Community Experience Distilled

GitHub Essentials

Unleash the power of collaborative workflow development using GitHub, one step at a time

Achilleas Pipinellis

[PACKT] PUBLISHING

www.it-ebooks.info

GitHub Essentials

Unleash the power of collaborative workflow development using GitHub, one step at a time

Achilleas Pipinellis

[PACKT] PUBLISHING

BIRMINGHAM - MUMBAI

www.it-ebooks.info

GitHub Essentials

Copyright © 2015 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: September 2015

Production reference: 1280915

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78355-371-6

www.packtpub.com

www.it-ebooks.info

Credits

Author
Achilleas Pipinellis

Reviewer
Umesh Ram Sharma

Commissioning Editor
Dipika Gaonkar

Acquisition Editor
Nikhil Karkal

Content Development Editor
Sumeet Sawant

Technical Editor
Saurabh Malhotra

Copy Editor
Trishya Hajare

Project Coordinator
Shweta H Birwatkar

Proofreader
Safis Editng

Indexer
Hemangini Bari

Production Coordinator
Nitesh Thakur

Cover Work
Nitesh Thakur

www.it-ebooks.info

About the Author

Achilleas Pipinellis is an open source enthusiast and tries to get involved in as many projects as possible. He was introduced to Linux almost 10 years ago and hasn't looked back ever since. His distribution of choice is Arch Linux, a lightweight and flexible system that adheres to the KISS philosophy.

He is currently working as a system administrator and likes to try new technologies, especially those that require some special deployment. He also enjoys writing technical guides and articles that help people learn about new technologies. He strongly believes that comprehensive documentation is essential to a project's growth and recognition.

In his free time he practices Aikido and enjoys going to conferences that promote the open source movement.

www.it-ebooks.info

About the Reviewer

Umesh Ram Sharma has more than 6 years of experience in the architecture, design, and development of scalable and distributed cloud-based applications. He has a master's degree in computer science and information technology and is also an expert in the practical and technical implementation of various offerings of J2EE stack, Hibernate, and Spring Stack.

He is currently working as a senior software engineer, with a growing interest in the DevOps area. He handles product infrastructure on AWS cloud and develops expertise around automated deployments. He has demonstrated great value by implementing deployment, configuration management with puppet and various technologies, such as AWS Cloud, J2EE, MySql, MongoDB, memcache, Apache Tomcat, and Hazelcast.

www.it-ebooks.info

www.PacktPub.com

Support files, eBooks, discount offers, and more

For support files and downloads related to your book, please visit `www.PacktPub.com`.

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 `service@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

Free access for Packt account holders

If you have an account with Packt at `www.PacktPub.com`, you can use this to access PacktLib today and view 9 entirely free books. Simply use your login credentials for immediate access.

www.it-ebooks.info

Table of Contents

Preface **vii**

Chapter 1: Brief Repository Overview and Usage of the Issue Tracker **1**

Exploring the repository's main page **1**
Creating a new repository 2
The commits page and a comparison with the git log command 4
The branches page and a comparison with the git branch command 7
The Raw, Blame, and History buttons 9
The Watch, Star, and Fork buttons 11
Changing the description and URL 12
Learning how to use the powerful benefits of the issue tracker **13**
Creating a new issue 14
Assigning issues to users 18
Labels 20
Why labels are a great asset to UX 20
Creating new label names and setting different colors 20
Using labels to group issues 24
Milestones 24
Why milestones are a great help when working with code versioning 24
Creating a new milestone 25
Adding issues to milestones 26
Using milestones to see which issues are resolved or are yet to be resolved 27
Tips and tricks 27
Learning about the README file 27
Navigating easily with keyboard shortcuts 28
Summary **28**

www.it-ebooks.info

Chapter 2: Using the Wiki and Managing Code Versioning 29
Using the wiki 29
Why wikis are a nice place to document your project 30
Create a new wiki page 30
Deleting a page 32
A Markdown-powered wiki – an introduction to Markdown 33
How to add a sidebar and a footer to your wiki 35
Watching a wiki page's commit history and reverting to a previous state if needed 38
Managing code versioning 41
Creating a release 41
Editing a release 45
Pushing a tag from the command line 46
Marking as prerelease 46
Making a draft of a release 48
Uploading your own files 48
Tips and tricks 49
Subscribing to new releases via atom feed 50
Editing the wiki locally 50
Installing gollum 50
Cloning the wiki and see the preview in your browser 50
Making changes locally and pushing to GitHub 51
Summary 52
Chapter 3: Managing Organizations and Teams 53
The difference between users and organizations 53
Organization roles and repository permission levels 54
Creating an organization 55
Global member privileges 58
Repositories 60
Teams – a great way to grant selective access to your organization projects 62
Creating a team 62
Inviting people 65
Accepting an invitation 68
Team members permissions 69
Request to join a team 71
Step 1 – as a user 71
Step 2 – as a user 71
Step 3 – as an owner or team maintainer 72
Adding repositories to a team 72

www.it-ebooks.info

The People tab **74**
Managing access levels 76
Difference between Members and Outside collaborators 79
Demoting to an outside collaborator 81
Invite members 82
Organization settings **83**
Profile 84
Team privacy 85
The third-party access 86
Audit log 86
Tips and tricks 88
How to transfer a project to an organization's namespace 88
How to convert a user account into an organization 91
Mention teams 94
Organization feed only in dashboard 95
Summary **96**
Chapter 4: Collaboration Using the GitHub Workflow **97**
Learn about pull requests **97**
Why pull requests are a powerful asset to work with 97
The connection between branches and pull requests 98
Create branches directly in a project – the shared repository model 98
Create branches in your fork – the fork and pull model 98
How to create and submit a pull request 99
Use the Compare & pull request button 99
Use the compare function directly 102
Use the GitHub web editor 103
Submit a pull request 107
Peer review and inline comments **108**
The layout of a pull request 109
Inline comments 113
Pull requests overview 116
Correct mistakes and re-push to branch 117
Merge the pull request **118**
Remove/restore a branch after the pull request is merged 118
Revert a pull request 119
Tips and tricks **119**
Close issues via commit messages 119
Task lists in pull requests 121
Downloading the diff of pull requests 123
A global list of your open pull requests 123

www.it-ebooks.info

Adding a LICENSE file using the web editor 123
Creating new directories using the web editor 124
Summary 124
Chapter 5: GitHub Pages and Web Analytics 125
GitHub Pages 125
User, organization, and project pages 126
Creating a user or an organization page 127
Creating a project page manually 127
Creating a project page with GitHub page generator 128
Updating a project page with GitHub page generator 132
Using a custom domain 132
How to customize your page using Jekyll 133
Installing Jekyll 133
Introduction to Jekyll 135
Read more about Jekyll 138
Web analytics 139
Graphs 139
Contributors – additions/deletions 140
See a repository's traffic – visitors, clones, and popular content 142
Commits over time 143
Frequency of updates 144
Network 144
Members 146
Pulse 146
Tips and tricks 147
Making use of pages metadata with Jekyll 148
Summary 150
Chapter 6: Exploring the User and Repository Settings 151
User settings 151
Profile 152
Setting up multiple e-mails 153
Managing your SSH keys 154
Setting up two-factor authentication 156
Repository settings 157
Changing the default branch that appears in repository's main page 158
Enabling/disabling the wiki 159
Enabling/disabling the issue tracker 159

www.it-ebooks.info

Adding collaborators 160
Transferring ownership – user to organization 161
Deleting a repository 162
Tips and tricks 162
Finding the size of your repositories 162
Fine-tuning e-mail notifications 163
Summary 163
Index 165

www.it-ebooks.info

www.it-ebooks.info

Preface

GitHub is the leading code-hosting platform with literally millions of open source projects having their code hosted on it. In conjunction with Git, it provides the means for a productive development workflow and is the preferred tool among developers. Starting with the basics of creating a repository, you will then learn how to manage the issue tracker, where your project can be discussed. Continuing our journey, we will explore how to use the wiki and write rich documentation that will accompany your project. Organization and team management will be the next stop and then onto the pull requests feature that made GitHub so well known. Next, we will focus on creating simple web pages hosted on GitHub and lastly we explore the settings that are configurable for a user and a repository.

What this book covers

Chapter 1, *Brief Repository Overview and Usage of the Issue Tracker*, explains some of the main features GitHub provides and what you can make out of them. The issue tracker is the heart of communication between a project's developers and/or users. Consider it as a notepad dedicated to each repository where you track bugs, reports, feature requests, and anything else that can be written down. GitHub has implemented many other features that sit on top of the issue tracker, such as labels and milestones, which provide the ability to better visualize and categorize all the issues.

Chapter 2, *Using the Wiki and Managing Code Versioning*, helps you learn how to create, edit, and maintain a wiki by providing a home for your documentation that will complement your project. You will also learn how to create a new release out of an existing branch or tag, accompanied with optional release notes. In this way, the end user can understand the changes from any previous versions.

www.it-ebooks.info

Chapter 3, *Managing Organizations and Teams*, helps you learn how to create and manage the organizations that you are the owner of. You will also learn how to create teams, add users to them, and assign different access levels according to your needs.

Chapter 4, *Collaboration Using the GitHub Workflow*, helps you learn how to work with branches and pull requests, the most powerful features of GitHub.

Chapter 5, *GitHub Pages and Web Analytics*, helps you learn how to build web pages around your project, hosted exclusively on GitHub. You have the ability to make static web pages using HTML, CSS, and JavaScript.

Chapter 6, *Exploring the User and Repository Settings*, explores the most common and essential settings of a user and a repository. As a user, there is a lot of information you can set up in your user settings page, such as associating more than one e-mail to your account, adding multiple SSH keys, or setting up two-factor authentication. Similarly, some functionalities of a repository can be set up via its settings page. For example, you can enable or disable the wiki pages and grant write access to the public or completely disable the issue tracker.

What you need for this book

For this book, you'll just need Git; any version will do.

Who this book is for

This book is intended for experienced or novice developers with a basic knowledge of Git. If you ever wanted to learn how big projects such as Twitter, Google, or even GitHub, collaborate on code, then this book is for you.

Conventions

In this book, you will find a number of styles of text 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: "Let's first create a `README` file and push it to GitHub in order to explore the commits page."

www.it-ebooks.info

A block of code is set as follows:

```
echo "# github-essentials" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin git@github.com:<username>/<repository>.git
git push -u origin master
```

Any command-line input or output is written as follows:

```
sudo gem install bundler
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "The **Network** graph shows the branch history of the main repository as well as its forks."

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Reader feedback

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

To send us general feedback, simply send an e-mail to `feedback@packtpub.com`, and mention the book title via 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 on `www.packtpub.com/authors`.

www.it-ebooks.info

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 example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

Downloading the color images of this book

We also provide you 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: `http://www.packtpub.com/sites/default/files/downloads/3716OT_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 would 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 on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

www.it-ebooks.info

Piracy

Piracy of copyright 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

You can contact us at `questions@packtpub.com` if you are having a problem with any aspect of the book, and we will do our best to address it.

www.it-ebooks.info

www.it-ebooks.info

1
Brief Repository Overview and Usage of the Issue Tracker

The landing page of a project on GitHub depicts the contents of a person's local Git repository. Apart from the tree-like structure of the files, GitHub also provides some additional features that bring the most well-known and frequently used Git commands to your browser. Among others, these include the branches, commits, and tags of your repository.

In addition to the features mentioned above, GitHub also provides an issue tracker for each repository. This is where the discussions take place, bugs are tracked and reported, features are requested, and pretty much anything else that is relevant to the project is discussed.

GitHub has also implemented many other features that sit on top of the issue tracker, such as labels and milestones that provide the ability for better visualization and categorization of all the issues. We will explore all the features extensively, so don't worry if you aren't familiar with these terms yet.

Project and repository, although not the same thing, will be considered to have equal meaning and will be used interchangeably throughout this book.

Exploring the repository's main page

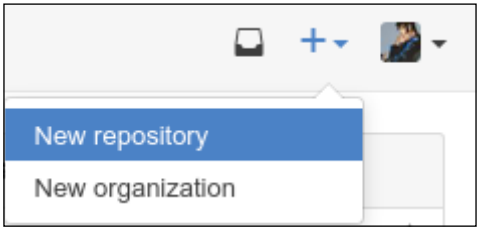
The main page of a repository is the place where people spend most of their time when visiting a project. In this section, you will learn how to create a repository and then we will explore the vast features of GitHub that bring Git's command line to your browser.

www.it-ebooks.info

Creating a new repository

Assuming you have already signed up in GitHub through `https://github.com/join`, we will see how to create a new repository that will host your code and explore the main repository's page.

Navigate to the top-right, click on the little cross beside your username, and choose **New repository**, as shown in the following screenshot:



Fill in a name under **Repository name**, which will ultimately form the URL under which your repository will be. This is the minimal action you need to perform in order to create a repository.

All the repositories on GitHub have the following URL scheme: `https://github.com/<username>/<repository_name>`

You can also provide a description so that people can tell with a glance what this is all about. Next option is whether your repository will be public or private. Generally, you go with public, unless you do not want your files to be seen by everybody. The private repos come with a price, though.

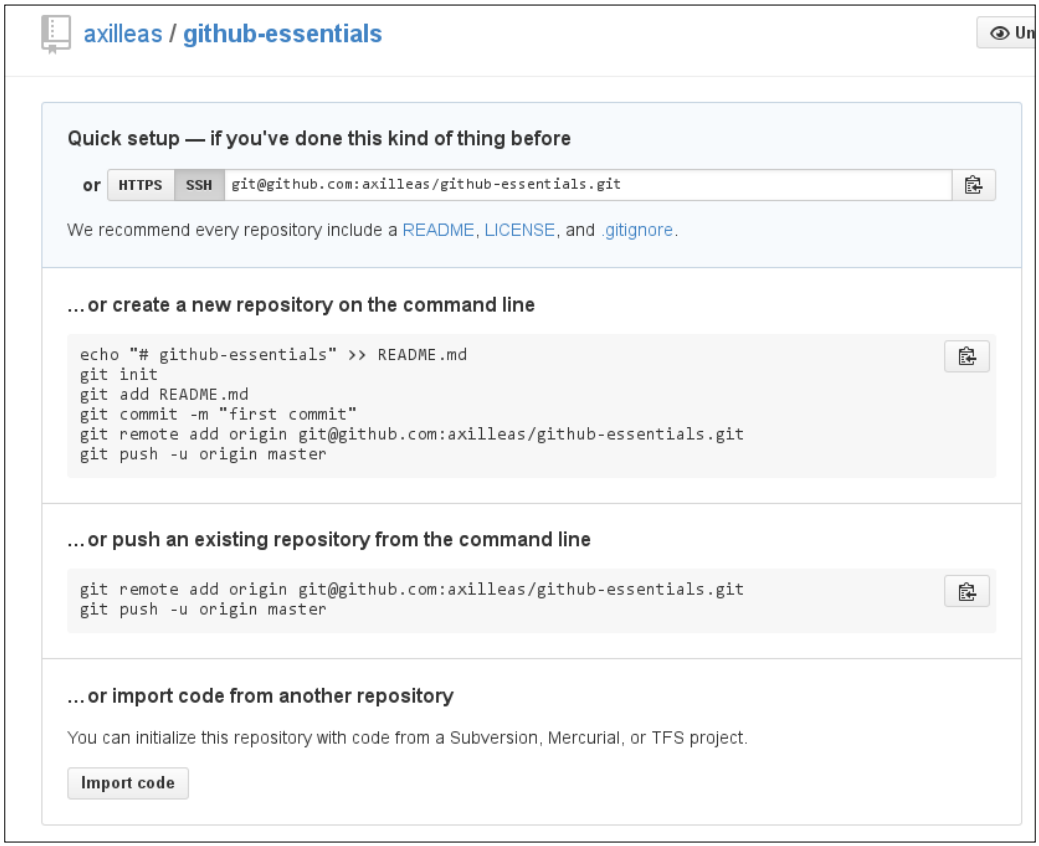
The very next thing GitHub provides is the ability to create the repository with a `README` file. Readme files usually include comprehensive information about the project you are hosting under your repository, such as installation guides, build and usage instructions, as well as guidelines on how one can contribute. You can always add a `README` file later, so leave this option unchecked for the time being.

Another nice feature is the ability to choose and include a `gitignore` file upon creation. You can choose from a collection of the useful `.gitignore` templates taken from `https://github.com/github/gitignore`.

Ultimately, the code that you will host on GitHub will be able to be forked and reused by third parties. If you are freshly starting a new repository, you can choose a license to include upon creation. Again, this is optional and you can always manually add a license file later.

www.it-ebooks.info

Let's hit the **Create repository** button and finish the repository creation. Here's what it looks like so far:



You can see that GitHub provides useful information on what to do next. If you already have an existing Git repository locally on your computer, you can push its code to GitHub or start fresh by following the instructions.

Since we will be working from the command line later, it is highly recommended to generate an SSH key to use with your GitHub account. Follow the guide at `https://help.github.com/articles/generating-ssh-keys/`. Also, make sure to properly configure your Git username and e-mail settings. For more information, see `https://help.github.com/articles/setting-your-username-in-git/` and `https://help.github.com/articles/setting-your-email-in-git/`.

www.it-ebooks.info

Congratulations on creating your first repository!

The next goal is to explore the repository's main page. This is the page you see when you navigate to `https://github.com/<username>/<repository>`, where you see the following:

- **<username>**: This is the username you registered with (found at the top-right corner)
- **<repository>**: This is the **Repository name** you filled in the previous steps.

The commits page and a comparison with the git log command

GitHub has a nice web UI that many common `git` commands can be presented to.

Let's first create a `README` file and push it to GitHub in order to explore the commits page:

1. Create the directory that will hold your code and `cd` into it:

```
mkdir -p ~/github-essentials
cd $_
```

2. Then, follow GitHub's instructions on a new project creation:

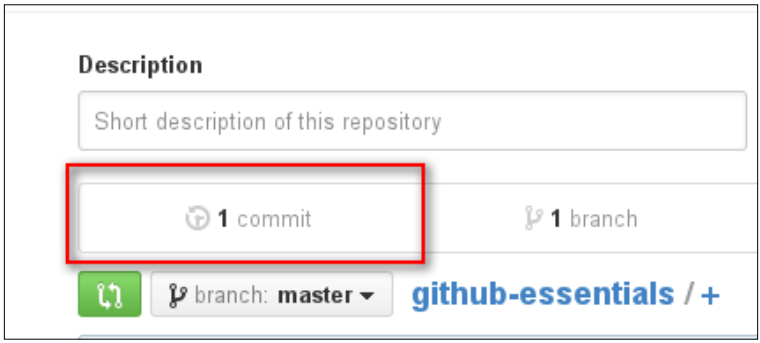
```
echo "# github-essentials" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin git@github.com:<username>/<repository>.git
git push -u origin master
```

Note that I use the Git protocol (`git@github.com`) that uses SSH underneath so I don't have to type my username and password each time. See the previous note on how to achieve this.

The directory name (in our example github-essentials) could be totally different from the repository name you entered upon creation. It is the remote URL you set with `git remote add` that must match with the repository URL GitHub provides.

www.it-ebooks.info

Every time you add more commits, their total number will also appear on the project's main page. In the preceding steps, we did our first commit, so the count is set to one and hence the **1 commit** option is shown in the following screenshot:



Click on the highlighted link as shown in the preceding screenshot to enter the commits page.

From here, you can browse the list of commits (so far we got only one) and visualize the output of `git log`. Let's compare those two commits:

Type `git log` in your local repository; the output should be similar to this:

```
commit 351b33bd7380a2434aaaea91eb8cb0ddc3b56852
Author: Achilleas Pipinellis <axilleas@axilleas.me>
Date:   Tue Apr 7 03:26:52 2015 +0300

    First commit.
```

Now, head over the commits page on GitHub. Here, you can see the same information depicted in a nice interface:

www.it-ebooks.info

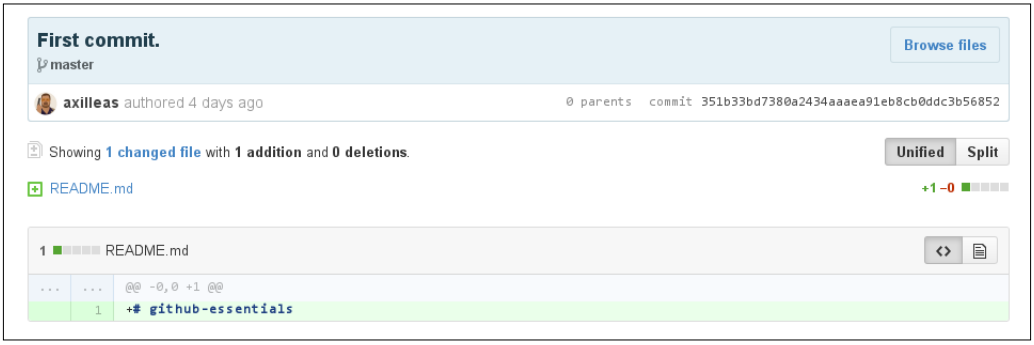
We can see the commit message, the time it was committed, and the SHA of the commit. Note that the SHA is stripped down to the first 7 characters out of 40. Clicking either on the SHA or on the commit message will show the changes introduced by that specific commit. Let's do that and compare what GitHub shows against the `git show <commit>` command:

```
commit 351b33bd7380a2434aaaea91eb8cb0ddc3b56852
Author: Achilleas Pipinellis <axilleas@axilleas.me>
Date:   Tue Apr 7 03:26:52 2015 +0300

    First commit.

diff --git a/README.md b/README.md
new file mode 100644
index 0000000..91bf601
--- /dev/null
+++ b/README.md
@@ -0,0 +1 @@
+# github-essentials
```

The result of the preceding code is shown in the following screenshot:



The commit message is shown in big bold letters since it conveys an important message. Right under it, there are the branches where the commit is included (currently it is only **master**).

www.it-ebooks.info

You can see the commit SHA, the author name, and the date right under the blue area. GitHub also tells you how many files changed during the last commit and how many additions/deletions were made during that commit.

Lastly, we see the added changes in green. If, instead, you remove something, it will be shown in a pinkish color as we will see later on.

The branches page and a comparison with the git branch command

Let's create a branch named `add_description` and checkout into it:

```
git checkout -b add_description
```

Next, edit `README.md`, add some text, make a new commit, and push it to GitHub:

```
echo "\n## Description\n\nGitHub for dummies" >> README.md
git add README.md
git commit -m "Add second level header to README file"
git push origin add_description
```

Now let's create a second branch named `new_feature`, out of the master branch and just push it to GitHub:

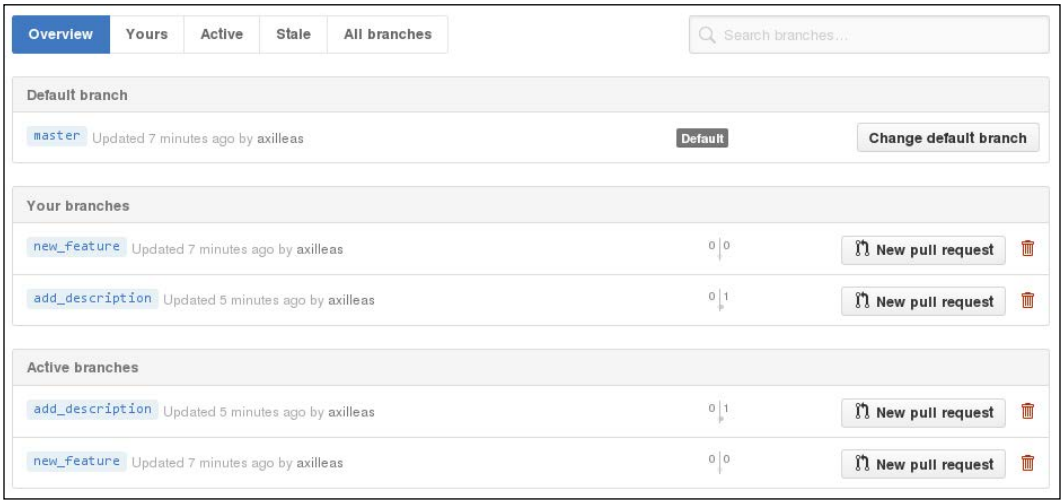
```
git checkout master
git branch new_feature
git push origin new_feature
```

Now is the time to switch to GitHub and see how all this information is presented.

In the main repository page, you can now see that there are **3 branches**:

www.it-ebooks.info

Click on the **branches** link to get more information:



The **Overview** page is, as the title suggests, an overview of the other tabs you see next to it. It tells us what the default branch is, what branches you have pushed from your account (same as the **Yours** tab), and the most active branches in the last three months sorted by date (same as the **Active** tab). The **Stale** tab represents the branches that haven't been updated more than three months.

You can change the default branch that appears in your project's homepage in the project's settings. This is covered in detail in *Chapter 6, Exploring the User and Repository Settings*.

You may notice that although we pushed the **new_feature** branch after we pushed **add_description**, its update time appears to be before **add_description**. This is only natural, since **new_feature** has the same commit date as our master branch that is dated before the **add_description** branch.

Now, if you look closely at a tab where the branches are shown, you can see written in small font the number of commits any branch is behind or ahead of the default branch, in our case master.

From the branches page, you can delete all the branches, except for the one you have set as default. Let's try and delete the **new_feature** branch. Click on the red trash icon and watch what happens.

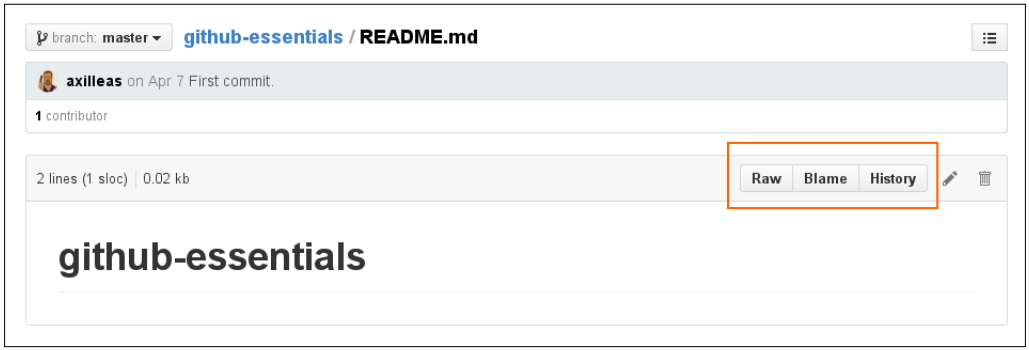
www.it-ebooks.info

GitHub gives you a chance to restore a recently deleted branch. Note that if you refresh the page or browse in another area of the page where you deleted the branch, the **Restore** button will disappear:

The **New pull request** button will be explored in a different chapter.

The Raw, Blame, and History buttons

These three buttons appear when viewing a single file of a repository. For example, let's visit the `README.md` file:

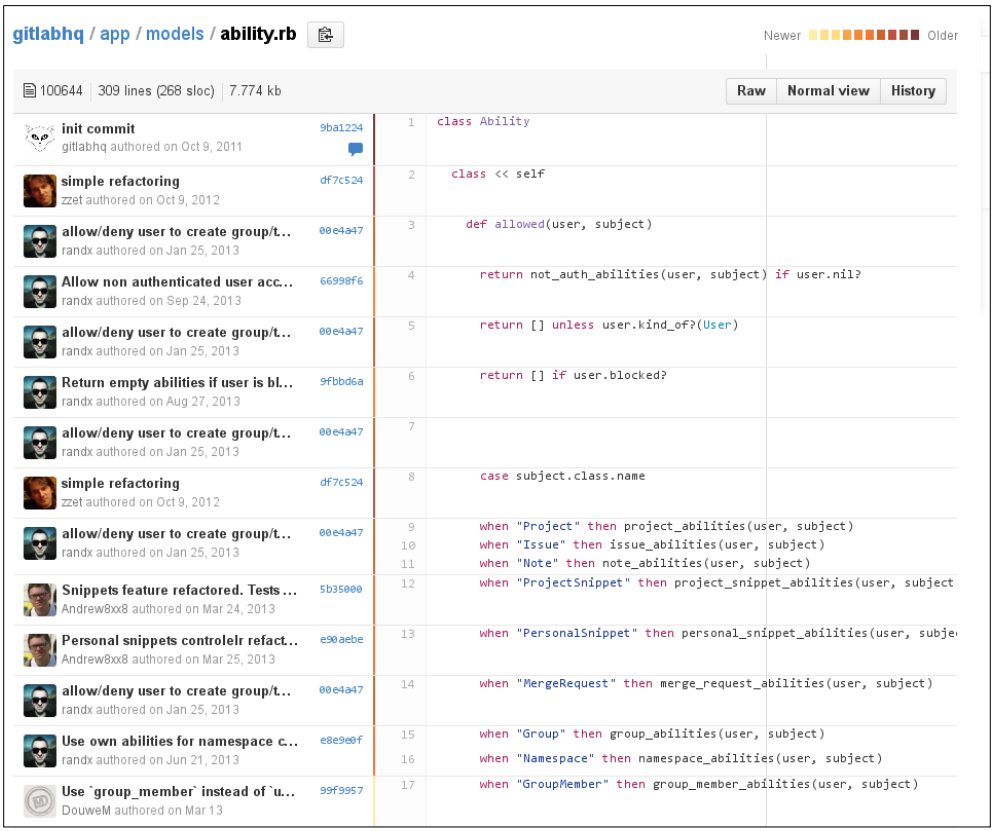


The **Raw** button, like the name suggests, opens the file in a raw form, meaning that any HTML formatting disappears. This is particularly useful when you want to download a single file. You will notice that many guides on the internet use this raw file format when they tell you to download something using command line tools such as Wget or Curl. If you ever tried to download a file from GitHub and all you got was an HTML file, remember the usage of raw.

www.it-ebooks.info

The **Blame** button makes use of Git's blame function. Basically, for each line of a file, Git informs you about who modified that line and when that line was modified. If you want to know more, visit `https://git-scm.com/docs/git-blame`.

In order to properly see how that works, I will not use our previously created `README.md` file, since there is not much information to see how GitHub uses this function of Git. Instead, I will use a file from another repository with more commits. Take, for example, `https://github.com/gitlabhq/gitlabhq/blame/master/app/models/ability.rb`, as shown in the following screenshot:



Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

www.it-ebooks.info

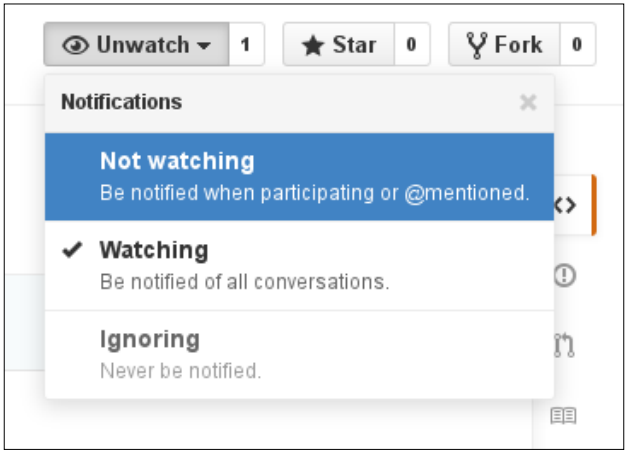
As compared to invoking `git blame` in the terminal, you can feel the superiority of GitHub's feature. Every line of code is annotated so you can see who, when, and what commit changed the particular line of the file. There is also this little nice feature of hotness. Older commits get a brown line whereas newer ones have a yellow color.

Finally, the **History** button is nothing more than Git's log function for a particular file.

The Watch, Star, and Fork buttons

You've probably spotted the three buttons sitting at the top-right of your repository page. These appear for every public repository, not only for your own.

The **Watch** button manages the level of subscription in a repository. GitHub notifies you with an e-mail whenever an action takes place in a repository you follow and, at the same time, lists them in the notifications area (`https://github.com/notifications`) where you can later mark them as read, as shown in the following screenshot:



There are three levels of subscription, going from never notified to Big Brother. You can choose to never be notified if a conversation begins, such as a new issue creation, comments left in a line of code, or if someone mentions you. This is the lower level of notification you can get. The next level is to be notified only if you explicitly take part in a conversation or if someone mentions you. Finally, the third option is to always get notified, whether or not you start a conversation or you get mentioned. This is the default behavior when you create a new repository.

www.it-ebooks.info

You can mention someone by prefixing their username with the at sign(@) . This is the special way in which GitHub can understand that you need someone's attention. Start typing the username and GitHub is smart enough to autocomplete it.

The **Star** button is a way to show your appreciation to a repository and its creator. It depicts the popularity of a project. Whenever you star a repository, it gets added to the list of your starred ones. You can see all your starred repositories at `https://github.com/stars`.

A list with the most starred projects on GitHub can be found at `https://github.com/search?utf8=%E2%9C%93&q=stars%3A%3E1&type=Repositories`.

You can see the people who have starred a repository by clicking the number next to the **Star/Unstar** button. For the repo I just created, you can see that I am the only stargazer:

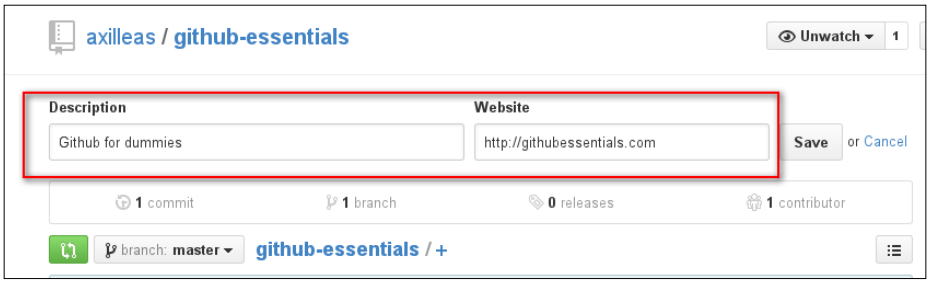
The **Fork** button and its purpose is what made GitHub excel in the first place. As we will see later in this book, its main use is when one wants to contribute to a project. When you fork a repository, it gets copied in your own namespace and that way you have full ownership in that copy; thus, you are able to modify anything you want. Go ahead and try it. Go to `https://github.com/axilleas/github-essentials` and press the **Fork** button. After a short while (depending on the size of the repository), you will be redirected to your own copy of this repo that you fully own.

Changing the description and URL

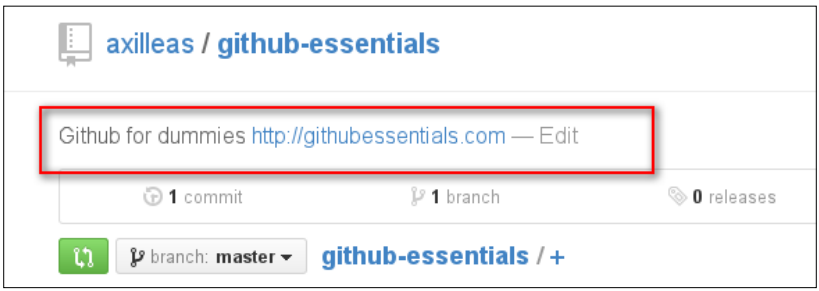
Previously, we learned how to add a description to our project. This was optional when creating a new repository, so if you opted out from creating it, let's see how to add it now.

www.it-ebooks.info

Head over the main repository page and you will be presented with two blank forms. In the **Description** field, put a descriptive note of your project; in **Website**, put the website URL that your project might have. This could also be your GitHub repository's URL. Here's what it looks like:



After you hit **Save**, you will immediately see the changes:



Learning how to use the powerful benefits of the issue tracker

GitHub provides a fully featured issue tracker, tightly tied to each repository.

Its primary use is that of a bug tracker, since reporting and discussing bugs play a vital role in the growth of your project. It can also be used to make feature requests, served as a discussion board of a blog or a project, and even used as a notepad for house repairing! For this, you can refer to the following links:

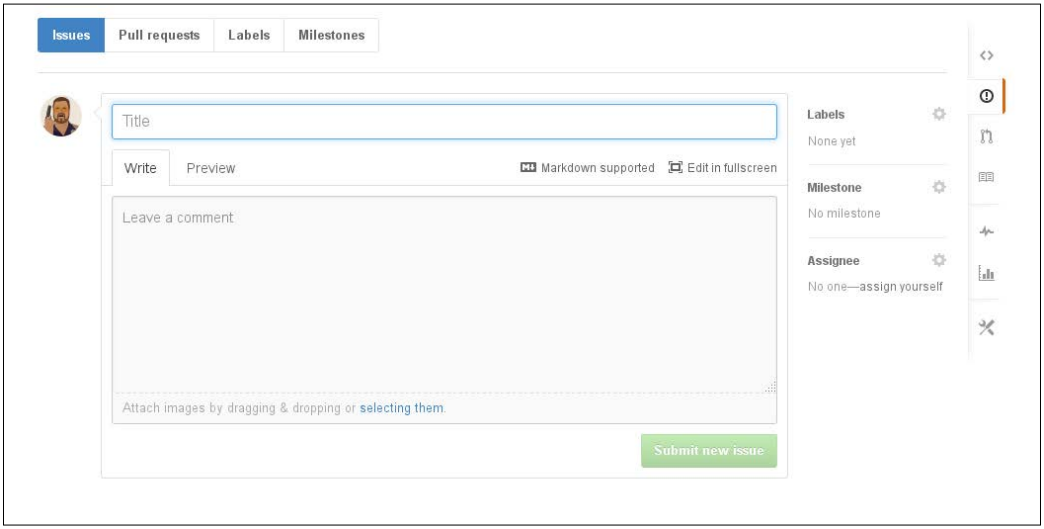
- `http://github.com/andreareginato/betterspecs/issues`
- `https://github.com/frabcus/house/issues`

www.it-ebooks.info

Creating a new issue

Go to `https://github.com/<username>/<repository>/issues` for an overview of all issue activity. If no one has ever opened an issue in your project, you will be presented with a blank page with GitHub urging you to open a new issue.

Let's go ahead and do this. Click on the big green button that says **New issue**:

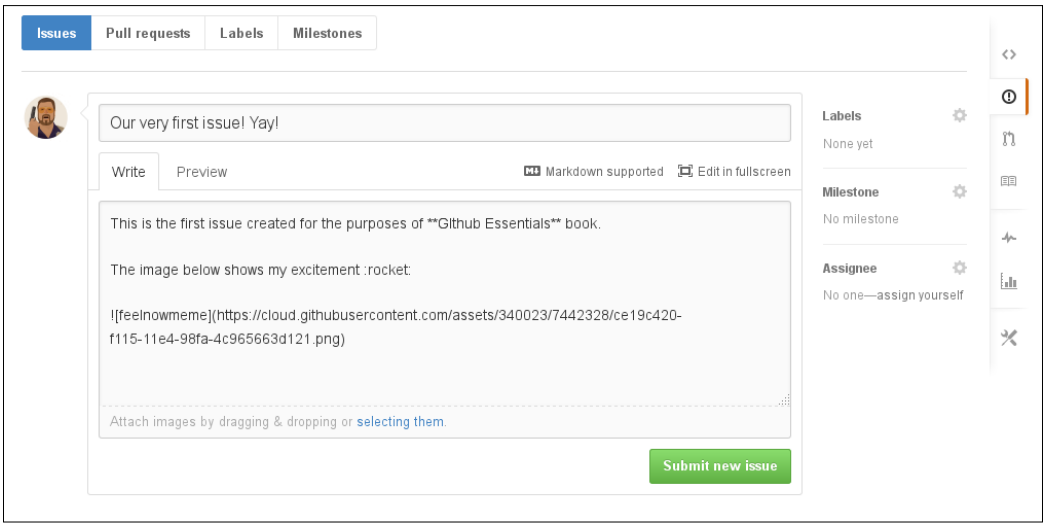


An issue is valid to be created when you at least provide a title. Watch the preceding screenshot carefully where the **Submit new issue** button is grayed out and cannot be clicked. The title should be as descriptive as possible to the message one tries to pass when creating an issue.

Below, under the **Write** tab, you can provide the details and essentially start a discussion with everyone who wants to participate (if the repository is public, that is). That's why GitHub cleverly suggests to **Leave a comment**.

www.it-ebooks.info

Besides writing, you can also attach images by a simple drag and drop or by selecting them using the folder navigation. Here's how the very first issue of this repository looks like:



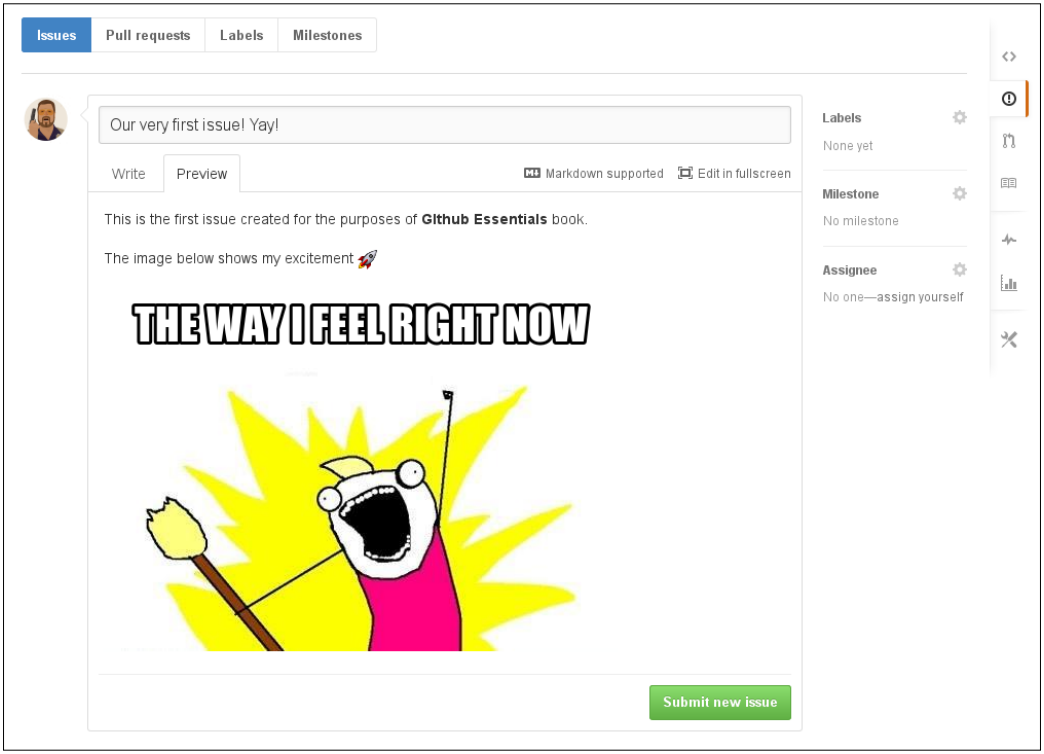
Next to the **Write** tab is the **Preview** tab. In order to understand its purpose, you must first learn about Markdown.

In brief, Markdown is a text-to-HTML conversion tool, so that you can write text that contains structural information and then automatically get converted to valid HTML. Written by John Gruber and adopted by GitHub (among many others), Markdown is the most well-known text-to-HTML conversion tool for its ease to use.

You can read all about how GitHub extends Markdown's functionality in the guide available at `https://guides.github.com/features/mastering-markdown/`.

www.it-ebooks.info

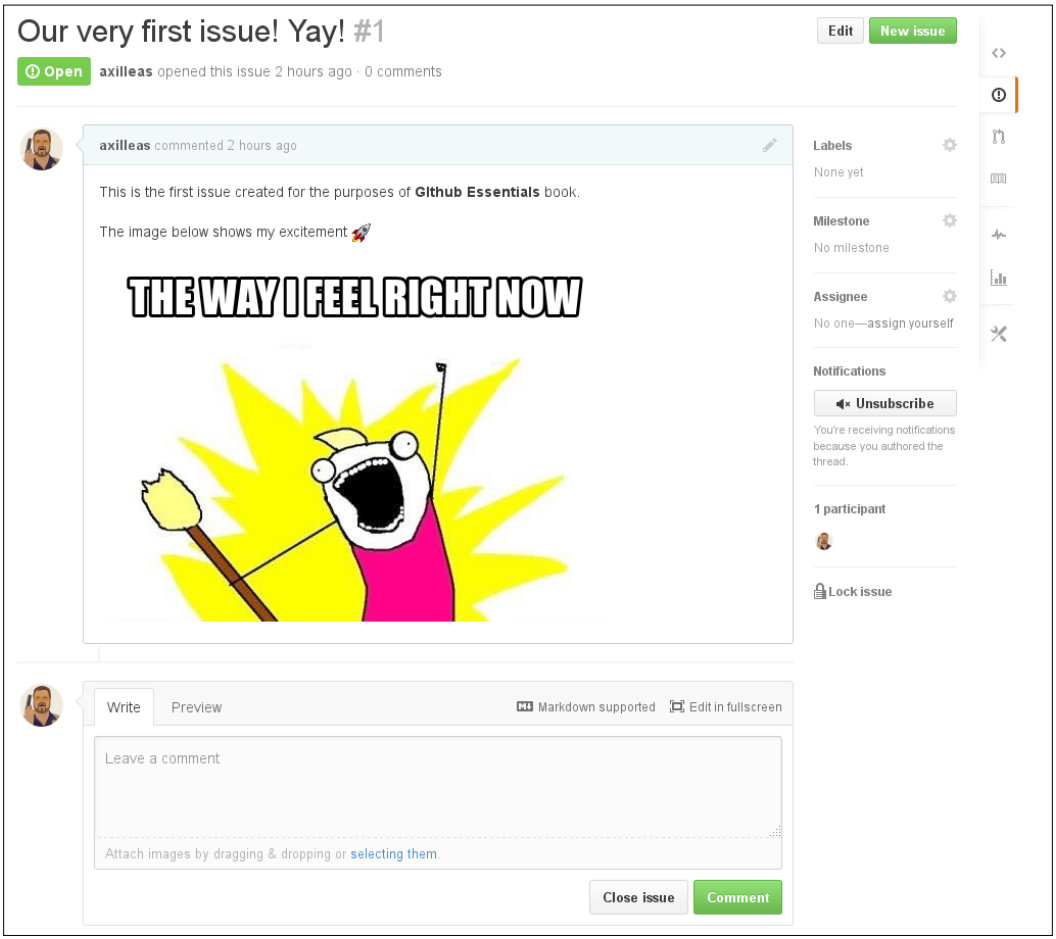
Now, back to our new issue; as the name suggests, **Preview** shows what the result will be when you submit the issue. It will style the regular text of the **Write** tab into a meaningful text accordingly, with URLs properly formatted, images shown, emojis displayed, and so on. The following screenshot shows how the preview of the issue is rendered:



As we will see later in this book, GitHub-flavored Markdown has many little gems that leverage the usage of the issue tracker. What you have seen here is just the tip of the iceberg.

www.it-ebooks.info

Feeling ready to submit it? Hit **Submit new issue** at the bottom of the page. Congratulations on making your first issue! The result will look like the following:



Each created issue is assigned a unique number that we can later use in other issues for reference. In our example, since this was the very first issue, it was assigned to number `#1`. Some useful information is provided in the title area. You can see that the issue is marked as **Open**, the username of the person who created it, the time it was created, and how many comments there are.

www.it-ebooks.info

If you later realize that you made a mistake, don't panic; you can always edit the issue you created. The **Edit** button allows you to edit the title, and the pencil icon is used for editing the description.

Close the issues with the **Close issue** button.

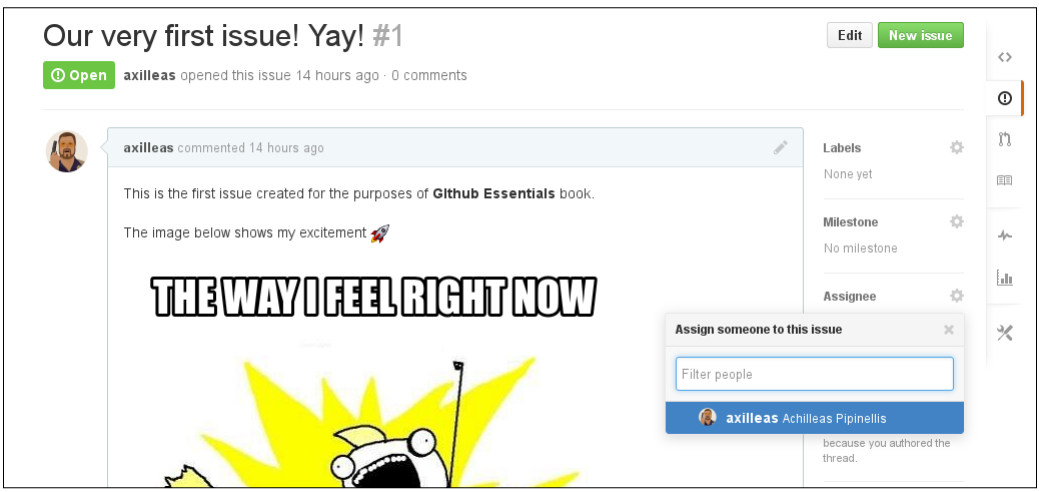
You can comment and close the issue at the same time if, for example, you also want to leave a note as to why the issue got closed. Start typing a comment and the button will change from **Close issue** to **Close and comment**.

Assigning issues to users

A repository can have more than one collaborator. A collaborator is a person who has push access to the repository and, in our case, can also edit and close issues.

User assignment works well in repositories with large traffic where a team is involved and is responsible for bug fixes, enhancements, and so on.

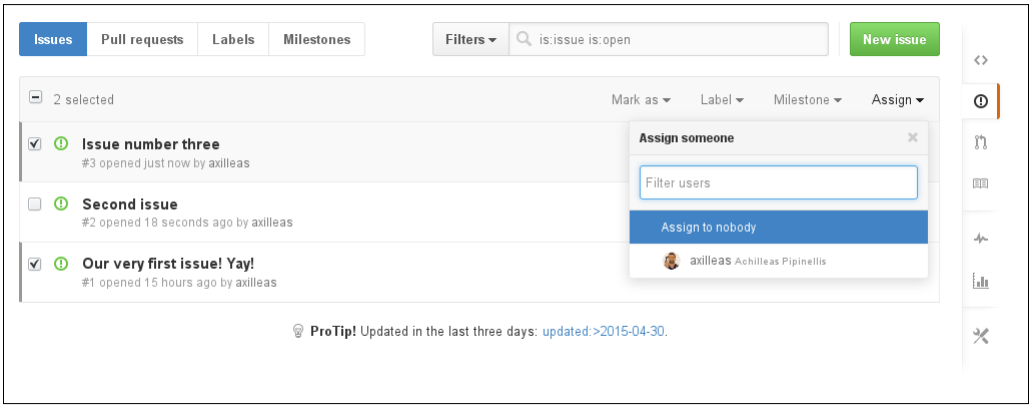
There are two ways to assign an issue to someone. First, as you have seen in the previous images, there is an **Assignee** section inside each issue:



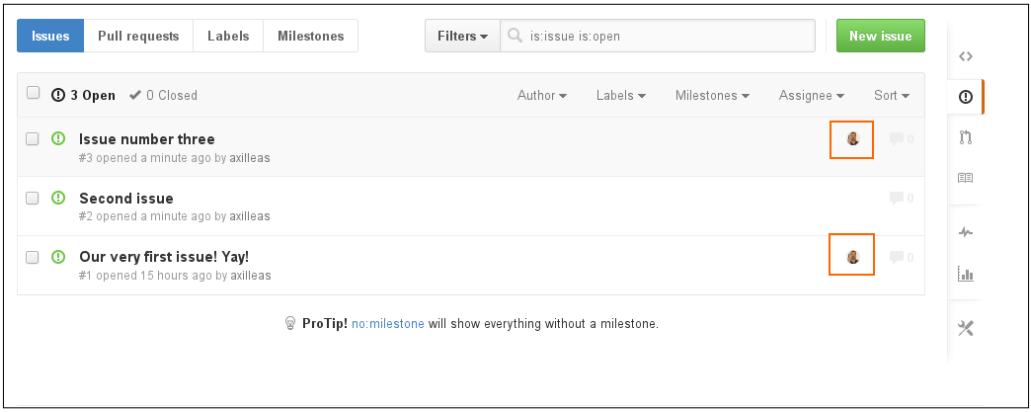
At this particular stage, there is only one collaborator, me, so only my name appears in the list. Okay, we learned how to assign an issue to a collaborator from inside the issue, but what happens if you have dozens of issues that you want to assign to someone? Going one by one and assigning them is a bit tedious and time consuming. You'll be happy to know that you can mass-assign issues to a person.

www.it-ebooks.info

For this purpose, let's create two more issues. Next, head over the **Issues** page, select the boxes of the issues that you want to assign, and select an assignee, as shown in the following screenshot:



After selecting the assignee, the issues will immediately get updated with the new information. You can see that the avatar of the assignee appears on each issue they are assigned to:



Each issue can have only one assignee.

www.it-ebooks.info

Labels

If you have worked with WordPress, labels are like tags. This is not to be confused with Git tags, though. We will explore how to create labels and use them effectively to easily categorize batches of issues.

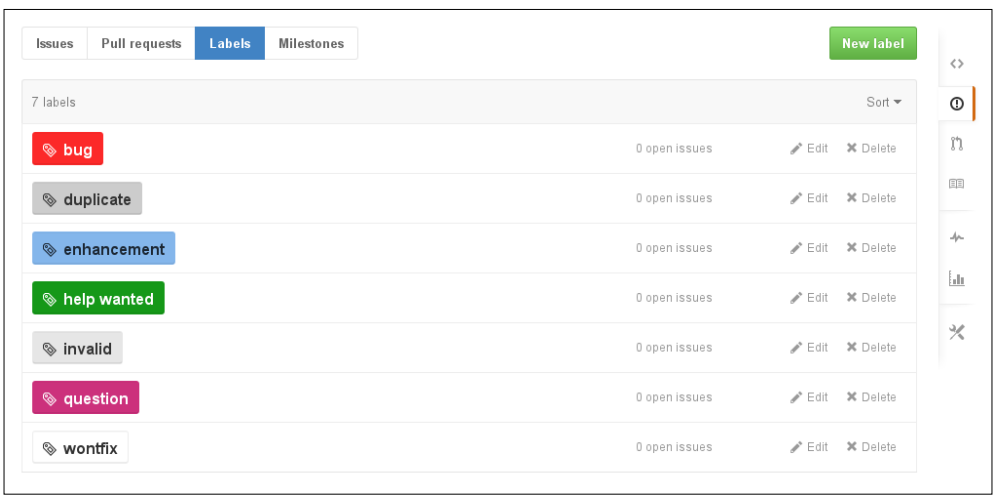
Why labels are a great asset to UX

Labels provide an easy way to categorize the issues based on descriptive titles such as **bug**, **feature**, and any other words you feel like using. They have colors and are visible throughout the issue tracker or inside each issue individually.

With labels, you can navigate to the issue tracker and filter any bloated information to visualize only the issues you are interested in. Let's see how that works.

Creating new label names and setting different colors

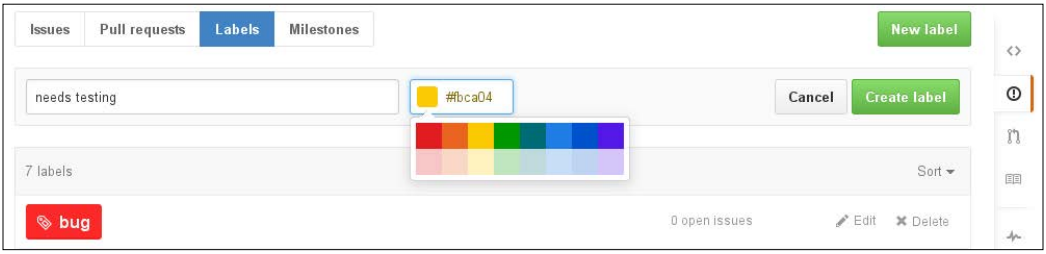
Head over the issue tracker and navigate to the label page by clicking on **Labels**, as shown in the following screenshot:



As you can see, GitHub sets up some predefined labels that are ready to use. The name and the color are fully customizable for new and existing ones.

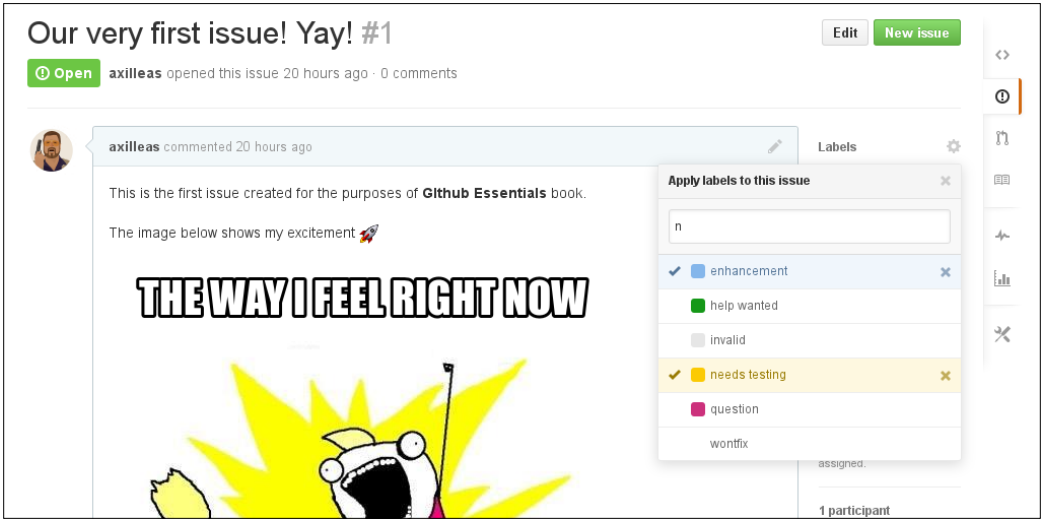
www.it-ebooks.info

Creating a new label is as easy as pressing the **New label** button, filling in the name, and choosing a color. In fact, a random color is already picked, so the only prerequisite is the name. I have created a new yellow label named `needs testing`, as shown in the following screenshot:



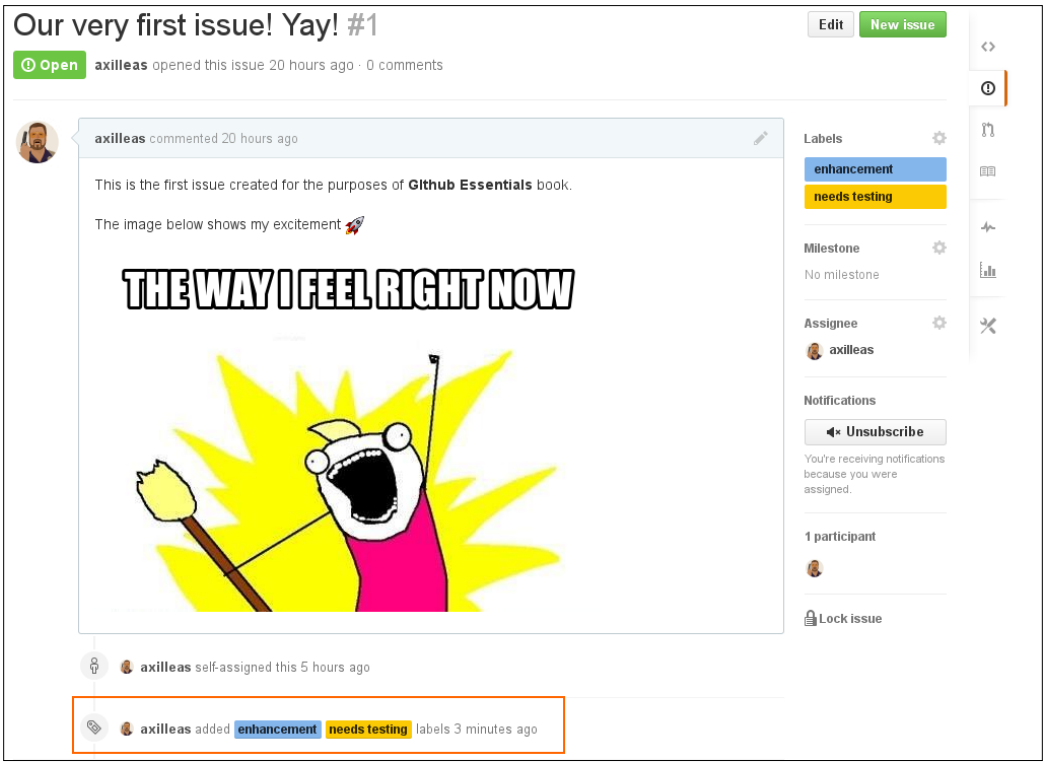
Immediately, the label is created and it appears in the list. Back to the issues, let's go inside the first one and give it the label we just created. Click on the gear for the dropdown to appear. Start typing to narrow down the search.

Now, we only have 8 labels, but imagine having more than 42. You'd have to scroll and scroll until you find it:



www.it-ebooks.info

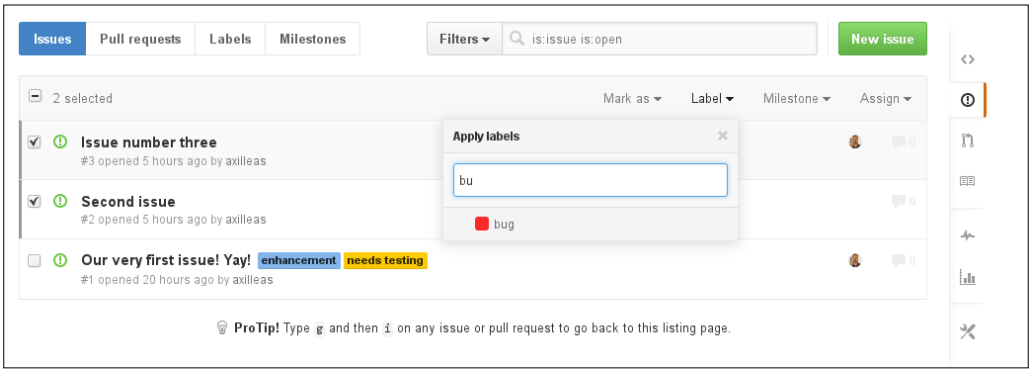
As you might have guessed, you can choose more than one label in an issue. After you choose them, just click anywhere out of the label window to save the action. Immediately, you see the changes:



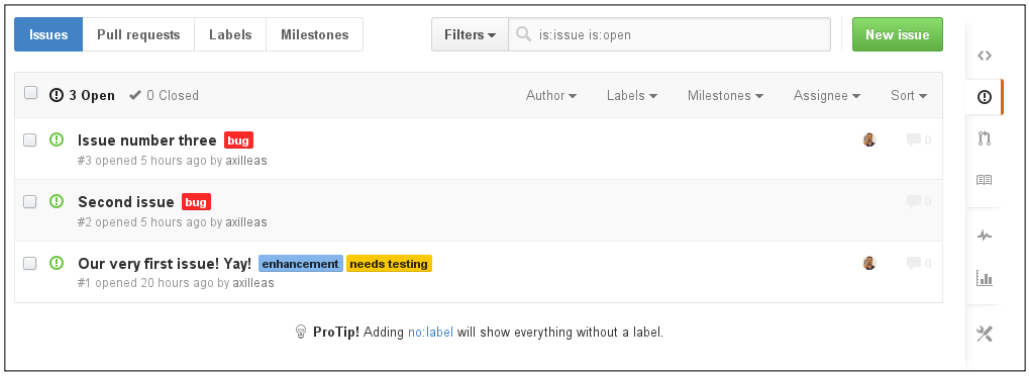
Notice how GitHub makes note of any change made to the issue. This way, you know who took a specific action and when the action was taken. Nothing escapes GitHub's eye! Try to remove the enhancement label to see what happens.

www.it-ebooks.info

As with the assignees, you can also mass-assign labels to issues. Let's try this by going to the main issues page and selecting some issues; then, choose the bug label:



The issue tracker gets updated and now you can have an overview of the issues with the labels assigned to them:

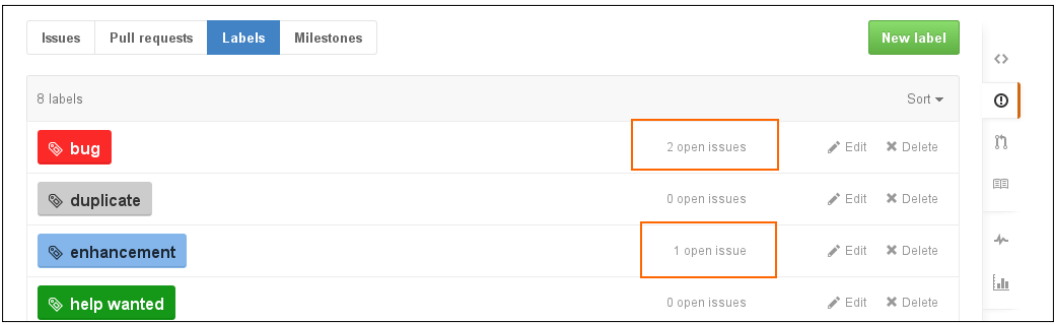


www.it-ebooks.info

Using labels to group issues

Suppose you have 100 opened issues, many labeled as bugs. Wouldn't it be cool if somehow only those issues appeared in the issues main page? Well guess what, clicking on the bug label, GitHub basically makes a query and, as a result, only the aforementioned issues appear. Grouping to the rescue!

Back to the **Labels** page, you can see that one can have an overview of the number of issues assigned to each label:



Milestones

Milestones, much like labels, are primarily used to group issues, but for different purposes. Consider a milestone like a special label that has a title, a description, and an optional due date.

Why milestones are a great help when working with code versioning

It is common knowledge that applications are released in versions. From the BIOS of your laptop to the web browser you use to explore the Internet, they all use versioning.

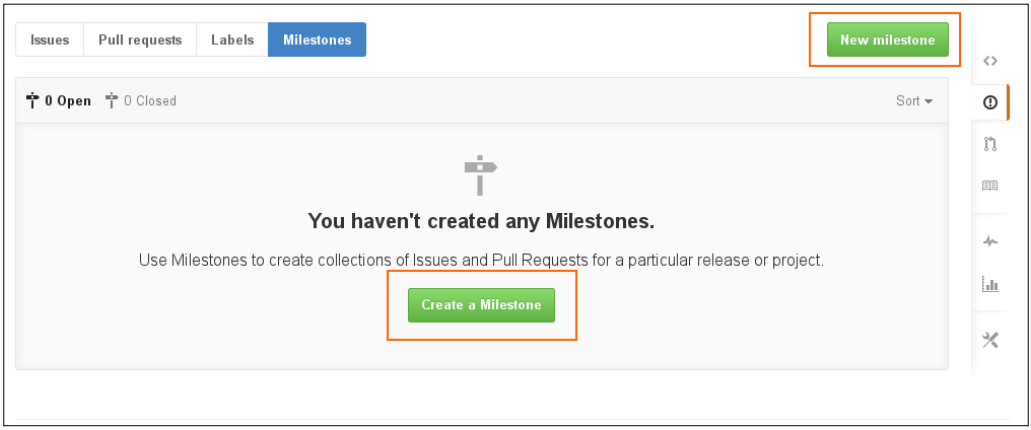
Many companies, or even community-driven open source projects, tend to have a road map that dictates the time the new product will be released to the public.

GitHub integrates this feature with the issue tracker. Let's dive in and learn how to create a new milestone, attach issues to it, and use the overview to see what issues remain resolved or unresolved.

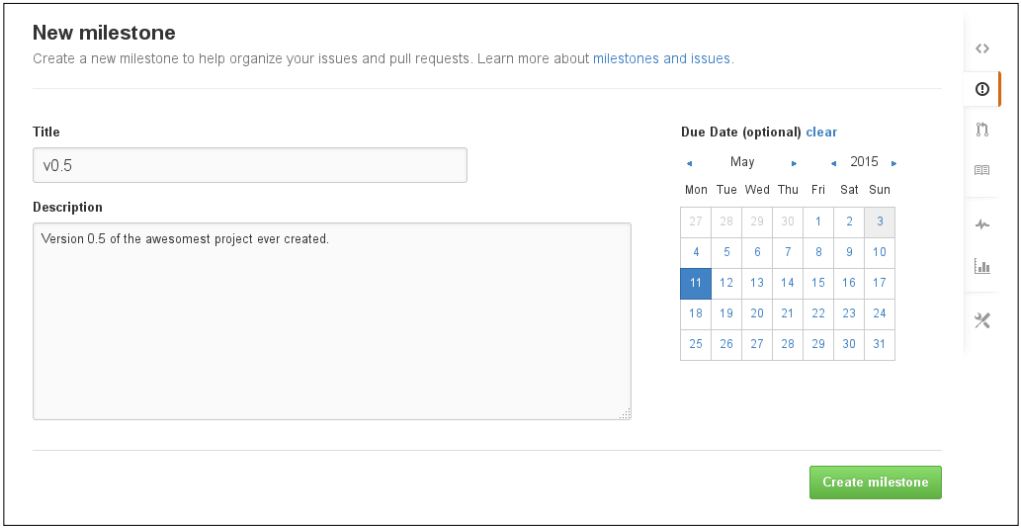
www.it-ebooks.info

Creating a new milestone

While at the main page of the issue tracker, click on the **Milestones** link, next to the **Labels** link. If no milestone has been created yet, you have two buttons that create a milestone. Generally, the **New milestone** button will be the main one to use, as shown in the following screenshot:

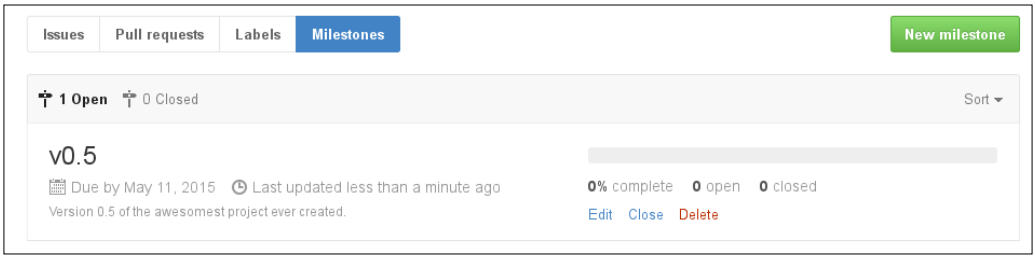


Now, let's create our first milestone. The only requirement is the title. The **Description** and **Due Date** fields are optional. However, to see what it looks like, let's add all the information:



www.it-ebooks.info

After you create it, it will appear in the **Milestones** page with all the information we previously entered:

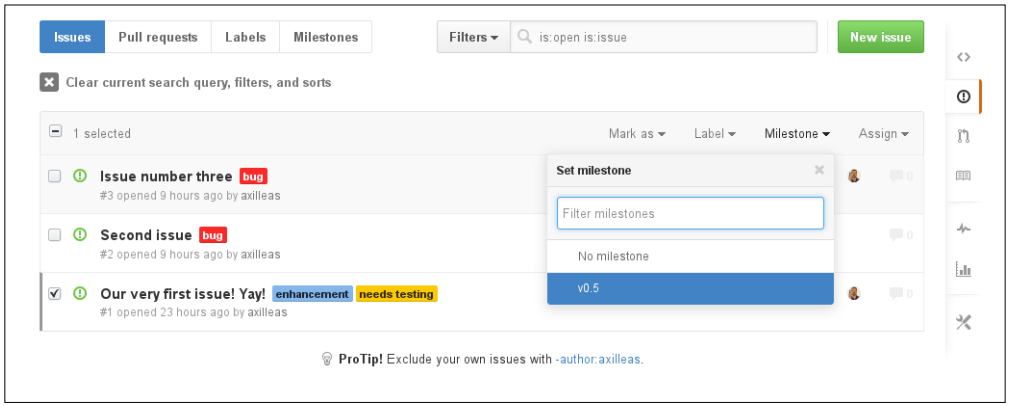


On the left-hand side, there is the name, the due date, the description, and a note of the time it was last updated. On the right-hand side, you can see the percentage of completion and the number of open and closed issues. Of course, you can edit, close, or delete it completely.

Adding issues to milestones

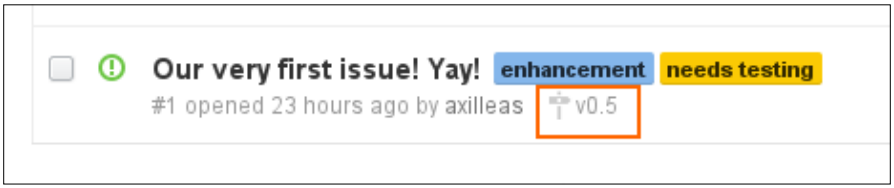
Now that we have at least one milestone, let's add an issue to it.

Again, there are two ways to add a milestone to an issue. Much like assignees and labels, you can do this inside each issue or mass-add it when in the issues main page. Here, I will try the second approach; you can try the first one on your own:



www.it-ebooks.info

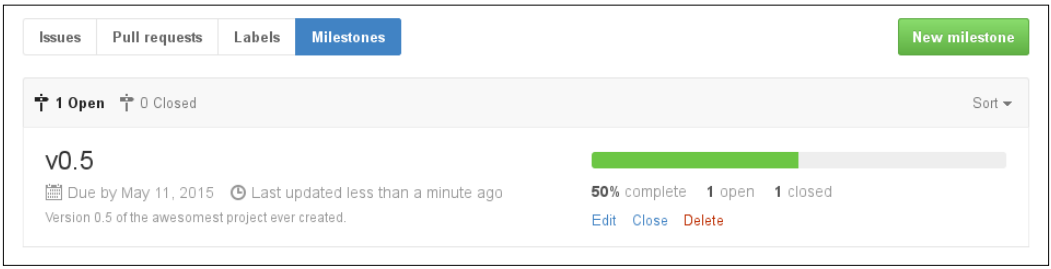
After selecting the milestone, the page will be refreshed and the issue will now be added to the selected milestone. If you watch carefully, you can see a small icon and the name of the milestone next to it:



Using milestones to see which issues are resolved or are yet to be resolved

When dealing with hundreds of issues, bug reports, and enhancements, it is nice to have an overview of what is resolved and what is not.

Let's add another issue to the milestone and immediately close it, marking it as resolved. Head over the milestones page and see that the bar is now half full (at **50%**):



Tips and tricks

The `README` files are essential for your project as they add useful information to the start page. Let's briefly explore this feature and then learn about keyboard shortcuts.

Learning about the README file

The `README` file is used to provide information about your project. Its contents are automatically shown on the front page of your repository, so it is always a good idea to provide one file.

www.it-ebooks.info

GitHub checks whether the `README` file comes with an extension; if it is supported for rendering, it automatically gets formatted according to its implementation.

For example, a `README` file can have a `.md` extension that stands for markdown, `.rst` that stands for restructured text, `.adoc` that stands for AsciiDoc, and so on.

If the extension is not supported, GitHub treats it like a regular text file and no formatting is done.

For a list of supported markups, go to `https://github.com/github/markup#markups`.

Navigating easily with keyboard shortcuts

A nice feature that GitHub has is the support of keyboard shortcuts. You can see what shortcuts are supported by hitting? on any page. A dialog box will pop up with all the supported shortcuts for that particular page. To see all shortcuts, click on the **Show All** link.

Summary

In this chapter, you learned how to create your first repository and explored its main page. You also learned how to effectively use the issue tracker in order to track your project's bugs, feature requests, and so on. Moreover, you learned how to use labels and milestones to better group the issues.

In the next chapter, we will learn about wikis as well as GitHub's feature about code release.

www.it-ebooks.info

2
Using the Wiki and Managing Code Versioning

GitHub provides a wiki-style place to add your project's documentation. You can create as many pages as you like and also grant public access to it so that everyone can edit it.

In addition, when you are the creator of a product and have users that rely on it, you will want it to be as stable as possible. Versioning helps to maintain an achievable goal. GitHub provides the right tools to release versions of your code, which in reality are just snapshots in time. Whenever you believe your project is ready to get out in the wild, whether bugs are fixed or new features are added, you can use the releases feature and deliver versioned tarballs to the world.

After finishing this chapter, you will have learned to create, edit, and maintain a wiki by providing a home for your documentation that will complement your project.

You will also learn how to create a new release out of an existing branch or tag accompanied with optional release notes. This way, the end user can understand the changes from any previous versions.

Using the wiki

When you first create a new repository, a wiki attached to this project is also created. It is enabled by default and everyone can add new content or modify existing pages. If you want to change this behavior, refer to *Chapter 6*, *Exploring the User and Repository Settings*, which shows how to accomplish this.

www.it-ebooks.info

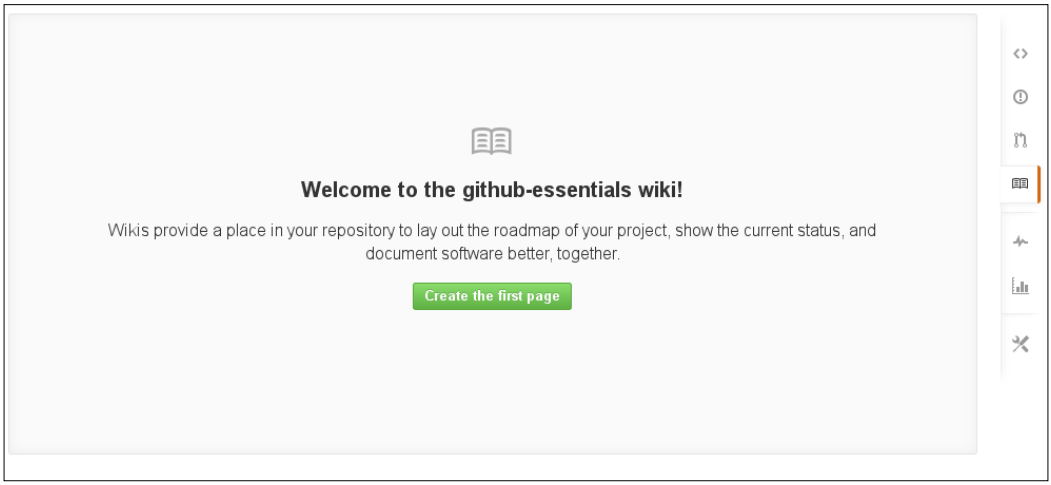
Why wikis are a nice place to document your project

Documentation is not to be taken lightly. To paraphrase a famous quote: "With great projects comes great documentation."

Although there are many tools that convert markup files, such as markdown to HTML, you may not want to use an external page to host your documentation. Enter GitHub wiki.

Create a new wiki page

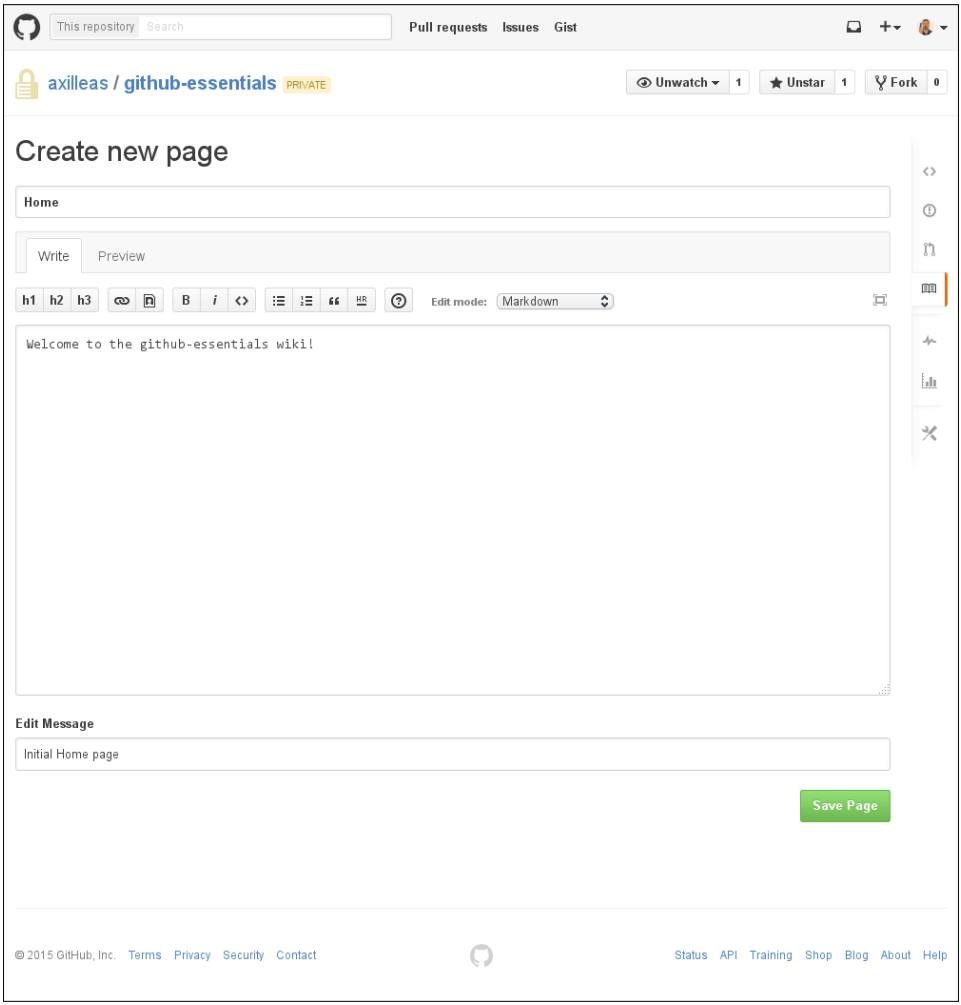
From the right-hand side bar, select the wiki tab (the one with the book icon) in order to head over the wiki. Since our wiki has no content yet, there doesn't exist any page. In this case, GitHub prompts you to create the first page:



Go ahead and hit the green button.

www.it-ebooks.info

Every time you add a new page to the wiki, the process is the same. At the top, there is the title. This is the only field that is mandatory in order to create a wiki page, as this is also used to form the URL from which you will have access to the page:



www.it-ebooks.info

When the very first wiki page is created, GitHub uses the title **Home** by default. Even if you pick another name, the **Home** page is created automatically and is used as the front page of your wiki. The name `Home` behaves in a way like `README` does for repositories, and it cannot be deleted.

Below the title area, there are two tabs. When the **Write** tab is active, you can begin to write in the blank area below. In case you choose to write in a markup language, the **Preview** tab renders the text and shows how it will be presented when you save the page.

Below the title, there is a nice toolbar that has the most common actions such as headers, bold text, italics, and lists. At the time of writing this book, GitHub supports nine markup languages to choose from. Pick one from the **Edit mode** drop-down list and the text will be rendered accordingly. For every language you pick from the menu, there is a little help page with the most common actions. Hit the question mark icon to see the help area.

Finally, when you are ready to save the page, you can provide a short message describing what the changes were about. Consider it like a Git commit message. Later, when we explore a page's history, the edit message will come in handy.

Whenever you are ready, press the **Save Page** button and the page will be created.

Deleting a page

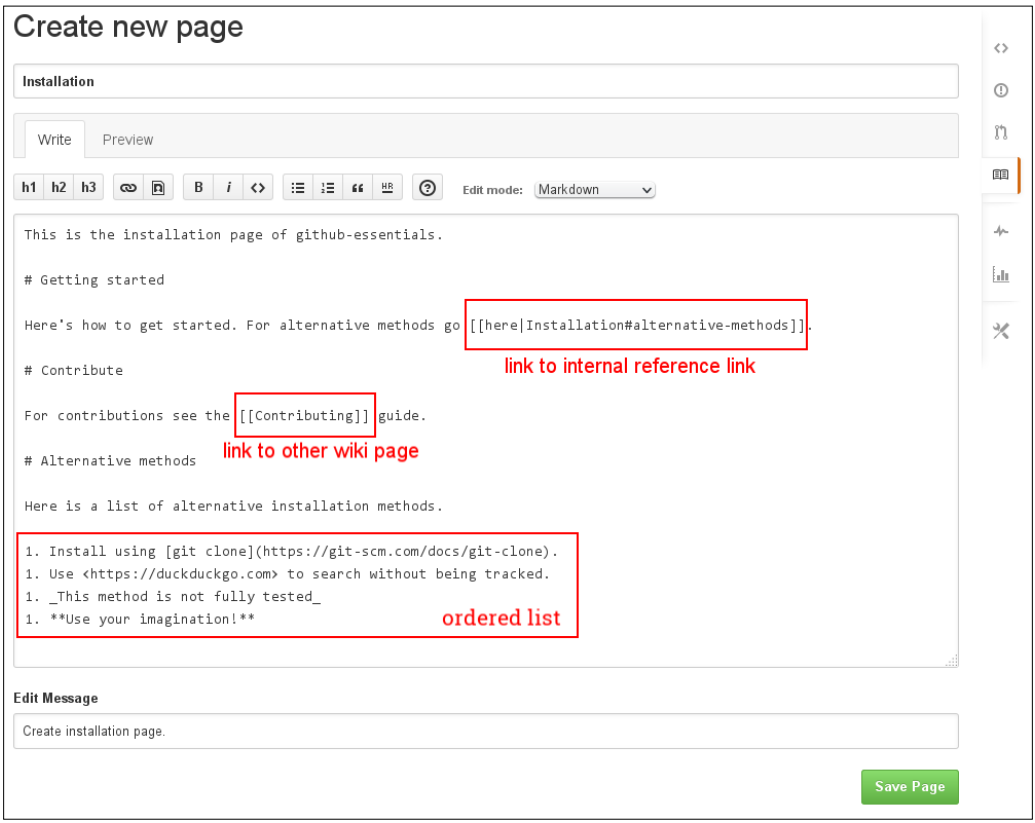
Every page except **Home** can be deleted. In order to do this, go to the page you want to delete and hit the **Edit** button at the right corner. As we will see later, deleting a page does not necessarily mean that it is purged forever. Read ahead and learn how to undo things.

www.it-ebooks.info

A Markdown-powered wiki – an introduction to Markdown

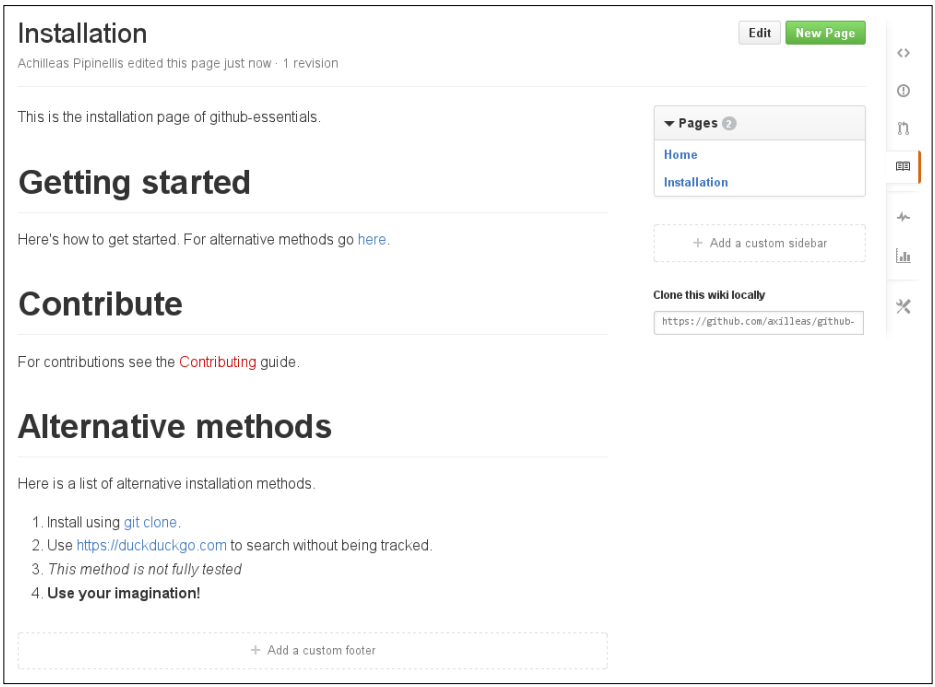
While GitHub supports multiple markup languages, we will explore Markdown as it is the most well-known one.

Let's create another page, namely `Installation`, with some content as follows:



www.it-ebooks.info

I have used several Markdown elements, and hitting **Preview** will show you how the page will be rendered when it is saved. After you hit **Save Page** button, the new **Installation** page will look like the following:



Some elements worth mentioning are the links. There are two kinds of links: external and internal. External ones are written by giving the full URL including the FQDN, whereas internal links only get called with the page name.

You can have external links that display the actual URL by surrounding them with `<>`, such as `<https://duckduckgo.com>`, and you can also put some random text such as `[git clone](https://git-scm.com/docs/git-clone)`. Inside the brackets, you can put any text you want, followed by the actual link inside parenthesis. Be careful to not leave any spaces between the second bracket and the first parenthesis.

Internal links are useful when you want to link to another page of your wiki. Imagine you have 42 pages and you have to type the whole URL whenever you would like to refer to another page. GitHub implements MediaWiki's markup in that case. Use double brackets `[[]]` and inside it, put the name of the wiki page you want to link to. In our example, I used `[[Contributing]]` and this will create a link to another page. Notice that if the link does not exist, it is rendered in red color. If you click on it, you will be redirected to create the page.

www.it-ebooks.info

When creating headers, you use # before the text. The number of # defines the header style that will be used. Each header gets a separate anchor link, which you can see if you place your mouse over it on a saved page. This anchor link can then be used to reference internal links.

In our example, you can see that I created three headers namely `Getting started`, `Contribute`, and `Alternative methods`. In `Getting started`, I placed an interlink with a reference to `Alternative methods`. The piece of markdown that did this is `[[here|Installation#alternative-methods]]`. This style introduces two new areas to explore.

Firstly, you can see that an alternate text can be used much like with external links. The only difference is that both the alternate text and the link are placed inside the double brackets separated by a pipe (|). Secondly, you can see how the call to the internal reference link is made. The page title goes first followed by the octothorp sign (#) and last is the header. It is important to understand that the header as part of the interlink gets transformed, where empty spaces are replaced with hyphens (-) and all special characters (`?`, `'!`, and so on) are lost.

You can always use the preview to test whether an anchor link will be rendered correctly.

Internal links are only supported inside the same wiki. You cannot link to another wiki with an internal link. In this case, you will have to use external links.

We have only scratched the surface regarding Markdown. You can read more about it in the nice cheat sheet at `https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet`.

How to add a sidebar and a footer to your wiki

If you have write access to a wiki, you should be able to see the **Add a custom sidebar** and **Add a custom footer** buttons.

GitHub has a default sidebar where it places all the pages of the wiki. This might not be useful since they are shown in name order and sometimes you want users to be able to access the important information without searching too much.

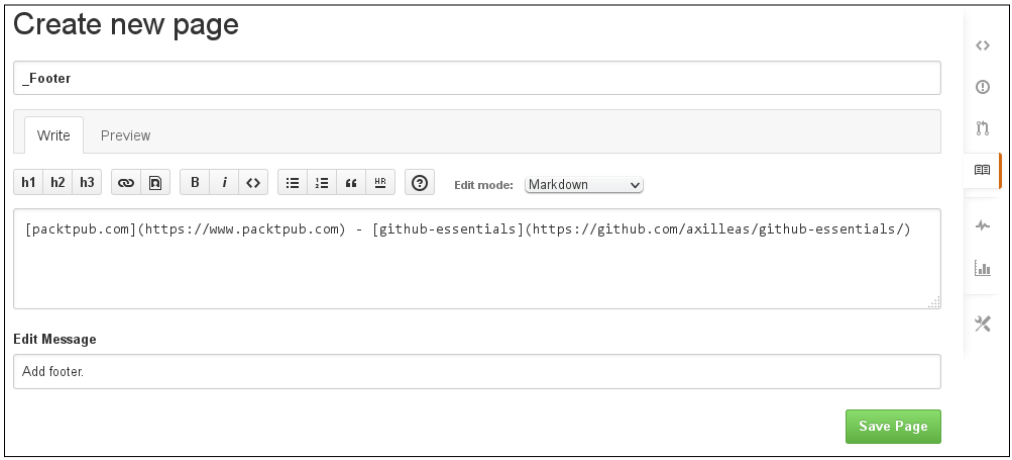
www.it-ebooks.info

Much like any other wiki page, the sidebar can be written in a markup language that GitHub supports. In the following example, I used **Markdown**:

As you can see, I used a bulleted list and links on each item. Indenting an item (one or more spaces) will provide the following result:

www.it-ebooks.info

Like sidebar, you can also create your own custom footer. For example, I used two external links with a custom text, as you can see in the following screenshot:



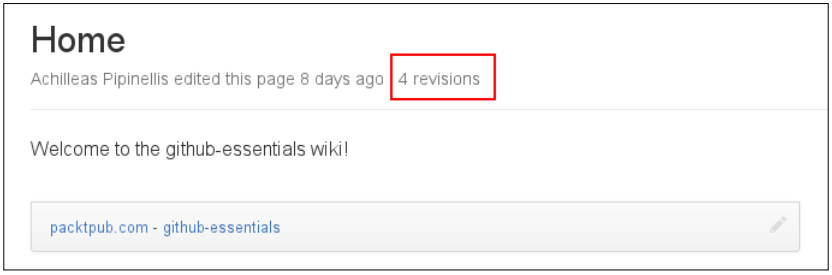
After all modifications, we get a nice wiki page with our custom sidebar and footer:

www.it-ebooks.info

Watching a wiki page's commit history and reverting to a previous state if needed

Would you be surprised if you were told that a wiki is essentially a separate Git repository? In the *Tips and tricks* section, we will see how to clone a wiki locally, make changes, and push back to GitHub.

As with all Git repositories, there are commits and a history log. Each page gets a filtered log of the commits and changes it has undergone. One quick way to access the history log is to click on the revisions link on each page. This can be found under each page title. Take, for example, the **Home** page:



A link to **Page History** can also be found when you edit a page.

One other way to see the history log is to append `/_history` to your page. So, for example, `https://github.com/axilleas/github-essentials/wiki/Home` becomes `https://github.com/axilleas/github-essentials/wiki/Home/_history`.

Here is what my **Home** page log looks like:

www.it-ebooks.info

From the preceding screenshot, you can get a lot of useful information. You can see that the username of the person who made the change shows first in the history table.

In this example, there is only mine, but in a wiki with many collaborators, you can easily tell who made what change. Then, you get the commit message that is super useful because you can tell from a glimpse what the change was about. The third column is about the time the change was made and, finally, there is the commit SHA of the change.

You can tell the default GitHub commit messages from the custom ones since they follow the pattern `Created/Updated/Destroyed Title of page (language)`. Where language the markup language that was used to create the page.

Now, let's use the power of reverting when things go south. Firstly, create a new page, save it, and then delete it. We cannot go to that specific page's history log since it is no longer there, so head over the main **History** page of all pages.

In order to revert things, you need to use the **Compare Revisions** button. You can choose between one or two revisions to revert from:

Here we chose two, but since they are one after another in the commit history, choosing only the last one would be the same. It's like comparing the changes between `git show HEAD` and `git diff sha2 sha1`, where `sha2` is the last commit `SHA` and `sha1` is the one before it. The `diff` is the same.

www.it-ebooks.info

After hitting the `Compare Revisions` button, we will see the change that was introduced with this commit:

Let's bring back the deleted page by hitting the **Revert Changes** button. At the time of writing, every time I tried to revert the deletion of a page, I was presented with 500 internal server errors. Despite the error, go back to the **History** page, and you will see that the revert was indeed performed and the deleted page was brought back from the grave:

With blue mark, you can see the last commit where the page was deleted and with red mark, you can see the previous one where the page was created.

Sometimes, you will not be able to compare any two revisions and revert due to conflicts. In this case, GitHub will warn you with a message:

This patch was not able to be reversed.

That's all there is about GitHub wikis. Next we will focus on managing code releases with the tools GitHub provides.

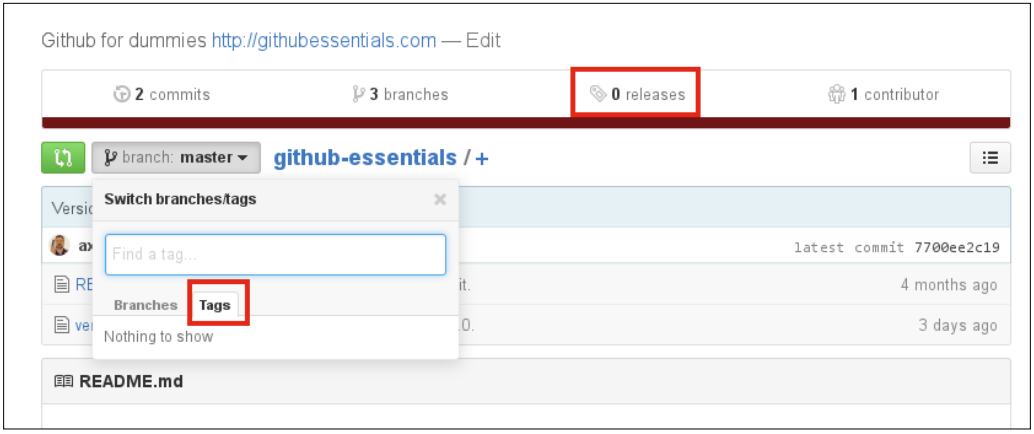
www.it-ebooks.info

Managing code versioning

In the world of software management, almost every piece of software is shipped with a version. It is a way to declare its evolvement through time; usually with the addition of enhancements or bug fixes. GitHub leverages the power of Git and provides a simple interface to ship your versioned software.

Creating a release

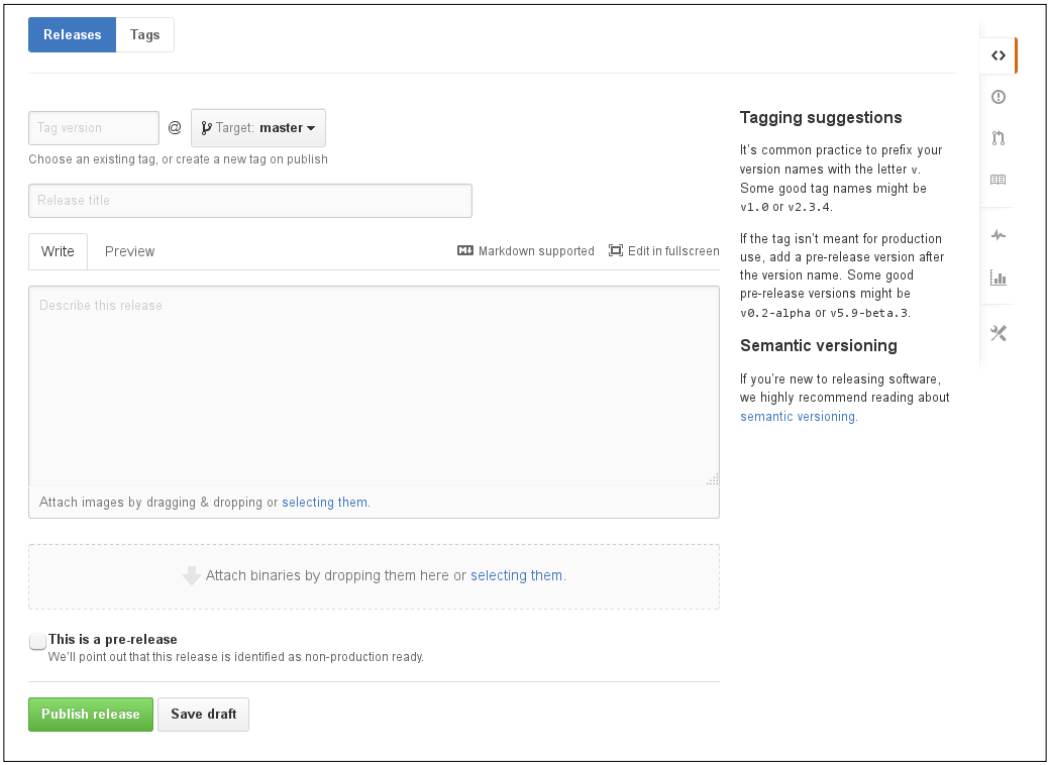
In GitHub, the notion of a release is tightly tied to Git tags. You can see the existing tags, if any, from the same menu where you change a branch, as shown in the following screenshot:



If you visit the **Releases** page and there is no tag created yet, you will be prompted to create one. Creating a release will automatically create a tag.

www.it-ebooks.info

Let's click on the **Create a new release** button. The following page will appear:



You only need to fill in the name of the **Tag version** box, everything else is optional. If the tag name you provide already exists, you will be presented with a duplicate tag name notification.

Your tag name can be any arbitrary value, but it is highly recommended to follow the semantic versioning scheme. Briefly describing what semantic versioning is, a release number consists of three numbers separated by dots in the form of `MAJOR.MINOR.PATCH`. You should then increment the following:

- The `MAJOR` version when you make incompatible API changes
- The `MINOR` version when you add functionality in a backwards-compatible manner
- The `PATCH` version when you make backwards-compatible bug fixes

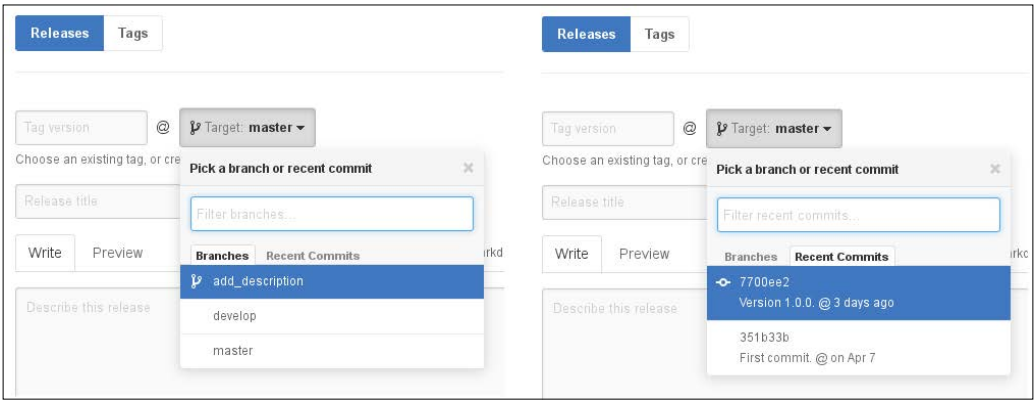
www.it-ebooks.info

Additional labels for prerelease and build metadata are available as extensions to the `MAJOR.MINOR.PATCH` format.

You can read more at `http://semver.org/`.

One great way to name your tags is to match the existing milestones. From the previous chapter, we already had a `v0.5` milestone so let's also name the new tag `v0.5`. Start typing it and if the tag does not exist, you will see the **Excellent! This tag will be created from the target when you publish this release** message.

You can choose the target branch or commit from the drop-down menu as shown in the following screenshot:

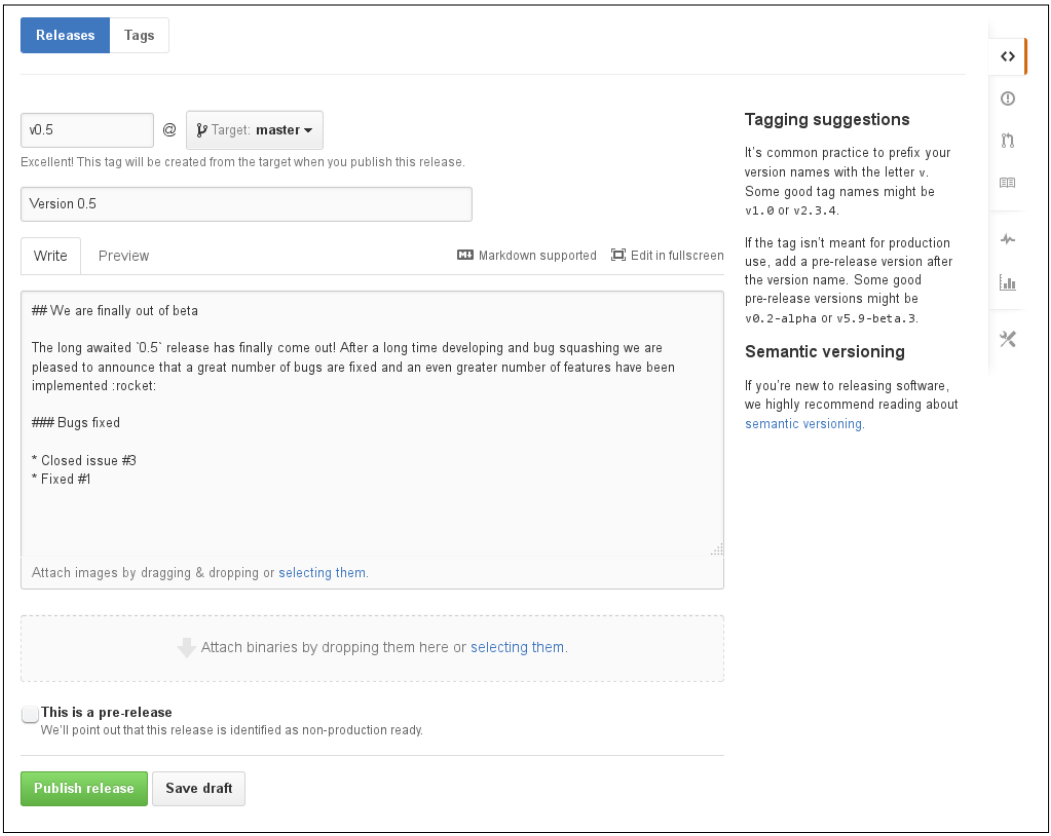


If you choose a branch, a tag pointing to the latest commit in that branch will be created. If you, instead, go to the **Recent Commits** tab, you can choose from a number of recent commits to create a tag from.

For the sake of our example, let's choose the master branch and enter a release title. Optionally, but recommended, you add a description of what this release is about. I like to consider the description like writing a blog post of what changed in this release.

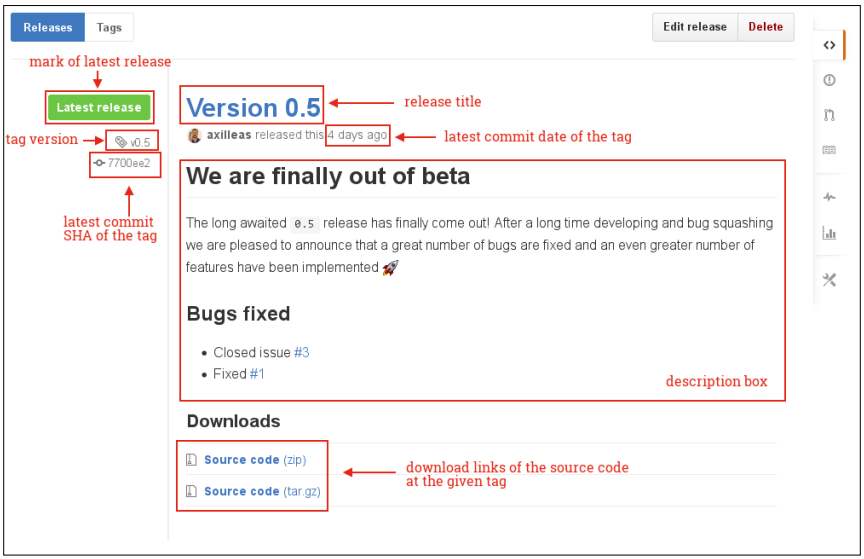
www.it-ebooks.info

You can, of course, use markdown like almost everywhere in GitHub and use the **Preview** button to see how it will be rendered:



www.it-ebooks.info

If you think everything is in order, hit the **Publish release** button. You can always edit any release anytime, so do not worry if you miss something. The following screenshot explains all the information about a release:



It's worth noting that the release date is the date of the latest commit. Although I just released `v0.5`, you can see it dates **4 days ago**.

If a release title is not provided, the tag name will be shown instead. Likewise, if a description is not provided, the latest commit message of the tag will be shown instead.

Editing a release

In order to edit a release, you must visit its own page, meaning clicking on the version name at the releases page. The edit button will then appear and you will be able to change what you like.

www.it-ebooks.info

Pushing a tag from the command line

Now, let's see how GitHub behaves when a tag already exists. I made a few changes to a file and created a new tag from the command line. Finally, I pushed this tag to GitHub as follows:

```
git tag v0.5.1
git push origin v0.5.1
```

If you now visit the tags page, you will see the new tag above the one we made before. In the releases page, click on **Draft a new release**. We will choose an existing tag, so at the **Tag version** field, type `v0.5.1`. GitHub found out that the tag already exists so it informed us that this is an existing tag:

Give it a title, a brief description, and then publish it. Since the `v0.5.1` tag refers to a latest commit rather than the previous release, it now gets marked as **Latest release**.

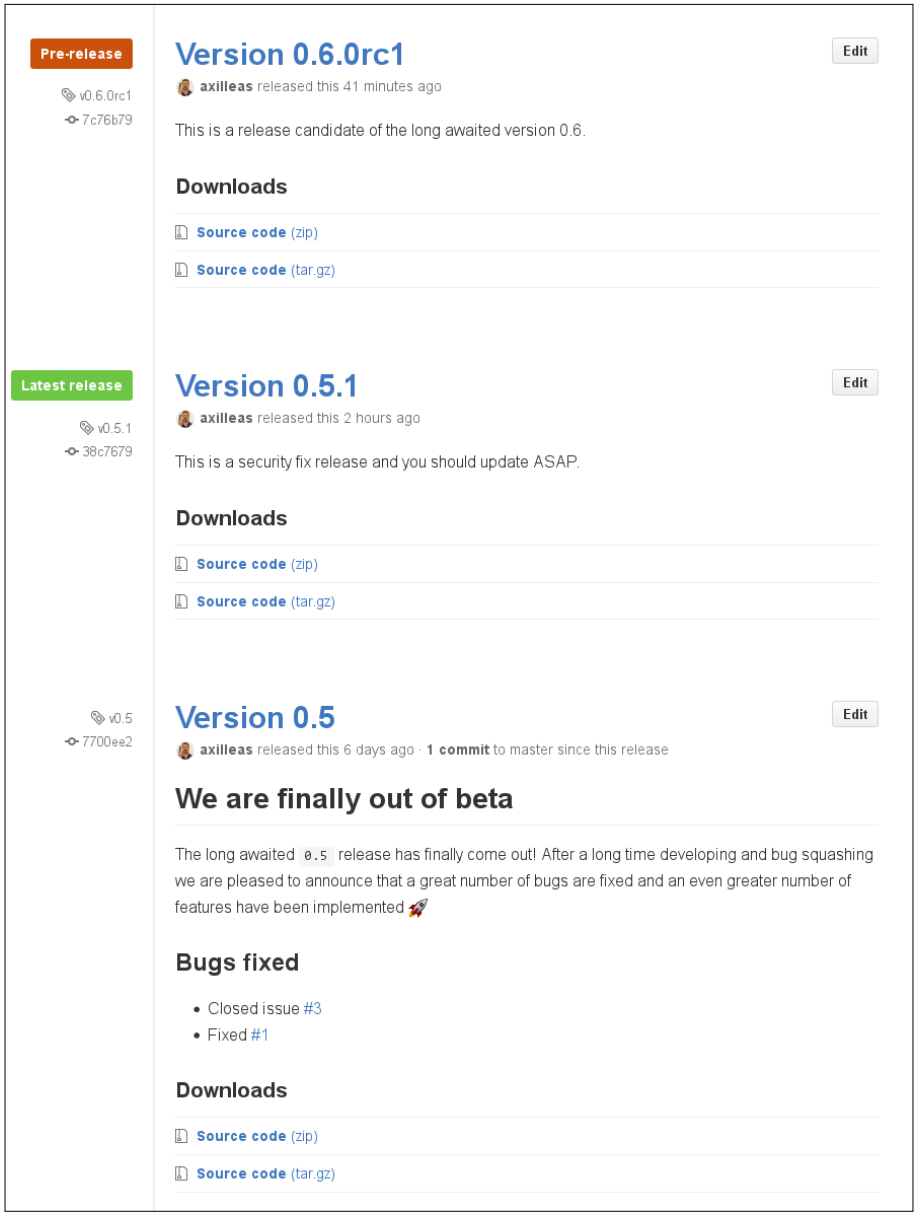
Marking as prerelease

A nice little decorative feature is that you can mark a release as a prerelease, meaning to inform the users that it's not ready for production, but they can still download and test it. Marking as prerelease is as easy as ticking the relevant option:

Let's then make a prerelease of the develop branch, which contains new commits that do not exist in the master yet, and name it `v0.6.0rc1`.

www.it-ebooks.info

After publishing it, here is how the releases page looks like:



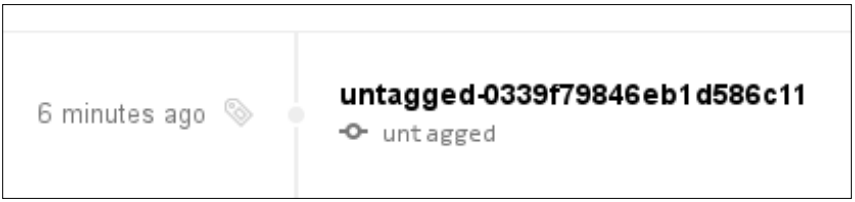
www.it-ebooks.info

Making a draft of a release

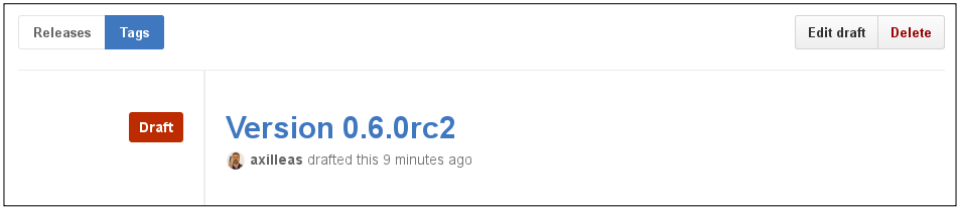
If you tend to provide a detailed description with each release, you may find the draft feature quite useful. You can repeatedly edit a release while adding the required information and then save it as a draft.

This way, you can spend less time when the time comes to publish it.

A draft release is marked as an untagged reference, as you will see in the releases page:



To continue editing it, click on the untagged link and hit **Edit draft**, as shown in the following screenshot:



Since you are working on a draft, you don't have to worry about changing the tag of the release or any other information for that matter. Drafts can only be viewed by those who have write access to the repository, so it is not shown to the public until they are published.

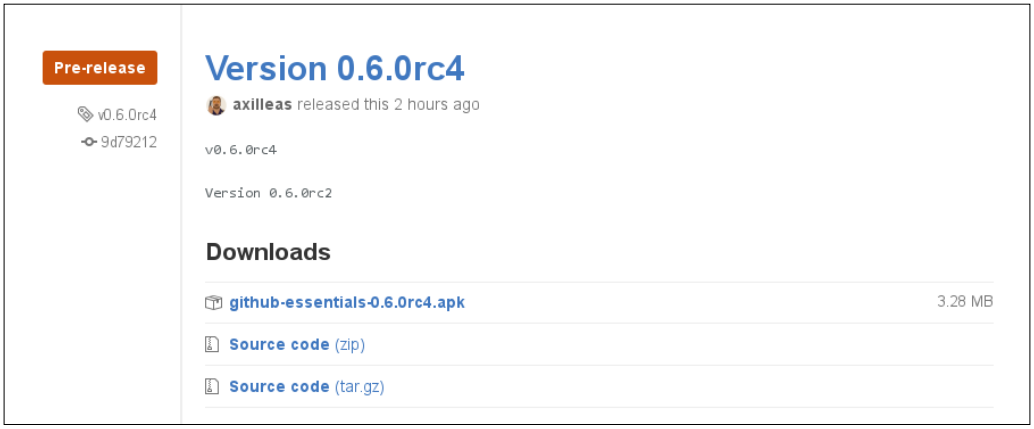
Uploading your own files

There are cases where you might want to provide precompiled binaries for a variety of operating systems. For your Android application, it would be the `apk` files; for Windows, `msi` or `exe`; for Debian, `deb`; for RedHat, `rpm` and so on.

www.it-ebooks.info

When you create a release, there is a window at the very bottom that tells you to attach any binaries. Here I uploaded a test `apk` file as you can see in the following screenshot:

You can upload multiple files, but bear in mind that GitHub limits the upload size to 2 GB for each file. After you successfully upload the new binary and publish the release, you can see the files you manually attached along with the source code GitHub released for you:



Tips and tricks

Here is a tip to get notified about new releases in an atom feed. Also, while you familiarize yourself with Git, you'd be happy to know that you can edit a project's wiki locally.

www.it-ebooks.info

Subscribing to new releases via atom feed

If you are used to subscribing to feeds to learn the news on your favorite blogs, you will be happy to know that you can subscribe to get notified about new releases on GitHub.

Simply go to the releases page and append `.atom` at the end of the URL. For example, `https://github.com/diaspora/diaspora/releases` becomes `https://github.com/diaspora/diaspora/releases.atom`. Cool, isn't it?!

Editing the wiki locally

As mentioned in the wiki commit history section, every wiki is a separate git repository. As such, you can clone it, make changes locally, and push back to GitHub.

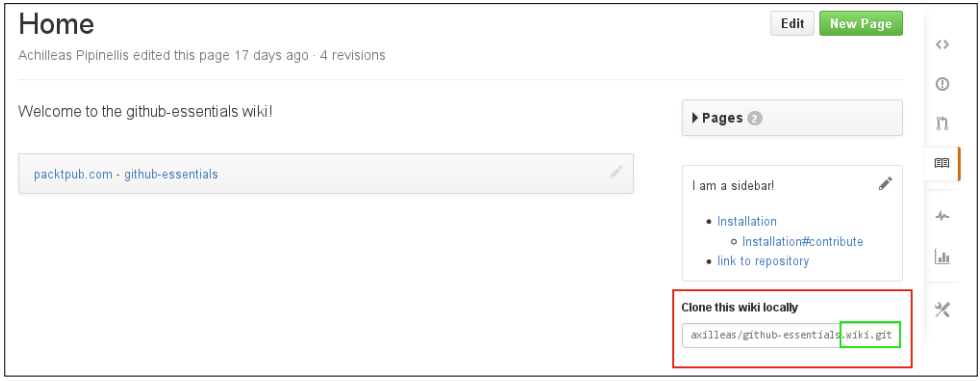
It is powered by the `gollum` ruby library that we will install and use to preview the wiki locally.

Installing gollum

The `gollum` library is packaged as a ruby gem, and the easiest and quickest way to install it is to follow the official wiki entry at `https://github.com/gollum/gollum/wiki/Installation`. You can probably avoid installing it system-wide, but this is not the scope of this guide.

Cloning the wiki and see the preview in your browser

Back in our wiki page, you should have noticed the download link. Every wiki repository has a remote URL encapsulated in green as shown in the following screenshot; essentially, it is the URL of the main git repository with `.wiki` appended to the URL:



www.it-ebooks.info

Use this URL and clone the wiki; then, run the `gollum` command inside that repository:

```
git clone https://github.com/axilleas/github-essentials.wiki.git
cd github-essentials.wiki
gollum
```

Although not stated, you can also clone the wiki using the git protocol and not HTTP.

If you see an output similar to the following, then `gollum` will successfully run and you can preview the wiki in your browser at `0.0.0.0:4567`:

```
[2015-07-26 01:09:34] INFO  WEBrick 1.3.1
[2015-07-26 01:09:34] INFO  ruby 2.1.6 (2015-04-13) [x86_64-linux]
== Sinatra (v1.4.6) has taken the stage on 4567 for development with
backup from WEBrick
[2015-07-26 01:09:34] INFO  WEBrick::HTTPServer#start: pid=20826
port=4567
```

The interface should be familiar with the GitHub wiki. Let's make a few changes.

Making changes locally and pushing to GitHub

Now that you have a running instance of the wiki, you can make changes in the browser much like in GitHub, or use an editor such as `vim` or `emacs` and edit the files directly.

Since you already know how to edit the wiki in the browser, let's use an editor and change the `Installation.md` file. After the edit, save the file and commit it to Git. Take a second to see the log with `git log` and compare it with the history of the commits in GitHub (at `https://github.com/<username>/<repository>/wiki/_history`).

Now push the changes back to GitHub and visit the history page again. The new commit should be there along with the new change.

If you want to write in a different markup language, other than Markdown, see the readme at `https://github.com/gollum/gollum#installation` for ways to install the gems needed.

www.it-ebooks.info

Summary

In this chapter, you learned the importance of documentation and how GitHub allows you to host a Markdown-powered wiki, along with every project. Creating, deleting, editing, and reverting pages should by now be familiar terms.

What is the connection between releases and tags, you ask? Well, if you read the second part of this chapter, you should already know what connects them and how to create releases and distribute them to the public.

In the next chapter, you will see the management of organizations and teams. Read on and learn how to harness the power of collaboration.

www.it-ebooks.info

3
Managing Organizations and Teams

It is important to know when to host a project under your namespace and when under an organization. With the organization, you have the ability to create teams and provide different access levels to people in the various repositories that are hosted under it.

In this chapter, we will go through creating an organization, inviting people, and granting them access to the repositories that are hosted under the organization. You will learn how to create teams and associate members of your organization to them as well as with the repositories.

This chapter was written with the new improved organization permissions in mind. It is a pack of new features and was first announced in June 2015. For more information, read the blog post at `https://github.com/blog/2020-improved-organization-permissions` and the relevant page at `https://github.com/orgs/improved-permissions`.

The difference between users and organizations

Apart from your user account that should be used only by yourself, GitHub provides the ability to create organizations managed by many users and, as we will see later, create teams within the organization.

www.it-ebooks.info

GitHub is a collaborative place and as such, projects with high contribution traffic need a handful of people to help with the maintenance.

This might not be the only reason why one should create an organization, though. Leaving aside the practical reasons, an organization is usually created when there is more than one person, each having equal rights to the projects that the organization will host.

You can see, for example, big names such as Twitter, Google, or even GitHub itself that have organizations under which dozens of projects are hosted.

Organization roles and repository permission levels

GitHub allows you to choose among three roles for a person in an organization: owners, members, and billing managers. We will not deal with the latter; if you want more information on this, refer to `https://help.github.com/articles/adding-a-billing-manager-to-your-organization/`.

Owners have full access to the organization and are in the highest level of the permissions chain. As far as the organization is concerned, they can invite and remove people, create and remove teams, create and remove repositories, as well as manage the permission levels of all people and repositories. They can also edit the organization settings.

Being a member is usually the default role when a new person gets in the organization. The least a member can do is create a new team and add existing team members and repositories to it.

The access level a member has over a repository can only be set by an owner.

A member can also be promoted to "Team maintainer" for a particular team. With this access level, they can add and remove team members. Consider it like an extra hidden role on top of being a member. Anyone who creates a new team is granted the "Team maintainer" role for that team.

www.it-ebooks.info

Now, be careful not to mix organization permissions with repository permissions. There are four kinds of permissions a repository can have: owner, admin, write, and read.

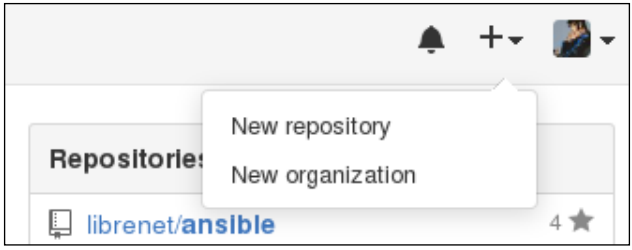
Owner is the top universal permission level that is granted to organization owners. With the **Admin** access, one has owner privileges but for the particular repository; you can push to it, delete it, add or remove a team, change the team's permission level, add outside collaborators, and so on. It is like creating a new repository under your personal namespace. With the **Write** access, you can push to the repository; **Read** grants you read permission which means clone and pull only.

For a comprehensive list of the various level permissions, check out GitHub's documentation at `https://help.github.com/articles/repository-permission-levels-for-an-organization-early-access-program/`.

Now that we have discern the different access levels and permissions, let's create our first organization.

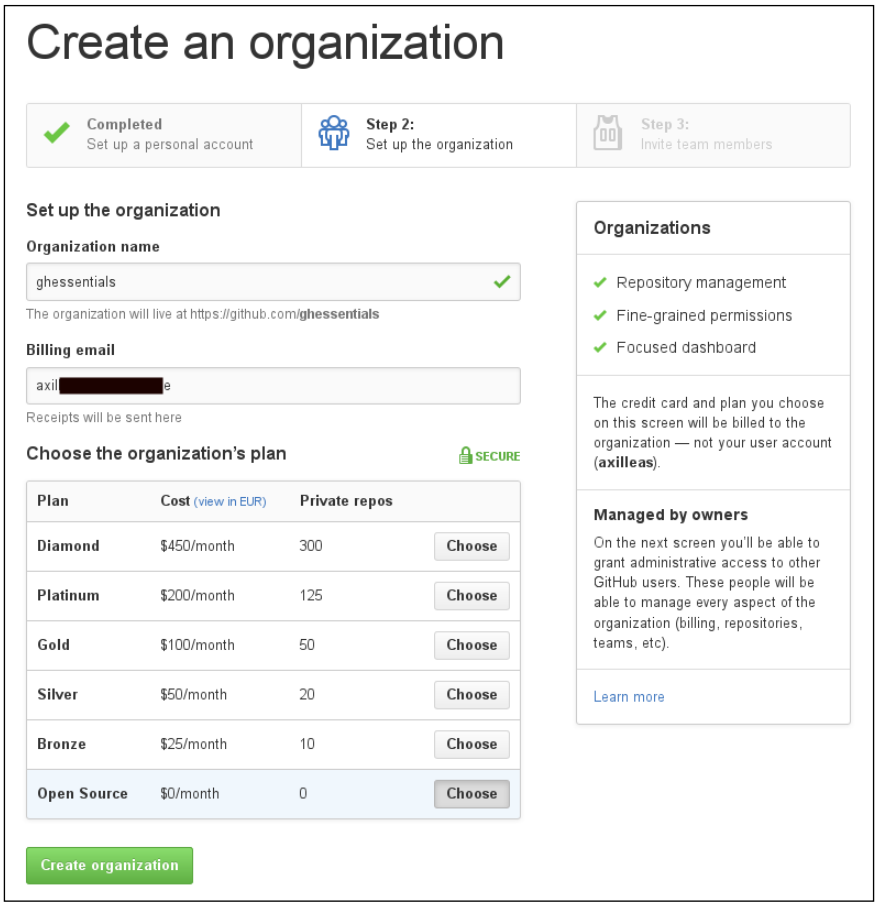
Creating an organization

In order to create an organization, find the cross button at the top header next to your avatar or visit `https://github.com/organizations/new` directly:



www.it-ebooks.info

On the next screen, pick up an organization name and fill in a billing e-mail. For testing purposes, you can give your personal e-mail, which you can change later if you want. For open source projects, the creation of an organization is free, which is the default plan. All these options are summarized in the following screenshot:

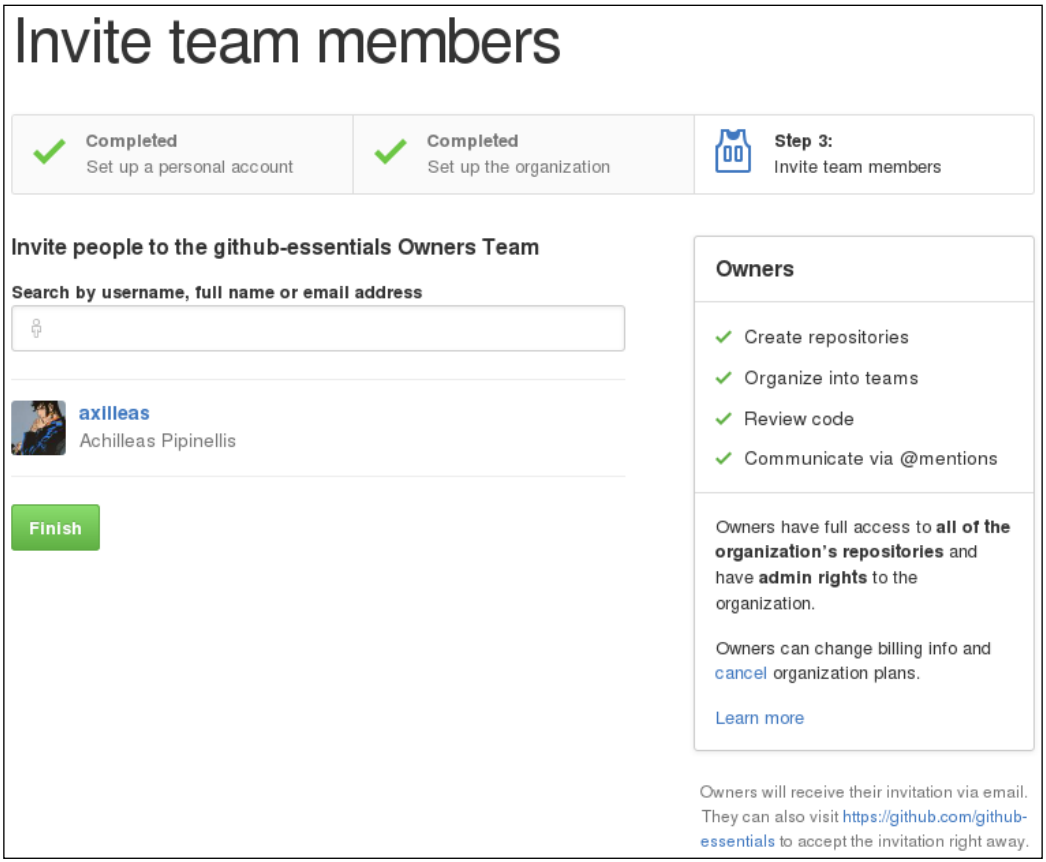


As you type the name, GitHub searches behind the scenes if it is already taken, and if that is the case, a message appears saying that **Username is already taken**.

www.it-ebooks.info

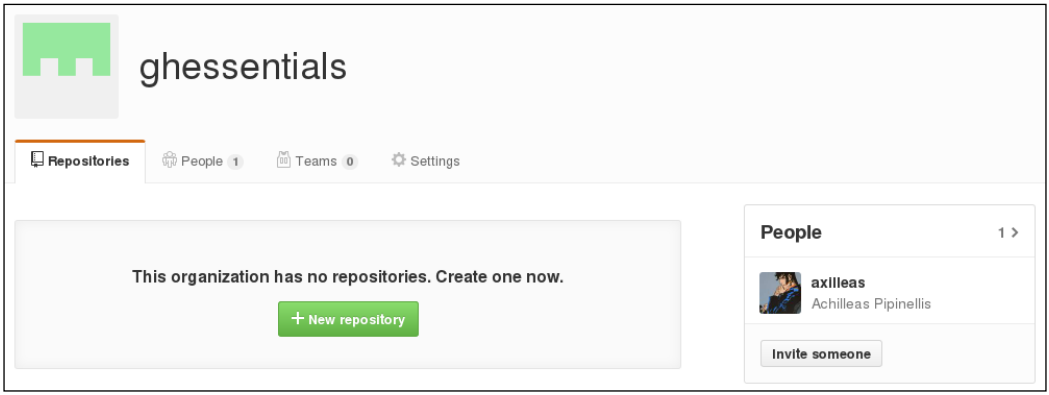
As you will notice, there cannot be a user and an organization with the same name. Namespaces must be unique.

In the next step, you can optionally invite some people to be part of the organization, but let's skip it for the time being and just hit **Finish**:



www.it-ebooks.info

The first thing you will see after the creation is your organization's dashboard, as shown in the following screenshot:

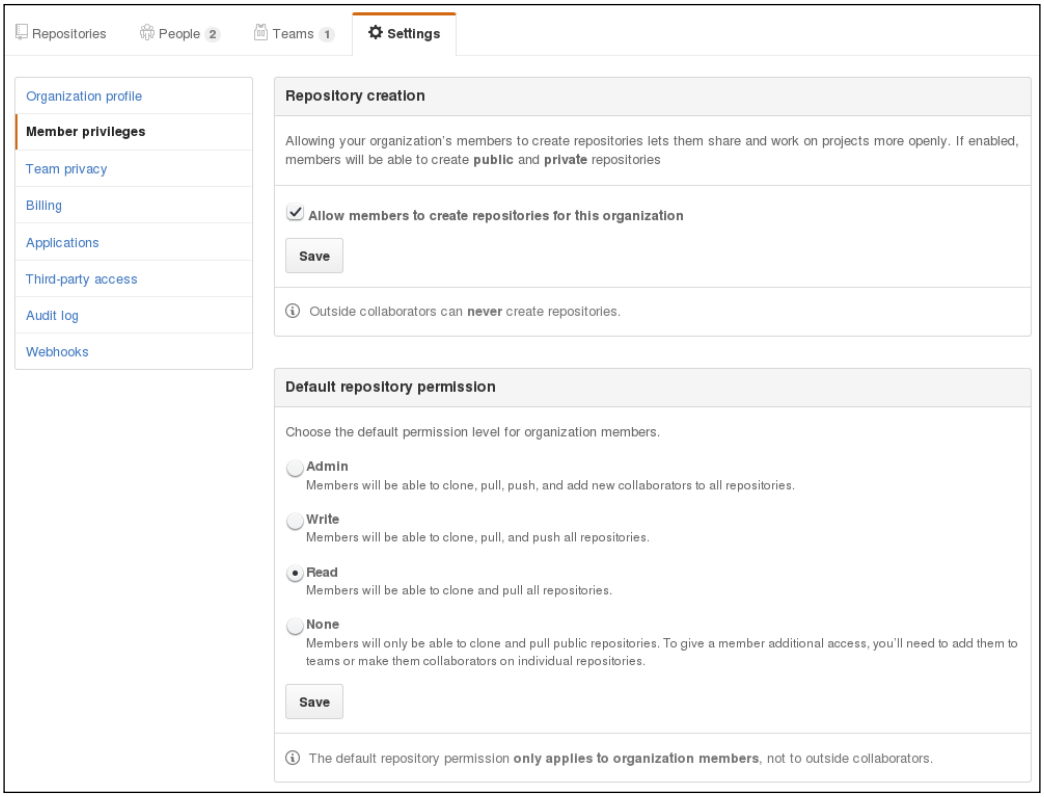


Before we dive into teams, people, and repositories, let's first check out some defaults by setting member privileges globally in the organization settings.

Global member privileges

Prior to inviting people and creating any repositories, let's examine two important settings and set some defaults. Head over to the settings page and select the **Member privileges** tab:

www.it-ebooks.info



As you can see, there are two settings. The first is about repository creation, and if enabled, any member of the organization will be able to create repositories under the organization namespace.

Enable this if you want to work more openly and disable it to be more strict.
By disabling it, only owners will be able to create repositories. Outside collaborators (see the *Difference between Members and Outside collaborators* section) will not be able to create any repositories regardless of this option.

www.it-ebooks.info

The other option is regarding the default repository permissions that organization members have on all repositories, new or old. You can choose from four options: **None**, **Read**, **Write**, and **Admin**. The **None** option is the least privileged one and it means that members can clone and pull public repositories only. With the **Read** access, a member can clone and pull every repository, public or private. The **Write** access means that apart from read access, members can also push to the repositories. Lastly, the **Admin** access will give every member the ability to pull and push to a repository as well as change its settings.

There is one caveat here. If you set this option to **Admin**, then all the members of the organization will automatically have the same highest privileges one can have upon a repository. This automatically nullifies the existence of teams, meaning that since everyone will have permission to do everything, there is no purpose for setting teams and their access levels to repositories.

The bottom line is, do not set this option higher than **Write** unless you plan not to use teams and want every member in the organization to have equal rights to repositories.

Throughout the rest of the chapter, the examples will be based on members being able to create repositories, and the default repository permission level will be **Read** unless otherwise noted.

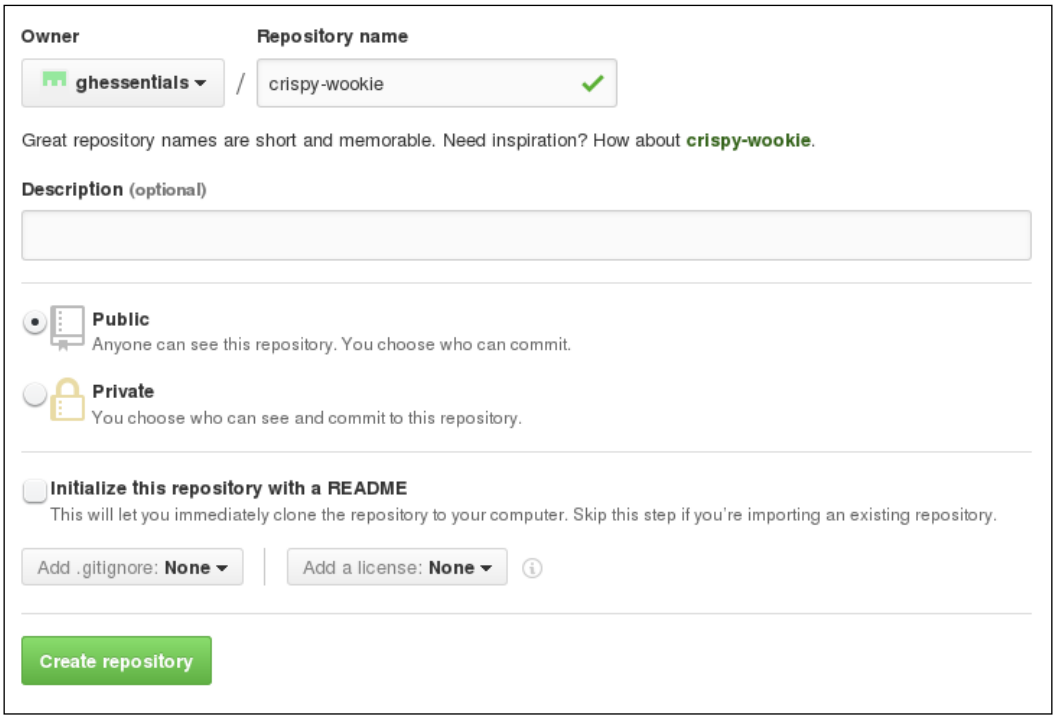
We will explore the rest of the settings later in this chapter.

Repositories

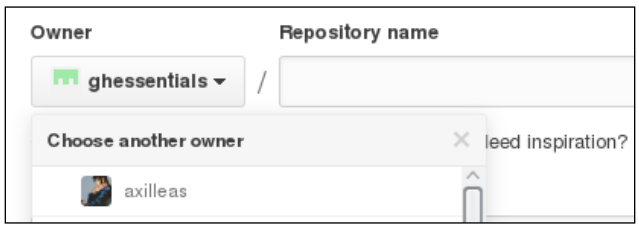
So, the first tab on your new dashboard is **Repositories**, and since there is none at that time, GitHub urges you to create one.

www.it-ebooks.info

Once you hit the **New repository** button, you will be taken to a familiar page, as shown in the following screenshot:



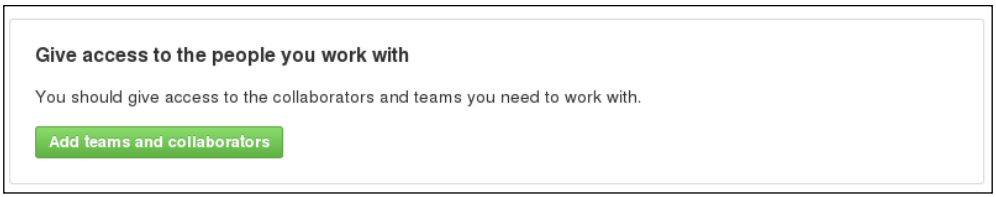
If you read *Chapter 1*, *Brief Repository Overview and Usage of the Issue Tracker*, you will notice that the only thing that changes when creating a repository is the namespace. If I wanted, I could have created the repository under my username by choosing it from the drop-down menu:



www.it-ebooks.info

Now that the repository is created, you can upload the code from your computer and start working on it.

You might have noticed in the repository's landing page when it was first created, GitHub has a message to add teams and collaborators:



If you want to grant access to certain people immediately, then you should follow that route. For our purposes, given this is a new organization, we must first learn about teams and their differences with outside collaborators as well as the different permissions on repositories.

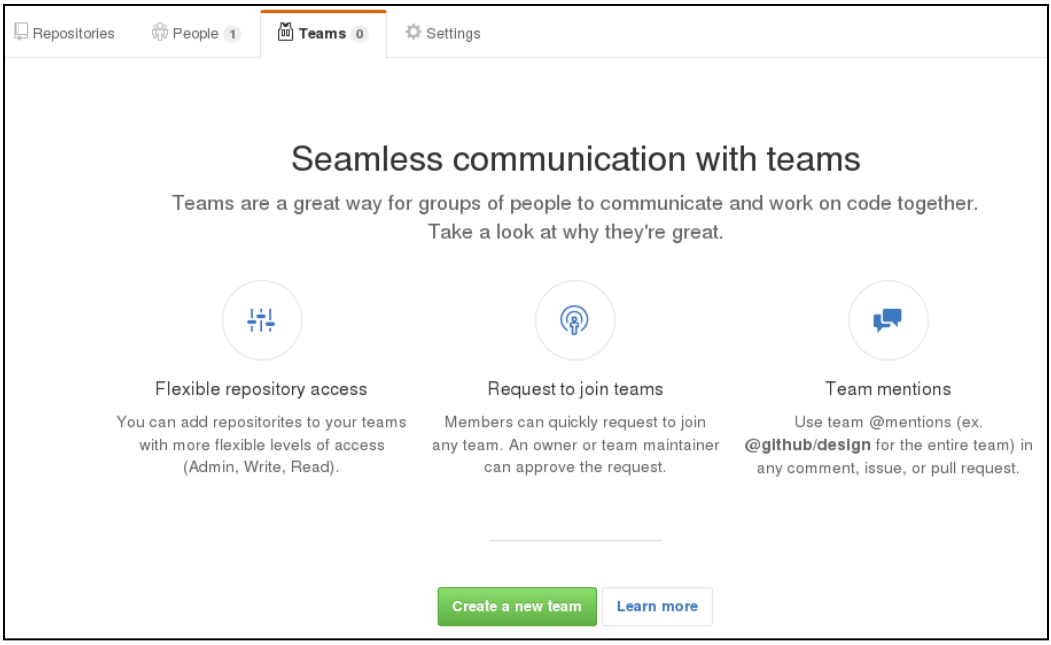
Teams – a great way to grant selective access to your organization projects

Team is how you control different access levels in your repositories. Next, we will see how to create a team and add members to it.

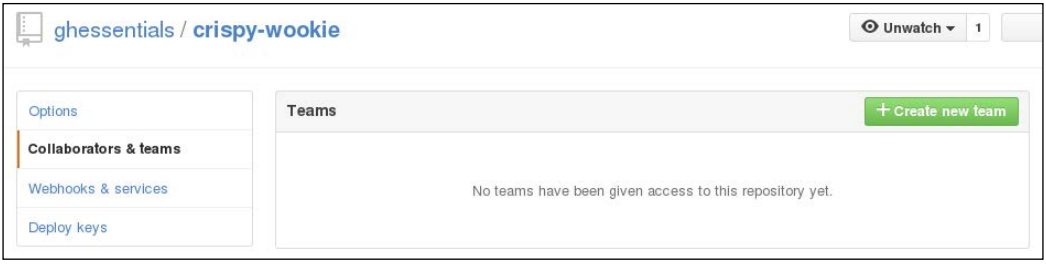
Creating a team

As with most cases in GitHub, you can create a team in different ways. The apparent way is to head over to the **Teams** tab and create a new team:

www.it-ebooks.info



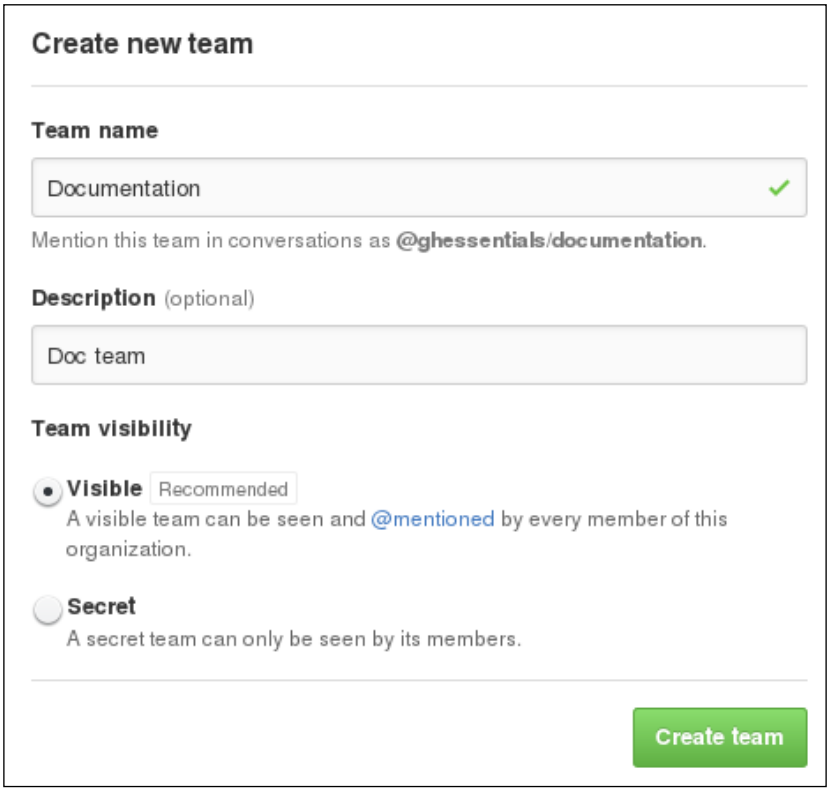
The other way is to head over the settings of a repository and under the **Collaborators and teams** tab, hit the **Create new team** button:



Notice that only a repository that lives under an organization namespace will have the **Teams** option. If you edit a personal project, you can only see the **Collaborators** box.

www.it-ebooks.info

When you first create a new team, you will be presented with the following form:

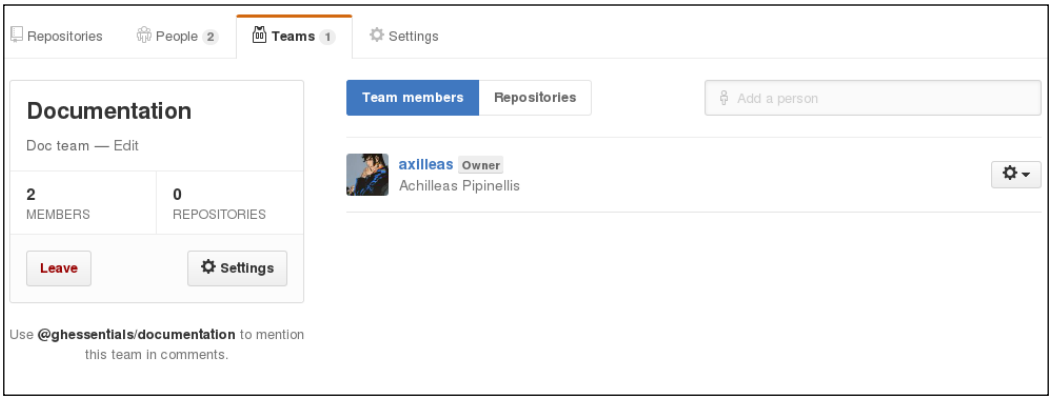


The team name is mandatory and the action is two-fold. You can enter a human-readable text with punctuation and capitalization, but notice that the name that will appear in the URL is converted to lowercase. For example, `GitHub Core` will be `github-core`. Any special character is stripped and converted to a dash (`-`).

You can put an optional description and then choose whether this team will be publicly visible or secret. A secret team will only be visible by its members and the owners.

www.it-ebooks.info

After the team is created, you will be taken to the team's page where you can see who the team members are and what repositories this team has access to. Also, you will be able to edit the team's settings, such as change its name or even delete it:

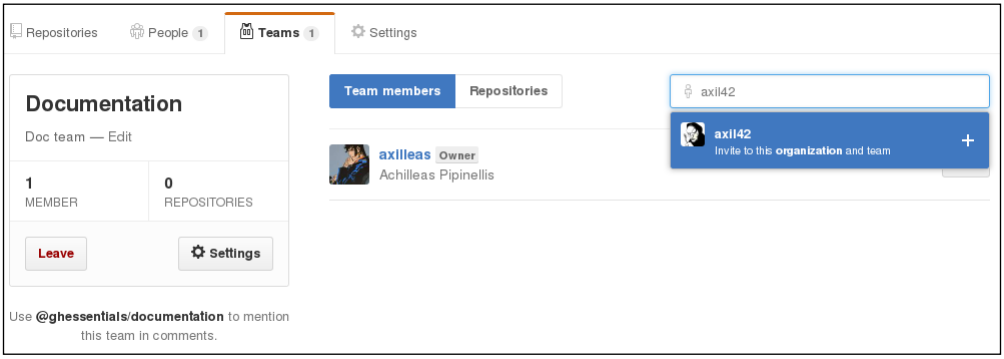


Inviting people

The whole point of having teams is to have people in them. So far, you were the only member of the organization; let's invite someone to join the team.

You can only invite people that are already registered GitHub users.

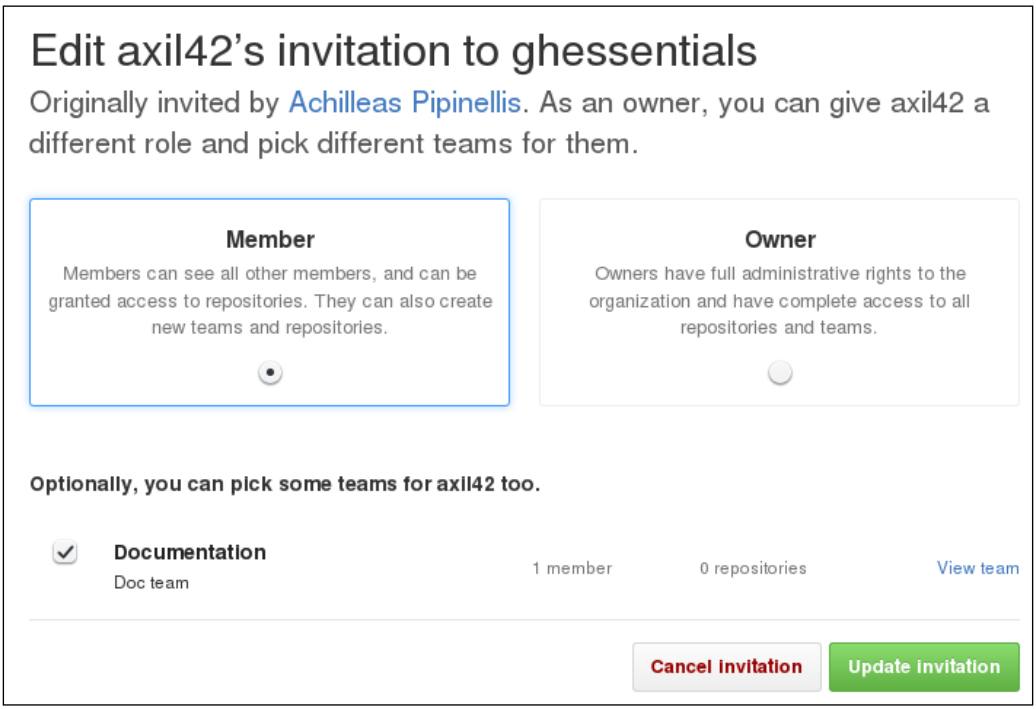
Head over to the **Teams** tab and select the team you want to invite someone to. On the right-hand side, you can see a search bar with a placeholder text **Add a person**. Start typing the username and GitHub will sort out until it finds the right one:



www.it-ebooks.info

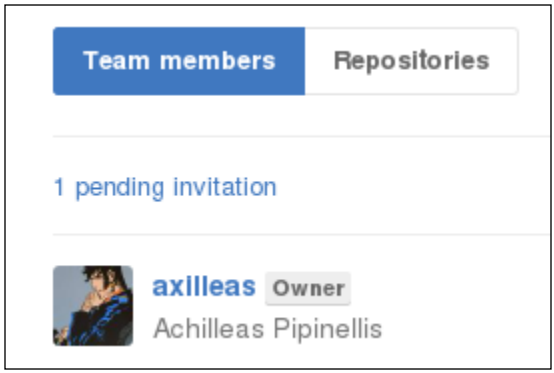
After you select the person, you can see that GitHub tells us that this person is invited to the team and as a member to the organization:

By default, the role one has when invited to a team is **Member**. If you want to change this to **Owner**, you can edit the invitation, change the role, and optionally assign different teams. You can cancel the invitation as well:

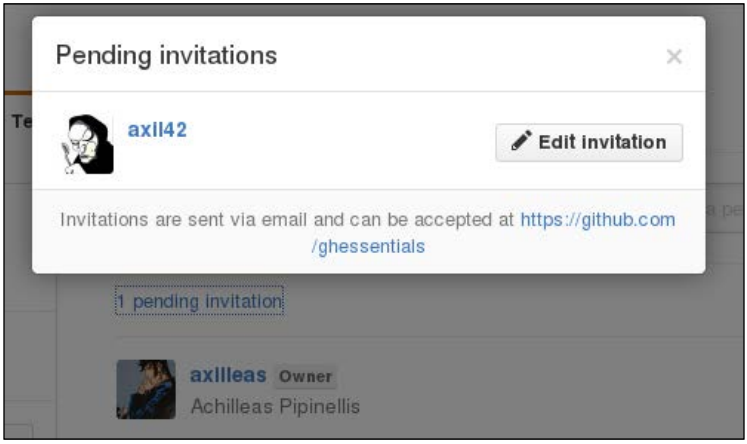


www.it-ebooks.info

Let's not do anything and just return to the previous page. You will notice that the previous message is gone and GitHub now shows how many pending invitations there are, as shown in the following screenshot:



If you click on the **1 pending invitation** link, a modal will be revealed where you can edit the invitation as we saw previously:



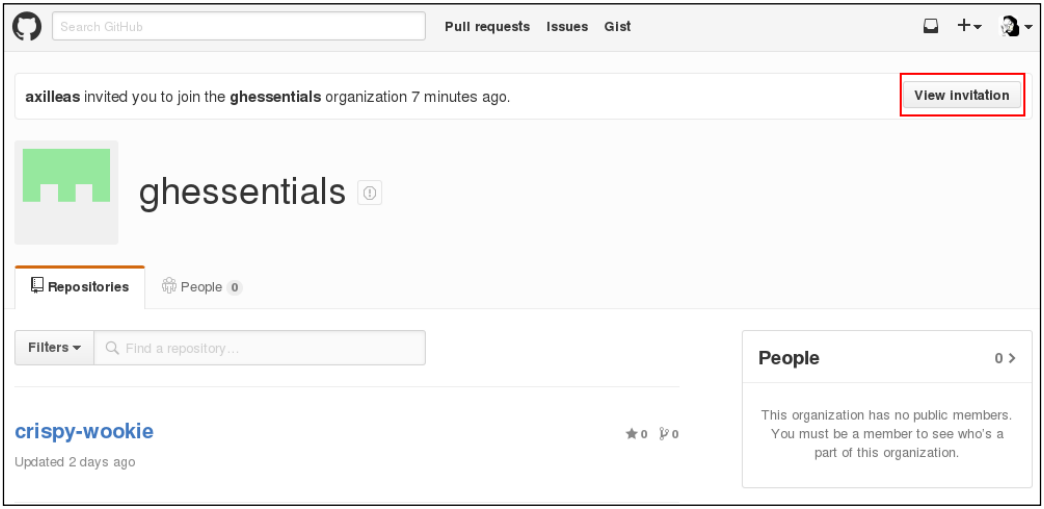
Only owners can invite new people to a team or the organization. A person has to be an organization member, for team maintainers to be able to add them to teams.

www.it-ebooks.info

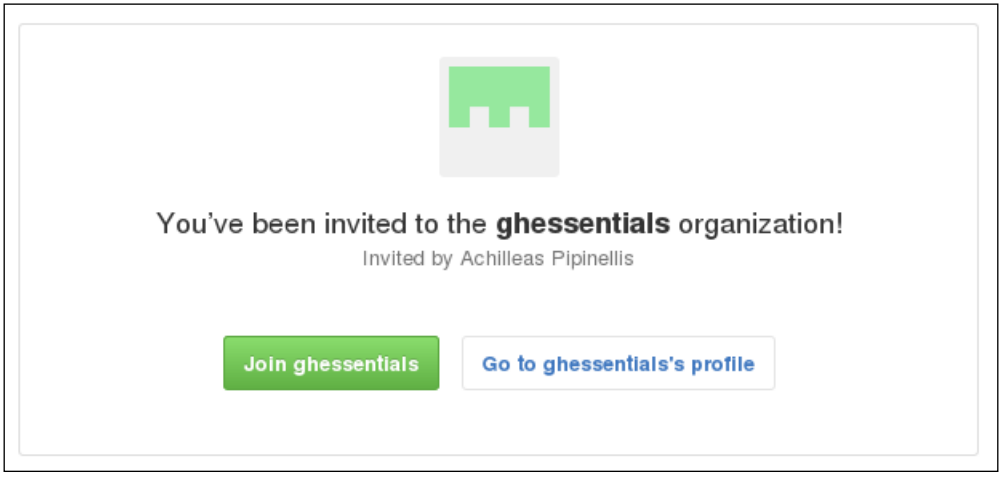
Accepting an invitation

GitHub will send out an e-mail notifying the invited person about the invitation. In it, there is a link to follow that lands you to the organization page where you can accept the invitation or you can just ignore the e-mail and visit the organization page directly.

Here I visited the organization's page as the person who got invited and as you can see I am presented with a message to join the organization:

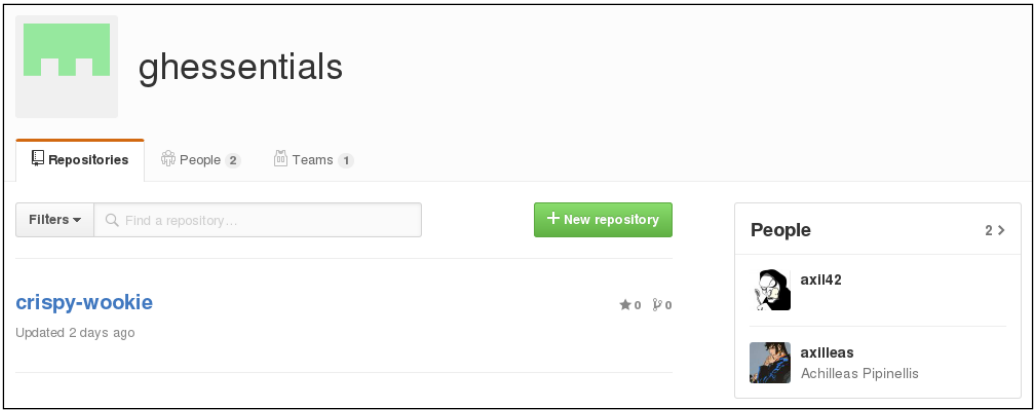


Either way, the final page where you will decide whether you will be joining the organization is as follows:



www.it-ebooks.info

Let's accept the invitation and start working as part of the organization. Now, you can see who the people are, the repositories, and the teams:



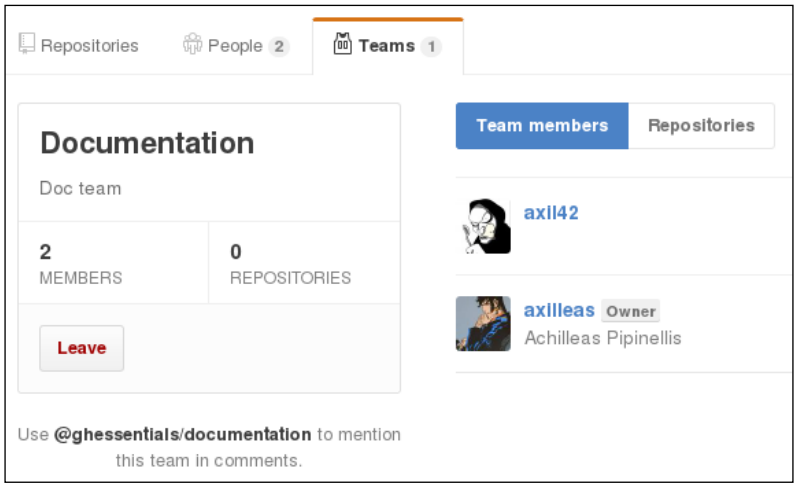
Team members permissions

In the team's page, you can also change a member's team access with the gear icon on the right. You have three options: promote a member to team maintainer (or demote to plain team member), convert a member to outside collaborator (we will see what that means in a bit), or remove a member from the team:

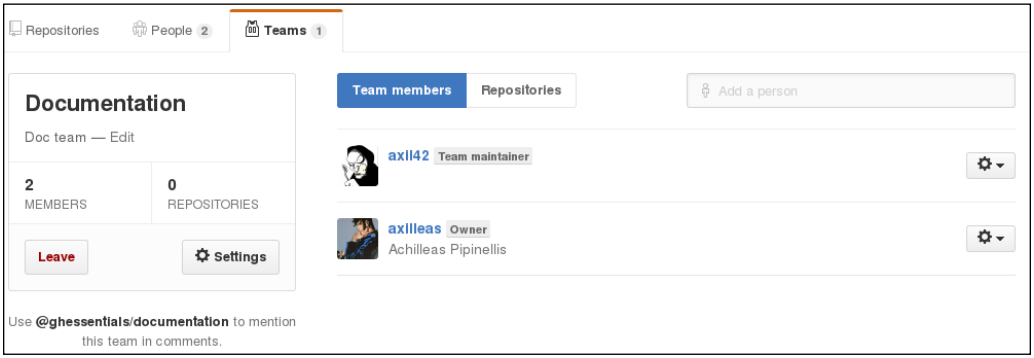
As an owner, I have the power to perform any of the preceding options. Notice that the drop-down menu is different for owners and team maintainers.

www.it-ebooks.info

Before we promote a member to team maintainer, here is what the member sees when in the team dashboard:



After the owner promotes them to team maintainer, they will have more privileges, as you can see in the following screenshot:



A team maintainer can now add or remove team members, edit the team's settings, and even delete it. They cannot, however, convert a member to an outside collaborator.

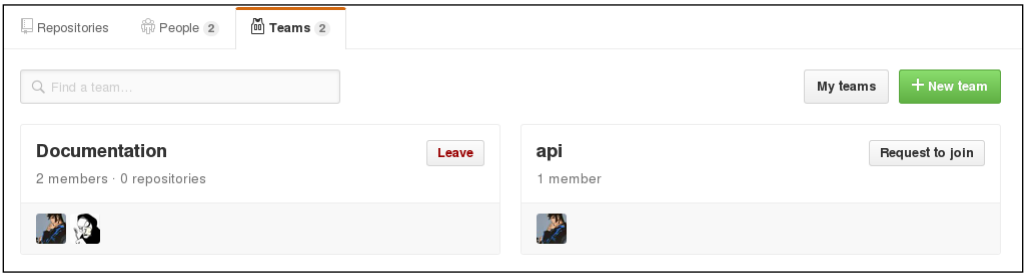
www.it-ebooks.info

Request to join a team

There may be occasions where a member wants to get in another team that has more privileges on a repository. If you are already a member of an organization, you can ask a member to join an existing team. Let's break it down in steps and see how this is done.

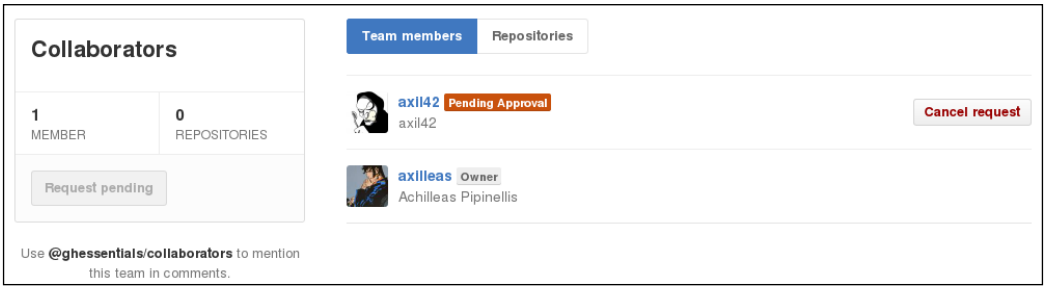
Step 1 – as a user

Head over to the **Teams** tab and you will notice a **Request to join** button in the teams you are not a member of yet:



Step 2 – as a user

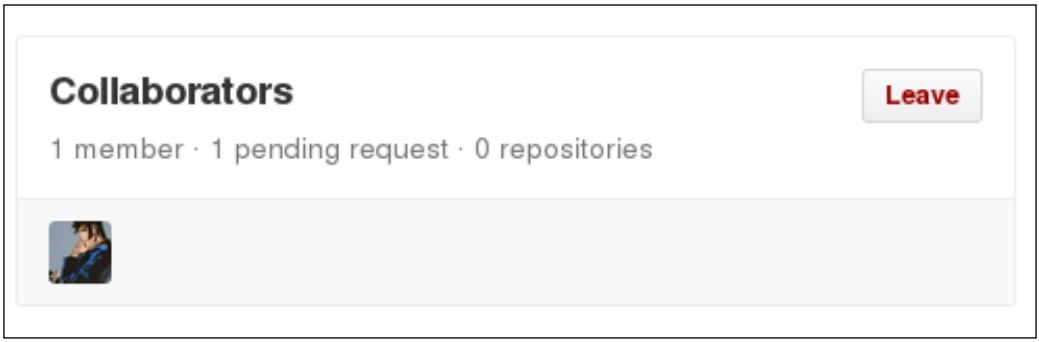
Once you request to join the team, an owner or a team maintainer will have to accept your request. If you now visit the team, you can see your pending request:



www.it-ebooks.info

Step 3 – as an owner or team maintainer

By the time you request to join a team, an e-mail notifies the owners and team maintainers about your request to join that team. As an owner or team maintainer, by visiting the **Team** tab, you can see that there is a pending request:



Go into this team and accept or reject the request:

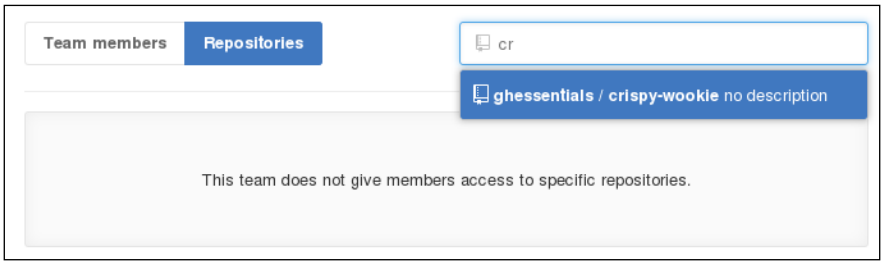
Adding repositories to a team

It's now time to add a repository to a team and explore the privileges of the team upon it. Remember that only owners can perform this action for all teams and team maintainers only for the team that they are maintainers of.

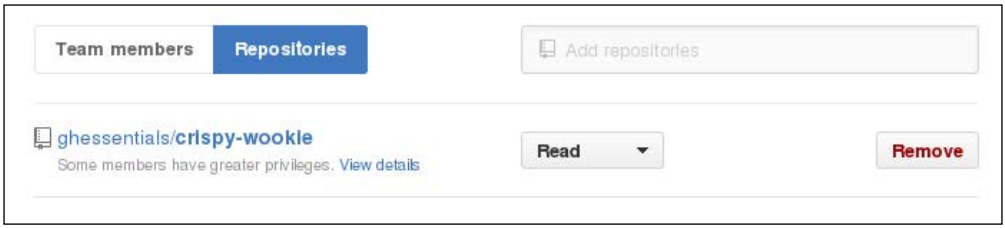
There are two ways to pair a team with a repository and vice versa. The first way is to search for a repository within a team and the second way is to search for a team within a repository.

www.it-ebooks.info

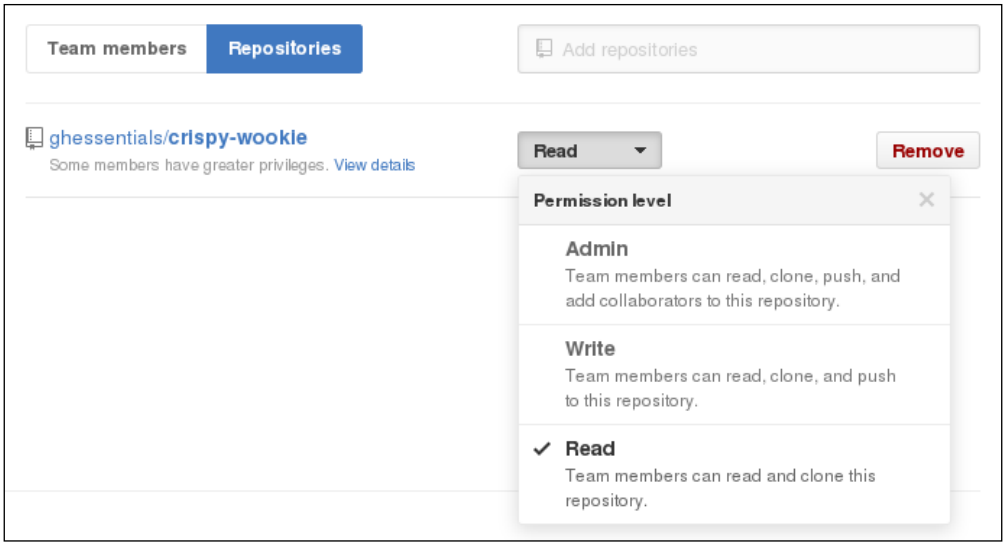
Let's try the first one. Head over the team you want to add repositories to and at the **Repositories** tab, start typing the name of the repository:



When you find what you are looking for, select it:



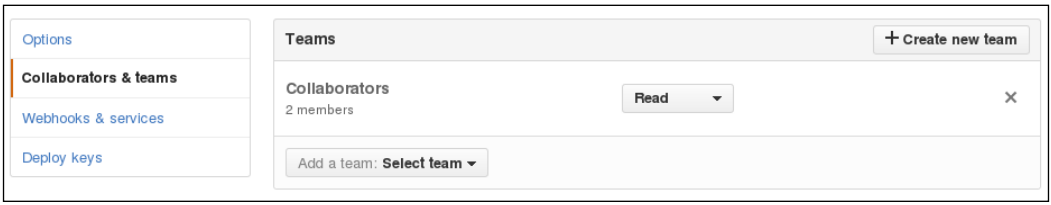
As you can see, the default access to this repository is **Read**. This is the default access level that we have set in the *Global member privileges* section. Regardless of the global option, you can set different permissions on each repository the team has access to:



www.it-ebooks.info

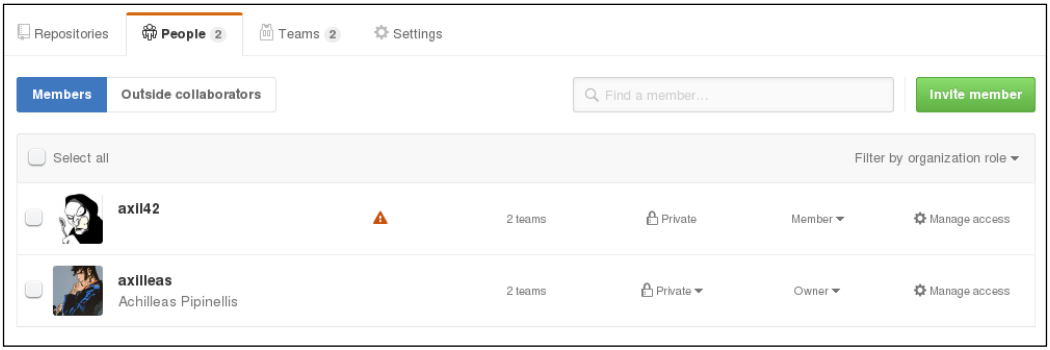
Team members cannot change the repository access level, but can remove the repository from a team.

Now, if you head over the repository's settings under the **Collaborators & teams** tab, you can see the team that is added:



The People tab

The **People** tab is where, as an owner, you can manage the organization members' privileges:



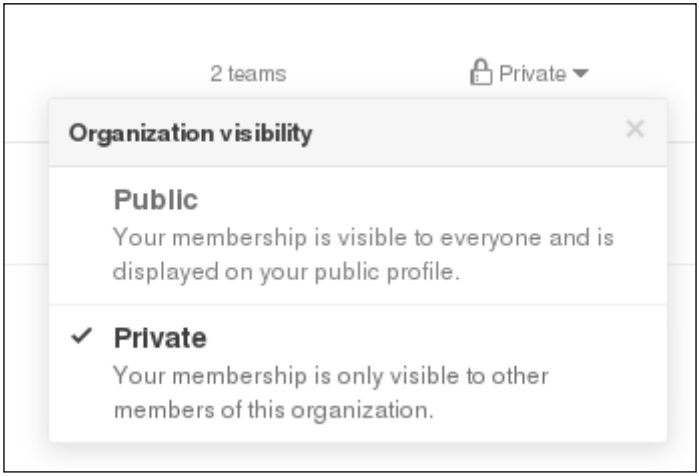
From the preceding image, you can see that, as an owner, you have a higher overview of the members in your organization. Let's examine what all of these settings mean.

The red triangle simply means that a member has not enabled Two-factor Authentication. From a security perspective, you would want every member of the organization to have enabled 2FA to prevent a potential account compromise that would lead to gaining access to the organization's repositories.

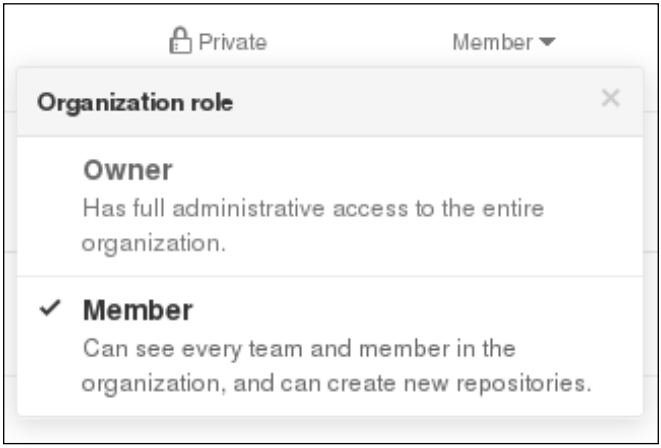
www.it-ebooks.info

Then, you can see in how many teams each member is part of. Clicking on the number will show you the exact teams.

The next thing is the visibility of the organization membership. Each user must set the visibility for themselves. Set it to **Private** to hide your membership and choose **Public** to publicize it. If you publicize it, the organization's avatar will show in your profile. Owners can change the visibility from public to private but not vice versa:



Next we examine the organization roles. Only owners have the ability to change a member's organization role and you can set it to either **Member** or **Owner**:

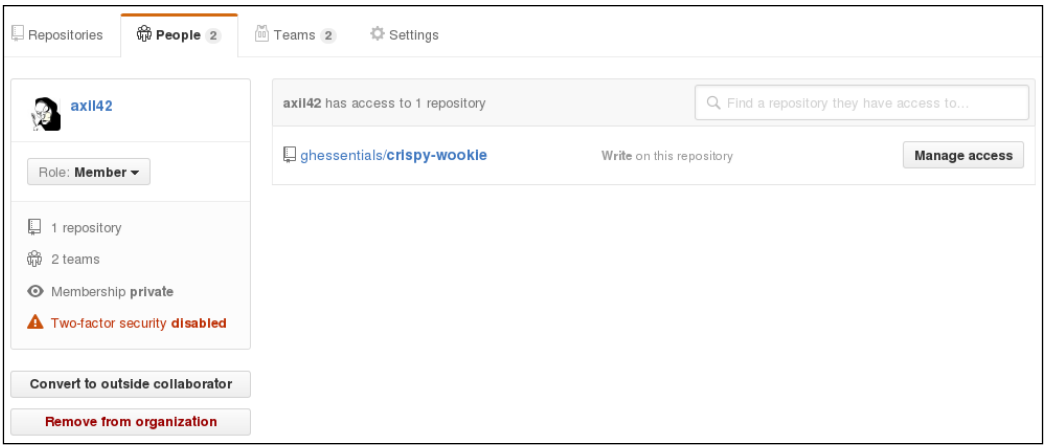


Lastly, there is the **Manage access** button that takes you to an individual's management page. Let's take a good look at it.

www.it-ebooks.info

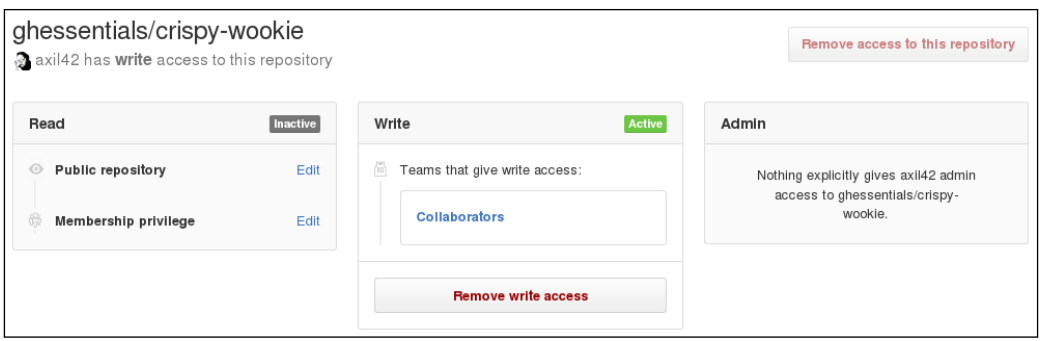
Managing access levels

Selecting **Manage access** will take you to a person's management page:



On the left-hand side box, you can see the same information as shown in the **People** dashboard. From here, you can change the member's role, covert them to outside collaborators, even remove them from the organization, purging every permission they might have on the organization's repositories.

On the right-hand side, you can see all the repositories they have access to and can manage their access level to every single one of them. Let's explore this setting by hitting the **Manage access** button:

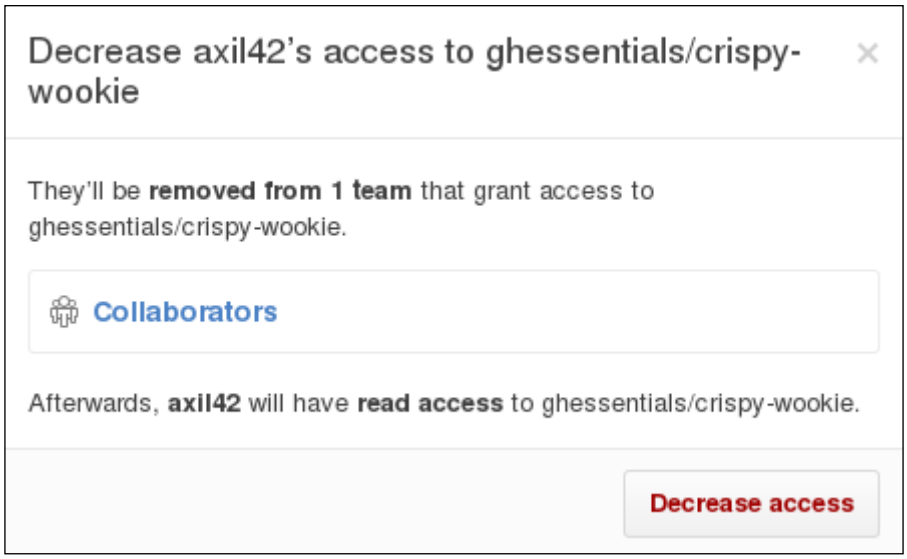


www.it-ebooks.info

According to what you have set in the global settings access, as we saw in the *Global member privileges* section, you can see the access level in the particular repository. Remember that the default access level is set to **Read**, but this person is also a member of a team that has the **Write** access to this repository.

These three boxes depict exactly that. Currently, the **Write** access is active since it is a superset of the global default setting with the **Read** access.

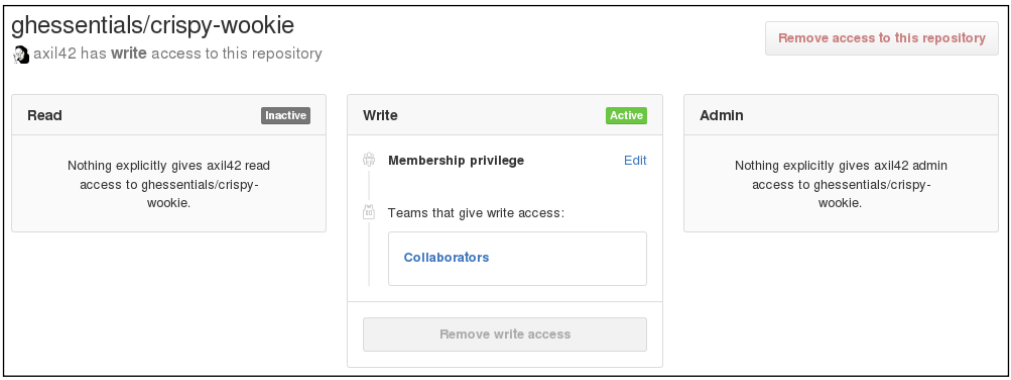
If you choose to **Remove write access** in this repository, this would remove the member from the team that granted them **Write** access in the first place and then they would have the default one that was set globally, which is **Read** in our case:



Likewise, if the team granted **Admin** access to the repository and the global setting was set to **Write**, removing admin access would result in demoting the person to have the **Write** access.

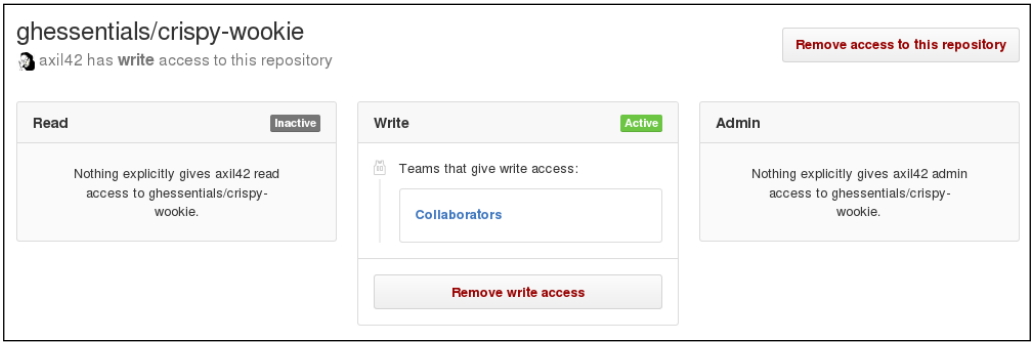
www.it-ebooks.info

Let's make a test and set the global setting to **Write** and see what happens. Click on the **Edit** button of the **Membership privilege** setting under the **Read** box. Then, change **Default repository permission** to **Write**, go back to the previous page, and see how it changed:



Even if the person was not a member of the **Collaborators** team, they would still have the **Write** access to the repository since this is now the global default. Also, notice how **Remove write access** is now grayed out. Since the global default and the access that the team grants are the same (**Write**), there is no point of demoting a member's access level.

Similarly, the **Remove access to this repository** button on the top-right is grayed out. What this does is remove the access completely, which means in order to see this button enabled, you must have set the global repository access permission to **None**. Let's give it a go, as shown in the following screenshot:



www.it-ebooks.info

Now, removing the access to this repository is going to revoke every access a team might have granted to this person.

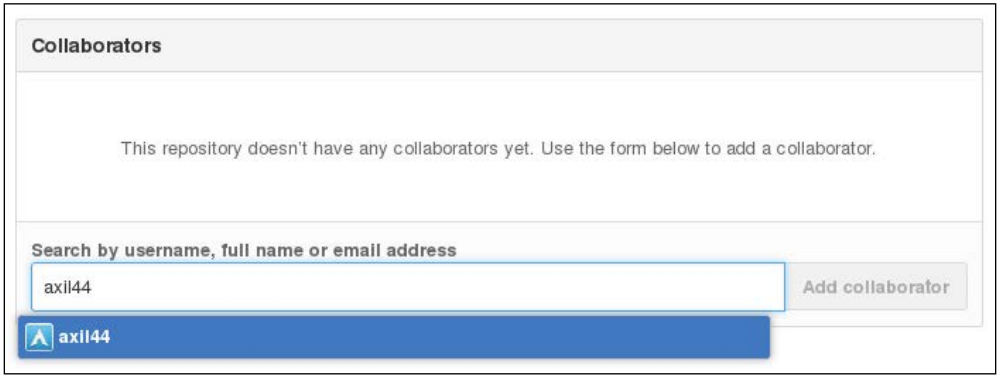
That's all there is to it when it comes to managing a member's access to a repository. Let's go ahead and finally see what these **Outside collaborators** are all about.

Difference between Members and Outside collaborators

As the name suggests, outside collaborators are non-organization members with repository access. Much like you can give write access to another user in your personal repository, you can give them write access to individual organizational repositories without being members.

At present, no one outside our organization has access to a repository. If you visit the **Outside collaborators** tab, GitHub will tell you that there doesn't exist any collaborator yet:

Let's add an outside collaborator by going to a repository's settings in the **Collaborators & teams** tab:

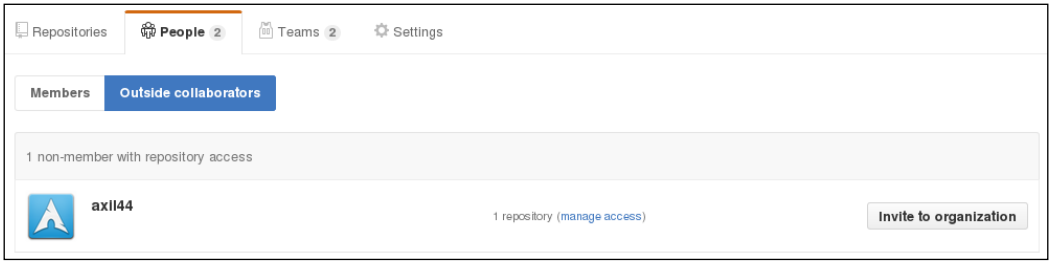


www.it-ebooks.info

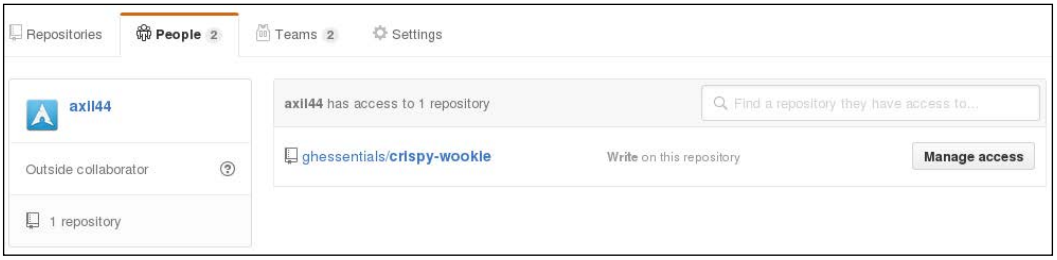
After selecting the user, hit the **Add collaborator** button:

Notice that the default access level is **Write** and not **Read**, which is set globally for the repositories. The default repository permission only applies to organization members, not to outside collaborators.

Now, let's head over the organization **People** page and select **Outside collaborators**:

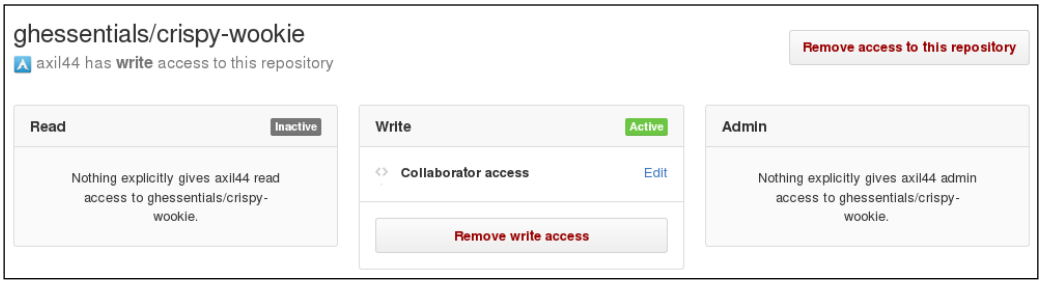


From this page, you can manage a collaborator's access and even invite them to the organization. Let's manage their access for now:



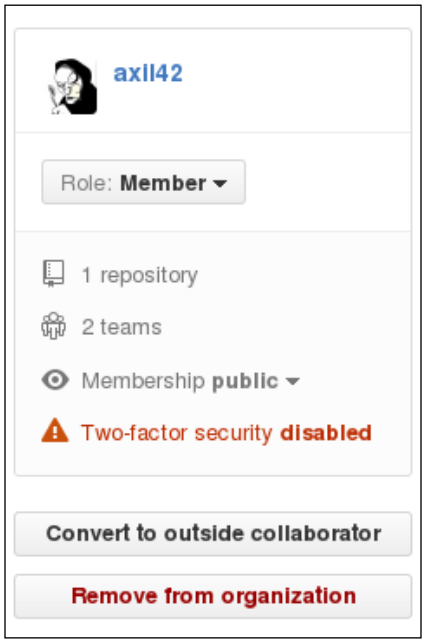
www.it-ebooks.info

This particular user has the **Write** access to one repository, whereas we can see this is an outside collaborator. By going to **Manage access**, you can remove the user's write access to the repository or remove their access altogether:



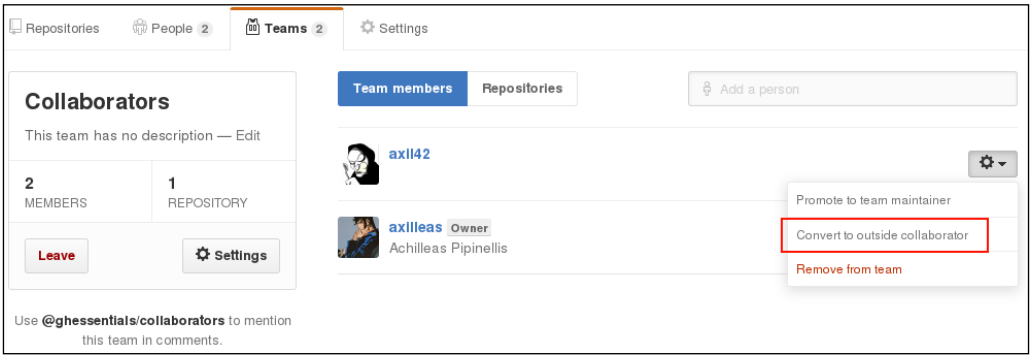
Demoting to an outside collaborator

Just as you can be promoted to a member if invited, you can also be demoted. Remember there are two places where this can happen. The first is in the **People** tab in **Manage access**:



www.it-ebooks.info

The other can be found, when in a team, at the **Team members** tab:



In either case, when converted to an outside collaborator, any repository access level that was given to this person in a team will be retained.

Invite members

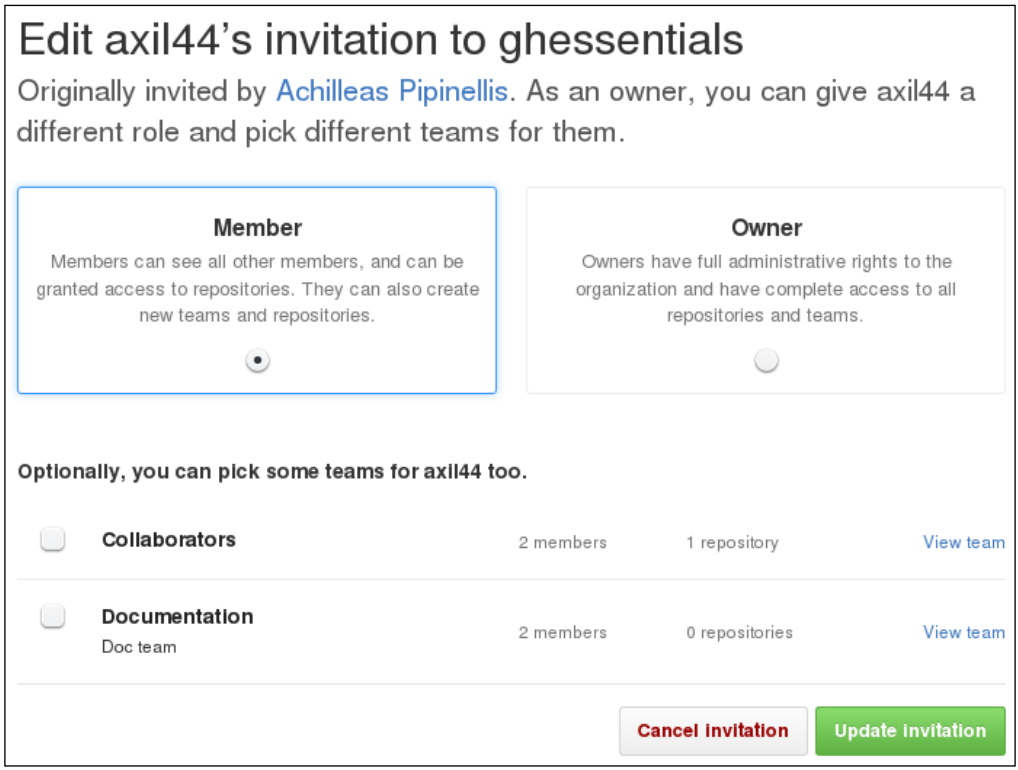
As we saw earlier, only owners can invite new people to the organization. We have already explored that when adding a new person in the team, if that person is not an organization member, they get invited.

Another direct way is to go the **People** tab and hit the **Invite member** green button:

You will then be taken to a search page to enter the username of the person you want to invite:

www.it-ebooks.info

After you select the user, you can set their role as well as add them to any existing teams:



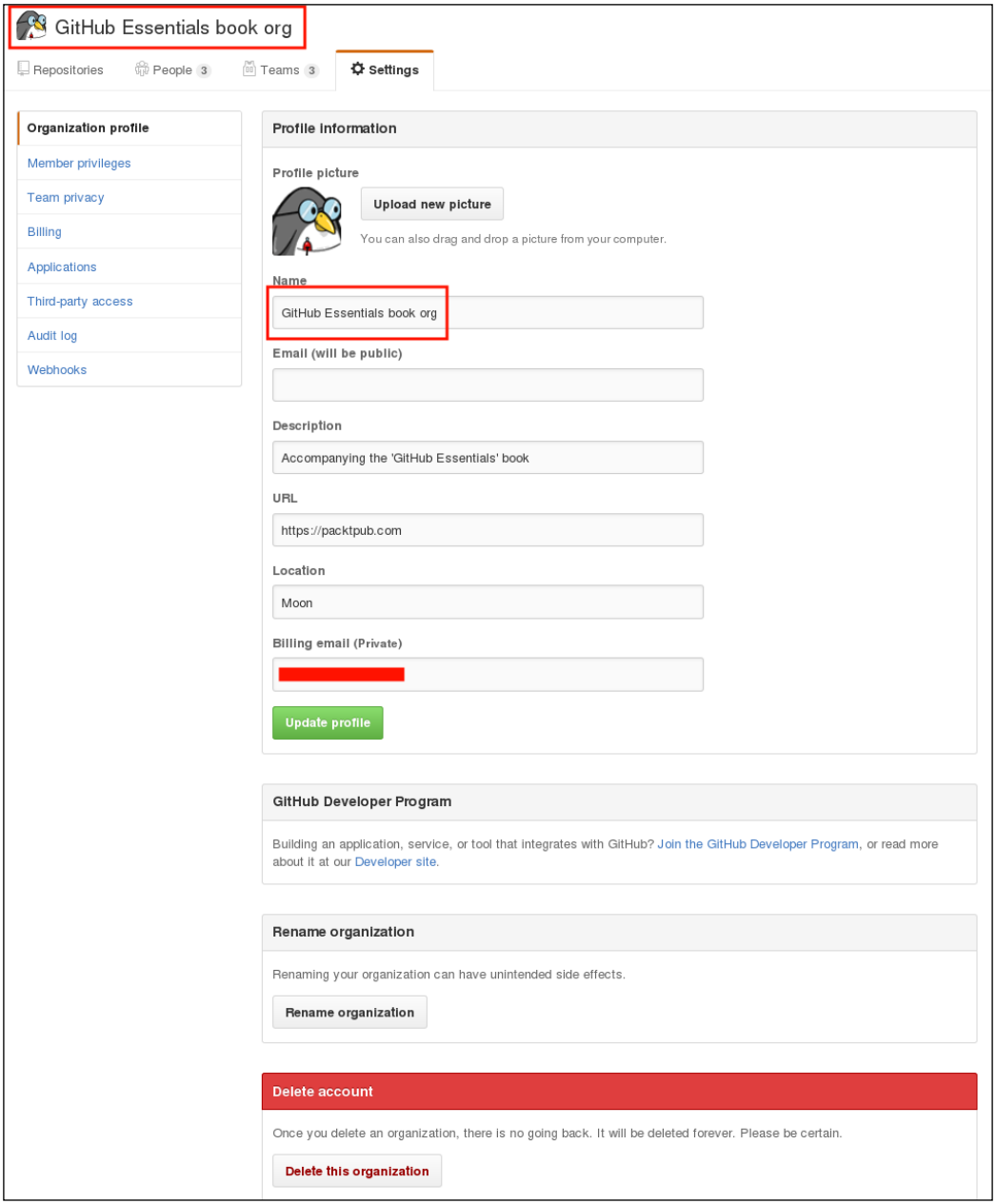
Organization settings

So far, we have only explored the **Member privileges** setting. Let's see the rest of them.

www.it-ebooks.info

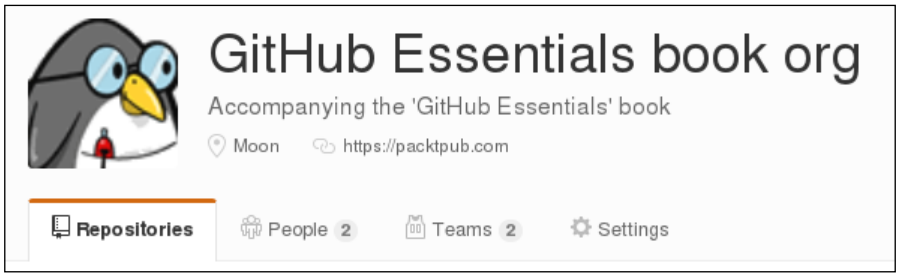
Profile

From the profile page, you can change the organization's name and its description, add a URL and a profile picture, rename the organization namespace, and even delete it:



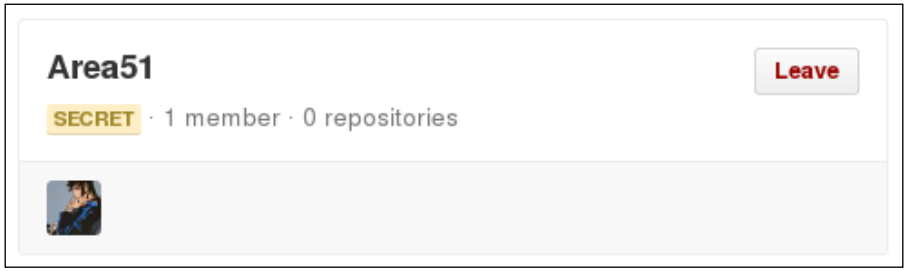
www.it-ebooks.info

After making all of these changes, you can see that the landing page of your organization will be a little bit prettier:

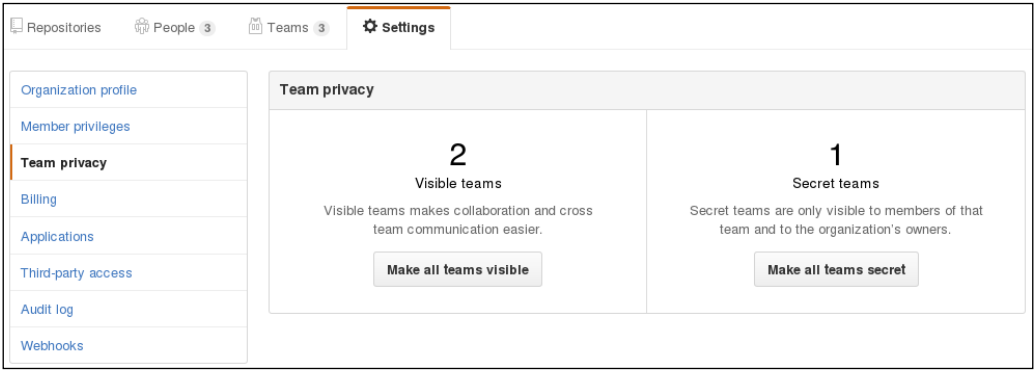


Team privacy

This is a nice feature if you want to turn all teams to private or public at once. For the sake of this example, I created another team; this time **SECRET**. It will be visible only to owners and whoever is that team's member:



The **Team privacy** settings will now look as follows:

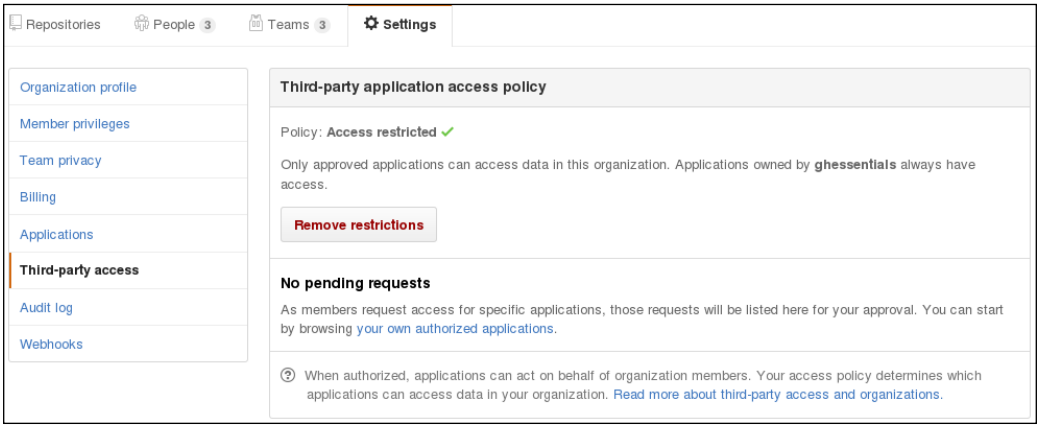


Pick the option you want and convert them all to public or make them secret.

www.it-ebooks.info

The third-party access

The third-party access is an advanced setting. Some applications and many web apps have a way to interact with GitHub and gather information about repositories, teams, user data, and so on. This is accomplished by the feature rich API that GitHub provides:



By default, this is set to be restricted. This means that if as a user you authorize an application, for example, to read all your repositories, it will not have access to the organization ones.

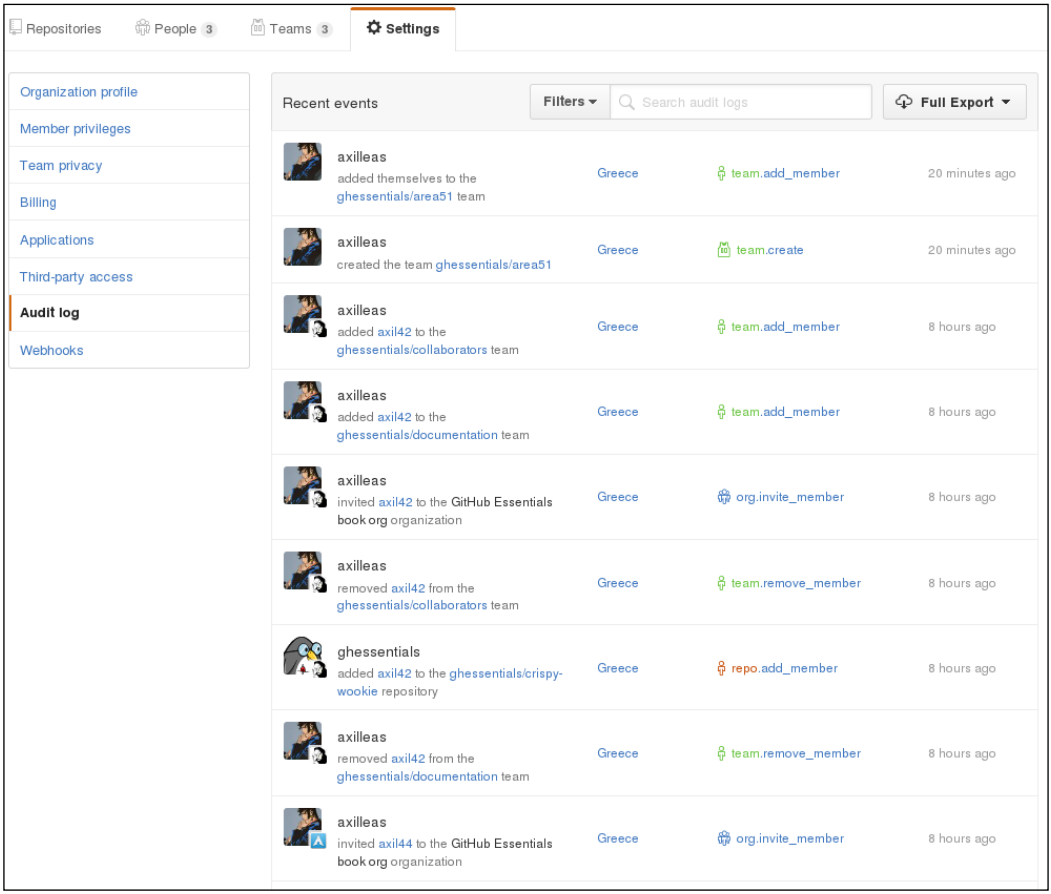
For more information, you can read GitHub's documentation at `https://help.github.com/articles/about-third-party-application-restrictions/`.

Audit log

If you ever worked in a Unix environment, you should know that almost everything that is done in the system gets logged somewhere (usually under `/var/log/`). In the same way, GitHub logs most actions in what it calls audit log.

www.it-ebooks.info

While I was performing all the examples with team creation, membership and repositories, I got quite a big audit log:



You can see what user made what action and the time this was performed. To better explore the log, GitHub provides the option to extract it in CSV or JSON format. You can even use the search function to filter specific events.

www.it-ebooks.info

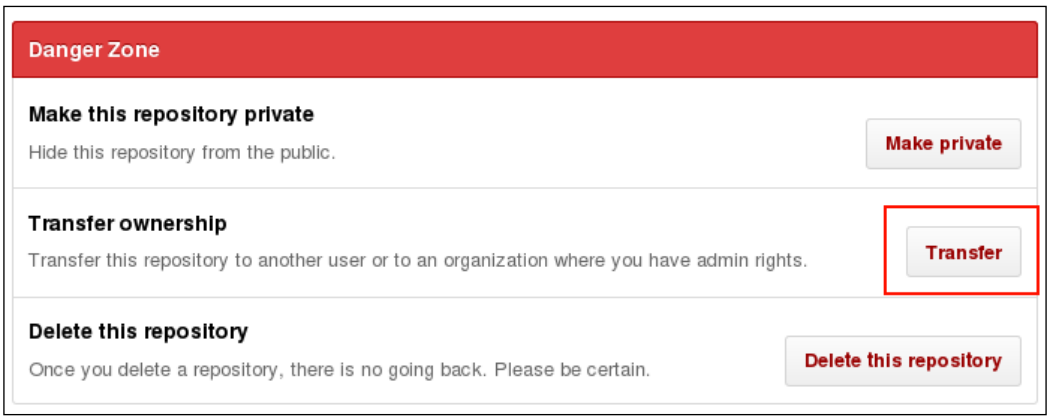
Tips and tricks

Here are some tips and tricks that complement what you have learned so far.

How to transfer a project to an organization's namespace

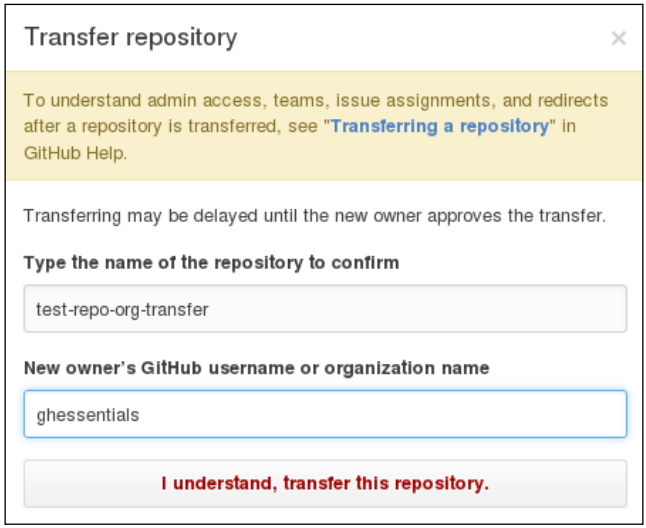
There will be times where a project of yours fits better under the umbrella of an organization. In this case, you can transfer it under the organization's namespace. You are able to transfer repositories to organizations only if you are at least their member.

Head over your project's settings and you will see a **Transfer** button, as shown in the following screenshot:

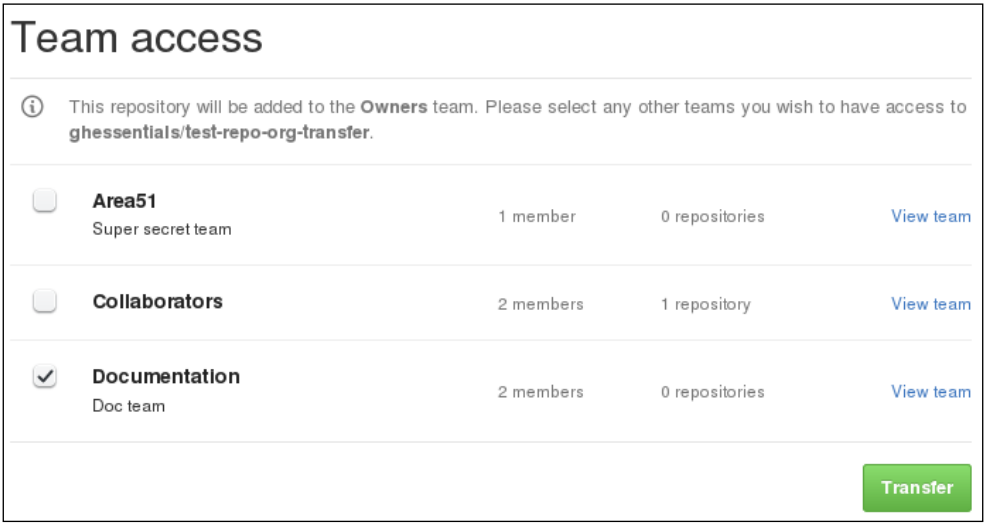


www.it-ebooks.info

A modal will appear where you have to confirm by providing the repository's name and the organization you wish this to be transferred to:

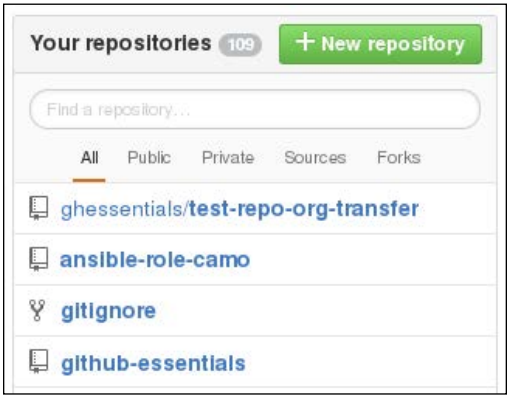


Since I am an owner of the organization, in the next step, I get to choose if I want any other teams to have access to this repository:

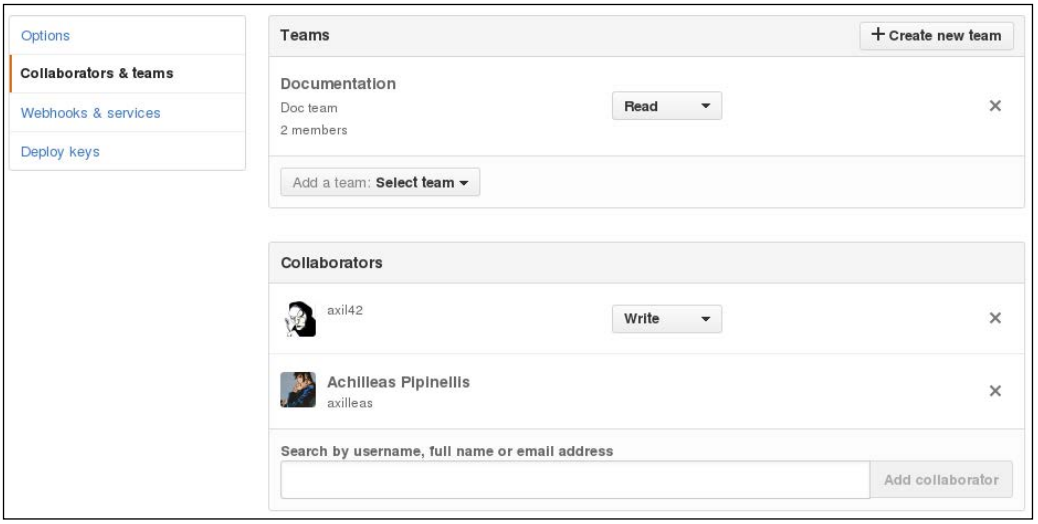


www.it-ebooks.info

Once ready, hit **Transfer**. GitHub will notify you that the transfer might take a few minutes, but if the repository is relatively small with a few collaborators, it will take an instant. If you watch closely in your dashboard where your repository list appears, you will notice that your previous repository now has the organization's namespace before it:



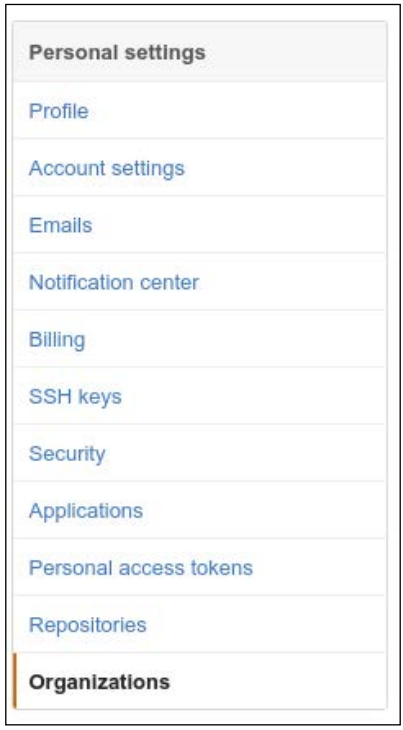
If you head over to the repository's settings in the **Collaborators & teams** tab, you will see that any previous collaborators have their rights retained and that they were converted to outside collaborators:



www.it-ebooks.info

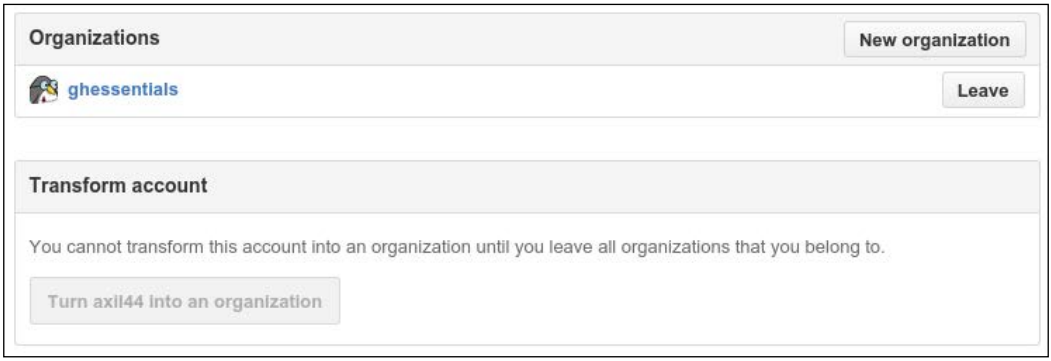
How to convert a user account into an organization

For the reasons mentioned in *The Difference between users and organizations* section, one might want to turn a personal account into an organization. This can be easily done in the user's settings under the **Organizations** tab:

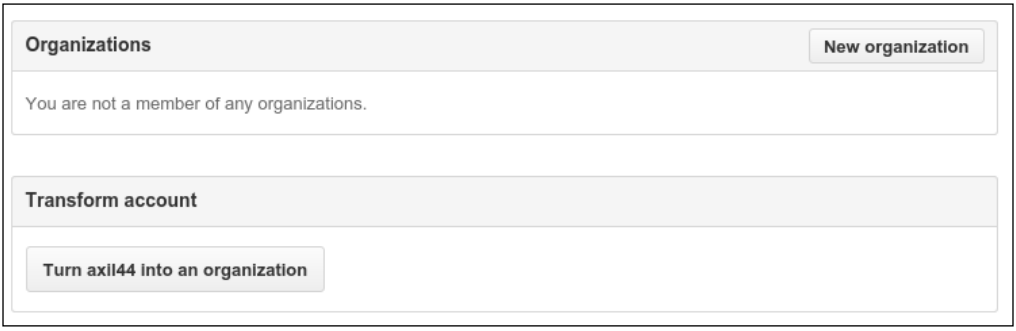


www.it-ebooks.info

There is one caveat, though. If a user is a member of any organization, they will have to first leave the organization and then convert their account:

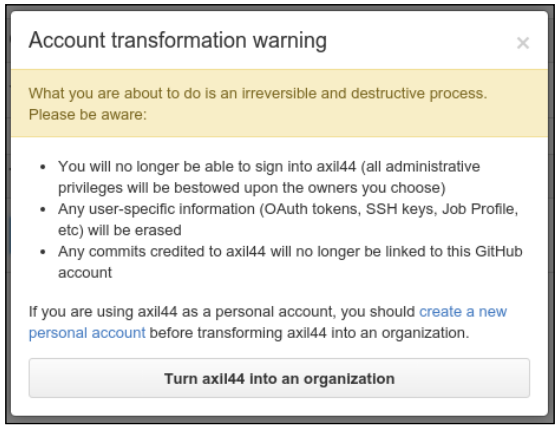


Once done, the option for the conversion will be available:

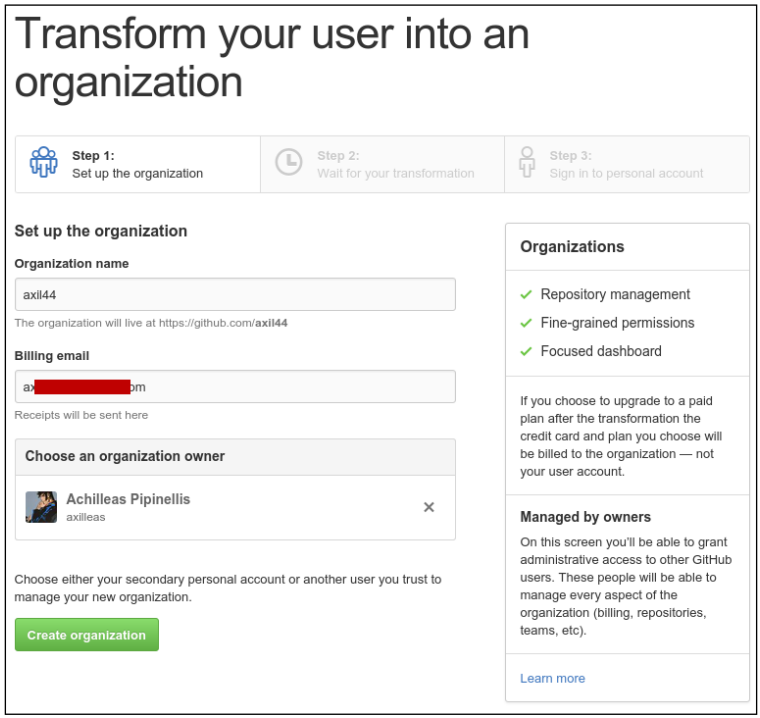


www.it-ebooks.info

Be sure that this is what you want because the action is irreversible:



In the next step, you have to choose an owner of the organization. Start typing and when the name appears, select it and hit **Choose an organization owner**, as shown in the following screenshot:

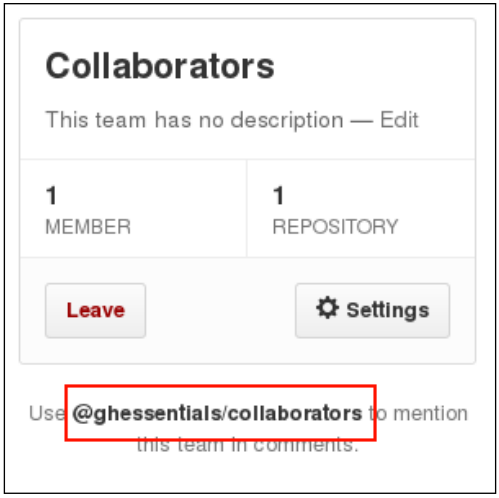


Finally, hit **Create organization** to start the process.

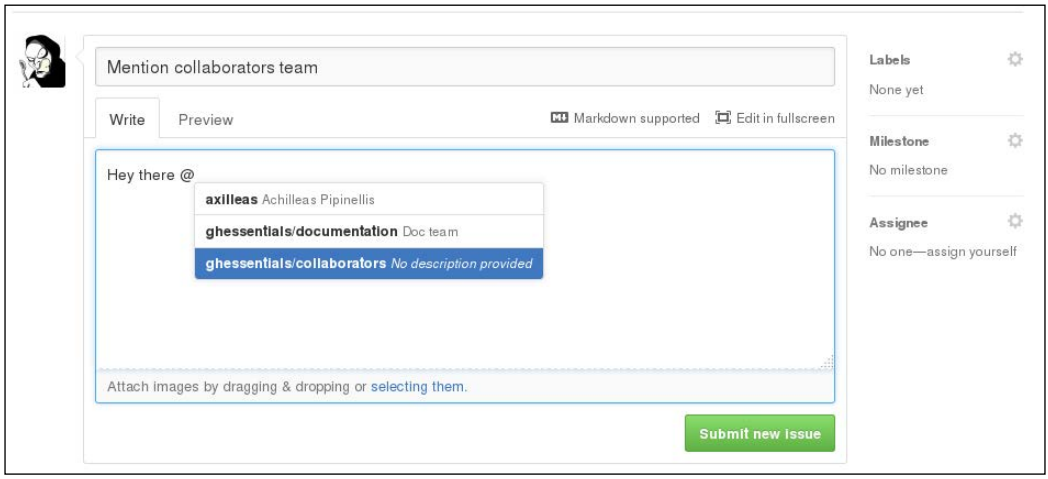
www.it-ebooks.info

Mention teams

A cool way to get a team's attention in issues and pull requests is by mentioning the whole team. This is achieved with the following syntax: `@organization/team`. For example, to get everybody's attention who are in the **Collaborators** team, you would use something like this: `@ghessentials/collaborators`. Only the members and owners of the organization can mention teams. You might have noticed this when you get in a team's dashboard:

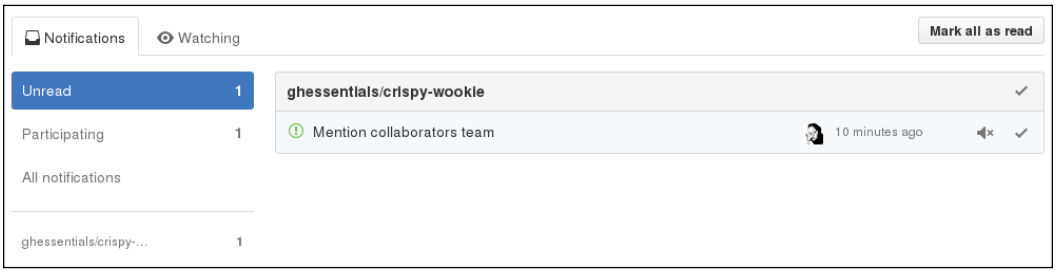


Let's create an issue and see how this works. Head over a repository that is under your organization and create a new issue and try to mention a team:



www.it-ebooks.info

In case you are part of the team or are an owner, you will receive an e-mail and a notification that will appear in your page at `https://github.com/notifications`:



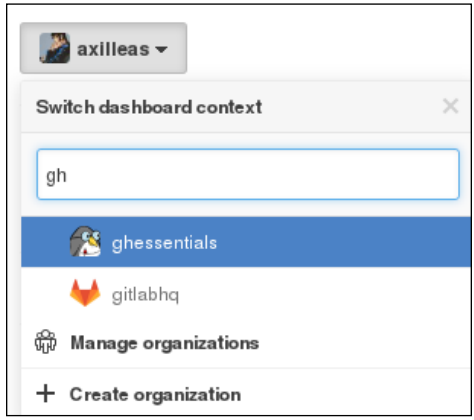
The auto complete results are restricted to repository collaborators and any other participants on the thread, so it's not a full global search.

Read more about this feature at `https://help.github.com/articles/writing-on-github/#name-and-team-mentions-autocomplete`.

Organization feed only in dashboard

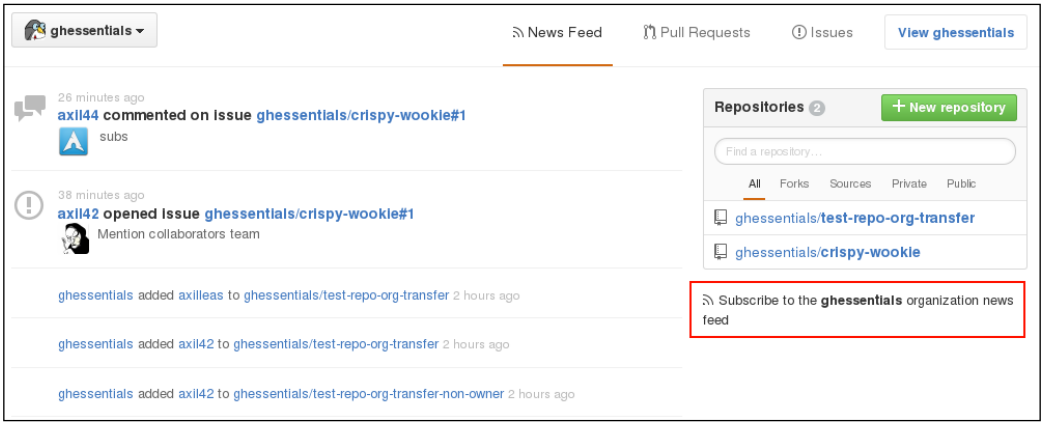
When signed in, visit `https://github.com`. Here is your dashboard activity of projects and the people you follow. Sometimes, when you are a member of many organizations the information might be cluttered, so you would rather filter the activity to one organization.

Just choose from the drop-down menu the organization you wish to filter and in **News Feed**, you will see activities only from that organization:



www.it-ebooks.info

If you want, you can also subscribe to the news feed using the atom feed link found on the right-hand side:



Summary

That's it! Congratulations on finishing this chapter. You should now be familiar with almost all organization features. Creating teams, inviting people, and managing repository access should feel much more easier.

If you found it difficult to follow, I would recommend you to just play with it, make a test organization, and go through the chapter again. Full disclosure, this is how this chapter was written. I created an organization, a second test user and through trial and error this chapter was born. Given that GitHub has released the new organization features in beta, I had to do a lot of testing. I think the result was worth the effort.

In the next chapter, we will explore GitHub's strongest point: collaboration and pull requests.

www.it-ebooks.info

4
Collaboration Using the GitHub Workflow

GitHub is a great tool for collaboration and as such, it has come up with a workflow based on the features it provides and the power of Git. It has named it the GitHub workflow (`https://guides.github.com/introduction/flow`).

In this chapter, we will learn how to work with branches and **pull requests**, which is the most powerful feature of GitHub.

Learn about pull requests

Pull request is the number one feature in GitHub that made it what it is today. It was introduced in early 2008 and is being used extensively among projects since then.

While everything else can be pretty much disabled in a project's settings (such as issues and the wiki), pull requests are always enabled.

Why pull requests are a powerful asset to work with

Whether you are working on a personal project where you are the sole contributor or on a big open source one with contributors from all over the globe, working with pull requests will certainly make your life easier.

www.it-ebooks.info

I like to think of pull requests as chunks of commits, and the GitHub UI helps you visualize clearer what is about to be merged in the default branch or the branch of your choice. Pull requests are reviewable with an enhanced diff view. You can easily revert them with a simple button on GitHub and they can be tested before merging, if a CI service is enabled in the project.

The connection between branches and pull requests

There is a special connection between branches and pull requests. In this connection, GitHub will automatically show you a button to create a new pull request if you push a new branch in your repository. As we will explore in the following sections, this is tightly coupled to the GitHub workflow, and GitHub uses some special words to describe the **from** and **to** branches. As per GitHub's documentation:

> *The base branch is where you think changes should be applied, the head branch is what you would like to be applied.*

So, in GitHub terms, head is your branch, and base the branch you would like to merge into.

Create branches directly in a project – the shared repository model

The shared repository model, as GitHub aptly calls it, is when you push new branches directly to the source repository. From there, you can create a new pull request by comparing between branches, as we will see in the following sections.

Of course, in order to be able to push to a repository you either have to be the owner or a collaborator; in other words you must have write access.

Create branches in your fork – the fork and pull model

Forked repositories are related to their parent in a way that GitHub uses in order to compare their branches. The fork and pull model is usually used in projects when one does not have write access but is willing to contribute.

www.it-ebooks.info

After forking a repository, you push a branch to your fork and then create a pull request in the source repository asking its maintainer to merge the changes. This is common practice to contribute to open source projects hosted on GitHub. You will not have access to their repository, but being open source, you can fork the public repository and work on your own copy.

How to create and submit a pull request

There are quite a few ways to initiate the creation of a pull request, as we you will see in the following sections.

The most common one is to push a branch to your repository and let GitHub's UI guide you. Let's explore this option first.

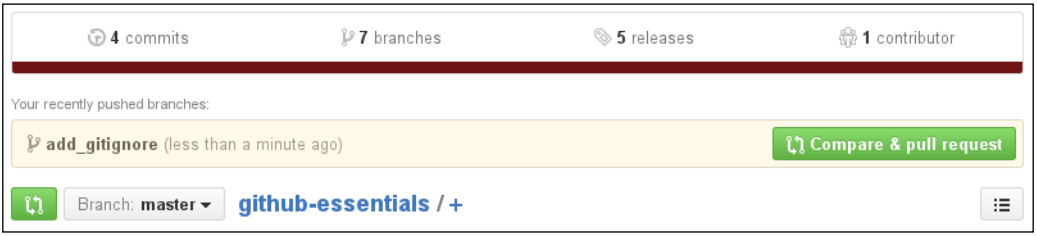
Use the Compare & pull request button

Whenever a new branch is pushed to a repository, GitHub shows a quick button to create a pull request. In reality, you are taken to the compare page, as we will explore in the next section, but some values are already filled out for you.

Let's create, for example, a new branch named `add_gitignore` where we will add a `.gitignore` file with the following contents:

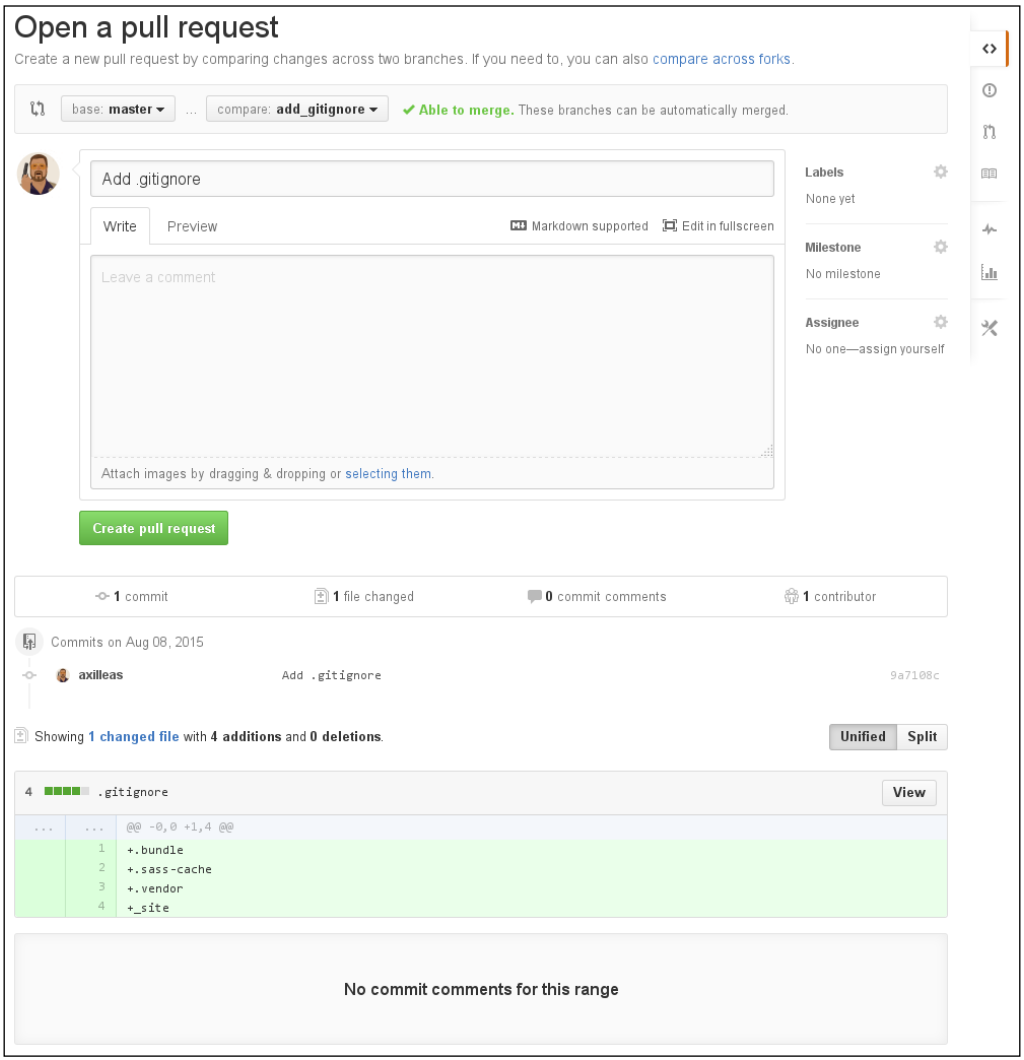
```
git checkout -b add_gitignore
echo -e '.bundle\n.sass-cache\n.vendor\n_site' > .gitignore
git add .gitignore
git commit -m 'Add .gitignore'
git push origin add_gitignore
```

Next, head over your repository's main page and you will notice the **Compare & pull request** button, as shown in the following screenshot:



www.it-ebooks.info

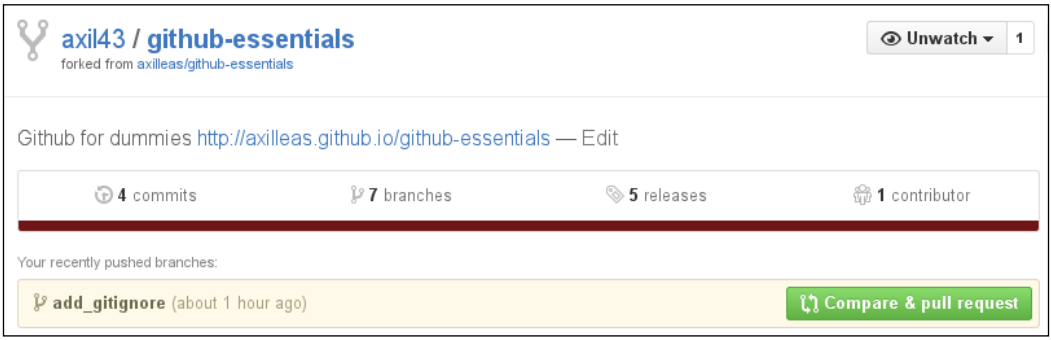
From here on, if you hit this button you will be taken to the compare page. Note that I am pushing to my repository following the shared repository model, so here is how GitHub greets me:



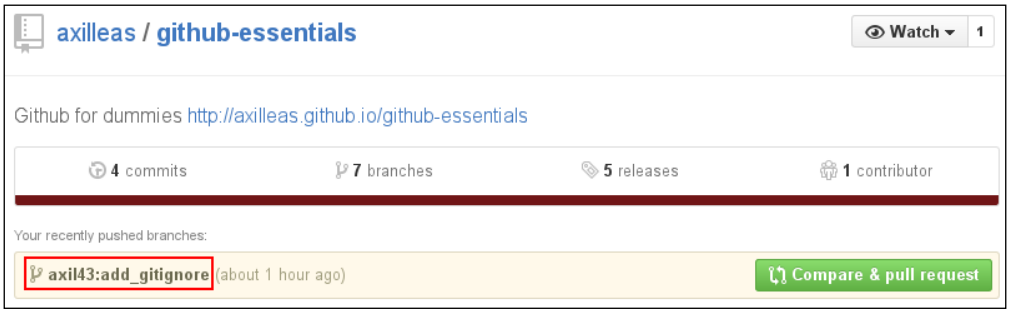
What would happen if I used the fork and pull repository model? For this purpose, I created another user to fork my repository and followed the same instructions to add a new branch named `add_gitignore` with the same changes. From here on, when you push the branch to your fork, the **Compare & pull request** button appears whether you are on your fork's page or on the parent repository.

www.it-ebooks.info

Here is how it looks if you visit your fork:

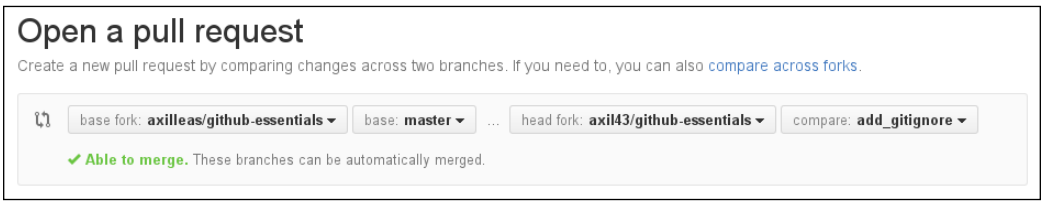


The following screenshot will appear, if you visit the parent repository:



In the last case (captured in red), you can see from which user this branch came from (**axil43:add_gitignore**).

In either case, when using the fork and pull model, hitting the **Compare & pull request** button will take you to the compare page with slightly different options:



Since you are comparing across forks, there are more details. In particular, you can see the base fork and branch as well as the head fork and branch that are the ones you are the owner of.

www.it-ebooks.info

GitHub considers the default branch set in your repository to be the one you want to merge into (base) when the **Create Pull Request** button appears.

Before submitting it, let's explore the other two options that you can use to create a pull request. You can jump to the *Submit a pull request* section if you like.

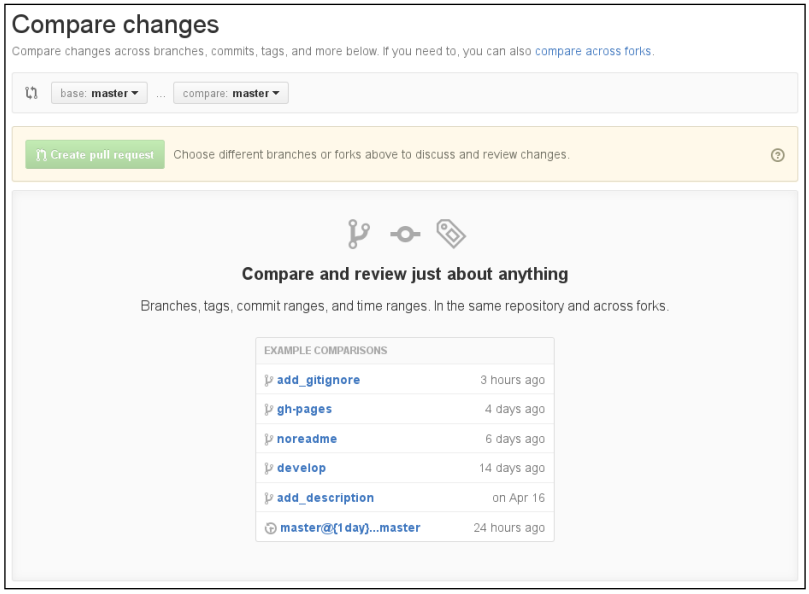
Use the compare function directly

As mentioned in the previous section, the **Compare & pull request** button gets you on the compare page with some predefined values. The button appears right after you push a new branch and is there only for a few moments. In this section, we will see how to use the compare function directly in order to create a pull request.

You can access the compare function by clicking on the green button next to the branch drop-down list on a repository's main page:

This is pretty powerful as one can compare across forks or, in the same repository, pretty much everything—branches, tags, single commits and time ranges.

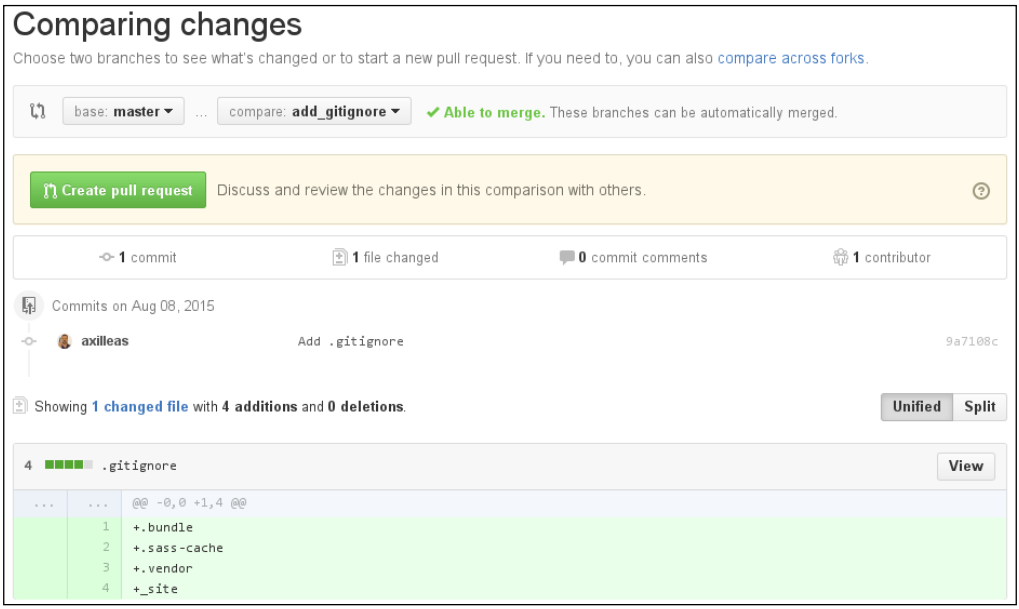
The default page when you land on the compare page is like the following one; you start by comparing your default branch with GitHub, proposing a list of recently created branches to choose from and compare:



www.it-ebooks.info

In order to have something to compare to, the base branch must be older than what you are comparing to.

From here, if I choose the `add_gitignore` branch, GitHub compares it to a master and shows the diff along with the message that it is able to be merged into the base branch without any conflicts. Finally, you can create the pull request:



Notice that I am using the compare function while I'm at my own repository. When comparing in a repository that is a fork of another, the compare function slightly changes and automatically includes more options as we have seen in the previous section.

As you may have noticed the **Compare & pull request** quick button is just a shortcut for using `compare` manually. If you want to have more fine-grained control on the repositories and the branches compared, use the compare feature directly.

Use the GitHub web editor

So far, we have seen the two most well-known types of initiating a pull request. There is a third way as well: using entirely the web editor that GitHub provides. This can prove useful for people who are not too familiar with Git and the terminal, and can also be used by more advanced Git users who want to propose a quick change.

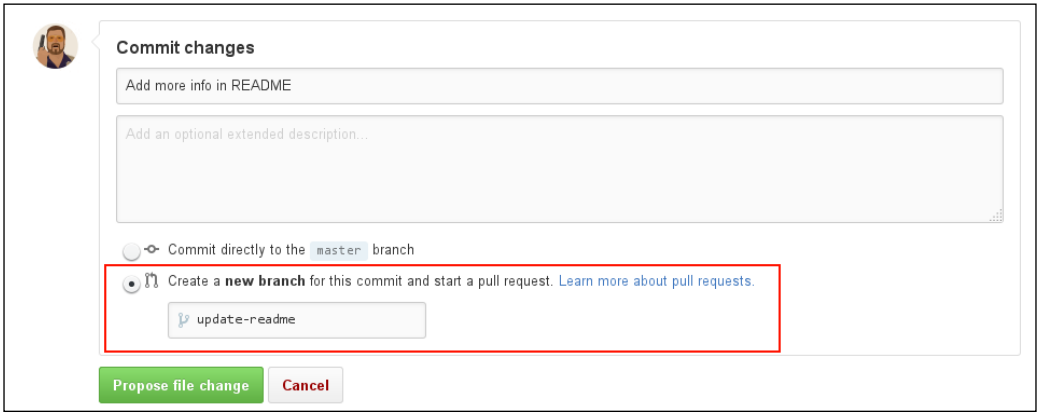
www.it-ebooks.info

As always, according to the model you are using (shared repository or fork and pull), the process is a little different. Let's first explore the shared repository model flow using the web editor, which means editing files in a repository that you own.

The shared repository model

Firstly, make sure you are on the branch that you wish to branch off; then, head over a file you wish to change and press the edit button with the pencil icon:

Make the change you want in that file, add a proper commit message, and choose **Create a new branch** giving the name of the branch you wish to create. By default, the branch name is `username-patch-i`, where `username` is your username and `i` is an increasing integer starting from `1`. Consecutive edits on files will create branches such as `username-patch-1`, `username-patch-2`, and so on. In our example, I decided to give the branch a name of my own:

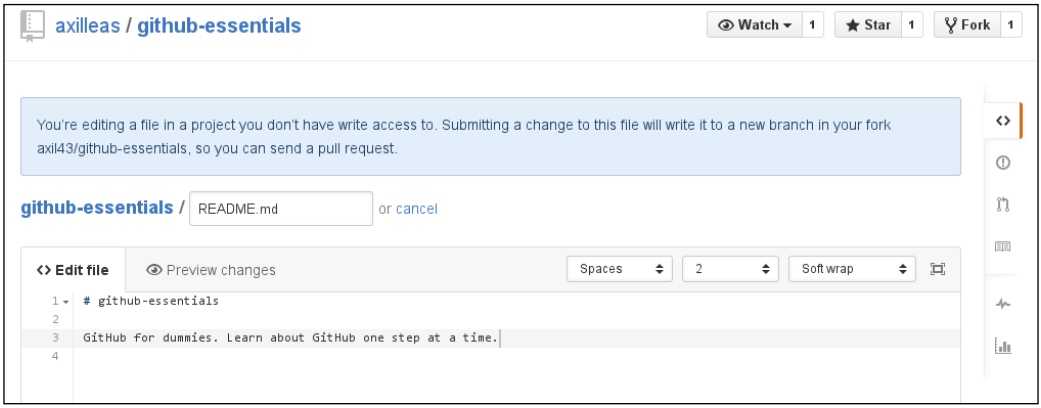


www.it-ebooks.info

When ready, press the **Propose file change** button. From this moment on, the branch is created with the file edits you made. Even if you close the next page, your changes will not be lost. Let's skip the pull request submission for the time being and see how the fork and pull model works.

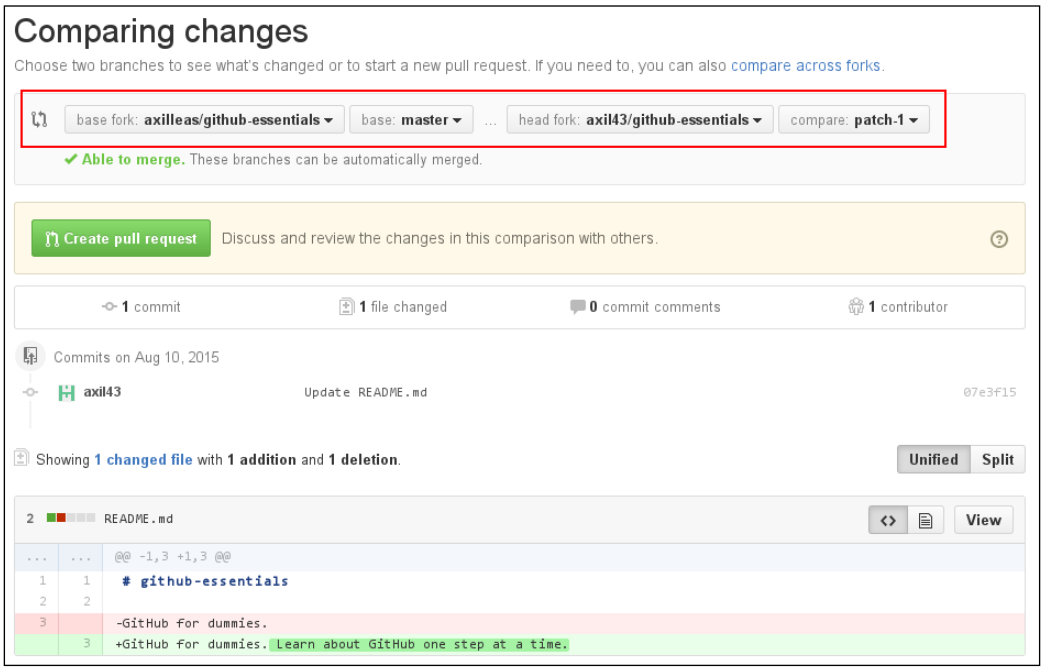
The fork and pull model

In the fork and pull model, you fork a repository and submit a pull request from the changes you make in your fork. In the case of using the web editor, there is a caveat. In order to get GitHub automatically recognize that you wish to perform a pull request in the parent repository, you have to start the web editor from the parent repository and not your fork. In the following screenshot, you can see what happens in this case:



www.it-ebooks.info

GitHub informs you that a new branch will be created in your repository (fork) with the new changes in order to submit a pull request. Hitting the **Propose file change** button will take you to the form to submit the pull request:

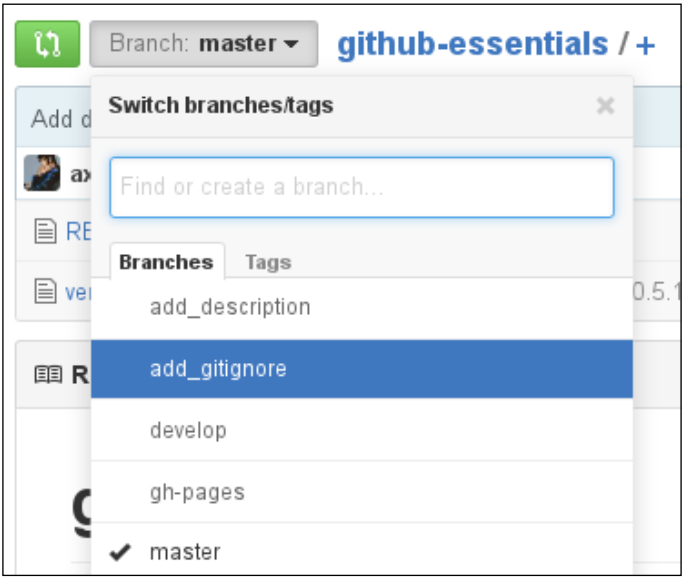


Contrary to the shared repository model, you can now see the base/head repositories and branches that are compared. Also, notice that the default name for the new branch is `patch-i`, where `i` is an increasing integer number. In our case, this was the first branch created that way, so it was named `patch-1`.

If you would like to have the ability to name the branch the way you like, you should follow the shared repository model instructions as explained in preceding section. Following that route, edit the file in your fork where you have write access, add your own branch name, hit the **Propose file change** button for the branch to be created, and then abort when asked to create the pull request. You can then use the **Compare & pull request** quick button or use the compare function directly to propose a pull request to the parent repository.

www.it-ebooks.info

One last thing to consider when using the web editor, is the limitation of editing one file at a time. If you wish to include more changes in the same branch that GitHub created for you when you first edited a file, you must first change to that branch and then make any subsequent changes. How to change the branch? Simply choose it from the drop-down menu as shown in the following screenshot:



Submit a pull request

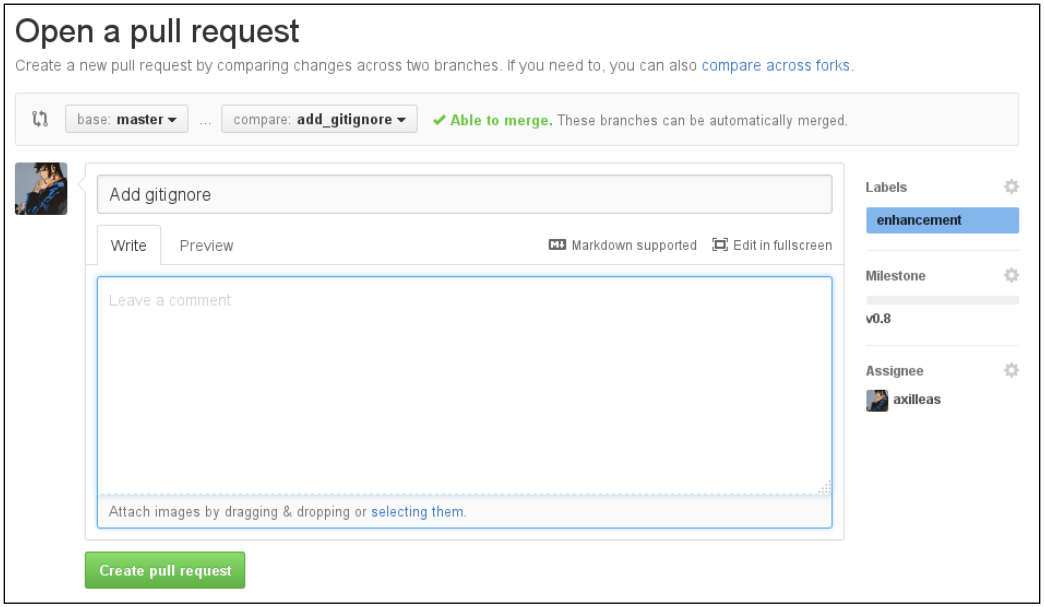
So far, we have explored the various ways to initiate a pull request. In this section, we will finally continue to submit it as well.

The pull request form is identical to the form when creating a new issue. For more details, refer to *Chapter 1*, *Brief Repository Overview and Usage of the Issue Tracker*, the *Learning how to use the powerful benefits of the issue tracker* section.

If you have write access to the repository that you are making the pull request to, then you are able to set labels, milestone, and assignee.

www.it-ebooks.info

The title of the pull request is automatically filled by the last commit message that the branch has, or if there are multiple commits, it will just fill in the branch name. In either case, you can change it to your liking. In the following image, you can see the title is taken from the branch name after GitHub has stripped the special characters. In a sense, the title gets humanized:



You can add an optional description and images if you deem proper. Whenever ready, hit the **Create pull request** button. In the following sections, we will explore how the peer review works and eventually merge the pull request.

Peer review and inline comments

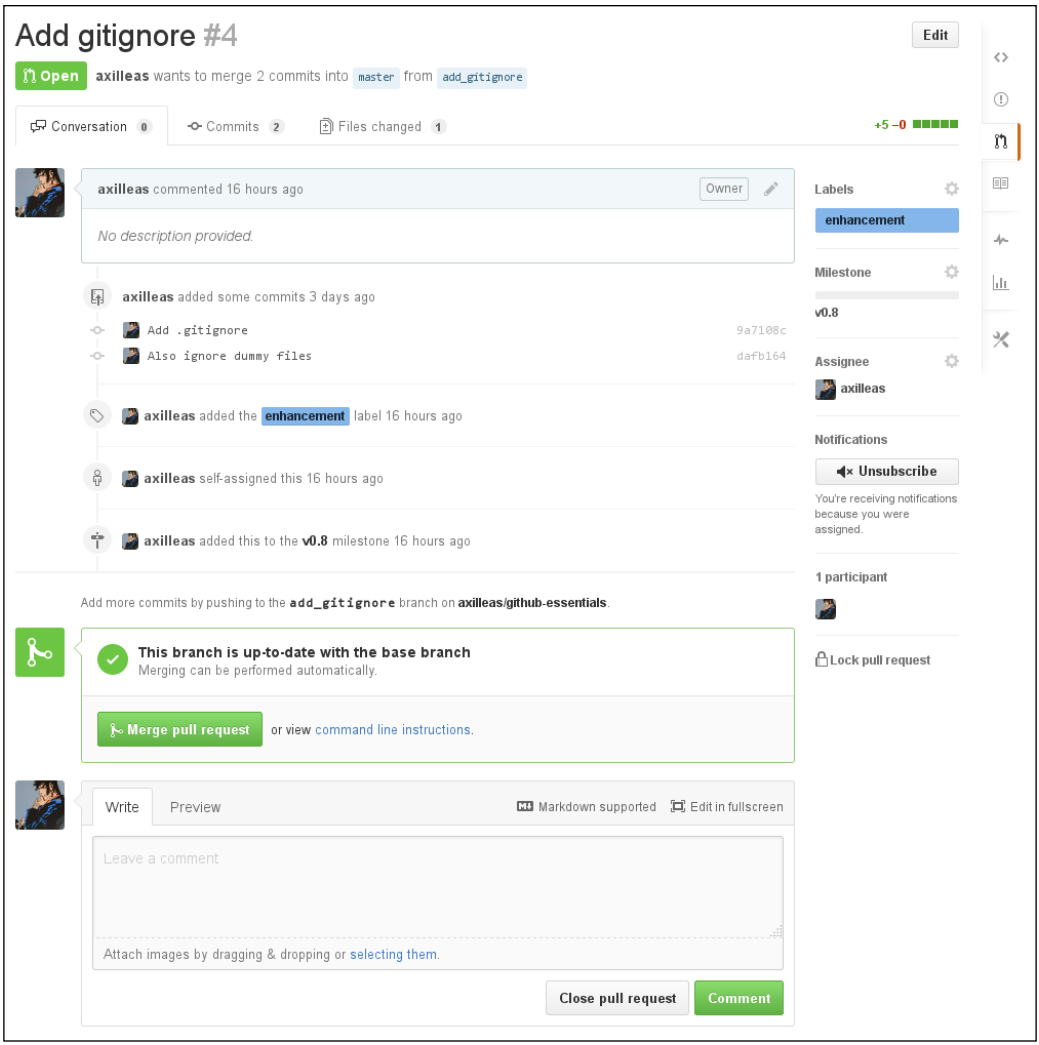
The nice thing about pull requests is that you have a nice and clear view of what is about to get merged. You can see only the changes that matter, and the best part is that you can fire up a discussion concerning those changes.

In the previous section, we submitted the pull request so that it can be reviewed and eventually get merged. Suppose that we are collaborating with a team and they chime in to discuss the changes. Let's first check the layout of a pull request.

www.it-ebooks.info

The layout of a pull request

Every pull request looks pretty much like the following:

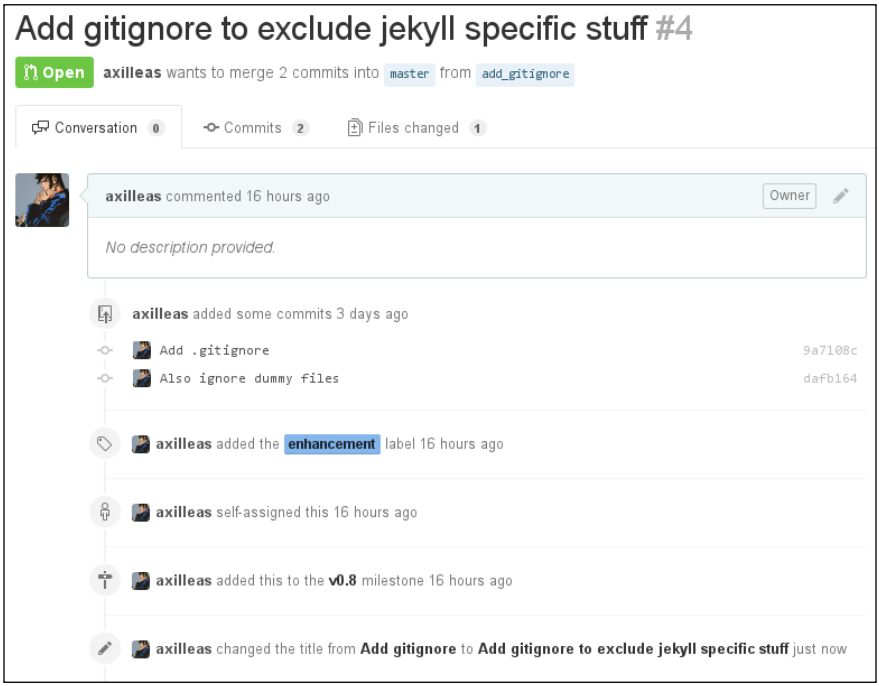


www.it-ebooks.info

From the preceding screenshot, you can tell what the specific number of the pull request is. It is like an identifier within the project and is not separated from the issues count. Issues and pull requests share the same ID counter. So, in the above example, you can see that although this is our first pull request, it has a number of **#4**; the previous three were issues:

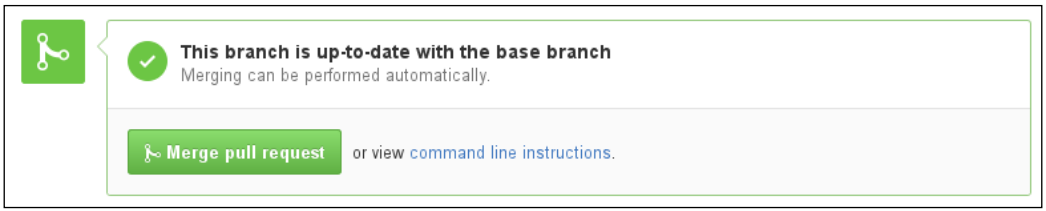
Then, there is the information that the pull request is **Open** and who wants to merge how many commits into what branch from what branch:

Right below the information I just described, there are three tabs: **Conversation**, **Commits**, and **Files changed**. In **Conversation**, except for the comments that we will see in the following screenshots, GitHub also adds information about the events concerning the particular pull request. You can see the action and the time it occurred. For example, take a look at the following screenshot; even little changes such as editing the title are being recorded:



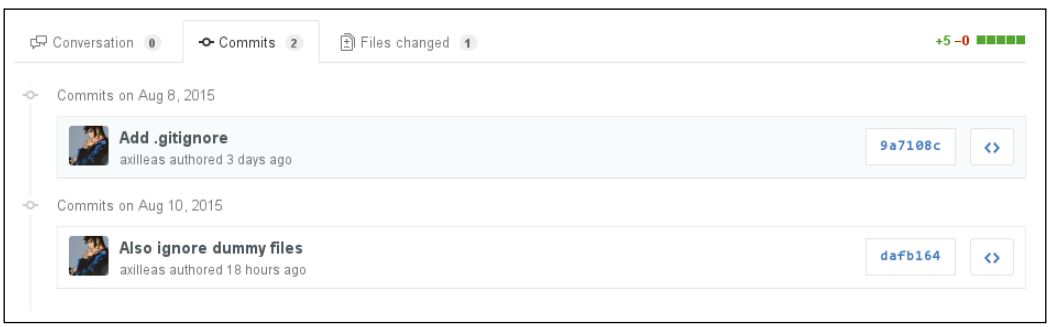
www.it-ebooks.info

The **Conversation** tab is also where the final call takes place. This is where the button to merge the pull request resides, and you can see its status as well. The button is green which means there are no conflicts between the changed files and the ones that are in the repository:



Finally, there is the comment form that is the same as in the issue tracker that we explored in *Chapter 1*, *Brief Repository Overview and Usage of the Issue Tracker*. You can leave any comments concerning the pull request here.

The **Commits** tab shows the commits made in this branch and the commits that are not yet in the branch you are merging into. For example, the `add_gitignore` branch has two commits that do not exist in `master`. GitHub shows the commits in chronological order along with other information, such as who the author is, and links to the commits:



www.it-ebooks.info

The last tab shows the files that are changed in this pull request. There are two ways to see the diff of the commits. The default one is to see the changes in a unified way, with additions and deletions on the same page as shown in the following screenshot:

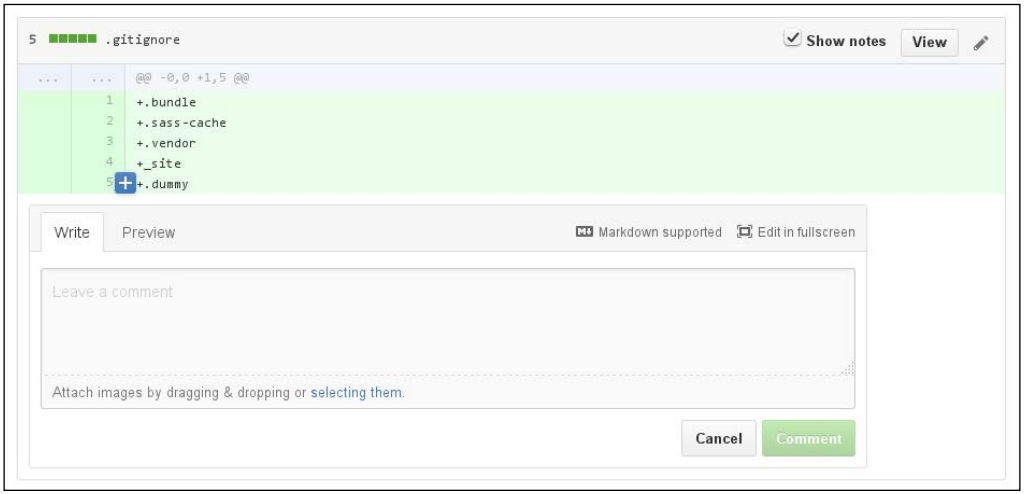
Note that for each addition, GitHub marks a line with a green background color. On the contrary, if you were to remove some lines, they would show in a pink color. I will leave that to you as an exercise.

The other way is to choose **Split**, and GitHub will show the diff in a side-by-side view:

www.it-ebooks.info

Inline comments

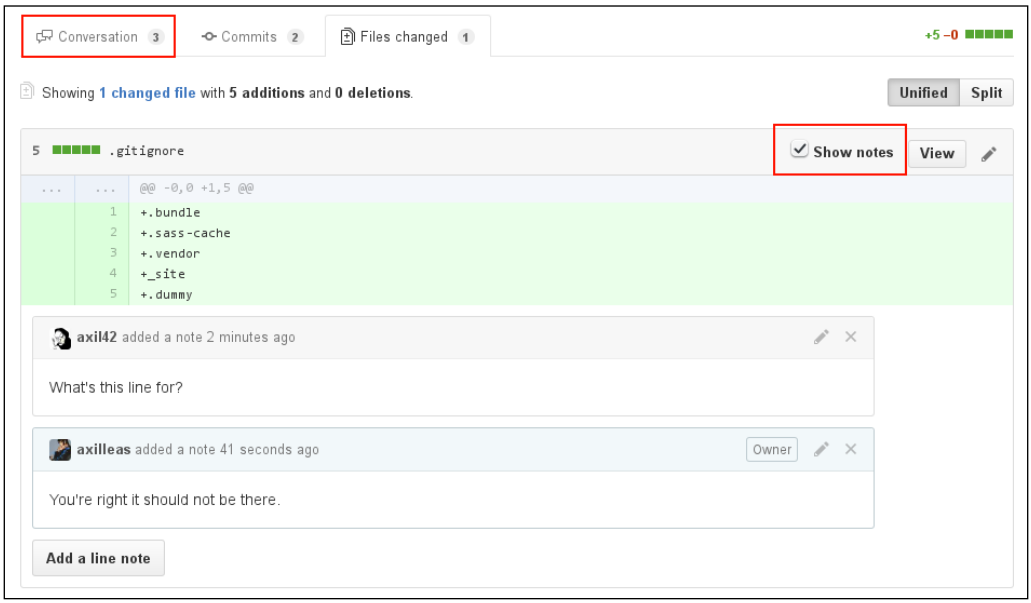
GitHub supports inline comments, so you can leave a comment under every changed line as seen in the **Files changed** tab to fire up a discussion. When hovering over a line, you will notice the cross image, as shown in the following screenshot; click on it and the comment form will appear:



With a few comments on the diff, we can see a couple of changes. First of all, inline comments count to the overall conversation, so the **Conversation** tab should pick that number.

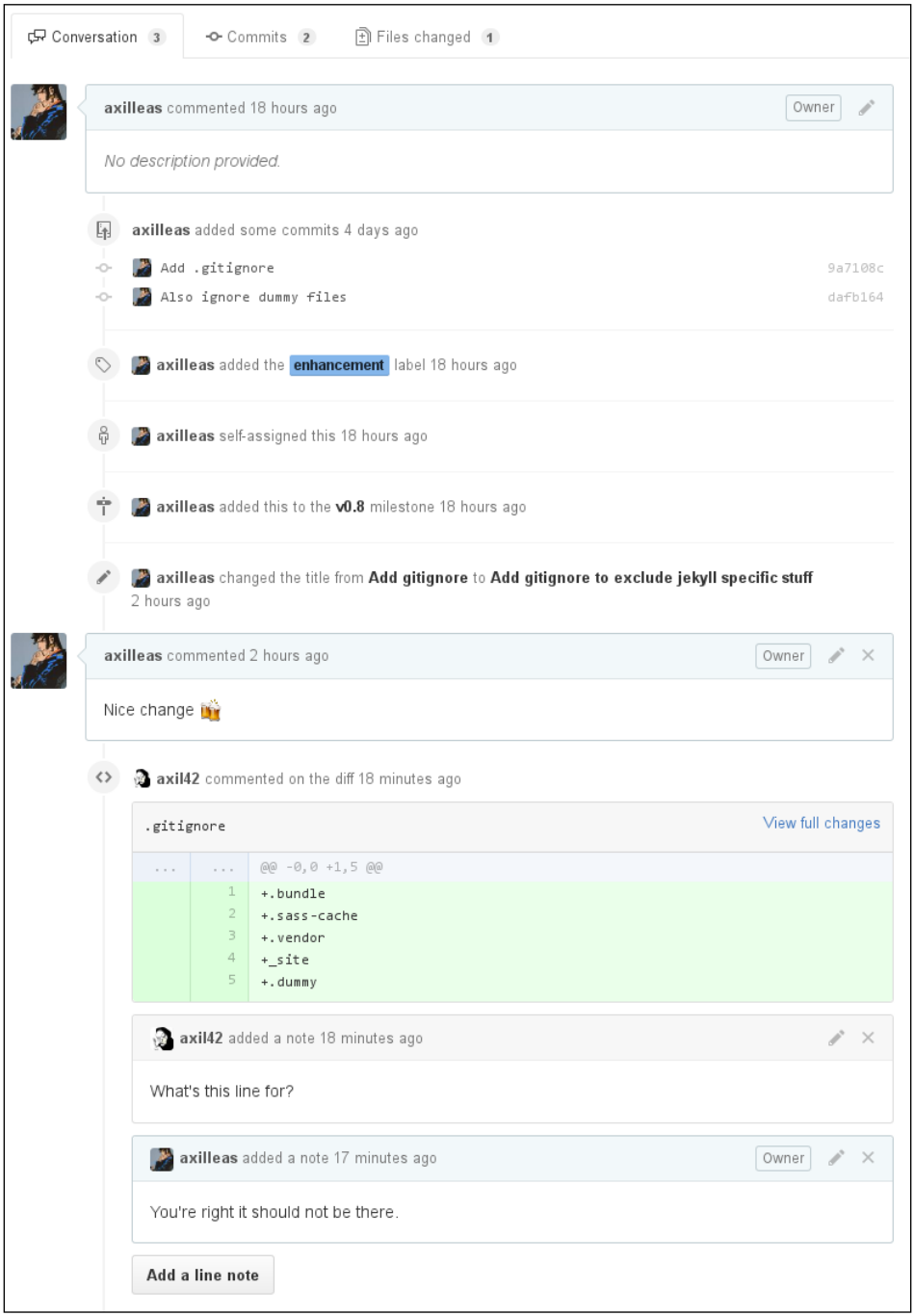
www.it-ebooks.info

Furthermore, when inline comments are present, you can see a **Show notes** choice button that can be toggled off/on in case you want to hide the comments and see the diff clearly:



www.it-ebooks.info

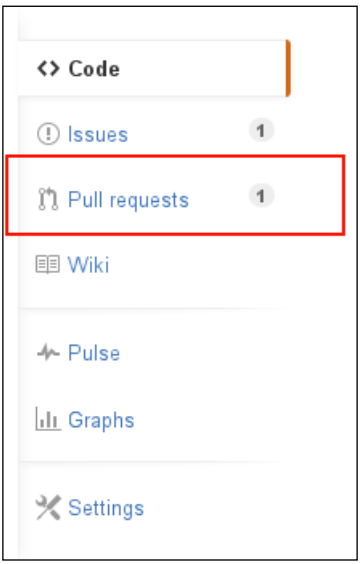
Now, if you head over the **Conversation** tab, the comments appear in the timeline:



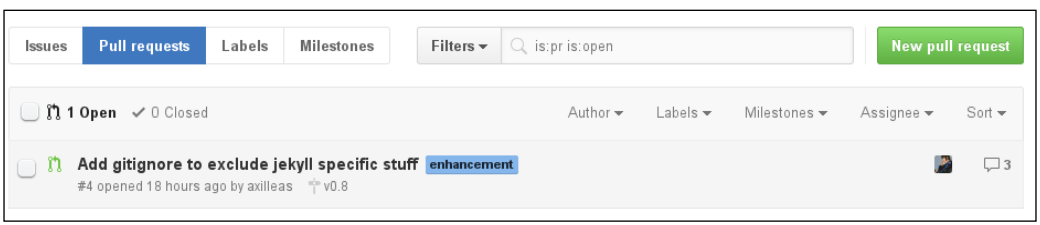
www.it-ebooks.info

Pull requests overview

Much like **Issues**, **Pull requests** have their own space where you can overview them. First of all, you can access them through the right sidebar:



Alternatively, when in the issue tracker, head over the **Pull requests** tab:



If you remember from *Chapter 1*, *Brief Repository Overview and Usage of the Issue Tracker*, you can mass assign issues to users, close them, reopen them, and set milestone and labels. This is the case with pull requests as well, except for merging and reopening them. In order to merge a pull request, you must press the button in each one.

www.it-ebooks.info

Correct mistakes and re-push to branch

So far, we have seen how conversations begin, but what happens when the changes you made need some tweaking to be considered as ready to merge?

In this case, you can push new commits to the branch associated with the pull request, and GitHub will pick up those changes and amend it. The new changes will show up and further feedback can be given. In the *Inline comments* section, my evil twin, user **axil42**, raised a concern about a wrong line being committed. We will now make a new commit and push it to the `add_gitignore` branch and see what happens:

```
git checkout add_gitignore
sed -i '/dummy/d' .gitignore
git add .gitignore
git commit -m 'Remove unneeded dummy file from .gitignore'
git push origin add_gitignore
```

Back in GitHub, three changes happened. Firstly, there was another commit added to the **Commits** tab. Then, in the **Files changed** tab, since the line on which the comments were relying on was removed, the comments no longer appeared. Instead, you can see that in the **Conversation** tab, this particular discussion was marked as outdated followed by the commit that made all this happen:

If you were by any chance in the **Files changed** tab while the last commit was pushed, GitHub would inform you about the changes and would urge you to refresh the page:

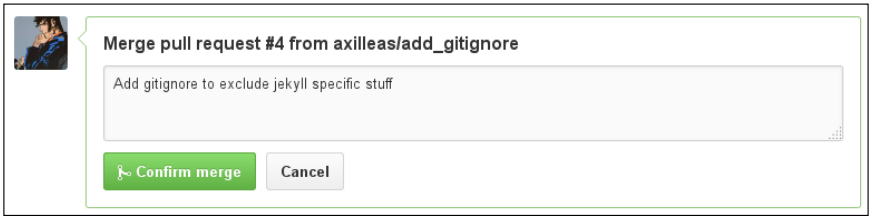
www.it-ebooks.info

Merge the pull request

After the conversation took place, changes were made and the peer review worked as expected, it's now time to finally merge the pull request.

If you don't have access to merge the pull request, you should see the following result:

On the other hand, owners or collaborators with write access can also merge pull requests. In this case, you should see the **Merge pull request** green button. Pressing this button will not merge it immediately, but you will have another chance to confirm:

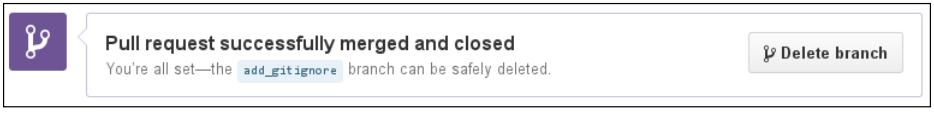


The commit message of this merge is the one in bold, and the one below that can be edited is the extended commit message which by default grabs the pull request title. In the extended commit message you can reference issue numbers with a special meaning. Read more in the *Tips and tricks* section of this chapter to learn how to automatically close issues from pull requests.

Once merged, you can see the green icons turning into purple. This indicates a merged pull request.

Remove/restore a branch after the pull request is merged

In order to have everything cleaned up and tidy, GitHub offers removing the merged branch with a simple button right after the pull request is merged:

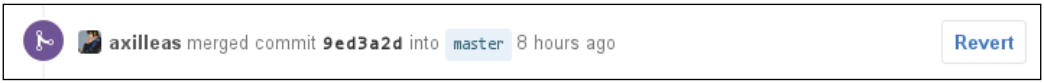


www.it-ebooks.info

After the deletion is completed, GitHub makes this an action event. If you changed your mind, you can always restore the removed branch again using the **Restore branch** button, as shown in the following screenshot:

Revert a pull request

There are cases where you might want to revert a pull request, and GitHub makes this extremely easy. Right after the merge happens, there will be a **Revert** button next to the merge action:



Pressing this button will create a new pull request with opposite commits to the ones the previous pull request included.

Tips and tricks

So far, we explored the main functionality of pull requests. Let's see a couple of things that leverage their power even more.

Close issues via commit messages

In *Chapter 1*, *Brief Repository Overview and Usage of the Issue Tracker*, in the *Tips and tricks* section, you learned how to reference issues inside the issue tracker. Extending this ability, you can reference issue numbers in commit messages in order to close some issues when the commit is merged to the default branch.

For this action to be triggered, you have to use some keywords. For example, `Closes #42` in the commit message will close issue 42 when that commit is merged with the default branch.

www.it-ebooks.info

As per the GitHub documentation, the following keywords will close an issue via a commit message:

- close
- closes
- closed
- fix
- fixes
- fixed
- resolve
- resolves
- resolved

Let's take, for example, an open issue like the following one and note down its number, which in this case is **2**:

Then, make a commit, which in its message has one of the preceding keywords, referencing the preceding issue number. We will follow the GitHub flow that we learned in this chapter, so first create a new branch:

```
git checkout -b fix_issue_2
```


www.it-ebooks.info

For the sake of our example, I modified one file in the repo and then committed with the following:

```
git commit -m 'Demo example of closing issues. Closes #2'
git push origin fix_issue_2
```

Then, open a new pull request to merge the branch we just created, and merge it like you learned in this chapter.

Going back to the issue tracker, you will no longer see issue `#2` among the open issues. Instead, go to the closed ones and you will see that issue `#2` is closed. GitHub provides all the needed information:

For more information on closing issues via commit messages, check out GitHub's documentation at `https://help.github.com/articles/closing-issues-via-commit-messages/`.

Task lists in pull requests

A nice feature when submitting a pull request, which is a work in progress, is the task lists. A work in progress pull request would mean that you work on a specific feature/bug, and so on, but there are many changes that cannot be committed in one go and you also need someone to peer review your progress while working on it.

In this case, you will find task lists quite handy. Let's create a pull request and in the description box, add the following:

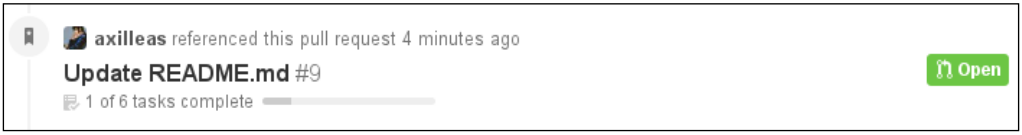
```
- [ ] First item
- [ ] Second item
- [ ] Third item
  - [ ] Fourth nested item
  - [ ] Fifth nested item
- [x] Sixth item , closes #2 (marked as resolved)
```


www.it-ebooks.info

The result will be a list with check boxes where you can manually check/uncheck the items whenever you complete a task:

If you head over the pull request tracker for an overview, you can see the task list showing below the pull request:

This works for cross references as well, and the result will be something like the following screenshot:



Task lists can also exist in issues.

www.it-ebooks.info

Downloading the diff of pull requests

For the hardcore fans of patch and diff files, GitHub has this nice feature where you can view and download the changes a pull request introduces in the format of a patch. Simply append `.patch` to the URL of a pull request. For example, `https://github.com/axilleas/github-essentials/pull/8` becomes `https://github.com/axilleas/github-essentials/pull/8.patch`. The contents of this file include all the commits of a pull request.

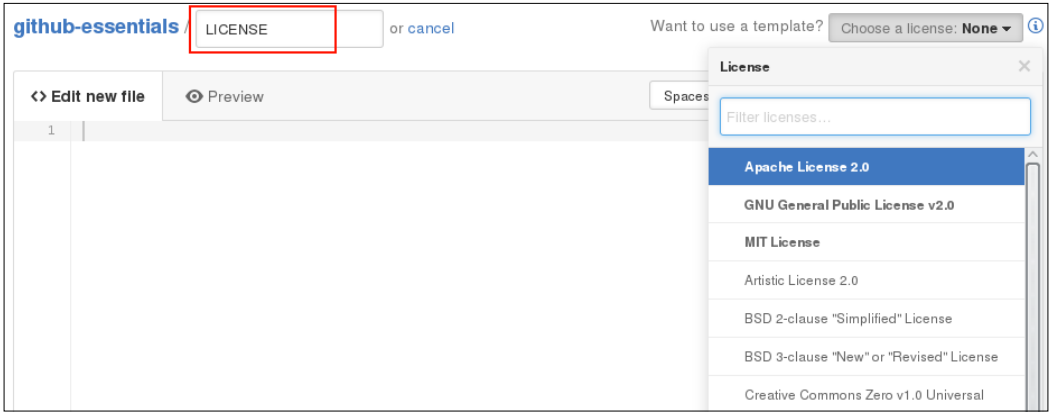
A global list of your open pull requests

Right next to the search bar at the top, there is a link named **Pull requests** that takes you to a page where you can find all your pull requests that are open. Go to `https://github.com/pull` directly to visit this page.

Adding a LICENSE file using the web editor

Much like you can edit the already existing files, you can also create new ones. In this case, we want to add a licence file and GitHub provides a way to choose among a variety of them. In your project's initial page, you will find a plus sign next to the repository name:

Click on it and in the next page, type `LICENCE` so that the drop-down menu, to choose from, appears like in the following screenshot:



www.it-ebooks.info

As we learned in this chapter, pick a license from the menu and commit the change or create a pull request. Here is an easter egg. You can type `LICENCE` the British way or type `LICENSE` the American way. GitHub is smart enough to respect this language quirk and, in fact, it doesn't even care about the case of the letters. For what it worth, typing `LiCENce` or `liCEnSe` is still considered the same!

Lastly, the word copying is also considered to be a synonym to licence, so the previous examples apply to this word as well.

Creating new directories using the web editor

Apart from creating new files, you can also create new directories via the web editor. Just click on the plus button, like I demonstrated in the previous trick of choosing a licence, and type the name of the directory ending with a slash (`/`). You can repeat this process as many times as you like.

The only caveat is that empty directories are not being picked up by Git and in extension by GitHub, so you will have to provide a file at the end if you want to commit this change.

Summary

In this chapter, we explored the GitHub workflow and the various ways to perform a pull request, as well as the many features GitHub provides to make that workflow even smoother. This is how the majority of open source projects work when there are dozens of contributors involved.

In the next chapter, we will see how to make pretty static web pages hosted solely on GitHub and how to read the analytics GitHub provides for each project.

www.it-ebooks.info

5
GitHub Pages and Web Analytics

In this chapter, you will learn to build web pages around your project, hosted for free and exclusively on GitHub, by using Jekyll that is a static site generator or providing your own HTML pages.

Continuing our exploration of the GitHub features, next comes the ability to visualize a repository's data. GitHub has implemented some nice features such as graphs that can depict among others the commit activity of the contributors, the traffic a repository gains, as well as the commit history in a network graph.

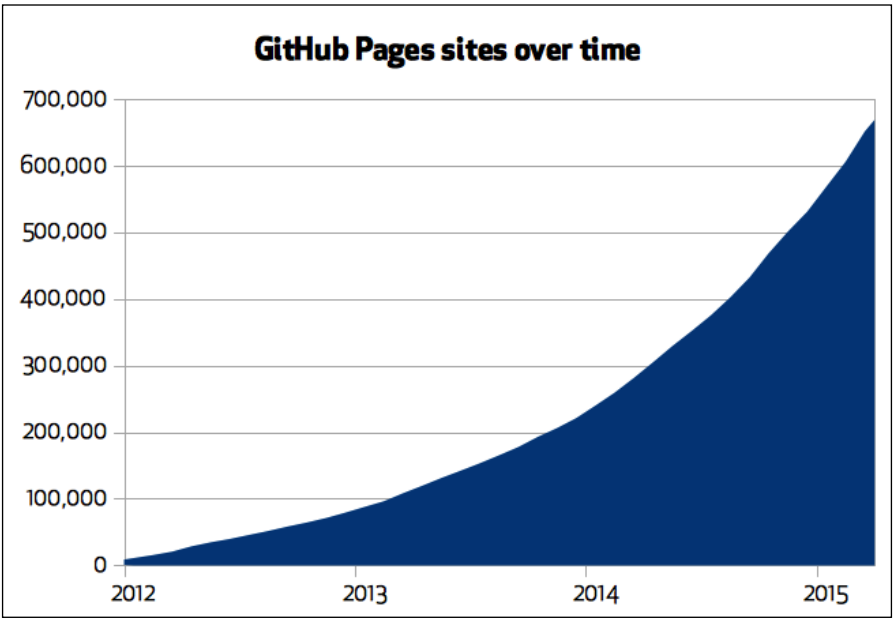
Let's dive in!

GitHub Pages

At the end of the year 2008, GitHub announced GitHub Pages (`https://github.com/blog/272-github-pages`), a static-site hosting service. Static sites have met a great rise during the past years and GitHub played a big part in it. A static site is a site that contains pages written in HTML, CSS, and Javascript. No server code, such as php, ruby, and Python is included.

www.it-ebooks.info

In an interesting post on April 27, 2015 (`https://github.com/blog/1992-eight-lessons-learned-hacking-on-github-pages-for-six-months`), GitHub released some statistics of the GitHub page usage over time:



The growth is exponential and shows that a great deal of users trust GitHub to host their personal or organization sites.

User, organization, and project pages

In order to create a functional website hosted on GitHub Pages, you must follow some conventions.

For users and organizations, a repository named `username.github.io` must be created, where `username` is your username or organization name, and files must be pushed to the master branch.

For project pages, if there is a branch named `gh-pages` in the repository, then its contents are served by GitHub. Regardless of whether you created user or organization GitHub Pages, the project page will be accessible via `http(s)://<username>.github.io/<repositoryname>`.

www.it-ebooks.info

In order to create a project page, you are required to at least create the user/org repository for GitHub Pages, even if it's empty. The logic behind this convention is that your `username.github.io` repository serves as the main page, like the home page of a blog or a site, and the individual projects live under it like in a subdirectory.

You may find that some people name their repositories after `username.github.com` and not `username.github.io`. This was before GitHub announced the domain transition; now all old domains will redirect to `github.io`.

Let's see in detail how to create either of these pages.

Creating a user or an organization page

Create a new empty repository named after your username; for example, `superman.github.io`.

After creating it, clone it locally and add a test `index.html` page:

```
git clone git@github.com:username/username.github.io.git
cd username.github.io
echo 'Welcome to your first page' > index.html
git add index.html
git commit -m 'Add the first webpage'
git push -u origin master
```

Right after the upload finishes, visit `<http://username.github.io>` and see the results.

That's it! You can start writing your own HTML pages and push to GitHub. The changes are instant.

Creating a project page manually

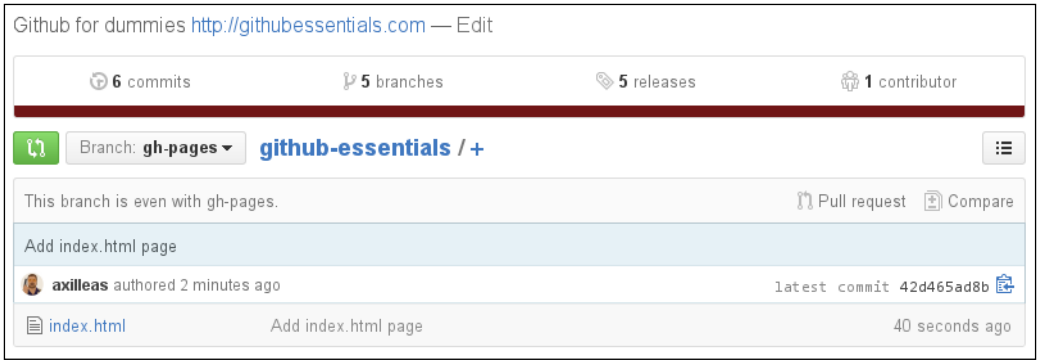
Project pages are somewhat different than user/org pages; they can reside inside every project under a branch named `gh-pages`.

www.it-ebooks.info

Here, I will create the `gh-pages` branch in the `github-essentials` repository, make a new `index.html` file, and commit and push to the `gh-pages` branch:

```
git checkout -b gh-pages
echo 'Welcome to your first page' > index.html
git add index.html
git commit -m 'Add index.html page'
git push origin gh-pages
```

The result of the preceding code snippet is shown in the following screenshot:



As you can see, the `index.html` file is uploaded and the repository page is ready for view. Since I uploaded it under my username (`axilleas`) in a repository named `github-essentials`, you know that the URL will be `http://axilleas.github.io/github-essentials`.

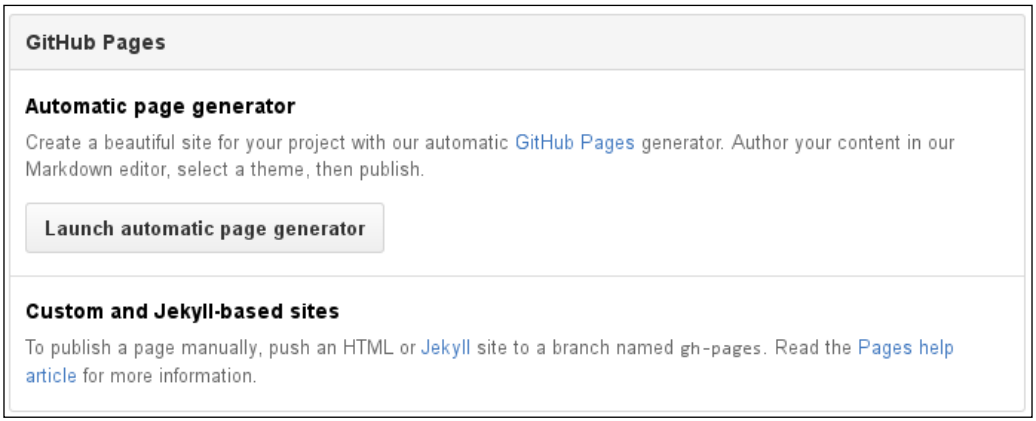
While you could manually modify the contents of the project's website, GitHub provides a better automatic method to update the contents of your web page in one go. Read the following sections to learn how to achieve this.

Creating a project page with GitHub page generator

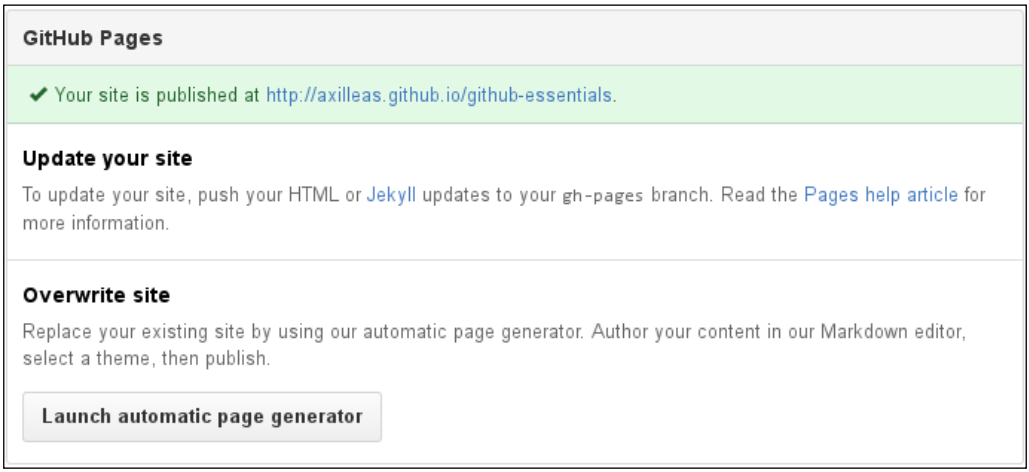
About four years after the launch of GitHub Pages, GitHub announced the addition of another feature, the GitHub page generator.

www.it-ebooks.info

This is an easy way to bootstrap a website for your project with just a few clicks. Under each project's settings, there is a section for **GitHub Pages** with this option:



In case `gh-pages` already exists, GitHub will have an option to use the automatic page generator by overwriting the content of the `gh-pages` branch:

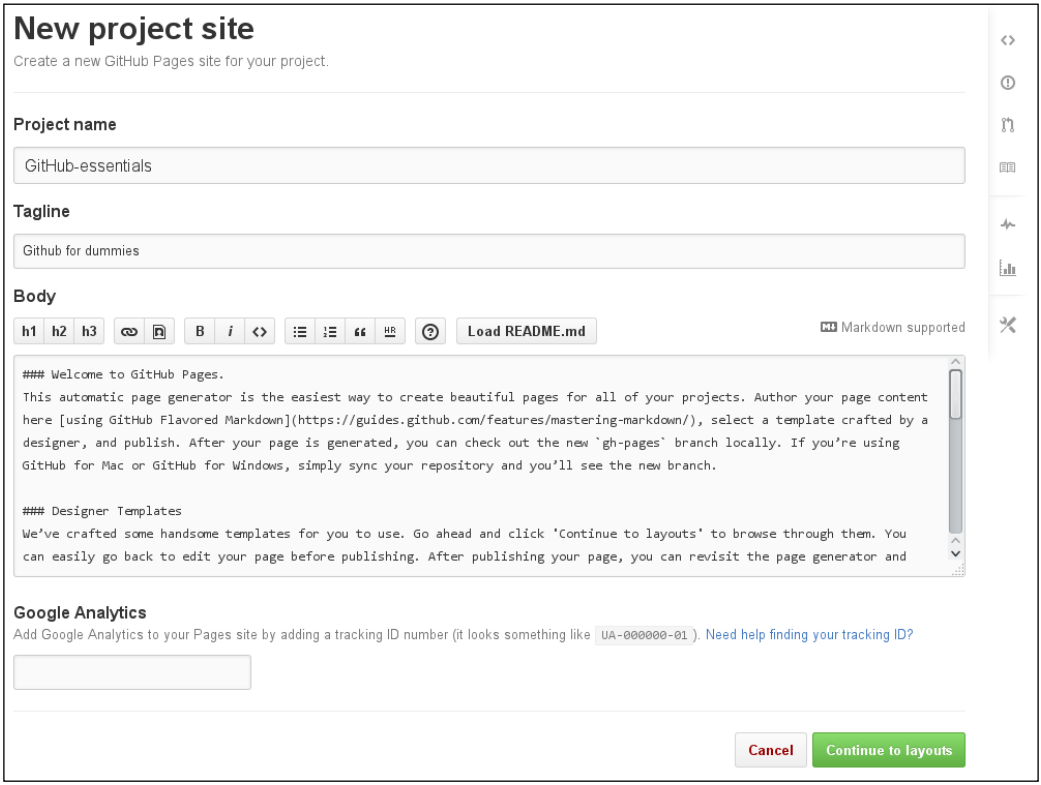


When you launch the page generator, in an interactive step-by-step guide, GitHub helps you to create a single HTML page with a variety of beautiful layouts to choose from.

Let's see how this is done.

www.it-ebooks.info

Firstly, you choose the **Project name** option that will appear as an `h1` HTML heading. Then, there is an optional **Tagline** option that will show as an `h2` HTML heading. In the body box, you put the main information that will be rendered in the `index.html` page and you write in Markdown as we saw in *Chapter 2, Using the Wiki and Managing Code Versioning*:

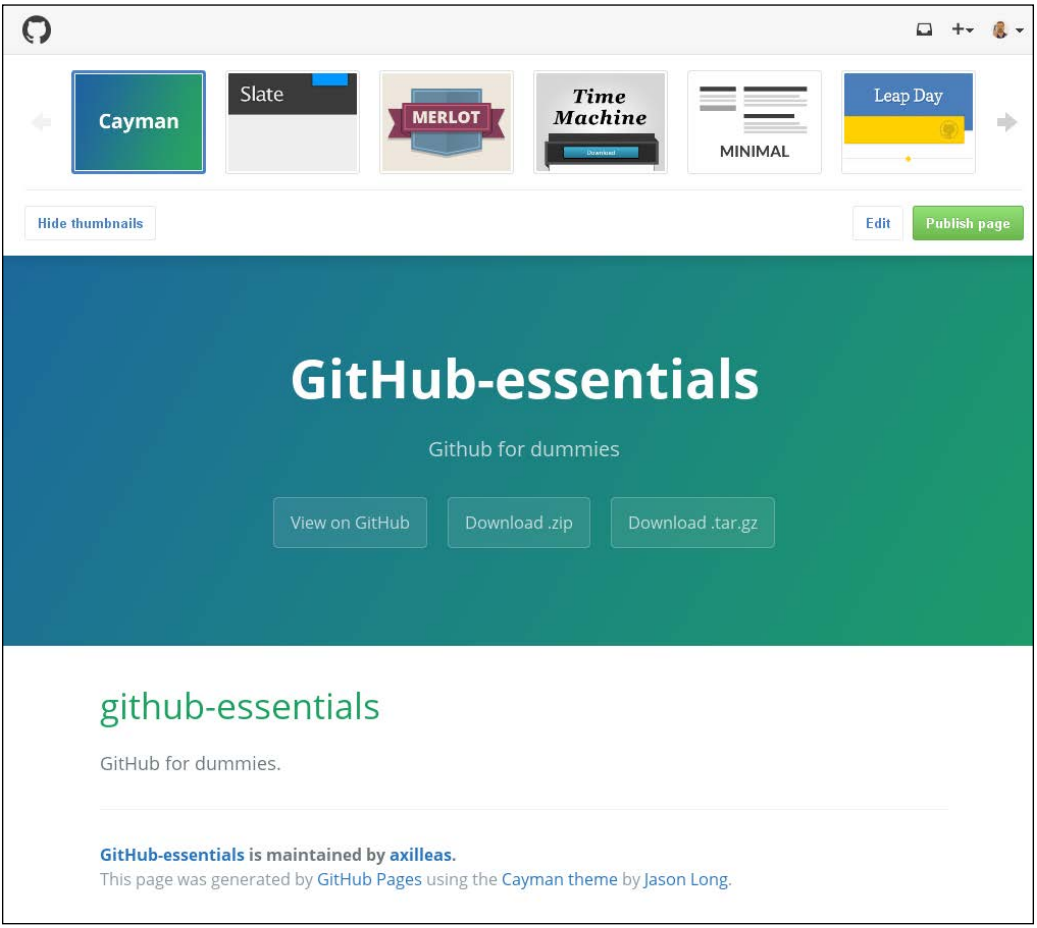


The interesting part is that, if there is a `README` file in your master branch (no matter what is its extension), you can load its contents directly to the body box by clicking on **Load README.md**. Since the editor supports Markdown, it is advised to use this markup language:

www.it-ebooks.info

In the preceding image, you can see that I loaded the `README` file and I can undo this action if needed.

Let's get on the next step by clicking on **Continue to layouts**:



Here, you can see a preview of the final web page. You can choose from 12 layouts and, as you may have noticed, each layout follows a pattern. There are links or buttons for the repository URL on GitHub and links for the ZIP or TAR file downloads. When ready, hit **Publish page** and the changes will be instant.

www.it-ebooks.info

Updating a project page with GitHub page generator

Although making pages with the GitHub page generator is very easy, it is a one-time process. In order to update the project website and thus the `gh-pages` branch, you have to do it manually.

If you followed the previous route of copying the `README` file as the body of your page, then you are probably updating the `README` file while your project evolves. In this case, if you visit the project's settings and launch the page generator, you can edit the body of the website by loading the `README` file once again.

Using a custom domain

Instead of using the classic GitHub Pages URL, you can use a custom domain name linked with your user/org GitHub page. That means you can tell GitHub that when someone asks for `user.com`, it will serve the content in `user.github.io`. Likewise, if you visit `user.github.io`, you will be redirected to `user.com`.

For this purpose, you will need to provide a `CNAME` file in the repository that is hosting the pages. For user/org pages, you will need to push a `CNAME` file at the master branch, whereas for single project pages, to the `gh-pages` branch.

The `CNAME` file should include one line of your **Fully Qualified Domain Name** (**FQDN**). After the `CNAME` file is in the repository, you must add the proper entries to your DNS provider. This is beyond the scope of this tutorial and you should consult the relevant GitHub documentation:

- `https://help.github.com/articles/adding-a-cname-file-to-your-repository`
- `https://help.github.com/articles/about-custom-domains-for-github-pages-sites/`
- `https://help.github.com/articles/tips-for-configuring-an-a-record-with-your-dns-provider/`
- `https://help.github.com/articles/tips-for-configuring-a-cname-record-with-your-dns-provider/`

To fully understand how redirects and GitHub Pages are used, take a look at a real-world example at `https://help.github.com/articles/adding-a-cname-file-to-your-repository/#example-of-a-real-world-cname-file`.

www.it-ebooks.info

How to customize your page using Jekyll

So far, we have seen how to create web pages manually by pushing HTML files or using the GitHub page generator for project pages. There is another way, which is more sophisticated, to build your website.

Every day, more and more people turn to using static websites for their personal projects, and even companies use it for their main sites or blog platform. A static site is faster and more secure than the one built with a server-side language such as php or ruby. On the other hand, maintaining a static site and updating its content completely manually is a tedious task.

For the above reasons, there exist the so-called static site generators, applications that use templates, markup languages, and configuration files, and convert these to pure HTML pages.

GitHub Pages use Jekyll that is a static generator written in ruby and is among the top open source static site generators (`https://www.staticgen.com/`).

Installing Jekyll

Jekyll is written in Ruby; thus, you must first install it. Refer to the official documentation (`https://www.ruby-lang.org/en/documentation/installation/`) to install Ruby in your operating system. If you are on a Unix system, you might prefer to use a Ruby manager (`https://www.ruby-lang.org/en/documentation/installation/#managers`). Bear in mind that you will need at least Ruby 2.0.0, which you can confirm by running `ruby --version`.

The next step is to install Bundler, which is a dependency manager for the various Ruby libraries called gems. If you installed Ruby via your package manager, run the following command to install bundler system-wide:

```
sudo gem install bundler
```

The bundler executable will be installed in `/usr/local/bin/bundle`. If you used a Ruby version manager, there is no need to run it with the `sudo` privileges.

Lastly, you must install Jekyll. GitHub provides a gem called `github-pages` that includes all the dependencies that are also used by GitHub when running Jekyll in their servers. That way, when you develop your site locally, you are very close to the site that will be eventually rendered by GitHub Pages.

www.it-ebooks.info

Make sure your user/org or project pages' repository does not include any files or directories. Create a file named `Gemfile` with the following contents:

```
source 'https://rubygems.org'
gem 'github-pages'
```

Then, run the following command:

```
bundle install --path .vendor/bundle
```

This will install Jekyll and all its dependencies in `.vendor/bundle` so that they don't clutter with your system.

If you want to speed up the process, you can run the bundle command such as `bundle install --jobs 4` where `jobs` is used for parallel download and installation. A suggested number is that of your CPU processors.

You can check whether Jekyll was correctly installed by running `jekyll` or `bundle exec jekyll`. You should see an output similar to the following:

```
jekyll 2.4.0 -- Jekyll is a blog-aware, static site generator in
  Ruby

Usage:

  jekyll <subcommand> Insert image options]

Options:
        -s, --source [DIR]  Source directory (defaults to ./)
        -d, --destination [DIR]  Destination directory (defaults
        to ./_site)
            --safe         Safe mode (defaults to false)
        -p, --plugins PLUGINS_DIR1[,PLUGINS_DIR2[,...]]  Plugins
        directory (defaults to ./_plugins)
            --layouts DIR  Layouts directory (defaults to
            ./_layouts)
        -h, --help         Show this message
        -v, --version      Print the name and version
        -t, --trace        Show the full backtrace when an error
        occurs

Subcommands:
```


www.it-ebooks.info

```
help                    Show the help message, optionally for a
given subcommand.
docs                    Launch local server with docs for Jekyll
v2.4.0
new                     Creates a new Jekyll site scaffold in PATH
build                   Build your site
doctor, hyde            Search site and print specific deprecation
warnings
serve, server           Serve your site locally
```

Introduction to Jekyll

Now, we will create a new boilerplate site that Jekyll provides in order to start building upon it. This is achieved with the `jekyll new path/to/site` command, but there is a caveat. It needs a clean directory; however, in our case, we have `Gemfile`, `Gemfile.lock`, and the `vendor` folder.

So, we will initialize the Jekyll site in a new directory and then move its contents to the current directory. This is because the Jekyll `config` file and templates must be in the root directory of the repository and not in a subfolder.

Run the following commands:

```
bundle exec jekyll new initdir
mv initdir/{.,}* .
rmdir initdir
```

After this, you should see the following directories and files:

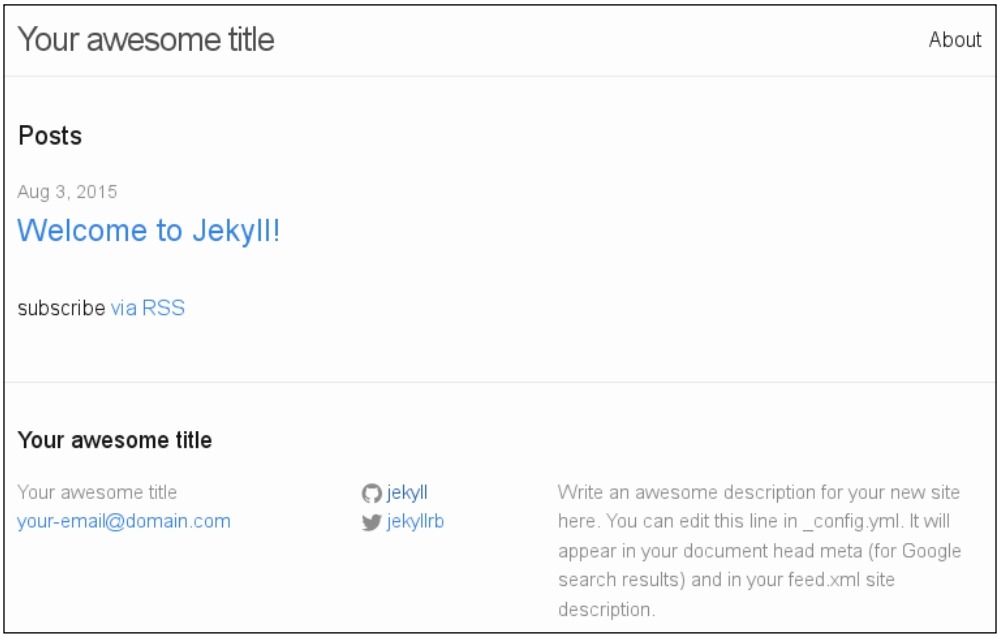
```
.bundle/
.git/
.gitignore
.vendor
Gemfile
Gemfile.lock
_config.yml
_includes/
_layouts/
_posts/
_sass/
about.md
css/
feed.xml
index.html
```


www.it-ebooks.info

Now, let's build the site locally and see how that looks:

```
bundle exec jekyll serve --watch
```

Open your browser to `http://0.0.0.0:4000` and you should see the default Jekyll boilerplate site:



The `-watch` switch enables the auto generation of files so that you don't have to stop and start the server all the time. However, if you edit `_config.yml`, you must restart the server, by stopping and running the preceding command again.

From there on, you can start hacking on the new website. For starters, try and edit `_config.yml` and change some options. After changing the title, description, and e-mail, stop and start Jekyll again to see the changes in effect.

www.it-ebooks.info

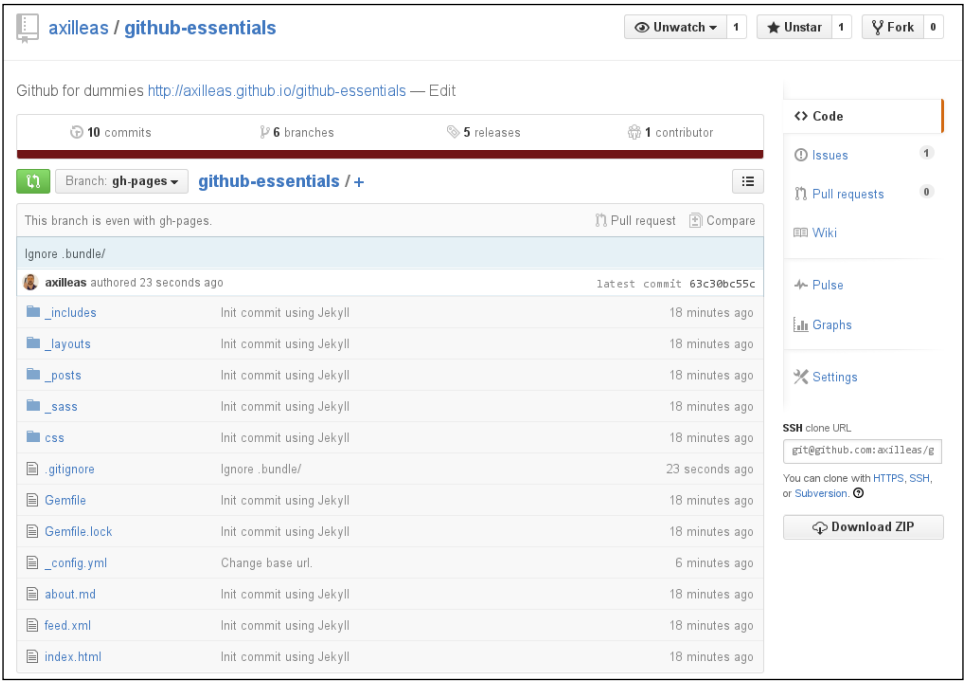
Finally, let's push to the user GitHub repository, but first add `.vendor` to `.gitignore` (you don't want to upload 120 MB to GitHub). To do this, run the following command:

```
echo .vendor/ >> .gitignore
echo .bundle/ >> .gitignore
git add .
git commit -m 'Init commit using Jekyll'
git push origin master
```

Visit the project page at `<username>.github.io` to confirm that all went well.

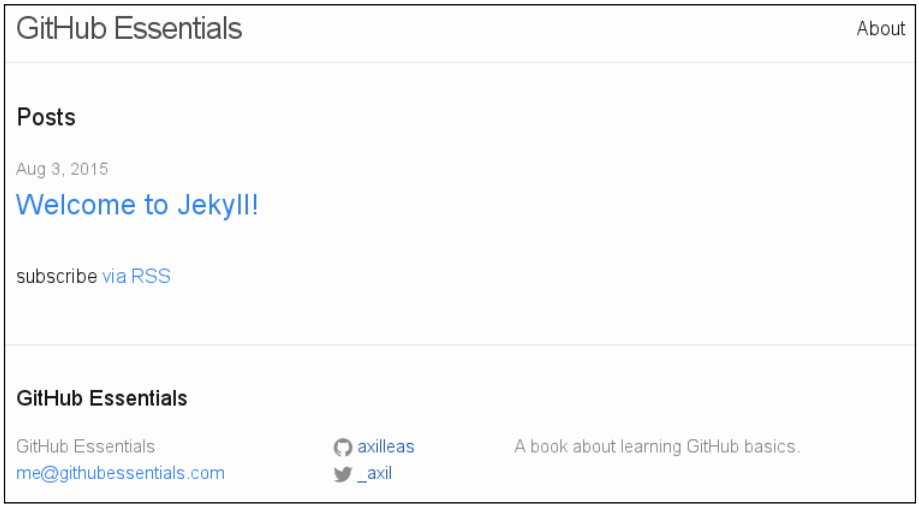
For project pages, make sure that the `baseurl` in `_config.yml` reads `/repository-name/`, otherwise the CSS files won't be picked up correctly. Also, remember that for project pages, you have to push at the `gh-pages` branch.

As an example, see how my repository page looks like:



www.it-ebooks.info

Then, see the generated site as follows:



The source code can be found at `https://github.com/axilleas/github-essentials/tree/gh-pages`.

Read more about Jekyll

As you may have noticed, we have only set up the base for developing with Jekyll. Refer to the Jekyll website for extensive documentation at `http://jekyllrb.com/docs/home/` and see a list of sites that run Jekyll at `https://github.com/jekyll/jekyll/wiki/sites`.

Other helpful articles are, of course, the GitHub help pages about Jekyll:

- `https://help.github.com/articles/using-jekyll-with-pages/`
- `https://help.github.com/articles/using-jekyll-plugins-with-github-pages/`

Finally, for an article about building a blog with Jekyll and GitHub Pages, refer to:

- `http://www.smashingmagazine.com/2014/08/build-blog-jekyll-github-pages/`

www.it-ebooks.info

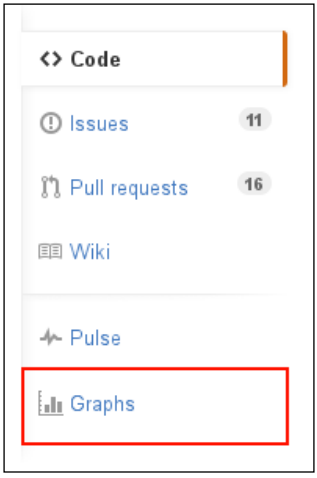
Web analytics

Due to GitHub's nature, a repository contains many metadata such as commits over time, who contributed what, number of contributors, number of forks, and even site referrals to various files.

GitHub provides some useful graphs and data from which you can deduct the information you require.

Graphs

Graphs can be accessed through the right sidebar of a repository:

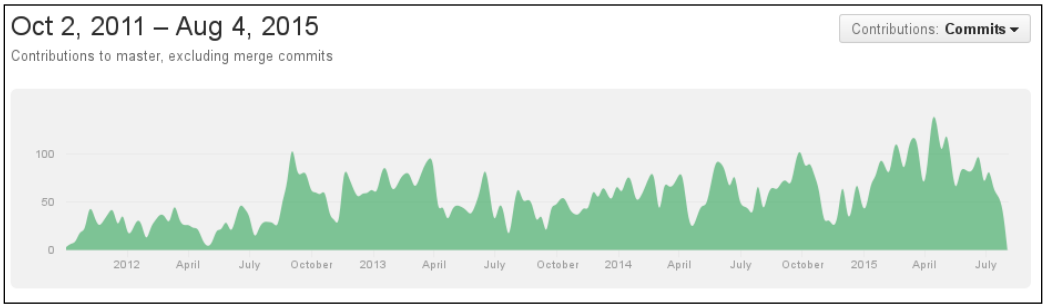


To better understand how this works, pick up a project with high traffic; for example, `https://github.com/twbs/bootstrap/graphs`. The default tab is the **Contributors** tab and also the one we will explore next.

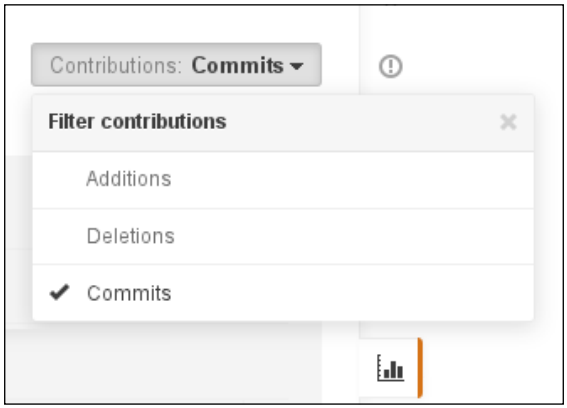
www.it-ebooks.info

Contributors – additions/deletions

An overview of the top 100 contributors of a project can be seen at the **Contributors** tab. The graph is created by the data of the default branch of a repository and it depicts the commits from the beginning of the project until the current day:

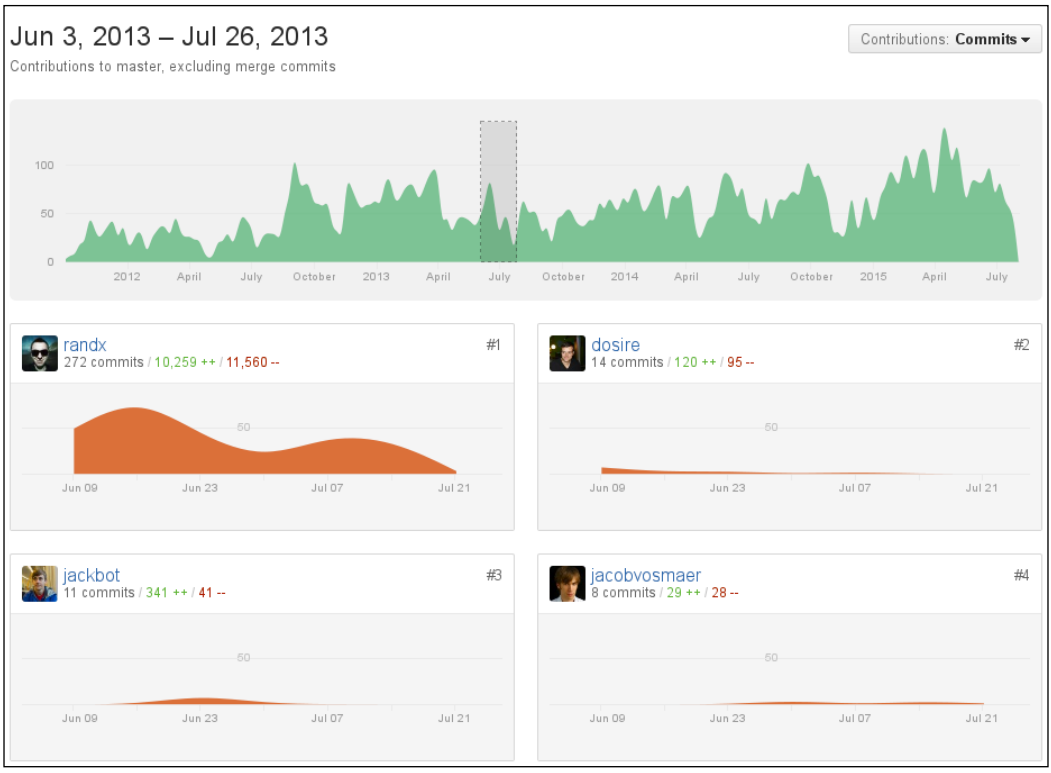


At the bottom there are the statistics of the total number of commits, along with the line additions and deletions of each of the top 100 contributors. The default filter is the commit count. If you want to change and see who made the most additions or deletions, you can toggle the filter with the **Filter contributions** drop-down menu on the right:



www.it-ebooks.info

The data can be even more fine-grained, by choosing a specific period from the graph by selecting an area. For example, I visited the `https://github.com/gitlabhq/gitlabhq/graphs/contributors` and I wanted to see the contributions data near July 2013:



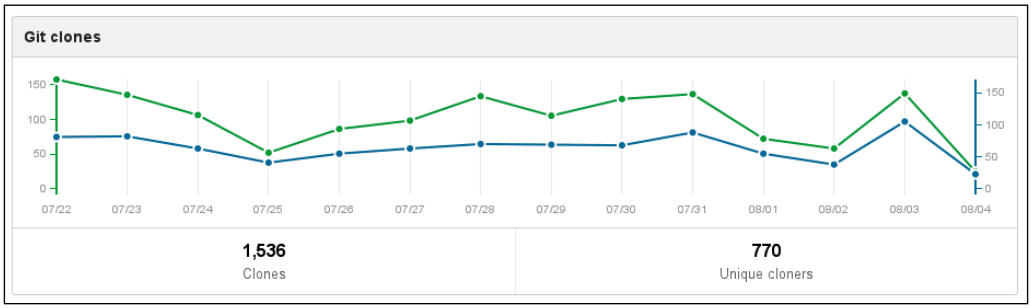
The graph does not include merge commits or commits with zero changes.

www.it-ebooks.info

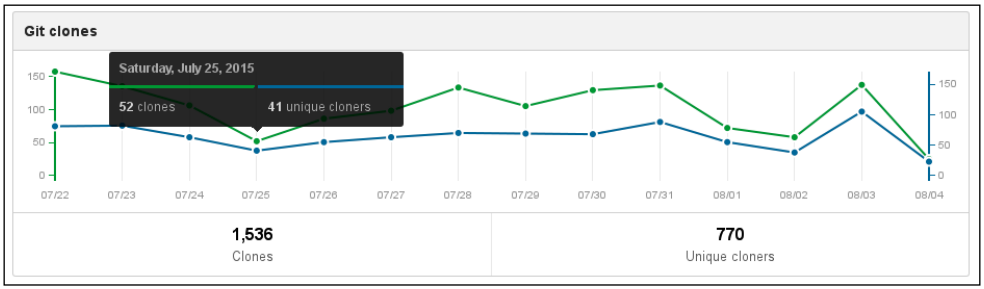
See a repository's traffic – visitors, clones, and popular content

This tab can only be seen by project owners or team members. For our example, I chose `https://github.com/gitlabhq/gitlabhq/` as this was the only big project I had access to each data.

The higher the traffic a repository has, the more data there is to explore. In the **Traffic** tab, there is information about the clones a repository had during a period of two weeks. In the following screenshot, you can see that the repository was cloned **1536** by **770** unique people between **07/22** and **08/04**. This means that on an average, in these 14 days, 770 people cloned the repository twice:

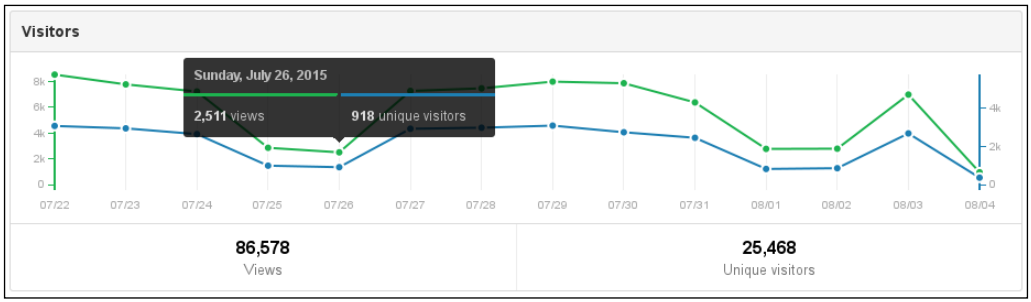


By hovering the mouse on the bullets, you can clearly see the clones and unique cloners as GitHub names them, per day. In the following example, on that particular day, almost all clones were made by different people:



www.it-ebooks.info

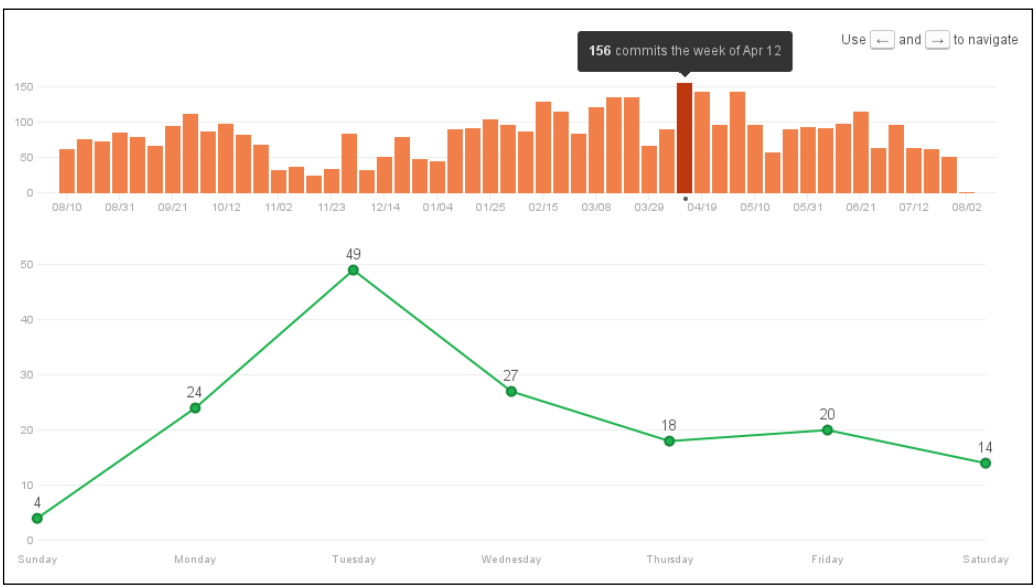
Likewise, you can see the total views for the last two weeks as well as how many unique visitors your repository had:



Right below these graphs, there are the referring sites and the popular content. Clicking on a site will take you to another page where the actual link appears. Search engines and GitHub's own search are excluded.

Commits over time

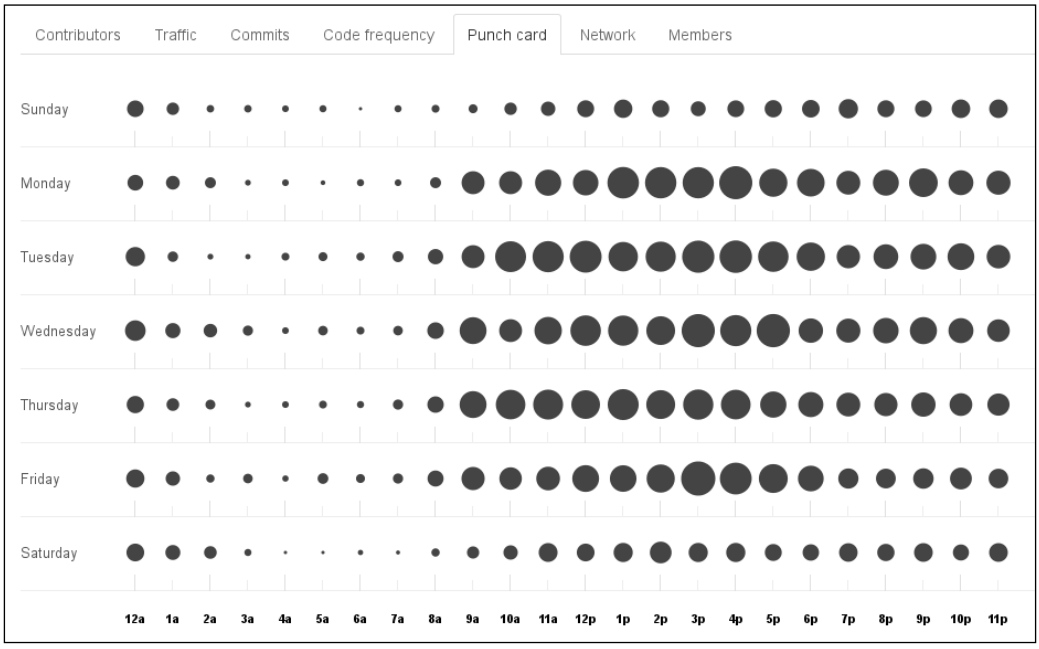
The **Commits** tab shows the commit activity during the period of the last year. In a nice bar graph, you can visualize the number of commits per week; if you click on one, another graph appears below that shows the number of commits per day of that particular week:



www.it-ebooks.info

Frequency of updates

The **Punch card** tab is interesting as it shows the weekly activity of your project based on the hour of the day. This way, you can see what time the project is more active. As per the GitHub documentation, the hours are in the time zone UTC+0. The following graph shows that most commits are done at noon and one can assume that most contributors are from Europe:

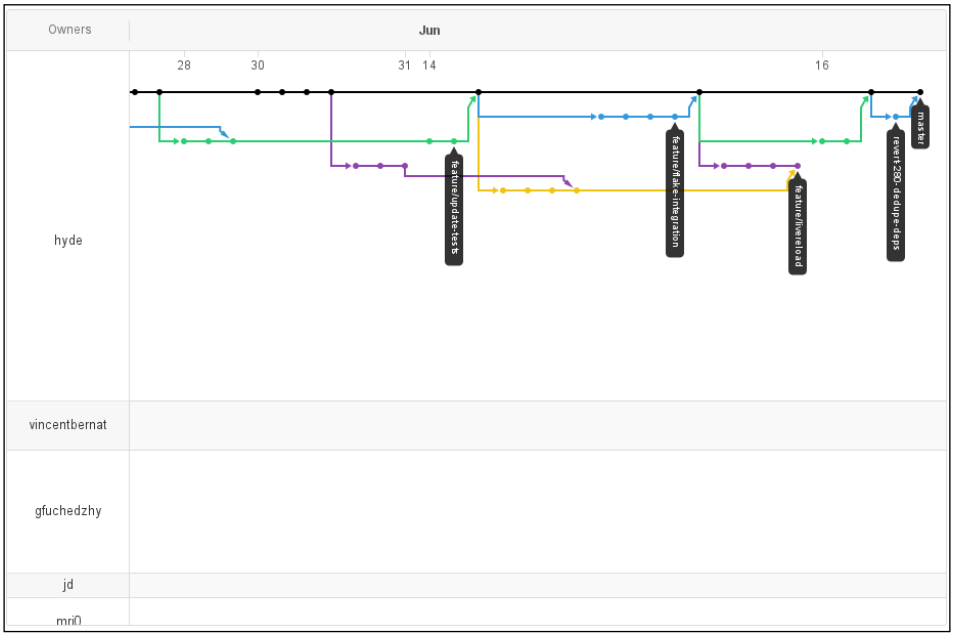


Network

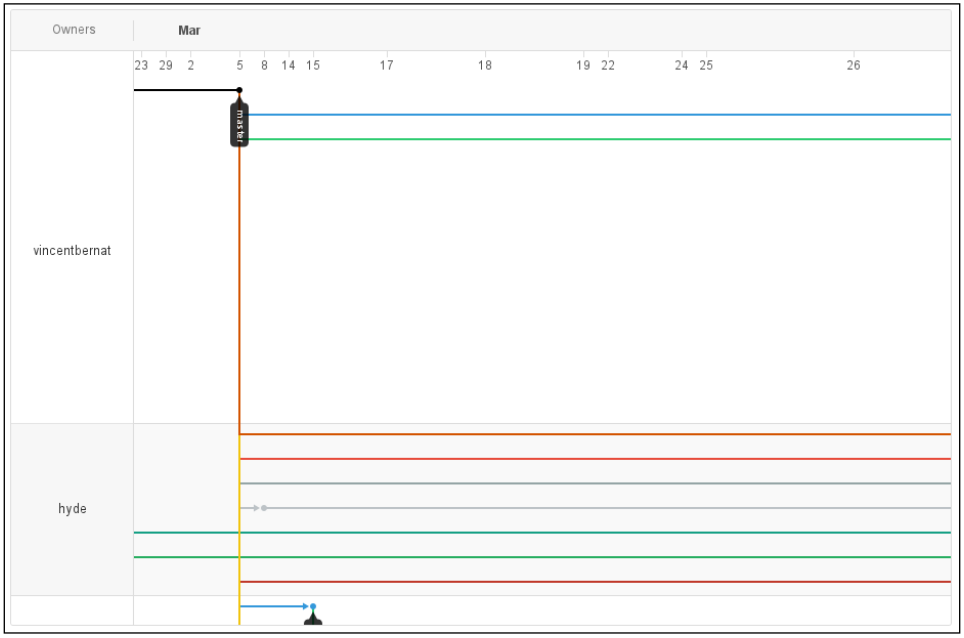
The **Network** graph shows the branch history of the main repository as well as its forks. You can click and drag the graph or use your keyboard arrows to see the older history. To view how another fork deviates from its parent, click on the owner name and you will be transferred to that repository network graph.

www.it-ebooks.info

For example, see the **Network** graph at `https://github.com/hyde/hyde/network`:



Here is one of graph of its forks:



www.it-ebooks.info

Finally, you can click on the little bullets and you will be transferred to that particular commit.

If a project has many forks, GitHub will not be able to render the **Network** graph.

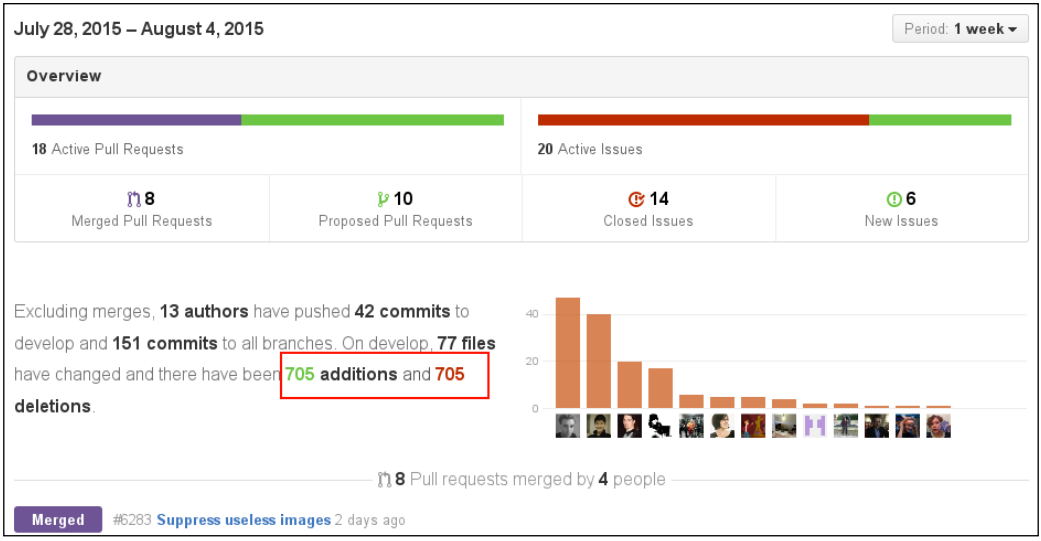
Members

The title might be misleading, since all this tab shows is the people who have forked the repository. It has nothing to do with the project collaborators.

Pulse

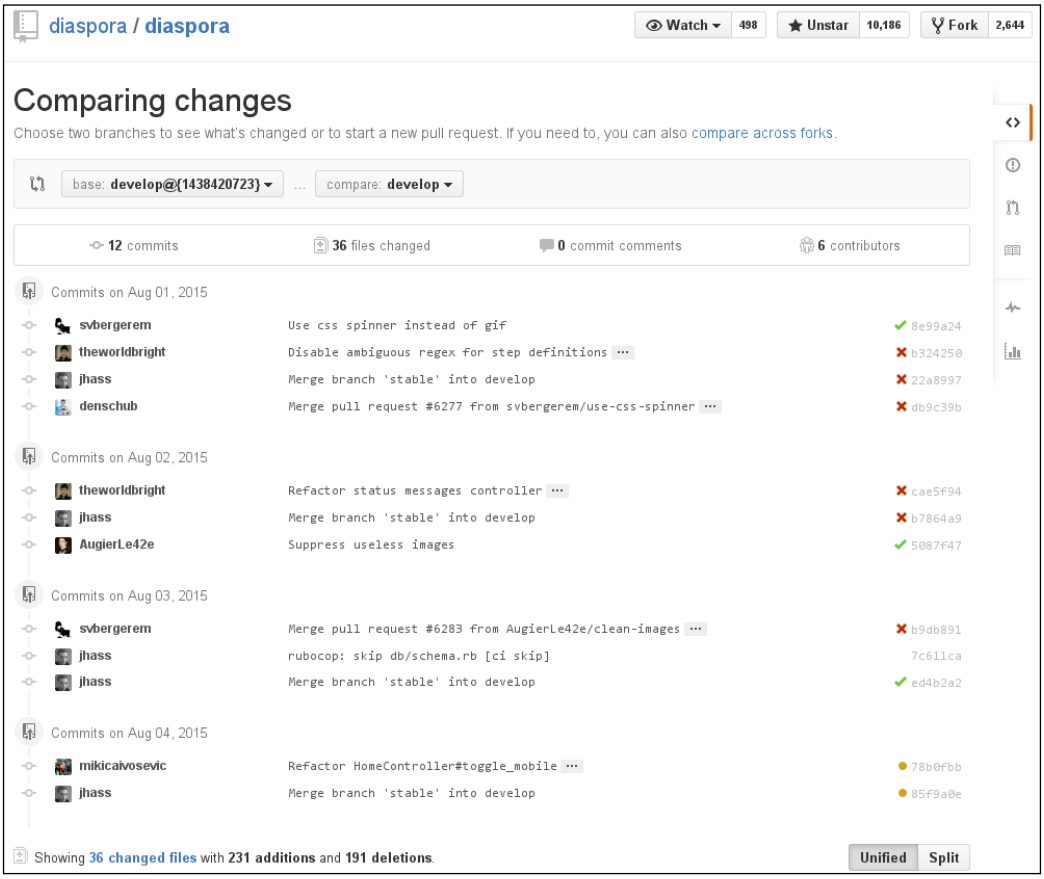
Pulse is an overview of a repository's activity. The default is to show the last 7 days. You can change the period from the drop-down menu on the right, by choosing 24 hours, 3 days, 1 week, or 1 month.

From here, you have a high overview of the merged and open pull requests, open and closed issues, as well as the top committers for that period:



www.it-ebooks.info

The interesting part is that you can click on the link that I put in red in the preceding screenshot and do a commit comparison for the period of time you have chosen in the overview, as shown in the following screenshot:



Tips and tricks

The next tip is to use some advanced techniques that use the GitHub API.

www.it-ebooks.info

Making use of pages metadata with Jekyll

GitHub provides some metadata when using Jekyll for GitHub Pages. This means that you can add certain keywords in the Jekyll templates and these will be rendered automatically.

For example, you could add the `{{ site.github.project_title }}` variable, and the project title would be filled by GitHub automatically.

Following the example in the *Introduction to Jekyll* section of this chapter, we will add a new post to the Jekyll site. Firstly, head over the repository directory and make sure you are in the `gh-pages` branch:

```
git checkout gh-pages
```

We will copy the default post to have it as a reference (the dates in your site will differ):

```
cp _posts/2015-08-03-welcome-to-jekyll.markdown _posts/2015-08-04-
  testing-github-metadata-with-jekyll.markdown
```

Open the new file and remove all contents, except for the following:

```
---
layout: post
title:  "Welcome to Jekyll!"
date:   2015-08-03 00:53:58
categories: jekyll update
---
```

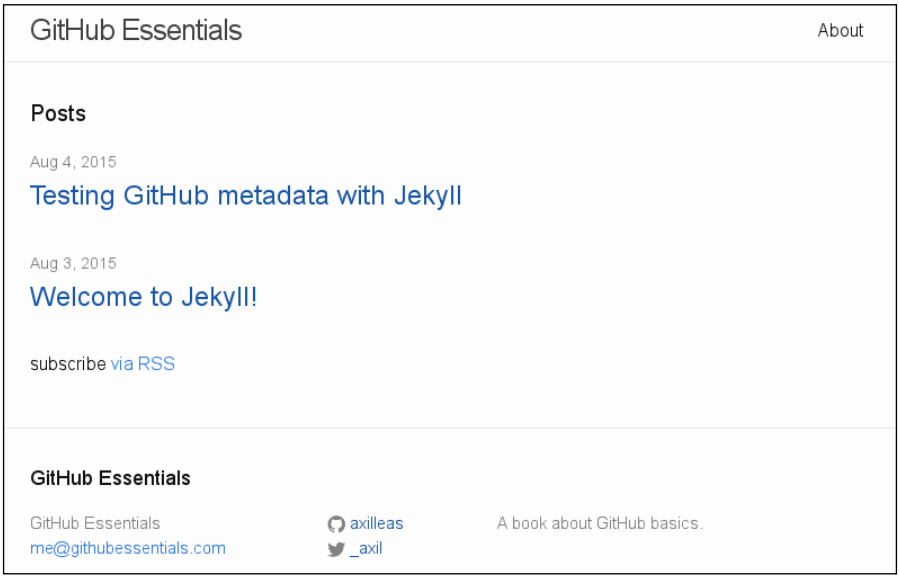
Then, edit it to look as follows:

```
---
layout: post
title:  "Testing GitHub metadata with Jekyll"
date:   2015-08-04 00:00:00
categories: jekyll github
---

The name of this project is {{ site.github.project_title }} and owned
by
{{site.github.owner_name}}.
```


www.it-ebooks.info

The post will appear in the front page:



Its contents will have the variables rendered:

You can read more at `https://help.github.com/articles/repository-metadata-on-github-pages/`.

www.it-ebooks.info

Summary

In this chapter, we learned the purpose of GitHub Pages and the various ways to upload your content. A quick introduction to Jekyll will hopefully set up the base for further reading and usage of this cool static site generator.

We also ran through the various visualizations that GitHub provides with its graphs and other tools, such as pulse, that are part of every repository.

In the next and final chapter, we will explore the user's and repository's settings.

www.it-ebooks.info

6
Exploring the User and Repository Settings

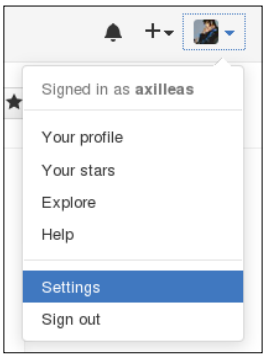
In this chapter, we will explore the most common and essential settings of a user and a repository.

As a user, there is a lot of information you can set up in your user settings page, such as associating more than one e-mail to your account, adding multiple SSH keys, or setting up two-factor authentication.

Similarly, some functionalities of a repository can be set up via its settings page. For example, you can enable or disable the wiki pages, or completely disable the issue tracker.

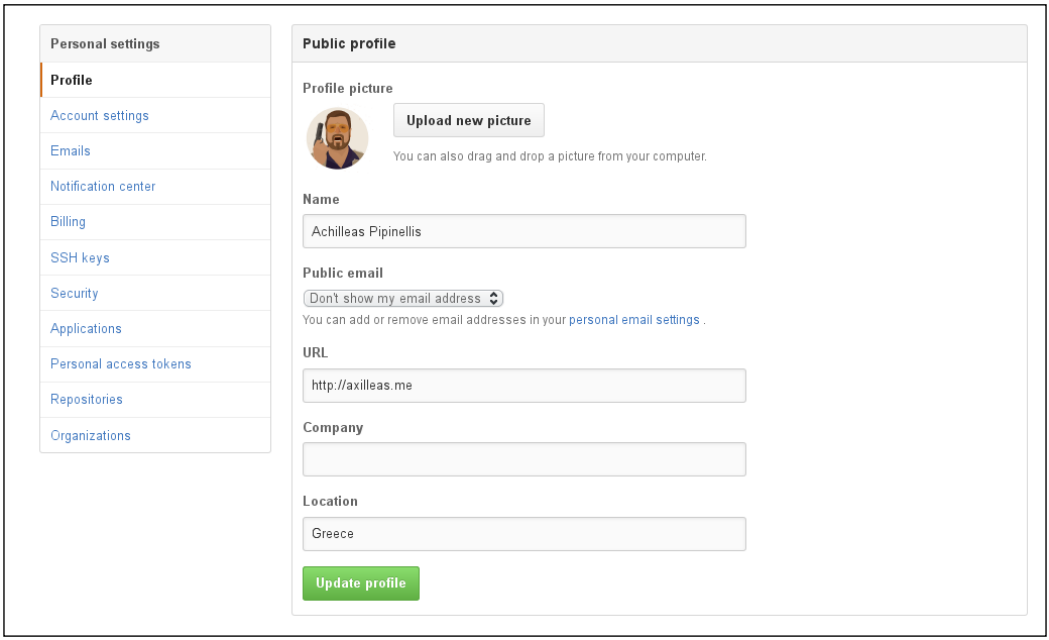
User settings

You can visit your user's settings page by navigating through the web interface (under your avatar's drop-down list) or visit `https://github.com/settings/profile` directly:



www.it-ebooks.info

For example, here is how my settings landing page looks like:



We are going to analyze the most important settings GitHub provides.

Profile

Under **Personal settings**, you can see the various options that you can customize to your liking.

The **Profile** settings is where you can fill in your personal information so that people know who you are. Consider it like socialization. After all, GitHub is the Facebook of geeks.

All the profile information is optional to fill in. You can see how this will look by visiting your username page at `https://github.com/<username>`.

www.it-ebooks.info

Setting up multiple e-mails

Every commit is associated with an e-mail address. GitHub's documentation states that:

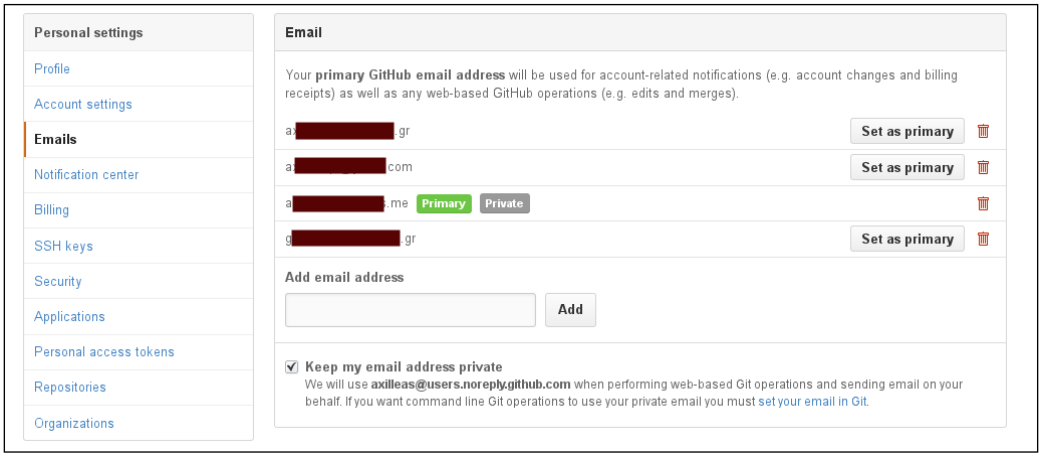
> *GitHub uses the email address you set in your local Git configuration to associate commits with your GitHub account.*

There is no limit to the e-mails that you add to your account, but you can have only one primary address. This is where GitHub will send you any notifications and it is this address that will be used when editing and committing files via the web interface.

You can add or remove e-mails and change the primary address by visiting `https://github.com/settings/emails`.

In this area, you can also choose whether or not your primary e-mail will be shown to the public.

In the following screenshot, you can see how that page looks like when you have multiple e-mails:



www.it-ebooks.info

Managing your SSH keys

GitHub provides two ways to authenticate a user when using Git. You can use Git over HTTP or Git over SSH. For a detailed explanation of the Git protocols, visit `https://git-scm.com/book/ch4-1.html`.

When using Git over HTTP, you must provide your username and password each time you make a change, unless you cache your GitHub password in Git. For more details, see the article at `https://help.github.com/articles/caching-your-github-password-in-git/`.

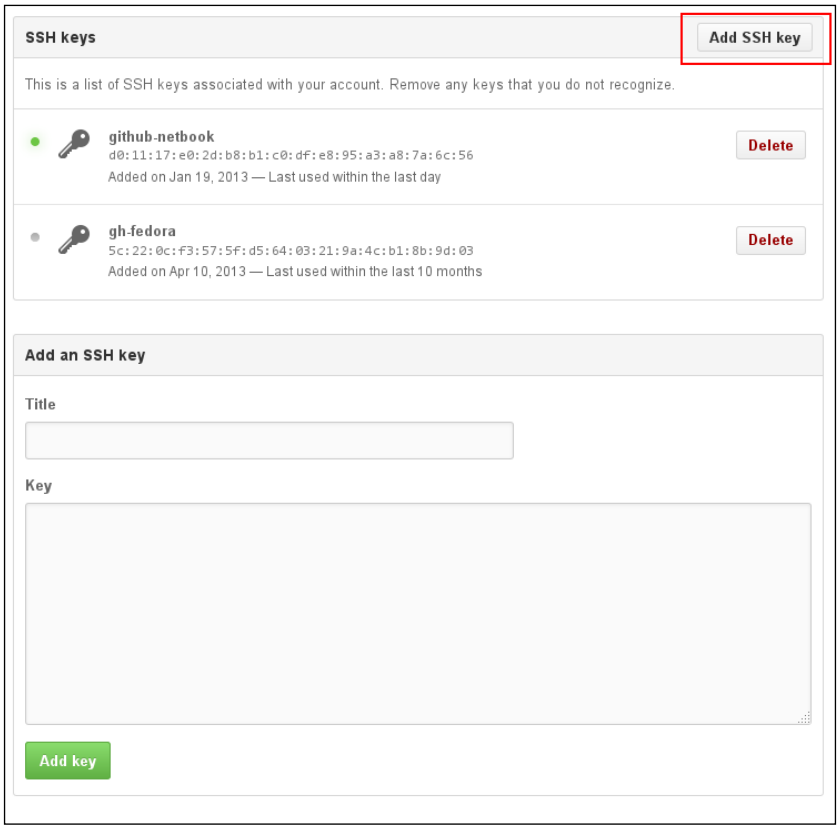
The preferred and more secure way is to use Git over SSH. The concept is that you create a personal unique SSH key pair whose public key you upload to your GitHub profile. You can repeat this process for as many times as you wish since GitHub allows you to have multiple SSH keys associated to your account. This way, you can have one key to use with your laptop and a different one with your desktop or your server.

In order to use Git over SSH, the remote-URL of the repository must look like `git@github.com:USERNAME/REPOSITORY.git`.

Under your **Settings** tab, there is the **SSH keys** option. Either navigate through the GitHub UI or go directly to `https://github.com/settings/ssh`.

www.it-ebooks.info

Here is an example of how it looks like to have two keys in your account and to add a third one:



www.it-ebooks.info

When you add an SSH public key, you give it a **Title** so that you can remember where this key came from. At the **Key** area, you paste the contents of the public key. As you can see, GitHub also provides some useful information such as the fingerprint of the key, when it was added, and when it was last used.

You cannot edit a key. If you want to set a different title, you will have to delete the old key and add it back again.

Setting up two-factor authentication

Setting up two-factor authentication provides an extra layer of security for your account. Using only the password to sign in can prove susceptible to security threats, since the attacker only needs a single piece of information.

This is done by having an extra authentication code generated on to your cell phone or tablet. In the case of a smartphone, you must install an application that can handle the **Time-based One-Time Password** (**TOTP**) technology. If you are looking for an open source application, go to `https://fedorahosted.org/freeotp/`.

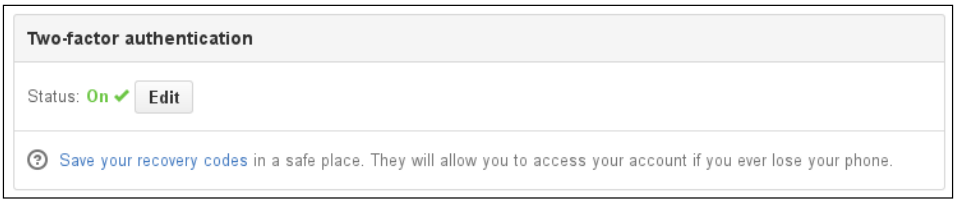
For a list of supported applications, you can read Wikipedia's article at `https://en.wikipedia.org/wiki/Time-based_Onetime_Password_Algorithm#Client_implementations`.

In case you do not own a smartphone, GitHub can also send the authentication code with an SMS. Since this involves delivery rates, there is a finite list of supported countries. See if yours is included here: `https://help.github.com/articles/countries-where-sms-authentication-is-supported/#supported-countries-for-sms-authentication`.

You can enable **Two-Factor Authentication** (**2FA**) under the **Security** page. Visit `https://github.com/settings/security` directly and press the **Set up two-factor authentication** button to start setting up 2FA. Pick either one of the methods and follow the on-screen instructions.

www.it-ebooks.info

After setting up 2FA, if you visit the security page, under your settings, you will see that the 2FA is enabled:



On the **Security** page, make sure to click the link **Save your recovery codes** and follow the on screen instructions to download and save the recovery codes in a secure place. There are 16 codes that will help you gain access to your account if, for some reason, you lose your cell phone or it gets stolen. Every recovery code can be used only once and you can generate a new batch by clicking on **Generate new recovery codes**. Keep them safe; preferably store them encrypted in an application such as **KeepassX**.

Now, every time you try to log in to GitHub from a browser for the first time, apart from the usual credentials username and password, you will be prompted to give the authorization code generated from the application of your smartphone or one of the 16 recovery codes.

Repository settings

There are quite a few settings one can fiddle with at the repository level. To access these settings, search for the wrench icon on the right sidebar:

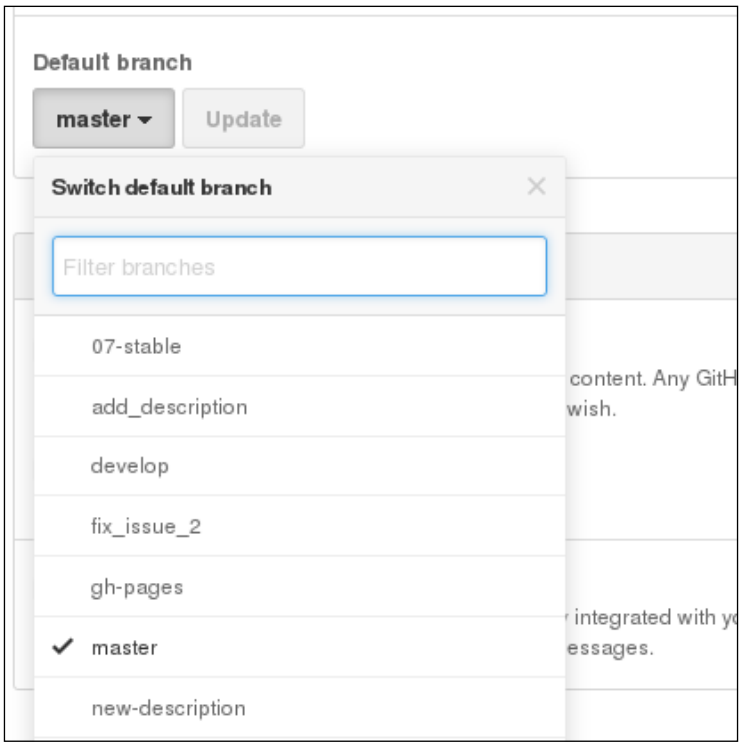
www.it-ebooks.info

Changing the default branch that appears in repository's main page

The default branch of a repository's main page is master. However, there are times when you want a different branch to be your default, based on one's workflow as we saw in *Chapter 4*, *Collaboration Using the GitHub Workflow*.

Let's just say, for example, that the **master** branch is where you push code that is considered stable and well-tested, whereas you have a different branch named **develop** that is used for daily pushes and testing new features. Based on this assumption, the **develop** branch gets updated more often than the **master** branch. Practically, you'd want your project to seem active; having a branch that gets updated every day in the front page is much more appealing.

In this case, you can go to the repository's settings page and choose the branch you would like to have as the default from the drop-down list:



www.it-ebooks.info

When someone clones the repository for the first time, Git checks out the default branch that is set through the project's settings.

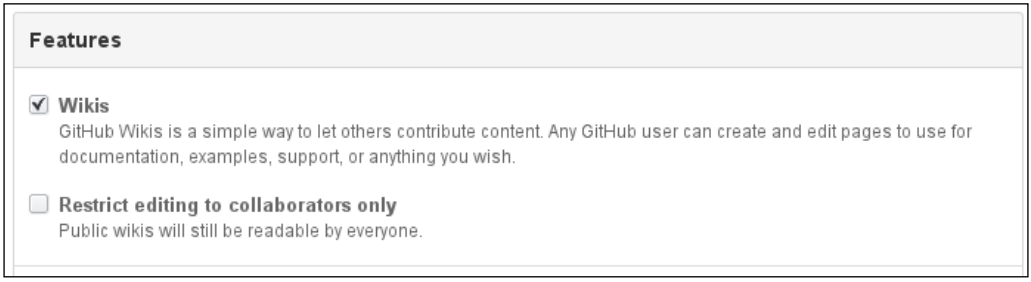
Enabling/disabling the wiki

In *Chapter 2*, *Using the Wiki and Managing Code Versioning*, we explained in depth why the wiki is a strong asset for a project. There are, however, cases where one does not need a wiki; for example, you might use an external one.

GitHub provides you with three options regarding the visibility of a wiki:

- Enable the wiki and make it public so that everyone has write access (default)
- Enable the wiki but only owners and collaborators have write access
- Disable the wiki altogether

The first behavior is the default one. You can find these settings under the **Features** block:

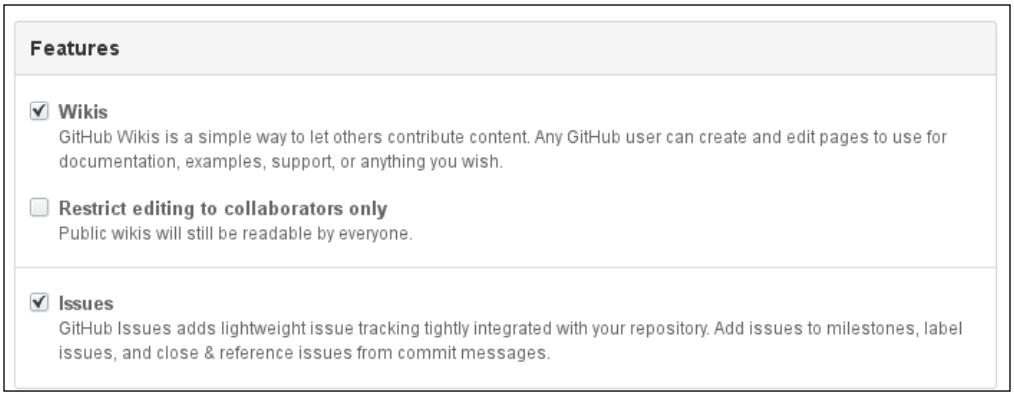


Enabling/disabling the issue tracker

Although GitHub's issue tracker is a powerful tool for collaboration and bug reporting, there are times where you would like to use a different tracker such as Redmine, Jira, or Bugzilla.

www.it-ebooks.info

In this case, GitHub's issue tracker can be disabled so that you don't have many places to track and lose control. This can be achieved in the repository's settings page under the **Features** block:



Any created issues are retained even after you disable the **Issues** feature. Get the tick back and your issue tracker will be the same as before.

Adding collaborators

By adding a collaborator, you are granting push access to the repository.
A repository can have many collaborators; there is no limit.

Visit the **Collaborators** tab under settings. Start typing the name of the user and the autocompletion is smart enough to present you with the user you are searching for:

www.it-ebooks.info

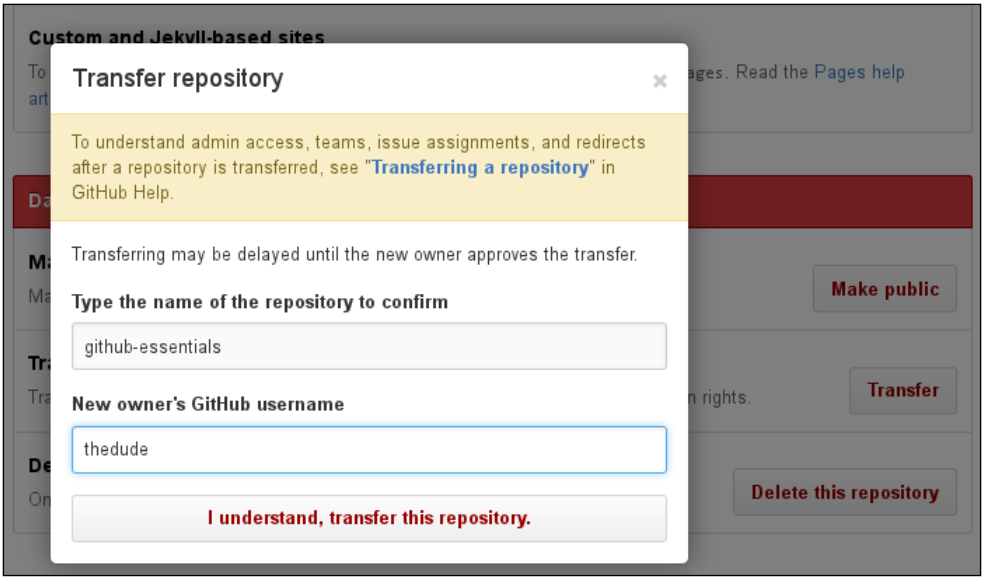
Transferring ownership – user to organization

Every repository is created under a namespace, be it a user or an organization. In the rare case where you would like to transfer a repository to another user, this can be done in the repository settings. Since this action is considered dangerous, you will find this setting inside a red code block signifying the importance of the task.

Basically, there are four types of transfer:

- user to user
- user to organization
- organization to user
- organization to organization

In order to initialize the transfer, you have to provide the name of the repository and the username of the new owner to confirm. This can be a user or an organization:



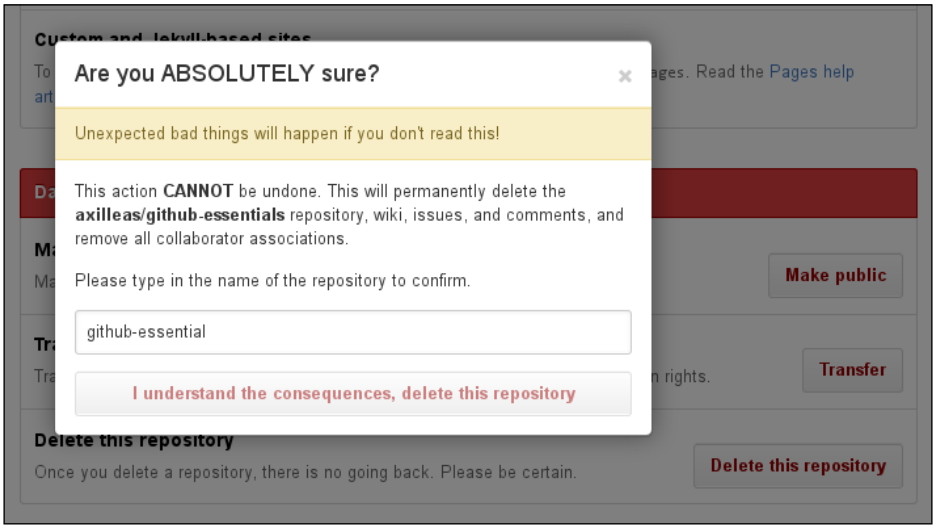
After you hit **I understand, transfer this repository**, an e-mail will be sent to the new owner(s) for confirmation. After they confirm, the procedure will be completed.

www.it-ebooks.info

Deleting a repository

You can delete a repository along with all its settings by pressing the **Delete this repository** button under **Danger Zone**. Bare in mind that this is a destructive action that will purge your repository, the issue tracker, any pull requests, the wiki, and in general everything related to it.

After pressing the holocaust button, you will presented with a modal asking for confirmation. For security reasons, you must provide the repository's name to confirm. In the following screenshot, you can see that unless you provide the right name, the button for deletion will be inactive:



Tips and tricks

Did you know that you could use different e-mail addresses depending on the organization you are member of? Are you aware of how much disk space your repository takes? If no, read below and learn how to perform these actions.

Finding the size of your repositories

If you are curious to know how big your repositories have become, you can visit `https://github.com/settings/repositories` and see it for yourself. Remember that GitHub also counts the size of the `.git` directory, so if you have thousands of commits, the repository size will be greater than its actual size (by actual size, I mean whatever files you see on GitHub).

www.it-ebooks.info

For example, the `diaspora` repository at the time of this writing seems to be **101.17MB**:

If I were to remove the `.git` directory, the size would be much smaller. Let's test it by using the following commands:

```
git clone https://github.com/diaspora/diaspora
du -sh diaspora
rm -rf diaspora/.git
du -sh diaspora
```

Removing the `.git` directory gets the size down to almost 14 MB!

Fine-tuning e-mail notifications

If you are a member of many organizations, you may want to use a different e-mail for notifications concerning the repository that a specific organization owns. You can achieve this by going to `https://github.com/settings/notifications`; under **Notification email** | **Custom routes**, choose the e-mail you want to receive the notifications to.

Summary

By finishing this chapter, you should be ready to fill in the details to build a public profile that is viewable by anyone interested to know more about you. Your account is yours and yours only, so it should be secured as much as possible. By now, you should have followed the steps to secure it with 2FA and be a little bit safer.

You have also learned how to configure repository's settings regarding its default branch and enabling or disabling features such as the issue tracker and the wiki. Another thing to remember is how to add collaborators to your project and how to transfer its ownership if needed.

These were the most important settings to consider, both user and project-related, and you should feel a little bit wiser towards the end.

www.it-ebooks.info

www.it-ebooks.info

Index

A

atom feed
releases, subscribing via 50

B

branches and pull requests connection
about 98
fork and pull model 98
shared repository model 98

C

code versioning
draft release, making 48
files, uploading 48, 49
managing 41
prerelease, marking 46
release, creating 41-45
release, editing 45
tag, pushing from command line 46
Contributors tab, graphs
about 140, 141
additions 140
deletions 140

F

Fully Qualified Domain Name (FQDN) 132

G

GitHub API
advanced techniques, for usage 147
GitHub documentation
references 132
github-essentials
reference 128
GitHub Pages
about 125
custom domain, using 132
for project pages 126
for users and organizations 126
organization page, creating 127
page, customizing with Jekyll 133
project page, creating manually 127, 128
project page, creating with GitHub page generator 128-131
project page, updating with GitHub page generator 132
reference 125
user page, creating 127
GitHub web editor
fork and pull model 105, 106
shared repository model 104
using 103, 104
GitHub workflow
reference 97
global member privileges 58-60
gollum library 50
graphs
about 139
Commits tab 143
Contributors tab 140, 141
reference 139
repository's traffic 142, 143

I

issue tracker
about 13
benefits 13

www.it-ebooks.info

issue, creating 14-17
issues, assigning to users 18, 19
keyboard shortcuts, navigating with 28
labels 20
milestones 24
tips and tricks 27

J

Jekyll
about 135-138
installing 133, 134
references 138
used, for customizing GitHub Page 133

K

KeepassX 157

L

labels, issue tracker
benefits 20
colors, setting 20-23
for grouping issues 24
label names, creating 20-23

M

milestones, issue tracker
about 24
benefits 24
creating 25, 26
issues, adding 26
for overview of what is resolved and what is not 27

N

Network graph
about 144
reference 145, 146

O

organization
creating 55-58
roles 54
organization settings
about 83
audit log 86, 87
organization feed, in dashboard 95
profile 84, 85
project, transferring to organization's namespace 88-90
team privacy 85
teams, mentioning 94, 95
third-party access 86
tips and tricks 88
user account, converting into organization 91-93

P

pages metadata
using, with Jekyll 148, 149
People tab
about 74, 75
access levels, managing 76-78
members, inviting 82
members, versus outside collaborators 79, 80
outside collaborator, demoting 81, 82
pull requests
about 97
benefits 97
branch, re-pushing 117
compare function, using 102, 103
Compare & pull request button, using 99-102
connection with branches 98
creating 99
GitHub web editor, using 103
inline comments 108, 113-115
layout 109-112

www.it-ebooks.info

merging 118
overview 116
peer review 108
reverting 119
submitting 107, 108
tips and tricks 119
pull requests, merging
branch, removing 118, 119
branch, restoring 118, 119
pulse 146, 147

R

README file 27, 28
Repositories tab 60-62
repository
Blame button 10
branches page 7-9
commits page 4-6
creating 2, 3
description, changing 12
Fork button 12
History button 11
main page 1
Raw button 9
Star button 12
Watch button 11
repository permission levels 54
repository settings
about 157
collaborators, adding 160
default branch, changing 158
issue tracker, disabling 159
issue tracker, enabling 160
ownership, transferring 161
repository, deleting 162
wiki, disabling 159
wiki, enabling 159

S

static site generators
URL 133

T

team
about 62
creating 62-64
invitation, accepting 68, 69
people, inviting 65-67
repositories, adding 72, 74
team members permissions
about 69-71
owner 72
team maintainer 72
user 71
Time-based One-Time Password (TOTP) technology 156
tips and tricks, pull requests
diff, downloading 123
global list, of open pull requests 123
issues, closing via commit messages 119-121
LICENSE file, adding 123, 124
new directories, creating with web editor 124
task lists 121, 122
tips and tricks, repository
about 162
e-mail notifications, fine-tuning 163
size of repositories, determining 162, 163
Two-Factor Authentication (2FA) 156

U

users and organizations
difference 53
user settings
about 151, 152

www.it-ebooks.info

multiple e-mails, setting up 153
Profile settings 152
SSH keys, managing 154-156
two-factor authentication, setting up 156

V

various level permissions
references 55

W

web analytics
about 139
frequency of updates 144
graphs 139
members 146
Network graph 144
pulse 146
wiki
benefits 30
changes, making locally 51
changes, pushing to GitHub 51
cloning 50, 51
commit history 38-40
footer, adding 35-37
Markdown-powered wiki 33-35
new wiki page, creating 30-32
page, deleting 32
sidebar, adding 35-37
using 29
wiki, editing locally
about 50
gollum, installing 50

www.it-ebooks.info

Thank you for buying
GitHub Essentials

About Packt Publishing

Packt, pronounced 'packed', published its first book, *Mastering phpMyAdmin for Effective MySQL Management*, in April 2004, and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution-based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern yet unique publishing company that focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website at `www.packtpub.com`.

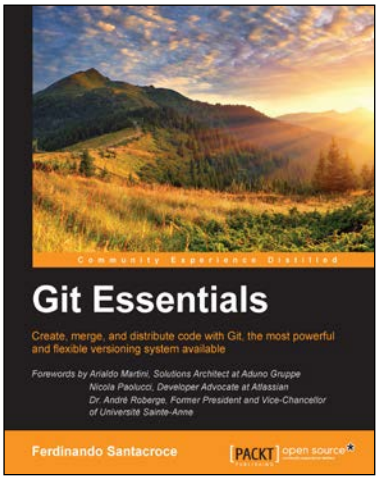
Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to `author@packtpub.com`. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, then please contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

www.it-ebooks.info

[PACKT]
PUBLISHING

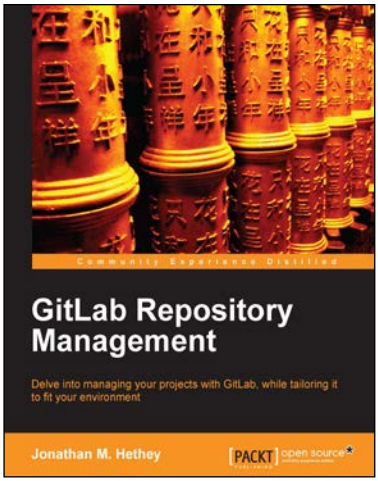


Git Essentials

ISBN: 978-1-78528-790-9 Paperback: 168 pages

Create, merge, and distribute code with Git, the most powerful and flexible versioning system available

1. Master all the basic concepts of Git to protect your code and make it easier to evolve.
2. Use Git proficiently, and learn how to resolve day-by-day tasks easily.
3. A step-by-step guide, packed with examples to help you learn and work with Git internals.



GitLab Repository Management

ISBN: 978-1-78328-179-4 Paperback: 88 pages

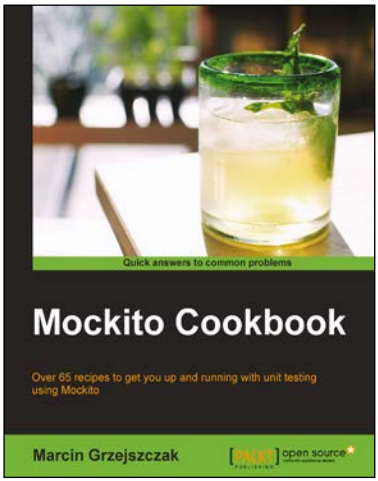
Delve into managing your projects with GitLab, while tailoring it to fit your environment

1. Understand how to efficiently track and manage projects.
2. Establish teams with a fast software developing tool.
3. Employ teams constructively in a GitLab environment.

Please check **www.PacktPub.com** for information on our titles

www.it-ebooks.info

[PACKT]
PUBLISHING

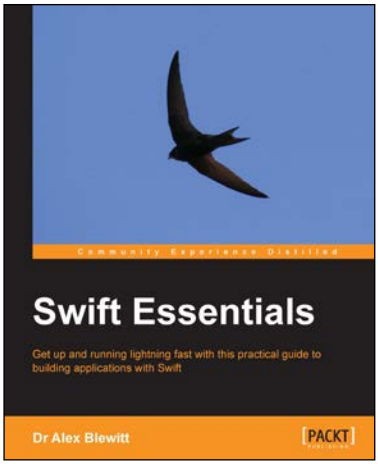


Mockito Cookbook

ISBN: 978-1-78398-274-5 Paperback: 284 pages

Over 65 recipes to get you up and running with unit testing using Mockito

1. Implement best practices to perform tests with Mockito.
2. Extend Mockito with other popular Java-based unit testing frameworks such as JUnit and Powermock.
3. A focused guide with many recipes on testing your software using Mockito.



Swift Essentials

ISBN: 978-1-78439-670-1 Paperback: 228 pages

Get up and running lightning fast with this practical guide to building applications with Swift

1. Rapidly learn how to program Apple's newest programming language, Swift, from the basics through to working applications.
2. Create graphical iOS applications using Xcode and storyboard.
3. Build a network client for GitHub repositories, with full source code on GitHub.

Please check **www.PacktPub.com** for information on our titles

www.it-ebooks.info