static IP addresses
- Understanding Linux name resolution
- Understanding Network Manager
- Getting started with OpenSSH
- Getting started with SSH key management
- Simplifying SSH connections with a `~/.ssh/config` file

www.it-ebooks.info

Setting the hostname

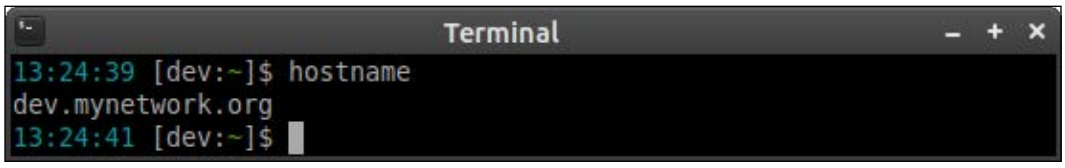
During installation, you were asked to create a hostname for your server. The default during installation is `ubuntu`, but you can (and should) come up with your own name. If you left the default, or if you want to practice changing it, we'll work through that in this section.

In most organizations, there is a specific naming scheme in place for servers and networked devices. I've seen quite a few variations, from naming servers after cartoon characters (who wouldn't want a server named daffy-duck?), to Greek Gods or Goddesses. Some companies choose to be a bit boring, and come up with naming schemes consisting of a series of characters separated by hyphens, with codes representing which rack the server is in, as well as its purpose. You can create your own naming convention if you haven't already, and no matter what you come up with, I won't judge you.

Your hostname identifies your server to the rest of the network. While the default `ubuntu` hostname is fine if you have just one host, it would get confusing really quickly if you kept the default on every Ubuntu server within your network. Giving each server a descriptive name helps you tell them apart from one another. But there's more to a server's name than its hostname, which we'll get into in *Chapter 7*, *Managing Your Ubuntu Server Network*, when we discuss DNS. But for now, we'll work through viewing and configuring the hostname, so you'll be ready to make your hostname official with a DNS assignment, when we come to it.

So, how do you view your hostname? One way is to simply look at your shell prompt; you've probably already noticed that your hostname is included there. While you can customize your shell prompt in many different ways, the default shows your current hostname. However, depending on what you've named your server, it may or may not show the entire name. Basically, the default prompt (known as a **PS1 prompt**, in case you were wondering) shows the hostname only until it reaches the first period. For example, if your hostname is `dev.mycompany.org`, your prompt will only show `dev`. To view the entire hostname, simply enter the hostname command:

```
hostname
```



Output from the hostname command

www.it-ebooks.info

Changing the hostname is fairly simple. To do this, we can use the `hostnamectl` command as root. If, for example, I'd like to change my hostname from `dev.mynetwork.org` to `dev2.mynetwork.org`, I would execute the following command:

```
# hostnamectl set-hostname dev2.mynetwork.org
```

Simple enough, but what does that command actually do? Well, I'd love to give you a fancy outline, but all it really does is change the contents of a text file (specifically, `/etc/hostname`). To see this for yourself, feel free to use the `cat` command to view the contents of this file before and after making the change with `hostnamectl`:

```
cat /etc/hostname
```

You'll see that this file contains only your hostname.

The `hostnamectl` command didn't exist before Ubuntu switched to `systemd` (versions 15.04 and earlier). If you're using a version earlier than that, you'll need to `edit /etc/hostname` manually.

Once you change your hostname, you may start seeing an error message similar to the following after executing some commands:

```
unable to resolve host dev.mynetwork.org
```

This error means that the computer is no longer able to resolve your local hostname. This is due to the fact that the `/etc/hostname` file is not the only file where your hostname is located; it's also referenced in `/etc/hosts`. Unfortunately, the `hostnamectl` command doesn't update `/etc/hosts` for you, so you'll need to edit that file yourself to make the error go away. As an example, here's what my `/etc/hosts` file looks like:

```
127.0.0.1       localhost
127.0.1.1       dev

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

In the bold area, you can see my original hostname (`dev`). By using a text editor, we can change this occurrence of the hostname.

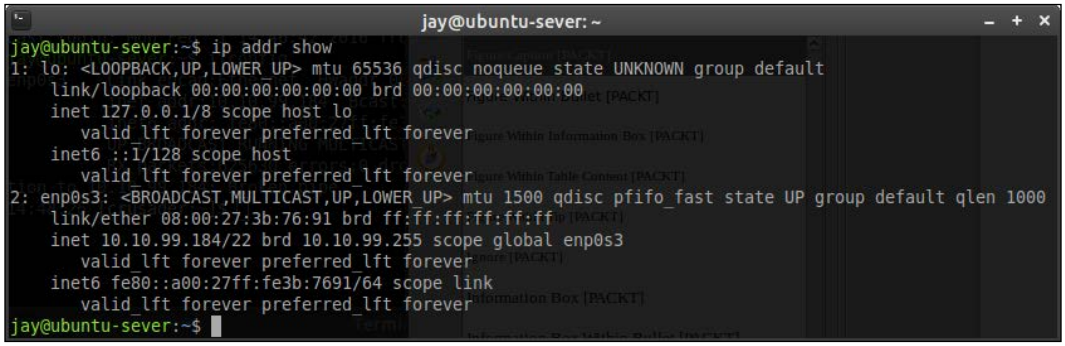
www.it-ebooks.info

Considering the extra work with changing our hostname, I see little benefit in using `hostnamectl` over manually editing `/etc/hosts` and `/etc/hostname`, so the choice is pretty much yours. You'll end up using a text editor to update `/etc/hosts` anyway, so you may as well perform the update the same way for both.

Managing network interfaces

Assuming our server's hardware has been properly detected, we'll have one or more network interfaces available for us to use. We can view information regarding these interfaces and manage them with the `ip` command. For example, we can use `ip addr` show to view our currently assigned IP address:

```
ip addr show
```



Viewing interface information with the ifconfig command

If for some reason you're not fond of typing, you can shorten this command all the way down to simply `ip a`. The output will be the same in either case. From the output, we can see several useful tidbits, such as the IP address for each device (if it has one), as well as its MAC address.

Using the `ip` command, we can also manage the state of an interface. We can bring a device down (remove its IP assignment and prevent it from connecting to networks), and then back up again:

```
# ip link set enp0s3 down
# ip link set enp0s3 up
```

In that example, I'm simply toggling the state for interface `enp0s3`. First, I'm bringing it down, and then I'm bringing it back up again.

www.it-ebooks.info

Bringing interfaces up and down is all well and good, but what's up with that naming convention? The new convention in Ubuntu 16.04 may seem a bit strange for those of you that have used earlier versions, and are more accustomed to network interface names such as `eth0`, `wlan0`, and so on. Since Ubuntu is based on Debian, it has adopted the new naming convention which will be used starting with Debian 9.0 (codenamed Stretch). Although Debian 9.0 hasn't been released yet at the time of writing, Ubuntu is still based on it, and has already adopted this change.

The new naming convention has been put in place in order to make interface naming more predictable. While you may argue that names such as `eth0` may be easier to memorize than say `enp0s3`, the change helps the name stay persistent between boots. When you add new network interfaces to a Linux system, there's always the possibility that other interface names may change as well. For example, if you have a single network card (`eth0`) and you add a second (which then becomes `eth1`), your configuration may break if the names were to get switched during the next boot. Imagine for a moment that one interface is connected to the Internet and another connected to a switch (basically, you have an Internet gateway). If the interfaces came up in the wrong order, Internet access would be disrupted for your entire office, due to the fact that the firewall rules you've written are being applied to the wrong interfaces. Definitely not a pleasant experience!

To help combat this, previous versions of Ubuntu (as well as Debian, and even CentOS), have opted to use `udev` to make the names stick. To achieve stickiness with interface names on older systems, the system would've created the following file:

```
/etc/udev/rules.d/70-persistent-net-rules
```

This file existed on older versions of some popular Linux distributions (including Ubuntu), as a workaround to this problem. This file contains some information that identifies specific qualities of the network interface, so that with each boot, it will always come up with the same name. Therefore, the card you recognize as `eth0` will always be `eth0`. If you have an older version of Ubuntu Server in use, you should be able to see this file for yourself. Here's some sample output of this file on one of my older installations:

```
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*", ATTR{address}=="01:2
2:4e:a5:f2:ec", ATTR{dev_id}=="0x0", ATTR{type}=="1", KERNEL=="eth*",
NAME="eth0"
```


www.it-ebooks.info

As you can see, it's using the MAC address of the card to identify it with `eth0`. But this becomes a small problem if I want to take an image of this machine and re-deploy it onto another server. This is a common practice—we administrators rarely start over from scratch if we don't have to, and if another server is similar enough to a new server's desired purpose, cloning it will be an option. However, when we restore the image onto another server, the `/etc/udev/rules.d/70-persistent-net-rules` file will come along for the ride. We'll more than likely find that the new server's network card will have a designation of `eth1`, even if we only have one interface. This is because the file already designated a device as `eth0`, so we'll need to correct this file ourselves in order to reclaim `eth0`. The new convention doesn't necessarily eliminate this headache either, since interface names can still change. But, at least we won't have a text file we'll need to manage.

The new naming scheme is effective as of `systemd v197` and later. For the most part, the new naming convention references the physical location of the network card on your system's bus. Therefore, the name it receives cannot change unless you were to actually remove the network card and place it in a different slot on the system's board, or change the position of the virtual network device in your hypervisor. If your machine does include a `/etc/udev/rules.d/70-persistent-net-rules` file, it will be honored and the old naming convention will be used instead. This is due to the fact that you may have upgraded to a newer version of your distribution (which features the new naming scheme, whereas your previous version didn't), so that your network devices will retain their names, thereby minimizing disruption.

As a quick overview of how the network names break down, `en` is for Ethernet, and `wl` is for wireless. Therefore, we know that the example interface I mentioned earlier (`enp0s3`) references a wired card. The `p` references which bus is being used, so `p0` refers to the system's first bus (the numbering starts at 0). Next, we have `s3`, which references PCI slot 3. Putting it together, `enp0s3` references a wired network interface card on the system's first bus, placed in PCI slot 3. The exact details of the new naming specification are beyond the scope of this chapter (and could even be a chapter of its own!), but hopefully this gives you a general idea of how the new naming convention breaks down. There's much more documentation online if you're interested in the nitty-gritty details. The important point here is that since the new naming scheme is based on where the card is physically located, it's much less likely to abruptly change. In fact, it can't change, as long as you don't physically switch the positions of your network cards inside the case.

www.it-ebooks.info

Getting back to managing our interfaces, another command worth discussion is `ifconfig`. The `ifconfig` command is part of the net-tools suite of utilities, which has been deprecated (for the most part). Its replacement is the `iproute2` suite of utilities, which includes the `ip` command we've already discussed. In summary, this basically means you should be using commands from the `iproute2` suite, instead of commands such as `ifconfig`. The problem, though, is that most administrators nowadays still use `ifconfig`, with no sign of it slowing down. In fact, the net-tools suite has been recommended for deprecation for years now, and just about every Linux distribution shipping today still has this suite installed by default. Those that don't, offer it as an additional package you can install. In the case of Ubuntu Server 16.04, net-tools commands such as `ifconfig` are alive and well.

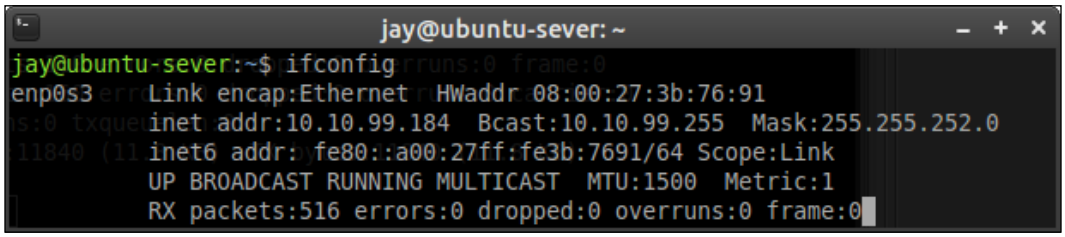
The reason commands such as `ifconfig` have a tendency to stick around so long after they've been deprecated usually comes down to the "change is hard" mentality, but quite a few scripts and programs out there are still using `ifconfig`, and therefore it's worth discussing here. Even if you immediately stop using `ifconfig`, and move to `ip` from now on, you'll still encounter this command in your travels, so you may as well know a few examples. Knowing the older commands will also help you if you find yourself on an older server.

First, when executed by itself with no options, `ifconfig` will print information regarding your interfaces like we did with `ip addr show` earlier. That seems pretty simple.

If you are unable to use `ifconfig` to view interface information using a normal user, try using the fully qualified command:

```
/sbin/ifconfig
```

The `/sbin` directory may or may not be in your `$PATH` (a set of directories your shell looks within for commands), so if your system doesn't recognize `ifconfig`, use the fully qualified command instead.



Viewing interface information with the ifconfig command

www.it-ebooks.info

Secondly, just like with the `ip` commands we practiced earlier, we can also bring an interface down or up with `ifconfig` as well:

```
# ifconfig enp0s3 down
# ifconfig enp0s3 up
```

There are, of course, other options and variations of `ip` and `ifconfig`, so feel free to look up the main pages for either if you want more information. For the purposes of this section, the main thing is to remember how to view your current IP assignments, as well as how to bring an interface up or down.

Assigning static IP addresses

With servers, it's very important that your IP addresses remain fixed and do not change for any reason. If an IP address does change (such as a dynamic lease), your users will experience an outage, services will fail, or entire sites may become unavailable. When you install Ubuntu Server, it will grab a dynamically assigned lease from your DHCP server, but after you configure the server the way you want it, it's important to get a permanent IP address in place right away. One exception to this rule is an Ubuntu-based VPS. Cloud providers that bill you for these servers will have an automatic system in place to declare an IP address for your new VPS, and will already have it configured to stick. But in the case of virtual or physical servers you manage yourself; you'll start off with a dynamic address.

In most cases, you'll have an IP address scheme in place at your office or organization, which will outline a range of IP addresses that are available for use with static assignments. If you don't have such a scheme, it's important to create one, so you will have less work to do later when you bring more servers online. We'll talk about setting up a DHCP server and IP address scheme in *Chapter 7*, *Managing Your Ubuntu Server Network*, but for now, I'll give you a few quick tips. Your DHCP server will have a range of IP addresses that will be automatically assigned to any host that requests an assignment. When setting up a static IP on a server, you'll want to make sure that the IP address is outside of the range that your DHCP server assigns. For example, if your DHCP server assigns IP's ranging from `10.10.10.100` through `10.10.10.150`, you'll want to use an IP address not included within that range for your servers.

www.it-ebooks.info

There are two ways of assigning a fixed address to a network host, including your servers. The first is by using a **Static IP assignment**, as I've already mentioned. With that method, you'll arbitrarily grab an IP address that's not being used by anything, and then configure your Ubuntu Server to use that address. In that case, your server is never requesting an IP address from your network's DHCP server. It simply obeys you and uses whatever you assign it. This is the method I'll be going over in this section.

The other way of assigning a fixed address to a server is by using a **Static Lease**. (This is also known as a **DHCP Reservation**, but I like the former because it sounds cooler). With this method, you configure your DHCP server to assign a specific IP address to specific hosts. In other words, your server will request an IP address from your local DHCP server, and your DHCP server is instructed to give a specific address to your server each time it asks for one. This is the method I prefer, which I'll go over in *Chapter 7*, *Managing Your Ubuntu Server Network*. I'll also explain why I prefer it when we get to it.

But, you don't always have a choice. It's often the case, that as a Linux administrator, you may or may not be in charge of the DHCP server. At my organization, our DHCP server runs Windows, so I don't touch it. I let the Windows administrators deal with it. Therefore, you may be given a free IP address from your network administrator, and then you'll proceed to configure your Ubuntu server to use it. To perform this configuration, we will edit the `/etc/network/interfaces` file. This file is a single place for you to manage settings for each of your network interfaces. An example of this file is below. I've left out the output regarding the loopback adapter, and what you'll see is the section relating to the primary network card on one of my servers:

```
# The primary network interface
auto enp0s3
iface enp0s3 inet dhcp
```

With this default file, we have a primary network interface (`enp0s3`), and we're allowing this interface to receive a dynamic IP assignment from a DHCP server. You can see this on the third line, where DHCP is explicitly called. If you haven't changed your `/etc/network/interfaces` file, and you're not using a VPS, yours will look very similar to this, with the interface name being the main difference.

www.it-ebooks.info

If we wanted to manually assign a static IP address, we would need to make some changes to this file. Here's an example interfaces file, configured with a static address:

```
# The primary network interface
auto enp0s3
iface enp0s3 inet static
    address 10.10.96.1
    netmask 255.255.252.0
    broadcast 10.10.96.255
    dns-search local.lan
    dns-nameservers 10.10.96.1
```

I've bolded the sections of the output that I've changed. Essentially, I've changed `dhcp` to static and then added five additional lines. With this configuration, I'm assigning a **Static IP address** to `10.10.96.1`, setting the **Subnet Mask** to `255.255.252.0`, (if you're not using subnetting, you probably want `255.255.255.0`), and the broadcast address to `10.10.96.255`. For name resolution, I'm setting the DNS search to `local.lan` (you can omit this line if you don't have a domain), and the DNS server address to `10.10.10.96.1`. If you wanted to configure a static IP address for your server, you would simply change this output to match your environment, making sure to especially change your interface name if it's not the same as mine (`enp0s3`), as well as the values for your connection.

Now that we've edited our interfaces file, we'll need to restart networking for the changes to take effect. But depending on how you're connected to the server, you may want to take some precautions first. The reason for this, is if you're connected to the server via a remote connection (such as SSH), you will lose connection as soon as we restart networking. Even worse, the second half of the network restart (the part that actually brings your interfaces back online) won't execute because the restart of networking will drop your remote connection before it would've completed. This, of course, isn't an issue if you have physical access to your server. The restart command (which I'll give you shortly), will complete just fine in that case. But with remote connections, your networking will go down if you restart it. The command to restart networking is the following:

```
# systemctl restart networking.service
```

On older Ubuntu Servers (before `systemd`), you would use the following command instead:

```
# /etc/init.d/networking restart
```


www.it-ebooks.info

In the case of remote connections, you can alleviate the issue of being dropped before networking comes back up by using `tmux`, a popular terminal multiplexer. A full run-through of `tmux` is beyond the scope of this book, but it is helpful to us in this case because it keeps commands running in the background, even if our connection to the server gets dropped. To use it, first install the package:

```
# apt-get install tmux
```

Then, activate `tmux` by simply typing `tmux` in your shell prompt.

From this point on, `tmux` is now responsible for your session. If you run a command within `tmux`, it will continue to run, regardless of whether or not you're attached to it. To see this in action, first enter `tmux` and then execute the `top` command. While `top` is running, disconnect from `tmux`. To do that, press *Ctrl* + *B* on your keyboard, release, and then press *D*. You'll exit `tmux`, but if you enter `tmux` a to reattach your session, you'll see that `top` was still running even though you disconnected. Following this same logic, you can enter `tmux` prior to executing one of the `restart` commands for your server. You'll still (probably) get dropped from your shell, but the command will complete in the background, which means that networking will go down, and then come back up with your new configuration.

The `tmux` utility is extremely powerful, and when harnessed, can really enhance your work-flow while using the Linux shell. Although a complete tutorial is outside the scope of this book, I highly recommend looking into using it. For a full walk-through, check out the book *Getting Started with tmux* by Victor Quinn, J.D at `https://www.packtpub.com/hardware-and-creative/getting-started-tmux` or check out some tutorial videos on YouTube.

With networking restarted, you should be able to immediately reconnect to the server and see that the new IP assignment has taken place by executing `ip addr show` or `ifconfig`. If, for some reason, you cannot reconnect to the server, you may have made a mistake while editing the `/etc/network/interfaces` file. In that case, you'll have to walk over to the console (if it's a physical server), or utilize your Virtual Machine manager to access the virtual console to log in and fix the problem. But as long as you've followed along and typed in the proper values for your interface and network, you should be up and running with a static IP assignment.

www.it-ebooks.info

Understanding Linux name resolution

In *Chapter 7, Managing Your Ubuntu Server Network*, we'll have a discussion on setting up a DNS server for local name resolution for your network. But before we get to that, it's also important to understand how Linux resolves names in the first place. Most of you are probably aware of the concept of a **Domain-Name Server** (**DNS**), which matches human-understandable domain names to IP addresses. This makes browsing your network (as well as the Internet) much easier. However, DNS isn't always the first thing that your Linux server will use when resolving names.

For more information on the order in which Ubuntu Server checks resources to resolve names, feel free to take a look at the `/etc/nsswitch.conf` file. There's a line in this file that begins with the word `hosts`. Here is the output of the relevant line from the file on my server:

```
hosts:          files dns
```

In this case, the server is configured to first check local files, and then DNS if the request isn't found. This is the default order, and I see little reason to make any changes here (but you certainly can). Specifically, the file the server will check is `/etc/hosts`. If it doesn't find what it needs there, it will move on to DNS.

There are many other lines in the `nsswitch.conf` file, but I won't discuss them here as they are out of scope of the topic of this section.

The `/etc/hosts` file, which we briefly discussed while working with our hostname, tells our server how to resolve itself (it has a hostname mapping to the localhost IP of `127.0.0.1`), but you are also able to create additional names to IP mappings here as well. For example, if I had a server (`minecraftserver.local.lan`) at IP `10.10.96.124`, I could add the following line to `/etc/hosts` to make my machine resolve the server to that IP each time, without it needing to consult a DNS server at all:

```
10.10.96.124 minecraftserver
```


www.it-ebooks.info

In practice, though, this is usually not a very convenient method by which to configure name resolution. Don't get me wrong, you can certainly list your servers in this file along with their IP addresses, and your server would be able to resolve those names just fine. The problem comes with the fact that this method doesn't scale. The name mappings apply only to the server you've made the `/etc/hosts` changes on; other servers wouldn't benefit since they would only check their own `/etc/hosts` file. You could add a list of servers to the hosts file on each server, but that would be a pain to manage. This is the main reason why having a central DNS server is a benefit to any network, especially for resolving the names of local resources. However, the `/etc/hosts` file is used every now and again in the enterprise as a quick one-off workaround, and you'll probably eventually end up needing to use this method for one reason or another.

As I mentioned, if the server doesn't find a match in `/etc/hosts` for the resource you're trying to find, it will move on to check the DNS server it was assigned. Another file on your server will determine the DNS server it will then use, clarified in the `/etc/resolv.conf` file. An example of this file is as follows:

```
# Dynamic resolv.conf(5) file for glibc resolver(3) generated by
resolvconf(8)
#     DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN
nameserver 10.10.96.1
nameserver 10.10.96.2
```

In this example, `/etc/resolv.conf` output is utilizing servers `10.10.96.1`, and `10.10.96.2`. Therefore, the server will first check `/etc/hosts` for a match of the resource you're looking up, and if it doesn't find it, it will then check `/etc/resolv.conf` in order to find out which server to check next. In this case, the server will check `10.10.96.1`.

The network assignment on this server is actually being managed by Network Manager, as you can see from the warning comment included in the file. You can certainly alter this file so that the server will check different DNS servers, but the next time its IP address renews or the connection is refreshed, this file will be overwritten and will be reset to use whatever DNS servers your DHCP server tells it to use. Therefore, if you really want to control how your server resolves names you can create a static assignment, as we did earlier in this chapter, using the `dns-nameservers` option in `/etc/network/interfaces` to override it. This isn't a problem if you're using a static IP assignment.

www.it-ebooks.info

In a typical Enterprise Linux network, you'll set up a local DNS server to resolve your internal resources, which will then forward requests to a public DNS server in case you're attempting to reach something that's not internal. We'll get to that in *Chapter 7*, *Managing Your Ubuntu Server Network*, but you should now understand how the name resolution process works on your Ubuntu Server.

Understanding Network Manager

Network Manager is a fantastic utility for managing network connectivity on your server, though it's not for everyone. This utility comes installed by default with Ubuntu Server, but if you're using a cloud appliance (such as a VPS), it's most likely disabled, or maybe even removed. The reason for this is because even though this utility is very useful, it's not necessarily the best fit for every use-case.

Network Manager is a service (a.k.a. daemon) that runs in the background and manages your network connections. On a desktop Linux distribution, it does everything from managing connectivity to managing your wireless networks. With it, you can configure profiles, and create custom connections you can switch between.

On servers, the desktop-related features of Network Manager aren't all that useful, but the network connectivity management portion certainly is. With Network Manager, it keeps an eye on whether or not your server is connected to a network. It even goes as far as to launch your DHCP client when network connectivity is started or restored, to facilitate the fetching of a dynamic IP address. If your network connection drops for any reason, Network Manager will request a dynamic address as soon as it comes back online.

If you're not using DHCP, however, Network Manager is of little benefit. In fact, it can be somewhat of a nuisance. The best rule of thumb in my opinion is to use Network Manager when you're using DHCP for IP assignments, and to disable it when you're manually configuring your network address via `/etc/network/interfaces`.

Stopping and disabling Network Manager can be done with the following commands (which you should only do in the case that you've set up a manual static IP assignment):

```
# systemctl stop NetworkManager
# systemctl disable NetworkManager
# systemctl restart networking
```


www.it-ebooks.info

As I've mentioned before, my preferred approach is static leases via DHCP (in other words, DHCP reservations for servers). The added benefit of static leases is that you benefit from being able to utilize Network Manager. It will keep your connection alive, and any time it reaches out for an IP address, your server will always get the address you've set aside for it in your DHCP server. (Again, we'll go over how to set this up in *Chapter 7*, *Managing Your Ubuntu Server Network*). In a situation where you need to set up a static IP, you may as well just disable Network Manager because it may just get in your way.

Getting started with OpenSSH

OpenSSH is quite possibly the most useful tool in existence for managing Linux servers. Of all the countless utilities available, this is the one I recommend that everyone master as early as possible. Technically, I could probably better fit a section for setting up OpenSSH in *Chapter 7*, *Managing Your Ubuntu Server Network*, but this utility is very handy, and we should start using it as soon as possible. In this section, I'll give you some information on OpenSSH and how to install it, and then I'll finish up the section with a few examples of actually using it.

OpenSSH allows you to open a shell on other Linux servers, allowing you to run commands as if you were there in front of the server. In a Linux Administrator's workflow, they will constantly find themselves managing a plethora of machines in different locations. OpenSSH works by having a daemon running on the server that listens for connections. On your workstation, you'll use your SSH Client to connect to the server, to begin running commands on it. SSH isn't just useful for servers either; you can manage workstations with it, as well as network appliances. I've even used SSH on my laptop to connect to my desktop to issue a `reboot` command, just because I was too lazy to walk all the way to my bedroom. It's extremely useful. And thankfully, it's also very easy to learn.

Depending on the choices you've made during installation, your Ubuntu Server probably has the OpenSSH server installed already. If you don't remember, just type `which sshd` at your shell prompt. If you have it installed, your output should read `/usr/bin/sshd`. If you haven't installed the server, you can do so with the following command:

```
# apt-get install openssh-server
```


www.it-ebooks.info

Keep in mind, though, that the OpenSSH server is not required to connect to other machines. Regardless of whether or not you install the server, you will still have the OpenSSH client installed by default. If you type which `ssh` at your shell prompt (omitting the 'd' from `sshd`) you should see an output of `/usr/bin/ssh`. If, for some reason, you don't have this package installed and you received no output from this command (which would be rare), you can install the OpenSSH client with the following command:

```
# apt-get install openssh-client
```

I want to underline the fact that you're not required to install the `openssh-server` package in order to make connections to other machines. You only need the `openssh-client` package to do that. For the vast majority of Linux administrators, I cannot think of a good reason to not have the `openssh-client` package installed. It's useful when you want to remotely manage another Linux machine or server, but by itself, it doesn't allow other users to connect to you. It's important to understand that installing the OpenSSH Server does increase your attack surface, though.

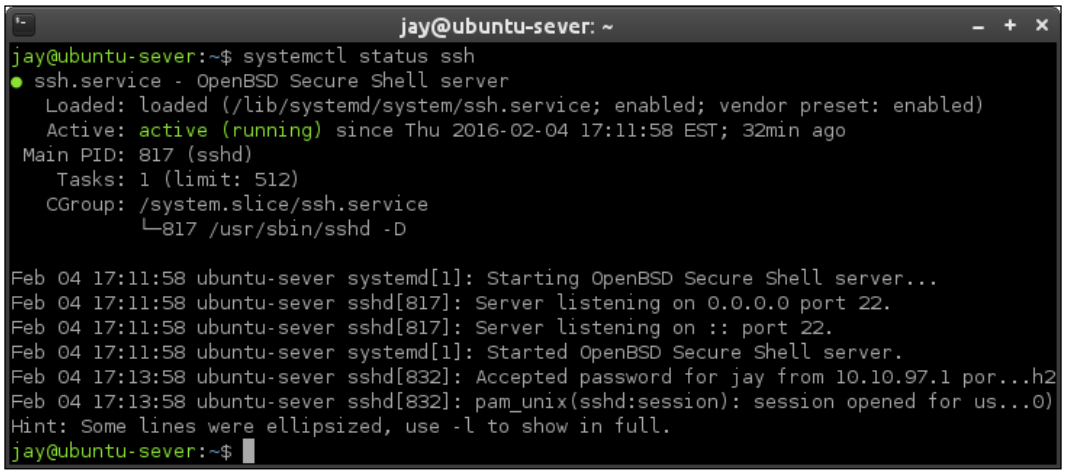
Whether or not to run an OpenSSH server on your machine usually comes down to a single question: Do you want to allow users to connect remotely to your server? Most of the time, the answer to that question is yes. The reason is that it's more convenient to manage all your Linux servers at your desk, without having to walk into your server room and plug in a display and keyboard every time you want to do something. But even though you most likely want to have an SSH server running, you should still keep in mind that OpenSSH would then be listening for and allowing connections. The fact that OpenSSH allows you to remotely manage other servers is its best convenience, and also its biggest weakness. If you are able to connect to a server via SSH, so can others.

There are several best practices for security that you'll want to implement if you have an OpenSSH server running. In *Chapter 12*, *Securing Your Server*, I will walk you through various configuration changes you can make to help minimize the threat of miscreants breaking into your server from the outside and wreaking havoc. Securing OpenSSH is actually not hard at all, and would probably only take just a few minutes of your time. Therefore, feel free to make a detour to *Chapter 12*, *Securing Your Server*, to read the section there that talks about securing OpenSSH, and then come back here when you're done. If you have a server that is directly accessible via the Internet, with users with weak passwords and you allowing connections via SSH, I can personally guarantee that it will be hijacked within two weeks. As a general rule of thumb though, you're usually fine as long as your user accounts have strong passwords, the OpenSSH package is kept up to date with the latest security updates, and you disable login via `root`.

www.it-ebooks.info

With all of that out of the way, we can get started with actually using OpenSSH. After you've installed the `openssh-server` package, you'll need to start it if it hasn't been already. By default, Ubuntu's `openssh-server` package is automatically configured to start and become enabled once installed. To verify, run the following command:

```
systemctl status ssh
```



Output from systemctl, showing a running SSH server

If OpenSSH is running as a daemon on your server, you should see output that tells you that it's active (running). If not, you can start it with the following command:

```
# systemctl start ssh
```

If the output of the `systemctl status ssh` command shows that the daemon is disabled (meaning it doesn't start up automatically when the server boots), you can enable it with the following command:

```
# systemctl enable ssh
```

On older Ubuntu servers (for example, 14.04 and 12.04), you can use the following two commands in order to start and enable the OpenSSH server respectively:

```
# service ssh starting
# update-rc.d ssh defaults
```

Don't worry about the `systemctl` or `service` commands just yet, we'll go over them in greater detail in *Chapter 6, Controlling and Monitoring Processes*.

www.it-ebooks.info

With the OpenSSH server started and running, your server should now be listening for connections. To verify, use the following command to list listening ports, restricting the output to SSH:

```
# netstat -tulpn |grep ssh
```

Output from netstat showing that SSH is listening

If, for some reason, your server doesn't show that it has an SSH server listening, double-check that you've started the daemon. By default, the SSH server listens for connections on port `22`. This can be changed by modifying the port declaration in the `/etc/ssh/sshd_config` file, but that's a story for a later chapter. While I won't be going over the editing of this file just yet, keep in mind that this file is the main configuration file for the daemon. OpenSSH reads this file for configuration values each time it's started or restarted.

To connect to a server using SSH, simply execute the `ssh` command followed by the name or IP address of the server you'd like to connect to:

```
ssh 10.10.96.10
```

By default, the `ssh` command will use the username you're currently logged in with for the connection. If you'd like to use a different username, specify it with the `ssh` command by including your username followed by the `@` symbol just before the IP address or hostname:

```
ssh jdoe@10.10.96.10
```

Unless you tell it otherwise, the `ssh` command assumes that your target is listening on port `22`. If it isn't, you can give the command a different port with the `-p` option followed by a port number:

```
ssh -p 2242 jdoe@10.10.96.10
```


www.it-ebooks.info

Once you're connected to the target machine, you'll be able to run shell commands and administer the system as if you were right in front of it. You'll have all the same permissions as the user you logged in with, and you'll also be able to use `sudo` to run administrative commands if you normally have access to do so on that server. Basically, anything you're able to do if you were standing right in front of the server, you'll be able to do with SSH. When you're finished with your session, simply type `exit` at the shell prompt, or press *Ctrl* + *D* on your keyboard.

If you started background commands on the target via SSH, use *Ctrl* + *D* to end your session, otherwise those processes will be terminated. We'll talk about background processes in *Chapter 6*, *Controlling and Monitoring Processes*.

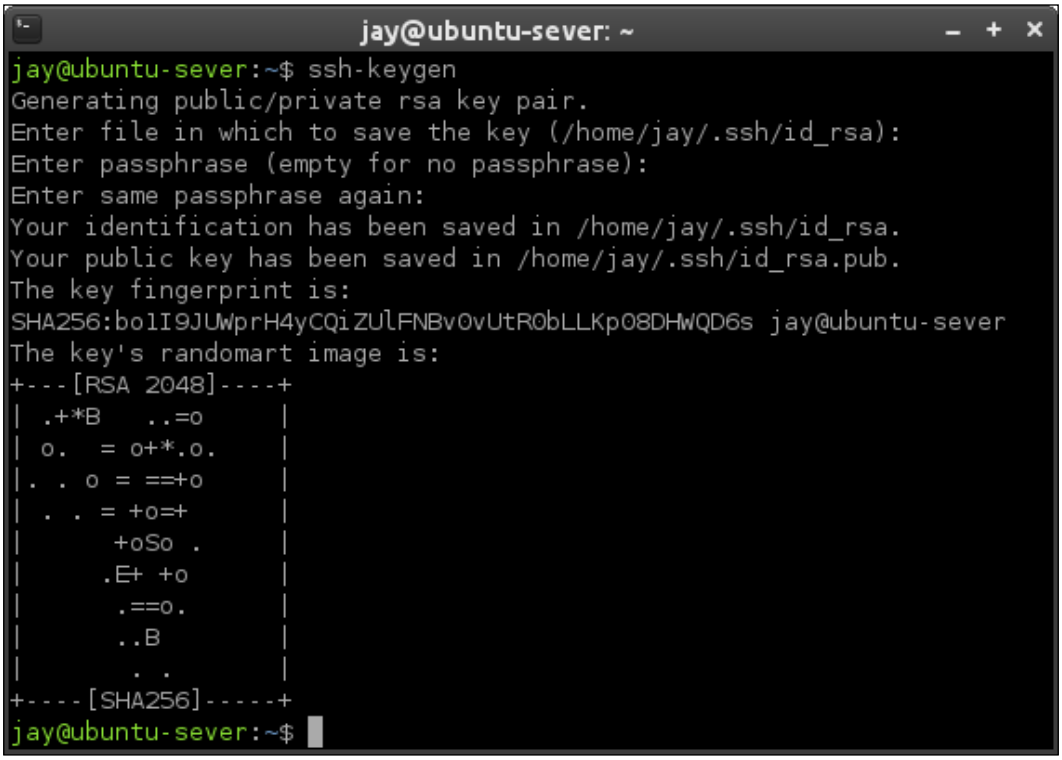
As you can see, OpenSSH is a miraculous tool that will benefit you by allowing you to remotely manage your servers from anywhere you allow SSH access from. Make sure to read the relevant section in *Chapter 7*, *Managing Your Ubuntu Server Network*, with regards to securing it, though. In the next section, we'll discuss SSH key management which also benefits convenience, but also allows you to increase security as well.

Getting started with SSH key management

When you connect to a host via SSH, you'll be asked for your password, and after you authenticate you'll be connected. Instead of using your password though, you can authenticate via **Public Key Authentication** instead. The benefit to this is added security, as your system password is never transmitted during the process of connecting to the server. When you create an SSH key-pair, you are generating two files, a `Public Key` and a `Private Key`. These two files are mathematically linked, so if you connect to a server that has your public key, it will know it's you because you (and only you), have the private key that matches it. While public key cryptography as a whole is beyond the scope of this book, this method is far more secure than password authentication, and I highly recommend that you use it. To get the most out of the security benefit of authentication via keys, you can actually disable password-based authentication on your server so that your SSH key is your only way in. By disabling password-based authentication and using only keys, you're increasing your server's security by an even larger margin. We'll go over that later in the book.

www.it-ebooks.info

To get started, you'll first need to generate your key. To do so, use the `ssh-keygen` command as your normal user account. The following screenshot shows what this process generally looks like:



Generating an SSH key pair

First, you'll be asked for the directory in which to save your key files, defaulting to `/home/<user>/.ssh`. You'll next be asked for a passphrase, which is optional. Although it does add an additional step to authenticating via keys, I recommend that you give it a passphrase (which should be different than your system password). You can press *Enter* for the passphrase without entering one if you do not want this.

What this command does is create a directory named `.ssh` in your `home` directory, if it doesn't already exist. Inside that directory, it will create two files, `id_rsa` and `id_rsa.pub`. The `id_rsa` file is your private key. It should never leave your machine, be given to another user, or be stored on any external media. If your private key leaks out, your keys can no longer be trusted. By default, the private key is owned by the user that created it, with `rw` permissions given only to its owner.

www.it-ebooks.info

The public key, on the other hand, can leave your computer and doesn't need to be secured as much. Its permissions are more lenient, being readable by everyone and writable by the owner. You can see this yourself by executing `ls -l /home/<user>/.ssh`. The public key is the key that gets copied to other servers to facilitate you being able to log in via the key pair. When you log in to a server that has your key, it checks that it's a mathematical match to your private key, and then lets you log in. You'll also be asked for your passphrase, if you chose to set one.

To actually transmit your public key to a target server, we use the `ssh-copy-id` command. In the following example, I'll show a variation of the command that's copying the key to a server named `area51`:

```
ssh-copy-id -i ~/.ssh/id_rsa.pub area51
```

With that command, replace `area51` with the hostname of the target server, or its IP address if you don't have a DNS record created for it. You'll be asked to log in via password first, and then your key will be copied over. From that point on, you'll log in via your key, falling back to being asked for your password if, for some reason, your key relationship is broken.

So what exactly did the `ssh-copy-id` command do? Where is your public key copied to, exactly? What happens with this command is that on the target server, an `.ssh` directory is created in your `home` directory if it didn't already exist. Inside that directory, a file named `authorized_keys` is created if it didn't exist. The contents of `~/.ssh/id_rsa.pub` on your machine are copied into the `~/.ssh/authorized_keys` file on the target server. With each additional key you add (for example, you connect to that server from multiple machines), the key is added to the end of the `authorized_keys` file, one per line.

Using the `ssh-copy-id` command is merely a matter of convenience, there's nothing stopping you from copying the contents of your `id_rsa.pub` file and manually pasting it into the `authorized_keys` file of the target server. That method will actually work just fine as well.

When you connect to a server via SSH that you have set up a key relationship with, SSH checks the contents of the `~/.ssh/authorized_keys` file on that server, looking for a key that matches the private key (`~/.ssh/id_rsa`) on your machine. If the two keys are a mathematical match, you are allowed access. If you set up a passphrase, you'll be asked to enter it in order to open your public key.

www.it-ebooks.info

If you decided not to create a passphrase with your key, you're essentially setting up password-less authentication, meaning you won't be asked to enter anything when authenticating. Having a key relationship with your server is certainly more secure than authenticating to SSH via a password, but it's even more secure with a passphrase and creating one is definitely recommended. Thankfully, using a passphrase doesn't have to be inconvenient. With an **SSH Agent**, you can actually cache your passphrase the first time you use it, so you won't be asked for it with every connection.

To benefit from this, your SSH Agent must be running. To start it, enter the following command as your normal user account on the machine you're starting your connections from (that is, your workstation):

```
eval $(ssh-agent)
```

This command will start an SSH agent, that will continue to run in the background of your shell. Then, you can unlock your key for your agent to use:

```
ssh-add ~/.ssh/id_rsa
```

At this point, you'll be asked for your passphrase. As long as you enter it properly, your key will remain open and you won't need to enter it again for future connections, until you close that shell or log out. This way, you can benefit from the added security of having a passphrase, without the inconvenience of entering your passphrase over and over.

There will at some point, be a situation where you'll want to change your SSH passphrase. To do that, you can execute the following command:

```
ssh-keygen -p
```

First, press `Enter` to accept the default file (`id_rsa`). Then, you'll be asked for your current passphrase, if you've created one, followed by your new passphrase twice.

These concepts may take a bit of practice if you've never used SSH before. The best way to practice is to set up multiple Ubuntu Server installations (perhaps several virtual machines), and practice using SSH to connect to them, as well as deploying your key to each machine via the `ssh-copy-id` command. It's actually quite easy once you get the hang of it.

www.it-ebooks.info

Simplifying SSH connections with a ~/.ssh/config file

Before we leave the topic of SSH, there's another trick that benefits convenience, and that is the creation of a `~/.ssh/config` file. This file doesn't exist by default, but if it's found, SSH will parse it and you'll be able to benefit from it.

The `~/.ssh/config` file allows you to list servers that you connect to often, which can simplify the `SSH` command automatically. The following are example contents from a hypothetical `~/.ssh/config` file that will help me illustrate what it does:

```
host myserver
    Hostname 192.168.1.23
    Port 22
    User jdoe

Host nagios
    Hostname nagios.local.lan
    Port 2222
    User nagiosuser
```

In the example contents, I have two hosts outlined, `myserver` and `nagios`. For each, I've identified a way to reach it by name or IP address (the `Hostname` line), as well as the `Port` and `User` account to use for the connection. If I use `ssh` to connect to either `Host` by the name I outlined in this file, it will use the values I have stored there, for example:

```
ssh nagios
```

Will operate the same as if I identified the connection details manually, which would've been the following:

```
ssh -p 2222 nagiosuser@nagios.local.lan
```

I'm sure you can see how much simpler it is to type the first command than the second. With the `~/.ssh/config` file, I can have some of those details automatically applied. Since I've outlined that my `nagios` server is located at `nagios.local.lan`, its SSH user is `nagiosuser`, and it's listening on port `2222`, it automatically will use those values even though I only typed `ssh nagios`. Furthermore, you can also override this entry as well. If you provide a different username when you use the `ssh` command, it will use that instead of what you have written in the `~/.ssh/config` file.

www.it-ebooks.info

In the first example (for server `myserver`), I'm providing an IP address for the connection, rather than a hostname. This is useful in a situation where you may not have a DNS entry for your target server. With this example, I don't have to remember that the IP address for `myserver` is `192.168.1.23`. I simply execute `ssh myserver` and it's taken care of for me.

The names of each server within the `~/.ssh/config` file are arbitrary, and don't have to match the target server's hostname. I could've named the first server `potato` and it would still route me to `192.168.1.23`, so I can create any sort of named shortcut I want, whatever I find is most convenient for me and easiest to remember. As you can see, maintaining a `~/.ssh/config` file for your most commonly used SSH connections will certainly help keep you organized and allow you to connect more easily.

Summary

In this chapter, we worked through several examples of connecting to other networks. We started off by configuring our hostname, managing network interfaces, assigning static IP addresses, as well as a look at how name resolution works in Linux. A decent portion of this chapter was dedicated to topics regarding OpenSSH, which is an extremely useful utility that allows you to remotely manage your servers. While we'll revisit OpenSSH in *Chapter 12*, *Securing Your Server*, with a look at boosting its security, overall we've only begun to scratch the surface of this tool. Entire books have been written about SSH, but the examples in this chapter should be enough to make you productive with it. The name of the game is to practice, practice, practice!

In the next chapter, we'll talk about managing software packages. We'll work through adding and removing packages, updating your system, adding additional repositories, and more!

www.it-ebooks.info

5
Managing Software Packages

Prior to this chapter, I've already walked you through several situations that required you to install a new package or two on your server, but we have yet to have any formal discussion on package management. In this chapter, we'll explore package management in more detail. The Ubuntu platform has a huge range of software available for it, featuring packages for everything from server administration to games. In fact, as of the time I write this chapter, there are over 50,000 packages in Ubuntu's repositories. That's a lot of software—and, in this chapter, we'll take a look at how to manage these packages. We'll cover how to install, remove, and update packages, as well as the use of related tools. As we go through these concepts, we will cover:

- Understanding Linux package management
- Installing and removing software
- Searching for packages
- Managing Apt repositories
- Keeping your server up to date
- Backing up and restoring packages
- Making use of Aptitude
- Installing Snap packages

www.it-ebooks.info

Understanding Linux package management

Nowadays, app stores are all the rage on most platforms; you'll have one central location from which to retrieve applications, allowing you to install them on your device. Computers, as well as phones and tablets, utilize a central software repository in which software is curated and made available. The Android platform has the Google Play store, Apple offers its App Store, and so on. For us Linux folk, this concept isn't new. The concept of software repositories (that young people refer to nowadays as app stores), have been around within the Linux community since long before cellular phones even had color screens.

Linux has had package management since the 90s, popularized by Debian and then Red Hat. Software repositories are generally made available in the form of mirrors, to which your server subscribes. Mirrors are available across a multitude of geographic areas, so, typically, your installation of Ubuntu Server would subscribe to the mirror closest to you. These mirrors are populated with software packages that you'll be able to install. Many packages depend on other packages, so various tools on the Linux platform exist to automatically handle these dependencies for you. Not all distributions of Linux feature package management and dependency resolution, but Ubuntu certainly does, benefiting from the groundwork already built by Debian.

Packages contained within these mirrors are constantly changing. Traditionally, an individual known as a **package maintainer** is responsible for one or more packages, and ships new versions to the repositories for approval and, eventually, distribution to mirrors. Sometimes, the new version of the package is simply a security update. With the majority of Ubuntu's packages being open source, anyone is able to look at the source code, find problems, and report issues. When vulnerabilities are found, the maintainer will then review the claim and then release an updated version to correct it. This process happens very quickly, as I've seen severe vulnerabilities patched even on the same day they were reported.

Also, new versions of packages are sometimes a feature update, which is an update released to introduce new features and isn't necessarily tied to a security vulnerability. This could be a new version of a desktop application such as Firefox or a server package such as MySQL. Most of the time, though, new versions of packages that are vastly different are held for the next Ubuntu release. The reason for this is that too much change can introduce instability. Instead, known working and stable packages are preferred, but given the fact that Ubuntu releases every six months, you don't have to wait very long.

www.it-ebooks.info

As I've mentioned earlier in this book, most administrators stick with the **Long-Term Support** (**LTS**) releases, of which Ubuntu 16.04 is one. Every fourth Ubuntu release becomes an LTS, so a new one is released every other year. With regards to package updates, this means that you can expect to remain current with the latest security updates for a period of five years. On normal (non-LTS) releases, you will be able to keep current for only nine months. This is also why LTS releases are favored on servers—most organizations prefer to implement a server and keep it working as is for as long as possible. Nine months is simply faster than most organizations are willing to move.

However, there is a major issue with using only LTS releases, and that comes in the form of hardware support. A large majority of hardware drivers are built right into the kernel. This means that if you have an older release of Ubuntu, you'll likely run into issues if you try to install it on a just-released piece of hardware, such as a brand new PC or server. In this situation, you may find that things such as networking or other hardware won't function properly. Thankfully, there is a third type of update that package maintainers will often release, and that is hardware enablement updates. Hardware enablement updates will either come in the form of a new minor version of a kernel (with drivers back-ported) or an all-new kernel altogether.

How one installs hardware enablement updates varies depending on how the update was released, but generally they are optional. Typically, the kernel that was installed when you first set up your server will be the version your server will use, but new installation media will be created that will install updated kernels for servers that have not been set up before. This is why it's a good idea to make sure you're always using the latest installation media when installing Ubuntu. Ubuntu 14.04, for example, originally shipped with kernel 3.13. Later, the kernel installed by the installation media was updated to 3.16 when Ubuntu 14.04.2 was released, then to 3.19 with 14.04.3, and then to kernel 4.2 with Ubuntu Server 14.04.4. Those who installed their server with Ubuntu 14.04.1 or lower would still be running kernel 3.13 unless the administrator had manually updated the hardware enablement stack.

Generally, there's no reason to upgrade to a newer hardware enablement stack, unless you need to benefit from the newer drivers or your current kernel is going to be discontinued. You should subscribe to Ubuntu's mailing lists in order to keep abreast of updates regarding kernel support for your release.

www.it-ebooks.info

Package management is typically very convenient in Ubuntu, with security updates and feature updates coming regularly. With just one command (which we'll get to shortly), you can install a package along with all of its dependencies. Having done manual dependency resolution myself, I can tell you first hand that having dependencies handled automatically is a very wonderful thing. The main benefit of how packages are maintained on a Linux server is that you generally don't have to search the Internet for packages to download, as Ubuntu's repositories contain most of the ones you'll ever need. As we continue through this chapter, you'll come to know everything you need in order to manage this software.

Installing and removing software

Throughout this book, I've had you use the `apt-get` command multiple times to download and install various packages, so, technically, you've already worked through at least some package management. But we've never had a formal discussion about this command, so in this section we'll go a little more in depth with it.

As you've seen before, we can use `apt-get` along with the `install` keyword and then a package name in order to install a new package on your system. For example, the following command will install the `openssh-server` package:

```
# apt-get install openssh-server
```

You can also install multiple packages at a time by separating each with a space, instead of installing each package one at a time. The following example will install three different packages:

```
# apt-get install <package1> <package2> <package3>
```

Removing packages follows a very similar syntax; you would only need to replace the keyword `install` with `remove`. For example, we can remove the `nano` package with the following command:

```
# apt-get remove nano
```

And, just like with the `install` keyword, you can remove multiple packages at the same time as well. The following example will remove three packages:

```
# apt-get remove <package1> <package2> <package3>
```

If you'd like to not only remove a package but also wipe out its configuration, you can use the `--purge` option:

```
# apt-get remove --purge <package>
```


www.it-ebooks.info

We can actually simplify these commands a bit. Some people may not know this, but the `apt-get` command can actually be shortened to just apt in recent releases of Ubuntu Server as well as in Debian. Feel free to try the following variations of the install and remove commands, respectively:

```
# apt install <package>
# apt remove <package>
```

As we go along, I'll continue to use `apt-get` instead of shortening the command to just `apt`, mainly to maintain compatibility with older versions of Ubuntu Server as much as possible. But, if you're using a newer version of Ubuntu Server, feel free to use the shortened commands in place of the ones I will provide.

So, what happens when you install a package? If you've run through the process before, you're probably accustomed to this process already. But, typically, the process begins with `apt` calculating dependencies. The majority of packages require other packages in which to function, so `apt` will check to ensure that the package you're requesting is available, and that its dependencies are available as well. First, you'll see a summary of the changes that apt wants to make to your server. In the case of installing the `apache2` package on an unconfigured Ubuntu Server, I see the following output on my system:

```
# apt-get install apache2
The following additional packages will be installed:
  apache2-bin apache2-data apache2-utils libapr1 libaprutil1 libaprutil1-
dbd-sqlite3
  libaprutil1-ldap liblua5.1-0 ssl-cert
Suggested packages:
  apache2-doc apache2-suexec-pristine | apache2-suexec-custom openssl-
blacklist
The following NEW packages will be installed:
  apache2 apache2-bin apache2-data apache2-utils libapr1 libaprutil1
  libaprutil1-dbd-sqlite3 libaprutil1-ldap liblua5.1-0 ssl-cert
0 upgraded, 10 newly installed, 0 to remove and 69 not upgraded.
Need to get 1,549 kB of archives.
After this operation, 6,470 kB of additional disk space will be used.
Do you want to continue? [Y/n]
```


www.it-ebooks.info

Even though I only asked for `apache2`, `apt-get` informs me that it also needs to install `apache2-bin`, `apache2-data`, `apache2-utils`, `libapr1`, `libaprutil1`, `libaprutil1-dbd-sqlite3`, `libaprutil1-ldap`, and `liblua5.1-0 ssl-cert` in order to satisfy the dependencies for the `apache2` package. Apt also suggests that I install `apache2-doc`, `apache2-suexec-pristine | apache2-suexec-custom`, and `openssl-blacklist`, though those are optional and are not required. If I wanted to install `apache2`, its dependencies, as well as the suggested packages, I could've run the following command instead:

```
# apt-get install --install-suggests apache2
```

Most of the time, though, you probably won't want to do this; it's usually better to keep your installed packages to a lean minimum and install only the packages you need. As we'll discuss in *Chapter 12*, *Securing Your Server*, the fewer packages you install, the smaller the attack surface of your server.

Another option that is common with installing packages via `apt` is the `-y` option, which assumes yes to the confirmation prompt where you choose if you want to continue or not. For example, my previous example output included the line `Do you want to continue? [Y/n]`. If we used `-y`, the command would've proceeded to install the package without any confirmation. This can be useful for administrators in a hurry, though I personally don't see the need for this unless you are scripting your package installations.

Continuing on with the example installation of `apache2`, my output looks like the following after I press *Y* and then *Enter* to accept the changes. Note that I've removed some redundant lines to save space on this page:

```
Setting up apache2 (2.4.18-1ubuntu1) ...
Enabling module mpm_event.
Enabling module authz_core.
Enabling module authz_host.
Enabling module authn_core.
Enabling module auth_basic.
Setting up ssl-cert (1.0.37) ...
Processing triggers for libc-bin (2.21-0ubuntu5) ...
Processing triggers for systemd (228-5ubuntu2) ...
Processing triggers for ureadahead (0.100.0-19) ...
Processing triggers for ufw (0.34-2) ...
```


www.it-ebooks.info

There were several other packages being installed and modules being enabled when I ran the command, but you should get the basic idea from the example output. As you can see, `apt` has done more than simply install the `apache2` package. It has also enabled some of the built-in modules for Apache, as well as setting up `systemd` to automatically start the Apache daemon. We see this in the lines that begin with processing triggers for several of the packages, which are additional steps added to the package to be completed post-install. Specifically, the "triggers" that are executed are done in order to receive processing by another package, in particular, the `systemd` daemon, which is responsible for starting and enabling daemons in newer Ubuntu Server releases.

It's rather neat that Ubuntu Server automatically configures most packages to have their daemons start up and also be enabled so that they start with each boot. This may seem like a no-brainer, but not everyone prefers this automation. As I've mentioned, the more packages installed on your server, the higher the attack surface, but running daemons are each a method of entry for miscreants should there be a security vulnerability. Therefore, some distributions don't enable and start daemons automatically when you install packages. The way I see it, though, you should only install packages you actually intend to use, so it stands to reason that if you go to the trouble of manually installing a package such as Apache, you probably want to start using it.

When you install a package with the `apt` keyword, it searches its local database for the package you named. If it doesn't find it, it will throw up an error. Sometimes, this error may be because the package isn't available or perhaps the version that `apt` wants to install no longer exists. Ubuntu's repositories move very fast. New versions of packages are added almost daily. When a new version of a package is added, its older equivalent may be removed. For this reason, it's recommended that you update your package sources from time to time. Doing so is easy, using the following example:

```
# apt-get update
```

This command doesn't actually update any packages, it merely checks in with your local mirror to see if any packages have been added or removed, and updates your local index. This command is useful because installations can fail if your sources aren't up to date. In most cases, the symptom will either be the package isn't found or apt bailing out of the process when it encounters a package it's looking for, but cannot find.

Installing software in Ubuntu Server is actually fairly easy, and some may argue it is easier than in other platforms. Having a centralized location to house its software certainly helps, and commands such as `apt-get` allow you to manage your software very effectively.

www.it-ebooks.info

Searching for packages

The naming conventions used for packages in Ubuntu Server aren't always obvious. Worse, package names are often very different from one distribution to another for even the same piece of software. While this book and other tutorials online will outline the exact steps needed to install software, if you're ever on your own, it really doesn't help much if you don't know the name of the package you want to install. For example, in the previous section I showed the sample output from installing Apache, which I did by installing the `apache2` package via `apt-get`. If you didn't know any better, you would probably search for the package by just `apache`. If you are coming from another platform (such as CentOS), you may instinctively have searched for `httpd`, which is a common name for Apache on that platform. In another example, you may have wanted to install an SSH server, but you probably wouldn't have guessed on your own that the package you want to install is not `ssh`, but rather is `openssh-server` when it comes to Ubuntu Server. If we don't give `apt` the exact name of the package we want, the installation will fail. You'll get an error similar to the following:

```
E: Unable to locate package httpd
```

The `apt-get` utility does no guesswork. It assumes you know what you're doing and you've already done the research in terms of what you want to install. Thankfully, it also has the capability of searching your package index as well. We can use the `apt-cache` utility in order to search for packages:

```
apt-cache search <search term>
```

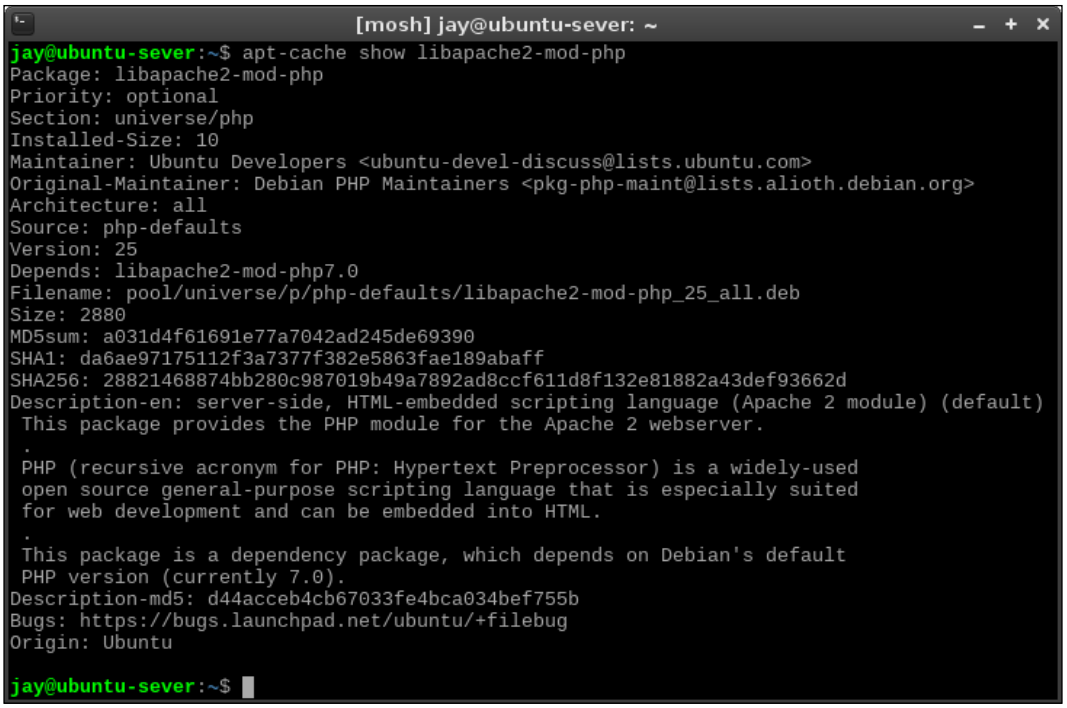
Output from this command will show a list of packages that match your search criteria, with their names and descriptions. If, for example, you wanted to install the PHP plugin for Apache and you didn't already know the name of the associated package, the following would narrow it down:

```
apt-cache search apache php
```

In the output, we will get a list of more than a handful of packages, but we can deduce from the package descriptions in the output that `libapache2-mod-php` is most likely the one we want. We can then proceed to install it using `apt-get`, as we would normally do. If we're not sure whether or not this is truly the package we want, we can view more information with the `apt-cache show` command:

```
apt-cache show libapache2-mod-php
```


www.it-ebooks.info



Showing details of the libapache2-mod-php package with apt-cache show

With this command, we can see additional details regarding the package we're considering installing. In this case, we learn that the `libapache2-mod-php` package also depends on PHP itself, so that means if we install this package, we'll get the PHP plugin as well as PHP.

Another method of searching for a package (if you have a web browser available), is to connect to the Ubuntu Packages Search page at `http://packages.ubuntu.com/` where you can navigate through the packages from their database for any currently supported version of Ubuntu. You won't always have access to a web browser while working on your servers, but, when you do, this is a very useful way to search through packages, view their dependencies, descriptions, and more.

www.it-ebooks.info

Managing software repositories

Often, the repositories that come pre-installed with Ubuntu will suffice for the majority of your server needs. Every now and then, though, you may need to install an additional repository in order to take advantage of software not normally provided by Ubuntu, or versions of packages newer than what you would normally have available. Adding additional repositories allows you to subscribe to additional sources of software and install packages from them the same as you would from any other sources.

Adding additional repositories should be considered a last resort, however. When you install an additional repository, you're effectively trusting the author of that repository with your company's server. Although I haven't ever seen this happen first hand, it's theoretically possible for authors of software to include backdoors or malware in software packages, and then make them available for others via a software repository. Therefore, you should only add repositories from sources that you have reason to trust.

In addition, it sometimes happens that a maintainer of a repository simply gives up on it and disappears. This I have seen happen first hand. In this situation, the repository may go offline (which would show errors during `apt` transactions that it's not able to connect to the repository), or worse, the repository stays online, but security updates are never made available, causing your server to become wide open for attack. Sometimes, you just don't have a way around it. You need a specific application and Ubuntu doesn't offer it by default. Your only option may be to compile an application from source or add a repository. The decision is yours, but just keep security in mind whenever possible.

Software repositories are essentially URLs in a text file, stored in one of two places. The main Ubuntu repository list is stored in `/etc/apt/sources.list`. Inside that file, you'll find a multitude of repositories for Ubuntu's package manager to pull packages from. In addition, files with an extension of `.list` are read from the `/etc/apt/sources.list.d/` directory and are also used whenever you use `apt`. I'll demonstrate both methods.

A typical repository line in either of these two files will look similar to the following:

```
deb http://us.archive.ubuntu.com/ubuntu/ xenial main restricted
```

The first section of each line will be either `deb` or `deb-src`, which references whether the `apt` command will find binary packages (`deb`) or source packages (`deb-src`), there. Next, we have the actual URL which `apt` will use in order to reach the repository. In the third section, we have the codename of the release; in this case, `xenial` (which refers to the codename for Ubuntu 16.04, "`Xenial Xerus`").

www.it-ebooks.info

Continuing, the fourth section of each repository line refers to the `Component`, which references whether or not the repository contains software that is free and open source, and is supported officially by Canonical (the company that oversees Ubuntu's development). The component can be one of `main`, `restricted`, `universe`, or `multiverse`. Repositories with a `main` component include officially supported software. This generally means that the software packages have source code available, so Ubuntu developers are able to fix bugs. Software marked restricted is still supported, but may have a questionable license. Universe packages are supported by the community, not Canonical themselves. Finally, multiverse packages contain software that is neither free nor supported, that you would be using at your own risk.

As you can see from looking at the `/etc/apt/sources.list` file on your server, it's possible for a repository line to feature software from more than one component. Each repository URL may include packages from several components, and the way you differentiate them is to only subscribe to the components you need for that repository. In our previous example, the repository line included both `main` and `restricted` components. This means that, for that particular example, the `apt` utility will index both free (`main`) and non-free (`restricted`) packages from that repository.

You can add new repositories to the `/etc/apt/sources.list` file (and it will function just fine), but that's not typically the preferred method. Instead, as I mentioned earlier, `apt` will scan the `/etc/apt/sources.list.d/` directory for text files ending with the `.list` extension. These text files are the same as the `/etc/apt/sources.list` file in the sense that you include one additional repository per line, but this method allows you to add a new repository by simply creating a file for it, and you can remove the repository by simply deleting that file. This is safer than editing the `/etc/apt/sources.list` file directly, since there's always a chance you can make a typo and disrupt your ability to download packages from the official repositories.

In *Chapter 10*, *Serving Web Content*, I'll show you how to set up ownCloud. I bring this up now because installing ownCloud is a great example of how to add a third-party repository to our server. You don't need to actually perform these steps unless you actually want to install ownCloud, which is a task better left for *Chapter 10*, *Serving Web Content*, (there's more steps to setting up ownCloud than simply installing its package). Instead, I'll just show the process so that you will understand how it works. This is generally the same process you would follow for other sources of software.

First, we need to know which repository line to use. In the case of ownCloud, the repository line listed on the download section of their site is the following:

```
deb http://download.owncloud.org/download/repositories/stable/
xUbuntu_16.04/ /
```


www.it-ebooks.info

To install the repository, we can create the following file with our text editor and place that repository line inside it:

`/etc/apt/sources.list.d/owncloud.list`

Having the repository file included in the `/etc/apt/sources.list.d/` directory isn't enough by itself, we also need to install the **GNU Privacy Guard** (**GnuPG**) key for the repository on our server. This key helps our system verify that the packages have been signed by the appropriate party. Without it, we would see errors when we try to install packages from this repository or when we update our system. To install a key for a repository, you should follow the instructions listed on the site for the repository you're adding. On the ownCloud site, they mention the following commands as the ones to use to add the key:

```
wget -nv https://download.owncloud.org/download/repositories/stable/
xUbuntu_16.04/Release.key -O Release.key
```

```
apt-key add - < Release.key
```

Next, we need to refresh our `apt` sources in order to take advantage of this new repository. The following command will take care of that:

```
# apt-get update
```

Providing the repository is online and functioning properly, we should now have the software from that repository available to us to install via `apt` as we would for any other `deb` packages.

Generally speaking, the instructions I have provided are just an example of the typical process. Every software vendor that has a repository available for their packages may have you perform some variation of this process. Always follow the instructions from the vendor when installing a new repository. Also, it's wise to make sure that you also research the vendor, since adding a third-party repository requires a certain amount of trust between you and the vendor (especially if the source packages aren't available for you or your team to audit).

On the Ubuntu platform, there also exists another type of repository, known as a **Personal Package Archive** (**PPA**). PPAs are essentially another form of apt repository, and you'll even interact with their packages with the `apt` command, as you would normally. PPAs are usually very small repositories, often including a single application that serves a single purpose. A PPA is common in situations where a vendor doesn't make their software available with their own repository, and may only make their application available in the form of source code you would need to manually download, compile, and install. With the PPA platform, anyone can compile a package from source and easily make it available for others to download.

www.it-ebooks.info

PPAs suffer from the same security concerns as repositories (you need to trust the vendor and so on), but are a bit worse considering that the software isn't audited at all. In addition, if the PPA was to ever go down, you'd stop getting security updates for the application you install from it. Only use PPAs when you absolutely need to.

There is one use case for PPAs that may be compelling, specifically, for a server platform that standard repositories aren't able to handle as well, and that is software versioning. Typically, a major server component such as PHP or MySQL may be locked to a specific major version with each Ubuntu Server release. What do you do if you need to use Ubuntu Server, but the application you need to run is not available at the version your organization requires? In the past, you would literally need to choose between the distribution or the package, with some organizations even using a different distribution of Linux just to satisfy the need to have a specific application at a specific version. You can always compile the application from source (assuming its source code is available), but that can cause additional headaches in the sense that you'd be responsible for compiling new security patches yourself whenever they're made available. Often, very specific PPAs will be created to house software packages at a specific major version.

PPAs are generally added to your server with the `apt-add-repository` command. The syntax is generally using the `apt-add-repository` command, with a colon, followed by a user name, and then the PPA name. The following command is a hypothetical example:

```
# apt-add-repository ppa:username/myawesomesoftware-1.0
```

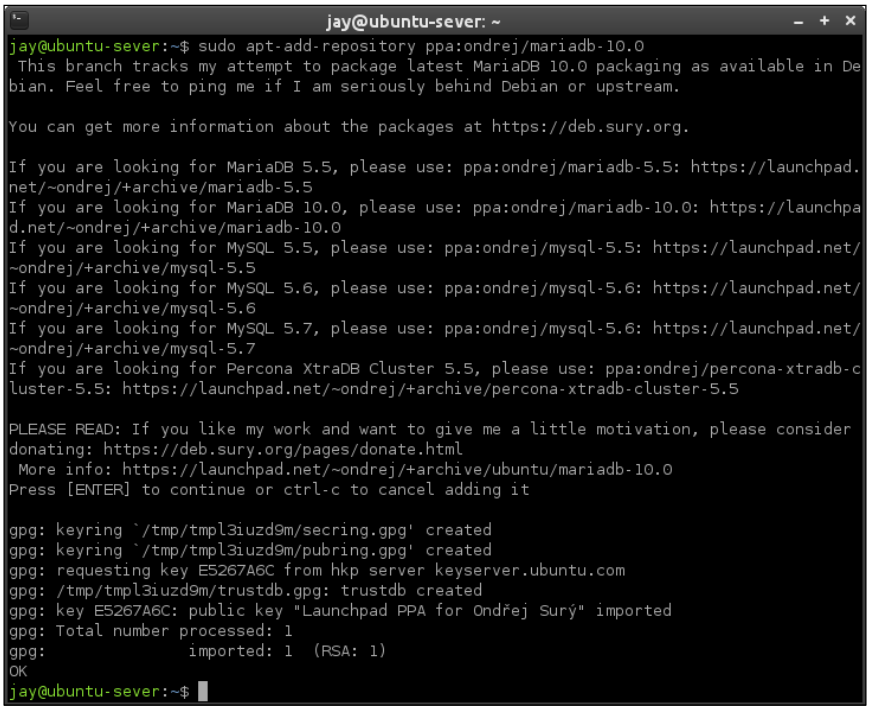
To begin the process, you would start your search by visiting Ubuntu's PPA website, which is available at the following URL:

`https://launchpad.net/ubuntu/+ppas`

Once you find a PPA you would like to add to your server, you can add it simply by finding the name of the PPA and then adding it to your server with the `apt-add-repository` command. You should take a look at the page for the PPA, though, in case there are different instructions. For the most part, the `apt-add-repository` command should work fine for you.

www.it-ebooks.info

In the following screenshot, you'll see an example run of this process, where I add a PPA for MariaDB to one of my servers:



An example of the process of adding a PPA to a server

With this PPA, I'm able to install MariaDB 10, which is useful because not all releases of Ubuntu Server offer this specific version. I started the process by searching for a MariaDB PPA on the PPA web page, where I found this specific PPA by a user known as `ondrej`. On the page for this PPA, I saw its name listed as `ppa:ondrej/mariadb-10.0`, so I combined that name with the `apt-add-repository` command (I was logged in as a normal user, so I needed to use `sudo`). When put together, the command ended up being the following:

```
# apt-add-repository ppa:ondrej/mariadb-10.0
```

Next, the user uploaded some very helpful text regarding the purpose of this PPA, as well as some links to other PPAs by that user that serve a similar purpose (specific versions of MariaDB that have been made available for various versions of Ubuntu Server). This text was displayed in the output of my terminal window, followed by a note that I can press *Enter* to add the PPA, or press *Ctrl* + *C* to cancel the process. I pressed *Enter*, then the repository (as well as its GnuPG key), was added to the server. From this point on, I can add software to my server from this repository by simply using the `apt-get` command as I normally would.

www.it-ebooks.info

So what exactly did this command do? We know that it installed the PPA to my server, but where on the filesystem did this get configured? Even though we added this repository with a special command, all it really did was create a text file in `/etc/apt/sources.list.d`, the same directory in which normal repositories can be placed that I mentioned earlier. The line the PPA added to my server is the same as for any other package repository:

```
deb http://ppa.launchpad.net/ondrej/mariadb-10.0/ubuntu xenial main
```

Therefore, to delete this repository, all I would need to do is remove the file it created:

```
# rm /etc/apt/sources.list.d/ondrej-ubuntu-mariadb-10_0-xenial.list
```

PPAs are one of the things that sets Ubuntu apart from Debian, and can be a very useful feature if harnessed with care. PPAs offer Ubuntu a flexible way of adding additional software that wouldn't normally be made available, though you will need to keep an eye on such repositories to ensure they are properly patched when vulnerabilities arise, and are used only when absolutely necessary.

Keeping your server up to date

Updated packages are made available for Ubuntu quite often, sometimes even daily. These updates mainly include the latest security updates, but may also include new features. Since Ubuntu 16.04 is an LTS release, security updates are much more common than feature updates. Installing the latest updates on your server is a very important practice, but, unfortunately, it's not something that all administrators keep up on for one reason or another.

Security updates very rarely affect your server, other than helping to keep it secure against the latest threats. However, it's always possible that a security update that's intended to fix a security issue ends up breaking something else. This is rare, but I've seen it happen. When it comes to production servers, it's often difficult to keep them updated, since it may be catastrophic to your organization to introduce change within a server that's responsible for a large portion of your profits. If a server were to go down, it could be very costly. Then again, if your servers become compromised and your organization ends up the subject of a CNN hacking story, you'll definitely wish you had kept your packages up to date!

www.it-ebooks.info

The key to a happy data center is to test all updates before you install them. Many administrators will feature a system where updates will "graduate" from one environment into the next. For example, some may create physical or virtual clones of their production servers, update them, and then see if anything breaks. If nothing breaks, then those updates would be allowed on the production servers. In a clustered environment, an administrator may just update one of the production servers, see how it gets impacted, and then schedule a time to update the rest. In the case of workstations, I've seen policies where select users are chosen for security updates before they are uploaded to the rest of the population. I'm not necessarily suggesting you treat your users as guinea pigs, but everyone's organization is different, and finding the right balance in regards to installing updates is very important. Although these updates represent change, there's a reason that Ubuntu's developers went through the hassle of making them available. These updates fix issues, some of which are security concerns that are already being exploited as you read this.

To begin the process of installing security updates, the first step is to update your local repository index. As we've discussed before, the way to do so is to run `apt-get update` as root or with `sudo`. This will instruct your server to check all of its subscribed repositories to see if any new packages were added or out of date packages removed. Then, you can start the actual process.

There are two commands you can use to update packages. You can run either `apt-get upgrade` and/or `apt-get dist-upgrade`.

The difference is that running `apt-get upgrade` will not remove any packages and is the safest to use. However, this command also won't pull down any new dependencies either. Basically, the `apt-get upgrade` command simply updates any packages on your server that are already installed, without adding or removing anything. Since this command won't install anything new, this also means your server will not have updated kernels installed either.

The `apt-get dist-upgrade` command will update absolutely everything available. It will make sure all packages on your server are updated, even if that means installing a new package as a dependency that wasn't required before. If a package needs to be removed in order to satisfy a dependency, it will do that as well. If an updated kernel is available, it will be installed.

Generally speaking, the `dist-upgrade` variation should represent your end goal, but it's not necessarily where you should start. Updated kernels are important, since your distribution's kernel receives security updates just as any other package. All packages should be updated eventually, even if that means something is removed because it's no longer needed or something new ends up getting installed. I generally start with the `apt-get upgrade` command to get things going:

```
# apt-get update && apt-get upgrade
```


www.it-ebooks.info

In one line, I'm executing both `apt-get update` as well as `apt-get upgrade`, using `&&` to inform the shell that I would like to run the latter command after the former finishes. After my server fetches the latest indexes of available packages, my output will show a list of the packages it wants to update, ending with output similar to the following:

```
88 upgraded, 0 newly installed, 0 to remove and 8 not upgraded.
Need to get 33.2 MB of archives.
After this operation, 218 kB of additional disk space will be used.
Do you want to continue? [Y/n]
```

As you can see, I haven't updated this server in a while, as I have `88` updates available. From the output, you can also see that there are eight updates that won't be installed as part of this command, since we're running `apt-get upgrade` instead of `apt-get dist-upgrade`. I can press *Y* and then *Enter* to start the process, but, before I do, I generally like to look at the output to see which packages are going to be updated. I've omitted the entire output for the sake of saving space, but you would see a list of the packages your server wants to update, so you can have a chance to see what it wants to do before you allow it to continue. This is important—if a package is marked for an update and it's critical to your organization, you may want to consider what may happen if the package is updated while people are using it. I learned this the hard way earlier in my career, when the MySQL package was updated on a server, which I allowed to happen during production hours thinking I could always restart the MySQL service in the evening to complete the install. Actually, the MySQL service was automatically restarted as part of installing the update, which ended up causing a few minutes of downtime while I also needed to restart the Apache service to reconnect to the database. Thankfully, this server wasn't heavily used. I bring this up just as an example to demonstrate why you should take a look at the list of packages your server wants to update before you allow it to do so.

Anyway, after you confirm the changes by pressing *Y* and then *Enter*, `apt` will fetch all of the listed packages and install them. This, of course, may take a while if you have `88` or more packages waiting, as I did when I ran through this process on my test server for the sake of grabbing example output. Assuming that this process finishes successfully, we can run the `apt-get dist-upgrade` command to update the rest; specifically, the packages that were held back because they would've installed new packages or removed existing ones. Even if I don't plan on running an actual `dist-upgrade`, I'll often run it just long enough to see what apt wanted to do, as I can always cancel the process by pressing *N* and then *Enter* when I see the confirmation prompt.

www.it-ebooks.info

In my case, when I ran `apt-get dist-upgrade` on my test server, I received the following output:

```
The following packages were automatically installed and are no longer
required:
  linux-headers-4.3.0-5 linux-headers-4.3.0-5-generic linux-image-4.3.0-
5-generic
  linux-image-extra-4.3.0-5-generic
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  gcc-6-base initramfs-tools-core linux-base linux-headers-4.4.0-4
  linux-headers-4.4.0-4-generic linux-image-4.4.0-4-generic
  linux-image-extra-4.4.0-4-generic
The following packages will be upgraded:
  gcc-5-base initramfs-tools initramfs-tools-bin libgcc1 libstdc++6
linux-generic
  linux-headers-generic linux-image-generic
8 upgraded, 7 newly installed, 0 to remove and 0 not upgraded.
Need to get 67.6 MB of archives.
After this operation, 290 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

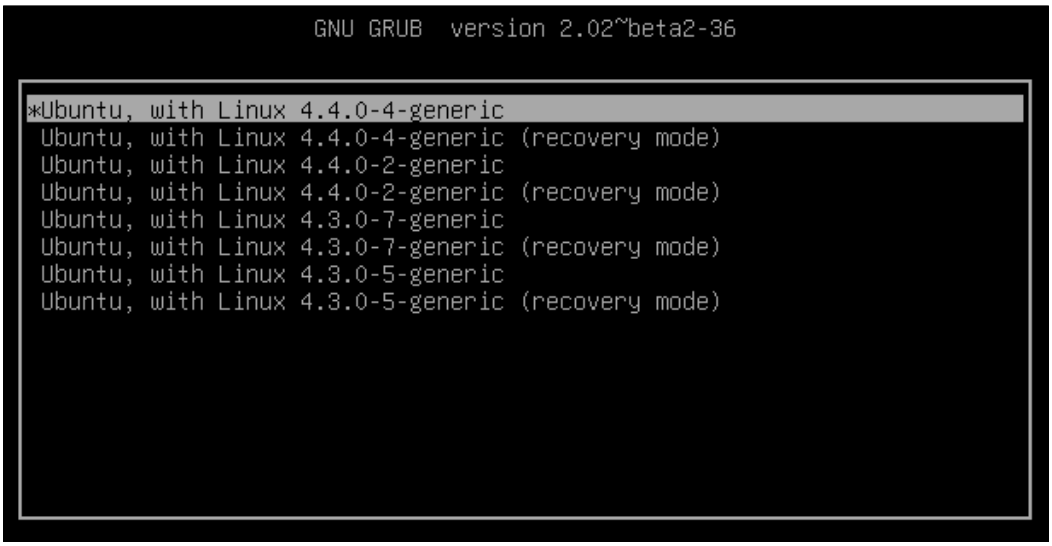
As you can see from the output, `apt` wanted to install some new packages to satisfy dependencies (`gcc-6-base`, `initramfs-tools-core`, `linux-base`, and several others), as well as update my running kernel (`linux-image-4.4.0-4-generic`) and its headers (`linux-headers-4.4.0-4-generic`). If I approve of installing these updates, I can press *Y* and then *Enter* at this point to accept these changes and allow `apt` to get to work on updating them.

Another reason why running `apt-get upgrade` followed by `apt-get dist-upgrade` is useful is because it narrows down the list of packages by quite a bit. When I ran `apt-get upgrade` on my server, I was offered `88` packages, so I didn't even bother to list the entire output, since that would've taken up most of a page. But, given the nature of `apt-get upgrade`, those packages were all generally safe to install, anyway, as that command doesn't change what packages you have installed, only their versions. That narrowed down the list of packages to just a handful when I ran the second command (`apt-get dist-upgrade`). Since those updates included kernel updates and some newly installed packages, I would have a decision to make, and I can do so without looking through more than 80 packages.

www.it-ebooks.info

In regards to updating the kernel, this process deserves some additional discussion. Some distributions are very risky when it comes to updating the kernel. On my Arch Linux installations, only one kernel is installed at any one time. Therefore, when that kernel gets updated, you really need to reboot the machine so that it can use it properly (sometimes, various system components may have difficulty in the case where you have a pending reboot after installing a new kernel).

Ubuntu, on the other hand, handles kernel upgrades very efficiently. When you update a kernel in Ubuntu, it doesn't replace the kernel your server is currently running on. Instead, it installs the updated kernel alongside your existing one. In fact, these kernels will continue to be stacked and none of them will be removed as new ones are installed. When new versions of the Ubuntu kernel are installed, the GRUB boot loader will be updated automatically to boot the new kernel the next time you perform a reboot. Until you do, you can continue to run on your current kernel for as long as you need to, and you shouldn't notice any difference. The only real difference is the fact you're not taking advantage of the additional security patches of the new kernel until you reboot, which you can do during your next maintenance window. The reason this method of updating is great is because if you run into a problem where the new kernel doesn't boot, you'll have a chance to press *Esc* at the beginning of the boot process, where you'll be able to choose the option **Advanced options for Ubuntu**, which will bring you to a list of all of your installed kernels. Using this list, you can select between your previous (known, working) kernels and continue to use your server as you were before you updated the kernel. This is a valuable safety net!



Selecting a different kernel during the boot process

www.it-ebooks.info

In my sample output of the `apt-get dist-upgrade` command, you may be curious about the following section if you aren't already aware of Ubuntu's way of package management:

```
The following packages were automatically installed and are no longer
required:
  linux-headers-4.3.0-5 linux-headers-4.3.0-5-generic linux-image-4.3.0-
5-generic
  linux-image-extra-4.3.0-5-generic
Use 'sudo apt autoremove' to remove them.
```

Here, we see a list of packages that `apt` doesn't think we need anymore. It may be right, but this is something we would need to investigate. In this case, it sees that we're installing a new kernel (and packages related to the kernel), so it doesn't think we'll need the old versions after the new ones are installed. This isn't just related to kernels; `apt` keeps track of packages that are installed as dependencies for other packages. When you remove a package that depends on those packages, or you update a package that required installation of something else (but no longer does), `apt` sees this as an opportunity to inform you that you can perform some cleanup. As instructed in the output, we can use the `apt-get autoremove` command as root or with `sudo` to remove them. This is a great way of keeping our installed packages clean, but should be used with care.

First, you should never remove outdated kernels from your server until you verify that the newly installed kernel is working correctly. Generally, you would probably want to wait at least a week before running `apt-get autoremove` when kernel packages are involved. When it comes to other packages, they are generally safe to remove with the `apt-get autoremove` command, since the majority of them will be packages that were installed as a dependency of another package that has since been removed. However, double check that you really do want to remove each of the packages before you allow it to do so.

Later on in this chapter, I'll give you an example of how to unmark a package for automatic removal, in case `autoremove` wants to remove a package you'd rather hang.

www.it-ebooks.info

After you update the packages on your server, you may want to restart services in order to take advantage of the new security updates. In the case of kernels, you would need to reboot your entire server in order to take advantage of kernel updates, but other updates do not require a reboot. Instead, if you restart the associated service, you'll generally be fine (if the update itself didn't already trigger a restart of a service). For example, if your DNS service (`bind9`) was updated, you would only need to execute the following to restart the service:

```
# systemctl restart bind9
```

Or, on older servers without `systemd`, execute the following:

```
# /etc/init.d/bind9 restart
```

These are two of the many unique points of the Linux platform: not needing to restart an entire server just for a few updates and being able to strategically plan your upgrade process. As I've mentioned before, it's important that you keep your servers up to date. It's also important that you come up with not only an update plan but also a roll-back plan in case things go wrong. You can do so by creating LVM snapshots (which can be restored in case an update goes wrong), or simply reinstalling an older version of a package manually (or by any other method you can come up with).

Previously downloaded packages are stored in the following directory:

```
/var/cache/apt/archives
```

There, you should find the actual packages that were downloaded as a part of your update process. In a case where you need to restore an updated package to a previously installed version, you can manually install a package with the `dpkg` command. Generally, the syntax will be similar to the following:

```
# dpkg -i /path/to/package.deb
```

To be more precise, you would use a command such as the following to reinstall a previously downloaded package, using an older Linux kernel as an example:

```
# dpkg -i /var/cache/apt/archives/linux-generic_4.3.0.7.8_amd64.deb
```

However, with the `dpkg` command, dependencies aren't handled automatically, so if you are missing a package that your target package requires as a dependency, the package will still be installed, but you'll have unresolved dependencies you'll need to fix. You can try to resolve this situation with `apt`:

```
# apt-get -f install
```


www.it-ebooks.info

The `apt-get -f install` command will attempt to fix your installed packages, looking for packages that are missing (but are required by an installed package), and will offer to install the missing dependencies for you. In the case where it cannot find a missing dependency, it will offer to remove the package that requires the missing packages if the situation cannot be worked out any other way.

Well, there you have it. At this point, you should be well on your way to not only installing packages, but keeping them updated as well. Before I close out this chapter, though, there are a few more concepts related to package management that will be useful for you to know.

Backing up and restoring packages

As you maintain your server, your list of installed packages will grow. If, for some reason, you needed to rebuild your server, you would need to reproduce exactly what you had installed before, which can be a pain. It's always recommended that you document all changes made to your server via a change control process, but, at the very least, keeping track of which packages are installed is an absolute must. In some cases, a server may only include one or two extra packages in order to meet its goal, but, in other cases, you may need an exact combination of software and libraries in order to get things working. Thankfully, the `dpkg` command allows us to export and import a list of packages to install.

To export a list of installed packages, we can use the following command:

```
dpkg –get-selections > packages.list
```

This command will dump a list of package selections to a standard text file. If you open it, you'll see a list of your installed packages, one per line. A typical line within the exported file will look similar to the following:

```
tmux                    install
```

With this list, we can import our selections back into the server if we need to reinstall Ubuntu Server, or into a new server that will serve a similar purpose. First, before we manage any packages, we should update our index:

```
# apt-get update
```

Next, we'll need to ensure we have the `dselect` package installed. At your shell prompt, type `which dselect` and you should see output similar to the following:

```
/usr/bin/dselect
```


www.it-ebooks.info

If you don't see output, you'll need to install the `dselect` package with `apt`:

```
# apt-get install dselect
```

Once that's complete, you can now import your previously saved package list, and have the missing packages reinstalled on your server. The following commands will complete the process:

```
# dselect update
# dpkg --set-selections < packages.list
# apt-get dselect-upgrade
```

After you have run those commands, the packages that are contained in your packages list, but aren't already installed, will be installed once you confirm the changes. This method allows you to easily restore the packages previously installed on your server, if for some reason you need to rebuild it, as well as setting up a new server to be configured in a similar way to an existing one.

Making use of aptitude

The `aptitude` command is a very useful, text-based utility for managing packages on your server. Some administrators use it as an alternative to `apt`, and it even has additional features that you won't find anywhere else. To get started, you'll need to install `aptitude`, if it isn't already:

```
# apt-get install aptitude
```

The most basic usage of `aptitude` allows you to perform functions you would normally be able to perform with `apt`. In the following table, I outline several example `aptitude` commands, as well as their `apt` equivalent:

`aptitude` **command**	**apt equivalent**
`# aptitude install <packagename>`	`# apt-get install <packagename>`
`# aptitude remove <packagename>`	`# apt-get remove <packagename>`
`# aptitude search <search term>`	`# apt-cache search <packagename>`
`# aptitude update`	`# apt-get update`
`# aptitude upgrade`	`# apt-get upgrade`
`# aptitude dist-upgrade`	`# apt-get dist-upgrade`

For the most part, the commands I listed in the preceding table operate in much the same way as their `apt` equivalents. The output from an `aptitude` command will typically look very close to the `apt` version. There are some noteworthy differences, though.

www.it-ebooks.info

First, compare the output from the search commands. Both commands will list package names, as well as a description for each package. The aptitude version places additional blank space in between the application name and description, which makes it look nicer. However, it also includes an additional column that the `apt` version doesn't include, which represents the state of the package. This column is the first column you'll see when you search for packages with aptitude, and will show `i` if the package is already installed, `p` if the package is not installed, and `v` if the package is virtual (a virtual package exists merely as a pointer to other packages). There are other values for this column; feel free to consult the man page for aptitude for more information.

Another useful trick with `aptitude` is the ability to fix a situation where the `apt-get` command lists a package as no longer being necessary, even though you would rather keep it around. If you recall, we went over this subject a few sections ago. Basically, if a package is installed as a dependency for another package (but the original package was removed), the package will be marked as automatically installed, which means it becomes a candidate for cleanup with the `apt-get autoremove` command. In some cases, you may wish to keep such a package around, and you'd rather the `autoremove` command not remove it from your system. To unmark a package as automatically installed, aptitude comes to our rescue with the following command:

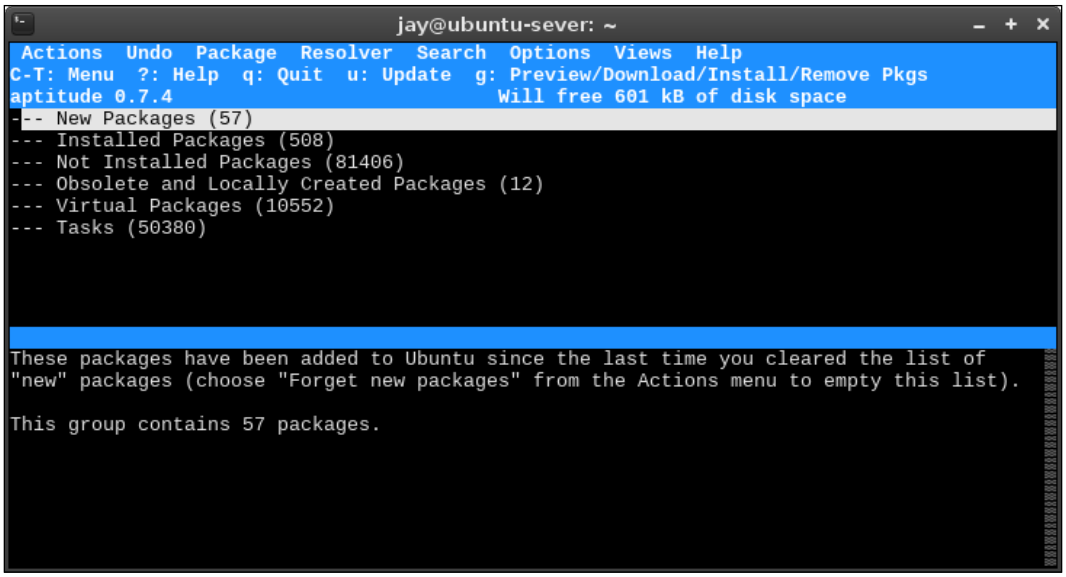
```
# aptitude unmarkauto <packagename>
```

After you unmark a package as automatically installed, it will no longer be a candidate for `autoremove`, and you can then use the `apt-get autoremove` command without fear that the package will be removed.

However, the `aptitude` command offers yet another nice feature in the form of a (somewhat) graphical interface you can interact with on the shell, if you enter the `aptitude` command (as root or with `sudo`) with no arguments or options:

```
# aptitude
```


www.it-ebooks.info



Aptitude in action

The `aptitude` command features an `ncurses` interface, which is essentially a terminal equivalent to a graphical application. You are able to interact with aptitude in this mode by using your arrow keys to move around, *Enter* to confirm selections, and *q* to quit. Using `Aptitude`'s graphical utility, you are basically able to browse the available packages, which are split into several categories. For example, you can expand **Not Installed Packages** to narrow down the list to anything not currently installed, then you can narrow down the list further by expanding a category (you simply press *Enter* to expand a list). Once you've chosen a category, you'll see a list of packages within that category that you can scroll through with your arrow keys. To manipulate packages, you'll access the menu by pressing *Ctrl* + *T* on your keyboard, which will allow you to perform additional actions. Once in the menu, you can mark the package as a candidate for installation by using your arrow keys to navigate to **Package**, and then you can mark the package as needing to be installed by selecting **Install**. You can also mark a package as needing to be removed or purged (meaning the package and its configuration are both removed). In addition, you can mark a package as automatically installed (**Mark Auto**) or manually installed (**Mark Manual**). Once you've made your selections, the first menu item (**Actions**) has an option to **Install/Remove** packages, which can be used to finalize your new selections or removals. To exit from `aptitude`, access the menu and then select **Quit**.

www.it-ebooks.info

There's more that you can do with the graphical version of `aptitude`, so feel free to play around with it on a test server and read its man pages to learn more. In addition to being an awesome way to manage packages, it also has a built-in game of Minesweeper!

As you can see, `aptitude` is a very useful utility, and some even prefer it to plain `apt`. Personally, I still use `apt`, though I always make sure `aptitude` is installed on my servers in case I need to benefit from its added functionality.

Installing Snap packages

Ubuntu 16.04 is the first LTS release of Ubuntu with support for Snap packages, which is a brand-new way for developers to distribute software. With Snap packages, developers can bundle all of the necessary prerequisites along with their application, greatly simplifying the deployment process for publishing their software and making it available to end users. Users benefit as well because Snap packages are easy to install and don't interfere with the underlying distribution.

To fully understand the benefits of using Snap packages, let's first consider the weaknesses of the current method of deploying software in Ubuntu, Apt repositories. Since Ubuntu's inception in 2004, the project has used Apt repositories to deploy software to its users. Before the release of a new version of Ubuntu, the repositories are "frozen," and from that point forward, no major changes are allowed, other than security updates. Historically, exceptions to this rule have been made, such as pushing out the latest versions of Firefox to desktop users. For the most part, though, once a new version of the Ubuntu distribution is released, no major updates to any applications are published. This means, for example, that an application will typically be stuck at the exact same version for the entire life cycle of that release of Ubuntu. If you want a newer version, your only options have been to either wait until the next release of Ubuntu or compile a newer version of the application yourself.

The benefit to the approach of using frozen repositories is generally that Ubuntu is more reliable. With no major changes throughout the life cycle of a release, you'll be using relatively stable versions of packages, rather than untested and bleeding edge software that may break. The downside is that, if you want newer software, you'll have to wait for a newer release of the distribution or find some other way of getting the software, which may include third-party repositories that may end up breaking the current packages you have installed.

www.it-ebooks.info

For developers, Apt repositories have been a blessing and a curse. When it comes to deploying software to Linux users, supporting Ubuntu has been a no-brainer. It's one of the largest Linux communities around. Developers have had the choice to push their applications directly to Debian or Ubuntu repositories, or create their own repository and offer their software through that. With the latter method, users will benefit from a newer application as soon as it's released by subscribing to the vendor's Apt repository. However, this methodology can also be frustrating for developers, as different versions of Ubuntu feature different libraries and additional testing is required to ensure the software works with every supported Ubuntu release. While there's certainly nothing wrong with that approach, and developers have used this approach for decades, it does represent an additional layer of complexity that a developer of a Linux application has to go through in order to reach their customers.

Snap packages operate independently of repositories. A user can install a Snap package in Ubuntu without disturbing the underlying distribution. This also allows developers to target their users more easily, since they no longer need to worry about which libraries a user may have installed. For example, if someone is developing an application that requires GTK version 3.20, but the target distribution only features GTK 3.18, that's no problem, as a Snap package can have the required libraries built-in, independent of the underlying distribution. The application would still work either way. End users benefit when it comes to Snap packages because they no longer need to wait for a new Ubuntu release in order to take advantage of the latest version of their favorite applications. As soon as developers make the Snap package available, end users will be able to install it.

As this book is being prepared for publication, Snap packages are still a very new concept and are only now starting to materialize. At the time of writing there aren't very many Snap packages available, and the technology is still maturing. Until Snap packages become more popular, installing packages via `apt` will still be the way to go for the time being. As I write this, there are only about 42 Snap packages available for Ubuntu, which is not many when you compare this number to the over 50,000 packages available via Apt. I predict that Snap packages will become very popular, and will be a major benefit for users. Developers will be able to target the Ubuntu platform more easily, and end users will be able to use newer versions of packages than they'd normally have available. It's a win for everyone.

To manage Snap packages, we use the `snap` command. The `snap` command features several options we can use to search for, install, and remove Snap packages from our server or workstation. To begin, we can use the `snap find` command to display a list of Snap packages available to us:

```
snap find
```


www.it-ebooks.info

With no options, the `snap find` command will display all Snap packages available, but if we wish to search for something in particular, we can also give it a string to search for. At the time of writing the list of available Snap packages is very small, so you probably won't need to use a search string, as you can see the entire library in one terminal screen. But, as the list grows in the future, you can shorten the list by supplying a name. One Snap package that's available currently (and makes for a useful example) is `nmap`, which you can search for with the following command:

```
snap find nmap
```

When I run the command, I see the following output.

```
nmap  7.12SVN-0.4  Nmap ("Network Mapper") is a free and open source
utility for network discovery and security auditing
```

The `nmap` utility is available in Ubuntu's default repositories, so you don't need to use the Snap package to install it. One thing you'll notice, though, is that the Snap version is newer than the version that ships with Ubuntu. At the time of writing Ubuntu's Apt version of the `nmap` package is 7.01, while Snap offers us a newer version, `7.12`. If we wish to install the Snap version, we can use the following command:

```
# snap install nmap
```

Now, if we check the location of `nmap` with the `which nmap` command, we see that the Snap version of `nmap` is run from a special place:

```
which nmap
/snap/bin/nmap
```

Now, when we run `nmap`, we're actually running it from `/snap/bin/nmap`. If we also have the `nmap` utility installed from Ubuntu's Apt repositories, then we can run either one at any time, since Snap packages are independent of the Apt packages. If we wish to run Ubuntu's version, we simply run it from `/usr/bin/nmap`.

Removing an installed Snap package is easy. We simply use the `remove` option:

```
# snap remove nmap
```

If we issue that command, then `nmap` (or whichever Snap package we designate) is removed from the system.

To update a package, we use the `refresh` option:

```
# snap refresh nmap
```

With that command, the package will be updated to the newest version available.

www.it-ebooks.info

As you can see, managing Snap packages is fairly straightforward. Using the `snap` suite of commands, we can install, update, or remove packages from our server or workstation. The `snap find` command allows us to find new Snap packages to install. Unfortunately, there aren't very many packages available for us to work with just yet, but, as the technology advances and evolves, I'm sure many great applications will be available via this method. By the time this book is released, I'm sure the number of Snap packages will have already grown.

Summary

In this chapter, we have taken a crash course through the world of package management. As you can see, Ubuntu Server offers an amazing number of software packages, and very useful tools we can use to manage them. We began the chapter with a discussion on how package management with Ubuntu works, then we worked through installing packages, searching for packages, and managing repositories. We have also discussed best practices for keeping our server up to date, as well as the commands available for us to install the latest updates. The `aptitude` command is also a neat alternative to the `apt` suite of commands, and in this chapter we looked at its GUI mode as well as how it differs from Apt. Snap packages were also covered, which is an exciting up-and-coming technology that will greatly enhance software distribution on Ubuntu.

In *Chapter 6*, *Controlling and Monitoring Processes*, we're going to take a look at monitoring and managing the processes running on our server. We'll take a look at job management, as well as starting and stopping processes.

www.it-ebooks.info

www.it-ebooks.info

6
Controlling and Monitoring Processes

On a typical Linux server, there can be over a hundred processes running at any given time. The purposes of these processes ranges from system services such as Apache, **Network Time Protocol** (**NTP**), and others, to processes that serve information to others, such as the Apache web server. As an administrator of Ubuntu servers, you will need to be able to manage these processes, as well as manage the resources available to them. In this chapter, we'll take a look at process management, including the `ps` command, managing `job` control commands, and more. As we work through these concepts, we will cover:

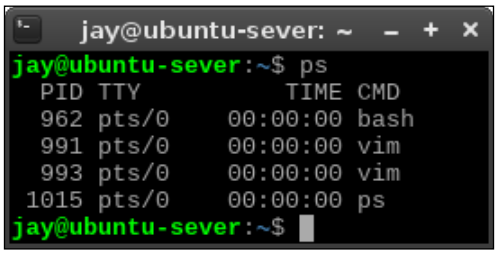
- Showing running processes with the `ps` command
- Managing jobs
- Killing misbehaving processes
- Utilizing `htop`
- Managing system processes
- Enabling services to run at boot time
- Managing processes on legacy servers with upstart
- Monitoring memory usage
- Scheduling tasks with Cron
- Understanding load average

www.it-ebooks.info

Showing running processes with the ps command

While managing our server, we'll need to understand what processes are running and how to manage these processes. Later in this chapter, we'll work through starting, stopping, and monitoring processes. But before we get to those concepts, we first need to be able to determine what is actually running on our server. The `ps` command allows us to do this.

When executed by itself, the `ps` command will show a list of processes running by the user that called the command.



The output of the ps command, when run as a normal user and with no options

In the example screenshot I provided, you can see that when I ran the `ps` command as my own user with no options, it showed me a list of processes that I am running as myself. In this case, I have a couple of `vim` sessions open, and in the last line, we also see `ps` itself, which is also included in the output.

On the left side of the output, you'll see a number for each of the running processes. This is known as the **Process ID** (**PID**), which is what I'll refer to it as from now on. Before we continue on, the PID is something that you really should be familiar with, so we may as well cover it right now.

Each process running on your server is assigned a PID, which differentiates it from other processes on your system. You may understand a process as `vim`, or `top`, or some other name. However, our server knows processes by their ID. When you open a program or start a process, it's given a PID by the kernel. As you work with managing your server, you'll find that the PID is useful to know, especially for the commands we'll be covering in this very chapter. If you want to kill a misbehaving process, for example, a typical work-flow would be for you to find the PID of that process and then reference that PID when you go to kill the process (which I'll show you how to do in a later section). PIDs are actually more involved than just a number assigned to running processes, but for the purposes of this chapter, that's the main purpose we'll need to remember.

www.it-ebooks.info

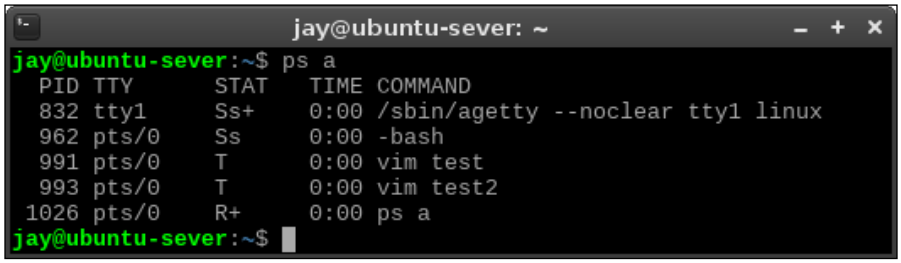
You can also use the `pidof` command to find the PID of a process if you know the name of it. For example, I showed a screenshot of two `vim` processes running with the PIDs `991` and `993` respectively. If you run the following command:

```
pidof vim
```

The output will give you the PID(s) of the process without you having to use the `ps` command.

Continuing with the `ps` command, there are several useful options you can give the command in order to change the way in which it shows its output. If you use the a option, you'll see more information than you normally would:

```
ps a
```



The output of the ps a command

With `ps a`, we're seeing the same output as before, but with additional information, as well as column headings at the top. We now see a heading for the **PID**, **TTY**, **STAT**, **TIME**, and **COMMAND**. From this new output, you can see that the two `vim` processes I have running are editing files named `test` and `test2` respectively. This is great to know, because if one of them was misbehaving, you would probably want to know which one was which so you know to stop the appropriate process.

The **TTY**, **STAT**, and **TIME** fields are new; we didn't see those when we ran `ps` by itself. We saw the other fields, although we didn't see a formal heading at the top. The **PID** column we've already covered, so I won't go into any additional detail about that. The **COMMAND** field tells us the actual command being run, which is very useful if we either want to ensure we're managing the correct process or see what a particular user is running (I'll demonstrate how to show processes for other users soon).

www.it-ebooks.info

What may not be obvious at first are the **STAT**, **TIME** and **TTY** fields. The **STAT** field gives us the status code of the process, which refers to which state the process is currently in. The state can be uninterruptible sleep (*D*), defunct (*Z*), stopped (*T*), interruptible sleep (*S*), and in the run queue (*R*). There is also paging (*W*), but that is not used anymore, so there's no need to cover it. Interruptible sleep means that the program is idle: it's waiting for input in order to awaken. Uninterruptible sleep is a state in which a process is generally waiting on input and cannot handle additional signals (we'll briefly talk about signals later on in this chapter). A defunct process (also referred to as a zombie process) has, for all intents and purposes, finished its job but is waiting on the parent to perform cleanup. Defunct processes aren't actually running, but remain in the process list and should normally close on their own. If such a process remains indefinitely and doesn't close, it can be a candidate for the `kill` command, which we will discuss later. A stopped process is generally a process that has been sent to the background, which will be discussed in the next section.

The **TTY** column tells us which TTY the process is attached to. A TTY refers to a **Teletypewriter**, which is a term used from a much different time-period. In the past, a Teletypewriter would be used to electronically send signals to a typing device on the other end of a wire. Obviously, we don't use machines like these nowadays, but the concept is similar from a virtual standpoint. On our server, we're using our keyboard to send input to a device which then displays output to another device. In our case, the input device is our keyboard and the output device is our screen, which is either connected directly to our server or is located on our computer which is connected to our server over a service such as SSH. On a Linux system, processes run on a TTY, which is (for all intents and purposes), a terminal that grabs output and manages that output, similar to a Teletypewriter in a virtual sense. A terminal is our method of interacting with our server.

In the screenshot I provided, we have one process running on a **TTY** of `tty1`, and the other processes are running on `pty0`. The `tty` we see is the actual terminal device, and `pty` references a virtual (pseudo) terminal device. Our server is actually able to run several `tty` sessions, typically 1 to 7. Each of these can be running their own programs and processes. To understand this better, try pressing *Ctrl* + *Alt*+ any function key, from *F1* through *F7* (if you have a physical keyboard plugged into a physical server). Each time, you should see your screen cleared and then moved to another terminal. Each of these terminals is independent of one another. Each of your function keys represents a specific **TTY**, so by pressing *Ctrl* + *Alt* + *F6*, you're switching your display to **TTY 6**.

www.it-ebooks.info

Essentially, you're switching from **TTY1** through to **TTY7**, with each being able to contain their own running processes. If you run `ps a` again, you'll see any processes you start on those TTYs show up in the output as a `tty` session, such as `tty2` or `tty4`. Processes that you start in a terminal emulator will be given a designation of `pty`, because they're not running in an actual TTY, but rather a Pseudo-TTY. This was a long discussion for something that ended up being simple (TTY or Pseudo-TTY), but with this knowledge you should be able to differentiate between a process running on the actual server or through a shell.

Continuing, let's take a look at the **TIME** field of our `ps` command output. This field represents the total amount of time the CPU has been utilized for that particular process. However, the time is `0:00` for each of the processes in the screenshot I've provided. This may be confusing at first. In my case, the `vim` processes in particular have been running for about 15 minutes or so since I took the screenshot, and they still show `0:00` utilization time even now. Actually, this isn't the amount of time the process has been running, but rather the amount of time the process has been actively engaging with the CPU. In the case of `vim`, each of these processes is just a buffer with a file open. For the sake of comparison, the Linux machine I'm writing this chapter on has a process ID of `759` with a time of `92:51`. PID `759` belongs to my X server, which provides the foundation for my graphical user environment and windowing capabilities. However, this laptop currently has an uptime of 6 days and 22 hours as I type this, which is roughly equivalent to 166 hours, which is not the same amount of time that PID `759` is reporting for its TIME. Therefore, we can deduce that even though my laptop has been running 6 days straight, the X server has only utilized 92 hours and 51 minutes of actual CPU time. In summary, the **TIME** column refers to the amount of time a process needs the CPU to calculate something and is not necessarily equal to how long something has been running, or for how long a graphical process is showing on your screen.

Let's continue on with the `ps` command and look at some additional options. First, let's see what we get when we add the `u` option to our previous example, which gives us the following example command:

```
ps au
```

When you run it, you should notice the difference from the command `ps a` right away. With this variation, you'll see processes listed that are being run by your user ID, as well as other users. When I run it, I see processes listed in the output for my user (`jay`), as well as `root`. The `u` option will be a common option you're likely to use, since most of the time while managing servers you're probably more interested in keeping an eye on what kinds of shenanigans your users are getting themselves into. But perhaps the most common use of the `ps` command is the following variation:

```
ps aux
```


www.it-ebooks.info

With the `x` option added, we're no longer limiting our output to processes within a TTY (either native or pseudo). The result is that we'll see a lot more processes, including system-level processes, that are not tied to a process we started ourselves. In practice, though, the `ps aux` command is most commonly used with `grep` to look for a particular process or string. For example, let's say you want to see a list of all `nginx` worker processes. To do that, you may execute a command such as the following:

```
ps aux | grep nginx
```

Here, we're executing the `ps aux` command as before, but we're piping the output into `grep`, where we're looking only for lines of output that include the string `nginx`. In practice, this is the way I often use `ps`, as well as the way I've noticed many other administrators using it. With `ps aux`, we are able to see a lot more output, and then we can narrow that down with a search criteria by piping into `grep`. However, if all we wanted to do was to show processes that have a particular string, we can also do the following:

```
ps u  -C nginx
```

Another useful variation of the `ps` command is to sort the output by sorting the processes using the most CPU first:

```
ps aux --sort=-pcpu
```

Unfortunately, that command shows a lot of output, and we would have to scroll back to the top in order to see the top processes. Depending on your terminal, you may not have the ability to scroll back very far (or at all), so the following command will narrow it down further:

```
ps aux --sort=-pcpu | head -n 5
```

Now that is useful! With that example, I'm using the `ps aux` command with the `--sort` option, sorting by the percentage of CPU utilization (`-pcpu`). Then I'm piping the output into the `head` command, where I'm instructing it to show me only five lines (`-n 5`). In fact, I can do the same but with the most-used memory instead:

```
ps aux --sort=-pmem | head -n 5
```

If you want to determine which processes are misbehaving and using a non-ordinary amount of memory or CPU, those commands will help you narrow it down. The `ps` command is a very useful command for your admin toolbox. Feel free to experiment with it beyond the examples I've provided; you can consult the man pages for the `ps` command to learn even more tricks. In fact, the second section of the man page for `ps` (under examples) gives you even more neat examples to try out.

www.it-ebooks.info

Managing jobs

Up until now, everything we have been doing on the shell has been right in front of us, from execution to completion. We've installed applications, run programs, and walked through various commands. Each time, we've had control of our shell taken from us, and we've only been able to start a new task when the previous one has finished. For example, if we were to install the `vim-nox` package with the `apt-get` command, we would watch helplessly while `apt` takes care of fetching the package and installing it for us. While this is going on, our cursor goes away and our shell completes the task for us without allowing us to queue up another command. We can always open a new shell to the server and multi-task by having two windows open at once, each doing different tasks. Actually, we don't have to do that, unless we want to. We can actually background a process without waiting for it to complete while working on something else. Then, we can bring that process back to the front to return working on it or to see whether or not it has finished successfully. Think of this as a similar concept to a windowing desktop environment, or user interfaces on the Windows or Mac OS X operating systems. We can work on an application, minimize it to get it out of the way, and then maximize it to continue working with it. Essentially, that's the same concept of backgrounding a process in a Linux shell.

In my opinion, the easiest way to learn a new concept is to try it out, and the easiest example I can think of is by using a text editor. In fact, this example is extremely useful and may just become a part of your daily workflow. To do this exercise, you can use any command line text editor you prefer, such as `vim` or `nano`. On Ubuntu Server, `nano` is usually installed by default, so you already have it if you want to go with that. If you prefer to use `vim`, feel free to install the `vim-nox` package:

```
# apt-get install vim-nox
```

If you want to use `vim`, you can actually install `vim` rather than `vim-nox`, but I always default to `vim-nox` since it features built-in support for scripting languages.

Again, feel free to use whichever text editor you want. Teaching you how to use a text editor is beyond the scope of this book, so just use whatever you feel comfortable with. I find that in the Linux community, `vim` seems to be the standard for most administrators. However, the `nano` text editor is much easier to use and should be your choice if you have no experience with using text editors on the Linux shell. In the following examples, I'll be using `vim`. However, if you use `nano`, just replace `vim` with `nano` every time you see it.

Anyway, to see backgrounding in action, open up your text editor. Feel free to open a file or just start a blank session. With the text editor open, we can background it at any time by pressing *Ctrl* + *Z* on our keyboard.

www.it-ebooks.info

You can only background `vim` when you are not in **Insert Mode**, since it captures *Ctrl* + *Z* rather than passing it to the shell.

Did you see what happened? You were immediately taken away from your editor and were returned to the shell so you can continue to use it. You should have seen some output similar to the following:

```
[1]+  Stopped                 vim
```

Here, we see the `job` number of our process, its status, and then the name of the process. Even though the process of your text editor shows a status of `Stopped`, it's still running. You can confirm this by using the following variation of the `ps` command:

```
ps au |grep vim
```

In my case, I see the `vim` process running with a PID of `1070`:

```
jay       1070  0.0  0.9  39092  7320 pts/0    T    15:53   0:00 vim
```

At this point, I can execute additional commands, navigate around my filesystem, and get additional work done. When I want to bring my text editor back, I can use the `fg` command to foreground the process, which will bring it right back. If I have multiple background processes, the `fg` command will bring back the one I was working on most recently.

I gave you an example of the `ps` command to show that the process was still running in the background, but there's actually a dedicated command for that purpose, and that is the jobs command. If you execute the jobs command, you'll see in the output a list of all the processes running in background. Here's some example output:

```
[1]-  Stopped                 vim file1.txt
[2]+  Stopped                 vim file2.txt
```

The output shows that I have two `vim` sessions in use, one modifying `file1.txt`, and the other modifying `file2.txt`. If I were to execute the `fg` command, that's going to bring up the vim session that's editing `file2.txt`, since that was the last one I was working in. That may or may not be the one I want to return to editing, though. Since I have the `job` ID on the left, I can bring up a specific background process by using its ID with the `fg` command:

```
fg 1
```


www.it-ebooks.info

Knowing how to background a process can add quite a bit to your workflow. For example, let's say, hypothetically, that I'm editing a `config` file for a server application, such as Apache. While I'm editing this `config` file, I need to consult the documentation (man page), for Apache because I forgot the syntax for something. I could open a new shell and an SSH session to my server and view the documentation in another window. This can get very messy if I open up too many shells. Better yet, I can background the current vim session, read the documentation, and then foreground the process with the `fg` command to return to working on it. All from one SSH session!

To background a process, you don't have to use *Ctrl + Z*; you can actually background a process right when you execute it by entering a command with the ampersand symbol (`&`) typed at the end. To show how this works, I'll use `htop` as an example. Admittedly, this may not necessarily be the most practical example, but it does work to show you how to start a process and have it backgrounded right away. We'll get to the `htop` command later in this chapter, but for now feel free to install this package and then run it with the ampersand symbol:

```
# apt-get install -y htop
htop &
```

The first command, as you already know, installs the `htop` package on our server. With the second command, I'm opening `htop` but backgrounding it immediately. What I'll see when it's backgrounded is its job ID and PID. Now, at any time, I can bring `htop` to the foreground with `fg`. Since I just backgrounded it, `fg` will bring `htop` back since it considers it the most recent. As you know, if it wasn't the most recent, I could reference its job ID with the `fg` command to bring it back even if it wasn't my most recently used job. Go ahead and practice using the ampersand symbol with a command and then bringing it back to the foreground. In the case of `htop`, it can be useful to start it, background it, and then bring it back any time you need to check the performance of your server.

Keep in mind, though, that when you exit your shell, all your backgrounded processes will close. If you have unsaved work in your `vim` sessions, you'll lose what you were working on. For this reason, if you utilize background processes, you may want to check to see if you have any pending jobs still running by executing the jobs command before logging out.

www.it-ebooks.info

In addition, you'll probably notice that some applications background cleanly, while others don't. In the case of using a text editor and `htop`, those applications stay quietly running in the background, allowing us to perform other tasks and allowing us to return to those commands later. However, some applications may still spit out diagnostic text regularly in your main window, whether they're backgrounded or not. To get even more control over your bash sessions, you can learn how to use a multiplexer, such as `tmux` or `screen`, to allow these processes to run in their own session such that they don't interrupt your work. Going over the use of a program such as `tmux` is beyond the scope of this book, but it is a useful utility to learn if you're interested.

Killing misbehaving processes

Regarding the `ps` command, by this point you know how to display processes running on your server, as well as how to narrow down the output by string or resource usage. But what can you actually do with that knowledge? As much as we hate to admit it, sometimes the processes our server runs fail or misbehave and you need to restart them. If a process refuses to close normally, you may need to kill that process. In this section, we introduce the `kill` and `killall` commands to serve that purpose.

The `kill` command accepts a PID as an option and attempts to close a process gracefully. In a typical workflow where you need to terminate a process that won't do so on its own, you will first use the `ps` command to find the PID of the culprit. Then, knowing the PID, you can attempt to kill the process. For example, if PID `31258` needed to be killed, you could execute the following:

```
# kill 31258
```

If all goes well, the process will end. You can restart it or investigate why it failed by perusing its logs.

To better understand what the `kill` command does, you first will need to understand the basics of **Linux Signals**. Signals are used by both administrators as and developers and can be sent to a process either by the kernel, another process, or manually with a command. A signal instructs the process of a request or change, and in some cases, to completely terminate. An example of such a signal is `SIGHUP`, which instructs processes that their controlling terminal has exited. One situation in which this may occur is when you have a terminal emulator open, with several processes inside it running. If you close the terminal window (without stopping the processes you were running), they'll be sent the `SIGHUP` signal, which basically tells them to quit (essentially, it means the shell quit or hung up).

www.it-ebooks.info

Other examples include `SIGINT` (where an application is running in the foreground and is stopped by pressing *Ctrl* + *C* on the keyboard), and `SIGTERM`, which when sent to a process asks it to cleanly terminate. Yet another example is `SIGKILL`, which forces a process to terminate uncleanly. In addition to a name, each signal is also represented by a value, such as `15` for `SIGTERM` and `9` for `SIGKILL`. Going over each of the signals is beyond the scope of this chapter (the advanced topics of signals are mainly only useful for developers), but you can view more information about them by consulting the man page if you're curious:

```
man 7 signal
```

For the purposes of this section, the two types of signal we're most concerned about are `SIGTERM(15)` and `SIGKILL(9)`. When we want to stop a process, we would send one of these signals to it, and the `kill` command allows us to do just that. By default, the `kill` command sends signal `15` (`SIGTERM`), which tells the process to cleanly terminate. If successful, the process will free its memory and gracefully close. With our previous example `kill` command, we sent signal `15` to the process, since we didn't clarify which signal to send.

Terminating a process with `SIGKILL(9)` is considered an extreme last resort. When you send signal `9` to a process, it's the equivalent of ripping out the carpet from underneath it or blowing it up with a stick of dynamite. The process will be force-closed without giving it any time to react at all, so it's one of those things you should avoid using unless you've literally tried absolutely everything you can think of. In theory, sending signal `9` can cause corrupted files, memory issues, or other shenanigans to occur. As for me, I've never actually run into long-term damage to software from using it, but theoretically it can happen, so you would want to only use it in extreme cases. One case where such a signal may be necessary is regarding `defunct` (zombie) processes in a situation where they don't close on their own. These processes are basically dead already and are typically waiting on their parent process to reap them. If the parent process never attempts to do so, they will remain on the process list. This in and of itself may not really be a big issue, since these processes aren't technically doing anything. But if their presence is causing problems and you can't kill them, you could try to send `SIGKILL` to the process. There should be no harm in killing a zombie process, but you would want to give them time to be reaped first.

To send signal `9` to a process, you would use the `-9` option of the `kill` command. It should go without saying, though: make sure you're executing it against the proper process ID!

```
# kill -9 31258
```


www.it-ebooks.info

Just like that, the process will vanish without a trace. Anything it was writing to will be in limbo, and it will be removed from memory instantly. If, for some reason, the process still manages to stay running (which is extremely rare), you probably would need to reboot the server to get rid of it, which is something I've only seen in a few, very rare cases. If `kill -9` doesn't get rid of the process, nothing will.

Another method of killing a process is with the `killall` command, which is probably safer than the `kill` command (if for no other reason than there's a smaller chance you'll accidentally kill the wrong process). Like `kill`, `killall` allows you to send `SIGTERM` to a process, but unlike `kill`, you can do so by name. In addition, `killall` doesn't just kill one process, it kills any process it finds with the name you've given it as an option. To use `killall`, you would simply execute `killall` along with the name of a process:

```
# killall myprocess
```

Just like the `kill` command, you can also send signal `9` to the process as well:

```
# killall -9 myprocess
```

Again, use that only when necessary. In practice, though, you probably won't use `killall -9` very often (if ever), because it's rare for multiple processes under the same process name to become locked. If you need to send signal `9`, stick to the `kill` command if you can.

The `kill` and `killall` commands can be incredibly useful in the situation of a "stuck" process, but these are commands you would hope you don't have to use very often. Stuck processes can occur in situations where applications encounter a situation from which they can't recover, so if you constantly find yourself needing to `kill` processes, you may want to check for an update to the package responsible for the service or check your server's RAM (which I will show you how to do in *Chapter 13, Troubleshooting Ubuntu Servers*).

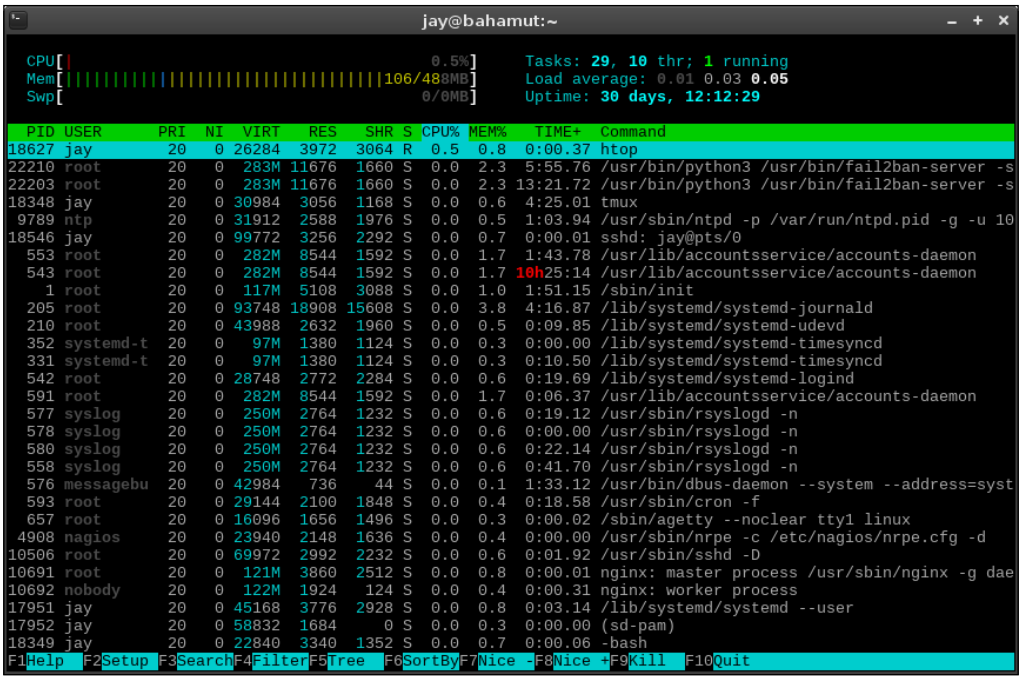
Utilizing htop

When wanting to view the overall performance of your server, nothing beats `htop`. Although not typically installed by default, `htop` is one of those utilities that I recommend everyone installs, since it's indispensable when wanting to check on the resource utilization of your server. If you don't already have `htop` installed, all you need to do is install the `htop` package:

```
# apt-get install htop
```


www.it-ebooks.info

When you run `htop` at your shell prompt, you will see the `htop` application in all its glory. In some cases, it may be beneficial to run `htop` as `root`, since doing so does give you additional options such as being able to kill processes, though this is not required.



Running htop

At the top of the `htop` display, you'll see a progress meter for each of your cores (the server in my screenshot only has one core), as well as a meter for memory and swap. In addition, the upper portion will also show you your **Uptime**, **Load average**, and the number of **Tasks** you have running. The lower section of `htop`'s display will show you a list of processes running on your server, with fields showing you useful information such as how much memory or CPU is being consumed by each process, as well as the command being run, the user running it, and its PID. To scroll through the list of processes, you can press *Page Up* or *Page Down* or use your arrow keys. In addition, `htop` features mouse support, so you are also able to click on columns at the top in order to sort the list of processes by that criteria. For example, if you click on **MEM%** or **CPU%**, the process list will be sorted by memory or CPU usage respectively. The contents of the display will be updated every two seconds.

www.it-ebooks.info

The `htop` utility is also customizable. If you prefer a different color scheme, for example, you can press *F2* to enter **Setup** mode, navigate to **Colors** on the left, and then you can switch your color scheme to one of the six that are provided. Other options include the ability to add additional meters, add or remove columns, and more. One tweak I find especially helpful on multi-core servers is the ability to add an average CPU bar. Normally, `htop` shows you a meter for each core on your server, but if you have more than one, you may be interested in the average as well. To do so, enter **Setup** mode again (*F2*), then with **Meters** highlighted, arrow to the right to highlight CPU and then press *F5* to add it to the left column. There are other meters you can add as well, such as **Load average**, **Battery**, and more.

When you open `htop`, you will see a list of processes for every user on the system. When you have a situation where you don't already know which user/process is causing extreme load, this is ideal. However, a very useful trick (if you want to watch a specific user) is to press *U* on your keyboard, which will open up the **Show processes of:** menu. On this menu, you can highlight a specific user by highlighting it with the up or down arrow keys and then pressing *Enter* to only show processes for that user. This will greatly narrow down the list of processes.

Another useful view is the **Tree** view, which allows you to see a list of processes organized by their parent/child relationship, rather than just a flat list. In practice, it's common for a process to be spawned by another process. In fact, all processes in Linux are spawned from at least one other process, and this view shows that relationship directly. In a situation where you are stopping a process only to have it immediately respawn, you would need to know what the parent of that process is in order to stop it from resurrecting itself. Pressing *F5* will switch `htop` to **Tree** view mode, and pressing it again will disable the **Tree** view.

As I've mentioned, `htop` updates its stats every two seconds by default. Personally, I find this to be ideal, but if you want to change how fast it refreshes, you can call `htop` with the `-d` option and then apply a different number of seconds (entered in tenths of seconds) for it to refresh. For example, to run `htop` but have it update every 7 seconds, start `htop` with the following command:

```
htop -d 70
```

To kill a process with `htop`, use your up and down arrow keys to highlight the process you wish to kill and press *F9*. A new menu will appear, giving you a list of signals you are able to send to the process with `htop`. `SIGTERM`, as we discussed before, will attempt to gracefully terminate the process. `SIGKILL` will terminate it uncleanly. Once you highlight the signal you wish to send, you can send it by pressing *Enter* or cancel the process with *Esc*.

www.it-ebooks.info

As you can see, `htop` can be incredibly useful, and has (for the most part) replaced the legacy `top` command that was popular in the past. The `top` command is available by default in Ubuntu Server and is worth a look, if only as a comparison to `htop`. Like `htop`, the `top` command gives you a list of processes running on your server, as well as their resource usage. There are no pretty meters and there is less customization possible, but the `top` command serves the same purpose. In most cases, though, `htop` is probably your best bet going forward.

Managing system processes

System processes, also known as daemons, are programs that run in the background on your server, and are typically started automatically when it boots. We don't usually manage these services directly as they run in the background to perform their duty, with or without needing our output. For example, if our server is a DHCP server and runs the `isc-dhcp-server` process, this process will run in the background, listening for DHCP requests and providing new IP assignments to them as they come in. Most of the time, when we install an application that runs as a service, Ubuntu will configure it to start when we boot our server, so we don't have to start it ourselves. Assuming the service doesn't run into an issue, it will happily continue performing its job forever until we tell it to stop. In Linux, services are managed by the init system, also referred to as `PID 1` since the init system of a Linux system always receives that PID.

Recently, the way in which processes are managed in Ubuntu Server has changed considerably. As of Ubuntu 15.04, Ubuntu has switched to `systemd` for its init system, which was previously Upstart until just recently. Ubuntu 16.04 is the first LTS release of Ubuntu with `systemd`, so if you generally stick to LTS releases, Ubuntu 16.04 may be your first exposure to it. The previous LTS release (Ubuntu 14.04) used Upstart, so depending on whether or not you've used Ubuntu since 15.04, that may be what you're accustomed to. Even though Ubuntu 16.04 uses `systemd`, and that is the preferred init system going forward, Ubuntu 14.04 is still supported at the time of writing, and it's very possible that you may have a mix of both as the older systems age out. For that reason, I'll go over both init systems in this section.

First, let's take a look at the way in which you manage services with `systemd`. With `systemd`, services are known as **Units**, though, for all intents and purposes, the terms mean the same thing. The `systemd` suite features the `systemctl` command, which allows you to start, stop, and view the status of processes on your server. To help illustrate this, I'll use Open SSH as an example, though you'll use the same command if you're managing other services, as all you would need to do is change the service name.

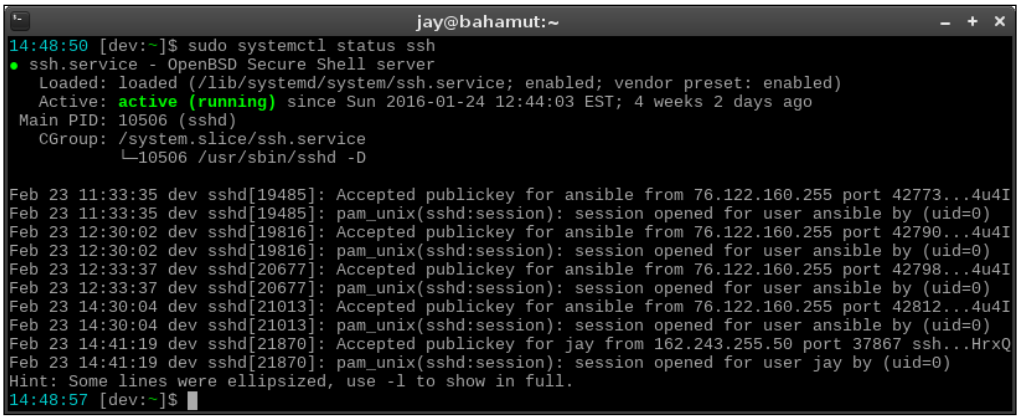
www.it-ebooks.info

The `systemctl` command, with no options or parameters, assumes the `list-units` option, which dumps a list of units to your shell. This can be a bit messy, though, so if you already know the name of a unit you'd like to search for, you can pipe the output into `grep` and search for a string. This is handy in a situation where you may not know the exact name of the unit, but you know part of it:

```
systemctl |grep ssh
```

If you want to check a service, the best way is to actually use the `status` keyword, which will show you some very useful information regarding the unit. This information includes whether or not the unit is running, if it's enabled (meaning it's configured to start at boot time), as well as the most recent log entries for the unit:

```
systemctl status ssh
```



Running htop

In my screenshot, you can see that I used `sudo` in order to run the `systemctl status ssh` command, but I didn't have to. You can actually check the status of most units without needing root access, but you may not see all the information available. If I were to have checked the status of the `ssh` unit without using root or `sudo`, I wouldn't have seen the log entries, since only root has access to view them. Another thing you may notice in the screenshot is that the name of the `ssh` unit is actually `ssh.service`, but you don't need to include the `.service` part of the name, since that is implied by default. Sometimes, while viewing the status of a process with `systemctl`, the output may be condensed to save space on the screen. To avoid this and see the full log entries, add the `-l` option:

```
systemctl status -l ssh
```


www.it-ebooks.info

Another thing to pay attention to is the `vendor preset` of the unit. As I've mentioned before, most packages in Ubuntu that include a service will enable it automatically, but other distributions which feature `systemd` may not. In the case of the `ssh` example, you can see that the `vendor preset` is set to `enabled`. This means that once you install `openssh-server`, the unit will automatically be enabled. You can confirm this by checking the **Active** line (where the example output says **active (running)**), which tells us that the unit is running. The **Loaded** line clarifies that the **Unit** is `enabled`, so we know that the next time we start the server, `ssh` will be loaded automatically. A unit automatically becoming enabled and starting automatically may vary in packages that are configured differently, so make sure you check this output whenever you install a new application.

Starting and stopping a unit is just as easy; all you have to do is change the keyword you use with `systemctl` to `start` or `stop` in order to have the desired effect:

```
# systemctl stop ssh
# systemctl start ssh
```

There are additional keywords, such as `restart` (which takes care of the previous two command examples at the same time), and some units even feature reload, which allows you to activate new configuration settings without needing to bring down the entire application. An example of why this is useful is with Apache, which serves web pages to local or external users. If you stop Apache, all users will be disconnected from the website you're serving. If you add a new site, you can use `reload` rather than `restart`, which will activate any new web pages you've added to the configuration without disturbing the existing ones. Not all units feature a `reload` option, so you should check the documentation of the application that provides the unit to be sure.

If a unit is not currently enabled, you can enable it with the `enable` keyword:

```
# systemctl enable ssh
```

It's just as easy to disable a unit as well:

```
# systemctl disable ssh
```

The `systemd` suite includes other commands as well, such as `journalctl`. The reason I refer to `systemd` as a suite (and not just as an init system) is because it handles more than just starting and stopping processes, with more features being added every year. The `journalctl` command allows us to view logfiles, but I'll wait until *Chapter 13*, *Troubleshooting Ubuntu Servers*, which is where I'll give you an introduction on how that works. For the purposes of this chapter, however, if you understand how to start, stop, enable, disable, and check the status of a **Unit**, you're good to go for now.

www.it-ebooks.info

If you have a server with an older version of Ubuntu Server installed, commands from the `systemd` suite will not work, as `systemd` is still relatively new. Older versions of Ubuntu use `Upstart`, which we will take a look at now.

Managing running services with `Upstart` is done with the `service` command. To check the status of a process with `Upstart`, we can do the following:

```
service ssh status
```

In my case, I get the following output:

```
ssh start/running, process 907
```

As you can see, we don't get as much output as we do with `systemctl` on `systemd` systems, but it at least tells us that the process is running and provides us with its PID. Stopping the process and starting it again gives us similar (limited) output:

```
# service ssh stop
ssh stop/waiting
# service ssh start
ssh start/running, process 1304
```

While the `Upstart` method of managing services may not give us as verbose information as `systemd`, it's fairly straightforward. We can start, stop, restart, reload, and check the status of a service of a given name. For the purposes of managing processes, that's pretty much it when it comes to `Upstart`.

If you're curious, `Upstart` stores the configuration for its services in the `/etc/init/` directory, with each service having its own file with a `.config` extension. Feel free to open one of these files to see how `Upstart` services are configured.

Some services are started using `init` scripts stored in `/etc/init.d`, rather than with `Upstart` or `systemd` commands. The `/etc/init.d` directory is used primarily with another `init` system, `sysvinit`, which is very old now by today's standards. However, some distributions that are still supported feature this even older `init` system, such as Debian 7 and Gentoo to name but two. In the early days, Ubuntu used `sysvinit` as well, before eventually replacing it with `Upstart`, so some of these `init` scripts remain, while some applications that have yet to be updated for newer `init` systems still use them. In most cases, `Upstart` and `systemd` can still manage these processes, but will call their `init` scripts in `/etc/init.d` if there is no built-in `Upstart` or `systemd` service file. Even if there is, you are still able to utilize this older method even on newer systems, but there's no telling how long this compatibility layer will remain. At the time of writing, though, the legacy `init` scripts are still well supported.

www.it-ebooks.info

The older methods of starting, stopping, restarting, reloading, and checking the status of services with the older `sysvinit` style is performed with the following commands respectively (again, using `ssh` as an example):

```
/etc/init.d/ssh start
/etc/init.d/ssh stop
/etc/init.d/ssh restart
/etc/init.d/ssh reload
/etc/init.d/ssh status
```

At this point, with multiple init systems, it may be confusing as to which methods you should be using. But it breaks down simply. If you're using a version of Ubuntu that's 15.04 or newer, go with the `systemd` examples, since that is the new init system in use nowadays. If your server is using an Ubuntu version older than 15.04, use the `upstart` commands. In either case, if the normal commands don't work for a particular service, try the older `sysvinit` commands. 99.9% of the time, though, the init system that your version of Ubuntu Server ships with will work since just about any application worth using has been ported over.

The `systemd` init system actually has additional features that its predecessors didn't have. For example, you can enable a service to start as a particular user, rather than system-wide. To do so, the application has to support being started as a user, which would allow you to enable it with a command similar to the following:

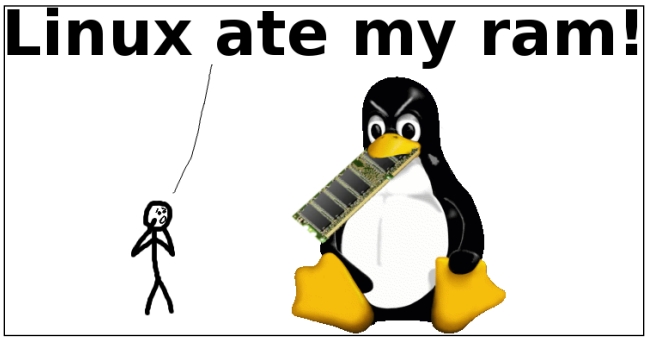
```
# systemctl enable myservice@myuser
```

In the hypothetical example I just provided, I'm enabling a Unit called `myservice` for user `myuser`. A real-life example of why enabling a service to run as a user is useful is `syncthing`. Although an in-depth walk-through of `syncthing` is beyond the scope of this book, it makes a great example since it supports being enabled system-wide as well as per user. This is a great thing, since `syncthing` is a file synchronization application, and each user on a system will have different files to sync, so each user can have his or her own configuration that won't affect other users. As usual, check the documentation that comes with the application you want to install to see what it supports as far as how to enable it.

www.it-ebooks.info

Monitoring memory usage

Understanding how Linux manages memory can actually be a somewhat complex topic, as understanding how much memory is truly free can be a small hurdle for newcomers to overcome. There's even a website dedicated to the topic of Linux RAM usage, which features a rather hilarious leading graphic: `http://www.linuxatemyram.com/`.



The front page of the linuxatemyram.com website

You'll soon see that how Linux manages memory on your server is actually fairly straight forward. For the purpose of monitoring memory usage on our server, we have the **free** command at our disposal, which we can use to see how much memory is being consumed at any given time. My favorite variation of this command is `free -m`, which shows the amount of memory in use in terms of megabytes. You can also use `free -g` to show the output in terms of gigabytes, but the output won't be precise enough on most servers. Giving the **free** command no option will result in the output being shown in terms of kilobytes:

```
jay@ubuntu-sever: ~
jay@ubuntu-sever:~$ free -m
              total        used        free      shared  buff/cache   available
Mem:            740          41          11           1         687         672
Swap:           765           2         763
jay@ubuntu-sever:~$
```

Output of the free -m command on an Ubuntu 16.04 server

To follow along, refer to the previous graphic, and I will explain how to interpret the results of the **free** command.

www.it-ebooks.info

At first glance, it may appear as though this server is in trouble, with only `11` MB free. You'll see this in the first row, third column under **free**. In actuality, the number you'll really want to pay attention to is the number under `available`, which is `672` MB in this case. That's now much memory is actually free. Since this server has `740` MB of total RAM available (you'll see this on the first row, under `total`), this means that most of the RAM is free, and this server is not really working that hard at all.

Some additional explanation is necessary to truly understand these numbers. You could very well stop reading this section right now as long as you take away from it that the `available` column represents how much memory is free for your applications to use. However, it's not quite that simple. Technically, when you look at the output, the server really does have 11 MB free. The amount of memory listed under `available` is legitimately being used by the system in the form of a cache, but would be freed up in the event that any application needed to use it. If an application starts and needs a decent chunk of memory in order to run, the kernel will provide it some memory from this cache if it needed it.

Linux, like most modern systems, subscribes to the belief that "unused RAM is wasted RAM." RAM that isn't being used by any process is given to what is known as a **Disk Cache**, which is utilized to make your server run more efficiently. When data needs to be written to a storage device, it's not directly written right away. Instead, this data is written to the disk cache (a portion of RAM that's set aside), and then synchronized to the storage device later in the background. The reason this makes your server more efficient is that this data being stored in RAM would be written to and retrieved faster than it would be from disk. Applications and services can synchronize data to the disk in the background without forcing you to wait for it. This cache also works for reading data, as when you first open a file, its contents are cached. The system will then retrieve it from RAM if you read the same file again, which would be more efficient. If you just recently saved a new file and retrieve it right away, it's likely still in the cache and then retrieved from there, rather than from the disk directly.

www.it-ebooks.info

To understand all of these columns, I'll outline the meaning of each in the following table:

Column	Meaning
total	The total amount of RAM installed on the server.
used	Memory that is used (from any source). This is calculated as follows: **total - free - buffers - cache**.
free	Memory not being used by anything, cache or otherwise.
shared	Memory used by tmpfs.
buff/cache	The amount of memory being used by the buffers and cache.
available	Memory that is free for applications to use.

Those of you that have been using Linux for a while may notice that the way in which memory is shown has changed in Ubuntu 16.04. In the past, there used to be three rows of information in the output from the `free` command, rather than two, with the second being dedicated to showing you the `buffers` and `cached`. In the new style of output, the `free` command doesn't even show the buffers and cache split into separate values, but rather shows the two combined. The following screenshot shows an example of the older style of output from the `free` command. Quite a few distributions from 2015 onwards will feature the new format:

```
             total       used       free     shared    buffers     cached
Mem:            494        444         50         12         44        315
-/+ buffers/cache:         84        410
Swap:          2047          3       2044
```

Output of the free -m command on an older server

With the older style, to determine how much memory is free, there is no available field as there is with the new output, so you use the second column of the second row as the amount of free memory, which is `410` MB in the screenshot. Other than the difference in the number of rows and the way the numbers are presented, the rest of the information is fairly the same.

In either case, the last row shows us how much `swap` is available and being used, and shows the information in the same way regardless of how old or new your server is and what version of the `free` command is installed. The first value of the final line shows us how much swap is available, the middle shows how much is being used, and the last shows us how much is free. As you probably already know, the lower the amount of swap being used, the better. However, Linux servers will almost always use a small portion of swap, even when free memory isn't really a problem. In the previous screenshot, the server is using 3 MB of swap, which isn't a big deal. If it was using a significant portion of swap, that would be a real problem which we would need to investigate.

www.it-ebooks.info

How frequently a Linux server utilizes swap is referred to as its **swappiness**. By default, the `swappiness` value on a Linux server is typically set to `60`. You can verify this with the following command:

```
cat /proc/sys/vm/swappiness
```

The higher the `swappiness` value, the more likely your server will utilize swap. If the `swappiness` value is set to `100`, your server will use swap as much as possible. If you set it to `0`, swap will never be used at all. This value correlates roughly to the percentage of RAM being used. For example, if you set `swappiness` to `20`, swap will be used when RAM becomes (roughly) 80 percent full. If you set it to `50`, swap will start being used when half your RAM is being used, and so on.

To change this value on the fly, you can execute the following command:

```
# sysctl vm.swappiness=30
```

This method doesn't set `swappiness` permanently, however. When you execute that command, `swappiness` will immediately be set to the new value and your server will act accordingly. Once you reboot, though, the `swappiness` value will revert back to the default. To make the change permanent, open the following file with your text editor:

```
/etc/sysctl.conf
```

A line in that file corresponding to `swappiness` will typically not be included by default, but you can add it manually. To do so, add a line such as the following to the end of the file and save it:

```
vm.swappiness = 30
```

Changing this value is one of many techniques within the realm of performance tuning. While the default value of 60 is probably fine for most, there may be a situation where you're running a performance-minded application and can't afford to have it swap any more than it absolutely has to. In such a situation, you would try different values for `swappiness` and use whichever one works best during your performance tests.

www.it-ebooks.info

Scheduling tasks with Cron

Earlier in this chapter, we worked through starting processes and enabling them to run all the time and as soon as the server boots. In some cases, you may need an application to perform a job at a specific time, rather than always be running in the background. This is where Cron comes in. With Cron, you can set a process, program, or script to run at a specific time—down to the minute. Each user is able to have his or her own set of Cron jobs (known as a `crontab`), which can perform any function that a user would be able to do normally. The root user has a `crontab` as well, which allows system-wide administrative tasks to be performed. Each `crontab` includes a list of Cron jobs (one per line), which we'll get into shortly. To view a `crontab` for a user, we can use the `crontab` command:

```
crontab -l
```

With no options, the `crontab` command will show you a list of jobs for the user that executed the command. If you execute it as root, you'll see root's `crontab`. If you execute it as user `jdoe`, you'll see the `crontab` for `jdoe`, and so on. If you want to view a `crontab` for a user other than yourself, you can use the `-u` option and specify a user, but you'll need to execute it as root or with `sudo` to view the `crontab` of other users:

```
# crontab -u jdoe -l
```

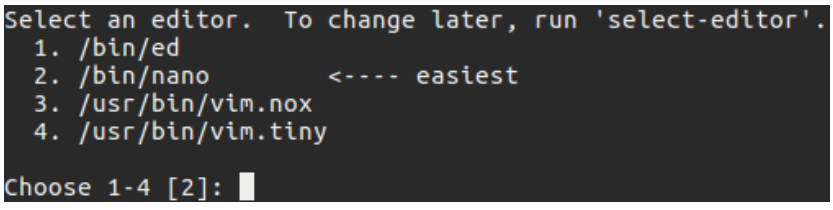
By default, no user has a `crontab` until you create one or more jobs. Therefore, you'll probably see output such as the following when you check the `crontabs` for your current users:

```
no crontab for jdoe
```

To create a Cron job, first log in as the user account you want the task to run under. Then, issue the following command:

```
crontab -e
```

If you have more than one text editor on your system, you may see output similar to the following:



Text editor selection while editing a crontab

www.it-ebooks.info

In this case, you'll simply press the number corresponding to the text editor you'd like to use when creating your Cron job. To choose an editor and edit your `crontab` with a single command; the following will work:

```
EDITOR=vim crontab -e
```

With that example, you can replace `vim` with whatever text editor you prefer. At this point, you should be placed in a text editor with your `crontab` file open. The default `crontab` file for each user features some helpful comments that give you some useful information regarding how Cron works. To add a new job, you would scroll to the bottom of the file (after all the comments), and insert a new line. Formatting is very particular here, and the example comments in the file give you some clue as to how each line is laid out. Specifically, this part:

```
m h  dom mon dow   command
```

Each Cron job has six fields, each separated by at least one space or tab spaces. If you use more than one space, or tab, Cron is smart enough to parse the file properly. In the first field, we have the minute in which we would like the job to occur. In the second field, we place the hour in the 24 hour format, from 0—23. The third field represents the day of the month. For this field, you can place a 5 (5th of the month), 23 (23rd of the month), and so on. The fourth field corresponds to the month, such as 3 for March or 12 for December. The fifth field is the day of the week, numbered from 0—6 to represent Sunday through Saturday. Finally, in the last field, we have the command to be executed. A few example `crontab` lines are as follows:

```
3 0 * * 4 /usr/local/bin/cleanup.sh
* 0 * * * /usr/bin/apt-get update
0 1 1 * * /usr/local/bin/run_report.sh
```

With the first example, the `cleanup.sh` script, located in `/usr/local/bin`, will be run at 12:03 am every Friday. We know this because the **minute** column is set to `3`, the **hour** column is set to `0`, the **day** column is `4` (Friday), and the **command** column shows a fully qualified command of `/usr/local/bin/cleanup.sh`.

A command being fully qualified is important with Cron. For example, in the second example, we could have simply typed `apt-get update` for the command and that would probably work just fine. However, not including the full path to the program is considered bad Cron etiquette. While the command would probably execute just fine, it depends on the application being found in the PATH of the user that's calling it. Not all servers are set up the same, so this might not work depending on how the shell is set up. If you include the full path, the job should run regardless of how the underlying shell is configured.

www.it-ebooks.info

Continuing with the second example, we're running `/usr/bin/apt-get update` to update our server's repository index every morning at midnight. The asterisks on each line refer to "any," so with the minute column being simply `*`, that means that this task is eligible for any minute. Basically, the only field we clarified was the **hour** field, which we set to `0` in order to represent 12:00 am.

With the third example, we're running the `/usr/local/bin/run_report.sh` script on the first day of every month at 1:00 am. If you notice, we set the third column (**day of month**) to `1`, which is the same as February 1st, March 1st, and so on. This job will be run if it's the first day of the month, but only if the current time is also 1:00 am, since we filled in the first and second column, which represent the minute and hour respectively.

Once you finish editing a user's `crontab` and save it, Cron is updated and from that point forward will execute the task at the time you selected. The `crontab` will be executed according to the current time and date on your server, so you would want to make sure that that is correct as well, otherwise you'll have your jobs execute at unexpected times. You can view the current date and time on your server by simply issuing the `date` command.

To get the hang of creating jobs with Cron, the best way (as always) is to practice. The second example Cron job is probably a good one to experiment with, as updating your repository index isn't going to hurt anything.

Understanding load average

Before we close this chapter, a very important topic to understand when monitoring processes and performance is **Load Average**, which is a series of numbers that represents your server's trend in CPU utilization over a given time. You've probably already seen these series of numbers before, as there are several places in which the load average appears. If you run the `htop` or `top` command, the load average is shown within the output of each. In addition, if you execute the `uptime` command, you can see the load average in the output of that command as well. You can also view your load average by viewing the text file that stores it in the first place:

```
cat /proc/loadavg
```

The load average is broken down into three sections, each representing one minute, five minutes, and fifteen minutes respectively. A typical load average may look something like the following:

```
0.36, 0.29, 0.31
```


www.it-ebooks.info

In this example, we have a load average of `0.36` in the one minute section, `0.29` in the five minute section, and `0.31` in the fifteen minute section. In particular, each number represents how many tasks were waiting on attention from the CPU for that given time period. Therefore, these numbers are really good. The server isn't that busy, since virtually no task is waiting on the CPU at any one moment. This is contrary to something such as overall CPU percentages, which you may have seen in task managers of other platforms. While viewing your CPU usage percentage can be useful, the problem with this is that your CPUs will constantly go from a high percent of usage to a low percent of usage, which you can see for yourself by just running `htop` for a while. When a task does some sort of processing, you might see your cores shoot up to 100 percent and then right back down to a lower number. That really doesn't tell you much, though. With load averages, you're seeing the trend of usage over three given time frames, which is more accurate in determining if your server's CPUs are running efficiently or are choking on a workload they just can't handle.

The main question, though, is when you should be worried, which really depends on what kinds of CPU are installed in your server. Your server will have one or more processors, each with one or more cores. To Linux, each of these cores, whether they are physical or virtual, are the same thing (a CPU). In my case, the machine I took the earlier output from has a CPU with four cores. The more CPUs your server has, the more tasks it's able to handle at any given time and the more flexibility you have with the load average.

When a load average for a particular time period is equal to the number of CPUs on the system, that means your server is at capacity. It's handling a consistent number of tasks that are equal to the number of tasks it can handle. If your load average is consistently more than the number of cores you have available, that's when you'd probably want to look into the situation. It's fine for your server to be at capacity every now and then, but if it always is, that's a cause for alarm.

I'd hate to use a cliché example in order to fully illustrate this concept, but I can't resist, so here goes. A load average on a Linux server is equivalent to the check-out area at a supermarket. A supermarket will have several registers open, where customers can pay to finalize their purchases and move along. Each cashier is only able to handle one customer at a time. If there are more customers waiting to check out then there are cashiers, the lines will start to back up and customers will get frustrated. In a situation where there are four cashiers and four customers being helped at a time, the cashiers would be at capacity, which is not really a big deal since no one is waiting. What can add to this problem is a customer that is paying by check and/or using a few dozen coupons, which makes the check-out process much longer (similar to a resource-intensive process).

www.it-ebooks.info

Just like the cashiers, a CPU can only handle one task at a time, with some tasks hogging the CPU longer than others. If there are exactly as many tasks as there are CPUs, there's no cause for concern. But if the lines start to back up, we may want to investigate what is taking so long. To take action, we may hire an additional cashier (add a new CPU) or ask a disgruntled customer to leave (kill a process).

Let's take a look at another example load average:

```
1.87, 1.53, 1.22
```

In this situation, we shouldn't be concerned, because this server has four CPUs, and none of them have been at capacity within the one, five, or fifteen minute time periods. Even though the load is consistently higher than 1, we have CPU resources to spare, so it's no big deal. Going back to our supermarket example, this is equivalent to having four cashiers with an average of almost two customers being assisted during any one minute. If this server only had one CPU, we would probably want to figure out what's causing the line to begin to back up.

It's normal for a server to always have a workload (so long as it's lower than the number of CPUs available), since that just means that our server is being utilized, which is the whole point of having a server to begin with (servers exist to do work). While typically, the lower the load average the better, depending on the context, it might actually be a cause for alarm if the load is too low. If your server's load average drops to an average of zero-something, that might mean that a service that would normally be running all the time has failed and exited. For example, if you have a database server that constantly has a load within the 1.x range that suddenly drops to 0.x, that might mean that you either have legitimately less traffic or the database server service is no longer running. This is why it's always a good idea to develop baselines for your server, in order to gauge what is normal and what isn't.

Overall, load averages are something you'll become very familiar with as a Linux administrator if you haven't already. As a snapshot in time of how heavily utilized your server is, it will help you to understand when your server is running efficiently and when it's having trouble. If a server is having trouble keeping up with the workload you've given it, it may be time to consider increasing the number of cores (if you can) or scaling out the workload to additional servers. When troubleshooting utilization, planning for upgrades, or designing a cluster, the process always starts with understanding your server's load average so you can plan your infrastructure to run efficiently for its designated purpose.

www.it-ebooks.info

Summary

In this chapter, we learned how to manage processes and monitor our server's resource usage. We began with a look at the `ps` command, which we can use to view a list of processes that are currently running. We also took a look at managing jobs, as well as killing processes that for one reason or another are misbehaving. We took a look at `htop`, which is a very handy utility for viewing an overview of our resource utilization. In addition, we learned how to monitor memory usage, schedule jobs with Cron, and gained an understanding of load average.

In *Chapter 7*, *Managing Your Ubuntu Server Network*, we'll dive back into networking, where we'll learn how to set up our own DHCP and DNS servers for our network, and more. See you there!

www.it-ebooks.info

www.it-ebooks.info

7
Managing Your Ubuntu Server Network

In *Chapter 4*, *Connecting to Networks*, we discussed connecting networks. We saw how to set the hostname, manage network interfaces, configure connections, use Network Manager, and more. In this chapter, we'll revisit networking, specifically to set up the resources that will serve as the foundation of our network. The majority of this chapter will focus on setting up the DHCP and DNS servers, which are very important components of any network. In addition, we'll also set up a **Network Time Protocol** (**NTP**) server to keep our clocks synchronized. We'll even take a look at setting up a server to act as an Internet gateway for the rest of our network.

Along the way, we'll cover the following topics:

- Planning your IP address scheme
- Serving IP addresses with `isc-dhcp-server`
- Creating a secondary DNS server
- Setting up an Internet gateway
- Keeping your system clock in sync with NTP

Planning your IP address scheme

The first step in rolling out any solution is to plan it properly. Planning out your network layout is one of the most important decisions you'll ever make in your organization. Even if as an administrator you're not responsible for the layout and just go along with what your network administrator provides, understanding this layout and being able to deploy your solutions to fit within it is also very important.

www.it-ebooks.info

Planning an IP address scheme involves predicting how many devices will need to connect to your network and being able to support them. In addition, a good plan will account for potential growth and allow expansion as well. The main thing that factors into this is the size of your user base. Perhaps you're a small office with only a handful of people, or a large corporation with thousands of users and hundreds of virtual machines. Even if you're only a small office, there's always room for growth if your organization is doing well, which is another thing to take into consideration.

Typically, most off-the-shelf routers and network equipment come with an integrated **Dynamic Host Control Protocol** (**DHCP**) server, with a default **Class C** (`/24`) network. Essentially, this means that if you do not perform any configuration at all, you're limited to 254 addresses. For a small office, this may seem like plenty. After all, if you don't even have 254 users on your network, you may think that you're all set. As I mentioned before, potential growth is always something to keep in mind. But even if we remove that from the equation, IP addresses are used up quite quickly nowadays—even when it comes to internal addressing. An average user may consume three IP addresses each, and sometimes more. For example, perhaps a user not only has a laptop (which itself can have both a wired and wireless interface, both consuming an IP), but perhaps they also have a mobile phone (which likely features Wi-Fi), and a **Voice over IP** (**VoIP**) phone (there goes another address). If that user somehow managed to convince their supervisor that they also need a desktop computer as well as their laptop, there will be a total of five IP addresses for that one user. Suddenly, 254 addresses doesn't seem like all that many.

The obvious answer to this problem is splitting up your network into **subnets**. Although I won't go into the details of how to subnet your network (this book is primarily about servers and not a course on network administration), I mentioned it here because it's definitely something you should take into consideration. In the next section, I'll explain how to set up your own DHCP server with a single network. However, if you need to expand your address space, you can easily do so by updating your DHCP configuration. When coming up with an IP address layout, always assume the worst and plan ahead. While it may be a cinch to expand your DHCP server, planning a new IP scheme roll-out is very time consuming, and to be honest, annoying.

www.it-ebooks.info

When I set up a new network, I like to divide the address space into several categories. First, I'll usually set aside a group of IP addresses specifically for DHCP. These addresses will get assigned to clients as they connect, and I'll usually have them expire and need to be renewed in about 1 day. Then, I'll set aside a block of IP addresses for network appliances, another block for servers, and so on. In the case of a typical 24-bit network, I might decide on a scheme like the following (assuming it's a small office with no growth planned):

```
Network: 192.168.1.0/24
Network equipment: 192.168.1.1 - 192.168.1.10
Servers: 192.168.1.11 - 192.168.1.99
DHCP: 192.168.1.100 - 192.168.1.240
Reservations: 192.168.1.241 - 192.168.1.254
```

Of course, no IP address scheme is right for everyone. The one I provided is simply a hypothetical example, so you shouldn't copy mine and use it on your network unless it matches your needs. I'll use this scheme for the remainder of this chapter, since it works fine as an example. To explain my sample roll-out, we start off with a 24-bit network, `192.168.1.0`. If you're accustomed to the classful model, this is the same as a Class C network. The address `192.168.1.0` refers to the network itself, and that IP is not assignable to clients. In fact, the last IP address in this block (`192.168.1.255`) is not assignable either, since that is known as the **Broadcast Address**. Anything that's sent to the broadcast address is effectively sent to every IP in the block, so we can't really use it for anything but broadcasts.

Next, I set aside a group of IP addresses starting with `192.168.1.1` through `192.168.1.10` for use by network appliances. Typical devices that would fit into this category would be managed switches, routers, wireless access points, and so on. These devices typically have an integrated web console for remote management, so it would be best to have a static IP address assignment. That way, I'll have an IP address available which I can use to access these devices. I like to set these up as the first devices so that they all get the lowest numbers when it comes to the last number (octet) of each IP. This is just personal preference.

Next, we have a block of IP addresses for servers, `192.168.1.11` through `192.168.1.99`. This may seem like quite a few addresses for servers, and it is. However, with the rise of virtualization and how simple it has become to spin up a server, this block could get used up faster than you think. It's up to you whether or not you need 88 addresses here, as I put in my example layout, so feel free to adjust accordingly.

www.it-ebooks.info

Now we have our DHCP pool, which consists of addresses `192.168.1.101` through `192.168.1.240`. These IP addresses are assignable to any devices that connect to our network. As I mentioned, I typically have these assignments expire in 1 day to prevent one-off devices from claiming and holding onto an IP address for too long, which can lead to devices fighting over a DHCP lease. In this situation, you'd have to clear your DHCP leases to reset everything, and I find that to be too much of a hassle. When we get to the section on setting up a DHCP server, I'll show you how to set the expiration time.

Finally, we have addresses `192.168.1.241` through `192.168.1.254` for the purposes of DHCP reservations. I call these "static leases", but both terms mean the same thing. These addresses will be assigned by DHCP, but each device with a static lease will be given the same IP address each time. You don't have to separate these into their own pool, since DHCP will not assign the same address twice. It may be a good idea to separate them, if only to be able to tell from looking at an IP address that it's a static lease, due to it being within a particular block. Static leases are good for devices that aren't necessarily a server, but still need a predictable IP address. An example of this may be an administrator's desktop PC. Perhaps they want to be able to connect to the office via VPN, and be able to easily find their computer on the network and connect to it. If the IP was dynamically assigned instead of statically assigned, it would be harder for them to find it.

After you carve up your IP addresses, the next thing is creating a spreadsheet to keep track of your static IP assignments. It doesn't have to be anything fancy, but it will certainly help you later. Ideally, your IP layout and the devices that populate it would be best placed within an internal wiki or knowledge-base solution that you may already be using. But if you don't have anything like that set up yet, a spreadsheet can serve a similar purpose. In my example IP spreadsheet, I include a designation of **(R)** if the IP address is a reservation or **(S)** if the IP address is a manually assigned static address. I also include the Mac address of each device, which will come in handy when we set up our DHCP server in the next section:

Servers			
Machine Name	**IP**	**Mac Address 1**	**Model**
D-Link AirPremier DAP-2695	10.10.96.2 (S)	NA	DAP-2695
galaxy	10.10.96.4 (R)	E0:3B:49:6E:6C:8E	Ratel Performance
nagios	10.10.96.3 (R)	52:54:01:88:F8:BC	KVM VM
hermes	10.10.96.1 (S)	00:23:4D:A5:F2:EB	Intel Atom
iris	10.10.96.5 (R)	52:54:01:D5:5D:0C	KVM VM
moogle	10.10.96.102 (R)	52:54:00:D4:3E:87	KVM VM
pi-dev	10.10.96.8 (R)	B8:27:EB:E2:29:03	Raspberry Pi 2
pluto	10.10.96.10 (R)	D0:51:99:37:A9:0D	custom i3 build
upsmon	10.10.96.7 (R)	B8:27:EF:BB:91:2D	Raspberry Pi 2

An example IP address layout spreadsheet

www.it-ebooks.info

Although subnetting is beyond the scope of this book, it's definitely something you should look into if you're not already familiar with it. As you can see from my example layout, our number of available addresses is rather limited with a 24-bit network. However, this layout will serve as an example we can follow that's good enough for the remainder of the chapter.

Serving IP addresses with isc-dhcp-server

While most network appliances you purchase nowadays often come with their own DHCP server, rolling your own gives you ultimate flexibility. Some of these built-in DHCP servers are full-featured and come with everything you need, while others may contain just enough features for it to function, but nothing truly exciting. Ubuntu servers make great DHCP servers, and rolling your own server is actually very easy to do.

First, the server that serves DHCP will need a static IP address. This means you'll need to configure the `/etc/network/interfaces` file with a manual IP assignment, with an IP address that no other device is using.

If you have yet to set a static IP, *Chapter 4*, *Connecting to Networks*, has a section that will walk you through the process.

Once you assign a static IP address, the next step is to install the `isc-dhcp-server` package:

```
# apt-get install isc-dhcp-server
```

Depending on your configuration, the `isc-dhcp-server` service may have started automatically, or your server may have attempted to start it and failed to do so with an error. You can check the status of the daemon with the following command:

```
systemctl status isc-dhcp-server
```

The output will either show that the service failed or is running. If the service failed to start, that's perfectly fine—we haven't even configured it yet. If it's running, then you need to stop it for now, since it's not a good idea to leave an unconfigured DHCP server running on your network. It might conflict with your existing one:

```
# systemctl stop isc-dhcp-server
```


www.it-ebooks.info

On older servers (before `systemd` was introduced), we can stop the service with the following command instead:

```
# /etc/init.d/isc-dhcp-server stop
```

Now that you've installed the `isc-dhcp-server` package, you'll have a default configuration file for it at `/etc/dhcp/dhcpd.conf`. This file will contain some default configuration, with some example settings that are commented out. Feel free to take a look at this file to get an idea of some of the settings you can configure. We'll create our own `dhcpd.conf` file from scratch. So when you're done looking at it, copy the existing file to a new name:

```
# mv /etc/dhcp/dhcpd.conf /etc/dhcp/dhcpd.conf.orig
```

Now, we're ready to create our own `dhcpd.conf` file. Open `/etc/dhcp/dhcpd.conf` in your preferred text editor. Since the file no longer exists (we moved it), we should start with an empty file. Here's an example `dhcpd.conf` file that I will explain so that you understand how it works:

```
default-lease-time 86400;
max-lease-time 86400;
option subnet-mask 255.255.255.0;
option broadcast-address 192.168.1.255;
option domain-name "local.lan";
authoritative;
subnet 192.168.1.0 netmask 255.255.255.0 {
    range 192.168.1.100 192.168.1.240;
    option routers 192.168.1.1;
    option domain-name-servers 192.168.1.1;
}
```

As always, change the values I've used with those that match your network. I'll explain each line so that you'll understand how it affects the configuration of your DHCP server.

```
default-lease-time 43200;
```

When a device connects to your network and requests an IP address, the expiration of the lease will be set to the number of seconds in `default-lease-time` if the device doesn't explicitly ask for a longer lease time. Here, I'm setting that to `43200` seconds, which is equivalent to half a day. This basically means that the device will need to renew its IP address every `43200` seconds, unless it asks for a longer duration.

```
max-lease-time 86400;
```


www.it-ebooks.info

While the previous setting dictated the default lease time for devices that don't ask for a specific lease time, the `max-lease-time` is the most that the device is allowed to have. In this case, I set this to one day (`86400` seconds). Therefore, no device that receives an IP address from this DHCP server is allowed to hold onto their lease for longer than this without first renewing it.

```
option subnet-mask 255.255.255.0;
```

With this setting, we're informing clients that their subnet mask should be set to `255.255.255.0`, which is for a default 24-bit network. If you plan to subnet your network, you'll put in a different value here. `255.255.255.0` is fine if all you need is a 24-bit network.

```
option broadcast-address 192.168.1.255;
```

With this setting, we're informing the client to use `192.168.1.255` as the broadcast address, which is typically the last address in the subnet and cannot be assigned to a host.

```
option domain-name "local.lan";
```

Here, we're setting the domain name of all hosts that connect to the server to include `local.lan`. Feel free to change this to the domain name of your organization, or you can leave it as is if you don't have one.

```
authoritative;
```

With the `authoritative;` setting (the opposite is `non authoritative;`), we're declaring our DHCP server as authoritative to our network. Unless you are planning to have multiple DHCP servers for multiple networks (which is rare), the `authoritative;` option should be included in your `config` file.

Now, we get to the most important part of our configuration file for DHCP. The following block details the specific information that will be provided to clients:

```
subnet 192.168.1.0 netmask 255.255.255.0 {
    range 192.168.1.100 192.168.1.240;
    option routers 192.168.1.1;
    option domain-name-servers 192.168.1.1;
}
```


www.it-ebooks.info

This block is probably self-explanatory, but we're basically declaring our pool of addresses for the `192.168.1.0` network. We're declaring a range of IPs from `192.168.1.100` through `192.168.1.240` to be available from clients. Now when our DHCP server provides an address to clients, it will choose one from this pool. We're also providing a default gateway (option routers) and DNS server of `192.168.1.1`. This is assuming that your router and local DNS server are both listed at that address, so make sure that you change it accordingly. Otherwise, anyone who receives a DHCP lease from your server will not be able to connect to anything. For the address pool (range), feel free to expand it or shrink it accordingly. For example, you might need more addresses than the 140 that are allowed in my sample range, so you may change it to something like `192.168.1.50` through `192.168.1.250`. Feel free to experiment.

If you have more than one network interface on your server, you'll want to designate one interface as the one that `isc-dhcp-server` should listen on for DHCP requests. You can do that by editing the `/etc/default/isc-dhcp-server` file, where you'll see a line toward the bottom similar to the following:

```
INTERFACES=""
```

If you only have one interface on your server, you don't need to do anything to this file or to that line (despite what several articles online may tell you). If you do have more than one, you can simply type the name of the interface within the quotes, that way your DHCP server won't get confused as to which interface it should be listening on:

```
INTERFACES="enp0s3"
```

In case you forgot, the command to list the details of the interfaces on your server is `ip addr`, or the shortened version, `ip a`.

Now that we have our DHCP server configured, we should be able to start it:

```
# systemctl start isc-dhcp-server
```

On older Ubuntu servers with `upstart` rather than `systemd`, we can use the following command to start it instead:

```
# /etc/init.disc-dhcp-server start
```

Next, double-check that there were no errors:

```
# systemctl status isc-dhcp-server
```

On older servers, the following should give us the status:

```
# /etc/init.d/isc-dhcp-server status
```


www.it-ebooks.info

Assuming all went well, your DHCP server should be running. When an IP lease is assigned to a client, it will be recorded in `/var/lib/dhcp/dhcpd.leases` file. While your DHCP server runs, it will also record information to your server's system log, located at `/var/log/syslog`. To see your DHCP server function in all its glory, you can follow the log as it gets written to with the following:

```
# tail -f /var/log/syslog
```

The `-f` flag of the `tail` command is indispensable, and it is something you'll likely use quite often as a server administrator. With the `-f` option, you'll watch the log as it gets written to, rather than needing to refresh it manually.

While your DHCP server runs, you'll see notices appear within the `syslog` file whenever a DHCP request was received and when a lease is offered to a client. A typical DHCP request will appear in the log similar to the following (the name of my DHCP server is `hermes`, in case you were wondering):

```
May  5 22:07:36 hermes dhcpd: DHCPDISCOVER from 52:54:00:88:f8:bc via
enp0s3
May  5 22:07:36 hermes dhcpd: DHCPOFFER on 192.168.1.103 to
51:52:01:87:f7:bc via enp0s3
```

Active and previous DHCP leases are stored in the `/var/lib/dhcp/dhcpd.leases` file, and a typical lease entry in that file would look similar to the following:

```
lease 192.168.1.138 {
  starts 0 2016/05/06 16:37:30;
  ends 0 2016/05/06 16:42:30;
  cltt 0 2016/05/06 16:37:30;
  binding state active;
  next binding state free;
  rewind binding state free;
  hardware ethernet 32:6e:92:01:1f:7f;
}
```

When a new device is added to your network and receives an IP address from your new DHCP server, you should see the lease information populate into that file. This file can be incredibly helpful, because whenever you connect a new device, you won't have to interrogate the device itself to find out what its IP address is. You can just check the `/var/lib/dhcp/dhcpd.leases` file. If the device advertises its hostname, you'll see it within its lease entry. A good example of how this can be useful is connecting a Raspberry Pi to your network. Once you plug it in and turn it on, you'll see its IP address in the `dhcpcd.leases` file, and then you can connect to it via SSH without having to plug in a monitor to it. Similarly, you can view the temporary IP address of a new network appliance you plug in so that you can connect to it and configure it.

www.it-ebooks.info

If you have any trouble setting up the `isc-dhcp-server` daemon, double-check that you have set all the correct and matching values within your static IP assignment (the `/etc/network/interfaces` file), as well as within your `/etc/dhcp/dhcpd.conf` file. For example, your server must be within the same network as the IPs you're assigning to clients. The error messages that the `isc-dhcp-server` service provides can be very misleading. In my tests, I've seen the service complain that it's not configured to listen on any address, when the actual problem was the server having an IP address that was not within the same scope as the pool addresses. As long as everything matches, you should be fine and it should start properly.

Setting up name resolution (DNS) with bind

I'm sure most of you are familiar with the purpose of a **Domain Name System** (**DNS**) server. Its simplest definition is that it's a service that's responsible for matching an IP address to a hostname. When you connect to the Internet, name-to-IP matching happens constantly as you browse. After all, it's much easier to connect to `https://www.google.com/` with its domain name, than it is to remember its IP address. When you connect to the Internet, your workstation or server will connect to an external DNS server in order to figure out the IP addresses for the websites you attempt to visit.

It's also very common to run a local DNS server, as well. The benefit is that you'll be able to resolve your local hostnames as well, something that an external DNS server would know nothing about. For example, if you want to connect to your mail server, it would be much easier to connect to it by a name (such as `mail.mydomain.local`) than it would be to remember its IP address. With a local DNS server, you would create what is known as a **Zone File**, which would contain information regarding the hosts and IP address in use within your network so that local devices would be able to resolve them. In the event that your local DNS server is unable to fulfill your request (such as a request for an external website), the server would pass the request along to an external DNS server, which would then carry out the request.

A detailed discussion of DNS, how it functions, and how to manage it is outside the scope of this book. However, a basic understanding is really all you need in order to make use of a DNS server within your network. In this section, I'll show you how to set up your very own DNS server to allow your devices to resolve local hostnames, which should greatly enhance your network.

First, we'll need to install the **Berkeley Internet Name Daemon** (**BIND**) package on our server:

```
# apt-get install bind9
```


www.it-ebooks.info

Then, we can start it:

```
# systemctl start bind9
```

On older servers, we can start `bind9` with the following command instead:

```
# /etc/init.d/bind9 start
```

At this point, we have the `bind9` service running on our server, though it's not actually configured to do much at this point. The most basic function of `bind` is to act as what's called a **Caching Name Server**, which means that the server doesn't actually match any names itself. Instead, it caches responses from an external server. We'll configure `bind` with actual hosts later, but setting up a caching name server is a good way to get started.

To do so, open the `/etc/bind/named.conf.options` file in your favorite text editor.

Within the file, you should see a block of text that looks similar to the following:

```
// forwarders {
//      0.0.0.0;
// };
```

Uncomment these lines. The forward slashes are the comment marks as far as this configuration file is concerned, so remove them. Then, we can add a few external DNS server IP addresses. For these, you can use the IP addresses for your ISP's DNS servers, or you could simply use Google's DNS servers (`8.8.8.8` and `8.8.4.4`) instead:

```
    forwarders {
    8.8.8.8;
    8.8.4.4;
};
```

After you save the file, restart the `bind9` service:

- With `systemd`:

  ```
  # systemctl restart bind9
  ```

- For older servers with `upstart`:

  ```
  # /etc/init.d/bind9 restart
  ```


www.it-ebooks.info

As long as you've entered in everything correctly, you should now have a working DNS server. Of course it isn't resolving anything, but we'll get to that. Now, all you should need to do is configure other devices on your network to use your new DNS server. The easiest way to do this is to reconfigure the `isc-dhcp-server` service we set up in the previous section. Remember the section that designates a pool of addresses from the server to the clients? It also contained a section to declare the DNS server your clients will use as well. Here's that section again, with the relevant lines in bold:

```
subnet 192.168.1.0 netmask 255.255.255.0 {
range 192.168.1.100 192.168.1.240;
option routers 192.168.1.1;
option domain-name-servers 192.168.1.1;
}
```

To configure the devices on your network to use your new DNS server, all you should need to do is change the configuration `option domain-name-servers 192.168.1.1;` to point to the IP address of your new server. When clients request a DHCP lease (or attempt to renew an existing lease), they will be configured with the new DNS server automatically. If you don't want to wait, you can simply edit the `/etc/resolv.conf` file on a host to point it to your new DNS server. If you need a refresher, we worked through that in *Chapter 4*, *Connecting to Networks*.

With the caching name server we just set up, hosts that utilize it will check it first for any hosts they attempt to look up. If they look up a website or host that is not within your local network, their requests will be forwarded to the forwarding addresses you configured for `bind`. In my example, I used Google's DNS servers, so if you used my configuration your hosts will first check your local server and then check Google's servers when resolving external names. Depending on your network hardware and configuration, you might even see a slight performance boost. This is because the DNS server you just set up is caching any lookups done against it. For example, if a client looks up `https://www.packtpub.com` in a web browser, your DNS server will forward the request along since that site doesn't exist locally and it will also remember the result. The next time a client within your network looks up that site, the response will be much quicker because your DNS server cached it.

To see this yourself, execute the following command twice on a node that is utilizing your new DNS server:

```
dig www.packtpub.com
```

In the response, look for a line toward the end that gives you your query time. It will look similar to the following:

```
;; Query time: 98 msec
```


www.it-ebooks.info

When you run it again, the query time should be much lower:

```
;; Query time: 1 msec
```

This is your caching name server in action! Even though we haven't even set up any zone files to resolve your internal servers, your DNS server is already adding value to your network. You just laid the groundwork we'll use for the rest of our configuration.

Now, let's add some hosts to our DNS server so we can start fully utilizing it. The configuration file for `bind` is located at `/etc/bind/named.conf`. In addition to some commented lines, it will have the following three lines of configuration within it:

```
include "/etc/bind/named.conf.options";
include "/etc/bind/named.conf.local";
include "/etc/bind/named.conf.default-zones";
```

As you can see, the default `bind` configuration is split among several configuration files. Here, it includes three others: `named.conf.options`, `named.conf.local`, and `named.conf.default-zones` (the first of which we already took care of editing). In order to resolve local names, we need to create what is known as a **zone File**, which is essentially a text file that includes some configuration, a list of hosts, and their IP addresses. In order to do this, we need to tell `bind` where to find the zone file we're about to create. Within `/etc/bind/named.conf.local`, we need to add a block of code like the following to the end of the file:

```
zone "local.lan" IN {
    type master;
    file "/etc/bind/net.local.lan";
};
```

Notice that the zone is named `local.lan`, which is the same name I gave our domain in our DHCP server configuration. It's best to keep everything consistent when we can. If you use a different domain name than the one I used in my example, make sure that it matches here as well. Within the block, we're creating a master zone file and informing bind that it can find a file named `net.local.lan`, stored in the `/etc/bind` directory. This should be the only change we'll need to make to the `named.conf.local` file, we'll only create a single zone file (for the purpose of this section). Once you save this file, we'll need to create our `/etc/bind/net.local.lan` file. So go ahead and open that file in a text editor. Since we haven't created it yet, it should be blank. Here's an example of this zone file, completely filled out with some sample configuration:

```
$TTL 1D
@ IN SOA local.lan. hostmaster.local.lan. (
```


www.it-ebooks.info

```
201608161 ; serial

8H ; refresh
4H ; retry
4W ; expire
1D ) ; minimum
IN A 192.168.1.1
;
@ IN NS hermes.local.lan.
fileserv        IN  A   192.168.1.3
mailserv        IN  A   192.168.1.5
mail            IN  CNAME   mailserv.
web01           IN  A   192.168.1.7
```

Feel free to edit this file to match your configuration. You can edit the list of hosts at the end of the file to match your hosts within your network, as the ones I included are merely examples. You should also ensure that the file matches the IP scheme for your network. Next, I'll go over each line in order to give you a deeper understanding of what each line of this configuration file is responsible for.

```
$TTL 1D
```

The **Time to Live** (**TTL**) determines how long a record may be cached within a DNS server. If you recall from earlier where we practiced with the `dig` command, you saw that the second time you queried a domain with `dig`, the query time was less than the first time you ran the command. This is because your DNS server cached the result, but it won't hold onto it forever. At some point, the lookup will expire. The next time you look up that same domain after cached result expired, your server will go out and fetch the result from the DNS server again. In my examples, I used Google's DNS servers. That means at some point, your server will query those servers again once the record times out.

```
@ IN SOA local.lan. hostmaster.local.lan. (
```

With the **Start of Authority** (**SOA**) line, we're establishing that our DNS server is authoritative over the `local.lan` domain. We also set `hostmaster@local.lan` as the e-mail address of the responsible party for this server, but we enter it here in a different format for `bind` (`hostmaster.local.lan`). This is obviously a fake address, but for the purposes of an internal DNS server, that's not an issue we'll need to worry about.

```
201608161 ; serial
```


www.it-ebooks.info

Of all the lines of configuration within a zone file, the `serial` is by far the one that will frustrate us the most. This is because it's not enough to simply update the zone file any time we make a change to it (change an IP address, add or remove a host, and so on); we also need to remember to increase the serial number by at least one. If we don't, `bind` won't be aware that we've made any changes, as it will look at the serial before the rest of the file. The problem with this is that you and I are both humans, and we're prone to forgetting things. I've forgotten to update the serial many times and became frustrated when the DNS server refuses to resolve new hosts that were recently added. Therefore, it's very important for you to remember that any time you make a change to any zone file, you'll need to also increment the serial. The format doesn't really matter; I used `201608161`, which is simply the year, two-digit month, two-digit day, and an extra number to cover us if we make more than one change in a day (which can sometimes happen). As long as you increment the serial by one every time you modify your zone file, you'll be in good shape—regardless of what format you use. However, the sample format I gave here is actually quite common in the field.

```
8H ; refresh
4H ; retry
4W ; expire
1D ) ; minimum
```

These values control how often slave DNS servers will be instructed to check in for updates. With the refresh value, we're instructing any slave DNS servers we may have to check in every 8 hours to see whether or not the zone records were updated. The `retry` field dictates how long the slave will wait to check in, in case there was an error doing so the last time. The last two options in this section, `expire` and `minimum`, set the minimum and maximum age of the zone file, respectively. As I mentioned though, a full discussion of DNS with `bind` could constitute be an entire book on its own. For now, I would just use these values until you have a reason to need to experiment.

```
IN A 192.168.1.1
@ IN NS hermes.local.lan.
```

Here, we identify the name server itself. In my case, the server is called `hermes` and it's located at `192.168.1.1`.

www.it-ebooks.info

Next, in our file we'll have several host entries to allow our resources to be resolved on our network by name. In my example, I have three hosts: `fileserv`, `mailserv`, and `web01`. In the example, these are all address records, which means that any time our server is asked to resolve one of these names, it will respond with the corresponding IP address. If our DNS server is set as a machine's primary DNS server, it will respond with `192.168.1.3` when asked for `fileserv`, and `192.168.1.7` when asked for `web01`. The entry for mail is special as it is not an address record, but instead a **Canonical Name** (**CNAME**) record. In this case, it just points back to `mailserv`. Essentially, that's what a `CNAME` record does: it creates a pointer of sorts to another resource. In this case, if someone tries to access a server named `mail`, we redirect them to the actual server `mailserv`. Notice that on the `CNAME` record, we're not inputting an IP address, but instead the hostname of the original resource:

```
fileserv        IN  A   192.168.1.3
mailserv        IN  A   192.168.1.5
mail            IN  CNAME   mailserv.
web01           IN  A   192.168.1.7
```

Now that we have a zone file in place, we should be able to start using it. First, we'll need to restart the `bind9` service:

```
# systemctl restart bind9
```

After the command finishes, check to see if there are any errors:

```
# systemctl status bind9
```

If there are any problems, you should see specific information in the output that should point you in the right direction. If not, you can also check the system log for clues regarding `bind` as well:

```
# cat /var/log/syslog | grep bind9
```

The most common mistake made is not being consistent within the file. For example, if you're using a different IP scheme (such as `10.10.10.0/24`), you'll want to make sure you didn't forget to replace any of my example IP addresses with the proper scheme. Assuming that everything went smoothly, you should be able to point devices on your network to use this new DNS server. Make sure you test not only pinging devices local to your network, but outside resources as well, such as websites. If the DNS server is working properly, it should resolve your local names, and then forward your requests to your external DNS servers (the two we set as forwarders), if it doesn't find what you're looking for locally. In addition, you'll also want to make sure that port `53` is open in your network's firewall, which is the port that DNS uses. It's extremely rare that this would be an issue, but I have seen it happen.

www.it-ebooks.info

To further test our DNS server, we can use the `dig` command, as we did before while we were experimenting with caching. Try `dig` against a local and external resource. For example, you could try `dig` against a URL as well as a local server:

```
dig webserv.local.lan
dig www.packtpub.com
```

You should see a response similar to the following:

```
;; Query time: 76 msec
;; SERVER: 192.168.1.1#53(192.168.1.1)
;; WHEN: Sun Mar 06 12:02:24 EST 2016
;; MSG SIZE  rcvd: 283
```

Notice the second line, beginning with `SERVER`. It should be showing the IP address of the local DNS server we just set up.

One thing that may confuse the output of the `dig` command is local resolvers, which are becoming more and more common. Several Linux distributions are including local DNS resolvers, which is essentially a local DNS server built into the distribution to handle caching, which also benefits security. You may see the following output instead:

```
;; SERVER: 127.0.1.1#53(127.0.1.1)
```

In this case, we have a local resolver (the IP address `127.0.1.1` points to the local machine). If this happens, there is nothing wrong with your DNS server. Local resolvers are becoming very common with some distributions, with Fedora being one example. If you do have a local resolver in place, consult the documentation for your distribution for how to temporarily disable or bypass it for purposes of our tests.

Creating a secondary DNS server

Depending on just one server to provide a resource to your network is almost never a good idea. If our DNS server has a problem and fails, our network users will be unable to resolve any names, internal or external. To rectify this, we can actually set up a slave DNS server that will cache zone records from the master, and allow name resolution to work in case the primary fails. This is not required, but redundancy is always a good thing.

www.it-ebooks.info

To set up a secondary DNS server, we first need to configure our primary server to allow it to transfer zone records to a slave server. To do so, we'll need to edit the `/etc/bind/named.conf.options` file, which currently looks similar to the following:

```
options {
        directory "/var/cache/bind";
        forwarders {
                8.8.8.8;
                8.8.4.4;
        };
        dnssec-validation auto;

        auth-nxdomain no;
        listen-on-v6 { any; };
};
```

I've omitted some redundant lines from the file (such as comments). When we edited this file the last time, we uncommented the forwarders section and added two external DNS servers there. In order to allow the transfer of zone records to another server, we'll need to add a new line to this file. Here's the contents of this file again, with a new line added to it:

```
options {
        directory "/var/cache/bind";
        allow-transfer { localhost; 192.168.1.2; };
        forwarders {
                8.8.8.8;
                8.8.4.4;
        };
        dnssec-validation auto;

        auth-nxdomain no;
        listen-on-v6 { any; };
};
```

Basically, we added one line to the file, which allows zone transfer to an IP address of `192.168.1.2`, which you'd want to change to be the IP address for your secondary DNS server. Next, restart the `bind9` service on your primary server, and it should be ready to allow transfer to another machine.

www.it-ebooks.info

On the slave DNS server, the first thing you'll need to do is to install the `bind9` packages as we did with the first server. Then, we'll configure the slave server quite similar to the master server, but with some notable differences. On the slave server, you'll omit the `allow-transfer` line that we added to the `/etc/bind/named.conf.options` file, since the slave server is only going to receive records, not transfer them. Like before, you'll uncomment the forwarders section in the `/etc/bind/named.conf.options` file. In the `/etc/bind/named.conf.local` file, you'll add an entry for our zone just as we did with the first server, but we'll configure this file differently than we did the last time:

```
zone "local.lan" IN {
    type slave;
    masters { 192.168.1.1; };
    file "/var/lib/bind/net.local.lan";
};
```

Inside the `/etc/bind/named.conf.local` file on the slave server, we first identify it as a slave (`type slave;`) and then we set the master server to `192.168.1.1` (or whatever the IP address of your primary DNS server is). Then, we save our zone file to a different location than we did on the master; in this case, it is `/var/lib/bind/net.local.lan`. This is due to how the permissions differ on the slave and the master server, and `/var/lib/bind` is a good place to store this file. After the new configuration is in place, restart `bind9` on both the slave and the master servers. At this point though, we should test both servers out one last time to ensure both are working properly.

In this chapter, I've mentioned the `dig` command a few times. It's a great way to interrogate DNS servers to ensure they're providing the correct records. One thing I haven't mentioned so far is that you can specify a DNS server for the `dig` command to check, rather than having it use whatever DNS server the system was assigned. In my examples, I have two DNS servers set up, one at `192.168.1.1` and the other at `192.168.1.2`. Specifying a DNS server to interrogate with the `dig` command is easy enough:

```
dig @192.168.1.1 fileserv
dig @192.168.1.2 fileserv
```


www.it-ebooks.info

You should see results from both of your DNS servers. When you check the status of the `bind9` daemon on your slave server (`systemctl status bind9`), you should see entries that indicate a successful transfer. The output will look similar to this:

```
May 06 13:19:47 ubuntu-server named[2615]: transfer of 'local.lan/IN'
from 10.10.99.184#53: connected using 192.168.1.2#35275

May 06 13:19:47 ubuntu-server named[2615]: zone local.lan/IN: transferred
serial 201602093

May 06 13:19:47 ubuntu-server named[2615]: transfer of 'local.lan/IN'
from 10.10.99.184#53: Transfer status: success

May 06 13:19:47 ubuntu-server named[2615]: transfer of 'local.lan/IN'
from 10.10.99.184#53: Transfer completed: 1 messages, 10 records, 290

May 06 13:19:47 ubuntu-server named[2615]: zone local.lan/IN: sending
notifies (serial 201602093)
```

From this point onwards, you should have a redundant `bind` slave to work with, but the server is useless until you inform your clients to use it. To do that, we need to revisit our DHCP server configuration yet again. Specifically, the `/etc/dhcp/dhcpd.conf` file. Here are the sample contents of the file again, so you don't need to flip back to the earlier section:

```
default-lease-time 86400;
max-lease-time 86400;
option subnet-mask 255.255.255.0;
option broadcast-address 192.168.1.255;
option domain-name "local.lan";
authoritative;
subnet 192.168.1.0 netmask 255.255.255.0 {
    range 192.168.1.100 192.168.1.240;
    option routers 192.168.1.1;
    option domain-name-servers 192.168.1.1;
}
```

On the last line, we inform clients to set their DNS server to `192.168.1.1`. However, since we now have a slave server, we should change this file to include it so that it's passed along to clients:

```
default-lease-time 86400;
max-lease-time 86400;
option subnet-mask 255.255.255.0;
option broadcast-address 192.168.1.255;
option domain-name "local.lan";
authoritative;
```


www.it-ebooks.info

```
subnet 192.168.1.0 netmask 255.255.255.0 {
    range 192.168.1.100 192.168.1.240;
    option routers 192.168.1.1;
    option domain-name-servers 192.168.1.1, 192.168.1.2;
}
```

Basically, all we did was add the secondary server to the `option domain-name-servers` line. We separated the first with a comma, then we ended the line with a semicolon as we did before. At this point (assuming you've tested your secondary DNS server and trust it), you can restart the `isc-dhcp-server` daemon, and your nodes will have the primary and secondary DNS servers assigned to them the next time they check in for an IP address or to renew their existing lease.

That about does it for our journey into setting up `bind`. From now on, you have a DHCP server and a DNS server or two in your network, providing addressing and name resolution. We've pretty much set up two of the most common services that run on commercial routers, but we haven't actually set up a router just yet. In the next section though, we'll change that.

Setting up an Internet gateway

As long as we're setting up network services, we may as well go all the way and set up a router to act as a gateway for our network. In most commercial routers, we'll have DNS and DHCP built-in, as well as routing. Quite often, these services will all run on the same box. Depending on how you set up your DNS and DHCP servers in the previous sections, you may have even set up your primary DNS and DHCP servers on the same machine, which is quite common. However, your Internet connection will likely be terminated on a separate box, possibly a commercial routing device or Internet gateway from your Internet service provider.

Depending on what kind of Internet connection you have, Linux itself can likely replace whatever device that your Internet modem connects to. A good example of this is a cable modem that your office or home router may utilize. In this case, the modem provides your Internet connection, and then your router allows other devices on your network to access it. In some cases, your modem and router may even be the same device.

www.it-ebooks.info

Linux servers handle this job very well, and if you want to consolidate your DHCP, DNS, and routing into a single server, that's a very easy (and common) thing to do. Specifically, you'll need a server with at least two Ethernet ports, as well as a network switch that would allow you to connect multiple devices. If you need to connect devices with wireless network cards, you'll need an access point as well. Therefore, depending on the hardware you have, this method of setting up your networking may or may not be efficient. But if you do have the hardware available, you'll be able to manage the entire networking stack with Ubuntu Server quite easily.

In fact, we'll only need to execute a single command to set up routing between interfaces, which is technically all that's required in order to set up an Internet gateway. But before we get into that, it's also important to keep in mind that if you do set up an Internet gateway, you'll need to pay special attention to security. The device that sits between your network and your modem will be a constant attack target, just like any other gateway device would be. When it comes to commercial routers, they're also attacked constantly. However, in most cases, they'll have some sort of default security or firewall built-in. In all honesty, the security features built-in to common routing equipment are extremely poor and most of them are easy to hack when someone wants in bad enough. The point is that these devices have some sort of security to begin with (regardless of how good or bad), whereas a custom Internet gateway of your own won't have any security at all until you add it.

When you set up an Internet gateway, you'll want to pay special attention to setting up the firewall, restricting access to SSH, using very strong passwords, keeping up to date on security patches, as well as installing an authentication monitor such as `fail2ban`. We'll get into those topics in *Chapter 12*, *Securing Your Server*. The reason I bring this up now though is that if you do set up an Internet gateway, you'll probably want to take a detour and read that chapter right away, just to make sure that you secure it properly.

Anyway, let's move on. A proper Internet gateway, as I've mentioned, will have two Ethernet ports. On the first, you'll plug in your cable modem or Internet device, and you'll connect a switch on the second. By default though, routing between these interfaces will be disabled, so traffic won't be able to move from one Ethernet port to the other. To rectify this, use the following command:

```
# echo 1 > /proc/sys/net/ipv4/ip_forward
```

That's actually it. With that single command, you've just made your server into a router. However, that change will not survive a reboot. To make it permanent, open the `/etc/sysctl` file as `root` and look for the following line:

```
#net.ipv4.ip_forward=1
```


www.it-ebooks.info

Uncomment the line and save the file. With that change made, your server will allow routing between interfaces even after a reboot. Of all the topics we've covered in this chapter, that one was probably the simplest. However, I must remind you again to definitely secure your server if it's your frontend device to the Internet, as computer security students always enjoy practicing on a real-life Linux server. With good security practices, you'll help ensure that they'll leave you alone, or at least have a harder time breaking in.

Keeping your system clock in sync with NTP

It's incredibly important for Linux servers to keep their time synchronized, as strange things can happen when a server's clock is wrong. One issue I've run into that's especially a problem is file synchronization utilities, which will exhibit strange behavior when there are time issues. However, Ubuntu servers feature the Network Time Protocol client and server within the default repositories to help keep your time in sync. If it's not already installed, all you should need to do is install the `ntp` package:

```
# apt-get install ntp
```

Once installed, the `ntp` daemon will immediately start and will keep your time up to date. To verify, check the status of the `ntp` daemon with one of the following commands depending on the age of your server:

```
systemctl status ntp
/etc/init.d/ntp status
```

The output should show that `ntp` is running:

```
Active: active (running) since Sun 2016-05-06 14:09:28 EST; 3s ago
```

If all you wanted was a working NTP client, then you're actually all set. The default configuration is fine for most. But further explanation will help you understand how this client is configured. You'll find the configuration file at `/etc/ntp/ntp.conf`, which will contain some lines of configuration detailing which servers your local `ntp` daemon will attempt to synchronize time with:

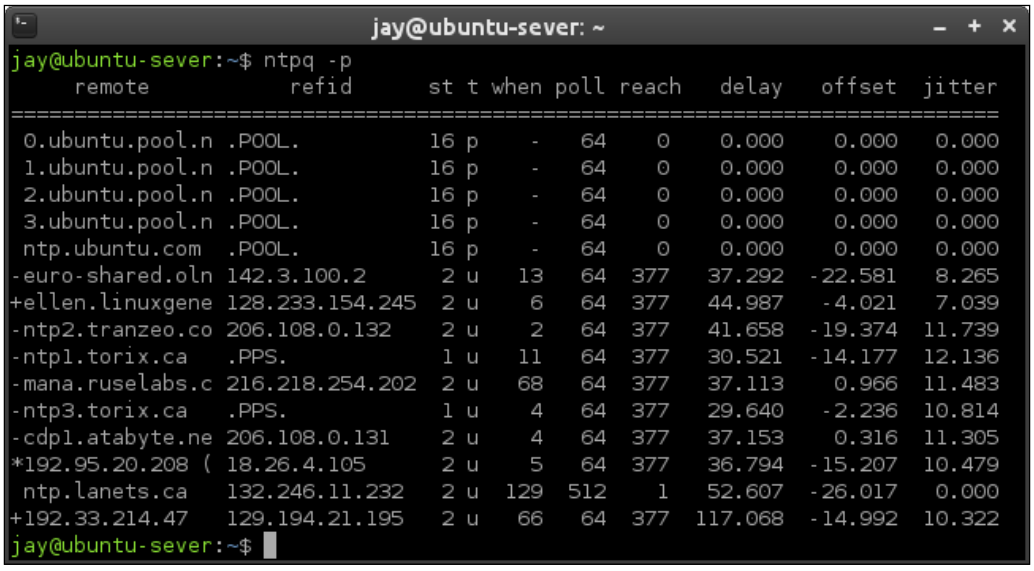
```
pool 0.ubuntu.pool.ntp.org iburst
pool 1.ubuntu.pool.ntp.org iburst
pool 2.ubuntu.pool.ntp.org iburst
pool 3.ubuntu.pool.ntp.org iburst
```


www.it-ebooks.info

As you can see, our server will synchronize with Ubuntu's time servers if we leave the default configuration as-is. For most users, that's perfectly fine. There's nothing wrong with using Ubuntu's servers. However, if you have a great number of clients in your organization, it may make sense to set up your own NTP server so that only one server is talking to the outside world instead of every one of your nodes. Essentially, this is how you can become a friendly neighbor online. If you have hundreds of workstations in your organization that are configured to check Ubuntu's NTP servers, that's quite a bit of traffic those servers will need to endure. To be fair, Ubuntu's servers won't have any trouble handling this traffic, though it's a nice gesture to configure one server in your network to synchronize with Ubuntu's time servers, then configure your local nodes to synchronize with just your local server. If everyone did this, then Ubuntu's time servers would see far less traffic.

Setting up a local NTP server is actually quite straightforward. All you should need to do is designate a server within your network for this purpose. You could even use the Internet gateway you set up in the previous section if you wanted to. Once you have that server set up to synchronize, you should be able to configure your local nodes to talk to that server and synchronize their clocks with it.

Before we get ahead of ourselves though, we should make sure that the server we installed NTP on is synchronizing properly. This will give us a chance to use the `ntpq` command, which we can use to view statistics about how well our server is synchronizing. The `ntpq -p` command should print out statistics we can use to verify connectivity:



Example output from the ntpq -p command

www.it-ebooks.info

To better understand the output of the `ntpq -p` command, I'll go through each column. First, the remote column details the NTP servers we're connected to. The `refid` column refers to the NTP servers that the remote servers are connected to. The `st` column refers to a server's stratum, which refers to how close the server is from us (the lower the number, the closer, and typically, better). The `t` column refers to the type, specifically whether the server is using **unicast**, **broadcast**, **multicast**, or **manycast**.

Continuing, the `when` column refers to how long ago it was since the last time the server was polled. The `poll` column indicates how often the server will be polled, which is 64 seconds for most of the entries in the example screenshot. The `reach` column contains the results of the most recent eight NTP updates. If all eight are successful, this field will read `377`. This number is in octal, so eight successes in octal will be represented by `377`. When you first start the `ntp` daemon on your server, it may take some time for this number to reach `377`.

Finally, we have the `delay`, `offset`, and `jitter` columns. The `offset` column refers to the delay in reaching the server, in milliseconds. `Offset` references the difference between the local clock and the server's clock. Finally, the `jitter` column refers to the network latency between your server and theirs.

Assuming that the results of the `ntpq -p` command look good and your server is synchronizing properly, you already essentially have an NTP server at your disposal, since there is little difference between a client and a server. As long as your server is synchronizing properly and port `123` is open between them through the firewall, you should be able to reconfigure your nodes to connect to your server, by changing the pool addresses in your clients configuration (within `/etc/ntp.conf`), to point to either the IP address or the **Fully Qualified Hostname** (**FQDN**) of your NTP server instead of Ubuntu's servers. Once you restart the `ntp` service on your other nodes, they should start synchronizing with the master server.

Before we close out this chapter, there is one additional change you should consider implementing to the `/etc/ntp.conf` file, however. Look for the following line within that file:

```
#restrict 192.168.123.0 mask 255.255.255.0 notrust
```

Change this line by first uncommenting it, changing the network address and subnet mask to match the details for your network, and then remove the `notrust` keyword at the end. The line should look similar to the following, but depending on your network configuration:

```
restrict 192.168.1.0 mask 255.255.255.0
```


www.it-ebooks.info

So, what does this option do for us? Basically, we're limiting access to our NTP server to local clients only, and we're only allowing read-only access for security purposes.

Now that you have a working NTP server, feel free to experiment and point your existing nodes to it and have them synchronize. Depending on the size of your network, a local NTP server may or may not make sense, but at the very least, NTP should be installed on every Linux workstation and server to ensure proper time synchronization. In most cases, the workstation version of Ubuntu will already be configured to synchronize to the Ubuntu time servers, but when it comes to the server version, NTP isn't typically installed for you.

Summary

In this chapter, we explored additional networking topics. We started off with some notes on planning an IP address scheme for your network so that you can create groups for the different types of nodes, such as servers and network equipment, as well as plan a pool of addresses for DHCP. We also set up a DHCP and DNS server, with an additional section on creating a DNS slave for added redundancy. We closed off this chapter with discussions on setting up an Internet gateway, as well as configuring NTP.

In the next chapter, we'll take a look at transferring and sharing files over the network. This will include covering NFS and Samba shares, as well as using `scp`, `rsync`, and `sshfs`. Stay tuned!

www.it-ebooks.info

8
Accessing and Sharing Files

Within an enterprise network, having one or more servers available to store files and make data accessible over the network is a great asset. Perhaps you've used a file server before, or even set one up on a different platform. With Ubuntu Server, there are multiple methods to not only store files, but to also transfer files from one node to another over a network link. In this chapter, we'll look into setting up a central file server using both Samba and NFS, as well as how to transfer files between nodes with utilities such as `scp` and `rsync`. We'll also go over some situations in which one solution may work better than another. As we go through these concepts, we will cover the following topics:

- File server considerations
- Sharing files with Windows users using Samba
- Setting up NFS shares
- Transferring files with `rsync`
- Transferring files with SCP
- Mounting remote filesystems with **SSH Filesystem** (**SSHFS**)

File server considerations

There are two methods you can use to share files with your users, Samba and NFS. In fact, there's nothing stopping you from hosting both Samba and NFS shares on a single server. However, each of the two popular solutions is valid for particular use cases. Before we get started with setting up our file server, we should first understand the differences between Samba and NFS, so we can make an informed decision as to which one is more appropriate for our environment. As a general rule of thumb, Samba is great for mixed environments (where you have Windows as well as Linux clients), and NFS is more appropriate for use in Linux or UNIX environments, but there's a bit more to it than that.

www.it-ebooks.info

Samba is a great solution for many environments, because it allows you to share files with Windows, Linux, and Mac OS X machines. Basically, pretty much everyone will be able to access your shares, provided you give them permission to do so. The reason this works is because Samba is a reimplementation of the **Server Message Block** (**SMB**) protocol, which is primarily used by Windows systems. However, you don't need to use the Windows platform in order to be able to access Samba shares, since many platforms offer support for this protocol. Even Android phones are able to access Samba file shares with the appropriate app, as well as other platforms.

You may be wondering, why I am going to cover two different solutions in this chapter. After all, if Samba shares can be accessed by pretty much everything and everyone, why bother with anything else? Even with Samba's many strengths, there are also weaknesses as well. First of all, permissions are handled very differently, so you'll need to configure your shares in specific ways in order to prevent access to users that shouldn't be able to see confidential data. With NFS, full support of standard UNIX permissions is provided, so you'll only need to be able to configure your permissions once. If permissions and confidentiality are important to you, you may want to look closer at NFS.

That's not to say that Windows systems cannot access NFS shares, because some versions actually can. By default, no version of Windows supports NFS outright, but some editions offer a plugin you can install that enables this support. The name of the NFS plugin in Windows has changed from one version to another (such as **Services for UNIX**, **Subsystem for UNIX-based Applications**, and **NFS Client**), but the idea is the same. You'll need to enable a specific plugin for your version of Windows in order to access NFS shares. The problem is that Microsoft limits access to the NFS plugin to only the more expensive Windows editions, such as Ultimate and Enterprise when it comes to Windows 7, and just Enterprise when it comes to Windows 8 and 10. Depending on how many Windows machines exist in your environment that need to access NFS shares, this could be a significant licensing cost to your organization. This is yet another reason why Samba is a great choice when you're dealing with the Windows platform; you can circumvent the cost of upgraded licenses, since even home editions of Windows can access Samba shares.

www.it-ebooks.info

In regards to an all Linux environment or in a situation where you only have Linux machines that need to access your shares, NFS is a great choice because its integration is much tighter with the rest of the distribution. Permissions can be more easily enforced and, depending on your hardware, performance may be higher. In addition, NFS can be a bit more time-consuming to set up in an environment, though by no means is it overly difficult to do. The specifics of your computing environment will ultimately make your decision for you. Perhaps you'll choose Samba for your mixed environment, or NFS for your all Linux environment. Maybe you'll even set up both NFS and Samba, having shares available for each platform. My recommendation is to learn and practice both, since you'll use both solutions at one point or another during your career.

Before you continue on to read the sections on setting up Samba and NFS, I recommend you first decide where in your filesystem you'd like to act as a parent directory for your file shares. This isn't actually required, but I think it makes for better organization. There is no one right place to store your shares, but personally I like to create a `/shared` directory at the `root` filesystem and create subdirectories for my network shares within it. For example, I can create `/shared/documents`, `/shared/public`, and so on for Samba shares. With regards to NFS, I usually create shared directories within `/exports`. You can choose how to set up your directory structure. As you read the remainder of this chapter, make sure to change my example paths to match yours if you use a different style.

Sharing files with Windows users using Samba

In this section, I'll walk you through setting up your very own Samba file server. I'll also go over a sample configuration to get you started so that you can add your own shares.

First, we'll need to make sure that the `samba` package is installed on our server:

```
# apt-get install samba
```


www.it-ebooks.info

When you install the `samba` package, you'll have a new daemon installed on your server, `smbd`. The `smbd` daemon will be automatically started and enabled for you. You'll also be provided a default configuration file for Samba, located at `/etc/samba/smb.conf`. For now, I recommend stopping Samba since we have yet to configure it:

```
# systemctl stop smbd
```

On older servers, you can stop Samba with the following command (note that the name of the daemon is also different on older Ubuntu versions):

```
# /etc/init.d/samba stop
```

Since we're going to configure Samba from scratch, we should start with an empty configuration file. Let's back up the original file, rather than overwrite it. The default file includes some useful notes and samples, so we should keep it around for future reference:

```
# mv /etc/samba/smb.conf /etc/samba/smb.conf.orig
```

Now, we can begin a fresh configuration. Although it's not required, I like to split my Samba configuration up between two files, `/etc/samba/smb.conf` and `/etc/samba/smbshared.conf`. You don't have to do this, but I think it makes the configuration cleaner and easier to read. First, here is a sample `/etc/samba/smb.conf` file:

```
[global]
server string = File Server
workgroup = MYWORKGROUP
security = user
map to guest = Bad User
name resolve order = bcast hosts wins
include = /etc/samba/smbshared.conf
```

As you can see, this is a really short file. Basically, we're including only the lines we absolutely need to in order to set up a file server with Samba. Next, I'll explain each line and what it does.

```
[global]
```

With the `[global]` stanza, we're declaring the global section of our configuration, which will consist of settings that will impact Samba as a whole. There will also be additional stanzas for individual shares, which we'll get to later.

```
server string = File Server
```


www.it-ebooks.info

The `server string` is somewhat of a description field for the `File Server`. If you've browsed networks from Windows computers before, you may have seen this field. Whatever you type here will show underneath the server's name in Windows Explorer. This isn't required, but it's nice to have.

```
workgroup = MYWORKGROUP
```

Here, we're setting the `workgroup`, which is the exact same thing as a `workgroup` on Windows PCs. In short, the workgroup is a namespace that describes a group of machines. When browsing network shares on Windows systems, you'll see a list of workgroups, and then one or more computers within that workgroup. In short, this is a way to logically group your nodes. You can set this to whatever you like. If you already have a workgroup in your organization, you should set it here to match the workgroup names of your other machines. The default workgroup name is simply **WORKGROUP** on Windows PCs, if you haven't customized the workgroup name at all.

```
security = user
```

This setting sets up Samba to utilize usernames and passwords for authentication to the server.

```
map to guest = Bad User
```

This option configures Samba to treat unauthenticated users as guest users. Basically, unauthenticated users will still be able to access shares, but they will have guest permissions instead of full permissions. If that's not something you want, then you can omit this line from your file. Note that if you do omit this, you'll need to make sure that both your server and client PCs have the same user account names on either side. Ideally, we would want to use directory-based authentication, but that's beyond the scope of this book.

```
name resolve order = bcast hosts wins
```

The `name resolve order` setting configures how Samba resolves hostnames. In this case, we're using the broadcast name first, followed by any mappings that might exist in our `/etc/hosts` file, followed by `wins`. Since `wins` has been pretty much abandoned (and replaced by DNS), we include it here solely for compatibility.

```
include = /etc/samba/smbshared.conf
```


www.it-ebooks.info

Remember how I mentioned that I usually split my Samba configurations into two different files? On this line, I'm calling that second file `/etc/samba/smbshared.conf`. The contents of the `smbshared.conf` file will be inserted right here, just as if we only had one file. We haven't created the `smbshared.conf` file yet. Let's take care of that next. Here's a sample `smbshared.conf` file.

```
[Documents]
    path = /share/documents
    force user = myuser
    force group = users
    public = yes
    writable = no

[Public]
    path = /share/public
    force user = myuser
    force group = users
    create mask = 0664
    force create mode = 0664
    directory mask = 0777
    force directory mode = 0777
    public = yes
    writable = yes
```

As you can see, I'm separating share declarations into its file. We can see several interesting things within `smbshared.conf`. First, we have two stanzas, `[Documents]` and `[Public]`. Each stanza is a share name, which will allow Windows users to access the share under `//servername/share-name`. In this case, this file will give us two shares: `//servername/Documents` and `//servername/Public`. The `Public` share is writable for everyone, though the `Documents` share is restricted to read-only. The `Documents` share has the following options.

```
path = /share/documents
```

This is the path to the share, which must exist on the server's filesystem. In this case, when a user reads files from `//servername/Documents` on a Windows system, they will be reading data from `/share/documents` on the Ubuntu Server that's housing the share.

```
force user = myuser
force group = users
```


www.it-ebooks.info

These two lines are basically bypassing user ownership. When a user reads this share, they are treated as `myuser` instead of their actual user account. Normally, you would want to set up LDAP or Active Directory to manage your user accounts and handle their mapping to the Ubuntu Server, but a full discussion of directory-based user access is beyond the scope of this book, so I provided the force options as an easy starting point. The user account you set here must exist on the server.

```
public = yes
writable = no
```

With these two lines, we're configuring what users are able to do once they connect to this share. In this case, `public = yes` means that the share is publicly available, though `writable = no` prevents anyone from making changes to the contents of this share. This is useful if you want to share files with others, but you want to restrict access from anyone being able to modify the content.

The `Public` share has some additional settings that weren't found in the `Documents` share.

```
create mask = 0664
force create mode = 0664
directory mask = 0777
force directory mode = 0777
```

With these options, I'm setting up how permissions of files and directories will be handled when new content is added to the share. Directories will be given `777` permissions and files will be given permissions of `664`. Yes, these permissions are very open; note that the share is named `Public`, which implies full-access anyway, and its intent is to house data that isn't confidential or restricted.

```
public = yes
writable = yes
```

Just as I did with the previous share, I'm setting up the share to be publicly available, but this time I'm also configuring it to allow users to make changes.

To take advantage of this configuration, we need to start the Samba daemon. Before we do though, we would want to double-check the directories we entered into our `smbshared.conf` file exist, so if you're using my example, you'll need to create `/shared/documents` and `/shared/public`. Also, the user account that was referenced in the force user and the group referenced in the force group must both exist, and have ownership over the shared directories. With that out of the way, feel free to start Samba:

```
# systemctl start smbd
```


www.it-ebooks.info

On older Ubuntu Servers, use the following command:

```
# /etc/init.d/samba start
```

That really should be all there is to it; you should now have a `Documents` and `Public` share on your file server that Windows users should be able to access. In fact, your Linux machines should be able to access these shares as well. On Windows, **Windows Explorer** has the ability to browse file shares on your network. If in doubt, try pressing the *Windows* key and the *R* key at the same time to open the run dialog, and then type the **Universal Naming Convention** (**UNC**) path to the share (`//servername/Documents` or `//servername/Public`). You should be able to see any files stored in either of those directories. In the case of the `Public` share, you should be able to create new files there as well.

On Linux systems, if you have a desktop environment installed, most of them feature a file manager that supports browsing network shares. Since there are a handful of different desktop environments available, the method varies from one distribution or configuration to another. Typically, most Linux file managers will have a network link within the file manager, which will allow you to easily browse your local shares. Otherwise, you can also access a Samba share by adding an entry for it in the `/etc/fstab` file, such as the following:

```
//myserver/shared/documents /mnt/documents cifs username=myuser,noauto
0 0
```

In order for the `fstab` entry to work, your Linux client will need to have the Samba client packages installed. If your distribution is Debian-based (such as Ubuntu), you would need to install the `smbclient` and `cifs-utils` packages:

```
# apt-get install smbclient cifs-utils
```

Then, assuming the local directory exists (`/mnt/documents` in the example), you should be able to mount the share with the following command:

```
# mount /mnt/documents
```

In the `fstab` entry, I included the `noauto` option so that your system won't mount the Samba share at boot time (you'll need to do so manually with the `mount` command). If you do want the Samba share automatically mounted at boot time, change `noauto` to `auto`. However, you may receive errors during the boot if for some reason the server hosting your Samba shares isn't accessible.

If you'd prefer to mount the Samba share without adding an `fstab` entry, the following example command should do the trick, just replace the share name and mount point to match your local configuration:

```
# mount -t cifs //myserver/Documents -o username=myuser /mnt/documents
```


www.it-ebooks.info

Setting up NFS shares

NFS is a great method of sharing files from a Linux or UNIX server to a Linux or UNIX server. As I mentioned earlier in the chapter, Windows systems can access NFS shares as well, but it comes with an additional licensing penalty. NFS is preferred in a Linux or UNIX environment though, since it fully supports Linux and UNIX style permissions. As you can see from our earlier dive into Samba, we essentially forced all shares to be treated as being accessed by a particular user, which was messy, but was the easiest example of setting up a Samba server. Samba can certainly support per-user access restrictions and benefit greatly from a centralized directory server, though that would basically be a book of its own! NFS is a bit more involved to set up, but in the long run, I think it's easier and integrates better.

Earlier, we set up a parent directory on our filesystem to house our Samba shares, and we should do the same thing with NFS. While it wasn't mandatory to have a special parent directory with Samba (I had you do that in order to be neat, but you weren't required to), NFS really does want its own directory to house all of its shares. It's not required with NFS either, but there's an added benefit in doing so, which I'll go over before the end of this section. In my case, I'll use `/exports` as an example, so you should make sure that directory exists:

```
# mkdir /exports
```

Next, let's install the required NFS packages on our server. The following command will install NFS and its dependencies:

```
# apt-get install nfs-kernel-server
```

Once you install the `nfs-kernel-server` package, the `nfs-kernel-server` daemon will start up automatically. However, it'll immediately fail and exit with the following error (which you'd see if you check the status of `nfs-kernel-server` with `systemctl`):

```
exportfs: can't open /etc/exports for reading
```

This error is to be expected, because we haven't actually configured anything yet. What's happening here is the `/etc/exports` file (which is the main file that NFS reads its share information from), doesn't contain any useful settings, just some commented lines. Let's back up the `/etc/exports` file, since we'll be creating our own:

```
# mv /etc/exports /etc/exports.orig
```


www.it-ebooks.info

To set up NFS, let's first create some directories that we will share to other users. Each share in NFS is known as an **Export**. I'll use the following directories as examples, but you can export any directory you like:

```
/exports/backup
/exports/documents
/exports/public
```

In the `/etc/exports` file (which we're creating fresh), I'll insert the following four lines.

```
/exports *(ro,fsid=0)
/exports/backup 192.168.1.0/255.255.255.0(rw,no_subtree_check)
/exports/documents 192.168.1.0/255.255.255.0(ro,no_subtree_check)
/exports/public 192.168.1.0/255.255.255.0(rw,no_subtree_check)
```

The first line is `Export Root`, which I'll go over a bit later. The next three lines are individual shares or exports. The backup, documents, and public directories are being shared from the `/exports` parent directory. Each of these lines is not only specifying which directory is being shared with each export, but also which network is able to access them. In this case, after the directory is called out in a line, we're also setting which network is able to access them (`192.168.1.0/255.255.255.0` in our case). This means that if you're connecting from a different network, your access will be denied. Each connecting machine must be a member of the `192.168.1.0/24` network in order to proceed (so make sure you change this to match your IP scheme). Finally, we include some options for each export, for example, `rw,no_subtree_check`.

As far as what these options do, the first (`rw`) is rather self-explanatory. We can set here whether or not other nodes will be able to make changes to data within the export. In the examples I gave, the documents export is read-only (`ro`), while the others allow read and write.

The next option on each example is `no_subtree_check`. This option is known to increase reliability, and is mainly implied by default. However, not including it may make NFS complain when it restarts, but nothing that will actually stop it from working. Particularly, this option disables what is known as **subtree checking**, which has had some stability issues in the past. Normally, when a directory is exported, NFS might scan parent directories as well, which is sometimes problematic, and can cause issues when it comes to open file handles.

There are several other options that can be included in an `export`, and you can read more about them by checking the man page for `export`:

```
man export
```


www.it-ebooks.info

One option you'll see quite often in the wild is `no_root_squash`. Normally, the `root` user on one system is mapped to nobody on the other for security reasons. In most cases, one system having `root` access to another is a bad idea. The `no_root_squash` option disables this, and it allows the `root` user on one end to be treated as the `root` user on the other. I can't think of a reason personally where this would be useful (or even recommended), but I have seen this option quite often in the wild, so I figured I would bring it up. Again, check the man pages for **export** for more information on additional options you can pass to your exports.

Next, we have one more file to edit before we can actually seal the deal on our NFS setup. Open `/etc/idmapd.conf` in your text editor. Look for an option that is similar to the following:

```
# Domain = localdomain
```

First, remove the `#` symbol from that line to uncomment it. Then, change the domain to match the one used within the rest of your network. You can leave this as-is as long as it's the same on each node, but if you recall from *Chapter 7*, *Managing Your Ubuntu Server Network*, we used a sample domain of `local.lan` in our DHCP configuration, so it's best to make sure you use the same domain name everywhere—even the domain provided by DHCP. Basically, just be as consistent as you can and you'll have a much easier time overall. You'll also want to edit the `/etc/idmapd.conf` file on each node that will access your file server, to ensure they are configured the same as well.

The `/etc/idmapd.conf` file is necessary for mapping permissions on one node to another. In *Chapter 2*, *Managing Users*, we talked about the fact each user has an ID (`UID`) assigned to them. The problem though is that from one system to another, a user will not typically have the same ID. For example, user `jdoe` may be `UID 1001` on **Server A**, but `1007` on **Server B**. When it comes to NFS, this greatly confuses the situation, because UIDs are used in order to reference permissions. Mapping IDs with `idmapd` allows this to stay consistent and handles translating each user properly, though it must be configured correctly and consistently on each node. Basically, as long as you use the same domain name on each server and client and configure the `/etc/idmapd.conf` file properly on each, you should be fine.

With our `/etc/exports` and `/etc/idmapd.conf` files in place, and assuming you've already created the exported directories on your filesystem, we should be all set to restart NFS to activate our configuration:

```
# systemctl restart nfs-kernel-server
```

On older servers, you'll use:

```
# /etc/init.d/nfs-kernel-server restart
```


www.it-ebooks.info

After restarting NFS, you should check the deamon's output via `systemctl` to ensure that there are no errors:

```
# systemctl status -l nfs-kernel-server
```

As long as there are no errors, our NFS server should be working. Now, we just need to learn how to mount these shares on another system. Unlike Samba, using a Linux file manager and browsing the network will not show NFS exports, we'll need to mount them manually. Client machines, assuming they are Debian-based (Ubuntu fits this description) will need the `nfs-common` package installed in order to access these exports:

```
# apt-get install nfs-common
```

With the client installed, we can now use the `mount` command to mount NFS exports on a client. For example, with regards to our documents export, we can use the following variation of the `mount` command to do the trick:

```
# mount myserver:/documents /mnt/documents
```

Replace `myserver` with either your server's hostname or its IP address. From this point forward, you should be able to access the contents of the documents export on your file server. Notice, however, that the exported directory on the server was `/exports/documents`, but we only asked for `/documents` instead of the full path with the example `mount` command. The reason this works is because we identified an export `root` of `/exports`. To save you from flipping back, here's the first line from the `/etc/exports` file, where we identified our export `root`:

```
/exports *(ro,fsid=0)
```

With the export root, we basically set the base directory for our NFS exports. We set it as read-only (`ro`), because we don't want anyone making any changes to the `/exports` directory itself. Other directories within `/exports` have their own permissions and will thus override the `ro` setting on a per-export basis, so there's no real reason to set our export root as anything other than read-only. With our export root set, we don't have to call out the entire path of the export when we mount it; we only need the directory name. This is why we can mount an NFS export from `myserver:/documents` instead of having to type the entire path. While this does save us a bit of typing, it's also useful because from the user's perspective, they aren't required to know anything about the underlying filesystem on the server. There's simply no value for the user to have to memorize the fact that the server is sharing a documents directory from `/exports`, all they're interested in is getting to their data. Another benefit is if we ever need to move our export root to a different directory (during a maintenance period), our users won't have to change their configuration to reference the new place; they'll only need to unmount and remount the exports.

www.it-ebooks.info

So, at this point, you'll have three directories being exported from your file server, and you can always add others as you go. However, anytime you add a new export, they won't be automatically added and read by NFS. You can restart NFS to activate new exports, but that's not really a good idea while users may be connected to them, since that will disrupt their access. Thankfully, the following command will cause NFS to reread the `/etc/exports` file without disrupting existing connections. This will allow you to activate new exports immediately without having to wait for users to finish what they're working on:

```
# exportfs -a
```

With this section out of the way, you should be able to export a directory on your Ubuntu Server, and then mount that export on another Linux machine. Feel free to practice creating and mounting exports until you get the hang of it. In addition, you should familiarize yourself with a few additional options and settings that are allowable in the `/etc/exports` file, after consulting with the man page on export. When you've had more NFS practice than you can tolerate, we'll move on to a few ways in which you can copy files from one node to another without needing to set up an intermediary service or daemon.

Transferring files with rsync

Of all the countless tools and utilities available in the Linux and UNIX world, few are as beloved as `rsync`. `rsync` is a utility that you can use to copy data from one place to another very easily, and there are many options available to allow you to be very specific on how you want the data transferred. Examples of its many use-cases include copying files while preserving permissions, copying files while backing up replaced files, and even setting up incremental backups. If you don't already know how to use `rsync`, you'll probably want to get lots of practice with it, as it's something you'll soon see will be indispensable during your career as a Linux administrator, and it is also something that the Linux community generally assumes you already know. `rsync` is not hard to learn. Most administrators can learn the basic usage in about an hour, but the countless options available will lead you to learn new tricks even years down the road.

www.it-ebooks.info

Another aspect that makes `rsync` flexible is the many ways you can manipulate the source and target directories. I mentioned earlier that `rsync` is a tool you can use to copy data from one place to another. The beauty of this is that the source and target can literally be anywhere you'd like. For example, the most common usage of `rsync` is to copy data from a directory on one server to a directory on another server over the network. However, you don't even have to use the network; you can even copy data from one directory to another on the same server. While this may not seem like a useful thing to do at first, consider that the target directory may be a mount-point that leads to a backup disk, or an NFS share that actually exists on another server. This also works in reverse, you can copy data from a network location to a local directory if you desired.

To get started with practicing with `rsync`, I recommend that you find some sample files to work with. Perhaps you have a collection of documents you can use, MP3 files, videos, text files, basically any kind of data you have lying around. It's important to make a copy of this data. If we make a mistake we could overwrite things, so it's best to work with a copy of the data, or the data you don't care about while you're practicing. If you don't have any files to work with, you can create some text files. The idea is to practice copying files from one place to another, it really doesn't matter what you copy or to where you send it. I'll walk you through some `rsync` examples that will progressively increase in complexity. The first few examples will show you how to back up a `home` directory, but later examples will be potentially destructive so you will probably want to work with sample files until you get the hang of it.

Here's our first example:

```
# rsync -r /home/myusr /backup
```

With that command, we're using `rsync` (as `root`) to copy the contents of the `home` directory for the `myuser` directory to a backup directory, `/backup` (make sure the target directory exists). In the example, I used the `-r` option, which means `rsync` will grab directories recursively as well (you should probably make a habit of including this if you use no other option). You should now see a copy of the `myuser home` directory inside your `/backup` directory.

www.it-ebooks.info

However, we have a bit of a problem. If you look at the permissions in the `/backup/myuser` directory, you can see that everything in the target is now owned by `root`. This isn't a good thing; when you back up a user's `home` directory, you'll want to retain their permissions. In addition, you should retain as much metadata as you can, including things like timestamps. Let's try another variation of `rysnc`. Don't worry about the fact that `/backup` already has a copy of the `myuser home` directory from our previous backup. Let's perform the backup again, but instead we'll use the `-a` option:

```
# rsync -a /home/myuser /backup
```

This time, we replaced the `-r` option with `-a` (archive) that retains as much metadata as possible (in most cases, it should make everything an exact copy). What you should notice now is that the permissions within the backup match the permissions within the user's `home` directory we copied from. The timestamps of the files will now match as well. This worked because whenever `rsync` runs, it will copy what's different from the last time it ran. The files from our first backup were already there, but the permissions were wrong. When we ran the second command, `rsync` only needed to copy that was different, so it applied the correct permissions to the files. If any new files were added to the source directory since we last ran the command, the new or updated files would be copied over as well.

`The archive` mode (the `-a` option that we used with the previous command) is actually very popular; you'll probably see it a lot in the wild. The `-a` option is actually a wrapper option that includes the following options at the same time:

```
-rlptgoD
```

If you're curious what each of these options do, consult the man page for `rsync`. As a spoiler, the `-r` option copies data recursively (which we already knew), the `-l` option copies symbolic links, `-p` preserves permissions, `-g` preserves group ownership, `-o` preserves the owner, and `-D` preserves device files. If you put those options together, we get `--rlptgoD`. Therefore, `-a` is actually equal to `-rlptgoD`. I find `-a` easier to remember.

`The archive` mode is great and all, but wouldn't it be nice to be able to watch what `rsync` is up to when it runs? Add the `-v` option and try the command again:

```
# rsync -av /home/myuser /backup
```


www.it-ebooks.info

This time, `rsync` will display on your terminal what it's doing as it runs (`-v` activates the **verbose** mode). This is actually one of my favorite variations of the `rsync` command, as I like to copy everything and retain all the metadata, as well as watch what `rsync` is doing as it works.

What if I told you that `rsync` supports SSH by default? It's true! Using `rsync`, you can easily copy data from one node to another, even over SSH. The same options apply, so you don't actually have to do anything different other than point `rsync` to the other server, rather than to another directory on your server:

```
# rsync -av /home/myuser admin@192.168.1.5:/backup
```

With this example, I'm copying the `home` directory for `myuser` to the `/backup` directory on server `192.168.1.5`. I'm connecting to the other server as the `admin` user. Make sure you change the user account and IP address accordingly, and also make sure the user account you use has access to the `/backup` directory. When you run this command, you should get prompted for the SSH password as you would when using plain SSH to connect to the server. After the connection is established, the files will be copied to the target server and directory.

Now, we'll get into some even cooler examples (some of which are potentially destructive), and we probably won't want to work with an actual `home` directory for these, unless it's a test account and you don't care about its contents. As I've mentioned before, you should have some test files to play with. When practicing, simply replace my directories for yours. Here's another variation worth trying:

```
# rsync -av --delete /src /target
```

Now I'm introducing you to the `--delete` option. This option allows you to synchronize two directories. Let me explain why this is important. With every `rsync` example up until now, we've been copying files from point A to point B, but we weren't deleting anything. For example, let's say you've already used `rsync` to copy contents from point A to point B. Then, you delete some files in point A. When you use `rsync` to copy files from point A to point B again, the files you deleted in point A won't be deleted in point B. They'll still be there. This is because by default, `rsync` copies data between two locations but it doesn't remove anything. With the `--delete` option, you're effectively synchronizing the two points, thus you're telling `rsync` to make them the same by allowing it to delete files in the target that are no longer in the source.

Next, we'll add the `-b` (backup) option:

```
# rsync -avb --delete /src /target
```


www.it-ebooks.info

This one is particularly useful. Normally, when a file is updated on `/src` and then copied over to `/target`, the copy on `/target` is overwritten with the new version. But what if you don't want any files to be replaced? The `-b` option renames files on the target that are being overwritten, so you'll still have the original file. If you add the `--backup-dir` option, things get really interesting:

```
# rsync -avb --delete --backup-dir=/backup/incremental /src /target
```

Now, we're copying files from `/src` to `/target` as we were before, but we're now sending replaced files to the `/backup/incremental` directory. This means that when a file is going to be replaced on the target, the original file will be copied to `/backup/incremental`. This works because we used the `-b` option (backup) but we also used the `--backup-dir` option, which means that replaced files won't be renamed, they'll simply be moved to the designated directory. This allows us to effectively perform incremental backups.

Building on our previous example, we can use the Bash shell itself to make incremental backups work even better. Here we have two commands:

```
# CURDATE=$(date +%m-%d-%Y)
# rsync -avb --delete --backup-dir=/backup/incremental/$CURDATE /src /
target
```

With this example, we grab the current date and set it to a variable (`CURDATE`). In the second command, we use the variable for the `--backup-dir` option. This will copy replaced files to a `backup` directory named after the `date` the command was run. Basically, if today's date was `05-01-2016`, the resulting command would be the same as if we've run the following:

```
# rsync -avb --delete --backup-dir=/backup/incremental/05-01-2016 /src /
target
```

If we were running this command every day, we would have the contents of `/src` copied to `/target`, with replaced files written to a backup directory with the current date. This is something we would probably want to set up in a script and execute daily via `cron`. This works well because `/target` could be a mounted NFS export, or external drive, so our script could be a daily backup script. Here's what it would look like if we made these commands into a script. Store the following in a text file such as `backup.sh`:

```
#/bin/bash
CURDATE=$(date +%m-%d-%Y)
rsync -avb --delete --backup-dir=/backup/incremental/$CURDATE /src /
target
```


www.it-ebooks.info

Now, we need to make `backup.sh` executable:

```
chmod +x backup.sh
```

Then, move the file to `/usr/local/bin` (which is a great place to store scripts):

```
# mv backup.sh /usr/local/bin
```

Now, you can run this `rsync` job anytime by calling out the full path to the script:

```
# /usr/local/bin/backup.sh
```

With this script in place, you can now set up a `cron` job to make it run daily. Refer to *Chapter 6*, *Controlling and Monitoring Processes* for more information on how to create `cron` jobs.

Hopefully, you can see how flexible `rsync` is and how it can be used to not only copy files between directories and/or nodes, but also to serve as a backup solution as well (assuming you have a remote destination to copy files to). The best part is that this is only the beginning. If you consult the man page for `rsync`, you'll see that there are a lot of options you can use to customize it even further. Give it some practice, and you should get the hang of it in no time.

Transferring files with SCP

A useful alternative to `rsync` is the **Secure Copy** (**SCP**) utility, which comes bundled with OpenSSH. It allows you to quickly copy files from one node to another. While `rsync` also allows you to copy files to other network nodes via SSH, SCP is more practical for one-off tasks; `rsync` is geared more toward more complex jobs. If your goal is to send a single file or a small number of files to another machine, SCP is a great tool you can use to get the job done. To utilize SCP, we'll use the `scp` command. Since you most likely already have OpenSSH installed, you should already have the `scp` command available. If you execute `which scp`, you should receive the following output:

```
/usr/bin/scp
```

If you don't see any output, make sure that the `openssh-client` package is installed.

Using SCP is very similar in nature to `rsync`. The command requires a source, a target, and a filename. To transfer a single file from your local machine to another, the resulting command would look similar to the following:

```
scp myfile.txt jdoe@192.168.1.50:/home/jdoe
```


www.it-ebooks.info

With this example, we're copying the file `myfile.txt` (which is located in our current working directory) to a server located at `192.168.1.50`. If the target server is recognized by DNS, we could've used the DNS name instead of the IP address. The command will connect to the server as user `jdoe` and place the file into that user's `home` directory. Actually, we can shorten that command a bit:

```
scp myfile.txt jdoe@192.168.1.50:
```

Notice that I removed the target path, which was `/home/jdoe`. I'm able to omit the path on the target, since the `home` directory is assumed if you don't give the `scp` command a target path. Therefore, the `myfile.txt` file will end up in `/home/jdoe` whether or not I included the path. If I wanted to copy the file somewhere else, I would definitely need to call out the location. Make sure you always include at least the colon when copying a file, since if you don't include it, you'll end up copying the file to your current working directory instead of the target.

The `scp` command also works in reverse as well:

```
scp jdoe@192.168.1.50:myfile.txt  .
```

With this example, we're assuming that `myfile.txt` is located in the `home` directory for the user `jdoe`. This command will copy that file to the current working directory of our local machine, since I designated the local path as a single period (which corresponds to our current working directory). Using `scp` in reverse isn't always practical, since you have to already know the file is where you expect it to be before transferring it.

With our previous `scp` examples, we've only been copying a single file. If we want to transfer or download an entire directory and its contents, we will need to use the `-r` option, which allows us to do a recursive copy:

```
scp -r /home/jdoe/downloads/linux_iso jdoe@192.168.1.50:downloads
```

With this example, we're copying the local folder `/home/jdoe/downloads/linux_iso` to remote machine `192.168.1.50`. Since we used the `-r` option, `scp` will transfer the `linux_iso` folder and all of its contents. On the remote end, we're again connecting via the user `jdoe`. Notice that the target path is just simply `downloads`. Since `scp` defaults to the user's `home` directory, this will copy the `linux_iso` directory from the source machine to the target machine under the `/home/jdoe/downloads` directory. The following command would've had the exact same result:

```
scp -r /home/jdoe/downloads/linux_iso jdoe@192.168.1.50:/home/jdoe/
downloads
```


www.it-ebooks.info

The `home` directory is not the only assumption the `scp` command makes, it also assumes that SSH is listening on port `22` on the remote machine. Since it's possible to change the SSH port on a server to something else, port `22` may or may not be what's in use. If you need to designate a different port for `scp` to use, use the `-P` option:

```
scp -P 65222 -r /home/jdoe/downloads/linux_iso
jdoe@192.168.1.50:downloads
```

With that example, we're connecting to the remote machine via port `65222`. If you've configured SSH to listen on a different port, change the number accordingly.

Although port `22` is always the default for OpenSSH, it's common for some administrators to change it to something else. While changing the SSH port doesn't add a great deal of benefit in regards to security (an intensive portscan will still find your SSH daemon), it's a relatively easy change to make, and making it even just a little bit harder to find is of benefit. We'll discuss this further in *Chapter 12*, *Securing Your Server*.

Like most commands in the Linux world, the `scp` command supports the verbose mode. If you wanted to see how the `scp` command progresses as it copies multiple files, add the `-v` option:

```
scp -rv /home/jdoe/downloads/linux_iso jdoe@192.168.1.50:downloads
```

Well, there you have it. The `scp` command isn't overly complex or advanced, but it's really great for situations in which you want to perform a one-time copy of a file from one node to another. Since it copies files over SSH, you benefit from its security, and it also integrates well with your existing SSH configuration. An example of this integration is the fact that `scp` recognizes your `~/.ssh/config` file (if you have one), so you can shorten the command even further. Go ahead and practice with it a bit, and in the next section, we'll go over yet another trick that OpenSSH has up its sleeve.

www.it-ebooks.info

Mounting remote filesystems with SSHFS

Earlier in this chapter, we took a look at several ways in which we can set up a Linux fileserver, using Samba and/or NFS. There's another type of file sharing solution I haven't mentioned yet, the **SSH File System** (**SSHFS**). NFS and Samba are great solutions for designating file shares that are to be made available to other users, but these technologies may be more complex than necessary if you want to set up a temporary file-sharing service to use for a specific period of time. SSHFS allows you to mount a remote directory on your local machine, and have it treated just like any other directory. The mounted SSHFS directory would be available for the life of the SSH connection. When you're finished, you simply disconnect the SSHFS mount.

There are some downsides when it comes to SSHFS, however. First, performance of file transfers wouldn't be as fast as with an NFS mount, since there's encryption that needs to be taken into consideration as well. However, unless you're performing really resource-intensive work, you probably won't notice much of a difference anyway. Another downside is that you'd want to save your work regularly as you work on files within an SSHFS mount, because if the SSH connection drops for any reason, you may lose data. This logic is also true of NFS and Samba shares, but SSHFS is more of an on-demand solution and not something intended to remain connected and in place all the time.

To get started with SSHFS, we'll need to install it:

```
# apt-get install sshfs
```

Now we're ready to roll. For SSHFS to work, we'll need a directory on both your local Linux machine as well as a remote Linux server. SSHFS can be used to mount any directory on the remote server you would normally be able to access via SSH. That's really the only requirement. What follows is an example command to mount an external directory to a local one via SSHFS. In your tests, make sure to replace my sample directories with actual directories on your nodes, as well as use a valid user account:

```
sshfs myuser@192.168.1.50:/share/myfiles /mnt/myfiles
```


www.it-ebooks.info

As you can see, the `sshfs` command is fairly straightforward. With this example, we're mounting `/share/myfiles` on `10.10.99.204` to `/mnt/myfiles` on our local machine. Assuming the command didn't give an error (such as access denied, if you didn't have access to one of the directories on either side), your local directory should show the contents of the remote directory. Any changes you make to the directory on either side will immediately propagate to the other. The SSHFS mount will basically function the same way as if you had mounted an NFS or Samba share locally.

When we're finished with the mount, we should unmount it. There are two ways to do so. First, we can use the `umount` command as `root` (just like we normally would):

```
# umount /mnt/myfiles
```

Using `umount` command isn't always practical for SSHFS, though. The user that's setting up the SSHFS link may not have `root` permissions, which means that he or she wouldn't be able to unmount it with the `umount` command. If you tried the `umount` command as a regular user, you would see an error similar to the following:

```
umount: /mnt/myfiles: Permission denied
```

It may seem rather strange that a normal user can mount an external directory via SSHFS, but not unmount it. Thankfully, there's a specific command a normal user can use, so we won't need to give them `root` or `sudo` access:

```
fusermount -u /mnt/myfiles
```

That should do it. With the `fusermount` command, we can unmount the SSHFS connection we set up, even without `root` access. The `fusermount` command is part of the **File System in Userspace** (**FUSE**) suite, which is what SSHFS uses as its virtual filesystem to facilitate such a connection. The `-u` option, as you've probably guessed, is for unmounting the connection normally. There is also the `-z` option, which unmounts the SSHFS mount *lazily*. This is a last resort that you should rarely need to use.

Connecting to an external resource via SSHFS can be simplified by adding an entry for it in `/etc/fstab`. Here's an example entry, using our previous example:

```
myuser@192.168.1.50:/share/myfiles     /mnt/myfiles     fuse.sshfs
rw,noauto,users,_netdev  0  0
```


www.it-ebooks.info

Notice that I used the `noauto` option in that `fstab` entry, which means that your system will not automatically attempt to bring up this SSHFS mount when it boots. Typically, this is ideal. The nature of SSHFS is to create on-demand connections to external resources, and we wouldn't be able to input the password for the connection while the system is in the process of booting anyway. Even if we set up password-less authentication, the SSH daemon may not be ready by the time the system attempts to mounted, so it's best to leave the `noauto` option in place and just use SSHFS as the on-demand solution it is. With this `/etc/fstab` entry in place, any time we would like to mount that resource via SSHFS, we would only need to execute the following command going forward:

```
mount /mnt/myfiles
```

Since we now have an entry for `/mnt/myfiles` in `/etc/fstab`, the `mount` command knows that this is an SSHFS mount, where to locate it, and which user account to use for the connection. After you execute the example `mount` command, you should be asked for the user's SSH password (if you don't have password-less authentication configured) and then the resource should be mounted.

SSH sure does have a few unexpected tricks up its sleeve. Not only is it the de facto standard in the industry for connecting to Linux servers, but it also offers us a neat way of transferring files quickly and mounting external directories. I find SSHFS very useful in situations where I'm working on a large number of files on a remote server, but want to work on them with applications I have installed on my local workstation. SSHFS allows us to do exactly that.

Summary

In this chapter, we explored multiple ways of accessing remote resources. Just about every network has a central location for storing files, and we explored two ways of accomplishing this with NFS and Samba. Both NFS and Samba have their place in the data center and are very useful ways we can make resources on a server available to our users who need to access them. We also talked about `rsync` and `scp`, two great utilities for transferring data without needing to set up a permanent share. We closed off the chapter with a look at SSHFS, which is a very handy utility for mounting external resources locally, on demand.

Next up is *Chapter 9*, *Managing Databases*. Now that we have all kinds of useful services running on our Ubuntu Server network, it's only fitting that we take a look at serving databases as well. Specifically, we'll look at MariaDB. See you there!

www.it-ebooks.info

www.it-ebooks.info

9
Managing Databases

The Linux platform has long been a very popular choice for hosting databases. Given the fact that databases power a large majority of popular websites across the Internet nowadays, this is a very important role for servers to fill. Ubuntu Server is also a very popular choice for this purpose, as its stability is a major benefit to the hosting community. This time around, we'll take a look at MariaDB, a popular fork of MySQL. The goal won't be to provide a full walkthrough of MySQL's syntax (as that would be a full book in and of itself), but we'll focus on setting up and maintaining database servers utilizing MariaDB and we'll even go over how to set up a master/slave relationship between them. If you already have a firm understanding of how to architect databases, you'll still benefit from this chapter as we'll also discuss what sets Ubuntu's implementation apart from other distributions (and there are some fairly sizable differences).

As we work through setting up our very own MariaDB server, we will cover the following topics:

- Preparations for setting up a database server
- Installing MariaDB
- Understanding how MariaDB differs in Ubuntu 16.04
- Taking a look at MariaDB configuration
- Managing databases
- Setting up a slave DB server

www.it-ebooks.info

Preparations for setting up a database server

Before we get started with setting up our database server, there are a few odds and ends to get out of the way. As we go through this chapter, we'll set up a basic database server using MariaDB. I'm sure more than a few of you are probably familiar with MySQL. MySQL is a tried and true solution which is in use in many data centers today. There's a good chance that a popular website or two that you regularly visit utilizes it on the backend. So you may be wondering then, why not go over that instead of MariaDB?

There are two reasons why this book will focus on MariaDB. First, the majority of the Linux community is migrating over to it (more on that later), and it's also a drop-in replacement for MySQL. This means that any databases or scripts you've already written for MySQL should work just fine with MariaDB. The reason I say "should" instead of giving you a personal guarantee is because there are always edge-cases when it comes to technology. By and large, it stands to reason that all the MySQL syntax you're accustomed to and any scripts you've written or databases you migrate over should indeed work just fine. The commands you practice with MariaDB should also function as you would expect on a MySQL server. This is great, considering that many MySQL installations are still in use in many data centers, and you'll be able to support those too. For the most part, there are very few reasons to stick with MySQL when your existing infrastructure can be ported over to MariaDB, and that's the direction the Linux community is headed toward anyway.

Why the change? If you've been paying attention to news within the Linux community nowadays, you may have seen articles from time to time regarding various distributions switching to MariaDB from MySQL. Red Hat is one such example; it switched to MariaDB in version 7 of Red Hat Enterprise Linux. Other distributions, such as Arch Linux and Fedora, went the same route. This was partly due to a lack of trust in Oracle, the company that now owns MySQL. When Oracle became the owner of MySQL, there were some serious questions raised within the open source community regarding the future of MySQL as well as its licensing. The Linux community, in general, doesn't seem to trust Oracle as a company, and in turn doubts its stewardship of MySQL. I'm not going to get into any speculation about Oracle, the future of MySQL, or any tin-foil hat conspiracies regarding its future since it's not relevant to this book (and I'm not a fan of drama). The fact, though, is that many distributions are moving toward MariaDB, and that seems to be the future. It's a great technology, and I definitely recommend it over MySQL for several reasons.

www.it-ebooks.info

MariaDB is more than just a fork of MySQL. On its own, it's a very competent database server. The fact that your existing MySQL implementations should be compatible with it eases adoption. But more than that, MariaDB makes some very worthwhile changes and improvements to MySQL that will only benefit you. It's kind of like the Marvel Comics character, Blade; he has all the strengths of vampires, but none of their weaknesses. Everything you love about MySQL can be found in MariaDB, plus some cutting-edge features that are exclusive to it. Some of the improvements within MariaDB include the fact that we'll receive faster security patches (since developers don't need to wait for approval from Oracle before releasing updates), as well as better performance. But even better is the fact that MariaDB features additional clustering options that are leaps and bounds better and more efficient than plain-old MySQL.

So hopefully I've sold you on the value of MariaDB. Ultimately, whether or not you actually use it will depend on the needs of your organization. I've seen some organizations opt to stick with MySQL, if only for the sole reason that it's what they know, and new technologies tend to scare management. I can understand that if a solution has proven itself in your data center, there's really no reason to change if your database stack is working perfectly fine the way it is. To that end, while I'll be going over utilizing MariaDB, most of my instructions should work for MySQL as well.

With regards to your server, a good implementation plan is key (as always). I won't spend too much time on this aspect, since by now I know you've probably been through a paragraph or two in this book where I've mentioned the importance of redundancy (and I'm sure I'll mention redundancy again a few more times before the last page). At this point, you're probably just setting up a lab environment or test server on which to practice these concepts before using your new-found skills in production. But when you do eventually roll out a database server into production, it's crucial to plan for long-term stability. Database servers should be regularly backed up, redundant (there I go again), and regularly patched. Later on in this chapter, I'll walk you through setting up a slave database server, which will take care of the redundancy part. However, that's not enough on its own, as regular backups are important. There are many utilities that allow you to do this, such as `mysqldump`, and also snapshots of your virtual machine (assuming you're not using a physical server). Both solutions are valid, depending on your environment. As someone that's lost an entire work day attempting to resurrect a fallen database server for a client (of which they had no backups or redundancy) my goal is simply to spare you that headache.

www.it-ebooks.info

As far as how much resources a database server needs, that solely depends on your environment. MariaDB itself does not take up a huge amount of resources, but as with MySQL, your usage is dependent on your workload. Either you'll have a few dozen clients connecting or a few thousand or more. But one recommendation I'll definitely make is to use LVM for the partition that houses your database files. This will certainly spare you grief in the long run. As we've discussed in *Chapter 3*, *Managing Storage Volumes*, LVM makes it very simple to expand a filesystem, especially on a virtual machine. If your database server is on a virtual machine, you can add a disk to the volume group and expand it if your database partition starts to get full, and your customers will never notice there was ever about to be a problem. Without LVM, you'll need to shut down the server, add a new volume, `rsync` your database server files over to the new location, and then bring up the server. Depending on the size of your database, this situation can span hours. Do yourself a favor, use LVM.

With that out of the way, we can begin setting up MariaDB. For learning and testing purposes, you can use pretty much any server you'd like, physical or virtual. Once you're ready, let's move on and we'll get started!

Installing MariaDB

Now we've come to the fun part, installing MariaDB. To get the ball rolling, we'll install the `mariadb-server` package:

```
# apt-get install mariadb-server
```

If your organization prefers to stick with MySQL, the package to install is `mysql-server` instead:

```
# apt-get install mysql-server
```

I don't recommend switching from MariaDB to MySQL (or vice versa) on the same server. I've seen some very strange configuration issues occur on servers that had one installed and then were switched to the other (even after wiping the configuration). For the most part, it's best to pick one and stick with it. As a general rule, MySQL should only be used if you have legacy databases to support. For brand-new installations, go with MariaDB.

www.it-ebooks.info

Going forward, I'll assume that you've installed MariaDB, though the instructions here shouldn't differ between them. Normally, you'd probably use the `systemctl` command at this point to investigate whether or not the MariaDB daemon has started and is running. I'll give you a bit of a spoiler: the daemon will indeed be started and enabled for you. However, the name of this daemon won't be quite what you expect:

```
# systemctl status mysql
```

No, that's not a misprint. The MariaDB daemon is called `mysql`. The old name is kept intact. So basically, whether you've installed the `mariadb-server` or `mysql-server`, the resulting service name will be the same. For now, let's stop the `mysql` daemon, since we have another step left:

```
# systemctl stop mysql
```

Next, we'll have MariaDB install all of its default database structure:

```
# mysql_install_db
```

There will be a great deal of text printed in your terminal, mostly about setting the `root` password. Don't worry about that now; we'll take care of setting the root password soon. For now, start the `mysql` daemon:

```
# systemctl start mysql
```

At this point, we officially have a fully functional database server. To connect to it and manage it, we'll use the `mysql` command. We'll be going over this later, but if you're curious what the MariaDB shell looks like, enter the `mysql` command as `root` that will take you right in:

```
# mysql
```

The `mysql` command works because the `mariadb-client` package was installed as a dependency when we installed `mariadb-server`, which is a utility that allows us to connect to databases. Entering the `mysql` command by itself with no options connects us to the database server on our local machine. This utility also lets us connect to external database servers to manage them remotely, which we'll discuss later. In the MariaDB shell, we can enter commands to view and manage our databases, as well as manage our database users. The MariaDB shell prompt will look like this:

```
MariaDB [(none)]>
```

We'll get into MariaDB commands and user management later. For now, you can exit the shell. To exit, you can type `exit` and press *Enter* or press *Ctrl* + *D* on your keyboard.

www.it-ebooks.info

Now that we've installed MariaDB, we should secure it. As `root`, enter the following command to begin the process:

```
# mysql_secure_installation
```

The first prompt will ask for your current `root` password:

```
Enter current password for root (enter for none):
```

We haven't set a `root` password yet, so just press *Enter*. Next, we'll be asked if we want to set a `root` password, which I recommend you do:

```
Set root password? [Y/n]
```

Simply press *Enter* to select the default (which is *Y*) and you'll be asked for the new root password. Choose and enter a secure root password and press *Enter*. You'll be asked to enter it a second time to confirm it. At the next prompt, you'll be asked if you want to remove anonymous users:

```
Remove anonymous users? [Y/n]
```

As you can see in the descriptive text that accompanies this prompt, MariaDB creates an anonymous user account, which is intended for testing, but shouldn't really exist on a production server. If you enter `Y` or accept the default and press *Enter*, this user will be removed:

```
Disallow root login remotely? [Y/n]
```

By default, MariaDB allows the `root` database user to connect remotely. The individual connecting remotely must know the root password in order to connect. However, if this is allowed, a miscreant can attempt to brute-force the server from a remote system. I can't think of a single situation in which allowing this is a good idea. If you answer `Y` here, the root MariaDB account will only be usable on the local machine. Disallowing the root login from remote machines is advised.

```
Remove test database and access to it? [Y/n]
```

Another testing feature that MariaDB includes by default is a database named `test`, which is accessible by all users. If you answer `Y` here, this database will be removed. We'll be creating our own database to practice with in this chapter, so there's no need for you to keep it around:

```
Reload privilege tables now? [Y/n]
```

Finally, you'll be asked if you'd like to reload privilege tables. Basically, anytime you make changes to users or database access, you should reload privileges. If you answer `Y` here, that will be done for you. I'll show you how to do this manually later, though.

www.it-ebooks.info

Now, our MariaDB is a bit more secure than it is by default. We still have more work to do on this server though, which we'll expand upon as we go along. While you can now move on to the next section, you might want to consider setting up another MariaDB server by following these steps on another machine. If you have room for another virtual machine, it might be a good idea for you to get this out of the way now, since we'll be setting up a slave database server later.

Taking a look at MariaDB configuration

Now that we have MariaDB installed, let's take a quick look at how its configuration is stored. While we won't be changing much of the configuration in this chapter (aside from adding parameters related to setting up a slave), it's a good idea to know where to find the configuration, since you'll likely be asked by a developer to tune the database configuration at some point in your career. This may involve changing the storage engine, buffer sizes, or countless other settings. A full walkthrough on performance tuning is outside the scope of this book, but it will be helpful to know how the settings for MariaDB are read, since Ubuntu's implementation is fairly unique.

The configuration files for MariaDB are stored in the `/etc/mysql` directory. Within that directory, you'll see the following files by default:

```
debian.cnf
debian-start
mariadb.cnf
my.cnf
my.cnf.fallback
```

You'll also see the following directories:

```
conf.d
mariadb.conf.d
```

The configuration file that MariadDB reads on startup is the `/etc/mysql/mariadb.cnf` file. This is where you'll begin pursuing when you want to configure the daemon, but we'll get to that soon. The `/etc/mysql/debian-start` file is actually a script that sets default values for MariaDB when it starts, such as setting some environment variables. It also defines a **SIGHUP** trap that is executed if the `mysql` process receives the **SIGHUP** signal, which will allow it to check for crashed tables.

www.it-ebooks.info

The `debian-start` script also loads the `/etc/mysql/debian.cnf` file, which sets some client settings for the `mysql` daemon. Here's a list of values from that file:

```
[client]
host     = localhost
user     = root
password =
socket   = /var/run/mysqld/mysqld.sock
[mysql_upgrade]
host     = localhost
user     = root
password =
socket   = /var/run/mysqld/mysqld.sock
basedir  = /usr
```

The defaults for these values are fine and there's rarely a reason to change them. Essentially, the file sets the default user, host, and socket location. If you've used MySQL before on other platforms, you may have seen many of those settings in the `/etc/my.cnf` file, which is typically the standard file for the `mysql` daemon. With MariaDB on Ubuntu Server, you can see that the default layout of files was changed considerably.

The `/etc/mysql/mariadb.cnf` file sets the global defaults for MariaDB. However, in Ubuntu's implementation, this default file just includes configuration files from the `/etc/mysql/conf.d` and the `/etc/mysql/mariadb.conf.d` directories. Within those directories, there are additional files ending with the `.cnf` extension. Many of these files contain default configuration values that would normally be found in a single file, but Ubuntu's implementation modularizes these settings into separate files instead. For our purposes in this book, we'll be editing the `/etc/mysql/conf.d/mysql.cnf` file, when it comes time to set up master and slave replication.

The other configuration files aren't relevant for the content of this book, and their current values are more than sufficient for what we need. When it comes to performance tuning, you may consider creating a new configuration file ending with the `.cnf` extension, with specific tuning values as provided by the documentation of a software package you want to run that interfaces with a database, or requirements given to you by a developer.

For additional information on how these configuration files are read, you can refer to the `/etc/mysql/mariadb.cnf` file, which includes some helpful contents at the top of the file that details the order in which these configuration files are read, as well as their purpose. Here's an excerpt of these comments from that file:

```
# The MariaDB/MySQL tools read configuration files in the following
order:
```


www.it-ebooks.info

```
# 1. "/etc/mysql/mariadb.cnf" (this file) to set global defaults,
# 2. "/etc/mysql/conf.d/*.cnf" to set global options.
# 3. "/etc/mysql/mariadb.conf.d/*.cnf" to set MariaDB-only options.
# 4. "~/.my.cnf" to set user-specific options.
```

As we can see, when MariaDB starts up, it first reads the `/etc/mysql/mariadb.cnf` file, followed `by.cnf` files stored within the `/etc/mysql/conf.d` directory, then the `.cnf` files stored within the `/etc/mysql/mariadb.conf.d` directory, followed by any user-specific settings stored within a `.my.cnf` file that may be present in the user's `home` directory.

With Ubuntu's implementation, when the `/etc/mysql/mariadb.cnf` file is read during startup, the process will immediately scan the contents of `/etc/mysql/conf.d` and `/etc/mysql/mariadb.conf.d`, because the `/etc/mysql/mariadb.cnf` file contains the following lines:

```
!includedir /etc/mysql/conf.d/
!includedir /etc/mysql/mariadb.conf.d/
```

As you can see, even though the order the configuration files are checked is set to check the `mariadb.cnf` file first followed by the `/etc/mysql/conf.d` and `/etc/mysql/mariadb.conf.d` directories, the `mariadb.cnf` file simply tells the `mysql` service to immediately check those directories.

This may be a bit confusing at first, because the default configuration for MariaDB in Ubuntu Server essentially consists of files that redirect to other files. But the main takeaway is that any configuration changes you make that are not exclusive to MariaDB (basically, configuration compatible with MySQL itself), should be placed within a configuration file the ends with the `.cnf` extension, and then stored within the `/etc/mysql/conf.d` directory. If the configuration you're wanting to add is for a feature exclusive to MariaDB (but not compatible with MySQL itself), the configuration file should be placed in the `/etc/mysql/mariadb.conf.d` directory instead. For our purposes, we'll be editing the `/etc/mysql/conf.d/mysql.cnf` file when it comes time to setting up our master/slave replication, since the method we'll be using is not specific to MariaDB.

In this section, we discussed MariaDB configuration and how it differs from its implementation in other platforms. The way the configuration files are presented is not the only difference in Ubuntu's implement of MariaDB; there are other differences as well. In the next section, we'll take a look at a few additional ways in which Ubuntu's implementation differs.

www.it-ebooks.info

Understanding how MariaDB differs in Ubuntu 16.04

Before we dive into managing our database server, we'll first go over a few ways in which MariaDB differs on Ubuntu 16.04 when compared to implementations on other distributions. For the most part, the differences are with regards to how authentication is handled to the MariaDB shell. The MariaDB shell is used to manage our database (which we'll get to shortly) and can be accessed with the `mysql` command:

```
# mysql
```

With that command, you'll immediately be given a MariaDB prompt, which will allow you to execute commands to manage your database configuration. However, you'll probably notice that the `mysql` command didn't prompt you for your `root` password. Instead, it immediately provided you with a MariaDB prompt. This may be surprising to those of you that have used MySQL before on other platforms. With other distributions, the default configuration will prompt you for the password for the root MariaDB shell, even if you're already logged in as the `root` Linux user on your server. On most implementations, you would execute a command such as the following to connect to the `root` MariaDB shell:

```
mysql -u root -p
```

That command first prompts you for your MariaDB root password, and then it provides you with a shell with which you can manage your server. That's still the case, but what's odd about Ubuntu's implementation is that you don't need the password in order to obtain access to the MariaDB prompt. In fact, when prompted, you can enter an invalid password on purpose and still be provided access anyway. The logic here is most likely that if you can access the `root` Linux user, you already have system-wide access anyway, so securing the `root` MariaDB shell from the `root` Linux user wouldn't benefit the system much. Still, this behavior is a bit peculiar if you're coming from another platform, such as Red Hat or even older versions of Debian or Ubuntu itself.

Another difference in Ubuntu 16.04's implementation of MariaDB is that normal Linux users are not able to access the MariaDB, even if you enter the correct password. For example, as a regular user, try the following command:

```
mysql -u root -p
```

No matter what you do, that command will not work unless you're logged into the server as `root` itself. You'll receive the following error:

```
ERROR 1698 (28000): Access denied for user 'root'@'localhost'
```


www.it-ebooks.info

I'm informing you about this to save you from frustration and help those of you who are familiar with the common behavior of standard MySQL. The takeaway is that, by default, you can only access the `root` MariaDB shell when you're logged in as the `root` Linux user, and standard users won't be able to access it. This is because MariaDB is configured to utilize UNIX sockets for authentication instead of using standard authentication. A walkthrough of UNIX sockets is beyond the scope of this book, but I'll show you how to return to the normal behavior later on.

For now, the recommended approach is to create another user within MariaDB to act as your administrative user. Once created, you can allow a database administrator access to this special MariaDB account, without giving him or her `root` access to the entire server. We'll create this administrative user in the next section.

Yet another difference in Ubuntu's MariaDB implementation is that standard authentication over the network will not work by default. Normally, you can use the following command to connect to the server from your workstation:

```
mysql -H <IP-address-of-host> -u <username> -p<password>
```

By default, that won't work. This is, again, due to UNIX sockets being used instead of standard authentication. I'll show you how to work around this later in this chapter.

Managing databases

Now that our MariaDB server is up and running (and we've covered some of the quirks regarding Ubuntu's implementation), we can finally look into managing it. In this section, I'll demonstrate how to connect to a database server using the `mysql` command, which will allow us to create databases, remove (drop) them, and also manage users and permissions.

At the end of the last section, I mentioned that the preferred method of managing databases on Ubuntu's implementation of MariaDB is to create an administrative user to be used in place of the `root` account, so we can start our discussion on database management with user management. To create this administrative user, we'll need to enter the MariaDB shell, which again, is simply a matter of executing the `mysql` command as `root`. Your prompt will change to the following:

```
MariaDB [(none)]>
```


www.it-ebooks.info

Now, we can create our new administrative user. I'll call mine `admin` in my examples, but you can use whatever name you'd like. In a company I used to work for, we used the user name `velociraptor` as our administrative user on our servers, since nothing is more powerful than a velociraptor. Feel free to use a clever name, but just make sure you remember it. Here's the command to create a new user in MariaDB (replace the user name and password in the command with your desired credentials):

```
CREATE USER 'admin'@'localhost' IDENTIFIED BY 'password';
FLUSH PRIVILEGES;
```

When it comes to MySQL syntax, the commands are not case-sensitive (though the data parameters are), but it's common to capitalize instructions to separate them from data. During the remainder of this chapter, we'll be executing some commands within the Linux shell, and others within the MariaDB shell. I'll let you know which shell each command needs to be executed in as we come to them, but if you are confused, just keep in mind that MariaDB commands are the only ones that are capitalized.

With the preceding commands, we're creating the `admin` user and restricting it to `localhost`. This is important because we don't want to open up the admin account to the world. We're also flushing privileges, which causes MariaDB to reload its privilege information. The `FLUSH PRIVILEGES` command should be run every time you add a user or modify permissions. I may not always mention the need to run this command, so you might want to make a mental note of it and make it a habit now.

As I mentioned, the previous command created the `admin` user but is only allowing it to connect from localhost. This means that an administrator would first need to log in to the server itself before he or she would be able to log in to MariaDB with the admin account. As an example of the same command (but allowing remote login from any other location), the following command is a variation that will do just that:

```
CREATE USER 'admin'@'%' IDENTIFIED BY 'password';
```

Can you see the percent symbol (`%`) in place of localhost? That basically means *everywhere*, which indicates we're creating a user that can be logged into from any source (even external nodes). By restricting our user to localhost with the first command, we're making our server just a bit more secure. You can also restrict access to particular networks, which is desired if you really do need to allow a database administrator access to the server remotely:

```
CREATE USER 'admin'@'192.168.1.%' IDENTIFIED BY 'password';
```


www.it-ebooks.info

That's a little better, but not as secure as limiting login to localhost. As you can see, the `%` character is basically a wildcard, so you can restrict access to needing to be from a specific IP or even a particular subnet.

So far, all we did is create a new user, we have yet to give this user any permissions. We can create a set of permissions (also known as **Grants**) with the `Grant` command. First, let's give our admin user full access to the DB server when called from localhost.

```
GRANT ALL PRIVILEGES ON *.* TO 'admin'@'localhost';
FLUSH PRIVILEGES;
```

Now, we have an administrative user we can use to manage our server's databases. We can use this account for managing our server instead of the `root` account. Any logged-on Linux user will be able to access the database server and manage it, provided they know the password. To access the MariaDB shell as the admin user that we created, the following command will do the trick:

```
mysql -u admin -p
```

After entering the password, you'll be logged into MariaDB as `admin`.

In addition, you can actually provide the password to the `mysql` command without needing to be prompted for it:

```
mysql -u admin -p<password>
```

Notice that there is no space in between the `-p` option and the actual password (though there is a space between `-u` and admin). This is normal and important. It's one of the quirks of MySQL and its brethren. Anyway, as useful as it is to provide the username and password in one shot, I don't recommend that you ever use it. This is because any Linux command you type is saved in the history, so anyone can view your command history and they'll see the password in plain text.

The admin account is only intended for system administrators who need to manage databases on the server. The password for this account should not be given to anyone other than staff employees or administrators that absolutely need it. Additional users can be added to the MariaDB server, each with differing levels of access. Keep in mind that our admin account can manage databases but not users. This is important, as you probably shouldn't allow anyone other than server administrators to create users. You'll still need to log in as `root` to manage user permissions.

www.it-ebooks.info

It may also be useful to create a read-only user for MariaDB for employees who need to be able to read data but not make changes. Back in the MariaDB shell (as `root`), we can issue the following command to create a read-only user:

```
GRANT SELECT ON *.* TO 'readonlyuser'@'localhost' IDENTIFIED BY
'password';
```

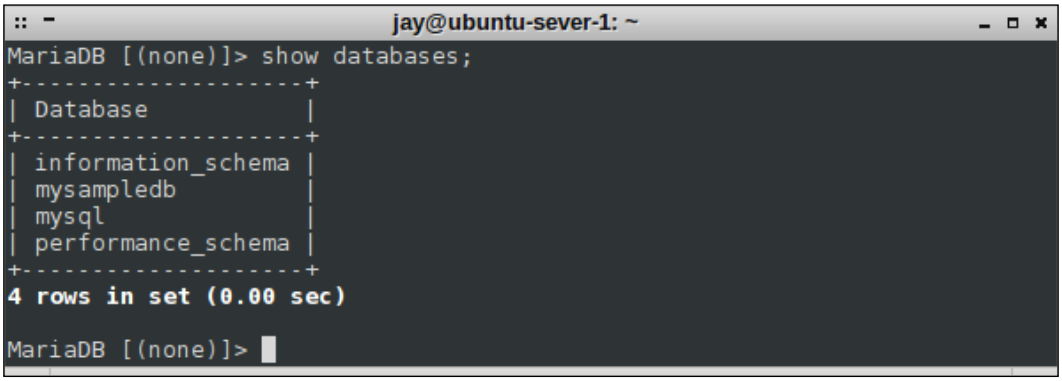
With this command, we've done two things. First, we created a new user and also set up `Grants` for that user with a single command. Second, we created a read-only user that can view databases but not manage them (we restricted the permissions to `SELECT`). This is more secure. In practice, it's better to restrict a read-only user to a specific database. This is typical in a development environment, where you'll have an application that connects to a database over the network, and needs to read information from it. We'll go over this scenario soon.

Next, let's create a database. At the MariaDB prompt, execute:

```
CREATE DATABASE mysampledb;
```

That was easy. We should now have a database on our server named `mysampledb`. To list all databases on our server (and confirm our database was created properly), we can execute the following command:

```
SHOW DATABASES;
```

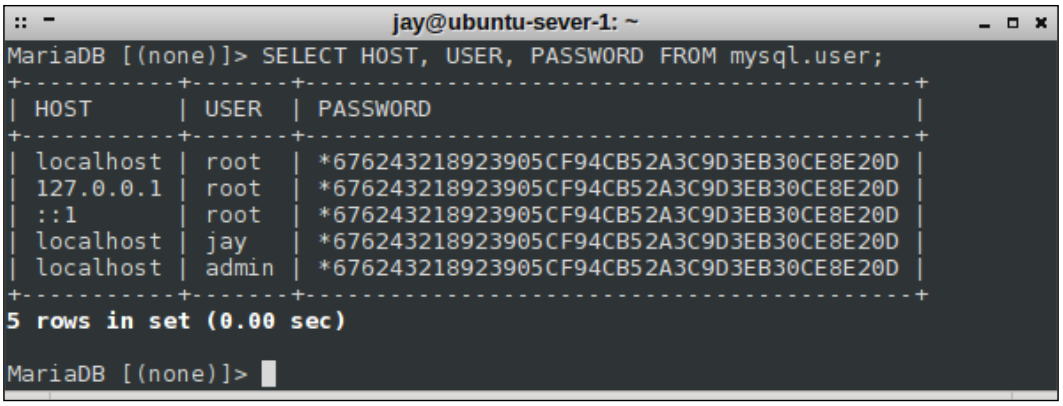


Showing a list of databases in MariaDB

The output will show some system databases that were created for us, but our new database should be listed among them. We can also list users just as easily:

```
SELECT HOST, USER, PASSWORD FROM mysql.user;
```


www.it-ebooks.info



Showing a list of MariaDB users

In a typical scenario, when installing an application that needs its own database, we'll create the database and then a user for that database. We'll normally want to give that user permission to only that database, with as little permission as required to allow it to function properly. We've already created the `mysampledb` database, so if we want to create a user with read-only access to it, we can do so with the following command:

```
GRANT SELECT ON mysampledb.* TO 'appuser'@'localhost' IDENTIFIED BY
'password';
```

With one command, we're not only creating the user `appuser`, but we're also setting a password for it, in addition to allowing it to have `SELECT` permissions on the `mysampledb` database. This is equivalent to read-only access. If our user needed full access, we could use the following instead:

```
GRANT ALL ON mysampledb.* TO 'appuser'@'localhost' IDENTIFIED BY
'password';
```

Now, our `appuser` has full access but only to the `mysampledb` database. Of course, we should only provide full access to the database if absolutely necessary. We can also provide additional permissions, such as `DELETE` (whether or not the user has access to delete data from the database), `CREATE` (which controls whether the user can add data to the database), `INSERT` (controls whether or not the user can add new rows to a database), `SELECT` (allows the user to read information from the database), `DROP` (allows the user to fully remove a database), and `ALL` (which gives the user everything). There are other permissions we can grant or deny, check the MariaDB documentation for more details. The types of permissions you'll need to grant to a user to satisfy the application you're installing will depend on the documentation for that software. Always refer to the installation instructions for the application you're attempting to install to determine which permissions are required for it to run.

www.it-ebooks.info

If you'd like to remove user access, you can use the following command to do so (substituting `myuser` with the user account you wish to remove and `host` with the proper host access you've previously granted the user):

```
DELETE FROM mysql.user WHERE user='myuser' AND host='localhost';
```

Now, let's go back to databases. Now that we've created the `mysampledb` database, what can we do with it? We'll add tables and rows, of course! A database is useless without actual data, so we can work through some examples of adding data to our database to see how this works. First, log in to the MariaDB shell as a user with full privileges to the `mysampledb` database. Now, we can have some fun and modify the contents. Here are some examples you can follow:

```
USE mysampledb;
```

The `USE` command allows us to select a database we want to work with. The MariaDB prompt will change from `MariaDB [(none)]>` to `MariaDB [mysampledb]>`. This is very useful, as the MariaDB prompt changes to indicate which database we are currently working with. We basically just told `MariaDB` that for all of the commands we're about to run, we would like them used against the `mysampledb` database.

Now, we can `CREATE` a table within our database. It doesn't matter what you call yours, since we're just practicing. I'll call mine `Employees`:

```
CREATE TABLE Employees (Name char(15), Age int(3), Occupation char(15));
```

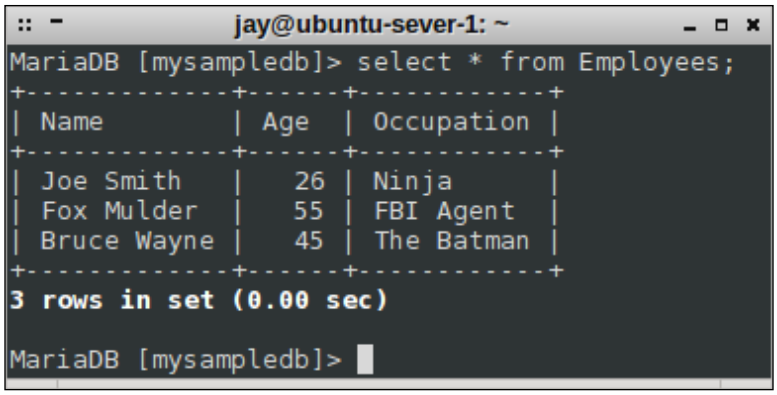
With this command, we've created a table named `Employees` that has three columns (`Name`, `Age`, and `Occupation`). To add new data to this table, we can use the following `INSERT` command:

```
INSERT INTO Employees VALUES ('Joe Smith', '26', 'Ninja');
```

The example `INSERT` command adds a new employee to our `Employees` table. When we use `INSERT`, we insert all the data for each of the columns. Here, we have an employee named `Joe`, who is `26` years old and whose occupation is a `Ninja`. Feel free to add additional employees, all you would need to do is formulate additional `INSERT` statements and provide data for each of the three fields. When you're done, you can use the following command to show all of the data within this table:

```
SELECT * FROM Employees;
```


www.it-ebooks.info



Viewing contents of the Employees table

To remove an entry, the following command will do what we need:

```
DELETE FROM Employees WHERE Name = 'Joe Smith';
```

Basically, we're using the `DELETE FROM` command, giving the name of the table we wish to delete from (`Employees`, in this case) and then using `WHERE` to provide some search criteria for narrowing down our command.

The `DROP` command allows us to delete tables or entire databases, and it should be used with care. I don't actually recommend you delete the database we just created, since we'll use it for additional examples. But if you really wanted to drop the `Employees` table, you could use:

```
DROP TABLE Employees;
```

Or to drop the entire database:

```
DROP DATABASE mysampledb;
```

There is, of course, much more to MariaDB and its MySQL syntax than the samples I provided, but this should be enough to get you through the examples in this book. As much as I would love to give you a full walkthrough of the MySQL syntax, it would easily push this chapter beyond 50 pages. If you'd like to push your skills beyond the samples of this chapter, there are great books available on the subject.

Before I close this section though, I think it will be worthwhile for you to see how to back up and restore your databases. To do this, we have the `mysqldump` command at our disposal. Its syntax is very simple, as you'll see. First, exit the MariaDB shell and return to your standard Linux shell. Since we've already created an `admin` user earlier in the chapter, we'll use that user for the purposes of our backup:

```
mysqldump -u admin -p --databases mysampledb > mysampledb.sql
```


www.it-ebooks.info

With this example, we're using `mysqldump` to create a copy of the `mysampledb` database and storing it in a file named `mysampledb.sql`. Since MariaDB requires us to log in, we authenticate to MariaDB using the `-u` option with the username admin and the `-p` option that will prompt us for a password. The `--databases` option is necessary because, by default, `mysqldump` does not include the database create statement nor the tables. However, the `--databases` option forces this, which just makes it easier for you to restore. Assuming that we were able to authenticate properly, the contents of the `mysampledb` database will be dumped into the `mysampledb.sql` file. This export should happen very quickly, since this database probably only contains a single table and a few rows. Larger production databases can take hours to dump.

Restoring a backup is fairly simple. We can utilize the `mysql` command with the backup file used as a source of input:

```
# mysql < mysampledb.sql
```

If we have UNIX sockets disabled and are using standard authentication for MariaDB, the command to import the database becomes the following:

```
mysql -u admin -p < mysampledb.sql
```

So, there you have it. The `mysqldump` command is definitely very handy in backing up databases. In the next section, we'll work through setting up a slave database server.

Setting up a slave DB server

Earlier in this chapter, we discussed installing MariaDB on a server. To set up a slave, all you really need to begin the process is set up another database server. If you've already set up two database servers, you're ready to begin. If not, feel free to spin up another VM and follow the process from earlier in this chapter that covered installing MariaDB. Go ahead and set up another server if you haven't already done so. Of your two servers, one should be designated as the master and the other the slave, so make a note of the IP addresses for each.

To begin, we'll first start working on the master. We'll need to edit the `/etc/mysql/conf.d/mysql.cnf` file. Currently, the file contains just the following line:

```
[mysql]
```


www.it-ebooks.info

Right underneath that, add a blank line and then the following code:

```
[mysqld]
log-bin
binlog-do-db=mysampledb
server-id=1
```

With this configuration, we're first enabling bin-logging, which is required for a master/slave server to function properly. These binary logs record changes made to a database, which will then be transferred to a slave.

Another configuration file that we'll need to edit is `/etc/mysql/mariadb.conf.d/50-server.cnf`. In this file, we have the following line:

```
bind-address = 127.0.0.1
```

With this default setting, the `mysql` daemon is only listening for connections on localhost (`127.0.0.1`), which is a problem since we'll need to connect to it from another machine (the slave). Change this line to the following:

```
bind-address = 0.0.0.0
```

Next, we'll need to access the MariaDB shell on the master, and execute the following commands:

```
GRANT REPLICATION SLAVE ON *.* to 'replicate'@'192.168.1.204' identified
by 'slavepassword';
```

Here, we're creating a replication user named `replicate` and allowing it to connect to our primary server from the IP address `192.168.1.204`. Be sure to change that IP to match the IP of your slave, but you can also use a hostname identifier such as `%.mydomain` if you have a domain configured, which is equivalent to allowing any hostname that ends with `.mydomain`. Also, we're setting the password for this user to `slavepassword`, so feel free to customize that as well to fit your password requirements (be sure to make a note of the password).

Given the fact that the default implementation of MariaDB utilizes Unix socket authentication instead of standard authentication, this can become a problem when setting up master and slave replication. So it's best to disable it and return to the standard approach. To do so, execute the following commands on the master server from within the MariaDB shell:

```
USE mysql;
UPDATE `user` SET `plugin` = '' WHERE `User` = 'root';
FLUSH PRIVILEGES;
```


www.it-ebooks.info

We should now restart the `mysql` daemon so that the changes we've made to the `mysql.cnf` file take effect:

```
# systemctl restart mysql
```

Before we continue, you'll probably notice that now that we've disabled UNIX socket authentication, you can no longer connect to the MariaDB shell with just the `mysql` command by itself. You'll need to provide the MariaDB root password when connecting to the `root` account. Again, here's the command to access the MariaDB shell as `root`:

```
mysql -u root -p
```

From this point forward, when we work with the master server, we'll need to log in with that method (or you can use the `admin` account we created earlier).

Anyway, back to setting up our master/slave replication. Next, we'll set up the slave server. But before we do that, there's a consideration to make now that will possibly make the process easier on us. In a production environment, it's very possible that data is still being written to the master server. The process of setting up a slave is much easier if we don't have to worry about the master database changing while we set up the slave. The following command, when executed within the MariaDB shell, will lock the database and prevent additional changes:

```
FLUSH TABLES WITH READ LOCK;
```

If you're absolutely sure that no data is going to be written to the master, you can disregard that step.

Next, we should utilize `mysqldump` to make both the master and the slave contain the same data before we start synchronizing them. The process is smoother if we begin with them already synchronized, rather than trying to mirror the databases later. Using `mysqldump` as we did in the previous section, create a dump of the master server's database and then import that dump into the slave. The easiest way to transfer the dump file is to use `rsync` or `scp`. Then, on the slave, use `mysql` to import the file.

Since we configured the server to use standard authentication for MariaDB, the command to back up the database on the master becomes the following:

```
mysqldump -u admin -p --databases mysampledb > mysampledb.sql
```

After transferring the `mysampledb.sql` file to the slave, you can import the backup into the slave server:

```
mysql -u root -p < mysampledb.sql
```


www.it-ebooks.info

Back on the slave, we'll need to edit the `/etc/mysql/conf.d/mysql.cnf` and then place the following code at the end:

```
[mysqld]
server-id=2
```

Make sure you restart the `mysql` process on the slave before continuing:

```
# systemctl restart mysql
```

From the `root` MariaDB shell on your slave, enter the following command. Change the IP address within the command accordingly:

```
CHANGE MASTER TO MASTER_HOST="192.168.1.184", MASTER_USER='replicate',
MASTER_PASSWORD='slavepassword';
```

Now that we're finished configuring the synchronization, we can unlock the master's tables. On the master server, execute the following command within the MariaDB shell:

```
UNLOCK TABLES;
```

Now, we can check the status of the slave to see whether or not it is running. Within the slave's MariaDB shell, execute the following command:

```
SHOW SLAVE STATUS\G;
```

Assuming all went well, we should see the following line within the output:

```
Slave_IO_State: Waiting for master to send event
```

If the slave isn't running (`Slave_IO_State` is blank), execute the following command:

```
START SLAVE;
```

From this point forward, any data you add to your database on the master should be replicated to the slave. To test, add a new record to the `Employees` table on the `mysampledb` database on the master server:

```
USE mysampledb;
INSERT INTO Employees VALUES ('Optimus Prime', '100', 'Transformer');
```

On the slave, check the same database and table for the new value to appear. It may take a second or two:

```
USE mysampledb;
SELECT * FROM Employees;
```


www.it-ebooks.info

If you see any errors within the `Slave_IO_State` line when you run `SHOW SLAVE STATUS\G`; or your databases aren't synchronizing properly, here are a few things you can try. First, make sure that the database master is listening for connections on `0.0.0.0` port `3306`. To test, run this variation of the `netstat` command to see which port the `mysql` process is listening on:

```
# netstat -tulpn |grep mysql
```

The output should be similar to the following:

```
tcp        0      0 0.0.0.0:3306            0.0.0.0:*
LISTEN      946/mysqld
```

If you see that the service is listening on `127.0.0.1:3306` instead, that means it's only accepting connections from localhost. Earlier in this section, I mentioned changing the bind-address in the `/etc/mysql/mariadb.conf.d/50-server.cnf` file. Make sure you've already done that and restart `mysql`. During my tests, I've actually had one situation where the `mysql` service became locked after I made this change, and attempting to restart the process did nothing (I ended up having to reboot the entire server, which is not typically something you'd have to do). Once the server came back up, it was listening for connections from the network.

If you receive errors on the slave when you run `SHOW SLAVE STATUS\G`; with regards to authentication, make sure you've run `FLUSH PRIVILEGES`; on the master. Even if you have, run it again to be sure. Also, double check that you're synchronizing with the correct username, IP address, and password. For your convenience, here's the command we've run on the master to grant replication permissions:

```
GRANT REPLICATION SLAVE ON *.* to 'replicate'@'192.168.1.204' identified
by 'slavepassword';
FLUSH PRIVILEGES
```

Here's the command that we've run on the slave:

```
CHANGE MASTER TO MASTER_HOST="192.168.1.184", MASTER_USER='replicate',
MASTER_PASSWORD='slavepassword';
```

Finally, make sure that your master database and the slave database both contain the same databases and tables. The master won't be able to update a database on the slave if it doesn't exist there. Flip back to my example usage on `mysqldump` if you need a refresher. You should only need to use `mysqldump` and import the database onto the slave once, since after you get the replication going, any changes made to the database on the master should follow over to the slave. If you have any difficulty with the `mysqldump` command, you can manually create the `mysampledb` and the `Employees` table on the slave, which is really all it needs for synchronization to start.

www.it-ebooks.info

Synchronization should then begin within a minute, but you can execute `STOP SLAVE`; followed by `START SLAVE`; on the slave server to force it to try to synchronize again without waiting.

And that should be all there is to it. At this point, you should have fully functional master and slave servers at your disposal. To get additional practice, try adding additional databases, tables, users, and insert new rows into your databases.

Summary

Depending on your skillset, you're either an administrator that is learning about SQL databases for the first time or you're a seasoned veteran who is curious how to implement a database server with Ubuntu Server. With the recent changes in Ubuntu Server as provided by its upstream database packages, there are quite a few differences on this platform when compared to MySQL and MariaDB implementations even on other platforms (especially in regards to authentication). In this chapter, we dove into Ubuntu's implementation of this technology, and worked through setting up our own database server. We also worked through some examples of the MySQL syntax, such as creating databases, as well as setting up users and their grants. We also worked through setting up a master and slave server for replication.

Database administration is a very vast topic, and we've only scratched the surface here. Being able to manage MySQL and MariaDB databases is a very sought-after skill for sure. If you haven't worked with these databases before, this chapter will serve as a good foundation for you to start your research.

In the next chapter, we'll use our database server to act as a foundation for OwnCloud, which we will set up as part of our look into setting up a web server. When you've finished practicing these database concepts, head on over to *Chapter 10*, *Serving Web Content*, where we'll journey into Apache.

www.it-ebooks.info

www.it-ebooks.info

10
Serving Web Content

The flexible nature of Ubuntu Server makes it an amazing platform to host your organization's web presence. In this chapter, we'll take a look at Apache, which is the leading web server software on the Internet. Ubuntu's implementation of Apache is very solid, and includes many useful plug-ins that can be used to extend it further. We'll go through installing, configuring, and extending this popular hosting solution, as well as securing it with SSL. In addition, we'll also take a look at installing ownCloud, which is a great solution for setting up your very own cloud environment for your organization to collaborate on and share files. As we work through concepts related to hosting web content on Ubuntu Server, we will cover:

- Installing and configuring Apache
- Installing additional Apache modules
- Securing Apache with SSL
- Setting up high availability with keepalived
- Installing and configuring ownCloud

Installing and configuring Apache

The best way to become familiar with any technology is to dive right in. We'll begin this chapter by installing Apache, which is simply a matter of installing the `apache2` package:

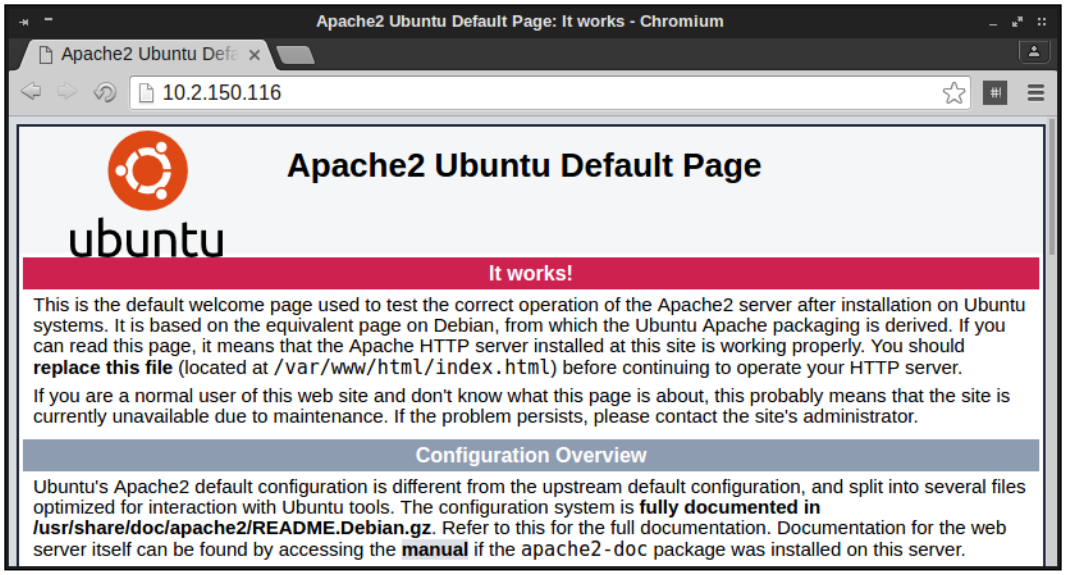
```
# apt-get install apache2
```

By default, Ubuntu will immediately start and enable the `apache2` daemon as soon as its package is installed. You can confirm this yourself with either of the two following commands, the latter of which you would use on older Ubuntu server installations:

```
# systemctl status apache2
# /etc/init.d/apache2 status
```


www.it-ebooks.info

In fact, at this point, you already have (for all intents and purposes) a fully functional web server. If you were to open a web browser and enter the IP address of the server you just installed Apache on, you should see Apache's sample web page:



Apache's default web page in Ubuntu Server

Of course, there's more to Apache than simply installing it. There are several configuration files in the `/etc/apache2` directory that govern how sites are hosted, as well as which directories Apache will look in to find web pages to host.

The directory that Apache serves web pages from is known as a **Document Root**, with `/var/www/html` being the default. Inside that directory, you'll see an `index.html` file, which is actually the default page you see when you visit an unmodified Apache server. Essentially, this is a test page that was designed to show you that the server is working, as well as some tidbits of information regarding the default configuration.

www.it-ebooks.info

You're not limited to hosting just one website on a server, though. Apache supports the concept of a **Virtual host**, which allows you to serve multiple websites from a single server. Each virtual host would consist of an individual configuration file, which differentiate themselves based on either name or IP address. For example, you could have an Apache server with a single IP address that hosts two different websites, such as `acmeconsulting.com` and `acmesales.com`. To set this up, you would create a separate configuration file for `acmeconsulting.com` and `acmesales.com` stored in your Apache configuration directory. Each configuration file would include a `<VirtualHost>` stanza, where you would place an identifier such as a name or IP address that differentiates one from the other. When a request comes in, Apache will serve either `acmeconsulting.com` or `acmesales.com` to the user's browser, depending on which criteria matched when the request came in. The configuration files for each site typically end with the `.conf` filename extension, and are stored in the `/etc/apache2/sites-available` directory, which we'll come back to in just a moment.

The basic workflow for setting up a new site (virtual host) will typically be similar to the following:

- The web developer creates the website and related files
- These files are uploaded to the Ubuntu Server, typically in a subdirectory of `/var/www` or another directory the administrator has chosen
- The server administrator creates a configuration file for the site, and copies it into the `/etc/apache2/sites-available` directory
- The administrator enables the site and reloads Apache

Enabling virtual hosts is handled a bit differently in Debian and Ubuntu than in other platforms. In fact, there are two specific commands to handle this purpose: `a2ensite` for enabling a site and `a2dissite` for disabling a site. Assuming a site with the URL of `acmeconsulting.com` is to be hosted on our Ubuntu Server, we would copy its files to `/var/www/acmeconsulting`, create the `/etc/apache2/sites-available/acmeconsulting.conf` configuration file, and enable the site with the following commands:

```
# a2ensite acmeconsulting.conf
# systemctl reload apache2
```

I'm not using absolute paths in my examples; as long as you've copied the configuration file to the correct place, the `a2ensite` and `a2dissite` commands will know where to find it.

www.it-ebooks.info

If we wanted to disable the site for some reason, we would execute the `a2dissite` command against the site's configuration file:

```
# a2dissite mysite.conf
# systemctl reload apache2
```

On older servers without the `systemctl` command, use the following to reload Apache:

```
# service apache2 reload
```

If you're curious how this works behind the scenes, when the `a2ensite` command is run against a configuration file, it basically creates a symbolic link to that file and stores it in the `/etc/apache2/sites-enabled` directory. When you run `a2dissite` to disable a site, this symbolic link is removed. Apache, by default, will use any configuration files it finds in the `/etc/apache2/sites-enabled` directory. After enabling or disabling a site, you'll need to refresh Apache's configuration, which is where the reload option comes in. This command won't restart Apache itself (so users who are using your existing sites won't be disturbed) but it does give Apache a chance to reload its configuration files. If you replaced reload with restart on the preceding commands, Apache will perform a full restart. You should only need to do that if you're having an issue with Apache or enabling a new plug-in, but in most cases the reload option is preferred on a production system.

The main configuration file for Apache is located at `/etc/apache2/apache2.conf`. Feel free to view the contents of this file; the comments contain a good overview of how Apache's configuration is laid out. The following lines within this file are of special interest:

```
# Include the virtual host configurations:
IncludeOptional sites-enabled/*.conf
```

As you can see, this is how Ubuntu has configured Apache to look for enabled sites within the `/etc/apache2/sites-enabled` directory. Any file stored there with the `.conf` file extension is read by Apache. If you wish, you could actually remove those lines and Apache would then behave as it does on other platforms, and the `a2ensite` and `a2dissite` commands would no longer have any purpose. However, it's best to keep the framework of Ubuntu's implementation intact, as separating the configuration files makes logical sense, and helps simplify the configuration. This chapter will go along with the Ubuntu way of managing configuration.

www.it-ebooks.info

An additional virtual host is not required if you're only hosting a single site. The contents of `/var/www/html` are served by the default virtual host if you make no changes to Apache's configuration. This is where the example site that ships with Apache comes from. If you only need to host one site, you could remove the default `index.html` file stored within this directory and replace it with the files required by your website. If you wish to test this for yourself, you can make a backup copy of the default `index.html` file, and create a new one with some standard HTML. You should see the default page change to feature the content you just added to the file.

The `000-default.conf` file is special, in that it's basically the configuration file that controls the default Apache sample website. If you look at the contents of the `/etc/apache2/sites-available` and the `/etc/apache2/sites-enabled` directories, you'll see the `000-default.conf` configuration file stored in `sites-available`, and `symlinked` in `sites-enabled`. This shows you that, by default, this site was included with Apache, and its configuration file was enabled as soon as Apache was installed. For all intents and purposes, the `000-default.conf` configuration file is all you need if you only plan on hosting a single website on your server. The contents of this file are as follows, but I've stripped the comments out of the file in order to save space on this page:

```
<VirtualHost *:80>
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/html

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

As you can see, this default virtual host is telling Apache to listen on port `80` for requests and to serve content from `/var/www/html` as soon as requests come in. The `<VirtualHost>` declaration at the beginning is listening to everything (the asterisk is a *wildcard*) on port `80`, so this is basically handling all web traffic that comes in to the server from port `80`. The `ServerAdmin` clause specifies the e-mail address that is displayed in any error messages shown in case there is a problem with the site.

The `DocumentRoot` setting tells Apache which directory to look for in order to find files to serve for connections to this virtual host. `/var/www/html` is the default, but some administrators choose to customize this. This file also contains lines for where to send logging information. The **access log** contains information relating to HTTP requests that come in, and by default is stored in `/var/log/access.log`. The **error log** is stored at `/var/log/error.log`, and contains information you can use whenever someone has trouble visiting your site. The `${APACHE_LOG_DIR}` variable equates to `/var/log` by default, and this is set in the `/etc/apache2/envvars` file, in case for some reason you wish to change this.

www.it-ebooks.info

If you wish to host another site on the same server by creating an additional virtual host, you can use the same framework as the original file, with some additional customizations. Here's an example for a hypothetical website, `acmeconsulting.com`:

```
<VirtualHost 192.168.1.104:80>
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/acmeconsulting

    ErrorLog ${APACHE_LOG_DIR}/acmeconsulting-error.log
    CustomLog ${APACHE_LOG_DIR}/acmeconsulting-access.log combined
</VirtualHost>
```

I've emphasized some important differences within this file. First, with this virtual host, I'm not listening for all connections coming in on port `80`, instead I'm specifically looking for incoming traffic going to IP address `192.168.1.104` on port `80`. This works because this server has two network cards, and therefore two IP addresses. With virtual hosts, I'm able to serve a different website, depending on which IP address the request is coming in on.

Next, I set the `DocumentRoot` to `/var/www/acmeconsulting`. Each virtual host should have their own individual `DocumentRoot` to keep each site separate from one another. On my servers, I will typically disable or remove the sample virtual host (the one that has the default `DocumentRoot` of `/var/www/html`). Instead, I use `/var/www` as a base directory, and each virtual host gets their own directory as a subdirectory of this base.

Another change I find useful is to give each virtual host its own log files. Normally, Apache will use `/var/log/apache2/error.log` and `/var/log/apache2/access.log` to store log entries for all sites. If you only have a single site on your server, that is fine. However, when you're serving multiple sites, I find it useful to give each site their own independent log files. That way, if you are having trouble with a particular site, you don't have to scroll through unrelated log entries to find what you're looking for when you're troubleshooting. In my example, I inserted the website name within the log file names, so this virtual host is logging errors in the `/var/log/apache2/acmeconsulting-error.log`, and the access log is being written to `/var/log/apache2/acmeconsulting-access.log`. These log files will be created for you automatically as soon as you reload Apache.

www.it-ebooks.info

With a server that only has a single IP address, you can still set up multiple virtual hosts. Instead of differentiating virtual hosts by IP, you can instead differentiate them by name. This is common on **Virtual Private Server** (**VPS**) installations of Ubuntu, where you'll typically have a single IP address assigned to you by your VPS provider. For name-based virtual hosts, we would use the `ServerName` option within our configuration. Refer to the following example to see how this would work. With this example, I'm adding name-based virtual hosts to their own file. I called mine `000-virtual-hosts.conf` and stored it in the `/etc/apache2/sites-available` directory. The contents are as follows:

```
<VirtualHost *:80>
    ServerName acmeconsulting.com
    DocumentRoot /var/www/acmeconsulting
</VirtualHost>

<VirtualHost *:80>
    ServerName acmesales.com
    DocumentRoot /var/www/acmesales
</VirtualHost>
```

For each virtual host, I'm declaring a `ServerName` with a matching `DocumentRoot`. With the first example, any traffic coming into the server requesting `acmeconsulting.com` will be provided a document root of `/var/www/acmeconsulting`. The second looks for traffic from `acmesales.com` and directs it to `/var/www/acmesales`. You can list as many virtual hosts here as you'd like to host on your server. Providing your server has enough resources to handle traffic to each site, you can host as many as you need.

As we continue through this chapter, we'll perform some additional configuration for Apache. At this point though, you should have an understanding of the basics of how Apache is configured in Ubuntu Server. For extra practice, feel free to create additional virtual hosts and serve different pages for them. The easiest way to test virtual hosts is with a virtual machine. Software such as VirtualBox allows you to set up virtual machines with multiple network interfaces. If you set each one to be a bridged connection, they should get an IP address from your local DHCP server, and you can then create virtual hosts to serve different websites depending on which interface the request comes in on.

www.it-ebooks.info

Installing additional Apache modules

Apache features additional modules that can be installed that will extend its functionality. These modules can provide additional features such as adding support for Python or PHP. Ubuntu's implementation of Apache includes two specific commands for enabling and disabling mods, `a2enmod` and `a2dismod`, respectively. Apache modules are generally installed via packages from Ubuntu's repositories. To see a list of modules available for Apache, run the following command:

```
aptitude search libapache2-mod
```

In case you don't have `aptitude` installed, you can instead run the following command:

```
apt-cache search libapache2-mod
```

Within the results, you'll see various module packages available, such as `libapache2-python` (which adds Python support) and `libapache2-mod-php7.0` (which adds PHP 7 support), among many others. Installing an Apache module is done the same way as any other package, with the `apt-get` or `aptitude` commands. In the case of PHP support, we can install the required package with the following command:

```
# apt-get install libapache2-mod-php7.0
```

Installing a module package alone is not enough for a module to be usable within Apache, though. Modules must be enabled in order for Apache to be able to utilize them. Again, we can use the `a2enmod` and `a2dismod` commands for enabling or disabling a module. If you wish to simply view a list of modules that Apache has available, the following command examples will help:

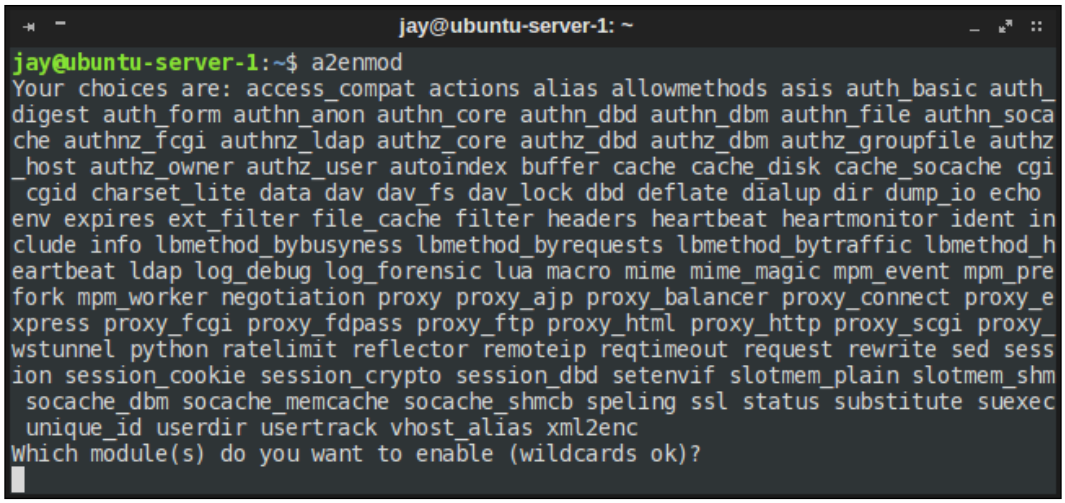
You can view a list of modules that are built-in to Apache with the following command:

```
apache2 -l
```

The modules shown in the output will be those that are built-in to Apache, so you won't need to enable them. If the module your website requires is listed in the output, you're all set.

To view a list of all modules that are installed and ready to be enabled, you can run the `a2enmod` command.

www.it-ebooks.info



The a2enmod command showing a list of available Apache modules

The end of the output of the `a2enmod` command will ask you whether or not you'd like to enable any of the modules:

```
Which module(s) do you want to enable (wildcards ok)?
```

If you wanted to, you could type the names of any additional modules you'd like to enable and then press *Enter*. Alternatively, you can press *Enter* without typing anything to simply return to the prompt.

If you give the `a2enmod` command a module name as an option, it will enable it for you. For enabling PHP 7 (which we'll need later) you can run the following command:

```
# a2enmod php7.0
```

Chances are though, if you've installed a package for an additional module, it was most likely was enabled for you during installation. With Debian and Ubuntu, it's very common for daemons and modules to be enabled as soon as their packages are installed, and Apache is no exception. In the case of the `libapache2-mod-php7.0` package I used as an example, the module should've been enabled for you once the package was installed:

```
# a2enmod php7.0
Module php7.0 already enabled
```


www.it-ebooks.info

If the module wasn't already enabled, we would see the following output:

```
Enabling module php7.0.
To activate the new configuration, you need to run:
service apache2 restart
```

As instructed, we'll need to restart Apache in order for the enabling of a module to take effect. The same also holds true when we disable a module as well. Disabling modules works pretty much the same way, you'll use the `a2dismod` command along with the name of the module you'd like to disable:

```
# a2dismod php7.0
Module php7.0 disabled.
To activate the new configuration, you need to run:
service apache2 restart
```

The modules you install and enable on your Apache server will depend on the needs of your website. For example, if you're going to need support for Python, you'll want to install the `libapache2-mod-python` package. If you're installing a third-party package, such as WordPress or Drupal, you'll want to refer to the documentation for those packages in order to obtain a list of which modules are required for the solution to install and run properly. Once you have such a list, you'll know which packages you'll need to install and which modules to enable.

Securing Apache with SSL

Nowadays, it's a great idea to ensure your organization's website is encrypted and available over HTTPS. Encrypting your web traffic is not that hard to do, and will help protect your organization against common exploits. Utilizing SSL doesn't protect you from all exploits being used in the wild, but it does offer a layer of protection you'll want to benefit from. In this section, we'll look at how to use SSL with our Apache installation. We'll work through enabling SSL, generating certificates, and configuring Apache to use those certificates with both a single site configuration, as well as with virtual hosts.

By default, Ubuntu's Apache configuration listens for traffic on port `80`, but not port `443` (HTTPS). You can check this yourself by running the following command:

```
# netstat -tulpn |grep apache
```


www.it-ebooks.info

The results will show the ports that Apache is listening on, which is only port `80` by default:

```
tcp6       0      0 :::80      :::*          LISTEN      2791/apache2
```

If the server were listening on port `443` as well, we would've seen the following output instead:

```
tcp6       0      0 :::80       :::*     LISTEN      3257/apache2
tcp6       0      0 :::443      :::*         LISTEN      3257/apache2
```

To enable support for HTTPS traffic, we need to first enable the `ssl` module:

```
# a2enmod ssl
```

Next, we need to restart Apache:

```
# systemctl restart apache2
```

In addition to the sample website we discussed earlier, Ubuntu's default Apache implementation also includes another site configuration file, `/etc/apache2/sites-available/default-ssl.conf`. Unlike the sample site, this one is not enabled by default. This configuration file is similar to the sample site configuration but it's listening for connections on port `443` and contains additional configuration items related to SSL. Here's the content of that file, with the comments stripped out in order to save space:

```
<IfModule mod_ssl.c>
        <VirtualHost _default_:443>
                ServerAdmin webmaster@localhost

                DocumentRoot /var/www/html

                ErrorLog ${APACHE_LOG_DIR}/error.log
                CustomLog ${APACHE_LOG_DIR}/access.log combined

                SSLEngine on

        SSLCertificateFile      /etc/ssl/certs/ssl-cert-snakeoil.pem
        SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key

                <FilesMatch "\.(cgi|shtml|phtml|php)$">
                                SSLOptions +StdEnvVars
                </FilesMatch>
                <Directory /usr/lib/cgi-bin>
                                SSLOptions +StdEnvVars
```


www.it-ebooks.info

```
                </Directory>
                BrowserMatch "MSIE [2-6]" \
                               nokeepalive ssl-unclean-shutdown \
                               downgrade-1.0 force-response-1.0
                # MSIE 7 and newer should be able to use keepalive
                BrowserMatch "MSIE [17-9]" ssl-unclean-shutdown

        </VirtualHost>
</IfModule>
```

We've already gone over the `ServerAdmin`, `DocumentRoot`, `ErrorLog`, and `CustomLog` options earlier in this chapter, but there are additional options within this file that we haven't seen yet. On the first line, we can see that this virtual host is listening on port `443`. We also see `_default_` listed here instead of an IP address. The `_default_` option only applies to unspecified traffic, which in this case means any traffic coming in to port `443` that hasn't been identified in any other virtual host. In addition, the `SSLEngine` on option enables SSL traffic. Right after that, we have options for our SSL certificate file and certificate, which we'll get to a bit later.

We also have a `<Directory>` clause, which allows us to apply specific options to a directory. In this case, the `/usr/lib/cgi-bin` directory is being applied to the `SSLOptions +StdEnvVars` setting, which enables default environment variables for use with SSL. This option is also applied to files that have an extension of `.cgi`, `.shtml`, `.phtml`, or `.php` through the `<FilesMatch>` option. The `BrowserMatch` option allows you to set options for specific browsers, though it's out of scope for this chapter. For now, just keep in mind that if you want to apply settings to specific browsers, you can.

By default, the `default-ssl.conf` file is not enabled. In order to benefit from its configuration options, we'll need to enable it, which we can do with the `a2ensite` command as we would with any other virtual host:

```
# a2ensite default-ssl.conf
```

Even though we just enabled SSL, our site isn't secure just yet. We'll need SSL certificates installed in order to secure our web server. We can do this in one of two ways, with self-signed certificates, or certificates signed by a certificate authority for a fee. Both are implemented in very similar ways, and I'll discuss both methods. For the purposes of testing, self-signed certificates are fine. In production, self-signed certificates would be sufficient in theory, but most browsers won't trust them by default, and will give you an error when you go to their page. Users of a site with self-signed certificates would need to bypass an error page before continuing to the site, and seeing this error may cause them to avoid your site altogether. You can install the certificates into each user's web browser, but that can be a headache. In production, it's best to use certificates signed by a vendor.

www.it-ebooks.info

As we go through this process, I'll first walk you through setting up SSL with a self-signed certificate so you can see how the process works. We'll create the certificate, and then install it into Apache. You won't necessarily need to create a website to go through this process, since you could just secure the sample website that comes with Apache if you wanted something to use as a proof of concept. After we complete the process, we'll take a look at installing certificates that were signed by a certificate authority.

To get the ball rolling, we'll need a directory to house our certificates. I'll use `/etc/apache2/certs` in my examples, though you can use whatever directory you'd like, as long as you remember to update Apache's configuration with your desired location and filenames:

```
# mkdir /etc/apache2/certs
```

For a self-signed certificate and key, we can generate the pair with the following command. Feel free to change the name of the key and certificate files to match the name of your website:

```
# openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/
apache2/certs/mysite.key -out /etc/apache2/certs/mysite.crt
```

You'll be prompted to enter some information for generating the certificate. Answer each prompt as they come along. Here's a list of the questions you'll be asked, along with my responses for each. Change the answers to fit your server, environment, organization name, and location:

```
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:Michigan
Locality Name (eg, city) []:Detroit
Organization Name (eg, company) [Internet Widgits Pty Ltd]:My Company
Organizational Unit Name (eg, section) []:IT
Common Name (e.g. server FQDN or YOUR name) []:myserver.mydomain.com
Email Address []:webmaster@mycompany.com
```

Now, you should see that two files have been created in the `/etc/apache2/certs` directory, `mysite.crt` and `mysite.key`, which represent the certificate and private key respectively. Now that these files have been generated, the next thing for us to do is to configure Apache to use them. Look for the following two lines in the `/etc/apache2/sites-available/default-ssl.conf` file:

```
SSLCertificateFile      /etc/ssl/certs/ssl-cert-snakeoil.pem
SSLCertificateKeyFile  /etc/ssl/private/ssl-cert-snakeoil.key
```


www.it-ebooks.info

Comment these lines out by placing a # symbol in front of both:

```
# SSLCertificateFile      /etc/ssl/certs/ssl-cert-snakeoil.pem
# SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key
```

Next, add the following two lines underneath the lines you just commented out. Be sure to replace the target directories and certificate file names with yours, if you followed your own naming convention:

```
SSLCertificateFile      /etc/apache2/certs/mysite.crt
SSLCertificateKeyFile /etc/apache2/certs/mysite.key
```

To make Apache benefit from the new configuration, reload the `apache2` daemon:

```
# systemctl reload apache2
```

With the new configuration in place, we're not quite done but close. We still have a small bit of configuration left to add. But before we get to that, let's return to the topic of installing SSL certificates that were signed by a certificate authority. The process for installing signed SSL certificates is pretty much the same, but the main difference is how the certificate files are requested and obtained. Once you have them, you will copy them to your file server and configure Apache the same way as we just did. To start the process of obtaining a signed SSL certificate, you'll need to create a **Certificate Signing Request** (**CSR**). A CSR is basically a request for a certificate, in file form, that you'll supply to your certificate authority to start the process of requesting a signed certificate. A CSR can be easily generated with the following command:

```
openssl req -new -newkey rsa:2048 -nodes -keyout server.key -out server.
csr
```

With the CSR file that was generated, you can request a signed certificate. The CSR file should now be in your current working directory. The entire process differs from one provider to another, but in most cases it's fairly straightforward. You'll send them the CSR, pay their fee, fill out a form or two on their website, prove that you are the owner of the website in question, and then the vendor will send you the files you need. It may sound complicated, but certificate authorities usually walk you through the entire process and make it clear what they need from you in order to proceed. Once you complete the process, the certificate authority will send you your certificate files, which you'll then install on your server. Once you configure the `SSLCertificateFile` and `SSLCertificateKeyFile` options in the `/etc/apache2/sites-available/default-ssl.conf` to point to the new certificate files and reload Apache, you should be good to go.

www.it-ebooks.info

There's one more additional step we should perform for setting this up properly. At this point, our certificate files should be properly installed, but we'll need to inform Apache of when to apply them. If you recall, the `default-ssl.conf` file provided by the `apache2` package is answering requests for any traffic not otherwise identified by a virtual host (the `<VirtualHost _default_:443>` option). We will need to ensure our web server is handling traffic for our existing websites when SSL is requested. We can add a `ServerName` option to that file, to ensure our site supports SSL.

Add the following option to the `/etc/apache2/sites-available/default-ssl` file, right underneath `<VirtualHost _default_:443>`:

```
ServerName mydomain.com:443
```

Now, when traffic comes in to your server on port `443` requesting a domain that matches the domain you typed for the `ServerName` option, it should result in a secure browsing session for the client. You should see the green padlock icon in the address bar (this depends on your browser), which indicates your session is secured. If you're using self-signed certificates, you'll probably see an error you'll have to skip through first, and you may not get the green padlock icon. This doesn't mean the encryption isn't working, it just means your browser is skeptical of the certificate since it wasn't signed by a known certificate authority. Your session will still be encrypted.

If you are planning on hosting multiple websites over HTTPS, you may want to consider using a separate virtual host file for each. An easy way to accomplish this is to use the `/etc/apache2/sites-available/default-ssl.conf` file as a template, and change the `DocumentRoot` to the directory that hosts the files for that site. In addition, be sure to update the `SSLCertificateFile` and `SSLCertificateKeyFile` options to point to the certificate files for the site, and set the `ServerName` to the domain that corresponds to your site. Here's an example virtual host file for a hypothetical site that uses SSL. I've highlighted lines that I've changed from the normal `default-ssl.conf` file:

```
<IfModule mod_ssl.c>
        <VirtualHost *:443>
                ServerName acmeconsulting.com:443

                ServerAdmin webmaster@localhost

                DocumentRoot /var/www/acmeconsulting

                ErrorLog ${APACHE_LOG_DIR}/acmeconsulting-error.log
                CustomLog ${APACHE_LOG_DIR}/acmeconsulting-access.log
combined
```


www.it-ebooks.info

```
                SSLEngine on

        SSLCertificateFile      /etc/apache2/certs/acmeconsulting/
acme.crt
        SSLCertificateKeyFile /etc/apache2/certs/acmeconsulting/acme.
key

                <FilesMatch "\.(cgi|shtml|phtml|php)$">
                                SSLOptions +StdEnvVars
                </FilesMatch>
                <Directory /usr/lib/cgi-bin>
                                SSLOptions +StdEnvVars
                </Directory>
                BrowserMatch "MSIE [2-6]" \
                                nokeepalive ssl-unclean-shutdown \
                                downgrade-1.0 force-response-1.0
                # MSIE 7 and newer should be able to use keepalive
                BrowserMatch "MSIE [17-9]" ssl-unclean-shutdown

        </VirtualHost>
</IfModule>
```

Basically, what I've is was create a new virtual host configuration file (using the existing `default-ssl.conf` file as a template). I called this new file `acme-consulting.conf` and I stored it in the `/etc/apache2/sites-available` directory. I changed the `VirtualHost` line to listen for anything coming in on port `443`. The line `ServerName acmeconsulting.com:443` was added, to make this file responsible for traffic coming in looking for `acmeconsulting.com` on port `443`. I also set the `DocumentRoot` to `/var/www/acmeconsulting`. In addition, I customized the error and access logs so that it will be easier to find log messages relating to this new site, since its log entries will go to their own specific files.

In my experience, I find that a modular approach such as what I've done with the sample virtual host file for HTTPS works best when setting up a web server that's intended to host multiple websites. With each site, I'll typically give them their own **Document Root**, certificate files, and log files. Even if you're only planning on hosting a single site on your server, using this modular approach is still a good idea, since you may want to host additional sites later on.

So, there you have it. You should now understand how to set up secure virtual hosts in Apache. However, security doesn't equal redundancy. In the next section, we'll look at one method in which we can make our site highly available.

www.it-ebooks.info

Setting up high availability with keepalived

Using `keepalived` is a great way to add high availability to an application or even a hosted website. `keepalived` allows you to configure a floating IP (also known as a **Virtual IP** or **VIP**) for a pool of servers, with this special IP being applied to a single server at a time. Each installation of `keepalived` in the same group will be able to detect when another server isn't available, and claim ownership of the floating IP whenever the master server isn't responding. This allows you to run a service on multiple servers, with a server taking over in the event another becomes unavailable.

`keepalived` is by no means specific to Apache. You can use it with many different applications and services. `keepalived` also allows you to create a load balanced environment as well. But for our purposes, it's useful to talk about here because we can use it to make our website highly available.

Let's talk a little bit about how `keepalived` can work with Apache in theory. Once `keepalived` is installed on two or more servers, it can be configured such that a master server will have ownership of the floating IP, and other servers will continually check the availability of the master, claiming the floating IP for themselves whenever the current master becomes unreachable. For this to work, each server would need to contain the same Apache configuration and site files. I'll leave it up to you in order to set up multiple Apache servers. If you've followed along so far, you should have at least one already. If it's a virtual machine, feel free to create a clone of it. If not, all you should have to do is set up another server following the instructions earlier in this chapter. If you're able to view a website on both servers, you're good to go.

To get started, we'll need to declare a floating IP. This can be any IP address that's not currently in use on your network. For this to work, you'll need to set aside an IP address for the purposes of `keepalived`. If you don't already have one, pick an IP address that's not being used by any device on your network. If you're at all unsure, you can use an IP scanner on your network to find an available IP address. There are several scanners which can accomplish this, such as `nmap` on Linux, or the angry IP scanner for Windows (I have no idea what made that IP scanner so angry, but it does work well).

www.it-ebooks.info

Be careful when scanning networks for free IP addresses, as scanning a network may show up in intrusion detection systems as a threat. If you're scanning any network other than one you own, always make sure you have permission from both the network administrator as well as management before you start scanning. Also, if you're scanning a company network, keep in mind that any hardware load balancers in use may not respond to pings, so you may want to also look at an IP layout from your network administrator as well and compare the results.

To save you the trouble of a Google search, the following `nmap` syntax will scan a subnet and report back regarding which IP addresses are in use. You'll need the `nmap` package installed first. Just replace the network address with yours:

```
nmap -sP 192.168.1.0/24
```

Next, we'll need to install `keepalived`. Simply run the following command on both of your servers:

```
# apt-get install keepalived
```

If you check the status of the `keepalived` daemon, you'll see that it attempted to start as soon as it was installed, and then immediately failed. If you check the status of `keepalived` with the `systemctl` command, you'll see an error similar to the following:

```
Condition: start condition failed
```

Since we haven't actually configured `keepalived`, it's fine for it to have failed. After all, we haven't given it anything to check, nor have we assigned our floating IP. By default, there is no sample configuration file created once you've installed the package, we'll have to create that on our own. Configuration for `keepalived` is stored in `/etc/keepalived`, which at this point should just be an empty directory on your system. If for some reason this directory doesn't exist, create it:

```
# mkdir /etc/keepalived
```


www.it-ebooks.info

Let's open a text editor on our primary server and edit the `/etc/keepalived/keepalived.conf` file, which should be empty. Insert the following code into the file. There are some changes you'll need to make in order for this configuration to work in your environment. As we continue, I'll explain what the code is doing and what you'll need to change. Keep your text editor open after you paste the code, and keep reading for some tips on how to ensure that this `config` file will work for you:

```
global_defs {
    notification_email {
    myemail@mycompany.com
    }
    notification_email_from keepalived@mycompany.com
    smtp_server 192.168.1.150
    smtp_connect_timeout 30
    router_id mycompany_web_prod
}
vrrp_instance VI_1 {
    smtp_alert
    interface enp0s3
    virtual_router_id 51
    priority 100

    advert_int 5
    virtual_ipaddress {
    192.168.1.200
    }
}
```

There's quite a bit going on in this file, so I'll explain the configuration section by section, so you can better understand what's happening.

```
global_defs {
    notification_email {
    myemail@mycompany.com
    }
    notification_email_from keepalived@mycompany.com
    smtp_server 192.168.1.150
    smtp_connect_timeout 30
    router_id mycompany_web_prod
}
```


www.it-ebooks.info

In the `global_defs` section, we're specifically configuring an e-mail host to use in order to send out alert messages. When there's an issue and `keepalived` switches to a new master server, it can send you an e-mail to let you know that there was a problem. You'll want to change each of these values to match that of your mail server. If you don't have a mail server, `keepalived` will still function properly. You can remove this section if you don't have a mail server, or comment it out. However, it's highly recommended that you set a mail server for `keepalived` to use.

```
vrrp_instance VI_1 {
    smtp_alert
    interface enp0s3
    virtual_router_id 51
    priority 100

    advert_int 5
```

Here, we're assigning some configuration options regarding how our virtual IP assignment will function. The first thing you'll want to change here is the interface. In my sample `config`, I have `enp0s3` as the interface name. This will need to match the interface on your server where you would like `keepalived` to be bound to. If in doubt, execute `ip a` at your shell prompt to get a list of interfaces.

The `virtual_router_id` is a very special number that can be anything from `0-255`. This number is used to differentiate multiple instances of `keepalived` running on the same subnet. Each member of each `keepalived` cluster should have the same `virtual_router_id`, but if you're running multiple pairs or groups of servers on the same subnet, each group should have their own `virtual_router_id`. I used `51` in my example, but you can use whatever number you'd like. You'll just have to make sure both web servers have the same `virtual_router_id`. I'll go over some tips setting up the other server shortly, so don't worry if you haven't configured `keepalived` on your second server yet.

Another option within this block that's important is the priority. This number must absolutely be different on each server. The `keepalived` instance with the highest priority is always considered the master. In my example, I set this number to `100`. Using `100` for the priority is fine, so long as no other server is using that number. Other servers should have a lower priority. For example, you can set the second web server's priority to `80`. If you set up a third or fourth one, you could set their priority to `60` and `40`, respectively. Those are just arbitrary numbers I picked off the top of my head. As long as each server has a different priority, and the server you'd like to be the master has the highest priority, you're in good shape.

```
    virtual_ipaddress {
    192.168.1.200
}
```


www.it-ebooks.info

In the final block, we're setting the virtual IP address. I chose `192.168.1.200` as a hypothetical example; you should choose an IP address outside of your DHCP range that's not being used by any device.

Now that you've configured `keepalived`, let's test it out and see if it's working. On your master server, start `keepalived`:

```
# systemctl start keepalived
```

After you start the daemon, take a look at its status to see how it's doing:

```
# systemctl status -l keepalived
```

If there are no errors, the status of the daemon should be active (running). If for some reason the status is something else, take a look at the log entries shown after you execute the `status` command, as `keepalived` is fairly descriptive regarding any errors it finds in your `config` file. If all went well, you should see the IP address you chose for the floating IP listed within your interfaces. Execute `ip a` at your prompt to see them. Do you see the floating IP? If not, keep in mind it can take a handful of seconds for it to show up. Wait 30 seconds or so and check your interfaces list again.

If all went well on the first server, we should set up `keepalived` on the second web server. Install the `keepalived` package on the other server if you haven't already done so, and then copy the `keepalived.conf` file from your working server to your new one. Here are some things to keep in mind regarding the second server's `keepalived.conf` file:

- Be sure to change the priority on the slave server to a lower number than what you used on the master server
- The `virtual_ipaddress` should be the same on both
- The `virtual_router_id` should be the same on both
- Start or restart `keepalived` on your slave server, and verify there were no errors when the service started up

Now, let's have some fun and see `keepalived` in action. What's follows is an extremely simple HTML file:

```
<html>
    <title>keepalived test</title>
    <body>
        <p>This is server #1!</p>
    </body>
</html>
```


www.it-ebooks.info

Copy this same HTML file to both servers and serve it via Apache. We've gone over setting up Apache in this chapter. The easiest and quickest way to implement this is to replace Apache's sample HTML file with the preceding one. Make sure you change the text `This is server #1!` to `This is server #2` on the second server. You should be able to use your browser and visit the IP address for each of your two web servers and see this page on both. The only difference between them should be the text inside the page.

Now, in your browser, change the URL you're viewing to that of your floating IP. You should see the same page as you did when you visited the IP address for `server #1`.

Let's now simulate a failover. On the master server, stop the `keepalived` service:

```
# systemctl stop keepalived
```

In our `config`, we set the `advert_int` option to `5`, so the second server should take over within 5 seconds of you stopping `keepalived` on the master. When you refresh the page, you should see the web page for `server #2` instead of `server #1`. If you execute `ip a` on the shell to view a list of interfaces on your slave server, you should also see that the slave is now holding your floating IP address. Neat, isn't it? To see your master reclaim this IP, start the `keepalived` daemon on your master server and wait a few seconds.

Congratulations, you set up a cluster for your Apache implementation. If the master server encounters an issue and drops off the network, your slave server will take over. As soon as you fix your master server, it will immediately take over hosting your web page. Although we made our site slightly different on each server, the idea is showing how hosting a website can survive a single server failure. You can certainly host more than just Apache with `keepalived`. Almost any service you need to be highly available that doesn't have its own clustering options is a good candidate for `keepalived`. For additional experimentation, try adding a third server.

Installing and configuring ownCloud

I figured we'd end this chapter with a fun activity: setting up our very own ownCloud server. ownCloud is a very useful web application that's handy for any organization. Even if you're not working on a company network, ownCloud is a great asset for even a single user. You can use it to synchronize files between machines, store and sync contacts, keep track of tasks you're working on, fetch e-mail from a mail server, and more. To complete this activity, you'll need a server to work with. You can use your existing web server, if you prefer.

www.it-ebooks.info

You'll also need an installation of MySQL or MariaDB, as ownCloud will need its own database. We went over installing and managing MariaDB databases in *Chapter 9*, *Managing Databases*. I'll give you all the commands you'll need to set up the database in this section, but refer back to *Chapter 9*, *Managing Databases* if any of these commands confuse you.

To get started, we'll need to add a repository for ownCloud on our server. Create a file named `owncloud.list` and store it in `/etc/apt/sources.list.d/`. Make the contents the following:

```
deb http://download.owncloud.org/download/repositories/stable/
xUbuntu_16.04/ /
```

We'll also need to add the `GPG` key for this repository. The following command will download the key to your current working directory:

```
wget -nv https://download.owncloud.org/download/repositories/stable/
xUbuntu_16.04/Release.key -O Release.key
```

Next, we install the key:

```
# apt-key add < owncloud.key
```

Now that we have the ownCloud repository installed on our server, we can install ownCloud with the following commands:

```
# apt-get update
# apt-get install owncloud-files
```

Once you install the `owncloud-files` package, you'll see that ownCloud's files have been copied to `/var/www/owncloud`. This location will work so long as your **Document Root** is `/var/www`, instead of Ubuntu's default of `/var/www/html`. If `/var/www` is not your **Document Root**, go ahead and make the necessary corrections. Don't bother navigating to ownCloud on your server just yet, it won't work until we finish a bit more configuration.

Next, we'll need to make a change to Apache. First, we'll need to ensure that the `libapache2-mod-php7.0` package is installed since ownCloud requires PHP, but we'll need some additional packages as well. You can install ownCloud's prerequisite packages with the following command:

```
# apt-get install libapache2-mod-php7.0 php7.0-curl php7.0-gd php7.0-intl
php7.0-mysql php7.0-xml php7.0-zip
```

Next, restart Apache so it can take advantage of the new PHP plug-in:

```
# systemctl restart apache2
```


www.it-ebooks.info

At this point, we'll need a MySQL or MariaDB database for ownCloud to use. This database can exist on another server, or you can share it on the same server you installed ownCloud on. If you haven't already set up MariaDB, a walk-through was covered during *Chapter 9*, *Managing Databases*. At this point, it's assumed that you already have MariaDB installed and running.

Log in to your MariaDB instance as your root user, or an admin user with full privileges. You can create the ownCloud database with the following command:

```
CREATE DATABASE owncloud;
```

Next, we'll need to add a new user to MariaDB for ownCloud, and give that user full access to the `owncloud` database. We can take care of both with the following command:

```
GRANT ALL ON owncloud.* to 'owncloud'@'localhost' IDENTIFIED BY 'super_
secret_password';
```

Make sure to change `super_secret_password` to a very strong (preferably randomly generated) password. Make sure you save this password in a safe place.

Before continuing, make sure that the `/var/www/owncloud` directory is available via Apache. The easiest way to do this, if you're not worried about other virtual hosts, is to change the default document root from `/var/www/html` to `/var/www`. If you do serve another virtual host from this server, you can create a new virtual host specifically for ownCloud, and set its document root to `/var/www/owncloud`. You can use another Apache `config` file as a base for creating a new virtual host if you choose to go that direction, rather than writing one from scratch.

Now we have all we need in order to configure ownCloud. You should now be able to visit your ownCloud instance in a web browser. Just enter a URL similar to the following, replacing the sample IP address with the one for your server. The following URL assumes you changed your default document root to `/var/www`:

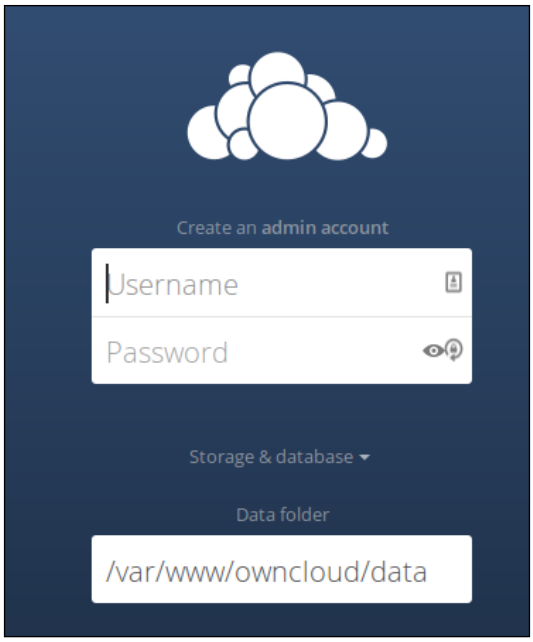
```
http://192.168.1.100/owncloud
```

If you're using a subdomain and gave ownCloud its own virtual host, that URL would then be something like this:

```
http://owncloud.yourdomain.com
```

You should see a page asking you to configure ownCloud:

www.it-ebooks.info



The introductory page for ownCloud

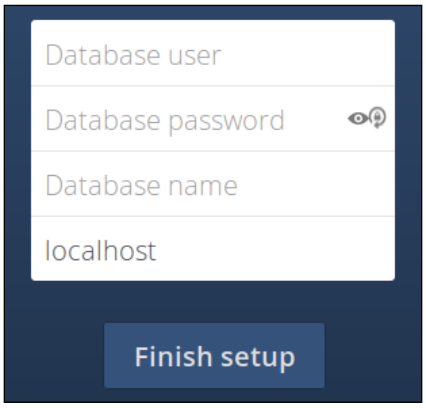
If you do not see this page, make sure that the `/var/www/owncloud` directory is accessible via Apache. Also, make sure you have an appropriate virtual host for ownCloud referencing this directory as its document root.

This page will ask you for several pieces of information. First, you'll be asked for **Username** and **Password**. This is not asking you for a pre-existing account, but actually to set up a brand new administrator account. This shouldn't be an account you'll use on a day-to-day basis, but instead an admin account you'll use only when you want to add users and maintain your system. Please note that it won't ask you to confirm the password, so you'll want to make certain you're entering the password you think you are. It's perhaps safer to type the password in a text editor, and then copy and paste the password into the **Password** box to make sure you don't lock yourself out.

The `Data` folder will default to `/var/www/owncloud/data`. This default is normally fine, but if you have configured your server to have a separate data partition, you can configure that here. If you plan on storing a large amount of data on your ownCloud server, setting up a separate partition for it may be a good idea. If you do, you can set that here. Otherwise, leave the default.

www.it-ebooks.info

In the next section, you'll be asked to fill in information for the ownCloud database we created earlier:



Configuring the database for ownCloud

The **Database user** and **Database password** will use the values we created when we set up the MariaDB database for ownCloud earlier. In my examples, I used `owncloud` for the username as well as the **Database name**. The password will be whatever it is you used for the password when we set up the database user account and grant. Finally, the database server defaults to localhost, which is correct as long as you set up the database on the same machine as the ownCloud server. If not, enter the server's IP address here.

That's it! Assuming all went well, ownCloud will set itself up in the background and you'll get the main screen. Since you only created an admin account so far, I recommend you create an account for yourself, as well as any friends or colleagues you'd like to check out your ownCloud server. To do so, click on the top-right corner of the ownCloud page, where it shows the user name you're logged in with. When you click on this, you'll see an option for **Users**. Within the users screen, you'll be able to add additional users to access ownCloud. Simply fill out the **Username** and **Password** fields at the top of the screen, and click on **Create**.

As an administrative user, you can enable or disable various apps that are used by your users. Out of the box, ownCloud only has the **Files** and **Gallery plug-ins** enabled, but there are many more plug-ins that you can use to extend its functionality. In the top-left corner of the main screen of ownCloud is the main menu, which says **Files** when you first visit the page. If you click on it, you'll get an **Apps** button you can click on. Clicking on this link will bring you to a screen where you can add additional apps to ownCloud. At first, ownCloud will list apps that you already have installed. If you click on **Not enabled** on the left, you'll see a listing of additional apps you can enable and benefit from.

www.it-ebooks.info

Feel free to enable additional apps to extend ownCloud's capabilities. Some of my must-haves include **Calendar** and **Contacts** (available in the **Productivity** section), **News** (available in the **Multimedia** section), and **ownNote** (available in the **Tool** section).

Now, you have your very own ownCloud server. I find ownCloud to be a very useful platform, something I've come to depend on for my calendar, contacts, and more. Some Linux desktop environments (such as GNOME) allow you to add your ownCloud account right to your desktop, which will allow calendar and contact syncing with your desktop or laptop. One thing that doesn't seem to work extremely well in ownCloud is the file syncing solution, which is still relatively unstable. Therefore, you might not want to synchronize your important files with ownCloud just yet. Development on ownCloud is progressing quite quickly, so it's only a matter of time before the syncing service stabilizes. By the time this book is printed, these issues may already be addressed. All other services work like a charm, however. I'm sure you'll agree that ownCloud is a very useful asset to have available.

Summary

In this action-packed chapter, we looked at serving web pages with Apache. We started out by installing and configuring Apache, and then added additional modules. We also covered the concept of virtual hosts, which allows us to serve multiple websites on a single server, even if we only have a single network interface. Then, we walked through securing our Apache server with SSL. With Apache, we can use self-signed certificates, or we can purchase SSL certificates from a vendor for a fee. We looked at both possibilities. `keepalived` is a handy daemon that we can use to make a service highly available. It allows us to declare a floating IP, which we can use to make an application such as Apache highly available. Should something go wrong, the floating IP will move to another server and as long as we direct traffic toward the floating IP, our service will still be available should the master server run into an issue. Finally, we closed out the chapter with a guide to installing ownCloud, which is an application I'm sure you'll find incredibly useful.

In the next chapter of our journey, we'll look into virtualization. Using the incredible tools that Ubuntu Server has at its disposal, we'll set up our very own virtualization server using KVM. You won't want to miss it.

www.it-ebooks.info

www.it-ebooks.info

11
Virtualizing Hosts and Applications

There has been a great deal of advancements in the IT space in the last several decades, and a few technologies have come along that have truly revolutionized the technology industry. I'm sure few would argue that the Internet itself is by far the most revolutionary technology to come around, but another technology that has created a paradigm shift in IT is virtualization. It changed the way we maintain our data centers, allowing us to segregate workloads into many smaller machines being run from a single server or hypervisor. Since Ubuntu features the latest advancements of the Linux kernel, virtualization is actually built right into it. After installing just a few packages, we can create virtual machines on our Ubuntu Server without the need for a pricey license agreement or support contract. In this chapter, I'll walk you through setting up your own Ubuntu-based virtualization solution. Along the way, I'll walk you through the following topics:

- Setting up a virtual machine server
- Creating virtual machines
- Bridging the virtual machine network
- Creating, running, and managing Docker containers

www.it-ebooks.info

Setting up a virtual machine server

I'm sure many of you have already used a virtualization solution before. In fact, I bet a great deal of readers are following along with this book while using a virtual machine running within a solution such as VirtualBox, Parallels, VMware, or one of the others. In this section, we'll see how to use an Ubuntu Server in place of those solutions. While there's certainly nothing wrong with solutions such as VirtualBox, Ubuntu has virtualization built right in, in the form of **Kernel-based Virtual Machine** (**KVM**). KVM offers a very fast interface to the Linux kernel to run your virtual machines with near-native speeds, depending on your use case.

I bet you're eager to get started, but there are a few quick things to consider before we dive in. First, of all the activities I've walked you through in this book so far, setting up our own virtualization solution will be the most expensive from a hardware perspective. The more virtual machines you plan on running, the more resources your server will need to have available (especially RAM). Thankfully, most computers nowadays ship with 4 GB of RAM at minimum, with 8 GB or more being fairly common. With most modern computers, you should be able to run virtual machines without too much of an impact. Depending on what kind of machine you're using, CPU and RAM may present a bottleneck, especially when it comes to legacy hardware.

For the purposes of this chapter, it's recommended that you have a PC or server available with a processor that's capable of supporting virtual machine extensions. The performance of virtual machines may suffer without this support, if you can even get VMs to run at all without these extensions. A good majority of CPUs on computers nowadays offer this, though some may not. To be sure, you can run the following command on the machine you intend to host KVM virtual machines on in order to find out whether your CPU supports virtualization extensions. A result of `1` or more means that your CPU does support virtualization extensions. A result of `0` means it does not:

```
egrep -c '(vmx|svm)' /proc/cpuinfo
```

Even if your CPU does support virtualization extensions, it's usually a feature that's disabled by default with most end-user PCs sold today. To enable these extensions, you'll need to enter the BIOS setup screen for your computer and enable the option. Depending on your CPU and chip-set, this option may be called **VT-x** or **AMD-V**. Some chip-sets may simply call this feature **virtualization support** or something along those lines. Unfortunately, I won't be able to walk you through how to enable the virtualization extensions for your hardware, since the instructions will differ from one machine to another. If in doubt, refer to the documentation for your hardware.

www.it-ebooks.info

There are two ways in which you can use and interface with KVM. You can choose to set up virtualization on your desktop or laptop computer, replacing a solution such as VirtualBox. Alternatively, you can set up a server on your network, and manage the VMs running on it remotely. It's really up to you, as neither solution will impact how you follow along with this chapter. Later on, I'll show you how to connect to a local KVM instance or a remote one. It's really simple to do, so feel free to set up KVM on whichever machine you prefer. If you have a spare server available, it would likely make a great KVM host. Not all of us have spare servers lying around though, so use what you have.

One final note: I'm sure many of you are using VirtualBox, as it seems to be a very popular solution for those testing out Linux distributions (and rightfully so, it's great!). However, you can't run both VirtualBox and KVM virtual machines on the same machine simultaneously. This probably goes without saying, but I wanted to mention it just in case you didn't already know. You can certainly have both solutions installed on the same machine, you just can't have a VirtualBox VM up and running, and then expect to start up a KVM virtual machine at the same time. The virtualization extensions of your CPU can only work with one solution at a time.

Another consideration to bear in mind is where on the filesystem you want to store your virtual machines. These can take up a lot of disk space. In my examples, I'm going to use the `/vm_store` directory. On your end, you can have a dedicated disk mounted to `/vm_store`, or you can simply create the `/vm_store` directory on your root filesystem if you only have a single partition. Just be cognizant of your free space, as virtual machines can become quite large. Go ahead and make this directory if you haven't already done so:

```
# mkdir /vm_store
```

We'll also need to create a group named `kvm`:

```
# groupadd kvm
```

Now, make the `root` user and the `kvm` group the owner of the `/vm_store` directory:

```
# chown root:kvm /vm_store
```

Let's set the permissions of `/vm_store` such that anyone in the `kvm` group will be able to modify its contents:

```
# chmod g+w /vm_store
```


www.it-ebooks.info

The primary user account you use on the server should be a member of the `kvm` group. That way, you'll be able to manage virtual machines without switching to `root` first. Make sure you log out and log in again after executing the next command so the changes take effect:

```
# usermod -aG kvm <user>
```

Even though KVM is built into the Linux kernel, we'll still need to install some packages in order to get everything going. These packages will require a decent number of dependencies, so it may take a few minutes for everything to install:

```
# apt-get install bridge-utils libvirt-bin qemu-kvm qemu-system
```

You'll now have an additional service running on your server, `libvirt-bin`. Once you've finished installing KVM's packages, this service will be started and enabled for you. Feel free to take a look at it to see for yourself:

```
# systemctl status libvirt-bin
```

On your laptop or desktop (or the machine you'll be managing KVM from), you'll need a few additional packages:

```
# apt-get install ssh-askpass virt-manager
```

We have all the tools installed that we will need, so all that we need to do is configure the KVM server for our use. There are a few configuration files we'll need to edit. The first is `/etc/libvirt/libvirtd.conf`. There are a number of changes you'll need to make to this file, which I'll outline below. First, you should make a backup copy of this file in case you make a mistake:

```
# cp /etc/libvirt/libvirtd.conf /etc/libvirt/libvirtd.conf.orig
```

Next, look for the following line:

```
unix_sock_group = "libvirtd"
```

Change that line to the following:

```
unix_sock_group = "kvm"
```


www.it-ebooks.info

Now, find this line:

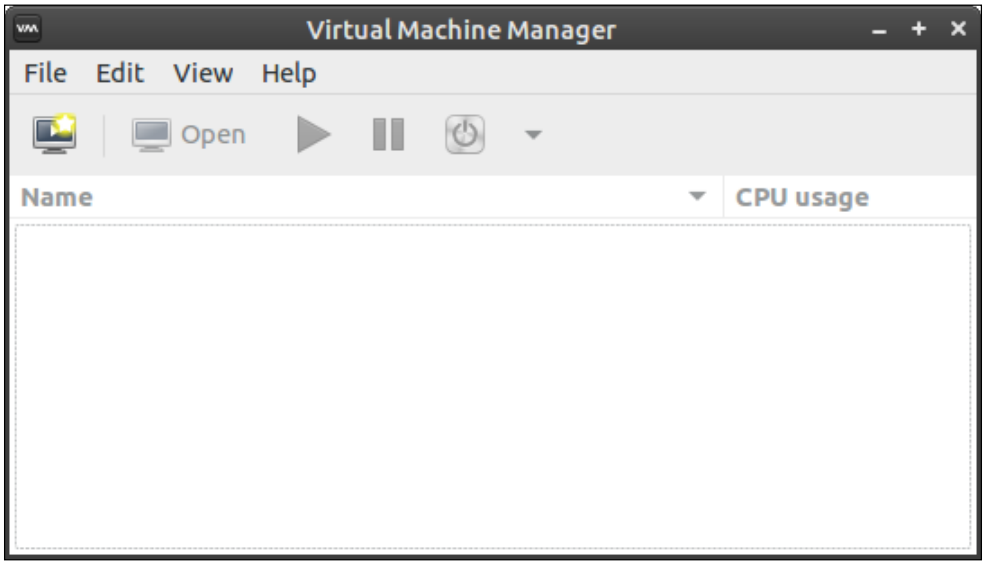
```
unix_sock_ro_perms = "0777"
```

Change it to this:

```
unix_sock_ro_perms = "0770"
```

The next file we'll need to edit is `/etc/libvirt/storage/default.xml`. This file is what defines your default storage group for your virtual machines. This file doesn't exist yet. To properly create it, we'll need to establish a connection from your laptop or desktop to your KVM server. To do that, we'll use `virt-manager`, which is a graphical utility we can use to manage our virtual machines. This is also a good chance for you to see how this works.

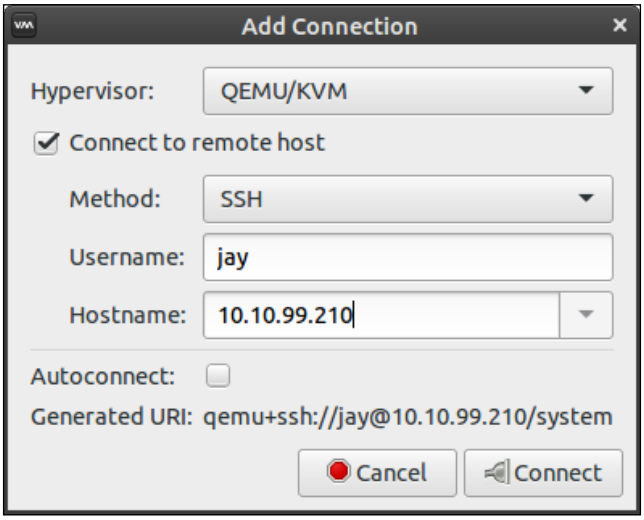
First, open `virt-manager` on your administration machine. It should be located in the **Applications** menu of your desktop environment, usually under the **System Tools** section under **Virtual Machine Manager**. If you have trouble finding it, simply run `virt-manager` at your shell prompt.



The virt-manager application

www.it-ebooks.info

The `virt-manager` utility is especially useful as it allows us to manage both remote and local KVM servers. From one utility, you can create connections to any of your KVM servers, including an external server or localhost if you are running KVM on your laptop or desktop. To create a new connection, click on **File** and select **Add Connection**. A new screen will appear, where we can fill out the details for the KVM server we wish to connect to:

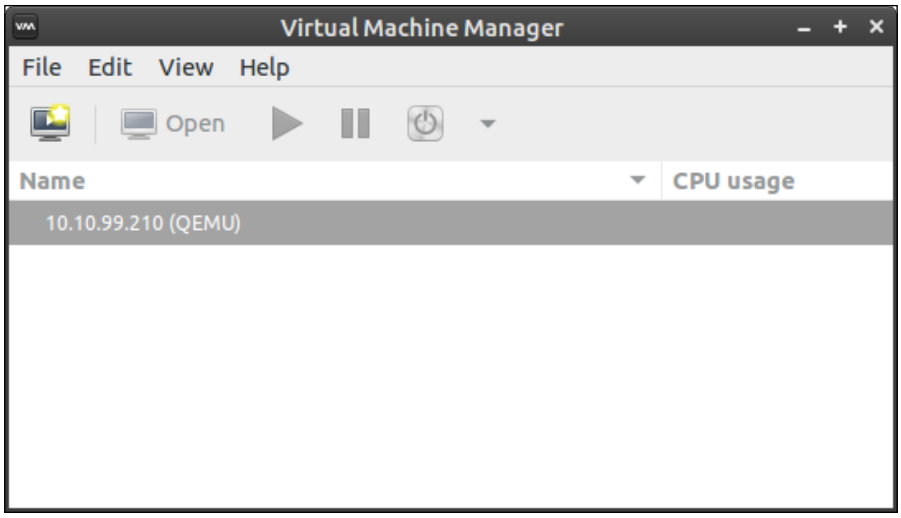


Adding a new connection to virt-manager

If you're connecting to an external KVM server, check the box that reads **Connect to remote host**. The method is SSH, so we can leave that as is. For the **Username**, type the username you use on your remote machine. You'll need to have already added your user account on that server to the `kvm` group for this to work. Next, type the IP address for the server you wish to connect to under the **Hostname** field. If you're connecting to a local instance of KVM, leave the **Connect to remote host** checkbox unchecked and ignore the other fields. Click on **Connect** to make a connection to your server or local KVM instance.

You might see a pop-up dialog box with the text **Are you sure you wish to continue connecting (yes/no)?** If you do, type `yes` and press *Enter*. Either way, you should be prompted for your password to your KVM server; type that in and press *Enter*. You should now have a connection listed within your `virt-manager` application:

www.it-ebooks.info



virt-manager with an active remote connection listed

We'll get back to managing our virtual machines with `virt-manager` a bit later. For now, you can close this application. Now that you've opened `virt-manager` and made a connection to your server (either local or remote), the `/etc/libvirt/storage` directory (which didn't exist before) has now been created for you. Inside that directory, you'll see a file called `default.xml`. This is the file I was referring to earlier that defines your default storage group. Let's edit it. Find the following line within the file:

```
<path>/var/lib/libvirt/images</path>
```

Change it to the following:

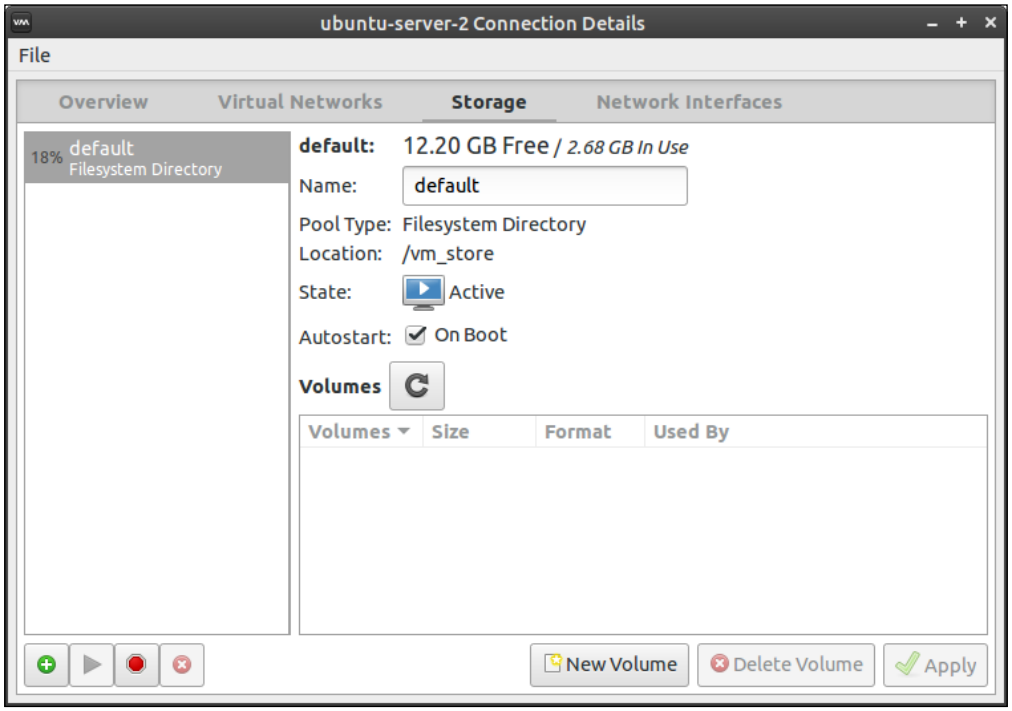
```
<path>/vm_store</path>
```

Now, restart `libvirt-bin` so that the changes will take affect:

```
# systemctl restart libvirt-bin
```


www.it-ebooks.info

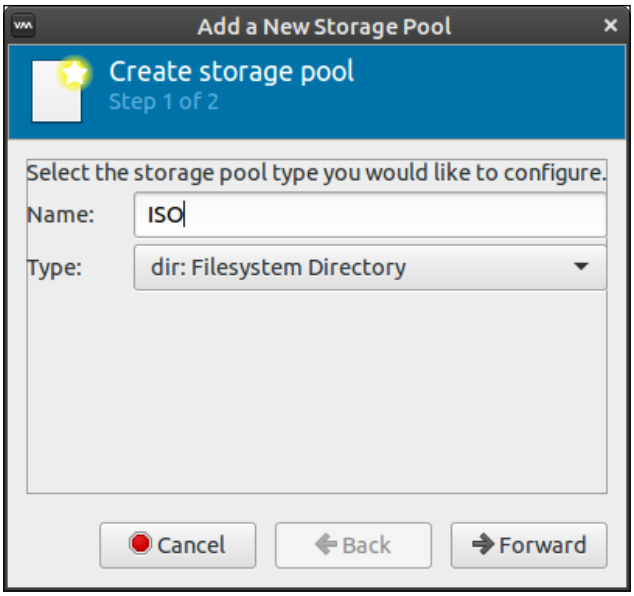
We're almost at a point where we'll be able to test our KVM server. But there's another storage group we'll want to create. We'll need a storage group for ISO images, for use when installing operating systems on our virtual machines. When we create a virtual machine, we can attach an ISO image from our ISO storage group to our VM, which will allow it to install the operating system. To create this storage group, let's open `virt-manager` again. Right-click on the listing for your server connection and then click on **Details**. You'll see a new window that will show details regarding your KVM server. Click on the **Storage** tab:



The storage tab of the virt-manager application

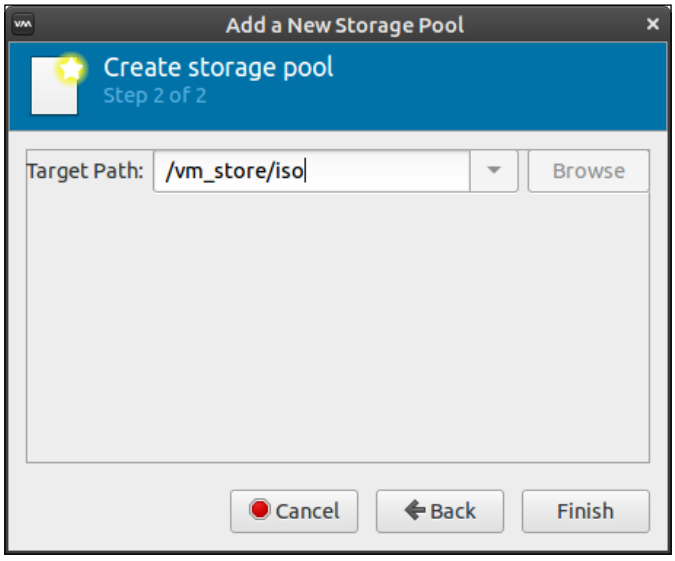
At first, you'll only see the default connection we edited earlier. This screen should show its **Location** as `/vm_store`. If it doesn't, click on the red stop sign icon near the bottom left to stop the pool and click on the play button right next to it to start it up again. This should refresh the default storage pool so that it shows the correct location. Now we can add our ISO storage pool. Click on the green plus symbol to create the new pool:

www.it-ebooks.info



The first screen while setting up a new storage pool

In the **Name** field, type `ISO`. You can actually name it anything you want, but `ISO` makes sense, considering it will be storing ISO images. Leave the **Type** setting as `dir: Filesystem Directory`:



The second screen while setting up a new storage pool

www.it-ebooks.info

For the **Target Path** field, `/vm_store/iso` works well. This directory will be created for you, if it doesn't already exist. Click on **Finish** to complete the process.

Congratulations! You now have a fully-configured KVM server for creating and managing virtual machines. Our server has a place to store virtual machines as well as ISO images. You should also be able to connect to this instance using `virt-manager`, as we've done in this section. Next, I'll walk you through the process of setting up your first VM.

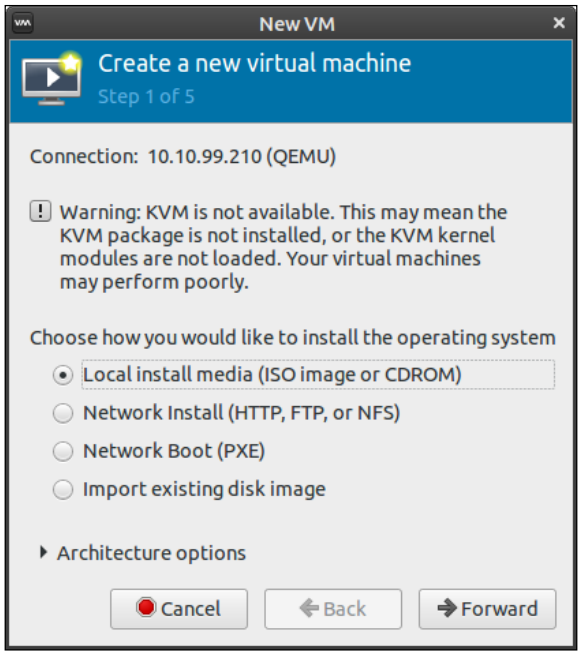
Creating virtual machines

Now the time has come to put your new virtual machine server to the test. In order for us to give this new server a test run, you'll need an ISO of an operating system to install. You can download just about any Linux distribution ISO image, and copy it to your `/vm_store/iso` directory to make it available for use. You can even copy a Microsoft Windows ISO image there and install that; I won't judge you. After you copy the file over to the `/vm_store/iso` directory, we're ready to begin.

Before continuing, I highly recommend that you set up public key authentication for SSH between your workstation and virtual machine server. If you're using a local connection, you won't need to do this. But when you're connecting to a remote KVM instance, without setting up public key authentication between your workstation and server, you will likely be asked for your SSH password repeatedly. It can be very annoying. If you haven't used public key authentication for SSH yet, please refer back to *Chapter 4, Connecting to Networks* for an overview.

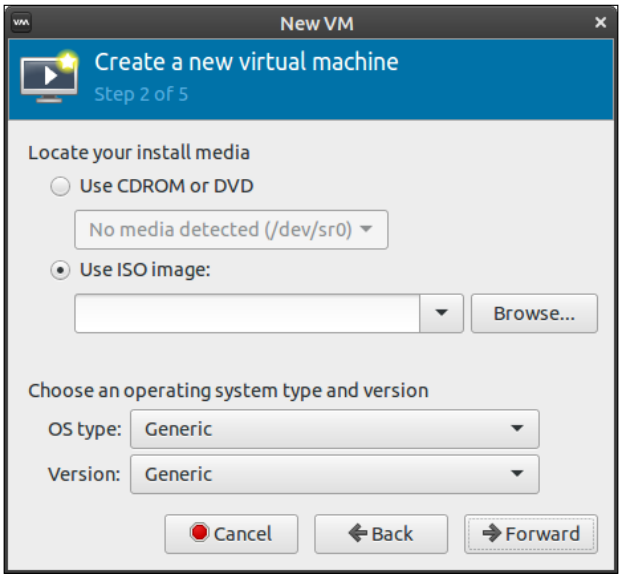
Back in `virt-manager`, right-click your sever connection and click on **New** to start the process of creating a new virtual machine.

www.it-ebooks.info



The first screen of the process of setting up a new virtual machine

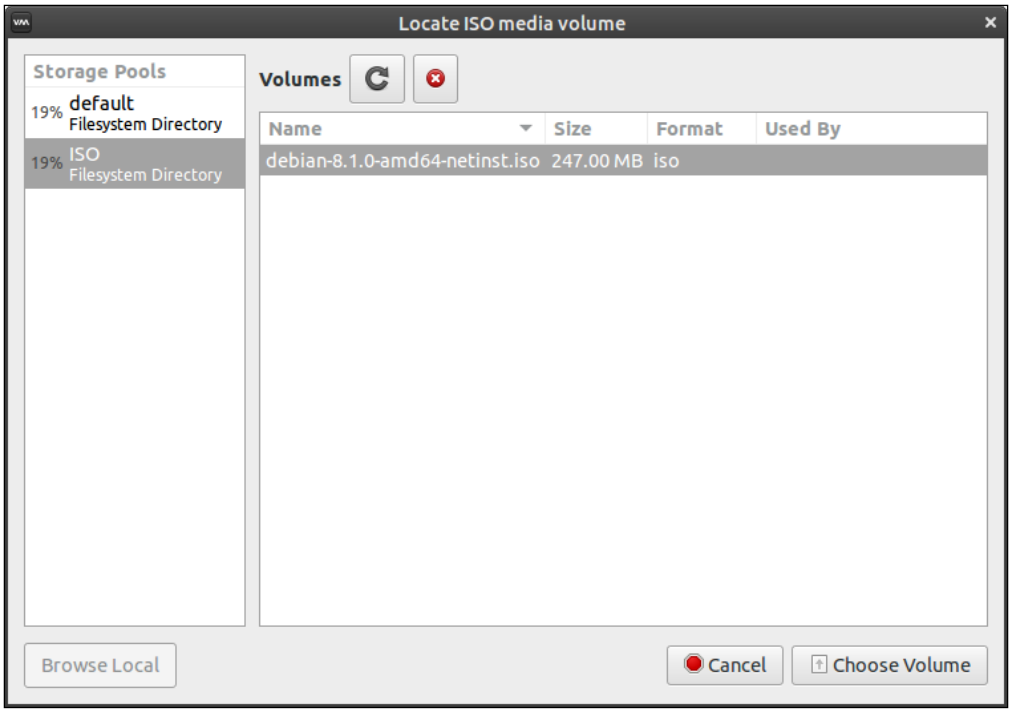
The default selection will be on **Local install media (ISO image or CDROM)**; leave this selection and click on **Forward**.



Creating a new VM and setting VM options

www.it-ebooks.info

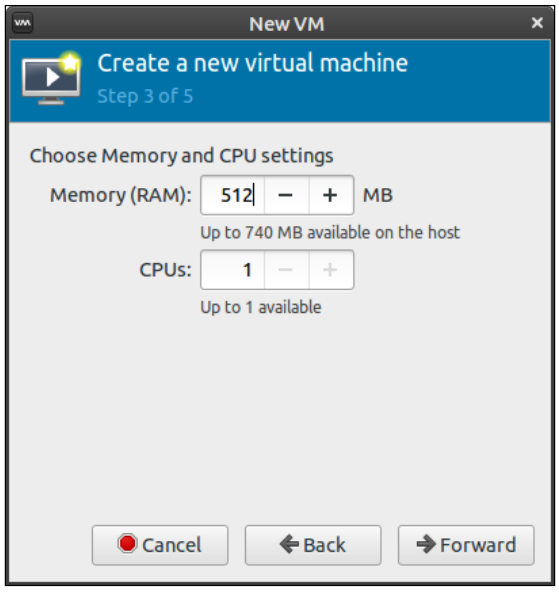
On the next screen, click on **Browse** to open up another window where you can select an ISO image you've downloaded.



Choosing an ISO image during VM creation

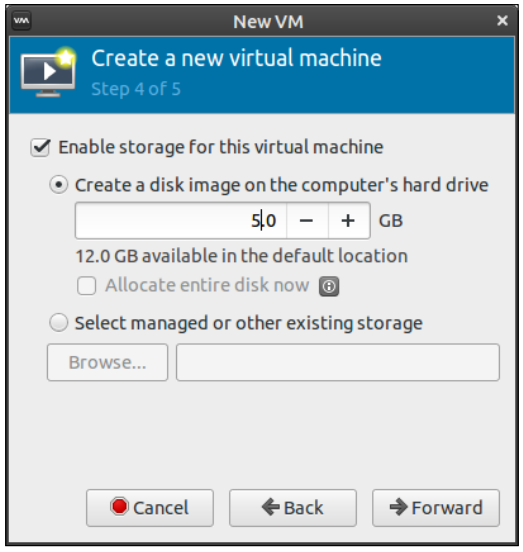
If you click on your **ISO** storage pool, you should see a list of ISO images you've downloaded. In my sample server, I added an install image for Debian, because it's small and quick to download. You can use whatever operating system you prefer. Click on **Choose volume** to confirm your selection.

www.it-ebooks.info



Adjusting RAM and CPU count for the new VM

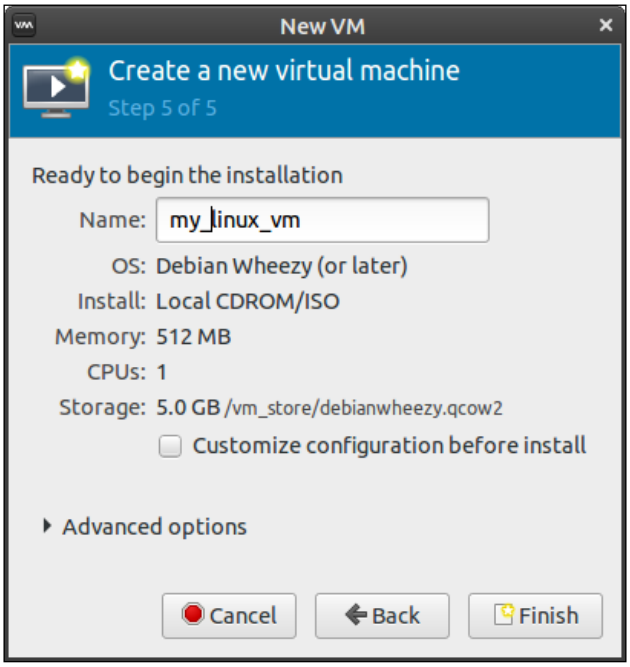
Next, you'll be asked to allocate RAM and CPU resources to the virtual machine. For most Linux distributions with no graphical user interface, 512 MB is plenty (unless your workload demands more). The resources you select here will depend on what you have available on your host. Click on **Forward** when you've finished allocating resources.



Creating a new virtual hard drive during VM creation

www.it-ebooks.info

Next, you'll allocate free disk space for your virtual machine's virtual hard disk. This space won't be used up all at once; by default KVM utilizes thin provisioning that basically just fills up the virtual disk as your VM needs space. You can select **Allocate entire disk now** if you'd like to claim all the space all at once, but that isn't necessary. Click on **Forward** when done.



Naming the new virtual machine

Finally, you'll name your virtual machine. This won't be the host name of the virtual machine; it's just the name you'll see when you see the VM listed in `virt-manager`. When you click on **Finish**, the VM will start and it will automatically boot into the install ISO you've attached to the VM near the beginning of the process. The installation process for that operating system will then begin.

When you click on the VM window, it will steal your keyboard and mouse and dedicate it to the window. Press *Ctrl* and *Alt* at the same time to release this control and regain full control of your keyboard and mouse.

www.it-ebooks.info

Unfortunately, I can't walk you through the installation process of your VM's operating system, since there are hundreds of possible candidates you may be installing. If you're installing another instance of Ubuntu Server, you can refer back to *Chapter 1*, *Deploying Ubuntu Server*, where we walked through the process. The process will be the same within the VM. From here, you should be able to create as many VMs as you need and have resources for.

Bridging the virtual machine network

Your KVM virtual machines will use their own network, unless you configure bridged networking. This means your virtual machines will get an IP address in their own network, instead of yours. By default, each machine will be a member of the `192.168.122.0/24` network, with an IP address within the range of `192.168.122.2` to `192.168.122.254`. If you're utilizing KVM VMs on your personal laptop or desktop, this behavior might be adequate. You'll be able to SSH into your virtual machines by their IP address, and you'll be able to communicate with the host from within a virtual machine with the IP address `192.169.122.1`. If this satisfies your use case, there's no further configuration you'll need to do.

Bridged networking allows your VMs to receive an IP address from the DHCP server on your network instead of its internal one, which will allow you to communicate with your VMs from any other machine on your network. This use case is preferable if you're setting up a central virtual machine server to power infrastructure for your small office or organization. With a bridged network on your VM server, each VM will be treated as any other network device. All you'll need is a wired network interface, as wireless cards typically don't work with bridged networking.

To set up bridged networking, we'll need to create a new interface on our server. Open up the `/etc/network/interfaces` file in your text editor as `root`. We already talked about this file in *Chapter 4*, *Connecting to Networks*, so I won't go into too much detail about it here. Basically, this file includes configuration for each of our network interfaces, and this is where we'll add our new bridged interface.

Make sure you make a backup of the original interfaces file, and then replace its contents with the following. Be sure to replace `enp0s3` (the interface name) with your actual wired interface name if it's different. There are three occurrences of it in the file. Take your time while configuring this file. If you make a single mistake, you will likely not have network access to the machine once it restarts.

```
auto lo
iface lo inet loopback

auto enp0s3
```


www.it-ebooks.info

```
iface enp0s3 inet manual

auto br0
iface br0 inet dhcp
        bridge_ports enp0s3
        bridge_stp off
        bridge_fd 0
        bridge_maxwait 0
```

After you make the change, you can restart networking, or simply reboot the server. If you have a monitor and keyboard hooked up to the server, restarting networking is the easiest way to activate the new configuration:

```
# systemctl restart networking
```

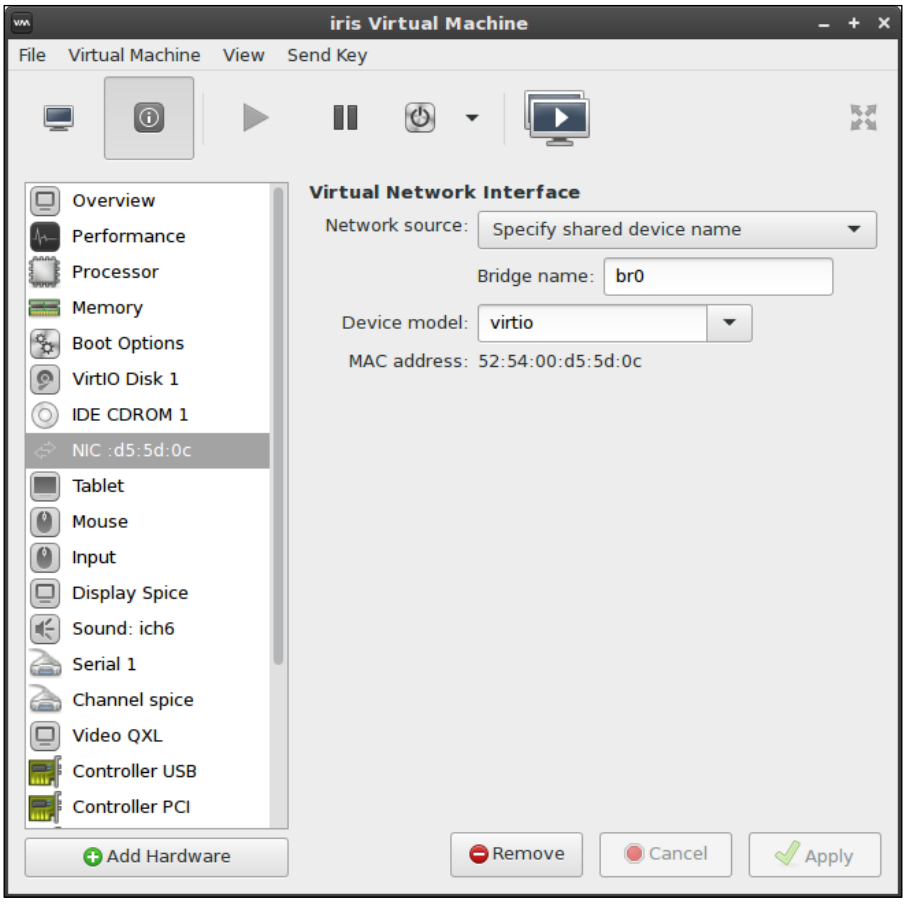
If you're connected to the server via SSH, restarting networking will likely result in the server becoming inaccessible because the SSH connection will likely drop as soon as the network stops. This will disrupt the connection and prevent networking from starting back up. If you know how to use `screen` or `tmux`, you can run the `restart` command from within either; otherwise, it may just be simpler for you to reboot the server.

After networking restarts or the server reboots, check whether you can still access network resources, such as pinging websites and accessing other network nodes from it. If you can, you're all set. If you're having any trouble, make sure you edited the `/etc/network/interfaces` file properly.

Now, you should see an additional network interface listed when you run `ip a`. The interface will be called `br0`. The `br0` interface should have an IP address from your DHCP server, in place of your `enp0s3` interface (or whatever it may be named on your system). From this point onwards, you'll be able to use `br0` for your virtual machine's networking, instead of the internal network. The internal KVM network will still be available, but you can select `br0` to be used instead when you create new virtual machines.

www.it-ebooks.info

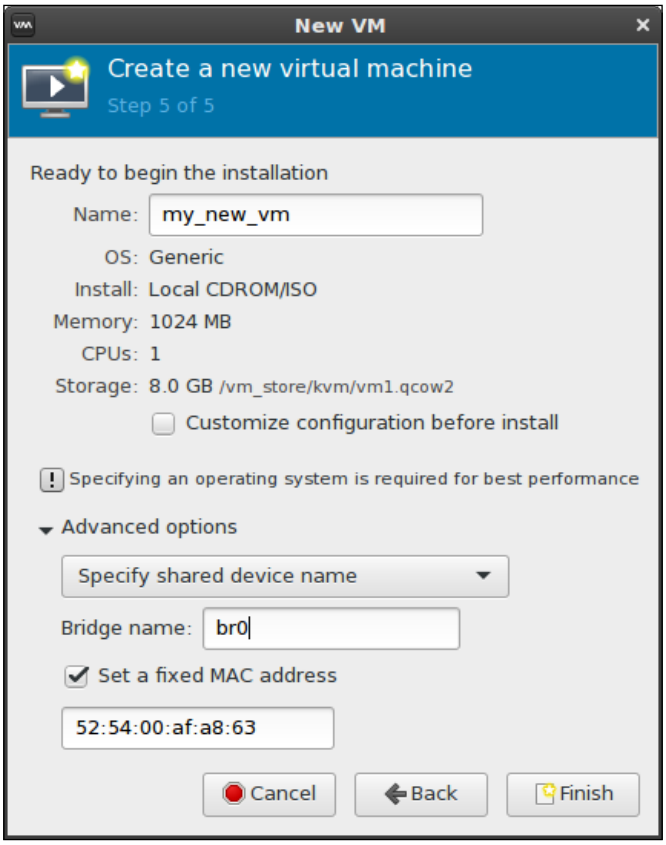
If you have a virtual machine you've already created that you'd like to switch to utilize your bridged networking, you can use the following steps to convert it. First, open `virt-manager` and double-click on your virtual machine. A new window with a graphical console of your VM will open. The second button along the top will open the **Virtual Hardware Details** tab, which will allow you to configure many different settings for the VM, such as the CPU count, RAM amount, boot device order, and more. Among the options on the left-hand side of the screen, there will be one that reads NIC and shows part of the virtual machine's network card's **MAC address**. If you click on this, you can configure the virtual machine to use your new bridge. Under the **Network Source** drop-down menu, select the option **Specify shared device** name. This will allow you to type an interface name in the **Bridge name** text box; type `br0` into that text box. Make sure that the **Device model** is set to `virtio`. Finally, click on **Apply**. You may have to restart the virtual machine for the changes to take affect:



Configuring a virtual machine to use bridge br0

www.it-ebooks.info

While creating a brand new virtual machine, there's an additional step you'll need to do in order to configure the VM to use bridged networking. On the last step of the process, where you set a name for the VM, you'll also see **Advanced** options listed near the bottom of the window. Expand this, and you'll be able to set your network name. Change the dropdown in this section to **Specify shared device** name, and set the bridge **Name** to `br0`. Now, you can click on **Finish** to finalize the VM as before, and it should use your bridge whenever it starts up:



Configuring bridge br0 on a newly created VM

From this point onwards, you should have not only a fully-configured KVM server or instance, but also a solution that can be treated as a full citizen of your network. Your VMs will be able to receive an IP address from a DHCP server, and communicate with other network nodes directly. If you have a very beefy KVM server, you may even be able to consolidate other network appliances into VMs to save space, which is basically the entire purpose of virtualization. Now, we'll move on to another form of virtualization, containerization.

www.it-ebooks.info

Creating, running, and managing Docker containers

Docker is a technology that seemed to come from nowhere and took the IT world by storm just a few years ago. The concept of containerization is not new, but Docker took this concept and made it very popular. The idea behind a container is that you can segregate an application you'd like to run from the rest of your system, keeping it sandboxed from the host operating system, while still being able to use the host's CPU and memory resources. Unlike a virtual machine, a container won't have a virtual CPU and memory of its own, as it shares resources with the host. This means that you will likely be able to run more containers on a server than virtual machines, since the resource utilization would be lower. In addition, you can store a container on a server, and allow others within your organization to download a copy of it and run it locally. This is very useful for developers who are developing a new solution, and would like others to test or run it. Since the Docker container contains everything the application needs to run, it's very unlikely that a systematic difference between one machine and another will cause the application to behave differently.

The Docker server, also known as a **Hub**, can be used remotely or locally. Normally, you'll pull down a container from the central Docker hub, which will make various containers available that are usually based on a Linux distribution or operating system. When you download it locally, you'll be able to install packages within the container or make changes to its files, as if it were a virtual machine. When you finish setting up your application within the container, you can upload it back to the Docker hub for others to benefit from, or your own local Hub for your local staff members to use. In some cases, some developers even opt to make their software available to others in the form of containers, rather than distribution-specific packages. Perhaps they find it easier to develop a container that can be used on every distribution than build separate packages for individual distributions.

Docker is only supported on 64-bit versions of Ubuntu. The required package is available in 32-bit Ubuntu, but it's not guaranteed to function properly. If you don't recall which version of Ubuntu Server you installed, run the following command:

```
uname -m
```

You should receive the following output:

```
x86_64
```


www.it-ebooks.info

Let's go ahead and get started. To set up your server to run or manage Docker containers, simply install the `docker.io` package:

```
# apt-get install docker.io
```

Yes, that's all there is to it. Installing Docker was definitely the easiest thing we've done during this entire chapter. Ubuntu includes Docker in its default repositories, so it's only a matter of installing this one package. You'll now have a new service running on your machine, simply titled `docker`. You can inspect it with the `systemctl` command, as with any other:

```
# systemctl status docker
```

Now that Docker is installed and running, let's take it for a test drive. After installing Docker, we have the `docker` command, which has various sub-commands to perform different functions. Let's try out `docker` search:

```
# docker search ubuntu
```

With this command, we are searching the Docker Hub for available containers based on Ubuntu. You could search for containers based on other distributions, such as Fedora or CentOS, if you wanted. The command will return a list of Docker images available that meet your search criteria.

Notice that I had you run the search command as `root`. This is required, unless you make your own user account a member of the `docker` group. I recommend you do that, and then log out and log in again. That way, you won't need to use `root` anymore. From this point on, I won't suggest using `root` for the remaining Docker examples. It's up to you whether you want to set up your user account with the `docker` group, or continue to run `docker` commands as `root`.

To pull down a `docker` image for our use, we can use the `docker pull` command, along with one of the image names we saw in the output of our `search` command:

```
docker pull ubuntu
```

With the preceding command, we're pulling down the latest Ubuntu container image available on the Docker hub. The image will now be stored locally, and we'll be able to create new containers from it. To create a new container from our downloaded image, use this command:

```
docker run -it ubuntu:latest /bin/bash
```


www.it-ebooks.info

Once you run that command, you'll notice that your shell prompt immediately changes. You're now within a shell prompt from your container. From here, you can run commands you would normally run within a real Ubuntu machine, such as installing new packages, changing configuration files, and more. Go ahead and play around with the container, and then we'll continue with a bit more theory on how this is actually working.

There are some potentially confusing aspects of Docker we should get out of the way first before we continue with additional examples. The most likely thing to confuse newcomers to Docker is how containers are created and destroyed. When you execute the `docker run` command against an image you've downloaded, you're actually creating a container. Each time you use the `docker run` command, you're not resuming the last container, but creating a new one. To see this in action, run a container with the `docker run` command I provided earlier, then type `exit`. Run it again, then type `exit`. You'll notice that the prompt is different each time you run the command. After the `root@ portion` of the bash prompt within the container, there is a part of a container ID. It'll be different each time you execute the `docker run` command, since you're creating a new container with a new ID each time.

To see the number of containers on your server, execute the `docker info` command. The first line of the output will tell you how many containers you have on your system, which should be the number of times you've run the `docker run` command. To see a list of all of these containers, execute the `docker ps -a` command:

```
docker ps -a
```

The output will give you the `CONTAINER ID` of each container, the `IMAGE` it was created from, the `COMMAND` being run, when the container was `CREATED` and its `STATUS`, as well as any `PORTS` you may have forwarded. The output will also display randomly generated names for each container, which are usually quite wacky. As I was going through the process of creating containers while writing this section, the code names for my containers were `tender_cori`, `serene_mcnulty`, and `high_goldwasser`. This is just one of the many quirks of Docker, and some of these can be quite hilarious.

The important output of the `docker ps -a` command is the container ID, the command, and the status. The ID allows you to reference a specific container. The command lets you know what command was being run. In our example, we executed `/bin/bash` when we started our containers. Using the ID, we can resume a container. Simply execute the `docker start` command with the container ID right after it. Your command will end up looking similar to this:

```
docker start 353c6fe0be4d
```


www.it-ebooks.info

The output will simply return the ID of the container, and then drop you back to your shell prompt—not the shell prompt of your container, but that of your server. You might be wondering at this point, how do I get back to the shell prompt for the container? We can use `docker attach` for that:

```
docker attach 353c6fe0be4d
```

You should now be within a shell prompt inside your container. If you remember from earlier, when you type `exit` to disconnect from your container, the container stops. If you'd like to exit the container without stopping it, press *Ctrl* + *P* and then press *Ctrl* + *Q* on your keyboard. You'll return to your main shell prompt, but the container will still be running. You can see this for yourself by checking the status of your containers with the `docker ps -a` command.

However, while these keyboard shortcuts work to get you out of the container, it's important to understand what a container is and what it isn't. A container is not a service running in the background, at least not inherently. A container is a collection of namespaces, such as a namespace for its filesystem or users. When you disconnect without a process running within the container, there's no reason for it to run, since its namespace is empty. Thus, it stops. If you'd like to run a container in a way that is similar to a service (it keeps running in the background), you would want to run the container in detached mode. Basically, this is a way of telling your container to run this process and don't stop running it until I tell you to. Here's an example of creating a container and running it in detached mode:

```
docker run -dit ubuntu /bin/bash
```

Normally, we use the `-it` options to create a container. This is what we used a few pages back. The `-i` option triggers interactive mode, while the `-t` option gives us a `psuedo-TTY`. At the end of the command, we tell the container to run the Bash shell. The `-d` option runs the container in the background.

It may seem relatively useless to have another Bash shell running in the background that isn't actually performing a task. But these are just simple examples to help you get the hang of Docker. A more common use case may be to run a specific application. In fact, you can even run a website from a Docker container by installing and configuring Apache within the container, including a virtual host. The question then becomes, how do you access the container's instance of Apache within a web browser? The answer is port redirection, which Docker also supports. Let's give this a try.

First, let's create a new container in detached mode. Let's also redirect port `80` within the container to port `8080` on the host:

```
docker run -dit -p 8080:80 ubuntu /bin/bash
```


www.it-ebooks.info

The command will output a container ID. This ID will be much longer than you're accustomed to seeing. This is because when we run `docker ps -a`, it only shows shortened container IDs. You don't need to use the entire container ID when you attach, you can simply use part of it as long as it's long enough to be different from other IDs:

```
docker attach dfb3e
```

Here, I've attached to a container with an ID that begins with `dfb3e`. I'm now attached to a Bash shell within the container.

Let's install Apache. We've done this before, but to keep it simple, just install the `apache2` package within your container. We don't need to worry about configuring the default sample web page or making it look nice. We just want to verify that it works. Apache should now be installed within the container. In my tests, the `apache2` daemon wasn't automatically started as it would've been on a real server instance. Since the latest container available on the Docker Hub for Ubuntu isn't yet upgraded to 16.04 at the time of writing (it's currently 14.04), the `systemctl` command won't work. We'll need to use the legacy start command for Apache:

```
# /etc/init.d/apache2 start
```

We can similarly check the status to make sure it's running:

```
# /etc/init.d/apache2 status
```

Apache should be running within the container. Now, press *Ctrl* + *P* and *Ctrl* + *Q* to exit the container, but allow it to keep running in the background. You should be able to visit the sample Apache web page for the container by navigating to `localhost:8080` in your web browser. You should see the default `It works!` page of Apache. Congratulations, you're officially running an application within a container.

Before we continue, think for a moment of all the use cases you can use Docker for. It may seem like a very simple concept (and it is), but it allows you to do some very powerful things. I'll give you a personal example. At a previous job, I worked with some Embedded Linux software engineers who had their preferred Linux distribution to run on their workstation computers. Some preferred Ubuntu, others preferred Debian, and a few even ran Gentoo. For developers, this poses a problem. The build tools are different in each distribution, because they all ship different versions of all development packages. The application they developed was only known to compile in Debian, and newer versions of the GCC compiler posed a problem for the application. My solution was to provide each developer with a Docker container based on Debian, with all the build tools baked in that they needed to perform their job. At this point, it no longer mattered which distribution they ran on their workstations. The container was the same no matter what they were running. I'm sure there are some clever use cases you can come up with.

www.it-ebooks.info

Anyway, back to our Apache container. It's now running happily in the background, responding to HTTP requests over port `8080` on the host. But, what should we do with it at this point? We can create our own image from it. Before we do so, we should configure Apache to automatically start when the container is started. We'll do this a bit differently inside the container than we would on an actual Ubuntu Server. Attach to the container and open the `/etc/bash.bashrc` file in a text editor within the container. Add the following to the very end of the file:

```
/etc/init.d/apache2 start
```

Save the file and exit your editor. Exit the container with the *Ctrl* + *P* and *Ctrl* + *Q* key combinations. We can now create a new image of the container with the `docker commit` command:

```
docker commit <Container ID> ubuntu:apache-server
```

That command will return us the ID of our new image. To view all the Docker images available on our machine, we can run the `docker images` command to have Docker return a list. You should see the original Ubuntu image we downloaded, along with the one we just created. We'll first see a column for the repository the image came from; in our case it is, Ubuntu. Next, we see tag. Our original Ubuntu image (the one we used `docker pull` to download) has a tag of latest. We didn't specify that when we first downloaded it, it just defaulted to latest. In addition, we see an image ID for both, as well as the size.

To create a new container from our new image, we just need to use `docker run`, but specify the tag and name of our new image. Note that we may already have a container listening on port `8080`, so this command may fail if that container hasn't been stopped:

```
docker run -dit -p 8080:80 ubuntu:apache-server /bin/bash
```

Speaking of stopping a container, I should probably show you how to do that as well. As you could probably guess, the command is `docker stop` followed by a container ID. This will send the `SIGTERM` signal to the container, followed by `SIGKILL` if it doesn't stop on its own after a delay:

```
docker stop <Container ID>
```

To remove a container, issue the `docker rm` command followed by a container ID. Normally, this will not remove a running container, but it will if you add the `-f` option. You can remove more than one `docker` container at a time by adding additional container IDs to the command, with a space separating each. Keep in mind that you'll lose any unsaved changes within your container if you haven't committed the container to an image yet:

```
docker rm <Container ID>
```


www.it-ebooks.info

The `docker rm` command will not remove images. If you want to remove a docker image, use the `docker rmi` command followed by an image ID. You can run the `docker image` command to view images stored on your server, so you can easily fetch the ID of the image you want to remove. You can also use the repository and tag name, such as `ubuntu:apache-server`, instead of the image ID. If the image is in use, you can remove it with the `-f` option:

```
docker rmi <Image ID>
```

Before we close our look into Docker, there's another related concept you'll definitely want to check out: `Dockerfiles`. A `Dockerfile` is a neat way of automating the building of Docker images by creating a text file with a set of instructions for their creation. The easiest way to set up a `Dockerfile` is to create a directory, preferably with a descriptive name for the image you'd like to create (you can name it whatever you wish, though), and inside it create a file named `Dockerfile`. Copy this text into your `Dockerfile` and I'll explain how it works:

```
FROM ubuntu
MAINTAINER Jay <jay@somewhere.net>

# Update the container's packages
RUN apt-get update; apt-get dist-upgrade

# Install apache2 and vim
RUN apt-get install -y apache2 vim

# Make Apache automatically start-up`
RUN echo "/etc/init.d/apache2 start" >> /etc/bash.bashrc
Let's go through this Dockerfile line by line to get a better
understanding of what it's doing.
```

We need an image to base our new image on, so we're using Ubuntu as a base. This will cause Docker to download the `ubuntu:latest` image from the Docker Hub, if we don't already have it.

```
MAINTAINER Jay <myemail@somewhere.net>
```

Here, we're setting the maintainer of the image. Basically, we're declaring its author.

```
# Update the container's packages
```

Lines beginning with a hash symbol (`#`) are ignored, so we are able to create comments within the `Dockerfile`. This is recommended to give others a good idea of what your `Dockerfile` is doing.

```
RUN apt-get update; apt-get dist-upgrade -y
```


www.it-ebooks.info

With the `RUN` command, we're telling Docker to run a specific command while the image is being created. In this case, we're updating the image's repository index and performing a full package update to ensure the resulting image is as fresh as can be. The `-y` option is provided to suppress any requests for confirmation while the command runs.

```
RUN apt-get install -y apache2 vim-nox
```

Next, we're installing both `apache2` and `vim-nox`. The `vim-nox` package isn't required, but I personally like to make sure all of my servers and containers have it installed. I mainly included it here to show you that you can install multiple packages in one line.

```
RUN echo "/etc/init.d/apache2 start" >> /etc/bash.bashrc
```

Earlier, we copied the startup command for the `apache2` daemon into the `/etc/bash.bashrc` file. We're including that here so we won't have to do this ourselves when containers are created from the image.

To build the image, we can use the `docker build` command, which can be executed from within the directory that contains the `Dockerfile`. Here's an example of using the `docker build` command to create an image tagged `packt:apache-server`:

```
docker build -t packt:apache-server
```

Once you run that command, you'll see Docker create the image for you, running each of the commands you asked it to. The image will be set up just the way you like. Basically, we just automated the entire creation of the Apache container we used as an example in this section. Once complete, we can create a container from our new image:

```
docker run -dit -p 8080:80 packt:apache-server /bin/bash
```

Almost immediately after running the container, the sample Apache site will be available on `localhost:8080` on the host. With a `Dockerfile`, you'll be able to automate the creation of your Docker images. There's much more you can do with Dockerfiles; feel free to peruse Docker's official documentation to learn more.

www.it-ebooks.info

Summary

In this chapter, we took a look at virtualization, as well as containerization. We walked through the installation of KVM and the configuration required to get our virtualization server up and running. We also took a look at Docker, which is a great way of virtualizing individual applications rather than entire servers. We installed Docker on our server, and we walked through managing containers by pulling down an image from the Docker Hub, customizing our own images, and creating Docker files to automate the deployment of Docker images. We also covered many of the popular Docker commands to manage our containers.

In *Chapter 12*, *Securing Your Server*, we'll take a look at securing our server from outside threats. We'll look into securing SSH, setting up a firewall, disk encryption, and more. Stay tuned!

www.it-ebooks.info

www.it-ebooks.info

12
Securing Your Server

It seems like every week there are new reports regarding companies getting their servers compromised. In some cases, entire databases end up freely available on the Internet, which may even include sensitive user information that can aid miscreants in stealing identities. Linux is by far a very secure platform, but it's only as secure as the administrator who sets it up. Security patches are made available every day, but someone has to install them. OpenSSH is a very handy utility, but it's also the first point of entry for an attacker who finds an insecure installation. Backups are a must-have but are also the bane of a company that doesn't secure them and then the data falls into the wrong hands. In some cases, even your own employees can cause intentional or unintentional damage. In this chapter, we'll look at some of the ways in which you can secure your server from threats. Along the way, we'll cover the following topics:

- Lowering your attack surface
- Securing OpenSSH
- Installing and configuring `Fail2ban`
- MariaDB best practices
- Setting up a firewall
- Encrypting and decrypting disks with LUKS
- Locking down `sudo`

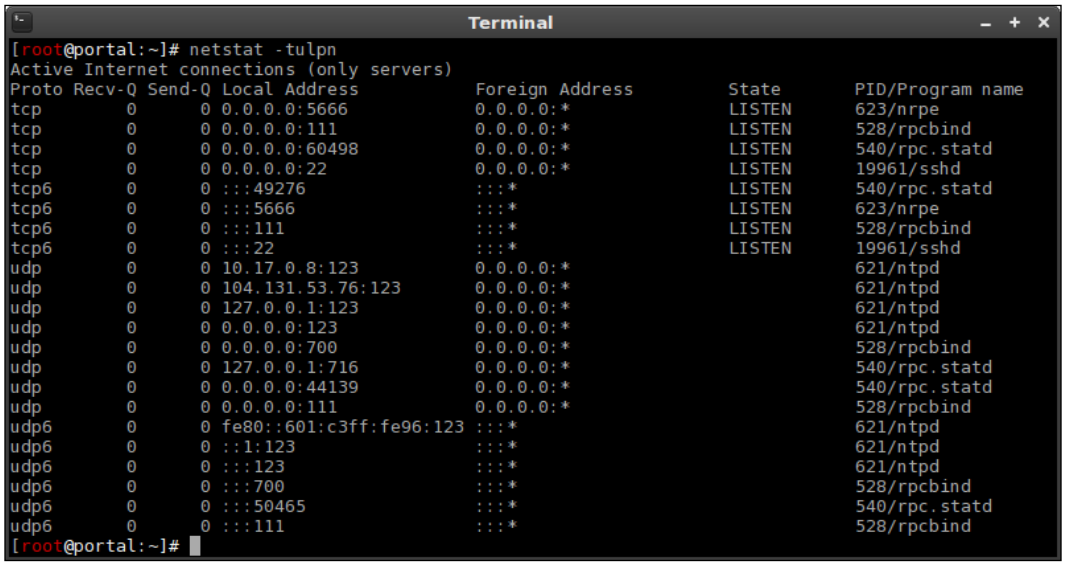
www.it-ebooks.info

Lowering your attack surface

After setting up a new server, an administrator should always perform a security check to ensure that it's as secure as it can possibly be. No administrator can think of everything and even the best among us can make a mistake, but it's always important that we do our best to ensure we secure a server as much as we can. There are many ways you can secure a server, but the first thing you should do is lower your attack surface. This means that you should close as many holes as you can, and limit the number of things that outsiders can potentially be able to access. In a nutshell, if it's not required to be available from the outside, lock it down. If it's not necessary at all, remove it.

To start inspecting your attack surface, the first thing you should do is see which ports are listening for network connections. When an attacker wants to break into your server, it's almost certain that a port scan is the first thing they will perform. They'll inventory which ports on your server are listening for connections, determine what kind of application is listening on those ports, and then try a list of known vulnerabilities to try and gain access. To inspect which ports are listening on your server, you can do a simple port query with the `netstat` command:

```
# netstat -tulpn
```



Running a port query with netstat

www.it-ebooks.info

You don't actually need to run the `netstat` command as `root` as I have, but if you do, the output will show more information than without, namely the names of the programs that are listening for connections (which makes it easier to associate ports, if you don't have them all memorized). From the output, you'll see a list of ports that are listening for connections. If the port is listening on `0.0.0.0`, then it's listening for connections from any network. This is bad. The server I took the screenshot from has several of these open ports. If the port is listening on `127.0.0.1`, then it's not actually accepting outside connections. Take a minute to inspect one of your servers with the `netstat` command, and note which services are listening for outside connections.

Armed with the knowledge of what ports your server is listening on, you can make a decision on what to do with each one. Some of those ports may be required, as the entire purpose of a server is to serve something, which usually means communicating over the network. All of these legitimate ports should be protected in some way, which usually means configuring the service with some sort of security settings (which will depend on the particular service), or enabling a firewall, which we'll get to a bit later. If any of the ports are not needed, you should close them down. You can either stop their daemon and disable it, or remove the package outright. I usually go for the latter, since it would just be a matter of reinstalling the package if I changed my mind.

This leads me to the very first step in lowering your attack surface, uninstalling unneeded packages. In my case, the server I took the screenshot from has several RPC services listening for connections. These services have to do with NFS, something that this server will never be serving or using in any way. Therefore, there's no harm in removing support for NFS from this server. It doesn't depend on it at all:

```
# apt-get remove rpcbind
```

Now that I've taken care of RPC, I'll look at other services this server is running that are listening for connections. OpenSSH is a big one. It's usually going to be the first target any attacker uses to try and gain entry into your server. What's worse, this is usually a very important daemon for an administrator to have listening. After all, how can you administer the server, if you can't connect to it? But, if you're able to connect to it, then conceivably so can someone else! What should we do? In this case, we'll want to lock the server down as much as possible. In fact, I'll be going over how to secure SSH in the next section. Of course, nothing we do will ever make a server bulletproof, but using the most secure settings we can is a great first step. Later on in this chapter, I'll also go over `Fail2ban`, which can also help add an additional layer of security to OpenSSH.

www.it-ebooks.info

As I've mentioned, I'm a big fan of removing packages that aren't needed. The more packages you have installed, the larger your attack surface is. It's important to remove anything you don't absolutely need. Even if a package isn't listed as an open port, it could be the target of some sort of vulnerability that may be affected by some other port or means. Therefore, I'll say it again: remove any packages you don't need. An easy way to get a list of all the packages you have installed is with the following command:

```
dpkg --get-selections > installed_packages.txt
```

This command will result in the creation of a text file that will contain a list of all the packages that you have installed on your server. Take a look at it. Does anything stand out that you know for sure you don't need? You most likely won't know the purpose for every single package, and there could be a thousand or more. A lot of the packages that will be contained in the text file are distribution-required packages you can't remove if you want your server to boot up the next time you go to restart it. If you don't know whether or not a package can be removed, do some research on Google. If you still don't know, maybe you should leave that package alone and move on to inspect others. By going through this exercise on your servers, you'll never really remember the purpose of every single package and library. Eventually, you'll come up with a list of typical packages most of your servers don't need, which you can make sure are removed each time you set up a new server.

Another thing to keep in mind is using strong passwords. This probably goes without saying, since I'm sure most of you that are reading this already understand the importance of strong passwords. However, I've seen hacks recently in the news caused by an administrator who set a weak password for their external facing database or web console, so you never know. The most important rule of all is that if you absolutely must have a service listening for outside connections, it absolutely must have a strong, randomly generated password. Granted, some daemons don't have a password associated with them (Apache is one example, it doesn't require authentication for someone to view a web page on port `80`). However, if a daemon does have authentication, it should have a very strong password. OpenSSH is an example of this. If you must allow external access to OpenSSH, that user account should have a strong randomly generated password. Otherwise, it will likely be taken over within a couple of weeks by a multitude of bots that routinely go around scanning for these types of things.

www.it-ebooks.info

Software within the Linux community is routinely updated with security patches. Developers publish these security updates for a reason. Sometimes updates will be released to provide us with new features. However, nine times out of ten, the update is going to be for the purpose of strengthening security. Maybe the update will close a security hole that would otherwise leave your server vulnerable. This is especially the case with Ubuntu Server 16.04, which is an **Long-term Support** (**LTS**) release. While LTS releases for Ubuntu do indeed receive updates that are released solely to add new features from time to time, feature updates are usually avoided since they also have the potential of lowering stability. Therefore, the majority of updates in an LTS release are published for the purposes of closing vulnerabilities. Install them. Typically, the best way to install updates is by running the following commands:

```
# apt-get update
# apt-get upgrade
# apt-get dist-upgrade
```

As we discussed in *Chapter 5*, *Managing Software Packages*, `apt-get` update will synchronize your Apt index with Ubuntu's repositories. The `apt-get upgrade` command will install any updated packages for your distribution, as long as they don't require the installation of dependencies that aren't already present or require the removal of another package. This is a safe first-step when installing updates. The `apt-get dist-upgrade` command will install all updates even if they do require other changes, so by running `apt-get upgrade` before `apt-get dist-upgrade`, you're getting the safe upgrades out of the way first. Then, you can run the `dist-upgrade` command to get whatever you're missing. The latter command will install updated kernel packages, which you'll also need. However, you'll need to schedule a reboot in order to benefit from a new kernel.

But how to install updates on a routine basis? The answer depends on your environment. Usually, most environments will utilize a configuration management utility, such as Ansible, Chef, or Puppet. In that case, you'd simply create an instruction in your `config` management solution to routinely install updates. Otherwise, you can safely place the `apt-get upgrade` command in a `cron` job to ensure all the low-impact updates are routinely installed, and you can make a decision on how to handle the remaining updates with the `dist-upgrade` command. Perhaps it may work out best to create clones of your production servers (for example, virtual machine copies), where you can install updates and see how they affect your server. If your organization isn't able to handle downtime, it's best to test updates before rolling them out. Creating a test lab for this purpose is ideal. If the updates pass the test phase, then they can be rolled out to production with reasonable confidence.

www.it-ebooks.info

Finally, it's important to employ the **Principle of Least Privilege** for all your user accounts. You've probably gotten the impression from several points I've made throughout the book that I distrust users. While I always want to think the best of everyone, sometimes the biggest threats can come from within (disgruntled employees, accidental deletions of critical files, and so on). Therefore, it's important to lock down services and users and allow them access to only perform the functions a user's job absolutely requires of them. This may involve but is certainly not limited to:

- Adding a user to the fewest possible number of groups
- Defaulting all network shares to read-only (users can't delete what they don't have permission to delete)
- Routinely auditing all your servers for user accounts that have not been logged into for a time
- Setting account expirations for user accounts, and requiring users to re-apply to maintain account status (this prevents hanging user accounts)
- Allowing user accounts to access as few system directories as possible (preferably none, if you can help it)
- Restricting `sudo` to specific commands (more on that later on in this chapter)

Above all, make sure you document each of the changes you make to your servers in the name of security. After you develop a good list, you can turn that list into a **Security Checklist** to serve as a baseline for securing your servers. Then, you can set reminders to routinely scan your servers for unused user accounts, unnecessary group memberships, and any newly opened ports.

Securing OpenSSH

While OpenSSH is generally more secure nowadays than it was in the past, it's still potentially a gaping hole in your server that miscreants will try to use in order to compromise your network. OpenSSH is very useful though; as administrators, we like OpenSSH because it gives us a convenient way of accessing multiple machines we manage all from one central computer. Securing OpenSSH isn't hard at all. In this section, I'll go over all the common ways in which you can secure OpenSSH on your servers. Specifically, I'll show you various tweaks you can make to the OpenSSH daemon's `config` file, which is `/etc/ssh/sshd_config` (covered in *Chapter 4, Connecting to Networks*). With each of the tweaks in this section, make sure you first search the file in order to see if the setting is there, and change it accordingly. If the setting is not present in the file, add it. After you make your changes, it's important to restart the OpenSSH daemon:

```
# systemctl restart ssh
```


www.it-ebooks.info

In the next section, I'll go over `Fail2ban`. `Fail2ban` is a really neat daemon to have running, and it adds an additional layer of security to multiple services, with OpenSSH being one. You should make sure to check out that section in this chapter after finishing this one. Okay, enough babble, let's get on with the tweaking!

First and foremost, you should change the port that OpenSSH listens on. By default, this daemon listens for connections on port `22`. This is exactly what attackers expect. Change it to some other port, preferably a high number that's above `10000` and not in use by any other service. Changing the port adds a bit of ambiguity to your server. Simply look for the port number in the `/etc/ssh/sshd_config` file and change it from its default of `22`:

```
Port 65332
```

I don't want to lure you into a false sense of security, though. Changing the SSH port won't make the service completely hidden. If an attacker is performing an in-depth port scan on your server, they'll still find it. So why change it, then? Well, first of all, it's an easy change to make. The only downsides I can think of are that you'll have to remember to specify the port number when using SSH, and you'll have to communicate the change to anyone that uses the server. To specify the port, we use the `-p` option with the `ssh` command:

```
ssh -p 65332 myhost
```

If you're using `scp`, you'll need to use an uppercase `P` instead:

```
scp -P myfile myserver:/path/to/dir
```

Even though changing the port number won't make your server bulletproof, we shouldn't underestimate the value. In a hypothetical example where an attacker is scanning servers on the Internet for an open port `22`, they'll skip your server and move on to the next. Only determined attackers that specifically want to break into your server will scan other ports looking for it. This also keeps your log file clean; you'll see intrusion attempts only from miscreants doing aggressive port scans, rather than random bots looking for open ports. If your server is Internet-facing, this will result in far fewer entries in the logs! OpenSSH logs connection attempts in the authorization log, located at `/var/log/auth.log`. Feel free to check out that log file to see what typical logging looks like. You'll need to access it as `root` or with `sudo`.

www.it-ebooks.info

Another change that's worth mentioning is which protocol OpenSSH listens for. Most versions of OpenSSH available in repositories today default to Protocol 2. This is what you want. Protocol 2 is much more secure than Protocol 1. You should never allow Protocol 1 in production under any circumstances. Chances are you're probably already using the default of Protocol 2 on your server, unless you changed it for some reason. I mention it here, just in case you have older servers still in production that are defaulting to the older protocol.

Next, I'll give you two tweaks for the price of one. There are two settings that deal with which users and groups are allowed to log in via `SSH`, `AllowUsers`, and `AllowGroups`, respectively. By default, every user you create is allowed to log in to your server via SSH. With regards to `root`, that's actually not allowed by default (more on that later). But each user you create is allowed in. Only users that must have access should be allowed in. There are two ways to accomplish this.

One option is to use `AllowUsers`. With the `AllowUsers` option, you can specifically set which users can log in to your server. With `AllowUsers` present (which is not found in the `config` file by default), your server will not allow anyone to use SSH that you don't specifically call out with that option. You can separate each user with a space:

```
AllowUsers larry moe curly
```

Personally, I find `AllowGroups` easier to manage. It works pretty much the same as `AllowUsers`, but with groups. If present, it will restrict OpenSSH connections to users who are a member of this group. To use it, you'll first create the group in question, if it doesn't already exist:

```
# groupadd sshusers
```

Then, you'll make one or more users a member of that group:

```
# usermod -aG sshusers myuser
```

Once you have added the group and made a user or two a member of that group, add the following to your `/etc/ssh/sshd_config` file, replacing the sample groups with yours. It's fine to use only one group. Just make sure you add yourself to the group before you log out, otherwise you'll lock yourself out!

```
AllowGroups admins sshusers gremlins
```

I recommend you use only one or the other. I think that it's much easier to use `AllowGroups`, since you'll never need to touch the `sshd_config` file again, you'll simply add or remove user accounts to and from the group to control access. Just so you're aware, `AllowUsers` overrides `AllowGroups`.

www.it-ebooks.info

I probably should've listed this first, but another important option is `PermitRootLogin`, which controls whether or not the `root` user account is able to make SSH connections. This should always be set to no. By default, this is usually set to `prohibit-password`, which means key authentication is allowed for `root` and passwords for `root` aren't accepted. I don't see any reason for this either. In my opinion, turn this off. Having `root` being able to log in to your server over a network connection is never a good idea. This is always the first user account attackers will try to use.

```
PermitRootLogin no
```

My next suggestion is by no means easy to set up, but worth it. By default, OpenSSH allows for users to authenticate via passwords. This is one of the first things I disable on all my servers. Allowing users to enter passwords to establish a connection means that attackers will also be able to brute-force your server. If passwords aren't allowed, then they can't do that. What's tricky is that before you can disable password authentication for SSH, you'll first need to configure and test an alternate means of authenticating, which will usually be public key authentication. This is something we've gone over in *Chapter 4*, *Connecting to Networks*. Basically, you can generate an SSH key pair on your local workstation, and then add that key to the `authorized_keys` file on the server, which will allow you in without a password. Again, refer to *Chapter 4*, *Connecting to Networks* if you haven't played around with this yet.

If you disable password authentication for OpenSSH, then public key authentication will be the only way in. If someone tries to connect to your server and they don't have the appropriate key, the server will deny their access immediately. If password authentication is enabled and you have a key relationship, then the server will ask the user for their password if their key isn't installed. In my view, after you set up access via public key cryptography, you should disable password authentication. Just make sure you test it first.

```
PasswordAuthentication no
```

Although this isn't a tip for your `/etc/ssh/sshd_config` file and it is probably an obvious point, any OpenSSH server that allows access to SSH from the Internet should have a very strong, randomly generated password. I know I've mentioned using strong passwords several times now, but I cannot *understate* the importance of strong passwords. Personally, I prefer that you never allow access to SSH from the Internet to any of your servers. But don't get me wrong; I understand that sometimes that's just not feasible and you may have one or two servers that absolutely need external SSH access. But if you do allow OpenSSH from the outside, use secure passwords! Linux is quite possibly one of the most secure platforms in existence, but nothing can save you if your password is simply `iluvcats`.

www.it-ebooks.info

There you are, those are my most recommended tweaks for securing OpenSSH. There's certainly more where that came from, but those are the settings you'll benefit from the most. In the next section, we'll add an additional layer, in the form of `Fail2ban`. With `Fail2ban` protecting OpenSSH and coupled with the tweaks I mentioned in this section, attackers will have a tough time trying to break into your server. For your convenience, here are all the OpenSSH configuration options I've covered in this section:

```
Port 65332
Protocol 2
AllowUsers larry moe curly
AllowGroups admins sshusers gremlins
PermitRootLogin no
PasswordAuthentication no
```

Installing and configuring Fail2ban

Fail2ban, how I love thee! `Fail2ban` is one of those tools that once I learned how valuable it is, I wondered how I ever lived so long without it. In the past, I used a utility known as **DenyHosts** to secure OpenSSH. `DenyHosts` protected SSH (no more, no less). It watched the server's log files, looking for authentication attempts. If it saw too many failures from a single IP address, it would create a firewall rule to block that IP. The problem was that it only protected OpenSSH. Another problem is that `DenyHosts` just kind of went away quietly. For some reason, it stopped being maintained and some distributions removed it outright. `Fail2ban` does what `DenyHosts` used to do (protect SSH) and more, as it also is able to protect other services as well.

Installing and configuring `Fail2ban` is relatively straightforward. First, install its package:

```
# apt-get install fail2ban
```

After installation, the `fail2ban` daemon will start up and be configured to automatically start at boot-time. Configuring `fail2ban` is simply a matter of creating a configuration file. But, this is one of the more interesting aspects of `Fail2ban`, you shouldn't use its default `config` file. The default file is `/etc/fail2ban/jail.conf`. The problem with this file is that it can be overwritten when you install security updates, if those security updates ever include `Fail2ban` itself. To remedy this, `Fail2ban` also reads the `/etc/fail2ban/jail.local` file, if it exists. It will never replace that file, and the presence of a `jail.local` file will supersede the `jail.conf` file. The simplest way to get started is to make a copy of `jail.conf` and save it as `jail.local`:

```
# cp /etc/fail2ban/jail.conf /etc/fail2ban/jail.local
```


www.it-ebooks.info

I'll go over some of the very important settings you should configure, so open up the `/etc/fail2ban/jail.local` file you just copied in a text editor. The first configuration item to change is located on line `50`:

```
ignoreip = 127.0.0.1/8
```

On this line, you should add additional networks that you don't want to be blocked by `Fail2ban`. `Fail2ban` is relentless; it will block any service that meets its block criteria, and it won't think twice about it. This includes blocking you. To rectify this, add your company's network here, as well as some other IP address you never want to be blocked. Make sure to leave the `localhost` IP intact:

```
Ignoreip = 127.0.0.1/8 192.168.1.0/24 192.168.1.245/24
```

In that example, I added the `192.168.1.0/24` network, as well as a single IP address of `192.168.1.245/24`. Add your networks to this line to ensure you don't lock yourself out.

Next, line `59` includes the `bantime` option. This option pertains to how many seconds a host is banned when `Fail2ban` blocks it. This option defaults to `600`:

```
bantime  = 600
```

Change this number to whatever you find reasonable, or just leave it as its default, which will also be fine. If a host gets banned, it will be banned for this specific number of seconds, until it will eventually be allowed again.

Continuing, we have the `maxretry` setting:

```
maxretry = 5
```

This is specifically the number of failures that need to occur before `Fail2ban` takes action. If a service it's watching reaches five failures, game over! The IP will be blocked for the number of seconds included in the `bantime` option. You can change this if you want to, if you don't find `5` failures to be reasonable. The highest I would set it to is `7`, for those users on your network that insist they're typing the correct password and they type the same thing over and over. Hopefully, they'll realize their error before their seventh attempt and won't need to call the helpdesk.

Skipping ahead all the way down to line `207` or thereabouts, we have the `Jails` section. From here, the `config` file will list several `Jails` you can configure, which is basically another word for something `Fail2ban` cares to pay attention to. The first is `[sshd]`, which configures its protection of the OpenSSH daemon. Look for this option underneath `[sshd]`:

```
port    = ssh
```


www.it-ebooks.info

The port being equal to `ssh` basically means that it's defaulting to port `22`. If you've changed your SSH port, change this to reflect whatever that port is. There are two such occurrences, one under `[sshd]` and another underneath `[sshd-ddos]`:

```
port    = 65332
```

Before we get too much further, I want to underscore the fact that we should test that `Fail2ban` is working after each configuration change we make. To do this, restart `Fail2ban` and then check its status:

```
# systemctl restart fail2ban
# systemctl status -l fail2ban
```

The status should always be active (running). If it's anything else, that means that `Fail2ban` doesn't like something in your configuration. Usually, that means that `Fail2ban`'s status will reflect that it exited. So, as we go, make sure to restart `Fail2ban` after each change and make sure it's not complaining about something. The status command will show lines from the log file for your convenience.

Another useful command to run after restarting `Fail2ban` is the following:

```
# fail2ban-client status
```

The output from that command will show all the jails that you have enabled. If you enable a new `Jail` in the `config` file, you should see it listed within the output of that command.

So, how do you enable a `Jail`? By default, all Jails are disabled, except for the one for OpenSSH. To enable a `Jail`, place the following within its `config` block in the `/etc/fail2ban/jail.local file`:

```
enabled = true
```

Earlier versions of Ubuntu included the line `enabled = false` for each `Jail` in the sample `jail.conf` file. In that case, you'd only need to change the line to `enabled = true` to enable a `jail`.

For example, if you want to enable the `apache-auth Jail`, find its section, and place `enabled = true` right underneath its stanza. For example, `apache-auth` will look like the following after you add the enabled line:

```
[apache-auth]
enabled = true
port     = http,https
logpath  = %(apache_error_log)s
```


www.it-ebooks.info

In that example, the `enabled = true` portion wasn't present in the default file. I added it. Now that I've enabled a new `jail`, we should restart `fail2ban`:

```
# systemctl restart fail2ban
```

Next, check its status to make sure it didn't explode on startup:

```
# systemctl status -l fail2ban
```

Assuming all went well, we should see the new `Jail` listed in the output of the following command:

```
# fail2ban-client status
```

On my test server, the output became the following once I enabled `apache-auth`:

```
Status
|- Number of jail: 2
`- Jail list:    apache-auth, sshd
```

If you enable a `Jail` for a service you don't have installed, `Fail2ban` may fail to start up. In my example, I actually did have `apache2` installed on that server before I enabled its Jail. If I hadn't, `Fail2ban` would likely exit, complaining that it wasn't able to find log files for Apache. This is the reason why I recommend that you test `Fail2ban` after enabling any Jail. If `Fail2ban` decides it doesn't like something, or something it's looking for isn't present, it may stop. Then, it won't be protecting you at all, which is not good.

The basic order of operations for `Fail2ban` is to peruse the `Jail config` file, looking for any Jails you may benefit from. If you have a daemon running on your server, there's a chance that there's a `Jail` for that. If there is, enable it and see if `Fail2ban` breaks. If not, you're in good shape. If it does fail to restart properly, inspect the status output and see what it's complaining about.

One thing you may want to do is add the `enabled = true` line to `[sshd]` and `[sshd-ddos]`. Sure, the `[sshd] Jail` is already enabled by default, but since it wasn't specifically called out in the `config` file, I don't trust it. So you might as well add an `enabled` line to be safe. There are several `Jails` you may benefit from. If you are using `SSL` with `Apache`, enable `[apache-modsecurity]`. Also, enable `[apache-shellshock]` while you're at it. If you're running your own mail server and have `Roundcube` running, enable `[roundcube-auth]` and `[postfix]`. There are a lot of default `Jails` at your disposal!

www.it-ebooks.info

Like all security applications, `Fail2ban` isn't going to automatically make your server impervious to all attacks, but it is a helpful additional layer you can add to your security regimen. When it comes to the `Jails` for OpenSSH, `Fail2ban` is worth its weight in gold, and that's really the least you should enable. Go ahead and give `Fail2ban` a go on your servers—just make sure you also white-list your own networks, in case you accidentally type your own SSH password incorrectly too many times. `Fail2ban` doesn't discriminate; it'll block anyone. Once you get it fully configured, I think you'll agree that `Fail2ban` is a worthy ally for your servers.

MariaDB best practices

MariaDB, as well as MySQL, is certainly a very useful resource to have at your disposal. However, it can also be used against you if configured improperly. Thankfully, it's not too hard to secure, but there are several points of consideration to make regarding your database server when developing your security design.

The first point is probably obvious to most of you, but I'll mention it just in case. Your database server should not be reachable from the Internet. I do understand that there are some edge cases when developing a network, and certain applications may require access to a MySQL database over the Internet. However, if your database server is accessible over the Internet, miscreants will try their best to attack it and gain entry. If there's any vulnerability in your version of MariaDB or MySQL, they'll most likely be able to hack into it.

In most organizations, a great way to implement a database server is to make it accessible by only internal servers. This means that while your web server would obviously be accessible from the Internet, its backend database should exist on a different server on your internal network and accept communications only from the web server. If your database server is a VPS, it should especially be configured to only accept communications from your web server, as VPS machines are accessible via the Internet by default. Therefore, it's still possible for your database server to be breached if your web server is also breached, but it would be less likely to be compromised if it resides on a separate and restricted server.

Some VPS providers, such as *Digital Ocean*, feature local networking, which you can leverage for your database server instead of allowing it to be accessible over the Internet. If your VPS provider features local networking, you should definitely utilize it.

www.it-ebooks.info

With regards to limiting which servers are able to access a database server, there are a few tweaks we can use to accomplish this. First, we can leverage the `/etc/hosts.allow` and `/etc/hosts.deny` files. With the `/etc/hosts.deny` file, we can stop traffic from certain networks or from specific services. With `/etc/hosts.allow`, we do the same but we allow the traffic. This works because IP addresses included in `/etc/hosts.allow` override `/etc/hosts.deny`. So basically, if you deny everything in `/etc/hosts.deny` and allow a resource or two in `/etc/hosts.allow`, you're saying, "deny everything, except resources I explicitly allow from the `/etc/hosts.deny` file".

To make this change, we'll want to edit the `/etc/hosts.allow` file first. By default, this file has no configuration other than some helpful comments. Within the file, we can include a list of resources we'd like to be able to access our server, no matter what. Make sure that you include your web server here, and also make sure that you immediately add the IP address you'll be using to SSH into the machine, otherwise you'll lock yourself out once we edit the `/etc/hosts.deny` file. Here are some example `hosts.allow` entries, with a description of what each example rule does:

```
ALL: 192.168.1.50
```

This rule allows a machine with an IP address of `192.168.1.50` to access the server:

```
ALL: 192.168.1.0/255.255.255.0
```

This rule allows any machine within the `192.168.1.0/24` network to access the server:

```
ALL: 192.168.1.
```

In this rule, we have an incomplete IP address. This acts as a wildcard, which means that any IP address beginning with `192.168.1` is allowed:

```
ALL: ALL
```

This rule allows everything. You definitely don't want to do this!

```
ssh: 192.168.1.
```

We can also allow specific daemons, as well. Here, I'm allowing OpenSSH traffic originating from any IP address beginning with `192.168.1`.

On your end, if you wish to utilize this security approach, add the resources on your database server you'll be comfortable accepting communications from. Make sure you at least add the IP address of another server with access to OpenSSH, so you'll have a way to manage the machine. You can also add all your internal IP addresses with a rule similar to the previous examples. Once you have this set up, we can edit the `/etc/hosts.deny` file.

www.it-ebooks.info

The `/etc/hosts.deny` file utilizes the same syntax as `/etc/hosts.allow`. To finish this little exercise, we can block any traffic not included in the `/etc/hosts.allow` file with the following rule:

```
ALL: ALL
```

The `/etc/hosts.allow` and `/etc/hosts.deny` files don't represent a complete layer of security, but is a great first step in securing a database server, especially one that might contain sensitive user or financial information. It's by no means specific to MySQL either, but I mention it here because databases very often contain data that if leaked, could potentially wreak havoc on your organization and even put someone out of business. A database server should only ever by accessible by the application that needs to utilize it.

Another point of consideration is user security. We walked through creating database users in *Chapter 9*, *Managing Databases*. In that chapter, we walked through the MySQL command for creating a user as well as a `grant`, performing both in one single command. This is the example I used:

```
GRANT SELECT ON mysampledb.* TO 'appuser'@'localhost' IDENTIFIED BY
'password';
```

What's important here is that we're allowing access to the `mysampledb` database by a user named `appuser`. If you look closer at the command, we're also specifying that this connection is allowed only if it's coming in from `localhost`. If we tried to access this database remotely, it wouldn't be allowed. This is a great default. But you'll also, at some point, need to access the database from a different server. Perhaps your web server and database server are separate machines, which is common in the Enterprise. You could do this:

```
GRANT SELECT ON mysampledb.* TO 'appuser'@'%' IDENTIFIED BY 'password';
```

However, in my opinion, this is a very bad practice. The `%` character in a MySQL `GRANT` command is a wildcard, similar to `*` with other commands. Here, we're basically telling our MariaDB or MySQL instance to accept connections from this user, from any network. There is almost never a good reason to do this. I've heard some administrators use the argument that they don't allow external traffic from their company firewall, so allowing MySQL traffic from any machine shouldn't be a problem. However, that logic breaks down when you consider that if an attacker does gain access to any machine in your network, they can immediately target your database server. If an internal employee gets angry at management and wants to destroy the database, they'll be able to access it from their workstation. If an employee's workstation becomes affected by malware that targets database servers, it may find your database server and try to brute-force it. I could go on and on with examples of why allowing access to your database server from any machine is a bad idea. Just don't do it!

www.it-ebooks.info

If we want to give access to a specific IP address, we can do so with the following instead:

```
GRANT SELECT ON mysampledb.* TO 'appuser'@'192.168.1.50' IDENTIFIED BY
'password';
```

With that example, only a server or workstation with an IP address of `192.168.1.50` is allowed to use the `appuser` account to obtain access to the database. That's much better. You can, of course, allow an entire subnet as well:

```
GRANT SELECT ON mysampledb.* TO 'appuser'@'192.168.1.% IDENTIFIED BY
'password';
```

Here, any IP address beginning with `192.168.1` is allowed. Honestly, I really don't like allowing an entire subnet. But depending on your network design, you may have a dozen or so machines that need access. Hopefully, the subnet you allow is not the same subnet your users' workstations use!

Finally, another point of consideration is security patches for your database server software. I know I talk about updates quite a bit, but as I've mentioned, these updates exist for a reason. Developers don't release patches for Enterprise software simply because they're bored, these updates often patch real problems that real people are taking advantage of right now as you read this. Install updates regularly. I understand that updates on server applications can scare some people, as an update always comes with the risk that it may disrupt business. But as an administrator, it's up to you to create a roll-out plan for security patches, and ensure they're installed in a timely fashion. Sure, it's tough and often has to be done after-hours. But the last thing I want to do is read about yet another company on the news with the contents of their database server leaked and posted on Pastebin. A good security design includes regular patching.

Setting up a firewall

Firewalls are a very important aspect to include in your network and security design. Firewalls are extremely easy to implement, but sometimes hard to implement well. The problem with firewalls is that they can sometimes offer a false sense of security to those who aren't familiar with the best ways to manage them. Sure, they're good to have, but simply having a firewall isn't enough by itself. The false sense of security comes when someone thinks that they're protected just because of the fact a firewall is installed and enabled, but they're also often opening traffic from any network to internal ports. Take into consideration the firewall that was introduced with Windows XP and enabled by default with Windows XP Service Pack 2. Yes, it was a good step but users simply clicked the "allow" button whenever something wanted access, which defeats the entire purpose of having a firewall. Windows implements this better nowadays, but the false sense of security it created remains to this day. Firewalls are not a "set it and forget it" solution!

www.it-ebooks.info

Firewalls work by allowing or disallowing access to a network port from other networks. Most good firewalls block outside traffic by default. When a user or administrator enables a service, they open a port for it. Then, that service is allowed in. This is great in theory, but where it breaks down is that administrators will often allow access from everywhere when they open a port. If an administrator does this, they may as well not even have a firewall at all. Think about it this way. If you need access to a server via OpenSSH, you may open up port `22` (or whatever port OpenSSH is listening on) to allow it through the firewall. But if you simply allow the port, it's open for everyone else as well. Think of most open ports as a door. If someone keeps trying to break through it, they'll eventually succeed.

When configured properly, a firewall will enable access to a port only from specific places. For example, rather than allowing port `22` for OpenSSH to your entire network, why not just allow traffic to port `22` from specific IP addresses or subnets? Now we're getting somewhere! In my opinion, allowing all traffic through a port is usually a bad idea, though some services actually do need this (such as web traffic to your web server). If you can help it, only allow traffic from specific networks when you open a port. This is where the use case for a firewall really shines.

In Ubuntu Server, the **Uncomplicated Firewall** (**UFW**) is a really useful tool for configuring your firewall. As the name suggests, it makes firewall management a breeze. To get started, install the `ufw` package:

```
# apt-get install ufw
```

By default, the UFW firewall is disabled. This is a good thing, because we wouldn't want to enable a firewall until after we've configured it. The `ufw` package features its own command for checking its status:

```
# ufw status
```

Unless you've already configured your firewall, the status will come back as inactive.

With the `ufw` package installed, the first thing we'll want to do is enable traffic via SSH, so we won't get locked out when we do enable the firewall:

```
# ufw allow from 192.168.1.156 to any port 22
```

You can probably see from that example how easy UFW's syntax is. With that example, we're allowing IP address `192.168.1.156` access to port `22` via TCP as well as UDP. In your case, you would change the IP address accordingly, as well as the port number if you're not using the OpenSSH default port. `any` in this case refers to any protocol (TCP or UDP).

www.it-ebooks.info

You can also allow traffic by subnet as well:

```
# ufw allow from 192.168.1.0/24 to any port 22
```

Although I don't recommend this, you can allow all traffic from a specific IP to access anything on your server. Use this with care, if you have to use it at all:

```
# ufw allow from 192.168.1.50
```

Now that we've configured our firewall to allow access via OpenSSH, you should also allow any other ports or IP addresses that are required for your server to operate efficiently. If your server is a web server for example, you'll want to allow traffic from ports `80` and `443`. This is one of those few exceptions where you'll want to allow traffic from any network, assuming your web server serves an external page on the Internet:

```
# ufw allow 80
# ufw allow 443
```

There are various other use patterns for the `ufw` command; refer to the man page for more. In a nutshell, these examples should enable you to allow traffic through specific ports, as well as via specific networks and IP addresses. Once you've finished configuring the firewall, we can enable it:

```
# ufw enable
Firewall is active and enabled on system startup
```

Just as the output suggests, our firewall is active and will start up automatically whenever we reboot the server.

The UFW package is basically an easy to use front-end to the `iptables` firewall, and it acts as the default firewall for Ubuntu. The commands we executed so far in this section trigger the `iptables` command, which is a command administrators can use to set up a firewall manually. A full walkthrough of `iptables` is beyond the scope of this chapter, and it's essentially unnecessary since Ubuntu features UFW as its preferred firewall administration tool and it's the tool you should use while administrating a firewall on your Ubuntu server. If you're curious, you can see what your current set of `iptables` firewall rules look like with the following command:

```
# iptables -L
```

With a well-planned firewall implementation, you can better secure your Ubuntu Server installation from outside threats. Preferably, each port you open should only be accessible from specific machines, with the exception being servers that are meant to serve data or resources to external networks. Like all security solutions, a firewall won't make your server invincible, but it does represent an additional layer attackers would have to bypass in order to do harm.

www.it-ebooks.info

Encrypting and decrypting disks with LUKS

An important aspect of security that many don't even think about is encryption. As I'm sure you know, backups are essential for business continuity. If a server breaks down, or a resource stops functioning, backups will be your saving grace. But what happens if your backup medium gets stolen or somehow falls into the wrong hands? If your backup is not encrypted, then anyone will be able to view its contents. Some data is just not sensitive, so encryption isn't always required. But anything that contains personally identifiable information, company secrets, or anything else that would cause any kind of hardship if leaked, should be encrypted. In this section, I'll walk you through setting up **Linux Unified Key Setup** (**LUKS**) encryption on an external backup drive.

Before we get into that though, I want to quickly mention the importance of full disk encryption for your distribution as well. Although this section is going to go over how to encrypt external disks, it's also possible to encrypt the volume for your entire Linux installation as well. In the case of Ubuntu, full disk encryption is an option during installation, for both the server and workstation flavors. This is especially important when it comes to mobile devices, such as laptops, which are stolen quite frequently. If a laptop is planned to store confidential data that you cannot afford to leak out, you should choose the option during installation to encrypt your entire Ubuntu installation. If you don't, anyone that knows how to boot a Live OS disc and mount a hard drive will be able to view your data. I've seen unencrypted company laptops get stolen before, and it's not a wonderful experience.

Anyway, back to the topic of encrypting external volumes. For the purpose of encrypting disks, we'll need to install the `cryptsetup` package:

```
# apt-get install cryptsetup
```

The `cryptsetup` utility allows us to encrypt and unencrypt disks. To continue, you'll need an external disk you can safely format, as encrypting the disk will remove any data stored on it. This can be an external hard disk, or a flash drive. Both can be treated the exact same way. In addition, you can also use this same process to encrypt a secondary internal hard disk attached to your virtual machine or server.

If you're using an external disk, use the `fdisk -l` command as `root` or the `lsblk` command to view a list of hard disks attached to your computer or server before you insert it. After you insert your external disk or flash drive, run the command again to determine the device designation for your removable media. In my examples, I used `/dev/sdb`, but you should use whatever designation your device was given. This is important, because you don't want to wipe out your `root` partition or an existing data partition!

www.it-ebooks.info

First, we'll need to use `cryptsetup` to format our disk:

```
# cryptsetup luksFormat /dev/sdb
```

You'll receive the following warning:

```
WARNING!
========
This will overwrite data on /dev/sdb irrevocably.


Are you sure? (Type uppercase yes):
```

Type `YES` and press *Enter* to continue. Next, you'll be asked for the passphrase. This passphrase will be required in order to unlock the drive. Make sure you use a good, randomly generated password and that you store it somewhere safe. If you lose it, you will not be able to unlock the drive. You'll be asked to confirm the passphrase.

Once the command completes, we can format our encrypted disk. At this point, it has no file-system. We'll need to create one. First, open the disk with the following command:

```
# cryptsetup luksOpen /dev/sdb backup_drive
```

The name `backup_drive` can be anything you want; it's basically just an arbitrary name you can refer to the disk as. At this point, the disk will be attached to `/dev/mapper/disk_name` where `disk_name` is whatever you called your disk in the previous command (in my case, `backup_drive`). Next, we can format the disk. The following command will create an `Ext4` filesystem on the encrypted disk:

```
# mkfs.ext4 -L "backup_drive" /dev/mapper/backup_drive
```

The `-L` option allows us to add a label to the drive, so feel free to change that label to whatever you prefer to name the drive.

With the formatting out of the way, we can now mount the disk:

```
# mount /dev/mapper/backup_drive /media/backup_drive
```

The `mount` command will mount the encrypted disk located at `/dev/mapper/backup_drive` and attach it to a mount point, such as `/media/backup_drive` in my example. The target mount directory must already exist. With the disk mounted, you can now save data onto the device as you would any other volume. When finished, you can unmount the device with the following commands:

```
# umount /media/backup_drive
# cryptsetup luksClose /dev/mapper/backup_drive
```


www.it-ebooks.info

First, we unmounted the volume just like we normally would. Then, we tell `cryptsetup` to close the volume. To mount it again, we would issue the following commands:

```
# cryptsetup luksOpen /dev/sdb backup_drive
# mount /dev/mapper/backup_drive /media/backup_drive
```

If we wish to change the passphrase, we can use the following command. Keep in mind that you should absolutely be careful typing in the new passphrase, so you don't lock yourself out of the drive. The disk must not be mounted or open in order for this to work:

```
# cryptsetup luksChangeKey /dev/sdb -S 0
```

The command will ask you for the current passphrase, and then the new one twice.

That's basically all there is to it. With the `crypsetup` utility, you can set up your own LUKS encrypted volumes for storing your most sensitive information. If the disk ever falls into the wrong hands, it won't be as bad of a situation as it would have been had the disk been unencrypted. Breaking a LUKS-encrypted volume would take considerable effort that wouldn't be feasible.

Locking down sudo

We've been using the `sudo` command throughout the book so far. In fact, we took a deeper look at it during *Chapter 2*, *Managing Users*. Therefore, I won't go into too much detail regarding `sudo` here, but some things bear repeating as `sudo` has a direct impact on security.

First and foremost, access to `sudo` should be locked down as much as possible. A user with full `sudo` access is a threat, plain and simple. All it would take is for someone with full `sudo` access to make a single mistake with the `rm` command to cause you to lose data or render your entire server useless. After all, a user with full `sudo` access can do anything `root` can do (which is everything).

By default, the user you've created during installation will be made a member of the `sudo` group. Members of this group have full access to the `sudo` command. Therefore, you shouldn't make any users a member of this group unless you absolutely have to. In *Chapter 2*, *Managing Users*, I talked about how to control access to `sudo` with the `visudo` command; refer to that chapter for a refresher if you need it. In a nutshell, you can lock down access to `sudo` to specific commands, rather than allowing your users to do everything. For example, if a user needs access to shut down or reboot a server, you can give them access to perform those tasks (and only those tasks) with the following setting:

```
charlie    ALL=(ALL:ALL) /usr/sbin/reboot,/usr/sbin/shutdown
```


www.it-ebooks.info

This line is configured via the `visudo` command, which we covered in *Chapter 2*, *Managing Users*.

For the most part, if a user needs access to `sudo`, just give them access to specific commands that are required as part of their job. If a user needs access to work with removable media, give them `sudo` access for the `mount` and `umount` commands. If they need to be able to install new software, give them access to the `apt` suite of commands, and so on. The fewer permissions you give a user, the better. This goes all the way back to the principle of least privilege that we went over near the beginning of this chapter.

Although most of the information in this section is not new to anyone that has already read *Chapter 2*, *Managing Users*, `sudo` access is one of those things a lot of people don't think about when it comes to security. The `sudo` command with full access is equivalent to giving someone full access to the entire server. Therefore, it's an important thing to keep in mind when it comes to hardening the security of your network.

Summary

In this chapter, we looked at the ways in which we can harden the security of our server. A single chapter or book can never give you an all-inclusive list of all the security settings you can possibly configure, but the examples we worked through in this chapter are a great starting point. Along the way, we looked at the concepts of lowering your attack surface, as well as the principle of least privilege. We also looked into securing OpenSSH, which is a common service that many attackers will attempt to use in their favor. We also looked into `Fail2ban`, which is a handy daemon that can block other nodes when there are a certain number of authentication failures. We also discussed configuring our firewall, using the **Uncomplicated Firewall** (**UFW**) utility. Since data theft is also unfortunately common, we covered encrypting our backup disks.

In *Chapter 13*, *Troubleshooting Ubuntu Servers*, we'll take a look at troubleshooting our server when things go wrong.

www.it-ebooks.info

www.it-ebooks.info

13
Troubleshooting Ubuntu Servers

So far, have we covered many topics, ranging from installing software to setting up services to provide value to our network. But what about when things go wrong? While it's impossible for us to account for every possible problem that may come up, there are some common places to look for clues during a time in which something goes wrong and a process isn't behaving quite as you expect it to. In this chapter, we'll take a look at some common starting points and techniques when it comes to troubleshooting issues with our servers.

In this chapter, we will cover:

- Evaluating the problem space
- Conducting a root-cause analysis
- Viewing system logs
- Tracing network issues
- Troubleshooting resource issues
- Diagnosing defective RAM

www.it-ebooks.info

Evaluating the problem space

After you identify the symptoms of the issue, the first goal in troubleshooting is to identify the problem space. Essentially, this means determining (as best you can) where the problem is most likely to reside, and how many systems and services are affected. Sometimes the problem space is obvious. For example, if none of your computers are receiving an IP address from your Ubuntu-based DHCP server, then you'll know straight away to start investigating the logs on that particular server in regards to its ability (or inability), to do the job designated for it. In other cases, the problem space may not be obvious. Perhaps you have an application that exhibits problems every now and then, but isn't something you can reliably reproduce. In that case, it may take some digging before you know just how large the scope of the problem might be. Sometimes, the culprit is the last thing you expect.

Each component on your network works together with other components, or at least, that's how it should be. A network of Linux servers, just as with any other network, is a collection of services (daemons) that compliment and often depend upon one another. For example, DHCP assigns IP addresses to all of your hosts, but it also assigns their default DNS servers as well. If your DNS server has encountered an issue, then your DHCP server would essentially be assigning a non-working DNS server to your clients. Identifying the problem space means that after you identify the symptoms, you'll also work toward reaching an understanding of how each component within your network contributes to, or is affected by, the problem. This will also help you identify the scope.

With regards to the scope, we identify how far the problem reaches, as well as how many users or systems are affected by the issue. Perhaps just one user is affected, or an entire subnet. This will help you determine the priority of the issue, and decide whether or not this is something essential that you need to fix now, or something that can wait until later. Often, prioritizing is half the battle, since each of your users will be under the impression that their issues are more important.

When identifying the problem space, as well as the scope, you'll want to answer the following questions as best as you can:

- What are the symptoms of the issue?
- When did this problem first occur?
- Were there any changes made around the network around that same time?
- Has this problem happened before? If so, what was done to fix it the last time?
- Which servers or nodes are impacted by this issue?
- How many users are impacted?

www.it-ebooks.info

If the problem is limited to a single machine, then a few really good places to start poking around is checking who is logged into the server and which commands have recently been entered. Quite often, I've found the culprit just by checking the bash history for logged on users (or users that have recently logged in). With each user account, there should be a `.bash_history` file in their `home` directory. Within this file is a list of commands that were recently entered. Check this file and see if anyone modified anything recently. I can't tell you how many times this alone has led directly to the answer. And what's even better, sometimes the bash history has led right to the solution. If a problem has occurred before and someone has already fixed it at some point in the past, chances are their efforts were recorded in the bash history, so you can see what the previous person did to solve the problem just by looking at it. To view the bash history, you can either view the contents of the `.bash_history` file in a user's `home` directory, or you can simply execute the `history` command as that user.

Additionally, if you check who is currently logged into the server, you may be able to pinpoint if someone is working on an issue already, or perhaps something they're doing caused the issue in the first place. If you enter the `w` command, you can see who is logged in to the server currently. In addition, you'll also see the IP address of the user that's logged in when you run this command. Therefore, if you don't know who corresponds to a user account listed when you run the `w` command, you can check the IP address in your DHCP server to find out who the IP address belongs to, so you can ask that person directly. In a perfect world, other administrators will send out a departmental email when they work on something to make sure everyone is aware. Unfortunately, many don't do this. By checking the logged in users as well as their bash history, you're well on your way to determining where the problem originated.

After identifying the problem space and the scope, you can begin narrowing down the issue to help find a cause. Sometimes, the culprit will be obvious. If a website stopped working and you noticed that the Apache configuration on your web server was changed recently, you can attack the problem by investigating the change and who made it. If the problem is a network issue, such as users not being able to visit websites, the potential problem space is much larger. Your Internet gateway may be malfunctioning, your DNS or DHCP server may be down, your Internet provider could be having issues, or perhaps your accounting department simply forgot to pay the Internet bill. As long as you are able to determine a potential list of targets to focus your troubleshooting on, you're well on your way to finding the issue. As we go through this chapter, I'll talk about some common issues that can come up and how to deal with them.

www.it-ebooks.info

Conducting a root-cause analysis

Once you solve a problem on your server or network, you'll immediately revel in the awesomeness of your troubleshooting skills. It's a wonderful feeling to have fixed an issue, becoming the hero within your technology department. But you're not done yet. The next step is looking toward preventing this problem from happening again. While we'll dive deeper into preventing future disasters in our next chapter, it's important to look at how the problem started as well as steps you can take in order to help stop the problem from occurring again. This is known as a **root-cause analysis**. A root-cause analysis may be a report you file with your manager or within your knowledge-base system, or it could just be a memo you document for yourself. Either way, it's an important learning opportunity.

A good root-cause analysis has several sides to the equation. First, it will demonstrate the events that led to the problem occurring in the first place. Then, it will contain a list of steps that you've completed to correct the problem. If the problem is something that could potentially recur, you would want to include information about how to prevent it from happening again in the future.

The problem with a root-cause analysis is that it's rare that you can be 100 percent accurate. Sometimes, the root-cause may be obvious. For example, suppose a user named `Bob` deleted an entire directory that contained files important to your company. If you login into the server and check the logs, you can see that `Bob` not only logged into the server, his bash history literally shows him running the `rm -rf /work/important-files` command. At this point, case is closed. You figured out how the problem happened, who did it, and you can restore the files from your most recent backup. But a root-cause is usually not that cut and dry.

One example I've personally encountered was a pair of virtual machine servers that were "fencing". At a company I once worked for, our Citrix-based virtual machine servers (which were part of a cluster), both went down at the same time, taking every Linux VM down with them. When I attached a monitor to them, I could see them both rebooting, over and over. After I got the servers to settle down, I started to investigate deeper. I read in the documentation for Citrix XenServer that you should never install a cluster of anything less than three machines, that it can create a situation exactly as I experienced. We only had two servers in that cluster, so I concluded that the servers were set up improperly and the company would need a third server if they wanted to cluster them.

www.it-ebooks.info

The problem though, is that example root-cause analysis wasn't 100 percent perfect. Were the servers having issues because they needed a third server? The documentation did mention that three servers were a minimum, but there's no way to know for sure that was the reason the problem started. However, not only was I not watching the servers when it happened, I also wasn't the individual who set them up, whom had already left the company. There was no way I could reach a 100 percent conclusion, but my root-cause analysis was sound in the sense that it was the most likely explanation (that we weren't using best practices). Someone could counter my root-cause analysis with the question "but the servers were running fine that way for several years." True, but nothing is absolute when dealing with technology. Sometimes, you never really know. The only thing you can do is make sure everything is set up properly according to the guidelines set forth by the manufacturer.

A good root-cause analysis is as sound in logic as you can be, though not necessarily bulletproof. Correlating system events to symptoms is often a good first-step, but is not necessarily perfect. After investigating the symptoms, solving the issue, and documenting what you've done to rectify it, sometimes the root-cause analysis writes itself. Other times, you'll need to read documentation and ensure that the configuration of the server or daemon that failed was implemented along with best practices. In a worst-case scenario, you won't really know how the problem happened or how to prevent it, but it should still be documented in case other details come to light later. And without documentation, you'll never gain anything from the situation.

A root-cause analysis should include details such as the following:

- A description of the issue
- Which application or piece of hardware encountered a fault
- The date and time the issue was first noticed
- What you found while investigating the issue
- What you've done to resolve the issue
- What events, configurations, or faults caused the issue to happen

www.it-ebooks.info

A root-cause analysis should be used as a learning experience. Depending on what the issue was, it may serve as an example of what not to do, or what to do better. In the case of my virtual machine server fiasco, the moral of the story was to follow best practices from Citrix and use three servers for the cluster instead of two. Other times, the end-result may be another technician not following proper directives or making a mistake, which is unfortunate. In the future, if the issue were to happen again, you'll be able to look back and remember exactly what happened last time and what you did to fix it. This is valuable, if only for the reason we're all human and prone to forgetting important details after a time. In an organization, a root-cause analysis is valuable to show stakeholders that you're able to not only address a problem, but reasonably able to prevent it from happening again.

Viewing system logs

After you identify the problem space, you can attack the potential origin of the problem. Often, this will involve reviewing log files on your server. Linux has great logging, and many of the applications you may be running are writing log files as events happen. If there's an issue, you may be able to find information about it in an application's log file.

Inside the `/var/log` directory, you'll see a handful of logs you can view, which differs from server to server depending on which applications are installed. In quite a few cases, an installed application will create its own log file somewhere within `/var/log`, either in a log file or a log file within a subdirectory of `/var/log`. For example, once you install Apache, it will create log files in the `/var/log/apache2` directory, which may give you a hint as to what may be going on if the problem is related to your web server. MySQL and MariaDB create their log files in the `/var/log/mysql` directory. These are known as **application logs**, which are basically log files created by an application and not the distribution. There are also **system logs**, such as the authorization or system logs. System logs are the log files created by the distribution, and allow you to view system events.

Viewing a log file can be done in several ways. One way is to use the `cat` command along with the path and file name of a log file. For example, the Apache access log can be viewed with the following command:

```
# cat /var/log/apache2/access.log
```


www.it-ebooks.info

One problem with the `cat` command is that it will print out the entire file, no matter how big it is. It will scroll by your terminal and if the file is large, you won't be able to see all of it. In addition, if your server is already taxed when it comes to performance, using `cat` can actually tie up the server for a bit in a case where the log file is massive. This will cause you to lose control of your shell until the file stops printing. You can press *Ctrl* + *C* to stop printing the log file, but the server may end up being too busy to respond to *Ctrl* + *C* and show the entire file anyway.

Another method is to use the `tail` command. By default, the `tail` command shows you the last ten lines of a file:

```
# tail /var/log/apache2/access.log
```

If you wish to see more than the last ten lines, you can use the `-n` option to specify a different amount. To view the last `100` lines, we would use the following:

```
# tail -n 100 /var/log/apache2/access.log
```

Perhaps one of the most useful features of the `tail` command is the `-f` option, which allows you to follow a log file. Basically, this means that as entries are written to the log file, it will scroll by in front of you. It's close to watching the log file in real time:

```
# tail -f /var/log/apache2/access.log
```

Once you start using the follow option, you'll wonder how you ever lived without it. If you're having a specific problem that you are able to reproduce, you can watch the log file for that application and see the log entries as they appear while you're reproducing the issue. In the case of a DHCP server not providing IP addresses to clients, you can view the output of the `/var/log/syslog` file (the `isc-dhcp-server` daemon doesn't have its own log file), and you can see any errors that come up as your clients try to re-establish their DHCP lease, allowing you to see the problem as it is happens.

Another useful command for viewing logs is `less`. The `less` command allows you to scroll through a log file with the page up and page down keys on your keyboard, which makes it more useful for viewing log files than the `cat` command. You can press *Q* to exit the file.

```
# less /var/log/apache2/access.log
```


www.it-ebooks.info

So now that you know a few ways in which you can view these files, which files should you inspect? Unfortunately, there's no one rule, as each application handles their logging differently. Some daemons have their own log file stored somewhere in `/var/log`. Therefore, a good place to check is in that directory, to see if there is a log file with the name of the daemon. Some daemons don't even have their own log file, and will use `/var/log/syslog` instead. You may try viewing the contents of the file, while using `grep` to find messages related to the daemon you're troubleshooting. In regards to the `isc-dhcp-server` daemon, the following would narrow down the `syslog` to messages from that specific daemon:

```
# cat /var/log/syslog |grep dhcpd
```

While troubleshooting security issues, the log file you'll definitely want to look at is the **authorization log**, located at `/var/log/auth.log`. You'll need to use the `root` account or `sudo` to view this file. The authorization log includes information regarding authentication attempts to the server, including logins from the server itself, as well as over OpenSSH. This is useful for several reasons, among them the fact that if something really bad happens on your server, you can find out who logged in to the server around that same time. In addition, if you or one of your users is having trouble accessing the server via OpenSSH, you may want to look at the authorization log for clues, as additional information for OpenSSH failures will be logged there. Often, the `ssh` command may complain about permissions of key files not being correct, which would give you an answer as to why public key authentication stopped working, as OpenSSH expects specific permissions for its files. For example, the private key file (typically `/home/<user>/.ssh/id_rsa`) should not be readable or writable by anyone other than its owning user. You'd see errors within the `/var/log/auth.log` mentioning such if that were the case.

Another use case for checking the `/var/log/auth.log` is for security, as a high number of login attempts may indicate an intrusion attempt. (Hopefully, you have `Fail2ban` installed, which we went over in the last chapter). An unusually high number of failed password attempts may indicate someone trying brute-force logging in to the server. That would definitely be a cause for concern, and you'd want to block their IP address immediately.

www.it-ebooks.info

The **system log**, located in `/var/log/syslog`, contains logging information for quite a few different things. It's essentially the Swiss army knife of Ubuntu's logs. If a daemon doesn't have its own log file, chances are its logs are being written to this file. In addition, information regarding `cron` jobs will be written here, which makes it a candidate to check when a `cron` job isn't being executed properly. The `dhclient` daemon, which is responsible for grabbing an IP address from a DHCP server, is also important. You'll be able to see from `dhclient` events within the system log when an IP address is renewed, and you can also see messages relating to failures if it's not able to obtain an IP address. Also, the `systemd init` daemon itself logs here, which allows you to see messages related to server startup as well as applications it's trying to run.

Another useful log is the `/var/log/dpkg.log` file, which records log entries relating to installing and upgrading packages. If a server starts misbehaving after you roll out updates across your network, you can view this log to see which packages were recently updated. This log will not only give you a list of updated or installed packages, but also a time-stamp from when the installation occurred. If a user installed an unauthorized application, you can correlate this log to the authentication log, to determine who logged in around that time, and then you can check that user's bash history to confirm.

Often, log files will get rotated after some time by a utility known as `logrotate`. Inside the `/var/log` directory, you'll see several log files with a `.gz` extension, which means that the original log file was compressed and renamed, and a new log file created in its place. For example, you'll see the syslog file for the system log in the `/var/log` directory, but you'll also see files named with a number and a `.gz` extension as well, such as `syslog.2.gz`. These are compressed logs. Normally, you'd view these logs by uncompressing them and then opening them via any of the methods mentioned in this section. An easier way to do so is with the `zcat` command, which allows you to view compressed files immediately:

```
# zcat /var/log/syslog.2.gz
```

There's also `zless`, which serves a similar purpose as the `less` command.

www.it-ebooks.info

Another useful command for checking logging information is `dmesg`. Unlike other log files, the `dmesg` is literally its `own` command. You can execute it from anywhere in the filesystem, and you don't even need `root` privileges to do so. The `dmesg` command allows you to view log entries from the Linux kernel's ring buffer, which can be very useful when troubleshooting hardware issues (such as seeing which disks were recognized by the kernel). When troubleshooting hardware, the system log is also helpful, but using the `dmesg` command may be a good place to check as well.

As I mentioned earlier, on an Ubuntu system there are two types of log files, system logs and application logs. System logs, such as the `auth.log` and the `dpkg.log`, detail important system events and aren't specific to any one particular application. Application logs become installed when you install their parent package, such as Apache or MariaDB. Application logs create log entries into their own log file. Some daemons you install will not create their own application log, such as `keepalived` and `isc-dhcp-server`. Since there's no general rule when it comes to which applications log is where, the first step in finding a log file is to see if the application you want log entries from creates its `own` log file. If not, it's likely using a system log.

When faced with a problem, it's important to practice viewing log files at the same time you try and reproduce the problem. Using follow mode with tail (`tail -f`) works very well for this, as you can watch the log file generate new entries as you try and reproduce the issue. This technique works very well in almost any situation where you're dealing with a misbehaving daemon. This technique can also help narrow down hardware issues. For example, I once dealt with an Ubuntu system where when I plugged in a flash drive, nothing happened. When I followed the log as I inserted and removed the flash drive, I saw the system log update and recognize each insertion and removal. So, clearly the Linux kernel itself saw the hardware and was prepared to use it. This helped me narrow down the problem to being that the desktop environment I was using wasn't updating to show the inserted flash drive, but my hardware and USB ports were operating perfectly fine. With one command, I was able to determine that the issue was a software problem and not related to hardware.

As you can see, Ubuntu contains very helpful log files which will aid you in troubleshooting your servers. Often, when you're faced with a problem, viewing relevant log entries and then conducting a Google search regarding them will result in a useful answer, or at least bring you to a bug report to let you know the problem isn't just limited to you or your configuration. Hopefully, your search results will lead you right to the answer, or at least to a work-around. From there, you can continue to work through the problem until it is solved.

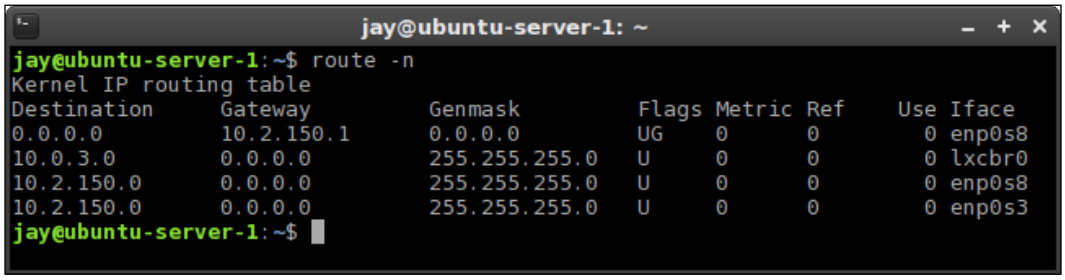
www.it-ebooks.info

Tracing network issues

It's amazing how important TCP/IP networking is to the world today. Of all the protocols in use in modern computing, it's by far the most widespread. But it's also one of the most annoying situations to figure out when it's not working well. Thankfully, Ubuntu features really handy utilities you can use in order to pinpoint what's going on.

First, let's look at connectivity. After all, if you can't connect to a network, your server is essentially completely useless. In most cases, Ubuntu recognizes just about all network cards without fail, and it will automatically connect your server or workstation to your network if it is within reach of a DHCP server. While troubleshooting, get the obvious stuff out of the way first. The following may seem like a no-brainer, but you'd be surprised how often one can miss something obvious. I'm going to assume you've already checked to make sure network cables are plugged in tight on both ends. Another aspect regarding cabling is that sometimes network cables themselves develop faults and need to be replaced. You should be able to use a cable tester and get a clean signal through the cable.

Routing issues can sometimes be tricky to troubleshoot, but by testing each destination point one by one, you can generally see where the problem lies. Typical symptoms of a routing issue may include being unable to access a device within another subnet, or perhaps not being able to get out to the Internet, despite being able to reach internal devices. To investigate a potential routing issue, first check your routing table. You can do so with the `route -n` command. This command will print your current routing table information:



Viewing the current routing table on an Ubuntu Server

In this example, you can see that the default gateway for all traffic is `10.2.150.1`. This is the first entry on the table, which tells us that all traffic to the destination `0.0.0.0` (which is everything) leaves via `10.2.150.1`. You should be able to ping this default gateway. We also see an additional subnet here, `10.2.150.0`. You should be able to ping nodes within this subnet as well.

www.it-ebooks.info

To start troubleshooting a routing issue, you would use the information shown after printing your routing table to conduct several ping tests. First, try to ping your default gateway. If you cannot, then you've found the issue. If you can, try running the `traceroute` command. This command isn't available by default, but all you'll have to do is install the `traceroute` package. After it's installed, you can run `traceroute` against a host, such as an external URL, to find out where the connection drops. The `traceroute` command should show every hop between you and your target. Each "hop" is basically another default gateway. You traverse through one gateway after another until you ultimately reach your destination. With the `traceroute` command, you can see where the chain stops. In all likelihood, you'll find that perhaps the problem isn't even on your network, but perhaps your Internet service provider is where the connection drops.

DNS issues don't happen very often, but by using a few tricks, you should be able to resolve them. Symptoms of DNS failures will usually result in a host being unable to access internal or external resources by name. Whether the problem is with internal or external hosts (or both) should help you determine whether it's your DNS server that's the problem, or perhaps the DNS server at your ISP.

The first step in pinpointing the source of DNS woes is to ping a known IP address on your network, preferably the default gateway. If you can ping it, but you can't ping the gateway by name, then you probably have a DNS issue. Also, make sure you try and ping external resources as well, such as a website. This will help you narrow down the scope of the issue.

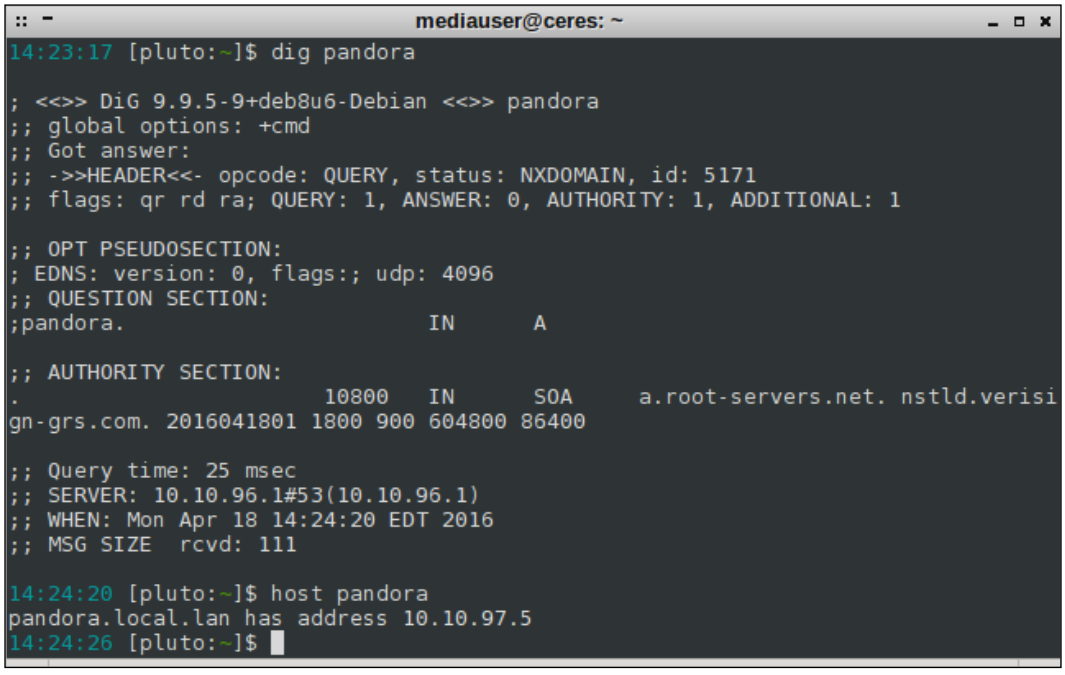
But which DNS server is your host using? To find out, check the `/etc/resolv.conf` file to determine which name servers were assigned to your machine. Are they what you expect? If not, you can temporarily fix this problem by removing the incorrect name server entries from this file, and replace them with the correct IP addresses. The reason I suggest this as a temporary fix and not a permanent one is because the next thing you'll need to do is investigate how the invalid IP addresses got there in the first place. Normally, these are assigned by your DHCP server. As long as your DHCP server is sending out the appropriate name server list, you shouldn't run into this problem. If you're using a static IP address, then perhaps there was a typo in the `/etc/resolv.conf` file.

A useful method of pinpointing DNS issues in regards to being unable to resolve external sites is to temporarily switch your DNS provider on your local machine. Normally, your machine is going to use your external DNS provider, such as the one that comes from your ISP. Your external DNS server is something we've gone through setting up in *Chapter 7, Managing Your Ubuntu Server Network*, specifically the forwarders section of the configuration for the `bind9` daemon. The forwarders used by the `bind9` daemon is where it sends traffic if it isn't able to resolve your request based on its internal list of hosts.

www.it-ebooks.info

You could consider bypassing this by changing your local workstation's DNS name servers to Google's, which are `8.8.8.8` and `8.8.4.4`. If you're able to reach the external resource after switching your name servers, you can be reasonably confident that your forwarders are the culprit. I've actually seen situations in which a web-site has changed its IP address, but the ISP's DNS servers didn't get updated quickly enough, causing some clients to be unable to reach a site they need to perform their job. Switching everyone to alternate name servers (by adjusting the forwarders option, as we did in *Chapter 7*, *Managing Your Ubuntu Server Network*) was the easiest way they could work around the issue.

Some additional tools to consider while checking your server's ability to resolve DNS entries are `dig` and `nslookup`. You should be able to use both commands to test your server's DNS settings. Both commands are used with a host name or domain name as an option. The `dig` command will present you with information regarding the address (`A`) record of the DNS zone file responsible for the IP address or domain. The `host` command should return the IP address of the host you're trying to reach. The `dig` command is also useful for troubleshooting caching. The first time you use the `dig` command, you'll see a response time (in milliseconds). The next subsequent time you run it, the response time should be much shorter.



Running the dig and hostname commands against a host in an internal network

www.it-ebooks.info

Hardware support is also critical when it comes to networking. If the Linux kernel doesn't support your network hardware, then you'll likely run into a situation where the distribution doesn't recognize or do anything when you insert a network cable, or in the case of wireless networking, doesn't show any nearby networks despite there being one or more. Unlike the Windows platform, hardware support is generally baked right into the kernel when it comes to Linux. While there are exceptions to this, the Linux kernel shipped with a distribution typically supports hardware the same age as itself or older. In the case of Ubuntu 16.04 LTS (which was released in April of 2016), it's able to support hardware released as of the beginning of 2016 and older. Future releases of Ubuntu Server will publish hardware entitlement updates, which will allow Ubuntu Server 16.04 to support newer hardware and chip-sets once it comes out. Therefore, it's always important to use the latest installation media when rolling out a new server. Typically, Ubuntu will release several point releases during the life of a supported distribution, such as 16.04.1, 16.04.2, and so on. As long as you're using the latest one, you'll have the latest hardware support that Ubuntu has made available at the time.

In other cases, hardware support may depend on external kernel modules. While it's true that the majority of Ubuntu's hardware support is baked right into the kernel, there are exceptions. The first thing you should try when faced with network hardware that's not recognized is to look up the hardware using a search engine, typically the search terms `<hardware name> Ubuntu` will do the trick. But, what do you search for? To find out the hardware string for your network device, try the `lspci` command:

```
lspci | grep -i net
```

The `lspci` command lists hardware connected to your server's PCI bus. Here, we're using the command with a case insensitive `grep` search for the word `net`:

```
lspci |grep -i net
```

This should return back a list of networking components available in your server. On my machine, for example, I get the following output:

```
01:00.1 Ethernet controller: Realtek Semiconductor Co., Ltd.
RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller (rev 12)
02:00.0 Network controller: Intel Corporation Wireless 8260 (rev 3a)
```


www.it-ebooks.info

As you can see, I have a wired and wireless network card on this machine. If one of them wasn't working, I could search online for information by searching for the hardware string and the keyword Ubuntu which should give me results pertaining to my exact hardware. If a package is required to be installed, the search results will likely give me some clues as to which package I need. Without having network access though, the worst-case scenario is that I may have to download the package from another computer, and transfer it to the server via a flash drive. That's certainly not a fun thing to need to do, but it does work if the latest Ubuntu installation media doesn't yet offer full support for your hardware.

Another potential problem point is DHCP. When it works well, DHCP is a wonderfully magical thing. When it stops working, it can be frustrating. But generally, DHCP issues often end up being a lack of available IP addresses, the DHCP daemon (`isc-dhcp-server`) not running, an invalid configuration, or hosts that have clocks that are out of sync.

If you have a server that is unable to obtain an IP address via DHCP and your network utilizes a Linux-based DHCP server, check the system log (`/var/log/syslog`) for events related to `dhcpd`. Unfortunately, there's no command you can run that I've ever been able to find that will print how many IP address leases your DHCP server has remaining, but if you run out, chances are you'll see log entries related to an exhausted pool in the system log. In addition, the system log will also show you attempts from your nodes to obtain an IP address as they attempt to do so. Feel free to use `tail -f` against the system log, to watch for any events relating to DHCP leases.

In some cases, a lack of DHCP leases being available can come down to having a very generous lease time enabled. Some administrators will give their clients up to a week for the lease time, which is generally unnecessary. A lease time of one day is fine for most networks, but ultimately the lease time you decide on is up to you. In *Chapter 7*, *Managing Your Ubuntu Server Network*, we looked at configuring our DHCP server, so feel free to refer to that chapter if you need a refresher on how to configure the `isc-dhcp-server` daemon.

www.it-ebooks.info

Although it's probably not the first thing you'll think of while facing DHCP issues, hosts having out of sync clocks can actually contribute to the problem. DHCP requests are `timestamped` on both the client and the server, so if the clock is off by a large degree on one, the time-stamps will be off as well, causing the DHCP server to become confused. Surprisingly, I've seen this come up fairly often. I recommend standardizing NTP across your network as early on as you can. DHCP isn't the only service that suffers when clocks are out of sync, file synchronization utilities also require accurate time. If you ensure NTP is installed on all of your clients and it's up to date and working, you should be in good shape. Although it's outside the scope of this book, using configuration management utilities such as Ansible, Chef, or Puppet to ensure NTP is not only configured but is running properly on all the machines in your network will only benefit you.

Of course, there are many things that can go wrong when it comes to networking, but the information here should cover the majority of issues. In summary, troubleshooting network issues generally revolves around `ping` tests. Trying to `ping` your default gateway, tracing failed endpoints with `traceroute`, and troubleshooting DNS and DHCP will take care of a majority of issues. Then again, faulty hardware such as failed network cards and bad cabling will no doubt present themselves as well.

Troubleshooting resource issues

I don't know about others, but it seems that a majority of my time troubleshooting servers usually comes down to pinpointing resource issues. By resources, I'm referring to CPU, memory, disk, input/output, and so on. Generally, issues come down to a user storing too many large files, a process going haywire that consumes a large amount of CPU, or a server running out of memory. In this section, we'll go through some of the common things you're likely to run into while administering Ubuntu servers.

First, let's revisit topics related to storage. In *Chapter 3*, *Managing Storage Volumes*, we have gone over concepts related to this already, and many of those concepts also apply to troubleshooting as well. Therefore, I won't spend too much time on those concepts here, but it's worth a refresher in regards to troubleshooting storage issues. First, whenever you have users that are complaining about being unable to write new files to the server, the following two commands are the first you should run. You are probably already well aware of these, but they're worth repeating:

```
df -h
df -i
```


www.it-ebooks.info

The first `df` command variation gives you information regarding how much space is used on a drive, in a "human readable" format (the `-h` option), which will print the information in terms of megabytes and gigabytes. The `-i` option gives you information regarding used and available inodes. The reason you should also run this, is because on a Linux system, it can report storage as full even if there's plenty of free space in terms of gigabytes. But if there are no remaining inodes, it's the same as being full, but the first command wouldn't show the usage as 100 percent when no inodes are free. Usually, the number of inodes a storage medium has available is extremely generous, and the limit is hard to hit. However, if a service is creating new log files over and over every second, or a mail daemon grows out of control and generates a huge backlog of undelivered mail, you'd be surprised how quickly inodes can empty out.

Of course, once you figure out that you have an issue with full storage, the next logical question becomes, what is eating up all my free space? The `df` commands will give you a list of storage volumes and their sizes, which will tell you at least which disk or partition to focus your attention on. My favorite command for pinpointing storage hogs, as I've mentioned in *Chapter 3*, *Managing Storage Volumes*, is the `ncdu` command. While not installed by default, `ncdu` is a wonderful utility for checking to see where your storage is being consumed the most. If run by itself, `ncdu` will can your server's entire filesystem. Instead, I recommend running it with the `-x` option, which will limit it to a specific folder as a starting point. For example, if the `/home` partition is full on your server, you might want to run the following to find out which directory is using the most space:

```
# ncdu -x /home
```

The `-x` option will cause `ncdu` to not cross filesystems. This means if you have another disk mounted within the folder you're scanning, it won't touch it. With `-x`, `ncdu` is only concerned with the target you give it.

If you aren't able to utilize `ncdu`, there's also the `du` command that takes some extra work. The `du -h` command, for example, will give you the current usage of your current working directory, with human-readable numbers. It doesn't traverse directory trees by default like `ncdu` does, so you'd need to run it on each subdirectory until you manually find the directory that's holding the most files.

www.it-ebooks.info

Another issue with storage mediums that can arise is issues with filesystem integrity. Most of the time, these issues only seem to come up when there's an issue with power, such as a server powering off unexpectedly. Depending on the server and the formatting you've used when setting up your storage volumes (and several other factors), power issues are handled differently from one installation to another. In most cases, a filesystem check (`fsck`) will happen automatically during the next boot. If it doesn't, and you're having odd issues with storage that can't be explained otherwise, a manual filesystem check is recommended. Scheduling a filesystem check is actually very easy:

```
# touch /forcefsck
```

The previous command will create an empty file, `forcefsck`, at the `root` of the filesystem. When the server reboots and it sees this file, it will trigger a filesystem check on that volume and then remove the file. If you'd like to check a filesystem other than the root volume, you can create the `forcefsck` file elsewhere. For example, if your server has a separate /home partition, you could create the file there instead to check that volume:

```
# touch /home/forcefsck
```

The filesystem check will usually complete fairly quickly, unless there's an issue it needs to fix. Depending on the nature of the problem, the issue could be repaired quickly or perhaps it will take a while. I've seen some really bad integrity issues that have taken over four hours to fix, but I've seen others fixed in a matter of seconds. Sometimes it will finish so quickly that it will scroll by so fast during boot that you may miss seeing it.

With regards to issues with memory, the free `-m` command will give you an overview of how much memory and swap is available on your server. It won't tell you what exactly is using up all your memory, but you'll use it to see if you're in jeopardy of running out. The **free** column from the output of the `free` command will show you how much memory is remaining, and allow you to make a decision on when to take action:

Output from the free command

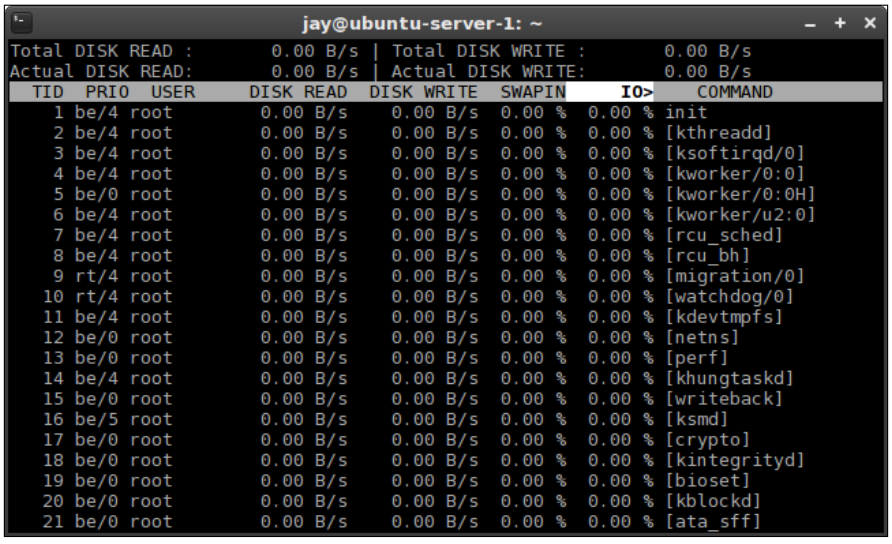
www.it-ebooks.info

In *Chapter 6, Controlling and Monitoring Processes*, we took a look at the `htop` command, which helps us answer the question of "what" is using up our resources. Using `htop` (once installed), you can sort the list of processes by CPU or memory usage by pressing *F6*, and then selecting a new sort field, such as `PERCENT_CPU` or `PERCENT_MEM`. This will give you an idea of what is consuming resources on your server, allowing you to make a decision on what to do about it. The action you take will differ from one process to another, and your solution may range from adding more memory to the server, or tuning the application to have a lower memory ceiling. But what do you do when the results from `htop` don't correlate to the usage you're seeing? For example, what if your load average is high, but no process seems to be consuming a large portion of CPU?

One command I haven't discussed so far in this book is `iotop`. While not installed by default, the `iotop` utility is definitely a must-have, so I recommend you install the `iotop` package. The `iotop` utility itself needs to be run as `root` or with `sudo`:

```
# iotop
```

The `iotop` command will allow you to see how much data is being written to or read from your disks. Input/output definitely contributes to a system's load, and not all resource monitoring utilities will show this usage. If you see a high load average but nothing in your resource monitor shows anything to account for it, check the **IO**. The `iotop` utility is a great way to do that, as if data is bottlenecked while being written to disk, that can account for a serious overhead in IO that will slow other processes down. If nothing else, it will give you an idea of which process is misbehaving, in case you need to kill it:



iotop running on an Ubuntu Server, showing a reasonably untaxed server

www.it-ebooks.info

The `iotop` window will refresh on its own, sorting processes by the column that is highlighted. To change the highlight, you'll only need to press the left and right arrows on your keyboard. You can sort processes by columns such as **IO**, **SWAPIN**, **DISK WRITE**, **DISK READ**, and others. When you're finished with the application, press *Q* to quit.

The utilities we looked at in this section are very useful when identifying issues with bottlenecked resources. What you do to correct the situation after you find the culprit will depend on the daemon. Perhaps there's an invalid configuration, or the daemon has encountered a fault and needs to be restarted. Often, checking the logs may lead you to an answer as to why a daemon misbehaves. In the case of a full storage, almost nothing beats `ncdu`, which will almost always lead you directly to the problem. Tools such as `htop` and `iotop` allow you to view additional information regarding resource usage as well, and `htop` even allows you to kill a misbehaving process right from within the application, by pressing *F9*.

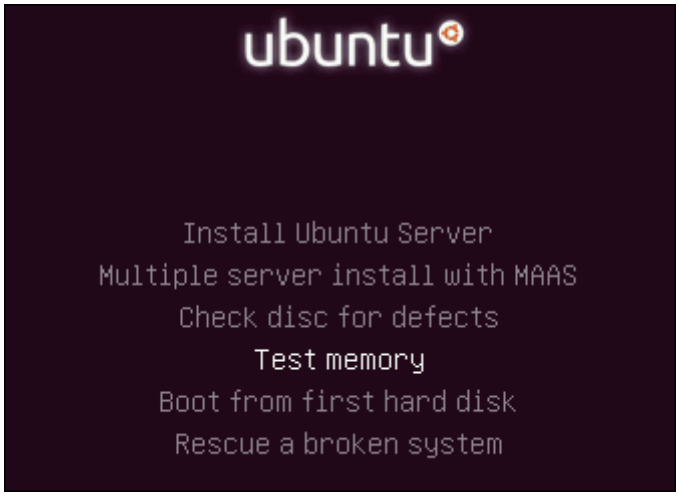
Diagnosing defective RAM

All server and computing components can and will fail eventually, but there are a few pieces of hardware that seem to fail more often than others. Fans, power supplies, and hard disks definitely make the list of common things administrators will end up replacing, but defective memory is also a situation I'm sure you'll run into eventually.

Although memory sticks becoming defective is something that could happen, I talk it is left to last in this chapter because it's one of those situations where I can't give you a definite list of symptoms to look out for that will point to memory being the source of an issue you may be experiencing. RAM issues are very mysterious in nature, and each time I've run into one, I've always stumbled across memory being bad only after troubleshooting everything else. It's for this reason that nowadays I'll often test the memory on a server or workstation first, since it's very easy to do. Even if memory has nothing to do with an issue, it's worth checking anyway since it could become a problem later.

www.it-ebooks.info

Most distributions of Linux (Ubuntu included) feature **Memtest86+** right on the installation media. Whether you create a bootable CD or flash drive, there's a memory test option available from the Ubuntu Server media. When you first boot from the Ubuntu Server media, you'll first be asked to choose your language, and then you'll be shown an installation menu. Among the choices there will be an option to **Test memory**:

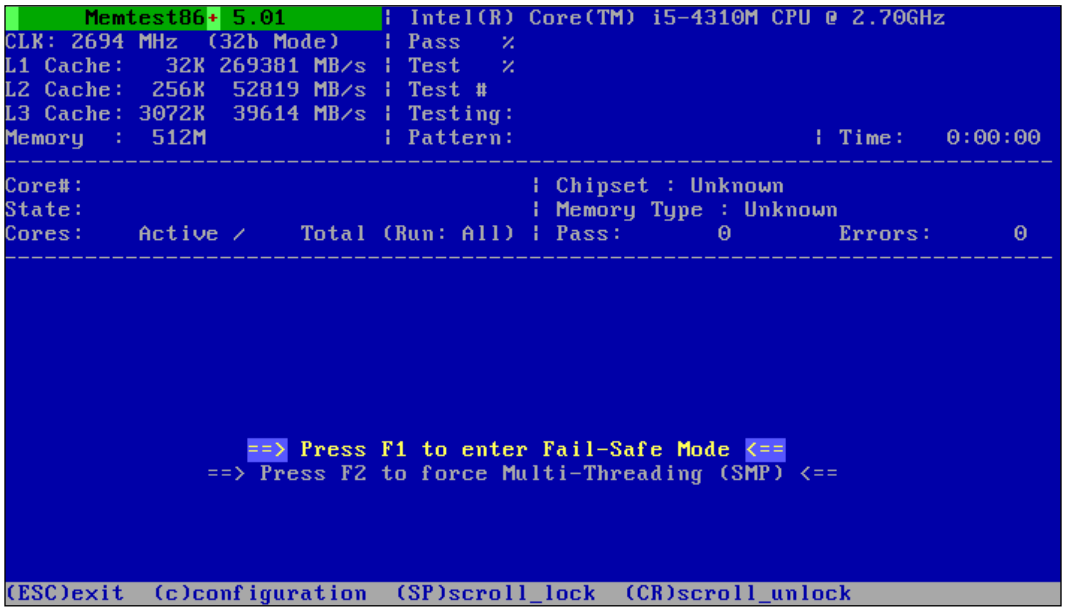


Ubuntu Server installation menu

Other editions of Ubuntu, such as the Ubuntu desktop distribution or any of its derivatives, also feature an option to test memory. Even if you don't have installation media handy for the server edition, you can use whichever version you have. From one distribution or edition to another, the **Memtest86+** program doesn't change very much.

www.it-ebooks.info

When you choose the **Test memory** option from your installation media, the **Memtest86+** program will immediately get to work and start testing your memory (press *Esc* to exit the test). The test may take a long time, depending on how much memory your workstation or server has installed. It can take minutes, or even hours to complete. Generally speaking, when your machine has defective RAM, you'll see a bunch of errors show up relatively quickly, usually within the first 5-10 minutes. If you don't see errors within 15 minutes, you're most likely in good shape. In my experience, every time I've run into defective memory, I'll see errors in 15 minutes or less (usually within 5). Theoretically though, you could very well have a small issue with your memory modules that may not show up until after 15 minutes, so you should let the test finish if you can spare the time for it.



Memtest86+ in action

The main question becomes when to run **Memtest86+** on a machine. In my experience, symptoms of bad memory are almost never the same from one machine to another. Usually, you'll run into a situation where a server doesn't boot properly, applications close unexpectedly, applications don't start at all, or perhaps an application is behaving irregularly. In my view, testing memory should be done whenever you experience a problem that doesn't necessarily seem straight-forward. In addition, you may want to consider testing the memory on your server before you roll it out into production. That way, you can assure that it starts out as free of hardware issues as possible.

www.it-ebooks.info

If the test does report errors, you'll next want to find out which memory module is faulty. This can be difficult, as some servers can have more than a dozen memory modules installed. To narrow it down, you'd want to test each memory module independently if you can, until you find out which one is defective. You should also continue to test the other modules, even after you discover the culprit. Reason being, having multiple memory modules going bad isn't outside the realm of possibility, considering whatever situation led to the first module becoming defective may have affected others.

Another tip I'd like to pass along regarding memory is that when you do discover a bad stick of memory, it's best to erase the hard disk and start over if you can. I understand that this isn't always feasible, and you could have many hours logged into setting up a server. Some servers can take weeks to rebuild, depending on their workload. But at least keep in mind that any data that passes through defective RAM can become corrupted. This means that data at rest (data stored on your hard disk) may be corrupt if it was sitting in a defective area of RAM before it was written to disk. When a server or workstation encounters defective RAM, you really can't trust it anymore. I'll leave the decision on how to handle this situation up to you (hopefully you'll never encounter it at all) but just keep this in mind as you plan your course of action. Personally, I don't trust an installation of any operating system after its hardware has encountered such issues.

I also recommend that you check the capacitors on your server's motherboard whenever you're having odd issues. Although this isn't necessarily related to memory, I mention it here because the symptoms are basically the same as bad memory when you have bad capacitors. I mean you're experiencing issues that seem erratic and don't have an obvious cause. I'm not asking you to get a voltage meter or do any kind of electrician work, but sometimes it may make sense to open the case of your server, shine a flashlight on the capacitors, and see if any of them appear to be leaking fluid or expanding. The reason I bring this up is because I've personally spent hours troubleshooting a machine (more than once) where I would test the memory and hard disk, and look through system logs without finding any obvious causes, only to later look at the hardware and discover capacitors on the motherboard were leaking. It would have saved me a lot of time if I had simply looked at the capacitors. And that's really all you have to do, just take a quick glance around the motherboard and look for anything that doesn't seem right.

www.it-ebooks.info

Summary

While Ubuntu is generally a very stable and secure platform, it's important to be prepared for when problems occur and that you know how to deal with them. In this chapter, we discussed common troubleshooting we can perform when our servers stop behaving themselves. We started off by evaluating the problem space, which gives us an understanding of how many users or servers are affected by the issue. Then, we looked into Ubuntu's log files, which are a treasure trove of information we can use to pinpoint issues and narrow down the problem. We also covered several networking issues that can come up, such as issues with DHCP, DNS, and routing.

In our final chapter, we'll take a look at preventing problems from occurring in the first place and recovering from disasters that manage to sneak through anyway. While it's impossible to prevent every problem that can possibly occur, it's critical that we understand how to make our servers as resilient as we possibly can. We'll take a look at a few approaches to accomplish this, such as backing up important files with `rsync`, recovering broken GRUB installations, as well as creating and restoring system images with Clonezilla.

www.it-ebooks.info

14
Preventing and Recovering from Disasters

In an enterprise network, a disaster can strike at any time. As administrators, we always do our best to design the most stable and fault-tolerant server implementations we possibly can, but what matters most is how we are able to deal with disasters. As stable as server hardware generally is, any component of a server can fail at any time. In the face of a disaster, we need a plan. How can you attempt to recover data from a failed disk? What do you do when your server all of a sudden decides it doesn't want to boot? How do you re-image a server quickly to get it back online? These are just some of the questions we'll answer as we take a look at several ways we can prevent and recover from disasters. In this final chapter, we'll cover the following topics:

- Preventing disasters
- Utilizing Git for configuration management
- Implementing a backup plan
- Creating system images with Clonezilla live
- Utilizing bootable recovery media

www.it-ebooks.info

Preventing disasters

As we proceed through this chapter, we'll look at ways we can recover from disasters. If we can prevent a disaster from occurring in the first place, then that's even better. We certainly can't prevent every type of disaster that can possibly happen, but having a good plan in place and following that plan will lessen the likelihood. A good disaster recovery plan will include a list of guidelines to be followed in regards to implementing new servers and managing current ones. This plan may include information such as an approved list of hardware (such as hardware configurations known to work efficiently in an environment) as well as rules and regulations for users, a list of guidelines to ensure physical and software security, proper training for end users, and methods of change control. Some of these concepts we've touched on earlier in the book, but they are worth repeating from the standpoint of disaster prevention.

First, we talked about the *Principle of least privilege* back in *Chapter 12*, *Securing Your Server*. The idea is to give your users as few permissions as possible. This is very important for security, as you want to ensure only those trained in their specific jobs are able to modify only the resources that they are required to. Accidental data deletion happens all the time. To take full advantage of this principle, create a set of groups as part of your overall security design. List departments and positions around your company, and the types of activities each are required to perform. Create system groups that correspond to those activities. For example, create an `accounting-ro` and `accounting-rw` group, for categorizing users within your accounting department that should have the ability to only read or read and write data. If you're simply managing a home file server, be careful of open network shares where users have read and write access by default. By allowing users to do as little as possible, you'll prevent a great deal of disasters right away.

Also in *Chapter 12*, *Securing Your Server*, we talked about best practices for the `sudo` command. While the `sudo` command is useful, it's often misused. By default, anyone that's a member of the `sudo` group can use `sudo` to do whatever they want. We talked about how to restrict `sudo` access to particular commands, which is always recommended. Only trusted administrators should have full access to `sudo`. Everyone else should have `sudo` permissions only if they really need it, and even then, only when it comes to commands that are required for their job. A user with full access to `sudo` can delete an entire filesystem, so it should never be taken lightly.

www.it-ebooks.info

In regards to network shares, it's always best to default to read-only whenever possible. This isn't just because of the possibility of a user accidentally deleting data, it's always possible for applications to malfunction and delete data as well. With a read-only share, the modification or deletion of files isn't possible. Additional read-write shares can be created for those that need it, but if possible, always default to read-only.

Although I've spent a lot of time discussing security in a software sense, physical security is important too. For the purposes of this book, physical security doesn't really enter the discussion much because our topic is specifically Ubuntu Server, and nothing you install in Ubuntu is going to increase the physical security of your servers. It's worth quickly noting, however, that physical security is every bit as important as securing your operating systems, applications, and data files. All it would take is someone tripping over a network cable in a server room to disrupt an entire subnet or cause a production application to go offline. Server rooms should be locked, and only trusted administrators should be allowed to access your equipment. I'm sure this goes without saying and may sound obvious, but I've worked at several companies that did not secure their server room. Nothing good ever comes from placing important equipment within arms-reach of unauthorized individuals.

In this section, I've mentioned *Chapter 12*, *Securing Your Server*, several times. In that chapter, we looked into securing our servers. A good majority of a disaster prevention plan includes a focus on security. This includes, but is not limited to, ensuring security updates are installed in a timely fashion, utilizing security applications such as failure monitors and firewalls, and ensuring secure settings for OpenSSH. I won't go over these concepts again here since we've already covered them, but essentially, security is a very important part of a disaster prevention plan. After all, users cannot break what they cannot access, and hackers will have a harder time penetrating your network if you design it in a security-conscious way.

Effective disaster prevention consists of a list of guidelines for things like user management, server management, application installations, security, and procedure documents. A full walk-through of proper disaster prevention would be an entire book in and of itself. My goal with this section is to provide you with some ideas you can use to begin developing your own plan. A disaster prevention plan is not something you'll create all at once, but is rather something you'll create and refine indefinitely as you learn more about security and what types of things to watch out for.

www.it-ebooks.info

Utilizing Git for configuration management

One of the most valuable assets on a server is its configuration. This is second only to the actual data stored on the server. Often, when we implement a new technology on a server, we'll spend a great deal of time editing configuration files all over the server to make it work as best as we can. This can include any number of things from Apache virtual host files, DHCP server configuration, DNS zone files, and more. If a server were to encounter a disaster from which the only recourse is to completely rebuild it, the last thing we'll want to do is re-engineer all of this configuration from scratch. This is where **Git** comes in.

Git is a development tool, used by software engineers everywhere for the purpose of version control for source code. In a typical development environment, an application being developed by a team of engineers can be managed by Git, with each engineer contributing to a repository that hosts the source code for their software. One of the things that makes Git so useful is how you're able to go back to previous versions of a file in an instant, as it keeps a history of all changes made to the files within the repository.

Git isn't just useful for software engineers, though. It's also a really useful tool we can leverage for keeping track of configuration files on our servers. For our use case, we can use it to record changes to configuration files, and push them to a central server for backup. When we make changes to configuration, we'll push the change back to our Git server. If, for some reason, we need to restore the configuration after a server fails, we can simply download our configuration files from Git back onto our new server. Another useful aspect of this approach is that if an administrator implements a change to a configuration file that breaks a service, we can simply revert back to a known-working commit and we'll be immediately back up and running.

Configuration management on servers is so important, in fact, I highly recommend that every Linux administrator takes advantage of version control for this purpose. Although it may seem a bit tricky at first, it's actually really easy to get going once you practice with it. Once you've implemented Git for keeping track of all your server's configuration files, you'll wonder how you've ever lived without it. I'll walk you through what you'll need to do in order to implement this approach. To get started, you'll want to install the `git` package:

```
# apt-get install git
```


www.it-ebooks.info

For this to work effectively, you'll need a Git server. You don't necessarily have to dedicate a server just for this purpose, you can use an existing server. The only important aspect here is that you have a central server onto which you can store your Git repositories. All your other servers will need to be able to reach it via your network. Some administrators will use GitHub for this, and while that may be fine for some, I'm going to walk you through using standard Git instead, as often you'll want to keep your configuration files private, rather than have them publicly hosted on GitHub. GitHub does offer private repositories, but for a fee. Setting up your own Git server won't cost you anything but time and disk space. On whatever machine you've designated as your Git server, install the `git` package on it as well. Believe it or not, that's all there is to it. Since Git uses OpenSSH by default, we only need to make sure that the `git` package is installed on the server as well as our clients. We'll need a directory on that server to house our Git repositories, and the users on your servers that utilize Git will need to be able to modify that directory.

Now, think of a configuration directory that's important to you, that you want to place into version control. A good example is the `/etc/apache2` directory on a web server. That's what I'll use in my examples in this section. But you're certainly not limited to that. Any configuration directory you would rather not lose is a good candidate. If you choose to use a different configuration path, change the paths I give you in my examples to that path.

On the server, create a directory to host your repositories. I'll use `/git` in my examples:

```
# mkdir /git
```

Next, you'll want to modify this directory to be owned by the administrative user you use on your Ubuntu servers. Typically, this is the user that was created during the installation of the distribution. You can use any user you want actually, just make sure this user is allowed to use OpenSSH to access your Git server. Change the ownership of the `/git` directory so it is owned by this user. My user on my Git server is `jay`, so in my case I would change the ownership with the following command:

```
# chown jay:jay /git
```

Next, we'll create our Git repository within the `/git` directory. For Apache, I'll create a bare repository for it within the `/git` directory. A bare repository is basically a skeleton of a Git repository, that doesn't contain any useful data, just some default configuration to allow it to act as a Git directory. To create the bare repository, `cd` into the `/git` directory, and execute:

```
git init --bare apache2
```


www.it-ebooks.info

You should see the following output:

```
Initialized empty Git repository in /git/apache2/
```

That's all we need to do on the server for now for the purposes of our Apache repository. On your client (the server that houses the configuration you want to place under version control), we'll copy this bare repository by cloning it. To set that up, create a `/git` directory on your Apache server (or whatever kind of server you're backing up), just as we did before. Then, `cd` into that directory, and clone your repository with the following command:

```
git clone 192.168.1.101:/git/apache2
```

For that command, replace the IP address with either the IP address of your Git server, or its hostname if you've created a DNS entry for it. You should see the following output, warning us that we've cloned an empty repository:

```
warning: You appear to have cloned an empty repository.
```

This is fine, we haven't actually added anything to our repository yet. If you were to `cd` into the directory we just cloned, and list its storage, you'd see it as an empty directory. If you use `ls -a` to view hidden directories as well, you'll see a `.git` directory inside. Inside the `.git` directory, we'll have configuration items for Git that allow this repository to function properly. For example, the `config` file within the `.git` directory contains information on where the remote server is located. We won't be manipulating this directory, I just wanted to give you a quick overview on what its purpose is. Note that if you delete the `.git` directory within your cloned repository, that basically removes version control from the directory and makes it a normal directory.

Anyway, let's continue. We should first make a backup of our current `/etc/apache2` directory on our web server, in case we make a mistake while converting it to being version controlled:

```
# cp -rp /etc/apache2 /etc/apache2.bak
```

Then, we can move all the contents of `/etc/apache2` into our repository:

```
# mv /etc/apache2/* /git/apache2/
```

The `/etc/apache2` directory is now empty. Be careful not to restart Apache at this point, it won't see its configuration files, and will fail to restart. Remove the (now empty) `/etc/apache2` directory:

```
# rm /etc/apache2
```


www.it-ebooks.info

Now, let's make sure that Apache's files are owned by `root`. The problem though, is if we use the `chown` command as we normally would to change ownership, we'll also change the `.git` directory to be owned by `root` as well. We don't want that, because the user responsible for pushing changes should be the owner of the `.git` folder. The following command will change the ownership of the files to `root`, but won't touch hidden directories such as `.git`:

```
# find /git/apache2 -name '.?*' -prune -o -exec chown root:root {} +
```

When you list the contents of your repository directory now, you should see that all files are owned by `root`, except for the `.git` directory, which should be owned by your administrative user account.

Next, create a symbolic link to your Git repository so the `apache2` daemon can find it:

```
# ln -s /git/apache2 /etc/apache2
```

At this point, you should see a symbolic link for Apache, located at `/etc/apache2`. If you list the contents of `/etc` while grepping for `apache2`, you should see it as a symbolic link:

```
ls -l /etc | grep apache2
lrwxrwxrwx 1 root root  37 2016-06-25 20:59 apache2 -> /git/apache2
```

If you reload Apache, nothing should change, since it should find the same configuration files as it did before, since its directory in `/etc` maps to `/git/apache2` which includes the same files it did before:

```
# systemctl reload apache2
```

If you see no errors, you should be all set. Otherwise, make sure you created the symbolic link properly.

Next, we get to the main attraction. We've copied Apache's files into our repository, but we didn't actually push those changes back to our Git server yet. To set that up, we'll need to associate the files within our `/git/apache2` directory into version control. The reason for this is because simply having files stored within the `git` repository folder isn't enough for Git to care about them. We have to tell Git to pay attention to individual files. We can add every file within our Git repository for Apache by entering the following command from within that directory:

```
git add .
```


www.it-ebooks.info

This basically tells Git to add everything in the directory to version control. You can actually do the following to add an individual file, if you wanted to add files one at a time:

```
git add <filename>
```

In this case, we want to add everything, so we used a period in place of a directory name to add the entire current directory.

If you run the `git status` command from within your Git repository, you should see output indicating that Git has new files that haven't been committed yet. A `git` commit simply finalizes the changes locally. Basically, it packages up your current changes to prepare them for being copied to the server. To create a commit of all the files we've added so far, `cd` into your `/git/apache2` directory, and run the following to stage a new commit:

```
git commit -a -m "My first commit."
```

With this command, the `-a` option tells Git that you want to include anything that's changed in your repository. The `-m` command allows you to attach a message to the commit, which is actually required.

Finally, we can push our changes back to the Git server:

```
git push origin master
```

The `git push` command takes the contents of our commit, and pushes everything up to the server. After a commit has been pushed, anyone who downloads a copy of the Git repository will receive the latest changes. If we've already downloaded a Git repository, we can have it check in to the server to download any changes that may have been made with the following command:

```
git pull
```

When we execute a `git pull`, we're telling Git to check in with the master (the server), and check to see if our local copy of the repository is up to date with the master. If it's not, Git will download any recent changes.

What you may find interesting if you've never worked with Git before, is that the server copy of a repository won't include the same file structure as what you've uploaded. The server repository is more or less a database of changes, with instructions of how to produce the actual files. This is why when you list the contents of the Git repository on the server side, you'll see a completely different listing of files. Whenever you clone a Git repository though, the resulting directory structure will be just as you left it. Git will reassemble the files exactly as they were when you clone the repository to a client.

www.it-ebooks.info

From this point forward, if you need to restore a repository onto another server, all you should need to do is perform a Git clone. To clone the repository into your current working directory, execute the following:

```
git clone 192.168.1.101:/git/apache2
```

Now, each time you make changes to your configuration files, you can perform a `git commit` and then `git push` to make the changes to the server to keep the content safe:

```
git commit -a -m "Updated config files."
git push origin master
```

If, for some reason, you aren't able to download a Git repository from your server via SSH, check the authorization log (`/var/log/auth.log` for clues). Since Git is using SSH by default, it will depend on the underlying connection in order to work, and the authorization log on the server will often include helpful information on why a connection may not be working.

Now we know how to create a repository, `push` changes to a server, and `pull` the changes back down. Finally, we'll need to know how to revert changes should our configuration be changed with non-working files. First, we'll need to locate a known-working commit. My favorite method is using the `tig` command. The `tig` package must be installed for this to work, but it's a great utility to have at your disposal:

```
# apt-get install tig
```

The `tig` command (which is just `git` backwards), gives us a semi-graphical interface to browse through our Git commits. To use it, simply execute the `tig` command from within a Git repository. In the following example screenshot, I've executed `tig` from within a repository for a DNS server's `bind9` daemon:

```
2016-04-11 14:04 Jay LaCroix o [master] {origin/master} Decommissioned glycon.
2016-04-07 09:55 Jay LaCroix o Removed decommissioned hosts.
2016-04-04 15:39 Jay LaCroix o Added new digital ocean servers.
2016-03-29 12:02 Jay LaCroix o Changed hostname of RCA base station.
2016-03-29 11:32 Jay LaCroix o Switched DNS to servers from alternate-dns.com.
2016-03-29 11:30 Jay LaCroix o Added IP address for desk phone.
2016-03-22 12:15 Jay LaCroix o Added iris.
2016-03-17 10:41 Jay LaCroix o Removed portal.
```

An example of the tig command, looking at a repository for the bind9 daemon

www.it-ebooks.info

While using `tig`, you'll see a list of Git commits, along with their dates and comments that were entered with each. To inspect one, press the up and down arrow keys to change your selection, then press *Enter* on the one you want to view. You'll see a new window, which will show you the commit hash (which is a long string of alpha-numeric characters), as well as an overview of which lines were added or removed from the files within the commit. To revert one, you'll first need to find the commit you want to revert to, and get its commit hash. The `tig` command is great for finding this information. In most cases, the commit you'll want to revert to is the one before the change took place. In my example screenshot, I removed some decommissioned hosts on **4/7/2016**. If I want to restore that file, I should revert to the commit before that, on **4/4/2016**. I can get the commit hash by highlighting that entry and pressing *Enter*. It's at the top of the window. Then, I can exit `tig` by pressing *Q*, and then revert to that commit. To checkout a specific commit, I can execute the `git checkout` command along with the hash for the commit:

```
git checkout  356dd6153f187c1918f6e2398aa6d8c20fd26032
```

And just like that, the entire directory tree for the repository instantly changes to exactly what it was before the bad commit took place. I can then restart or reload the daemon for this repository, and it will be back to normal. At this point, you'd want to test the application to make sure that the issue is completely fixed. After some time has passed and you're finished testing, you can make the change permanent. To do so, we will first switch back to the most recent commit:

```
git checkout master
```

Then, we permanently switch master back to the commit that was found to be working properly:

```
git revert --no-commit 356dd6153f187c1918f6e2398aa6d8c20fd26032
```

Then, we can commit our reverted Git repository, and `push` it back to the server:

```
git commit -a -m "The previous commit broke the application. Reverting."
git push origin master
```


www.it-ebooks.info

As you can see, Git is a very useful ally to utilize when managing configuration files on your servers. This benefits disaster recovery, because if a bad change is made that breaks a daemon, you can easily revert the change. If the server were to fail, you can recreate your configuration almost instantly by just cloning the repository again. There's certainly a lot more to Git than what we've gone over in this section, so feel free to pick up a book about it if you wish to take your knowledge to the next level. But in regards to managing your configuration with Git, all you'll need to know is how to place files into version control, update them, and clone them to new servers. Some services you run on a server may not be good candidates for Git, however. For example, managing an entire MariaDB database via Git would be a nightmare, since there is too much overhead with such a use case and database entries will likely change too rapidly for Git to keep up. Use your best judgment. If you have some configuration files that are only manipulated every once in a while, they'll be a perfect candidate for Git.

Implementing a backup plan

Creating a solid backup plan is one of the most important things you'll ever do as a server administrator. Even if you're only using Ubuntu Server at home as a personal file server, backups are critical. During my career, I've seen disks fail many times. I'll often hear arguments about which hard disk manufacturer beats others in terms of longevity, but I've seen disk failures so often, I don't trust any of them. All disks will fail eventually, it's just a matter of when. And when they do fail, they'll usually fail hard with no easy way to recover data from them. Later on in this chapter, we'll look at a few data recovery tricks, but honestly, if you need to recover data from a failed disk, your approach is already wrong. A sound approach to managing data is such that any disk or server can fail, and it won't matter, since you'll be able to regenerate your data from other sources, such as a backup or secondary server.

There's no one best backup solution, since it all depends on what kind of data you need to secure, and what software and hardware resources are available to you. For example, if you manage a database that's critical to your company, you should back it up regularly. If you have another server available, set up a replication slave so that your primary database isn't a single point of failure. Not everyone has an extra server lying around, so sometimes you have to work with what you have available. This may mean that you'll need to make some compromises, such as creating regular snapshots of your database server's storage volume, or regularly dumping a backup of your important databases to an external storage device.

www.it-ebooks.info

The `rsync` utility is one of the most valuable pieces of software around to server administrators. It allows us to do some very wonderful things. In some cases, it can save us quite a bit of money. For example, online backup solutions are wonderful in the sense that we can use them to store off-site copies of our important files. However, depending on the volume of data, they can be quite expensive. With `rsync`, we can back up our data in much the same way, with not only our current files copied over to a backup target, but also differentials as well. If we have another server to send the backup to, even better.

At one company I've managed servers for, they didn't want to subscribe to an online backup solution. To work around that, a server was set up as a backup point for `rsync`. We set up `rsync` to back up our file server to a secondary server. Once the initial backup was complete, the secondary server was sent to one of our other offices in another state. From that point forward, we only needed to run `rsync` weekly, to back up everything that had been changed since the last backup. Sending files via `rsync` to the other site over the Internet was rather slow, but since the initial backup was already complete before we sent the server there, all we needed to back up each week was differentials. This is not only an example of how awesome `rsync` is and how we can configure it to do pretty much what paid solutions do, but also the experience was a good example of utilizing what you have available to you.

We've already gone over `rsync` in *Chapter 8*, *Accessing and Sharing Files*, so I won't repeat too much of that information here. But since we're on the subject of backing up, the `--backup-dir` option is worth mentioning again. This option allows you to copy files that would normally be replaced and copy them to another location. As an example, here's the `rsync` command I mentioned in *Chapter 8*, *Accessing and Sharing Files*:

```
CURDATE=$(date +%m-%d-%Y)

rsync -avb --delete --backup-dir=/backup/incremental/$CURDATE /src /
target
```


www.it-ebooks.info

This command was part of the topic of creating an `rsync` backup script. The first command simply captures today's date and stores it into a variable named `$CURDATE`. In the actual `rsync` command, we refer to this variable. The `-b` option (part of the `-avb` option string), tells `rsync` to make a copy of any file that would normally be replaced. If `rsync` is going to replace a file on the target with a new version, it will move the original file to a new name before overwriting it. When it's about to overwrite a file, the `--backup-dir` option tells `rsync`, to put it somewhere else instead of copying it to a new name. We give the `--backup-dir` option a path, where we want the files that would normally be replaced to be copied to. In this case, the backup directory includes the `$CURDATE` variable, which will be different every day. For example, a backup run on `5/28/2016` would have a backup directory of the following path, if we used the command I gave as an example:

```
/backup/incremental/5-28-2016
```

This essentially allows you to keep differentials. Files on `/src` will still be copied to `/target`, but the directory you identify as a `--backup-dir` will contain the original files before they were replaced that day.

On my servers, I use the `--backup-dir` option with `rsync` quite often. I'll typically set up an external backup drive, with the following three folders:

```
current
archive
logs
```

The `current` directory always contains a current snapshot of the files on my server. The `archive` directory on my backup disks is where I point the `--backup-dir` option to. Within that directory will be folders named with the dates that the backups were taken. The `logs` directory contains log files from the backup. Basically, I redirect the output of my `rsync` command to a log file within that directory, each log file being named with the same `$CURDATE` variable so I'll also have a backup log for each day the backup runs. I can easily look at any of the logs for which files were modified during that backup, and then traverse the archive folder to find an original copy of a file. I've found this approach to work very well. Of course, this backup is performed with multiple backup disks that are rotated every week, with one always off-site. It's always crucial to keep a backup off-site in case of a situation that could compromise your entire local site.

www.it-ebooks.info

The `rsync` utility is just one of many you can utilize to create your own backup scheme. The plan you come up with will largely depend on what kind of data you want to protect and what kind of downtime you're willing to endure. Ideally, we would have an entire warm-site with servers that are carbon-copies of our production servers, ready to be put into production should any issues arise. However, that's also very expensive, and whether or not you can implement such a routine will depend on your budget. However, Ubuntu has many great utilities available that you can use to come up with your own system that works. If nothing else, utilize the power of `rsync` to back up to external disks and/or external sites.

Creating system images with Clonezilla live

Clonezilla is a wonderful solution for backing up entire disks or partitions. Using Clonezilla, you can recover a server to bare-metal, including not only data, but also the entire operating system. Clonezilla essentially allows you to create hard disk images, similar to paid tools such as **Acronis**. It can be used as either bootable live media, or it can be installed on a server and clients can reach it via **Preboot eXecution Environment** (**PXE**) boot.

Clonezilla itself is not actually related to Ubuntu in any way, but I bring it up in this book because this tool is extremely useful and a disk imaging solution is a very worthy aspect of a good deployment scheme. Using Clonezilla, you can use disk images to greatly reduce setup time, and also to back up complex setups. For example, imagine that you typically have a check-list of 20 configuration tweaks that need to be made to each server before you put one into production. With Clonezilla, you can set up one server with all the tweaks you need for your network, and create an image of it. Then, for any additional server you wish to deploy, you can simply restore the Clonezilla image with all your tweaks baked-in, and never have to do that work again.

Another good example is for workstation operating systems. Perhaps you have a standard operating system with a pre-defined set of applications. You can configure all the tweaks and applications you use in any environment, and save it to an image. Deploying end-user workstations then becomes a breeze. While I bring up Clonezilla in the context of deploying Ubuntu Server and Ubuntu Workstation, it's not specific to Linux at all. I've seen it used for creating base images for Windows as well as Mac OS X. While I won't go over the latter two use cases, just keep in mind that in most cases Clonezilla is flexible and doesn't care which operating system you're imaging. An imaging solution benefits disaster recovery because it allows us to get a new machine up and running quickly when one fails.

www.it-ebooks.info

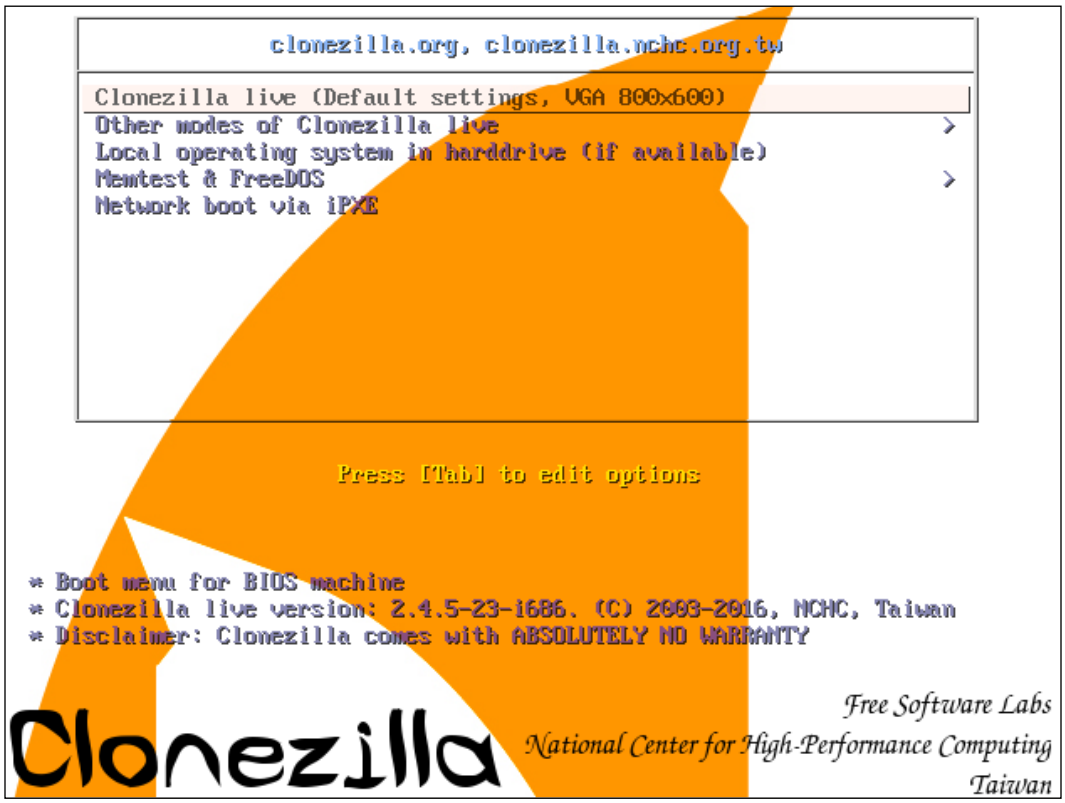
As I've mentioned, there are two styles of use with Clonezilla. You can create live media and boot a computer from it, and save images to or restore images from a network share. Additionally, you can set up Clonezilla as a server and clients can use PXE boot to boot to Clonezilla over the network. In this section, I'll be going over the former, using live media. In my opinion, there is very little value in setting up a Clonezilla server. The reason being that, the overhead is extreme for something like an imaging solution. Sure, setting up a Clonezilla server seems nice in theory, but having to maintain such a server is added overhead you don't need. It's not much slower to boot from external media than the network. In addition, documentation for setting up Clonezilla as a server is often sparse and unreliable. If I wrote a walk-through for setting up such a server, chances are the instructions would change again by the time this book were printed. Take it from me—spare yourself the headache and use the live media method!

With the live media method, you'll create Clonezilla media from either an ISO or ZIP file. If you prefer a bootable CD-ROM, you'll use the ISO file. If you need to create a bootable USB flash drive, you'll use the ZIP file version. I'm assuming by this point in our adventure that you already know how to use an ISO file to create a bootable CD. If you need to use flash media instead, you can refer to the documentation on Clonezilla's site for creating media. I won't go over creating boot media here, since everyone's computer and installed applications are different. In summary though, creating a bootable CD for Clonezilla is done the same way as any other Linux distribution. You can find the necessary files at the Clonezilla website `http://clonezilla.org/`.

The next thing you'll need to use for Clonezilla is a network file share location. In *Chapter 8*, *Accessing and Sharing Files*, we took a look at sharing files with NFS as well as Samba. Clonezilla prefers a share named `images`, though you can name it whatever you wish (you'll just have to type it in when Clonezilla asks). Basically, you'll need a file share on another computer or server on your network where Clonezilla can read and save its files. The easiest way to do this is to create a Samba share on a Linux server, or a regular shared folder on a Windows machine. Clonezilla can access both Samba and Window's shares, but needs to be granted read and write access to it. During the process, Clonezilla will ask you for a username and password for the share, so be sure you have an account created on the server that Clonezilla will be able to use. If in doubt, create a user named `clonezilla` on your file server, with a password, and full access to your images share.

www.it-ebooks.info

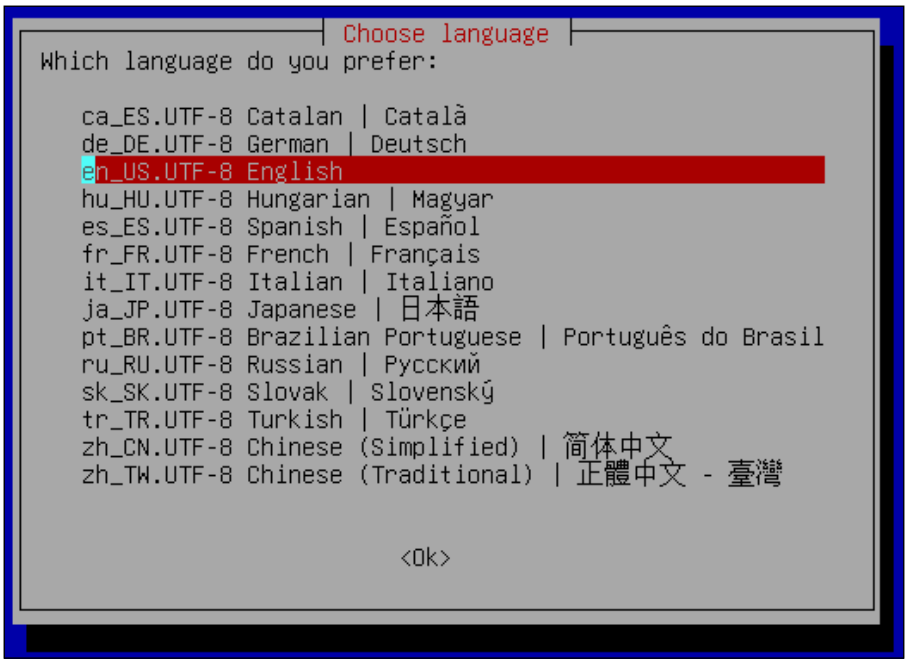
Now I'll walk you through the process of creating a new disk image. To get started, set up an operating system on a reference computer. This is going to be the setup you'll want to deploy to other machines. It really doesn't matter what you use as the reference system. But for fun, try setting up an Ubuntu Workstation (either on a computer or on a VM) and set it up just the way you like it. Change your wallpaper, install your favorite applications, tweak your settings just the way you like them, and make yourself at home. Then, boot that computer or workstation from the Clonezilla boot media you've created so we can take an image of it. You'll see the main menu, which will look like the following screenshot. You'll see several options within the Clonezilla boot menu:



Clonezilla's main menu

www.it-ebooks.info

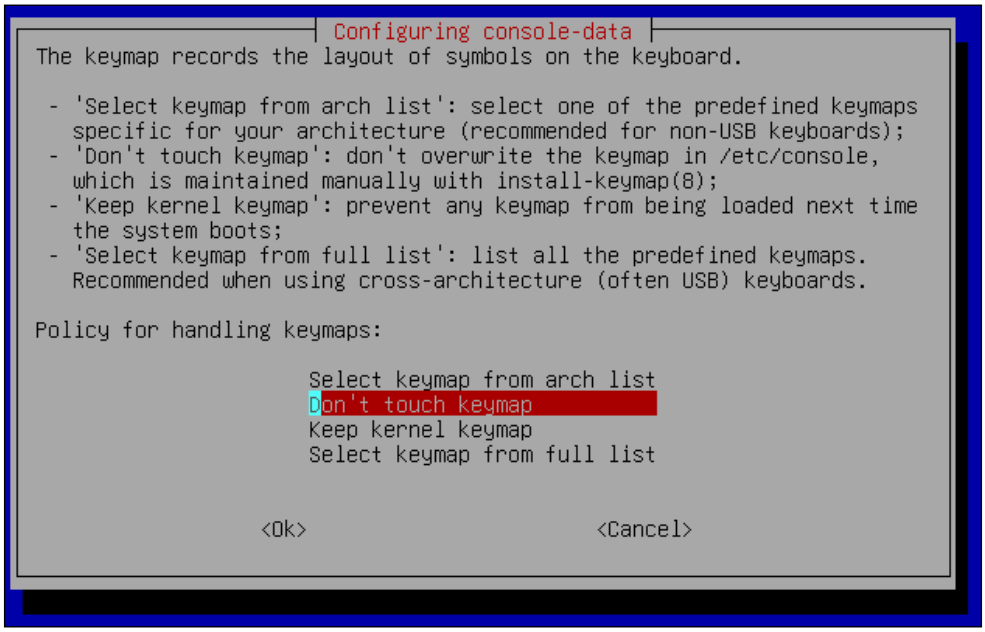
The first option, **Clonezilla live (Default settings, VGA 800x600)** is fine for most. However, there's another option you can consider, which is nested underneath the section **Other modes of Clonezilla live**. There, you'll find another option, **Clonezilla live (To RAM. Boot media can be removed later)**. That option allows you to boot Clonezilla live in a way that the entire utility will be loaded to RAM, allowing you to eject the disc or remove the flash drive after it finishes copying it to memory. This can help if you want to image multiple computers with only a single boot media. Regardless of which option you use, the next steps in this section will be the same. On the next screen that appears after Clonezilla finishes loading, you'll choose your language:



Clonezilla's language selection menu

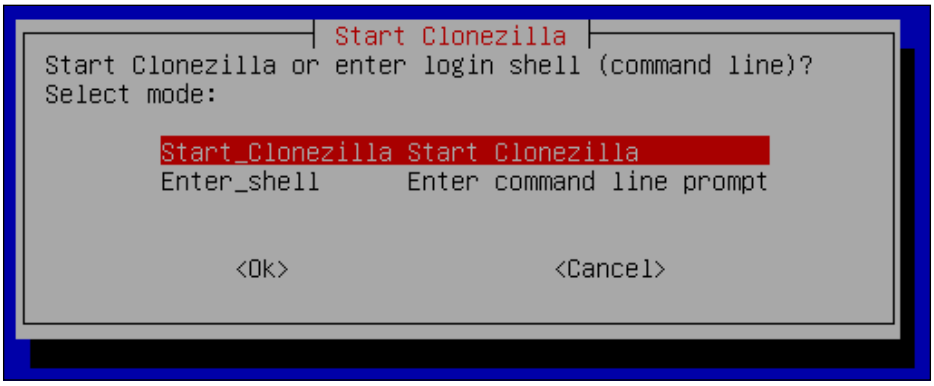
www.it-ebooks.info

At this screen, choose your language by using the up and down arrows keys, and press *Enter* to confirm your selection. Next, you'll have a chance to choose your **keymap**. You can select a different keymap from a list by selecting the relevant option. Otherwise, you can just choose the default option, **Don't touch keymap**. In most cases, that should suffice. You won't need to do much typing to use Clonezilla.



Clonezilla's keymap selection menu

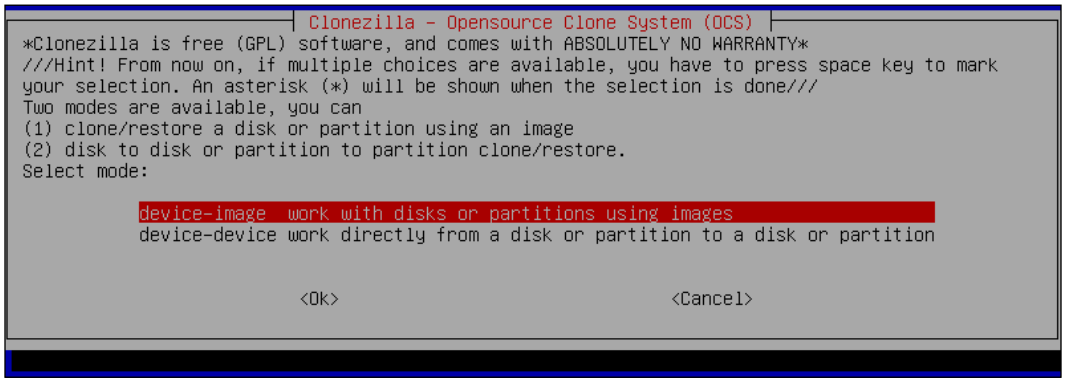
Next, you'll have an option to **Start_Clonezilla Start Clonezilla**. Press *Enter* to choose that option.



Clonezilla's start screen

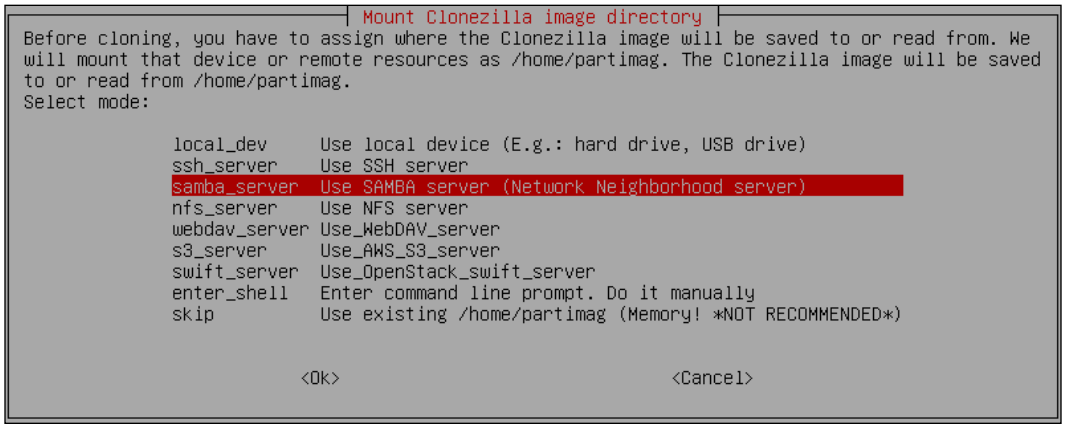
www.it-ebooks.info

On the next screen, you'll choose between whether or not you'll want to utilize disk images, or work directly from one disk to another. In addition to being able to save disk images, Clonezilla will allow you to clone one disk to another. For the sake of this section, we'll choose **device-image**, but keep in mind that Clonezilla allows you to clone directly to disks, should you ever need to utilize that option.



Clonezilla's OCS screen

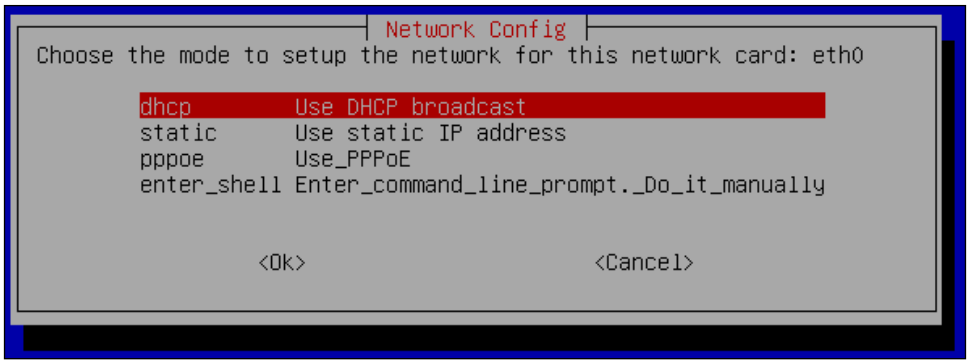
Next, Clonezilla will ask you where it can expect to save or write disk images. You can use the **samba_server** option here, to use a network file share. This is what I'll show you. But in addition to using Samba, Clonezilla is able to mount NFS shares as well. Also, you can use the **local_dev** option if you'd like to bypass networking altogether and use an external hard disk or a flash drive to act as storage for your images. The rest of the section will continue on as if we've chosen the **samba_server** option. However, keep note that Clonezilla features many flexible options for where it can store its data.



Clonezilla's image directory

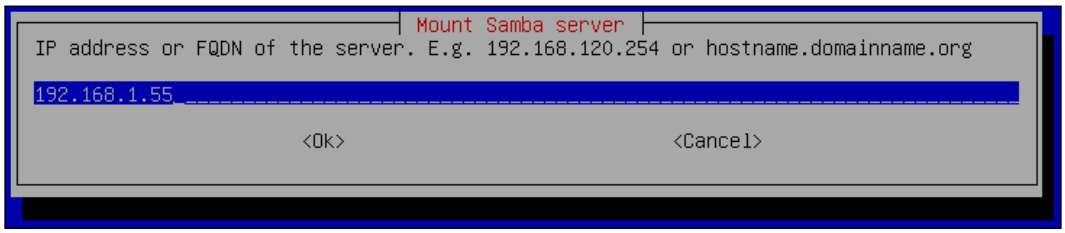
www.it-ebooks.info

On the next screen, we'll tell Clonezilla how to obtain an IP address. You can either choose **dhcp** or you can give it a static IP address if you wish. I'll continue on as if we've chosen **dhcp**, but if you'd like to use a static IP address, go ahead and choose that option and type in the values.



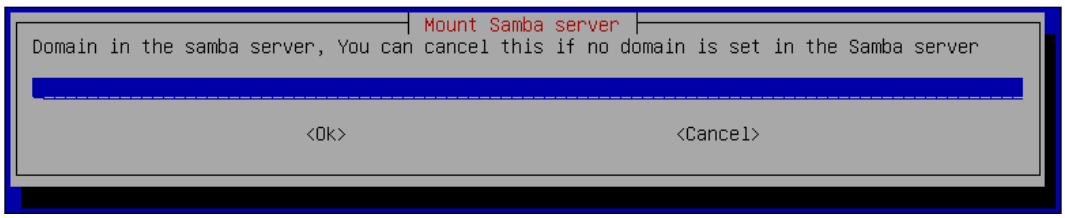
Clonezilla's Network Config options

Next, Clonezilla will ask you for the IP address or hostname for the server hosting your images share. Enter that IP address or hostname, and press *Enter*:



Entering an IP address for the network server for Clonezilla

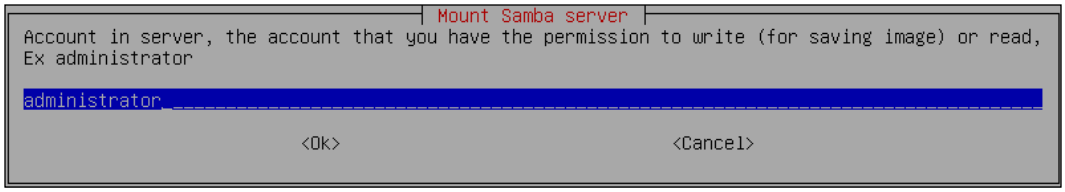
Clonezilla will then ask you to enter your domain. You can leave this blank and press *Enter* if you don't have one. If you're using a Windows share on a server that's a member of a Windows domain, feel free to enter that here:



Domain option during Clonezilla process

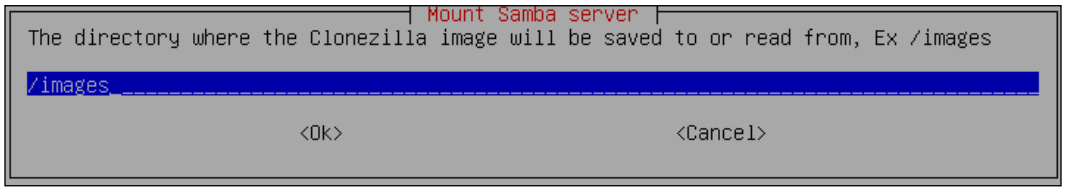
www.it-ebooks.info

Next, Clonezilla will ask you for the username that has permission to access your images share, defaulting to administrator. Change this to your username for that share, and press *Enter*.



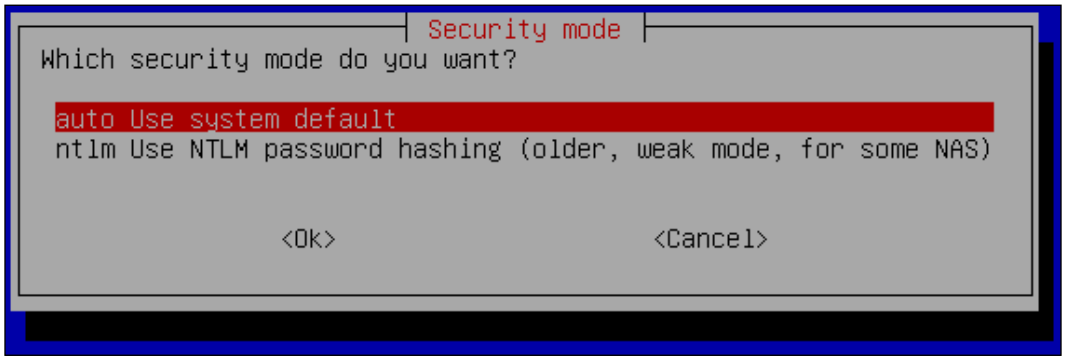
Entering a username for Clonezilla

Now Clonezilla will ask you for the name of your images share, defaulting to `/images`. The share name is relative to the share `root` of your server, so keep that in mind when entering it in. For example, if the network path to your share is `//myserver/shares/images`, you would enter `/shares/images`.



Choosing an image share

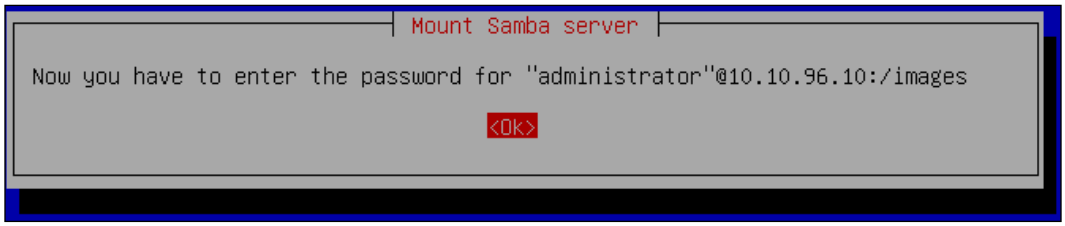
Next, you'll be asked for the **security mode**. Just press *Enter* for this, I don't foresee any modern situations in which you'll need to use **NT LAN Manager** (**NTLM**).



Security mode option during Clonezilla process

www.it-ebooks.info

The next prompt will tell you that you'll need to enter your password for the share. Don't enter it yet, instead just press *Enter*:



Notice of password entry

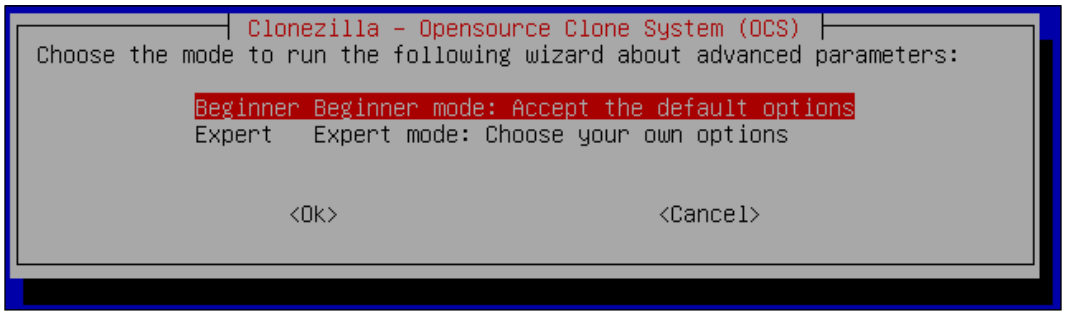
After you press *Enter*, you'll see on the bottom of the screen that you'll be prompted for the password for the user you're accessing the images share with. Go ahead and enter it:

```
Mounting samba server by:
mount -t cifs "//10.10.96.10/images" /home/partimag -o user="administrator"
Password for administrator@//10.10.96.10/images:
```

Entering the password for the Clonezilla share

A screen will now show, showing the current mounts on your machine with your images share listed. You'll see the text **Press Enter to continue**. Do that.

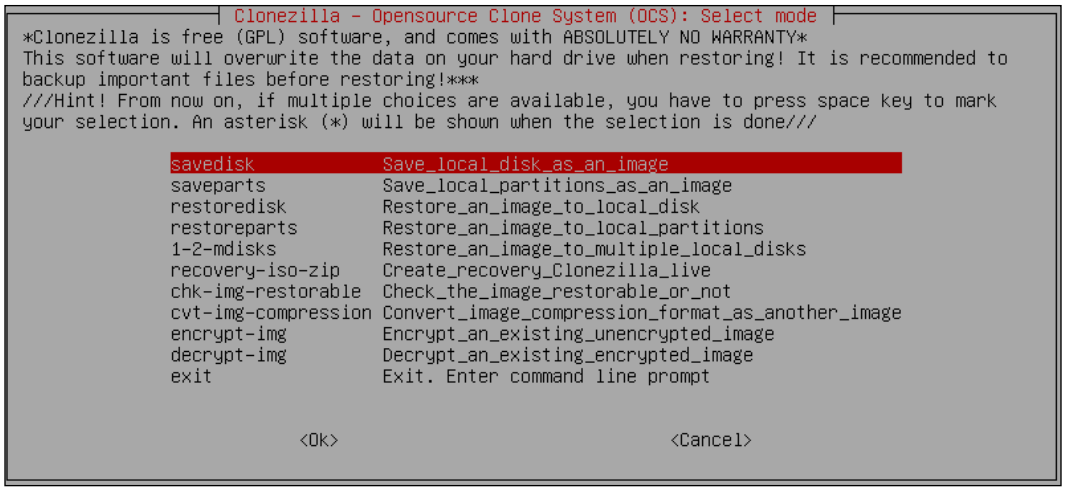
Now, you'll select either **Beginner mode** or **Expert mode** for Clonezilla. There are useful options underneath **Expert mode**, but for now, choose the default option for **Beginner**:



Clonezilla mode selection prompt

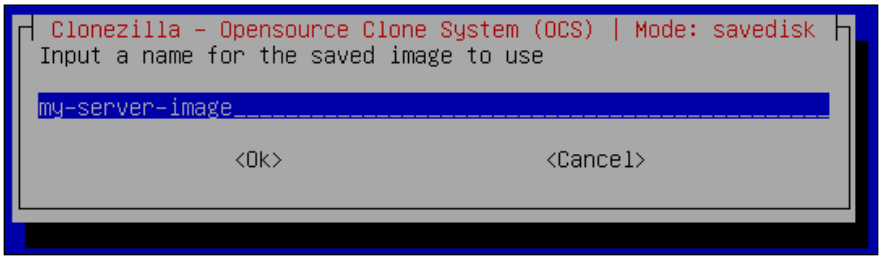
Now we get to the most important screen of the entire process. We've mounted a network share, and authenticated our access to it. Now, we tell Clonezilla what we want it to do. In our case, we want to choose the option **savedisk** so we can create a new image:

www.it-ebooks.info



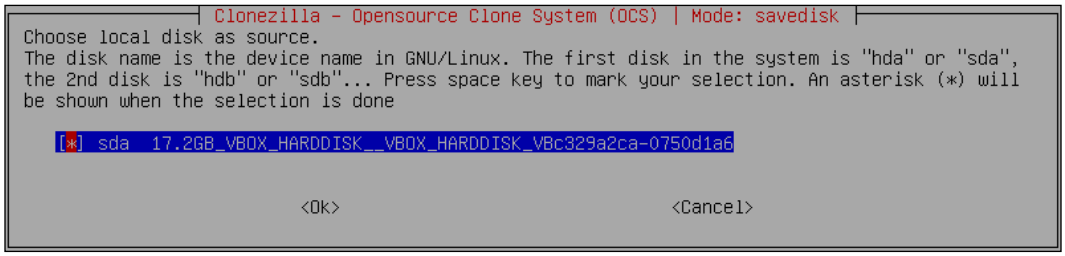
Clonezilla's OCS selection screen.

Next, Clonezilla will ask you for the name of your image. You can name it anything you want. Press *Enter* after you've given the image a name.



Naming the server image

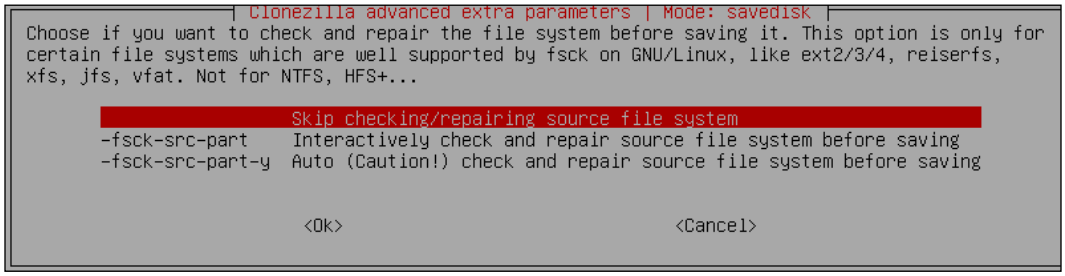
After you give the image a name, Clonezilla will ask you which disk you want to take an image of. If you only have one disk, it will automatically select it. If you have more than one, use your up and down arrow keys to change the selection, and press the *Spacebar* to tell Clonezilla to take an image of it. Press *Enter* when done.



Selecting savedisk

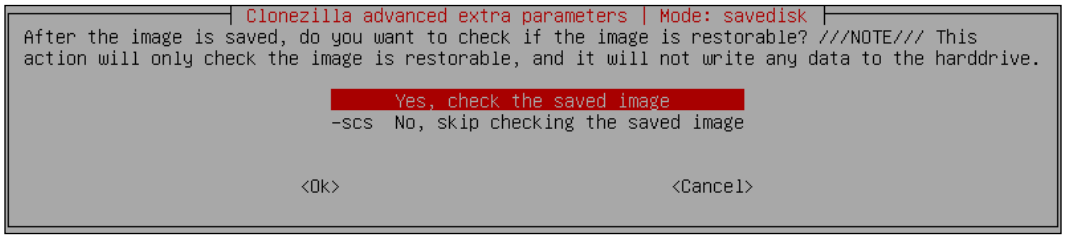
www.it-ebooks.info

On the next screen, you'll be asked if you want Clonezilla to run an `fsck` (filesystem check) on the disk you're taking an image from. If you have any reason to believe that the disk's integrity may be questionable, go ahead and choose the option **Interactively check and repair source file system before saving**. Otherwise, just press *Enter*:



Choosing whether or not to run fsck while saving an image

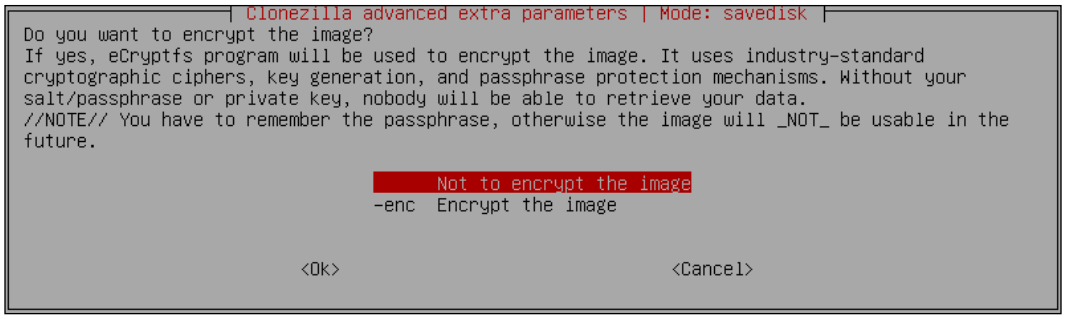
Next, Clonezilla will ask you whether or not you want it to perform a sanity check of the image after it's done saving. While it may add additional time to the process of saving the image, I recommend you choose **Yes, check the saved image**. With this option, if anything goes wrong when the image is saved and it doesn't pass a sanity check, Clonezilla will warn you so you can try to create it again.



Choosing whether or not to check the image after it's saved

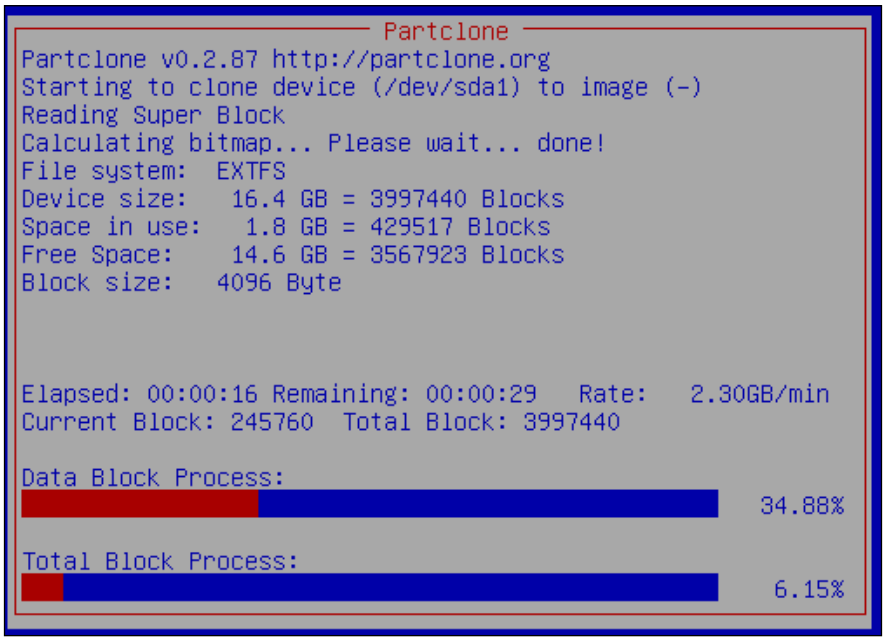
Clonezilla features the ability to encrypt your image if you need to keep private data on it. I'll leave this one up to you. In most cases, you'll encrypt data stored on a server, not the server image itself. I'll go on as if we've chosen the default, **Not to encrypt image**:

www.it-ebooks.info



Clonezilla features an option to encrypt your image for privacy

Almost there! Next, Clonezilla will ask you to **Press Enter to continue**, and then will ask if you're sure you'd like to proceed through the process. Press *Y* and then *Enter* to start the process of saving the image.



Clonezilla in the process of saving an image

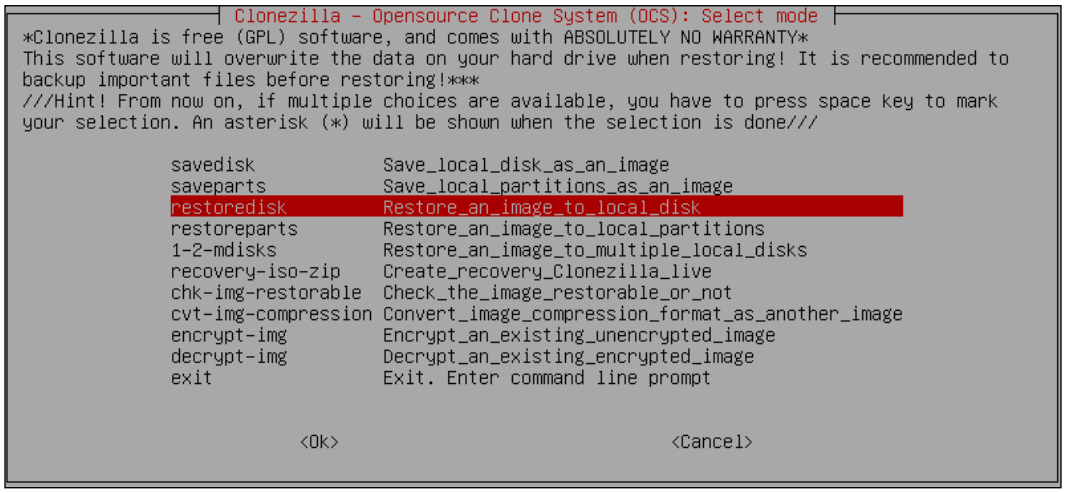
Now Clonezilla will save the image onto your file server. The process usually goes by fairly quickly, but it all depends on how much of your source disk has been used. If you chose the option to have Clonezilla check your image, you'll see another set of progress bars appear after the image is done saving. Clonezilla will let you know when it's done, and it will give you an option to shut down or reboot the reference computer.

www.it-ebooks.info

The process for restoring an image is quite similar to the process for saving it. There are a few things to keep in mind, however:

- You can restore an image onto a server with a disk just as large as the disk the image was taken from or larger. You cannot, for example, restore an image taken on a server with a 256 GB drive onto a server with a 128 GB drive. In that case, the server you're restoring the image to must have a 256 GB drive or larger.
- The image will only contain used space, so if you have an image taken from a server with a 128 GB drive, the image will be much smaller. Since there's also compression being used, the image will actually be smaller than the actual used space on a server.
- If your server is using LVM for the disk you're taking an image from, the image may be just as large as the entire disk, since Clonezilla is not able to see used space within LVM, it appears to Clonezilla as if the entire disk is being used.

To restore an image, boot the machine you want to restore to with the Clonezilla media, and follow the same process we've followed in this section up until the select mode screen appears. This is the screen that gives you an option to save your disk or restore it. Simply choose the **restoredisk** option:



Back to the Clonezilla select mode screen

Clonezilla will now show you a list of the images you've saved into your images share on your network. Simply select the image you want to restore, proceed through a few confirmation screens, and Clonezilla will restore your image.

www.it-ebooks.info

Although it may seem a bit out of place for a walk-through for using Clonezilla in a book about Ubuntu Server, this is one of those utilities that will simplify deploying Ubuntu Server, as well as workstation operating systems all across your network. You'll only need to configure a base server one time, and then from that point on, you can restore your image onto other servers and start off from an already established foundation. To make the most of this process, think of some tweaks you can implement on every server you set up. If you create an image with those tweaks already implemented, you'll have less work to do when you set up a new server. And while there are certainly quite a few screens you traverse through while using Clonezilla, the process is actually faster than installing Ubuntu from its ISO image.

Utilizing bootable recovery media

I've mentioned live media several times within this book. The concept of live media is a wonderful thing, as we can boot into a completely different working environment from the operating system installed on our device, and perform tasks without disrupting installed software on the host system. The desktop version of Ubuntu, for example, offers a complete computing environment we can use in order to not only test hardware and troubleshoot our systems, but also to browse the Web just as we would on an installed system. In terms of recovering from disasters, live media becomes a saving grace.

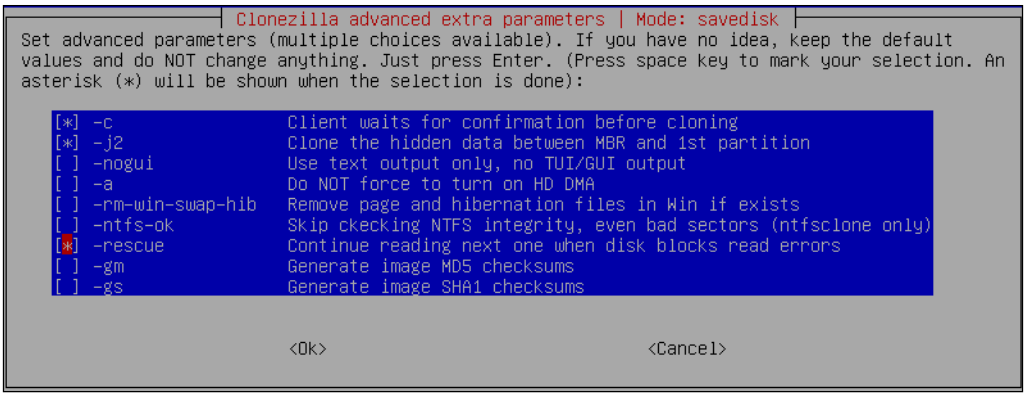
As administrators, we'll run into one problem after another. This gives us our job security. Computers often seemingly have a mind of their own, failing when least expected. Our servers and desktops can encounter a fault at any time, and live media allows us to separate hardware issues from software issues, by troubleshooting from a known and good working environment.

One of my favorites when it comes to live media is the desktop version of Ubuntu. Although geared primarily toward end users who wish to install Ubuntu on a laptop or desktop, as administrators we can use it to boot a machine that normally wouldn't, or even recover data from failed disks. For example, I've used Ubuntu live media to recover data from both failed Windows and Linux systems, by booting the machine with the live media and utilizing a network connection to move data from the bad machine to a network share. Often, when a computer or server fails to boot, the data on its disk is still accessible. Assuming the disk wasn't encrypted during installation, you should have no problem accessing data on a server or workstation using live media such as Ubuntu live media.

www.it-ebooks.info

Sometimes, certain levels of failure require us to use different tools. While Ubuntu's live media is great, it doesn't work for absolutely everything. One situation is a failing disk. Often, you'll be able to recover data using Ubuntu's live media from a failing disk, but if it's too far gone, then the Ubuntu media will have difficulty accessing data from it as well. Another useful tool when it comes to failing disks is (believe it or not) Clonezilla, which we have just gone over in the previous section. Clonezilla's primary focus is disk imaging, but we can use it for recovery as well. It doesn't always work, but it's worth a shot. If you recall, during the process of using Clonezilla, you were asked whether you'd like to use Beginner or Expert mode, which I had you use **Beginner mode**. In **Expert mode**, we have an additional option we can leverage when saving an image, known as **rescue**.

When you select **Expert mode** during the process of saving an image with Clonezilla, you'll see an additional screen that's not normally a part of the process, the extra parameters screen. It looks like the following:



Back to the Clonezilla select mode screen

The **-rescue** option is listed among the other extra parameters on this screen. To activate it, move the selection to it with the down arrow key, and press the *Spacebar* to check the box next to it, which activates the feature. The way this option works is if Clonezilla encounters a bad sector, instead of failing to create the image, it will skip that bad sector and move on to the next. Basically, it will still create an image of your drive, even if some of the sectors are unreadable. Then, you can restore this image onto a known good drive, and then attempt a file system check with your operating system. In some cases, the image being restored onto a good disk will allow the operating system to repair the file system damage caused by the bad sectors, and then be completely fixed. In other cases, the disk is either too far gone, or the bad sectors are in areas that contain important data that the resulting image will be missing. This trick is certainly worth a try, though. I've actually saved an unbootable server using this method, and that was great considering the company who owned the server never backed it up a single time!

www.it-ebooks.info

In addition, you can actually restore a system's ability to boot, using a third-party utility known as **Boot Repair**. This utility will allow you to restore GRUB on a system which has a damaged boot sector and is no longer able to boot. GRUB is a crucial service, since it's the bootloader that's responsible for a system's ability to boot a Linux distribution. Should GRUB ever stop functioning properly, the Boot Repair utility is a good way to fix it. Having to reinstall or repair GRUB is actually quite common on systems that dual boot some flavor of Linux with Windows, as Windows will often overwrite the boot-loader (thus wiping out your option to boot Linux) when certain updates are installed. Windows always assumes it's the only operating system in use, so it has no problem wiping out other operating systems that share the same machine.

The Boot Repair utility is not recommended if the underlying system is using **EFI System Partition** (**ESP**).

To use this utility, first boot the affected machine from Ubuntu live media (the desktop version) and connect to the Internet. Then, you can install the Boot Repair utility with the following commands:

```
# add-apt-repository ppa:yannubuntu/boot-repair
# apt-get update && apt-get install boot-repair
```

Now, the **Boot Repair** utility is installed within the live system. You can access it from the Ubuntu dash, or you can simply enter the boot-repair command from the terminal to open it:

The main screen of the Boot Repair utility

www.it-ebooks.info

For most situations, you can simply choose the default option, **Recommended repair**. There are advanced options as well if you'd like to do something different; you can view these options by expanding the **Advanced options** menu. This will allow you to change where GRUB is reinstalled, in a case where you're restoring GRUB onto a different drive than the main drive. It also allows you to add additional options to GRUB in a case where you're having issues with certain hardware configurations, and even allows you to upgrade GRUB to its most recent version. Using **Boot Repair**, you should be able to resurrect an unbootable system, in a case where the underlying GRUB installation has become damaged.

As you can see, live media can totally save the day. Clonezilla and Ubuntu live image for its desktop version are great boot-disks to have available to you. Ubuntu live image alone is something I've made use of constantly, mainly for recovering data from failing machines, or virus-ridden Windows machines. One of the best aspects of using Ubuntu live image is that you won't have to deal with the underlying operating system and software set at all, you can bypass both by booting into a known working desktop, and then copy any important files from the drive right onto a network share. Another important feature of Ubuntu live media is the memory test option. Quite often, strange failures on a computer can be traced to defective memory. Other than simply letting you install Ubuntu, the live media is a Swiss-army knife of many tools you can use to recover a system from disaster. If nothing else, you can use live media to pinpoint whether a problem is software or hardware related. If a problem can be reproduced within the installed environment but not within a live session, chances are a configuration problem is to blame. If a system also misbehaves within a live environment, it may help you identify a hardware issue. Either way, every good administrator should have live media available to troubleshoot systems and recover data when the need arises.

www.it-ebooks.info

Summary

In this chapter, we looked at several ways in which we can prevent and recover from disasters. Having a sound prevention and recovery plan in place is an important key to managing servers efficiently. We need to ensure we have backups of our most important data ready for whenever servers fail, and we should also keep backups of our most important configurations. Ideally, we'll always have a warm site set up with pre-configured servers ready to go in a situation where our primary servers fail, but one of the benefits of open-source software is that we have a plethora of tools available to us that we can use to create a sound recovery plan. In this chapter, we looked at leveraging rsync as a useful utility for creating differential backups, and we also looked into setting up a Git server that we can use for configuration management, which is also a crucial aspect of any sound prevention plan. We also took a look at Clonezilla, which we can use to create deployment images to allow us to set up new servers quickly. We are also able to use Clonezilla to attempt data recovery, should we run into an issue with one of our disks. We also talked about the importance of live media in diagnosing issues.

And with this chapter, this book comes to a close. I'd like to thank each and every one of you, my readers, for taking the time to read this book. Writing this book has been an extremely joyful experience. I'd also like to thank *Packt Publishing* for giving me the opportunity to write a book about one of my favorite technologies. Writing this book was definitely an honor. When I first started with Linux in 2002, I never thought I'd actually be an author, teaching the next generation of Linux administrators the tricks of the trade. I wish each of you the best of luck, and I hope this book has been beneficial to you and your career.

www.it-ebooks.info

www.it-ebooks.info

Index

Symbols

~/.ssh/config file
about 133
SSH connections, simplifying with 133

A

Acronis 386
additional storage volumes
adding 80-84
administrator access
configuring, with sudo 63-66
AMD-V 298
Apache
configuring 269-275
installing 269-275
securing, with SSL 278-284
Apache modules
installing 276-278
application logs 354
apt-get dist-upgrade command 329
apt-get upgrade command 329
aptitude command
about 157-159
using 157-159
attack surface
lowering 326-329
authorization log 356

B

backup plan
implementing 383-386
Berkeley Internet Name Daemon (BIND) package 204
bind
name resolution (DNS), setting up with 204-211
bootable recovery media
utilizing 399-402
bootable Ubuntu Server flash drive
creating, in Linux 10-12
creating, in Mac 13-15
creating, in Windows 7-9
Boot Repair 401
Broadcast Address 197
B-tree file system (Btrfs) 83

C

Caching Name Server 205
Canonical Name (CNAME) 210
Certificate Signing Request (CSR) 282
Clonezilla
about 386
reference 387
Clonezilla live
system images, creating with 386-399
configuration management
Git, utilizing for 376-383
containers, Docker
creating 315-322
managing 315-322
running 315-322
Cron
tasks, scheduling with 188-190

www.it-ebooks.info

D

databases
managing 255-261
database server
setting up, requisites 246, 247
DenyHosts 334
DHCP Reservation 119
Digital Ocean
about 95
URL 2
Digital Ocean VPS 76
directory permissions
setting 68-74
disasters
preventing 374, 375
Disk Cache 185
disks
decrypting, with Linux Unified Key Setup (LUKS) 344, 345
encrypting, with Linux Unified Key Setup (LUKS) 344, 345
disk usage
viewing 76-80
Docker 315
Document Root 270
Dynamic DNS 108
Dynamic Host Control Protocol (DHCP) 196

E

EFI System Partition (ESP) 401
encryption 3
encryption at rest 3

F

Fail2ban
about 334
configuring 334-337
installing 334-337
file permissions
setting 67-74
files
sharing, with Windows users 223-228
transferring, with rsync 233-238
transferring, with SCP 238-240
file server
considerations 221-223
Filesystem Hierarchy Standard (FHS) 88
firewall
setting up 341-343
Fully Qualified Hostname (FQDN) 219

G

Git
about 376
utilizing, for configuration management 376-383
GNU Privacy Guard (GnuPG) 146
Group ID (GID) 46
groups
managing 56-58

H

hard links
about 106
using 107-109
high availability
setting up, with keepalived 285-290
htop
about 176-179
using 176-179
Hub 315

I

inodes 77
Internet gateway
setting up 215-217
IP addresses
serving, with isc-dhcp-server 199-204
IP address scheme
planning 195-198
isc-dhcp-server
IP addresses, serving with 199-204

J

jobs
handling 171-173

www.it-ebooks.info

K

keepalived
high availability, setting up with 285-290
Kernel-based Virtual Machine (KVM) 298
keymap 390
killall command 174
kill command 174

L

Linode
URL 2
Linux name resolution 122, 123
Linux networks
about 111
hostname, setting 112, 113
network interfaces, managing 114-118
setting up 111
static IP addresses, assigning 118-121
Linux package management 136, 137
Linux RAM
URL 184
Linux Signals 174
Linux Unified Key Setup (LUKS)
about 25, 344
disks, decrypting 344, 345
disks, encrypting 344, 345
load average 190-192
Logical Volumes 98
Long-Term Support (LTS) 5, 137
LVM volumes
utilizing 97-106

M

MariaDB
about 338
best practices 338-341
installing 248-251
in Ubuntu 16.04, differences 254, 255
MariaDB configuration 251-253
memory usage
about 184-187
monitoring 184-187
MySQL 338

N

name resolution, DNS
setting up, with bind 204-211
Ncurses Disk Usage Utility 79
netstat command 327
Network Manager 124
Network Time Protocol (NTP) 165, 195
NFS 221
NFS Client 222
NFS shares
setting up 229-232
NT LAN Manager (NTLM) 393

O

OpenSSH
about 125, 126, 325, 327
securing 330-334
working with 125-129
ownCloud
configuring 290-294
installing 290-294

P

package maintainer 136
packages
about 142, 143
backing up 156
restoring 156
searching 142, 143
password policies
managing 59-63
passwords
managing 59-63
Personal Package Archive (PPA) 146
Physical Address Extension (PAE) 7
Physical Volumes 98
Pluggable Authentication Module (PAM) 62
PPA
URL 147
Preboot eXecution Environment (PXE) boot 386

www.it-ebooks.info

Principle of Least Privilege 330
processes
killing 174, 175
process ID (PID) 166
PS1 prompt 112
ps command
about 166
running processes, displaying with 166-170
Public Key Authentication 129

R

Raspberry Pi
Ubuntu Server, installing on 37-39
remote file systems
mounting, with SSHFS 241-243
rescue 400
role 3
root-cause analysis 352
rsync
about 384
files, transferring with 233-238
running processes
displaying, with ps command 166-170

S

Samba
about 221
files, sharing with Windows users 223-228
SCP (Secure Copy)
files, transferring with 238-240
secondary DNS server
creating 211-215
Security Checklist 330
server
updating 149-155
Server Message Block (SMB) 222
Services for UNIX 222
slave DB server
setting up 262-267
Snap packages
about 160-162
installing 160-162
software
installing 138-141
uninstalling 138-141
software repositories
about 144-148
managing 144-148
SSH Agent 132
SSH connections
simplifying, with ~/.ssh/config file 133
SSH File System (SSHFS)
remote file systems, mounting with 241-243
SSH key management
about 129-132
working with 129
SSL
Apache, securing with 278-284
Start of Authority (SOA) 208
Static IP assignment 119
Static Lease 119
storage volumes
/etc/fstab file 89-93
formatting 84-87
hard links, using 106-109
mounting 87-89
partitioning 84-87
swap, managing 94-97
symbolic links, using 106-109
unmounting 87-89
subnets 196
Subsystem for UNIX-based Applications 222
subtree checking 230
sudo
administrator access, configuring with 64-66
locking down 346, 347
sudo command 374
swappiness 187
symbolic links
about 106
using 106-109
system clock
keeping, in sync with NTP 217-219
system images
creating, with Clonezilla live 386-399
system log 357
system logs 354
system processes
about 179-183
handling 179-183

www.it-ebooks.info

T

tasks
scheduling, with Cron 188-190
Teletypewriter 168
temporary 88
Time to Live (TTL) 208
troubleshooting, Ubuntu Servers
about 349
defective RAM, diagnosing 368-371
network issues, tracing 359-364
problem space, evaluating 350, 351
resource issues, troubleshooting 364-368
root-cause analysis, conducting 352-354
system logs, viewing 354-358

U

Ubuntu Packages
URL 143
Ubuntu Server
32-bit installation 6
64-bit installation 7
about 1
environment, setting up 2
installing 18-37
installing, on Raspberry Pi 37, 38
obtaining 4-6
partition layout, planning 16, 17
role, determining 3
troubleshooting 349
URL 4
user management 41
Uncomplicated Firewall (UFW) 342
units 179
Universally Unique Identifier (UUID) 90
Universal Naming Convention (UNC) 228
Universal USB installer
reference 6
User ID (UID) 46
user management
/etc/passwd 49-53
/etc/shadow 49-53
about 41
administrator access, configuring with sudo 63-67
default configuration files, distributing, with /etc/skel 53, 54
directory permissions, setting 67-74
file permissions, setting 67-74
groups, managing 56-58
password policies, managing 59-63
passwords, managing 59-63
root, using 42, 43
users, creating 43-48
users, removing 43-48
users, switching 54, 55
users
switching between 54, 55

V

VirtualBox
URL 2
Virtual IP (VIP) 285
virtualization support 298
virtual machine network
bridging 311-314
virtual machines
creating 306-311
virtual machine server
setting up 298-306
Virtual Private Server (VPS) 2, 275
Voice over IP (VoIP) 196
Volume Groups 98
VT-x 298

W

Windows users
files, sharing with 223-228

Z

Zone File 204

www.it-ebooks.info