



learn  
enough

# DEVELOPER TOOLS TO BE DANGEROUS

COMMAND LINE,  
TEXT EDITOR, AND GIT  
VERSION CONTROL  
ESSENTIALS



MICHAEL HARTL

FREE SAMPLE CHAPTER



# Praise for Learn Enough Tutorials

---

“Going through *Learn Enough Git* is wonderful. I am actually learning... I’ve done three other Git tutorials and still felt so lost. Doing it all now makes so much sense. It’s like a light bulb.”

—Janelle Staar

“I bought the *Learn Enough Command Line to Be Dangerous* last fall, and it’s paid off sooooo many times in my new job. During my first week, I had a manager sitting right beside me giving me the ‘go here, go there, do this, etc.’ Having watched, read, and done the exercises, I was confident in getting around the CLI [command-line interface]—and even had him asking, ‘What was that shortcut?’ For this, I thank you. Now I need a ‘Learn even more CLI to be dangerouser.’”

—Thomas Thackery

“I must say, this Learn Enough series is a masterpiece of education. Thank you for this incredible work!”

—Michael King

“I want to thank you for the amazing job you have done with the tutorials. They are likely the best tutorials I have ever read.”

—Pedro Iatzky

*This page intentionally left blank*

LEARN ENOUGH  
DEVELOPER TOOLS  
TO BE DANGEROUS

# Learn Enough Series from Michael Hartl



Visit [informit.com/learn-enough](http://informit.com/learn-enough) for a complete list of available publications.

The **Learn Enough** series teaches you the developer tools, Web technologies, and programming skills needed to launch your own applications, get a job as a programmer, and maybe even start a company of your own. Along the way, you'll learn technical sophistication, which is the ability to solve technical problems yourself. And Learn Enough always focuses on the most important parts of each subject, so you don't have to learn everything to get started—you just have to learn enough to be dangerous. The Learn Enough series includes books and video courses so you get to choose the learning style that works best for you.

# LEARN ENOUGH DEVELOPER TOOLS TO BE DANGEROUS

---

Command Line, Text Editor, and  
Git Version Control Essentials

Michael Hartl

◆◆ Addison-Wesley

Boston • Columbus • New York • San Francisco • Amsterdam • Cape Town

Dubai • London • Madrid • Milan • Munich • Paris • Montreal • Toronto • Delhi • Mexico City

São Paulo • Sydney • Hong Kong • Seoul • Singapore • Taipei • Tokyo

Cover image: Philipp Tur/Shutterstock

Figures 1.1, 1.2, 1.4, 1.6-1.8, 5.4, 5.6-5.9, 5.11, 10.9, 11.8, 11.10, 11.13, A.1: Screenshot © 1995-2021 The Open Group

Figure 1.3: Screenshot © 2021 The Linux Foundation

Figures 3.3, 7.31, 7.32: Screenshot © Regex101

Figures 5.1, 6.1-6.3, 6.10-6.22, 6.24-6.36, 7.1-7.30, 7.33, 7.34, 7.39, 8.3-8.8, 9.1-9.12, 10.3, 11.3-11.7, 11.15-11.17, 11.22-11.24, 11.26: Screenshot © 2021 GitHub, Inc.

Figures 6.4-6.9, 7.35, A.2-A.7: Screenshot © 2021, Amazon Web Services, Inc.

Figures 7.36-7.38: Screenshot © 2020 Wbond

Figure 8.2: Screenshot © 2021 Apple Inc.

Figures 10.2, 10.6, 11.25: Photo of whale, GUDKOV ANDREY/Shutterstock

Figures 11.12, 11.14, 11.19, 11.21, 11.27: Photo of polar bear, Vaclav Sebek/Shutterstock

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at [corpsales@pearsoned.com](mailto:corpsales@pearsoned.com) or (800) 382-3419.

For government sales inquiries, please contact [governmentsales@pearsoned.com](mailto:governmentsales@pearsoned.com).

For questions about sales outside the U.S., please contact [intlcs@pearson.com](mailto:intlcs@pearson.com).

Visit us on the Web: [informit.com/aw](http://informit.com/aw)

Library of Congress Control Number: 2022930143

Copyright © 2022 Softcover Inc.

All rights reserved. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, request forms and the appropriate contacts within the Pearson Education Global Rights & Permissions Department, please visit [www.pearson.com/permissions](http://www.pearson.com/permissions).

ISBN-13: 978-0-13-784345-9

ISBN-10: 0-13-784345-3

**ScoutAutomatedPrintCode**

# Contents

---

Preface	xiii
About the Author	xix

## **PART I    COMMAND LINE    1**

### **Chapter 1    Basics    3**

1.1	Introduction	5
1.2	Running a Terminal	6
	1.2.1 Exercises	10
1.3	Our First Command	10
	1.3.1 Exercises	15
1.4	Man Pages	15
	1.4.1 Exercises	19
1.5	Editing the Line	20
	1.5.1 Exercises	23
1.6	Cleaning Up	23
	1.6.1 Exercises	23
1.7	Summary	24
	1.7.1 Exercises	24



**Chapter 2 Manipulating Files 25**

- 2.1 Redirecting and Appending 26
  - 2.1.1 Exercises 29
- 2.2 Listing 30
  - 2.2.1 Hidden Files 33
  - 2.2.2 Exercises 34
- 2.3 Renaming, Copying, Deleting 35
  - 2.3.1 Unix Terseness 38
  - 2.3.2 Exercises 39
- 2.4 Summary 40
  - 2.4.1 Exercises 40

**Chapter 3 Inspecting Files 43**

- 3.1 Downloading a File 43
  - 3.1.1 Exercises 45
- 3.2 Making Heads and Tails of It 46
  - 3.2.1 Wordcount and Pipes 47
  - 3.2.2 Exercises 48
- 3.3 Less Is More 49
  - 3.3.1 Exercises 51
- 3.4 Greping 52
  - 3.4.1 Exercises 57
- 3.5 Summary 58
  - 3.5.1 Exercises 58

**Chapter 4 Directories 61**

- 4.1 Directory Structure 61
  - 4.1.1 Exercises 64
- 4.2 Making Directories 64
  - 4.2.1 Exercises 66
- 4.3 Navigating Directories 66
  - 4.3.1 Exercises 69
- 4.4 Renaming, Copying, and Deleting Directories 70
  - 4.4.1 Grep Redux 73

- 4.4.2 Exercises 73
- 4.5 Summary 74
  - 4.5.1 Exercises 74
- 4.6 Conclusion 75

## **PART II TEXT EDITOR 77**

### **Chapter 5 Introduction to Text Editors 79**

- 5.1 Minimum Viable Vim 84
- 5.2 Starting Vim 85
  - 5.2.1 Exercises 89
- 5.3 Editing Small Files 89
  - 5.3.1 Exercises 91
- 5.4 Saving and Quitting Files 91
  - 5.4.1 Exercises 95
- 5.5 Deleting Content 96
  - 5.5.1 Exercises 96
- 5.6 Editing Large Files 97
  - 5.6.1 Exercises 101
- 5.7 Summary 101
  - 5.7.1 Exercises 101

### **Chapter 6 Modern Text Editors 103**

- 6.1 Choosing a Text Editor 104
  - 6.1.1 Sublime Text 104
  - 6.1.2 Visual Studio Code (VSCode) 105
  - 6.1.3 Atom 105
  - 6.1.4 Exercises 106
- 6.2 Opening 106
  - 6.2.1 Syntax Highlighting 111
  - 6.2.2 Previewing Markdown 113
  - 6.2.3 Exercises 115
- 6.3 Moving 117
  - 6.3.1 Exercises 118
- 6.4 Selecting Text 119

- 6.4.1 Selecting a Single Word 122
- 6.4.2 Selecting a Single Line 123
- 6.4.3 Selecting Multiple Lines 124
- 6.4.4 Selecting the Entire Document 124
- 6.4.5 Exercises 125
- 6.5 Cut, Copy, Paste 127
  - 6.5.1 Jumpcut 128
  - 6.5.2 Exercises 132
- 6.6 Deleting and Undoing 132
  - 6.6.1 Exercises 135
- 6.7 Saving 135
  - 6.7.1 Exercises 136
- 6.8 Finding and Replacing 138
  - 6.8.1 Exercises 143
- 6.9 Summary 143

## **Chapter 7    Advanced Text Editing    145**

- 7.1 Autocomplete and Tab Triggers 145
  - 7.1.1 Autocomplete 145
  - 7.1.2 Tab Triggers 147
  - 7.1.3 Exercises 151
- 7.2 Writing Source Code 152
  - 7.2.1 Syntax Highlighting 153
  - 7.2.2 Commenting Out 155
  - 7.2.3 Indenting and Dedenting 156
  - 7.2.4 Goto Line Number 164
  - 7.2.5 80 Columns 164
  - 7.2.6 Exercises 165
- 7.3 Writing an Executable Script 166
  - 7.3.1 Exercises 174
- 7.4 Editing Projects 175
  - 7.4.1 Fuzzy Opening 176
  - 7.4.2 Multiple Panes 179
  - 7.4.3 Global Find and Replace 181
  - 7.4.4 Exercises 187

- 7.5 Customization 188
  - 7.5.1 Exercises 189
- 7.6 Summary 191
- 7.7 Conclusion 193

## **PART III GIT 195**

### **Chapter 8 Getting Started with Git 197**

- 8.1 Installation and Setup 200
  - 8.1.1 Exercises 202
- 8.2 Initializing the Repo 203
  - 8.2.1 Exercises 204
- 8.3 Our First Commit 204
  - 8.3.1 Exercises 207
- 8.4 Viewing the Diff 208
  - 8.4.1 Exercises 209
- 8.5 Adding an HTML Tag 210
  - 8.5.1 Exercises 215
- 8.6 Adding HTML Structure 216
  - 8.6.1 Exercises 219
- 8.7 Summary 220

### **Chapter 9 Backing Up and Sharing 221**

- 9.1 Signing Up for GitHub 221
- 9.2 Remote Repo 222
  - 9.2.1 Exercises 226
- 9.3 Adding a README 227
  - 9.3.1 Exercises 232
- 9.4 Summary 234

### **Chapter 10 Intermediate Workflow 235**

- 10.1 Commit, Push, Repeat 235
  - 10.1.1 Exercises 240
- 10.2 Ignoring Files 241
  - 10.2.1 Exercises 243

- 10.3 Branching and Merging 243
  - 10.3.1 Rebasing 251
  - 10.3.2 Exercises 251
- 10.4 Recovering from Errors 252
  - 10.4.1 Exercises 257
- 10.5 Summary 258

## **Chapter 11 Collaborating 259**

- 11.1 Clone, Push, Pull 260
  - 11.1.1 Exercises 266
- 11.2 Pulling and Merge Conflicts 269
  - 11.2.1 Non-conflicting Changes 270
  - 11.2.2 Conflicting Changes 276
  - 11.2.3 Exercises 280
- 11.3 Pushing Branches 283
  - 11.3.1 Exercises 292
- 11.4 A Surprise Bonus 292
  - 11.4.1 Exercises 294
- 11.5 Summary 295
- 11.6 Advanced Setup 296
  - 11.6.1 A Checkout Alias 297
  - 11.6.2 Prompt Branches and Tab Completion 299
  - 11.6.3 Exercises 300
- 11.7 Conclusion 302

## **Appendix Development Environment 305**

- A.1 Dev Environment Options 306
- A.2 Cloud IDE 307
- A.3 Native OS Setup 312
  - A.3.1 macOS 313
  - A.3.2 Linux 321
  - A.3.3 Windows 322
- A.4 Conclusion 322

- Index 323

# Preface

---

*Learn Enough Developer Tools to Be Dangerous* is designed to teach you three essential tools for modern software development: the Unix command line, a text editor, and version control with Git. All three are ubiquitous in the contemporary technology landscape, and yet there are surprisingly few resources for learning them from scratch and seeing how they all fit together. *Learn Enough Developer Tools to Be Dangerous*, which assumes no prerequisites other than general computer knowledge, was created to fill this gap.

The skills you'll learn in this book are valuable whether your interest is in collaborating with developers or becoming a developer yourself. No matter what you want to do—level up in your current job, start a new career, or even start your own company—*Learn Enough Developer Tools to Be Dangerous* is a great place to start.

The individual subjects covered by this book are potentially enormous; entire books can (and have been) written about each of them. But such giant tomes can be overwhelming, especially for beginners, and they generally involve covering many things you don't actually need right away. Instead, this book focuses on the most important aspects of the respective technologies, grounded in the philosophy that you don't have to learn everything to get started—you just have to learn enough to be *dangerous*.

In addition to teaching you specific skills, *Learn Enough Developer Tools to Be Dangerous* also helps you develop *technical sophistication*—the seemingly magical

ability to solve practically any technical problem. Technical sophistication includes concrete skills like command lines, text editors, and version control, as well as fuzzier skills like Googling the error message and knowing when to just reboot the darn thing. Throughout this book, there are abundant opportunities to develop technical sophistication in the context of real-world examples.

Finally, although the individual parts of the book are as self-contained as possible, they are also extensively cross-referenced to show how the different tools fit together. You'll learn how to use the command line to launch a text editor, make your changes in the editor, and then return to the command line to record the changes with Git. The result is an integrated introduction to the foundations of software development that's practically impossible to find anywhere else.

## Command Line

Part I of *Learn Enough Developer Tools to Be Dangerous*, also known as *Learn Enough Command Line to Be Dangerous*, is an introduction to the Unix command line for complete beginners. It doesn't even assume you know what a "command line" is (though you'll still probably learn a thing or two even if you do). In particular, unlike most command-line tutorials, it doesn't assume you know how to use a text editor (which is the subject of Part II). All of this means you need only basic computer skills (like being able to install new software on your system) to get started.

Like all Learn Enough tutorials, *Learn Enough Command Line to Be Dangerous* is structured as a technical narrative, with each step carefully motivated by real-world uses. Chapter 1 covers the basic notion of a Unix command and shows you how to use your system to learn more about itself. Chapter 2 shows how to use the command line to do things like move, rename, and delete files. Chapter 3 shows how to look inside files (even really big ones), and even how to search through them. Finally, Chapter 4 teaches you how to use the command line to create and navigate directories (folders) to organize files on your system.

The result of finishing *Learn Enough Command Line to Be Dangerous* is a mastery of the basics of a tool that is rarely covered explicitly and yet is everywhere in modern computing. This is especially true of computing in the *Unix tradition*, which includes operating systems like Linux, Android, macOS, and iOS (basically everything but Microsoft Windows, though nowadays even Windows lets you run Linux). This means you'll have a big head start if you're interested in things like web or mobile app development.

## Text Editor

Part II, also known as *Learn Enough Text Editor to Be Dangerous*, covers a category of application—known as a *text editor*—that many people don’t even know exists, and yet is absolutely essential for professional-grade software development. Text editors are used to make files containing *plain text*, which is the document format used for virtually all Web technologies (like HTML and CSS) and programming languages (JavaScript, Ruby, Python, etc.). As such, knowledge of a text editor is a necessary prerequisite for learning those other important subjects.

Because there is such a wide variety of text editors and user preferences, *Learn Enough Text Editor to Be Dangerous* focuses on the main features shared by virtually all editors. Chapter 5 starts by introducing the powerful Vim text editor, which is available on practically every Unix system in the known universe. Chapter 6 then introduces so-called “modern” text editors, mainly using the free and open-source Atom editor but focusing on features shared with other editors like Sublime Text and Visual Studio Code. As a bonus, this chapter includes an integrated introduction to the popular Markdown formatting language. Chapter 7 then covers more advanced subjects like tab triggers and editing source code, and also shows how to write a *shell script* to extend the capabilities of the command line covered in Part I.

## Git

Part III, also known as *Learn Enough Git to Be Dangerous*, covers version control with Git. In line with the approach of the other two parts, *Learn Enough Git to Be Dangerous* doesn’t even assume you know what “version control” is (though any familiarity with the subject will still be helpful). As a software system designed to let you track changes in projects, version control might have been considered optional as recently as the early 2000s, but for modern software development it is absolutely essential, and Git has emerged as the clear winner.

*Learn Enough Git to Be Dangerous* shows how to use Git by tracking changes in a real-world project consisting of a small website (thereby giving you get a head start on web development as well). Chapter 8 shows how to set up a new Git *repository* as a container for your project, beginning with a file consisting of some simple HTML (the markup language of the World Wide Web). Chapter 9 explains how to create a remote backup for your project at GitHub, a popular site for sharing code. Chapter 10 then shows how to use Git to make and record changes to your project, including important techniques known as *branching* and *merging*. Finally, Chapter 11 shows how to use Git to collaborate with other users, including learning how to resolve the kinds



of inevitable file conflicts that arise. As a special bonus, you'll learn how to use a free service called GitHub Pages to deploy your site to the live Web.

### Additional Features

In addition to the main tutorial material, *Learn Enough Developer Tools to Be Dangerous* includes a large number of exercises to help you test your understanding and to extend the material in the main text. The exercises include frequent hints and often include the expected answers, with community solutions available by separate subscription at [www.learnenough.com](http://www.learnenough.com).

For completeness, *Learn Enough Developer Tools to Be Dangerous* includes an appendix on setting up a development environment, including instructions for native systems (macOS, Linux, Windows) and a preconfigured cloud IDE (integrated development environment). This material is also available for free online at [www.learnenough.com/dev-environment](http://www.learnenough.com/dev-environment), which can be consulted for the most up-to-date instructions.

### Final Thoughts

*Learn Enough Developer Tools to Be Dangerous* is designed as a foundational text for modern software development. After learning the developer tools covered in this tutorial, and especially after beginning to develop your technical sophistication, you'll be ready for a huge variety of other resources, including books, blog posts, and online documentation. You'll also have the prerequisites needed for the other Learn Enough tutorials: *Learn Enough HTML, CSS and Layout to Be Dangerous*, *Learn Enough JavaScript to Be Dangerous*, and *Learn Enough Ruby to Be Dangerous*. You can even go on to learn professional-grade web development with the *Ruby on Rails™ Tutorial*.

### Learn Enough Scholarships

Learn Enough is committed to making a technical education available to as wide a variety of people as possible. As part of this commitment, in 2016 we created the Learn Enough Scholarship program (<https://www.learnenough.com/scholarship>). Scholarship recipients get free or deeply discounted access to the Learn Enough All Access subscription, which includes all of the Learn Enough online book content, embedded videos, exercises, and community exercise answers.

As noted in a 2019 RailsConf Lightning Talk (<https://youtu.be/AI5wmnzzBqc?t=1076>), the Learn Enough Scholarship application process is incredibly simple: just fill out a confidential text area telling us a little about your situation. The scholarship

criteria are generous and flexible—we understand that there are an enormous number of reasons for wanting a scholarship, from being a student, to being between jobs, to living in a country with an unfavorable exchange rate against the U.S. dollar. Chances are that, if you feel like you’ve got a good reason, we’ll think so, too.

So far, Learn Enough has awarded more than 2,500 scholarships to aspiring developers around the country and around the world. To apply, visit the Learn Enough Scholarship page at [www.learnenough.com/scholarship](http://www.learnenough.com/scholarship). Maybe the next scholarship recipient could be you!

Register your copy of *Learn Enough Developer Tools to Be Dangerous* on the InformIT site for convenient access to updates and/or corrections as they become available. To start the registration process, go to [informit.com/register](http://informit.com/register) and log in or create an account. Enter the product ISBN (9780137843459) and click Submit. Look on the Registered Products tab for an Access Bonus Content link next to this product, and follow that link to access any available bonus materials. If you would like to be notified of exclusive offers on new editions and updates, please check the box to receive email from us.

*This page intentionally left blank*

# About the Author

---

**Michael Hartl** (<https://www.michaelhartl.com/>) is the creator of the *Ruby on Rails Tutorial* (<https://www.railstutorial.org/>), one of the leading introductions to web development, and is cofounder and principal author at Learn Enough (<https://www.learnenough.com/>). Previously, he was a physics instructor at the California Institute of Technology (Caltech), where he received a Lifetime Achievement Award for Excellence in Teaching. He is a graduate of Harvard College, has a Ph.D. in Physics from Caltech, and is an alumnus of the Y Combinator entrepreneur program.

*This page intentionally left blank*

## CHAPTER 11

---

# Collaborating

Now that we've covered some of the tools needed to use Git effectively on solo projects, it's time to learn about what is perhaps Git's greatest strength: making it easier to collaborate with other people. This is especially the case when using repository hosts like GitHub (<https://github.com/>) or Bitbucket (<https://bitbucket.org/>), but it is also possible to host Git repositories on private servers (sometimes using software like GitLab (<https://about.gitlab.com/>) to get many GitHub-like benefits).

Because this tutorial is designed for individual readers, we won't actually be able to collaborate with others, but this chapter will explain how you can practice "collaborating" with yourself. There are many different collaboration scenarios, and they vary significantly by team and by project, so we'll focus on the important case of multiple collaborators who all have *commit rights* to a particular repo. This model is appropriate for teams where everyone can make changes without explicit approval from a project maintainer.

Open-source projects typically use a different flow involving *forking* and *pull requests*, but the details differ enough that it's best to defer to the collaboration instructions of each particular project. Consider, for example, the instructions for contributing to Ruby on Rails ([https://guides.rubyonrails.org/contributing\\_to\\_ruby\\_on\\_rails.html](https://guides.rubyonrails.org/contributing_to_ruby_on_rails.html)). With the commands from this tutorial and your technical sophistication (Box 8.2), you'll be in a good position to understand and follow such instructions if you decide to get involved in contributing to open-source software or other projects under version control with Git.

For reference, important commands from this chapter are summarized in Section 11.5.

## 11.1 Clone, Push, Pull

As an example of a common collaboration workflow, we'll simulate the case of two developers working on the same project, in this case the simple website developed in this tutorial. We'll start with Alice (Figure 11.1)<sup>1</sup> working in the original **website** directory, and we'll create a second directory (**website-copy**) for her collaborator Bob (Figure 11.2).<sup>2</sup>

As a first step, Alice runs **git push** just to make sure all her changes are on the remote repository:



**Figure 11.1:** Alice, working on **website**.

---

1. *Alice's Adventures in Wonderland* original illustrations by John Tenniel. Colorized image courtesy of The Print Collector/Alamy Stock Photo.

2. Image courtesy RTRO/Alamy Stock Photo.



**Figure 11.2:** Bob (with son Tim), working on **website-copy**.



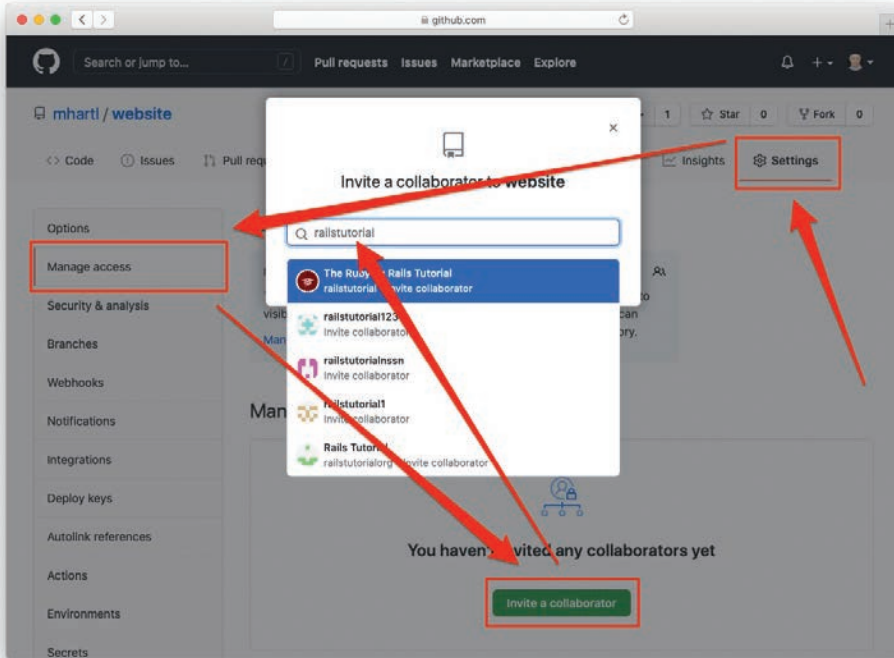
```
[website (main)]$ git push
```

In real life, Alice would now need to add Bob as a collaborator on the **website** repository, which she could do at GitHub by clicking on Settings > Manage Access > Invite a collaborator and then put Bob’s GitHub username in the invitation box (Figure 11.3). Because we’re collaborating with ourselves, we can skip this step.

Once Bob gets the notification that he’s been added to the **website** repository, he can go to GitHub to get the *clone URL*, as shown in Figure 11.4. This URL lets Bob make a full copy of the repository (including its history) using **git clone**.

Ordinarily, Bob would probably use his own **repos** directory, with a project called **website** as in Alice’s original, but because we’re only simulating the collaboration we’ll use the name **website-copy** for clarity. In addition, when doing something





**Figure 11.3:** The GitHub page to add collaborators.

a little artificial like this, I like to use a temp directory called `~/tmp`,<sup>3</sup> so create this directory if it doesn't already exist on your system:

```
$ cd
$ mkdir tmp
```

Then `cd` to it and clone the repo to the local directory:



3. The idea behind a temp directory is to have a place to put temporary files that won't necessarily persist for long. Many operating systems have a system-wide temp directory (often called `/tmp`), but I also like to have one under my home directory for personal use.

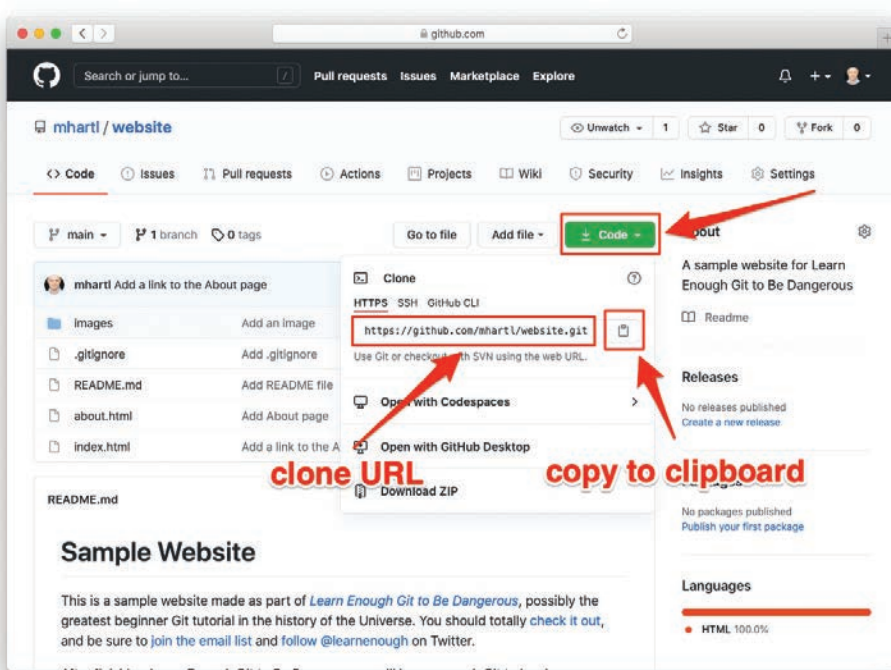


Figure 11.4: Finding the clone URL at GitHub.

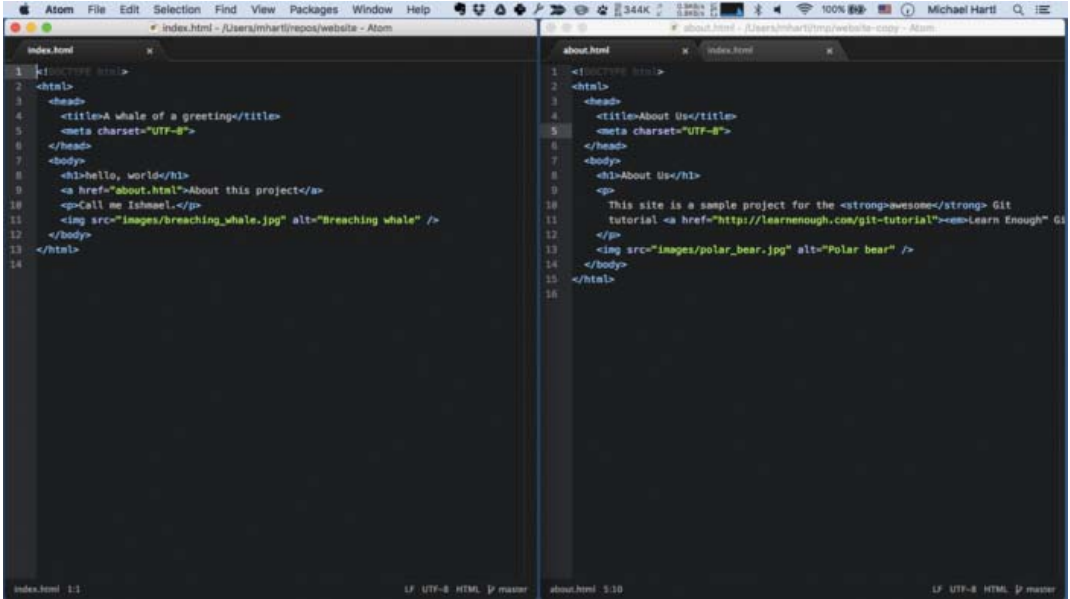
```
[~]$ cd tmp/
[tmp]$ git clone <clone URL> website-copy
Cloning into 'website-copy'...
[tmp]$ cd website-copy/
```

Here we've included the argument **website-copy** to **git clone**, thereby showing how to use a different name than the original repo, but usually you just run **git clone <clone URL>**, which uses the default repo name (in this case, **website**).

Now we're ready to open the copy of the project and start making edits:



```
[website-copy (main)]$ atom .
```



**Figure 11.5:** The `website` and `website-copy` editors running side by side.

For the purposes of this exercise, I recommend placing the editor windows for `website` and `website-copy` side by side, as shown in Figure 11.5.

To begin the collaboration, we'll have Bob make a change to the site by wrapping the tutorial title on the About page in a link, like this:

```
<a href="https://www.learnenough.com/git-tutorial">...</a>
```

Here the ellipsis ... represents the full title of the tutorial, *Learn Enough Git to Be Dangerous*. The resulting line is too long to display here, but we can wrap it, as shown in Figure 11.6, with the result as shown in Figure 11.7.

If we look at the diff using `git diff`, we see the wrapped line (Figure 11.8), which appears in a browser as shown in Figure 11.9.

Having added the link, Bob can commit his changes and push up to the remote repository:



**Figure 11.6:** Toggling soft wrap in Atom.

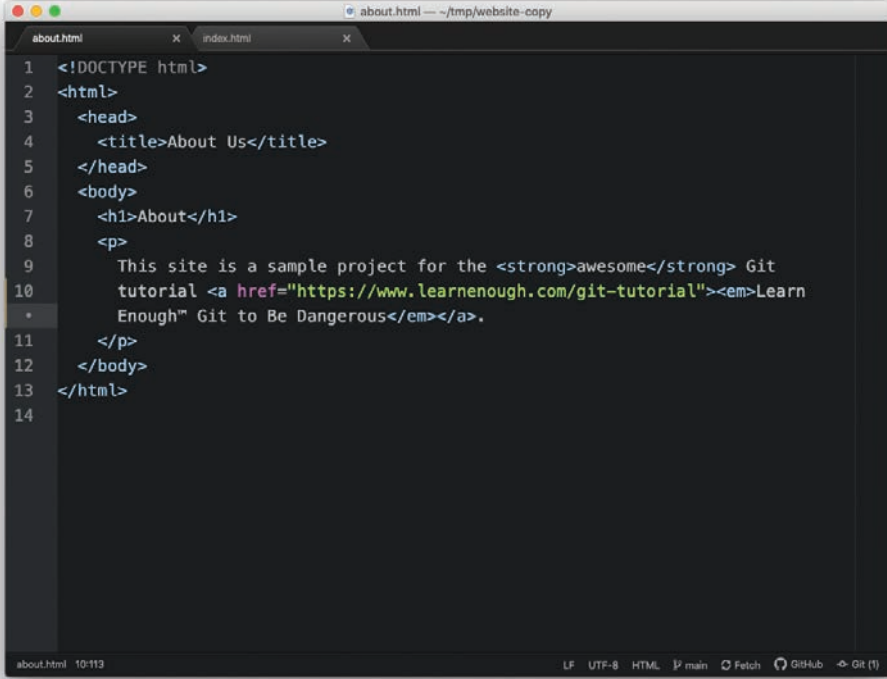


```
[website-copy (main)]$ git commit -am "Add link to tutorial title"
[website-copy (main)]$ git push
```

At this point, Bob might send Alice a notification that there's a change ready, or Alice might just be diligent about checking for changes. In either case, Alice can get the changes from the remote origin by running **git pull**. I suggest opening up a new tab in your terminal window for Alice's directory (as shown in Figure 11.10) and then pull as follows:



```
[website (main)]$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (1/1), done.
remote: Total 3 (delta 2), reused 3 (delta 2), pack-reused 0
Unpacking objects: 100% (3/3), 336 bytes | 168.00 KiB/s, done.
```



```

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>About Us</title>
5   </head>
6   <body>
7     <h1>About</h1>
8     <p>
9       This site is a sample project for the <strong>awesome</strong> Git
10      tutorial <a href="https://www.learnenough.com/git-tutorial"><em>Learn
11      Enough</em> Git to Be Dangerous</em></a>.
12     </p>
13   </body>
14 </html>

```

**Figure 11.7:** The About page with soft wrap activated.

```

From https://github.com/mhartl/website
 cad4761..9a9cecf  main      -> origin/main
Updating cad4761..9a9cecf
Fast-forward
 about.html | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

```

With that, Alice’s project should have Bob’s commit, and her copy of the About page should be identical to Figure 11.9. (Checking that Bob’s commit is present in the log is left as an exercise.)

### 11.1.1 Exercises

1. As Alice, run **git log** to verify that the commit was pulled down correctly. Double-check the details using **git log -p**.

```
Default (-zsh) 10
[website-copy (main)]$ git diff
diff --git a/about.html b/about.html
index 367dd8e..a9d0e92 100644
--- a/about.html
+++ b/about.html
@@ -7,7 +7,7 @@
<h1>About</h1>
<p>
  This site is a sample project for the <strong>awesome</strong> Git
-  tutorial <em>Learn Enough™ Git to Be Dangerous</em>.
+  tutorial <a href="https://www.learnenough.com/git-tutorial"><em>Learn Eno
ugh™ Git to Be Dangerous</em></a>.
</p>
</body>
</html>
[website-copy (main)]$
```

**Figure 11.8:** The diff with a wrapped line.

2. The whale picture added in Listing 10.1 (Figure 10.1) requires attribution under the Creative Commons Attribution-NoDerivs 2.0 Generic license. As Alice, link the image to the original attribution page, as shown in Listing 11.1. Commit the result and push to GitHub.
3. As Bob, pull in the changes from the previous exercise. Verify by refreshing the browser and by running `git log -p` that Bob's repo has been properly updated.



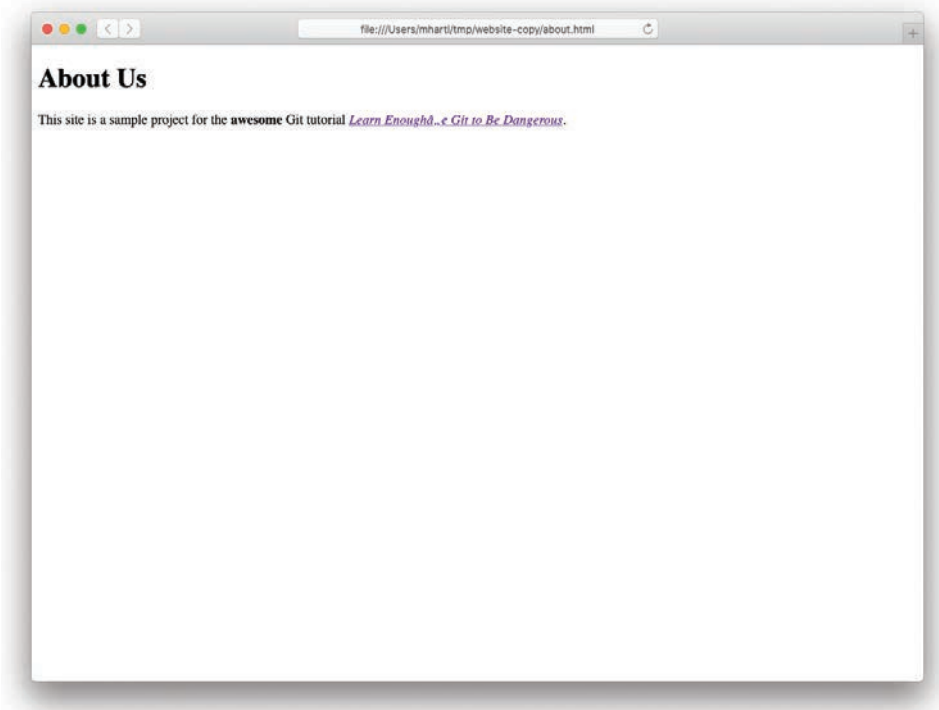
**Listing 11.1:** Linking to the whale image's attribution page.

```
~/repos/website/index.html
```

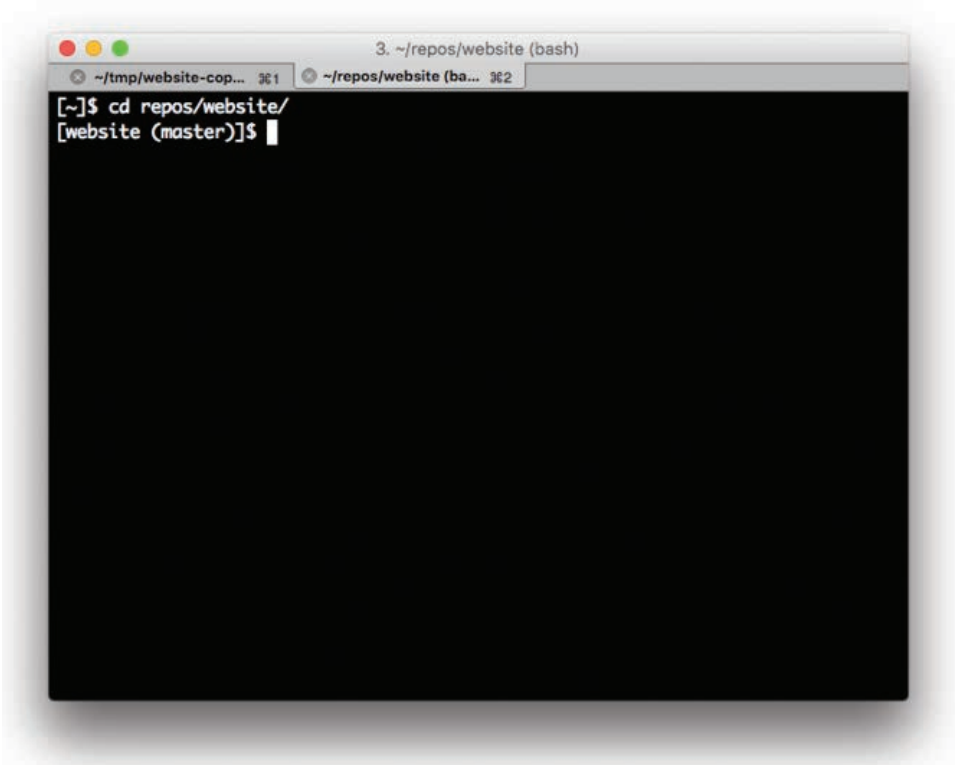
---

```
.  
.   
.   
<a href="https://www.flickr.com/photos/28883788@N04/10097824543">  
    
</a>  
.   
.   
.
```

---



**Figure 11.9:** Linking the Git tutorial title on the About page.



**Figure 11.10:** Using a new terminal tab for the original directory.

## 11.2 Pulling and Merge Conflicts

In Section 11.1, Alice didn't make any changes while Bob was making his commit, so there was no chance of conflict, but this is not always the case. In particular, when two collaborators edit the same file, it is possible that the changes might be irreconcilable. Git is pretty smart about merging in changes, and in general conflicts are surprisingly rare, but it's important to be able to handle them when they occur. In this section, we'll consider both non-conflicting and conflicting changes in turn.



## 11.2.1 Non-conflicting Changes

We'll start by having Alice and Bob make *non*-conflicting changes in the same file. Suppose Alice decides to change the top-level heading on the About page from “About” to “About Us”, as shown in Listing 11.2.



**Listing 11.2:** Alice's change to the About page's **h1**.

```
~/repos/website/about.html
```

---

```
<!DOCTYPE html>
<html>
  .
  .
  .
  <h1>About Us</h1>
  .
  .
  .
</body>
</html>
```

---

After making this change, Alice commits and pushes as usual:



```
[website (main)]$ git commit -am "Change page heading"
[website (main)]$ git push
```

Meanwhile, Bob decides to add a new image (Figure 11.11)<sup>4</sup> to the About page. He first downloads it with **curl** as follows:

---

4. Image courtesy of Vaclav Sebek/Shutterstock.



**Figure 11.11:** An image for Bob to add to the About page.



```
[website-copy (main)]$ curl -o images/polar_bear.jpg \  
> -L https://cdn.learnenough.com/polar_bear.jpg
```

(As noted in Section 10.1, you should type the backslash character `\` but you *shouldn't* type the literal angle bracket `>`.) He then adds it to **about.html** using the **img** tag, as shown in Listing 11.3, with the result shown in Figure 11.12.



**Listing 11.3:** Adding an image to the About page.

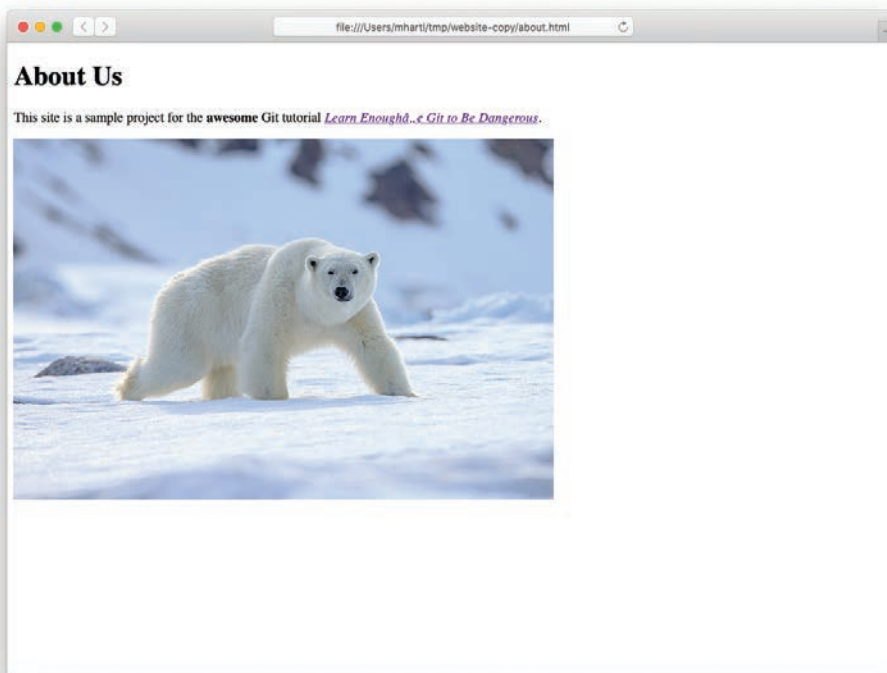
```
~/tmp/website-copy/about.html
```

---

```
<!DOCTYPE html>
<html>
  .
  .
  .
  
</body>
</html>
```

---

Note that Bob has included an **alt** attribute in Listing 11.3, which is a text alternative to the image. The **alt** attribute is actually required by the HTML5 standard, and including it is a good practice because it's used by web spiders and by screen readers for the visually impaired.



**Figure 11.12:** The About page with an added image.

Having made his change, Bob commits as usual:



```
[website-copy (main)]$ git add -A  
[website-copy (main)]$ git commit -m "Add an image"
```

When he tries to push, though, something unexpected happens, as shown in Listing 11.4.



---

**Listing 11.4:** Bob's push, rejected.

```
[website-copy (main)]$ git push  
To https://github.com/mhartl/website.git  
! [rejected]      main -> main (fetch first)  
error: failed to push some refs to 'https://github.com/mhartl/website.git'  
hint: Updates were rejected because the remote contains work that you do  
hint: not have locally. This is usually caused by another repository pushing  
hint: to the same ref. You may want to first integrate the remote changes  
hint: (e.g., 'git pull ...') before pushing again.  
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

---

Because of the changes Alice already pushed, Git won't let Bob's push go through: As indicated by the first highlighted line in Listing 11.4, the push was rejected by GitHub. As indicated by the second highlighted line, the solution to this is for Bob to **pull**:



```
[website-copy (main)]$ git pull
```






---

**Listing 11.5:** The Git log after Bob merges in Alice's changes. (Exact results will differ.)
 

---

```
[website-copy (main)]$ git log
commit 679afb8771b1893a865c3775a2786390a936db26 (HEAD -> main)
Merge: 7a69702 baafb1b
Author: Michael Hartl <michael@michaelhartl.com>
Date: Thu Apr 1 12:28:00 2021 -0700

    Merge branch 'main' of https://github.com/mhartl/website

commit 7a6970229233346ce10cfefb3ace91b1d37c4cb2
Author: Michael Hartl <michael@michaelhartl.com>
Date: Thu Apr 1 12:26:26 2021 -0700

    Add an image

commit baafb1bd473d553f1532267edfbbf09faf813bf2 (origin/main, origin/HEAD)
Author: Michael Hartl <michael@michaelhartl.com>
Date: Thu Apr 1 12:25:18 2021 -0700

    Change page heading
```

---

If Bob now pushes, it should go through as expected:



```
$ git push
```

This puts Bob's changes on the remote repo, which means Alice can pull them in:

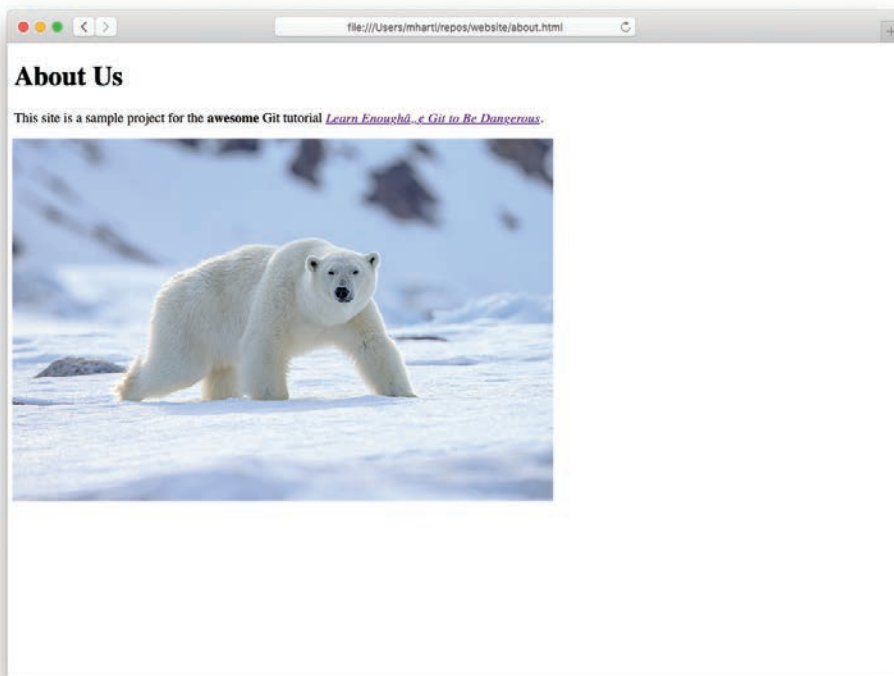


```
$ git pull
```

Alice can confirm that her repo now includes Bob's changes by inspecting the Git log, which should match the results you got in Listing 11.5. Meanwhile, she can refresh her browser to see Bob's cool new ursine addition (Figure 11.14).

## 11.2.2 Conflicting Changes

Even though Git's merge algorithms can often figure out how to combine changes from different collaborators, sometimes there's no avoiding a conflict. For example, suppose both Alice and Bob notice that the required **alt** attribute is missing from the whale image included in Listing 10.1 and decide to correct the issue by adding one.



**Figure 11.14:** Confirming that Alice's repo includes Bob's added image.

First, Alice adds the **alt** attribute “Breaching whale” (Listing 11.6).



**Listing 11.6:** Alice’s image **alt**.

`~/repos/website/index.html`

---

```
<!DOCTYPE html>
<html>
  .
  .
  .
  <a href="https://www.flickr.com/photos/28883788@N04/10097824543">
    
  </a>
</body>
</html>
```

---

She then commits and pushes her change:<sup>5</sup>



```
[website (main)]$ git commit -am "Add necessary image alt"
[website (main)]$ git push
```



---

5. Listing 11.6 and Listing 11.7 include the attribution link added in the Section 11.1.1 exercises.



**Listing 11.7: Bob's image alt.**

~/tmp/website-copy/index.html

---

```

<!DOCTYPE html>
<html>
  .
  .
  .
  <a href="https://www.flickr.com/photos/28883788@N04/10097824543">
    
  </a>
</body>
</html>

```

---

Meanwhile, Bob adds his own **alt** attribute, “Whale” (Listing 11.7), and commits his change:



```
[website-copy (main)]$ git commit -am "Add an alt attribute"
```

If Bob tries to **push**, he'll be met with the same rejection message shown in Listing 11.4, which means he should pull—but that comes at a cost:



```

[website-copy (main)]$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (1/1), done.
remote: Total 3 (delta 2), reused 3 (delta 2), pack-reused 0
Unpacking objects: 100% (3/3), 415 bytes | 207.00 KiB/s, done.
From https://github.com/mhartl/website
   679afb8..81c190a  main      -> origin/main
Auto-merging index.html

```

```
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
[website-copy (main|MERGING)]$
```

As indicated in the second highlighted line, Git has detected a merge conflict from Bob's pull, and his working copy has been put into a special branch state called **main|MERGING**.

Bob can see the effect of this conflict by viewing **index.html** in his text editor, as shown in Figure 11.15. Supposing Bob prefers Alice's more descriptive **alt** text, he can resolve the conflict by deleting all but the line with **alt="Breaching whale"**, as seen in Figure 11.16. (In fact, as seen in Figure 11.15, Atom includes two "Use me" buttons to make it easy to pick one of the options. Clicking on the bottom "Use me" button gives the same result shown in Figure 11.16.)

After saving the file, Bob can commit his change, which causes the prompt to revert back to displaying the **main** branch, and at that point he's ready to **push**:



```
[website-copy (main|MERGING)]$ git commit -am "Use longer alt attribute"
[website-copy (main)]$ git push
```

Alice's and Bob's repos now have the same content, but it's still a good idea for Alice to pull in Bob's merge commit:



```
[website (main)]$ git pull
```

Because of the potential for conflict, it's a good idea to do a **git pull** before making any changes on a project with multiple collaborators (or even just being edited by the same person on different machines). Even then, on a long enough timeline some conflicts are inevitable, and with the techniques in this section you're now in a position to handle them.

```

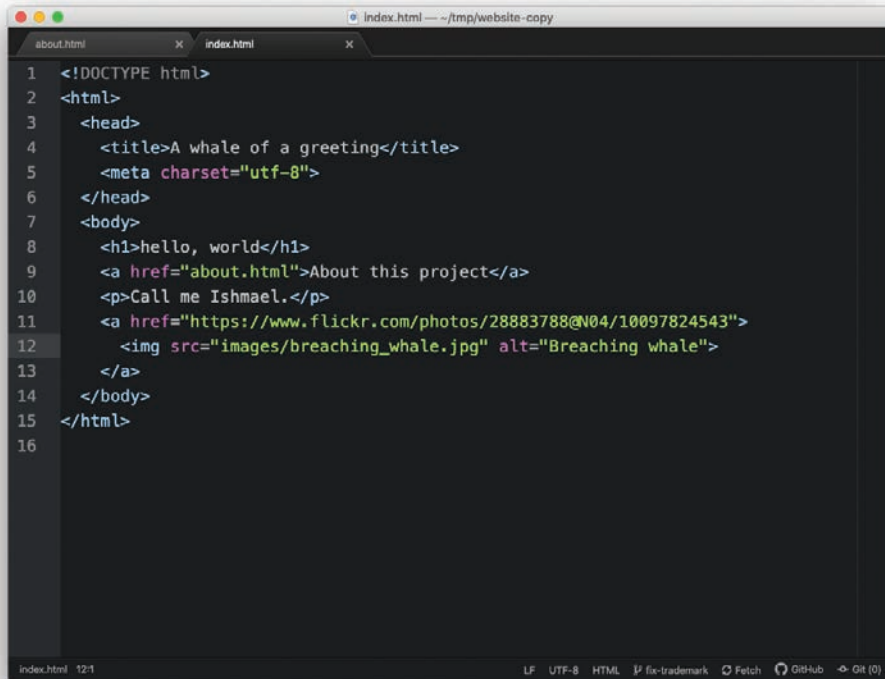
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title>A whale of a greeting</title>
5  </head>
6  <body>
7  <h1>hello, world</h1>
8  <a href="about.html">About this project</a>
9  <p>Call me Ishmael.</p>
10 
11 <a href="https://www.flickr.com/photos/28883788@N04/10097824543">
12 <<<<<< HEAD
13 
14 =====
15 
16 >>>>>> e36f104fef425cf83bdebc4f80ba5f649524c4fc
17 </a>
18 </body>
19 </html>
20

```

**Figure 11.15:** A file with a merge conflict.

### 11.2.3 Exercises

1. Change your default Git editor from Vim to Atom. *Hint:* Google for it. (This is an absolutely *classic* application of technical sophistication (Box 8.2): With a well-chosen Google search, you can often go from “I have no idea how to do this” to “It’s done” in under 30 seconds.)
2. The polar bear picture added in Listing 11.3 (Figure 11.11) requires attribution under the Creative Commons Attribution 2.0 Generic license. As Alice, link the image to the original attribution page, as shown in Listing 11.8. Then run **git commit -a** *without* including **-m** and a command-line message. This should drop you into the default Git editor. Quit the editor *without* including a message, which cancels the commit.

A screenshot of a code editor window titled "index.html" showing HTML code. The code is as follows:

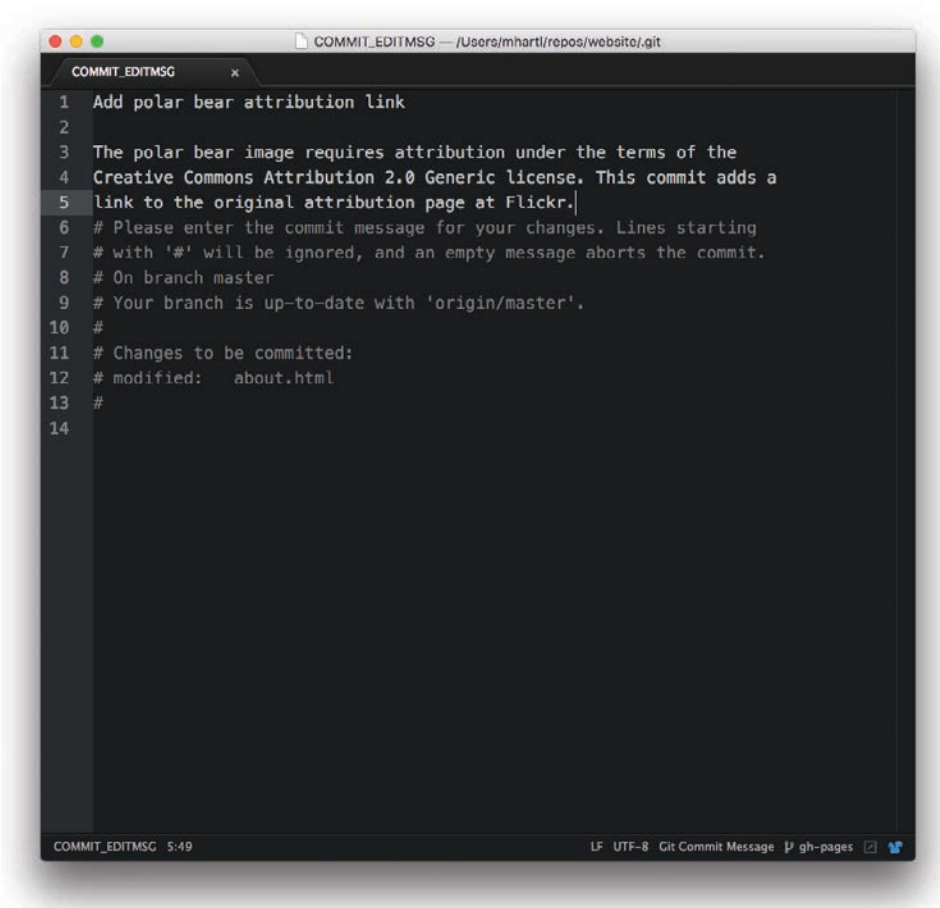
```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>A whale of a greeting</title>
5     <meta charset="utf-8">
6   </head>
7   <body>
8     <h1>hello, world</h1>
9     <a href="about.html">About this project</a>
10    <p>Call me Ishmael.</p>
11    <a href="https://www.flickr.com/photos/28883788@N04/10097824543">
12      
13    </a>
14  </body>
15 </html>
16
```

The editor interface includes a tab for "index.html", a line number column on the left, and a status bar at the bottom with icons for LF, UTF-8, HTML, fix-trademark, Fetch, GitHub, and Git (0).

**Figure 11.16:** The HTML file edited to remove the merge conflict.

3. Run **git commit -a** again, but this time add the commit message “Add polar bear attribution link”. Then hit return a couple of times and add a longer message of your choice. (One example appears in Figure 11.17.) Save the message and exit the editor.
4. Run **git log** to confirm that both the short and longer messages correctly appear. After pushing the changes to GitHub, navigate to the page for the commit to confirm that both the short and longer messages correctly appear.
5. As Bob, pull in the changes to the About page. Verify by refreshing the browser and by running **git log -p** that Bob’s repo has been properly updated.





**Figure 11.17:** Adding a longer message in a text editor.

**Listing 11.8:** Linking to the polar bear image's attribution page.

*~/repos/website/about.html*

---

```
.
.
.
<a href="https://www.flickr.com/photos/puliarfanita/22959238329">
  
</a>
.
.
.
```

---

## 11.3 Pushing Branches

In this section, we'll apply our newfound collaboration skills to get Alice to request a bugfix from Bob, who will make the correction and then share the result with Alice. In the process, we'll learn how to collaborate on branches other than `main`, thereby applying the material from Section 10.3 as well.

Recall from Section 10.3 that the trademark character <sup>TM</sup> is currently broken on the About page (Figure 10.7). Alice suspects the fix for this involves adding some markup to the HTML template for the website's pages, but she's already agreed to attend a tea party (Figure 11.18),<sup>6</sup> so she only has time to add a couple of *HTML comments* requesting for Bob to add the relevant fix, as shown in Listing 11.9 and Listing 11.10. (We'll cover HTML comments further in *Learn Enough HTML to Be Dangerous* (<https://www.learnenough.com/html>).



**Figure 11.18:** Alice has a tea party to attend and so asks Bob to fix the website.

---

6. *Alice's Adventures in Wonderland* original illustrations by John Tenniel. Image courtesy of The History Collection / Alamy Stock Photo.



**Listing 11.9:** A stub for the fix to the ™ problem.

`~/repos/website/about.html`

---

```
<!DOCTYPE html>
<html>
  <head>
    <title>About Us</title>
    <!-- Add something here to fix trademark -->
  </head>
  .
  .
  .
</html>
```

---

**Listing 11.10:** A stub to add the ™ fix to the index page.

`~/repos/website/index.html`

---

```
<!DOCTYPE html>
<html>
  <head>
    <title>A whale of a greeting</title>
    <!-- Add something here to fix trademark -->
  </head>
  .
  .
  .
</html>
```

---

Notice that Alice has wisely asked Bob to fix the index page as well (Listing 11.10) even though the current error only occurs on the About page. This way, any ™ or similar characters added to `index.html` will automatically work in the future. (As noted in Section 10.3, having to make such changes in multiple places is annoying, and it's also brittle and error-prone. The correct solution is to use *templates*, which we'll cover starting in *Learn Enough CSS & Layout to Be Dangerous* (<https://www.learnenough.com/css-and-layout>).

Alice has decided to follow a common convention and use a separate branch for the bugfix, which in this case she calls **fix-trademark**:



```
[website (main)]$ git checkout -b fix-trademark
[website (fix-trademark)]$
```

This shows something important: It's possible to make changes to the working directory (in this case, the additions from Listing 11.9 and Listing 11.10) *before* creating a new branch, as long as those changes haven't yet been committed.

Having made the new branch for the fix, Alice can make a commit and push up the branch using **git push**:



```
[website (fix-trademark)]$ git commit -am "Add placeholders for the TM fix"
[website (fix-trademark)]$ git push -u origin fix-trademark
```

Here Alice has used exactly the same **push** syntax used in Listing 9.1 to push the repo up to GitHub in the first place, with **fix-trademark** in place of **main**.

If Alice sends Bob a note before she heads off to her tea party, Bob will know to do a **git pull** to pull in Alice's changes:



```
[website-copy (main)]$ git pull
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (1/1), done.
```



```
remote: Total 4 (delta 3), reused 4 (delta 3), pack-reused 0
Unpacking objects: 100% (4/4), 444 bytes | 148.00 KiB/s, done.
From https://github.com/mhartl/website
* [new branch] fix-trademark -> origin/fix-trademark
Already up to date.
```

Bob can check his local working directory for the **fix-trademark** branch that Alice created and pushed, but it isn't there:



```
[website-copy (main)]$ git branch
* main
```

The reason is that the branch is associated with the remote **origin**, and such branches aren't displayed by default. To see it, Bob can use the **-a** option (for "all"):<sup>7</sup>



```
[website-copy (main)]$ git branch -a
* main
remotes/origin/HEAD -> origin/main
remotes/origin/fix-trademark
remotes/origin/main
```

To start work on **fix-trademark** on his local copy, Bob just needs to check it out. By using the same name (i.e., **fix-trademark**), he arranges for it to be associated with the upstream branch on GitHub, which means that **git push** will automatically push up his changes:

---

7. In fact, **git branch --all** works, but when using Git at the command line it's more common to use the short forms of the options.



```
[website-copy (main)]$ git checkout fix-trademark
Branch fix-trademark set up to track remote branch fix-trademark from origin.
Switched to a new branch 'fix-trademark'
[website-copy (fix-trademark)]$
```

At this point, Bob can **diff** against **main** to see what he's dealing with:

```
[website-copy (fix-trademark)]$ git diff main
diff --git a/about.html b/about.html
index 173e5fe..4d4b780 100644
--- a/about.html
+++ b/about.html
@@ -2,6 +2,7 @@
<html>
  <head>
    <title>About Us</title>
+   <!-- Add something here to fix trademark -->
  </head>
  <body>
    <h1>About Us</h1>
diff --git a/index.html b/index.html
index 024ada5..d8e946f 100644
--- a/index.html
+++ b/index.html
@@ -2,6 +2,7 @@
<html>
  <head>
    <title>A whale of a greeting</title>
+   <!-- Add something here to fix trademark -->
  </head>
  <body>
    <h1>hello, world</h1>
```

Now all Bob has to do is actually implement the fix. If you'd like a challenging exercise in technical sophistication, try Googling around to see if you can figure out what the problem might be, and also how you might fix it. In case you'd like to do this, I'll wait here while you look...

All right, the problem is that the page doesn't have the right *character encoding* to display non-ASCII characters like <sup>TM</sup>, ®, or £. The fix involves using a tag called **meta**

to tell browsers to use a character set (or **charset** for short) called UTF-8, which will let our page display anything that's part of the enormous set of Unicode characters. The result, which you would not necessarily be able to guess, appears in Listing 11.11 and Listing 11.12.



**Listing 11.11:** A fix for the ™ problem.

*~/tmp/website-copy/about.html*

---

```
<!DOCTYPE html>
<html>
  <head>
    <title>About Us</title>
    <meta charset="utf-8">
  </head>
  .
  .
  .
</html>
```

---

**Listing 11.12:** Adding the ™ fix to the index page.

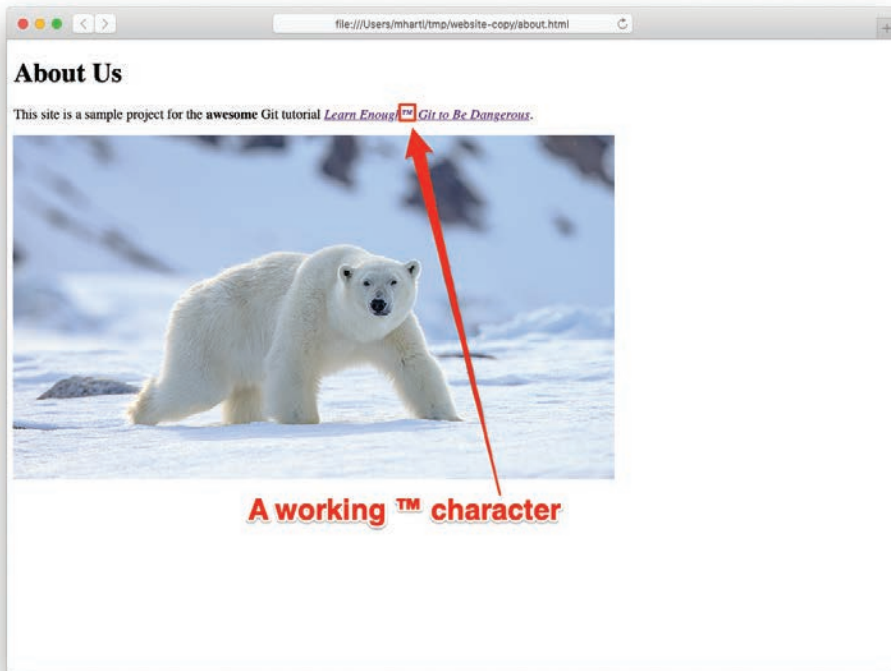
*~/tmp/website-copy/index.html*

---

```
<!DOCTYPE html>
<html>
  <head>
    <title>A whale of a greeting</title>
    <meta charset="utf-8">
  </head>
  .
  .
  .
</html>
```

---

Like the **img** tag introduced in Section 10.1, **meta** is a void element and so has no closing tag.



**Figure 11.19:** Confirming a working trademark character.

Having made the change, Bob can confirm the fix by reloading the page in his browser, as shown in Figure 11.19.

Confident that his solution is correct, Bob can now make a commit and push the fix up to the remote server:



```
[website-copy (fix-trademark)]$ git commit -am "Fix trademark character display"
[website-copy (fix-trademark)]$ git push
```



**Figure 11.20:** Bob’s reward for a job well-done.

With that, Bob sends a note to Alice that the fix is pushed, and heads out for some well-deserved rest (Figure 11.20).<sup>8</sup>

Alice, now back from her tea party, gets Bob’s note and pulls in his fix:

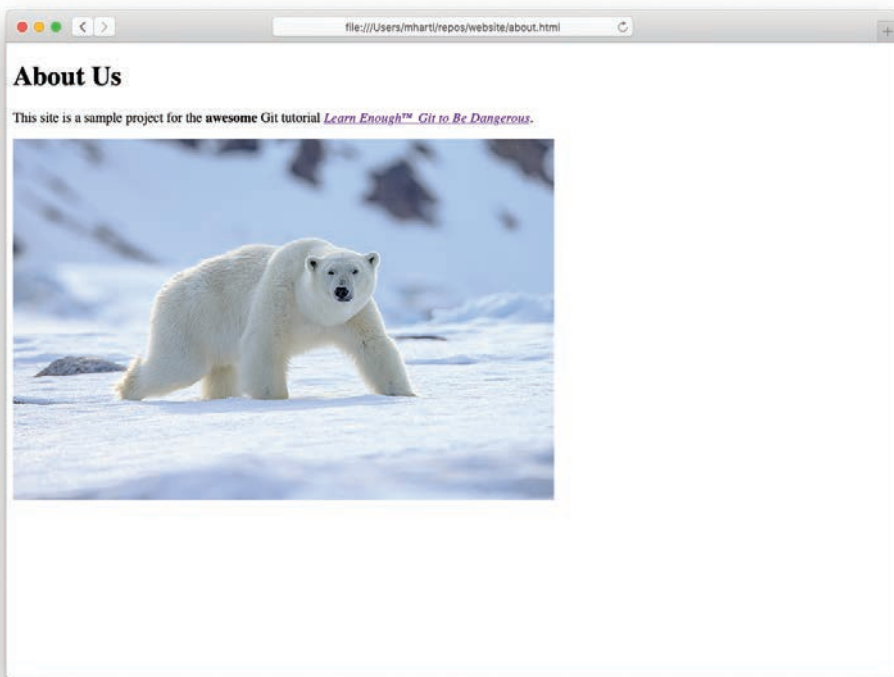


```
[website (fix-trademark)]$ git pull
```

She refreshes her browser to confirm that the <sup>TM</sup> character displays properly on her end of things (Figure 11.21), and then merges the changes into **main**:

---

8. Image courtesy of Maxim Safronov/Shutterstock.



**Figure 11.21:** Reconfirming the trademark fix before merging.



```
[website (fix-trademark)]$ git checkout main
[website (main)]$ git merge fix-trademark
[website (main)]$ git push
```

With the final **git push**, Alice arranges for the remote **main** branch on GitHub to get the fix. (Syncing up Bob’s **main** branch is left as an exercise (Section 11.3.1).)

Of course, **git push** publishes the change only to a remote Git repository. Wouldn’t it be nice if there were a way to confirm that the <sup>TM</sup> character—and the rest of the website—displays correctly on the live Web?

### 11.3.1 Exercises

1. Bob's `main` branch doesn't currently have Alice's merge, so check out `main` as Bob and do a `git pull`. Confirm using `git log` that Alice's merge commit is now present.
2. Delete the `fix-trademark` branch locally. Do you need to use the `-D` option (Section 10.3.2), or is `-d` sufficient?
3. Delete the remote `fix-trademark` branch on GitHub. *Hint*: If you get stuck, Google for it.

## 11.4 A Surprise Bonus

As hinted at the end of the last section, it would be nice to be able to confirm that the new character encoding works on a live web page. But this requires knowing how to deploy a live site to the Web, and that's beyond the scope of a humble Git tutorial, right? Amazingly, the answer is no. The reason is that GitHub offers a free service called *GitHub Pages*, and *any* repository at GitHub containing static HTML is automatically available as a live website.

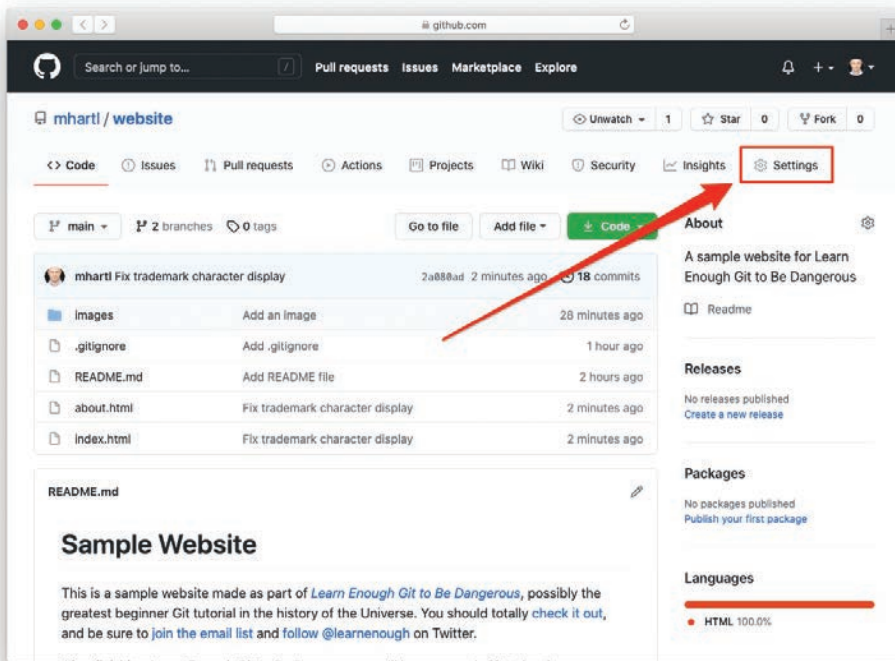
There is one minor prerequisite to using GitHub Pages, which is that you have to verify your email address with GitHub. Once you've done that, though, all you need to do is configure your repository to use GitHub Pages on the `main` branch, which you can do by going to the settings (Figure 11.22) and then selecting the `main` option (Figure 11.23) and saving the changes (Figure 11.24).

That's it! Our website is now available at the URL

```
https://<name>.github.io/website/
```

where `<name>` is your GitHub username. Since my username is `mhartl`, my copy of this tutorial's website is at `mhartl.github.io/website/`, as shown in Figure 11.25.

Note that the URL `https://<name>.github.io/website/` automatically displays `index.html`, which is the usual convention on the Web: The index page is understood to be the default, so there's no need to type it in. This is not the case with other pages, though, and if you follow the link to the About page you'll see that the filename appears in the address bar (Figure 11.26). As seen in Figure 11.27, the trademark character <sup>TM</sup> also renders correctly on the live website, just as we hoped it would.

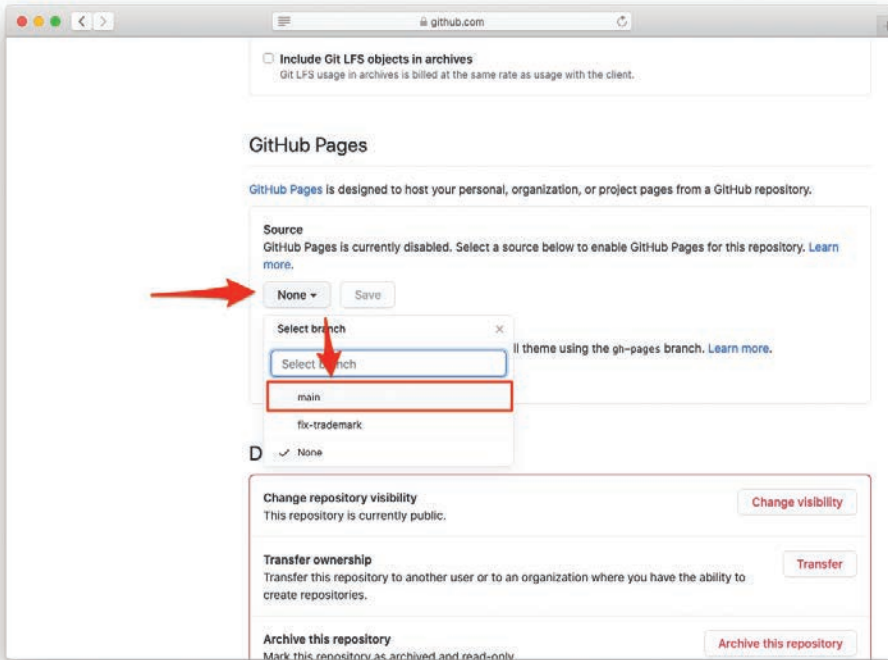


**Figure 11.22:** The settings for a GitHub repository.

Because static HTML pages by definition don't change from one page view to the next, GitHub can *cache* them efficiently, storing them for the next user who visits the site. This makes GitHub Pages sites both fast and cheap to serve (which is why GitHub can afford to offer them for free). You can even configure GitHub Pages to work with a custom domain, letting you replace `<name>.github.io` with something like `www.example.com`; see the free tutorial *Learn Enough Custom Domains to Be Dangerous* (<https://www.learnenough.com/custom-domains>) to learn how to do it. This combination of high performance and support for custom domains makes GitHub Pages suitable for production websites—for example, the Learn Enough blog (<https://news.learnenough.com/>) is a static website running on a custom domain at GitHub Pages.

The example website in this tutorial is really just a toy, but it's a great start, and we'll build on this foundation to make a nearly industrial-grade website in *Learn Enough HTML to Be Dangerous* and a fully industrial-grade site in *Learn Enough CSS & Layout to Be Dangerous*.

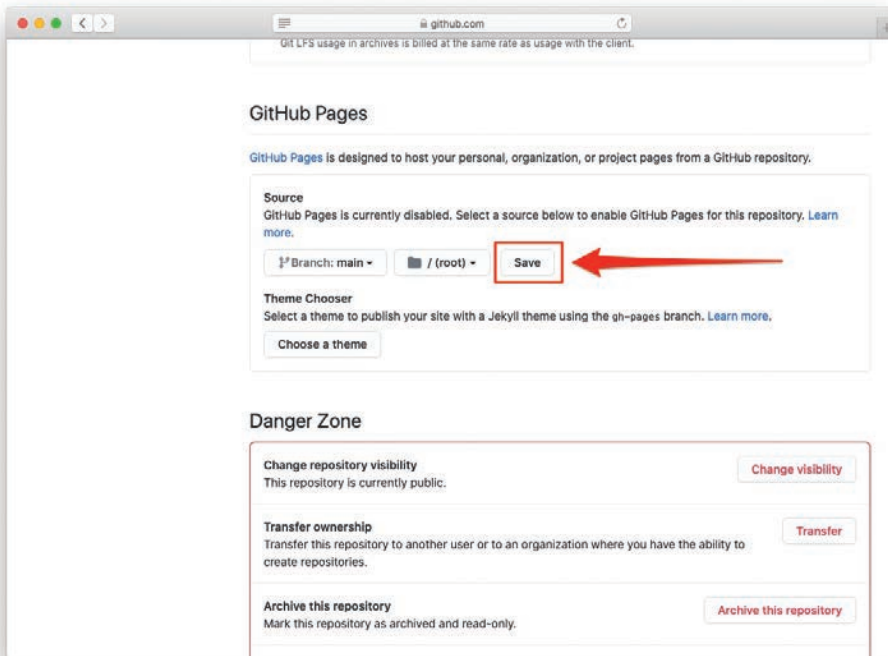




**Figure 11.23:** Serving our website from the `main` branch.

### 11.4.1 Exercises

1. On the About page, add a link back to `index.html`. Commit and push your change and verify that the link works on the production site.
2. As noted in Section 2.3, two of the most important Unix commands are `mv` and `rm`. Git provides analogues of these commands, which have the same effect on local files while also arranging to track the changes. Experiment with these commands via the following sequence: Create a file with some *lorem ipsum* text, add & commit it, rename it with `git mv` & commit, then remove it with `git rm` & commit again. Examine the results of `git log -p` to see how Git handled the operations.

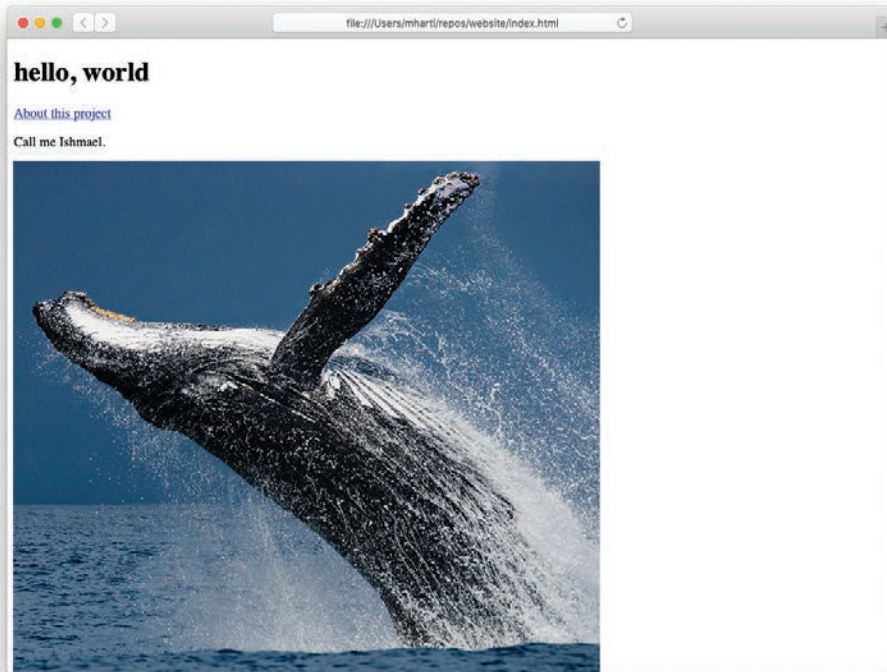


**Figure 11.24:** Saving the new GitHub Pages settings.

3. To practice the process of making a new Git repository, make a second project called **second\_website** in the **repos** directory. Create an **index.html** file with the content “hello, again!” and follow the steps (starting in Section 8.2) needed to deploy it to the live Web.
4. Make a third, secret project called **secret\_project**. Touch files called **foo**, **bar**, and **baz** in the main project directory, and then follow the steps to initialize the repository and commit the initial results. Then, to practice using a service other than GitHub, create a free private repository at Bitbucket.

## 11.5 Summary

Important commands from this chapter are summarized in Table 11.1.



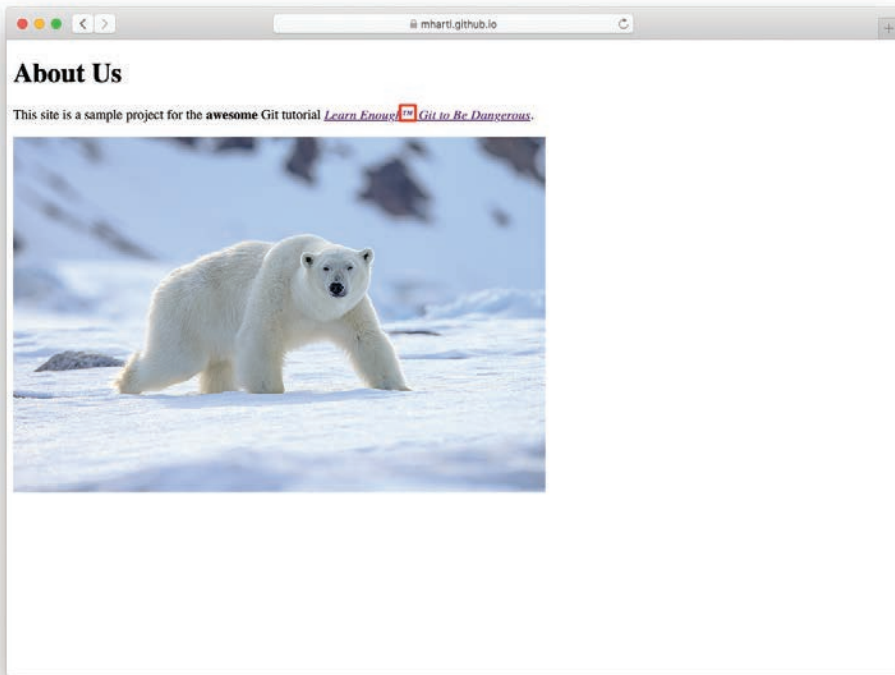
**Figure 11.25:** A production website at GitHub Pages.



**Figure 11.26:** An explicit `about.html` in the address bar.

## 11.6 Advanced Setup

This section contains some optional advanced Git setup. The main features are adding an alias for checking out branches, adding the branch name to the Unix prompt, and enabling branch name tab completion. Following the steps in this section should be within your capabilities if you completed *Learn Enough Command Line to Be Dangerous* (<https://www.learnenough.com/command-line>) and *Learn Enough Text Editor to Be Dangerous*, but they can be tricky, so use your technical sophistication (Box 8.2) if you



**Figure 11.27:** The About page in production.

get stuck. If you'd rather skip these steps for now, you can proceed directly to the conclusion (Section 11.7).

*Note for Mac users:* The instructions below assume you are using Bash, as described in Box 2.3. To learn how to set up your Git system using Z shell instead, see the Learn Enough blog post “Using Z Shell on Macs with the Learn Enough Tutorials” (<https://news.learnenough.com/mac-os-bash-zshell>).

### 11.6.1 A Checkout Alias

In Chapter 8, we added global configuration settings for the name and email address (Listing 8.3) to be included automatically when making commits. Now we'll add a third config setting, an *alias* to make it easier to check out branches.

**Table 11.1:** Important commands from Chapter 11.

Command	Description	Example
<code>git clone &lt;URL&gt;</code>	Copy repo (incl. full history) to local disk	<code>\$ git clone https://ex.co/repo.git</code>
<code>git pull</code>	Pull in changes from remote repository	<code>\$ git pull</code>
<code>git branch -a</code>	List all branches	<code>\$ git branch -a</code>
<code>git checkout &lt;br&gt;</code>	Check out remote branch and configure for	<code>\$ git checkout fix-trademark</code>

Throughout this tutorial, we've used `git checkout` to check out branches (e.g., Listing 10.3), but most experienced Git users configure their systems to use the shorter command `git co`.<sup>9</sup> The way to do this is with a Git *alias*: Much as the Bash aliases covered in Section 5.4 let us add commands to our Bash shell, Git aliases let us add commands to our Git system. In particular, the way to add the `co` alias is to run the command shown in Listing 11.13.

**Listing 11.13:** Adding an alias for `git co`.

```
$ git config --global alias.co checkout
```

In effect, this adds `co` as a new Git command, and running Listing 11.13 allows us to replace `checkout` in commands like

```
$ git checkout main
```

with the more compact `co` command, as follows:

```
$ git co main
```

For maximum compatibility with systems that don't have `co` configured, this tutorial has always used the full `checkout` command, but in real life I nearly always use `git co`.

9. This choice is no doubt influenced by the analogous command `svn co` used by Subversion, one of Git's main predecessors.

## 11.6.2 Prompt Branches and Tab Completion

In this section, we'll add two final advanced customizations. First, we'll arrange for the command-line prompt to include the name of the current branch. Second, we'll add the ability to fill in Git branch names using *tab completion* (Box 2.4), which is especially convenient when dealing with longer branch names. Both of these features come as shell scripts with the Git source code distribution, which can be downloaded as shown in Listing 11.14.

---

**Listing 11.14:** Downloading scripts for branch display and tab completion.

---

```
$ curl -o ~/.git-prompt.sh -L https://cdn.learnenough.com/git-prompt.sh
$ curl -o ~/.git-completion.bash \
>     -L https://cdn.learnenough.com/git-completion.bash
```

---

Here the **-o** flag arranges to save the files locally under slightly different names from the ones on the server, prepending a dot `.` so that the files are hidden (Section 2.2.1) and saving them in the home directory `~`.

After downloading the scripts as in Listing 11.14, on some systems we need to make them executable, which we can do with the **chmod** command (Section 7.3):

```
$ chmod +x ~/.git-prompt.sh
$ chmod +x ~/.git-completion.bash
```

Next, we need to tell the shell about the new commands, so open up the Bash profile file in your favorite editor (which for simplicity I'll assume is Atom):

```
$ atom ~/.bashrc
```

Then add the configuration shown in Listing 11.15 to the bottom of the file. Also, make sure to delete any other lines starting with **PS1** (which you'll have to do if you modified **.bashrc** as shown in Listing 6.6).

**Listing 11.15:** Adding Git configuration to Bash.

```
~/ .bashrc
```

---

```
.
.
.
# Git configuration
# Branch name in prompt
source ~/.git-prompt.sh
PS1='\W$(__git_ps1 " (%s)")\ $ '
export PROMPT_COMMAND='echo -ne "\033]0;${PWD/#$HOME/~}\007"'
# Tab completion for branch names
source ~/.git-completion.bash
```

---

*Note:* The vertical dots in Listing 11.15 indicate omitted content and should not be copied literally. This is the sort of thing you can figure out using your technical sophistication (Box 8.2). Speaking of which, I have hardly any idea of what most of the code in Listing 11.15 means; part of having technical sophistication means being able to copy things from the Internet and getting them to work even when you have no idea what you’re doing (Figure 11.28<sup>10</sup>).

Once we’ve saved the result of editing `.bashrc`, we have to `source` it to make the changes active (as seen in Listing 5.5):

```
$ source ~/.bashrc
```

At this point, the prompt for a Git repository’s default `main` branch should look something like this:

```
[website (main)]$
```

If you skipped ahead from Section 8.1 to complete this section, you’ll have to wait until Section 8.2 to see this effect. Checking that tab completion is working is left as an exercise (Section 11.6.3).

### 11.6.3 Exercises

1. Create a branch called `really-long-branch-name` using `git co -b`.
2. Switch back to the `main` branch using `git co`.

---

10. Image courtesy of Adam Frank/Shutterstock.



**Figure 11.28:** It's OK—neither does anyone else.

3. Check out the branch **really-long-branch-name** using tab completion by typing **git checkout r**→ at the command-line prompt.
4. What does your prompt look like? Verify that the correct branch name appears in the prompt.
5. Check out the **main** branch using **git co m**→. (This shows that tab completion works with the **co** alias set up in Listing 11.13.) What does the prompt look like now?
6. Use **git branch -d r**→ to delete **really-long-branch-name**, thus verifying that tab completion works with **git branch** as well as with **git checkout**. (In fact, tab completion works with most relevant Git commands.)



## 11.7 Conclusion

Congratulations! You now know enough Git to be *dangerous*—which means, with Part I and Part II, you know enough *developer tools* to be dangerous as well.

There's a lot more to learn, and if you continue down this technical path you'll keep getting better at using Git for years to come, but with the material in this tutorial you've got a great start. For now, you're probably best off working with what you've got, applying your technical sophistication (Box 8.2) when necessary. Once you've gotten a little more experience under your belt, I recommend seeking out additional resources. Here are some suggestions for getting started:

- *Pro Git* by Scott Chacon and Ben Straub (<https://git-scm.com/book/en/v2>)
- Learn Git at Codecademy (<https://www.codecademy.com/learn/learn-git>)
- Git tutorials (<https://www.atlassian.com/git/tutorials>) by Atlassian (makers of Bitbucket)
- Tower Git tutorials (<https://www.git-tower.com/learn/>)

At this point, you are in an excellent position to collaborate with millions of software developers around the world. You are also well on your way to becoming a developer yourself. Regardless of your ultimate goals, you can continue improving your dev skills with the rest of the core Learn Enough sequence:

### 1. *Learn Enough Developer Tools to Be Dangerous*

- (a) Part I: *Learn Enough Command Line to Be Dangerous*
- (b) Part II: *Learn Enough Text Editor to Be Dangerous*
- (c) Part III: *Learn Enough Git to Be Dangerous* (you are here)

### 2. **Web Basics**

- (a) *Learn Enough HTML to Be Dangerous* (<https://www.learnenough.com/html>)
- (b) *Learn Enough CSS & Layout to Be Dangerous* (<https://www.learnenough.com/css-and-layout>)
- (c) *Learn Enough JavaScript to Be Dangerous* (<https://www.learnenough.com/javascript>)

### 3. **Application Development**

- (a) *Learn Enough Ruby to Be Dangerous* (<https://www.learnenough.com/ruby>)
- (b) *Ruby on Rails Tutorial* (<https://www.railstutorial.org/>)

(c) *Learn Enough Action Cable to Be Dangerous* (<https://www.learnenough.com/action-cable>) (optional)

Good luck!

*This page intentionally left blank*

# Index

---

## Symbols

- \* (star), 32
- ^ (Ctrl), 20
- > (angle bracket), 236, 271
- \ (backslash), 236, 271
- >: (redirect) operator, 27
- >> (append) operator, 27
- 80-column limit, 164–165

## A

- About pages, 243
  - adding images to, 271, 272
  - links to, 246
  - opening, 245
  - soft wraps, 266
- absolute paths, 61
- access, personal access tokens, 225
- accounts, GitHub, 221–222. *See also* GitHub
- adding
  - collaborators, 262
  - commit messages, 215, 219
  - HTML tags, 210–215
  - images, 235–240, 271, 272
  - indented lines, 161
  - links to About pages, 246
  - paragraphs (HTML), 216
  - README files, 227–233
  - repositories to GitHub, 221, 222–227 (*see also* GitHub)
  - structures (HTML), 216–220
  - tags (HTML), 214
- advanced text editing, 145. *See also* text editors
- alerts, macOS terminal, 6
- algorithms, SHAs (Secure Hash Algorithms), 207, 255
- aliases, defining, 94
- alt attributes, 272, 276
- Amazon Web Services. *See* AWS (Amazon Web Services)
- anchor tags, 246
- Android, 5
- angle bracket (>), 236, 271
- appending files, 26–29
- append (>) operator, 27
- applications
  - categories of, 197
  - developing, 10
  - Rails Tutorial (Atom), 176
  - types of, 306
  - version control, 197
- applying
  - GitHub Pages, 292–294
  - multiple panes, 179–181
  - tab triggers, 151
  - technical sophistication, 222
- arguments, 8
- Atom, 82, 88, 103
  - 80-column limit, 164–165

- dedenting lines, 163, 164
- highlighting syntax, 111–113
- installing shell commands, 107
- navigating, 117–119
- opening files, 118
- opening HTML files, 212
- previewing markdown, 113–114, 115
- Rails Tutorial, 176
- README files, 230
- Ruby programs in, 154
- selecting, 105
- selecting documents, 124–125, 126
- starting, 106–114

autocomplete, 145–147

- cross-references, 146, 147

automating common tasks, 166. *See also* scripts

AWS (Amazon Web Services), 308, 309

## B

backslash (\), 236, 271

bar, 31, 32

Bash, 90. *See also* scripts

- `export` command, 172
- profiles, 95
- quitting files, 92
- saving files, 137
- switching macOS to, 35–36

bash shell, 70

baz, 31, 32

body tags (HTML), 216

boldface text, 79, 81, 245

branching, 243–251

- formatting, 285
- names (Git), 202, 203, 243
- pushing branches, 283–292
- rebasing, 251

browsers, consistent behavior across, 247

## C

caches, 226, 293

caret (^), 20

case-sensitivity

- grepping, 53, 95
- less programs, 100

`cat` command, 27

- viewing files, 28

`cd` command, 65, 66, 68

characters

- confirming, 289, 292–294
- encoding, 287
- non-ASCII, 287
- printing, 10

`checkout` command, 249, 254

cleaning up command lines, 23

`clear` command, 23

CLIs (command-line interfaces), 5

cloning, 261–266

closing tags (HTML), 213

Cloud9, 103, 308, 309, 310. *See also* modern text editors

- customizing, 188–191
- highlighting syntax, 111–113
- IDE (integrated development environment), 107
- navigating filesystems, 110
- previewing markdown, 113–114
- starting, 106–114
- word wrap, 112, 113, 114

code. *See also* source code

- 80-column limit, 164–165
- commenting out, 155–156
- dedenting, 156–164
- `goto` line number, 164
- hello world, 153
- highlighting syntax, 153–155
- indenting, 156–164
- snippets, 188
- writing, 152–166

collaboration, 201, 202, 259

- adding collaborators, 262
- cloning, 261–266
- conflicting changes, 276–280
- conflicts, 269–282
- non-conflicting changes, 270–276
- push, pull, clone, 260–266
- pushing branches, 283–292
- version control, 198 (*see also* version control)

columns, 80-column limit, 164–165

combining commands, 69

command line

- cleaning up, 23

- commands, 5
- editing, 20–23
- man pages, 15–20
- mathematics, 18–19
- overview of, 3–6
- running terminals, 6–10
- tab completion, 37–38
- troubleshooting, 14–15
- writing commands, 10–15
- Xcode command-line tools, 314
- command-line interfaces. *See* CLIs (command-line interfaces)
- commands, 5, 8
  - cat, 27
  - cd, 65, 66, 68
  - checkout, 249, 254
  - clear, 23
  - combining, 69
  - cp, 36
  - curl, 106
  - diff, 28–29, 208–209
  - echo, 10, 19, 23, 26, 28, 253
  - ekill (escalating kill), 168, 169, 170, 171, 173
  - exit, 23
  - export, 172
  - find, 68
  - formatting, 38–39
  - Git, 200 (*see also* Git)
  - git add, 205
  - git commit, 206, 208
  - git status, 205, 206
  - grep, 52–58, 68, 73
  - head, 46–48
  - installing shell commands (Atom), 107
  - kill, 167
  - less, 51
  - lr, 94
  - ls, 29–34
  - man, 16 (*see also* man pages)
  - mkdir, 64–66
  - Most Important Vim Command, 87–88, 92
  - mv, 36, 39
  - navigating, 99
  - open, 68
  - patterns, 11
  - pwd, 65
  - redirecting, 47
  - Redo, 134
  - rehash, 320
  - repeating previous, 44–45
  - rm, 38
  - rmdir, 71, 72
  - rm -rf, 72
  - search (/), 100
  - source, 95
  - ssh (secure shell), 6
  - starting, 10
  - sudo, 63
  - tail, 46–48
  - touch, 31
  - Undo, 134
  - Unix, 6
  - unzip, 175
  - vim, 85, 90
  - which, 43, 321
  - writing, 10–15
  - x, 96, 97
- commenting out code, 155–156
- commit messages, 206, 215, 219. *See also*
  - committing
- commit rights, 259
- committing
  - files, 235–245
  - repositories, 204–207, 210
- common tasks, automating, 166. *See also* scripts
- computers, 3
  - learning jargon, 25–26
- configuring
  - autocomplete, 145–147
  - commands, 38–39
  - development environments, 306, 307
  - directories, 64–66
  - files, 32
    - .gemrc files, 320
  - Git, 200–202
  - GitHub, 293
  - global configuration settings, 201
  - ignoring files, 241–243
  - prompts, 9

- remote repositories, 222–227
- tab triggers, 147–152
- confirming
  - characters, 289, 292–294
  - files, 239, 253, 276
- conflicts. *See also* errors
  - conflicting changes, 276–280
  - deleting lines, 279
  - merging, 269–282
  - non-conflicting changes, 270–276
  - pulling, 269–282
- content, deleting, 96–97
- converting Markdown, 113
- copying
  - commands, 223
  - Cut/Copy/Paste menu items, 127–132
  - directories, 70–74
  - files, 35–39
  - Jumpcut, 128–132
  - projects, 221 (*see also* GitHub)
- cp command, 36
- credentials, 226
- cross-references, 146
  - autocomplete, 146, 147
- Ctrl (^), 20
- curl program, 43, 44
- customizing
  - domains, 293
  - escalating kill scripts, 168
  - prompts, 138
  - tab triggers, 149, 150
  - text editors, 188–191
- Cut/Copy/Paste menu items, 127–132
- cutting
  - Cut/Copy/Paste menu items, 127–132
  - Jumpcut, 128–132

## D

- debugging, 156
  - goto line number, 164
- dedenting, 156–164
- defaults, terminal prompts, 138
- define statements, 149
- defining tab triggers, 149, 150
- def tab trigger, 153

- Delete command, 134
- deleting
  - directories, 70–74
  - files, 35–39
  - lines, 98, 279
  - text, 132–135
- detecting file types, 171
- developing web applications, 10
- development environments, 305
  - cloud IDEs, 307–312
  - native OS setup, 312–322
  - options, 306–307
- diff command, 28, 208–209
- directories, 61. *See also* files
  - adding files to, 235
  - copying, 70–74
  - creating, 64–66
  - deleting, 70–74
  - listing, 30
  - modifying, 65, 67, 204
  - moving, 70–74
  - navigating, 66–70
  - renaming, 70–74
  - structures, 61–64
  - temp, 262
  - text\_files, 71
  - untracked, 239
- displaying. *See* viewing
- documents
  - editing, 79 (*see also* text editors)
  - pasting multiple times, 133
  - selecting, 124–125, 126
- domains, customizing, 293
- downloading
  - files, 43–46
  - long files, 96

## E

- echo command, 10, 16, 19, 23, 26, 28, 253
- editing
  - autocomplete, 145–147
  - command lines, 20–23
  - fuzzy opening, 176–179
  - global find and replace, 181–187
  - HTML (Hypertext Markup Language), 148

- large files, 97–101
  - moving windows, 264
  - projects, 175–188
  - small files, 89–91
  - splitting into multiple panes, 179–181
  - tab triggers, 147–152
  - text editors, 15, 79–84 (*see also* text editors)
  - text in Microsoft Word, 82
  - `ekill` (escalating kill) command, 168, 169, 170, 171, 173
  - elements
    - of dev environments, 306
    - void, 236
  - Emacs, 88
  - emphasis, text, 80, 81
  - emulated tabs, 156
  - encoding characters, 287
  - ending commands, 12
  - environment variables, 172
  - errors, 73
    - `goto` line number, 164
    - recovery, 252–258
    - syntax highlighting, 155
  - ESC (escape key), 91, 93, 94
  - escalating kill scripts, 167, 168, 169, 173
  - executable scripts
    - searching, 169
    - writing, 166–175
  - `exit` command, 23
  - `export` command, 172
  - expressions, regular (regexes), 53, 184
- F**
- files
    - adding to directories, 235
    - appending, 26–29
    - committing, 235–245
    - confirming, 239, 253, 276
    - conflicts, 270–286 (*see also* conflicts)
    - copying, 35–39
    - creating, 32
    - deleting, 35–39, 96–97
    - detecting file types, 171
    - downloading, 43–46, 96
    - editing, 97–101
    - editing small, 89–91
    - fuzzy opening, 176–179
    - `.gemrc`, 320
    - hidden, 33–34
    - hierarchies, 61 (*see also* directories)
    - ignoring, 241–243
    - listing, 29–34
    - modifying, 25–26, 208
    - modifying system files, 61
    - moving to end of, 121
    - opening, 106–114, 118
    - opening in multiple panes, 182
    - previewing, 230
    - quitting, 91–96
    - README, 227–233
    - redirecting, 26–29
    - renaming, 35–39
    - saving, 91–96, 135–138
    - searching, 73
    - temporary, 242
    - tools, 49–52
    - unstaged, 205
    - untracked, 205, 239
    - viewing, 28, 43, 100 (*see also* viewing)
    - viewing HTML, 213
    - word wrap, 108, 109, 110, 111
  - filesystems, navigating, 108, 109, 110
  - `find` command, 68
  - finding
    - clone URLs, 263
    - global find and replace, 181–187
    - regexes (regular expressions), 185, 186
    - results, 183
    - strings, 140
    - text, 138–143
  - Find menu, 138
  - flags, 9. *See also* options
  - folders, 30. *See also* directories; files
    - directories, 61
    - files, 32
  - `foo`, 31, 32
  - forcing file quits, 93
  - forking, 259
  - formatting. *See also* configuring
    - 80-column limit, 164–165



- autocomplete, 145–147
- branching, 285
- commands, 38–39
- dedenting, 156–164
- development environments, 306, 307
- directories, 64–66
- executable scripts, 166–175
- files, 32
- functions, 149
- goto line number, 164
- highlighting syntax, 153–155
- indenting, 156–164
- new work environments, 311
- prompts, 9
- remote repositories, 222–227
- tab triggers, 147–152
- tags, 245
- top-level headings (HTML), 211, 213

functions

- arguments of, 8
- formatting, 149

fuzzy opening, 106, 176–179, 245

## G

.gemrc files, 320

Git, 198

- adding structures (HTML), 216–220
- adding tags (HTML), 210–215
- branching, 243–251
- branch names, 202, 203
- collaboration, 259 (*see also* collaboration)
- commands, 200
- committing repositories (repo), 204–207, 210
- conflicts, 269–282
- error recovery, 252–258
- ignoring files, 241–243
- initializing repositories (repo), 203–204
- installing, 200–202
- logs, 274, 275
- upgrading, 201, 310
- viewing diff, 208–209
- viewing SHAs (Secure Hash Algorithms), 255
- workflow, 235 (*see also* workflow)

git add command, 205

git commit command, 206, 208

GitHub, 221

- adding README files, 227–233
- configuring, 293
- creating remote repositories, 222–227
- passwords, 226
- security, 225
- settings, 293
- starting, 221–222
- templates, 224

GitHub Pages, 292–294

git status command, 205, 206

global configuration settings, 201

grep command, 68, 73

grepping, case-sensitivity, 53, 95

groups, match, 184, 187

## H

hashes, 207

- SHAs (Secure Hash Algorithms), 255

head command, 46–48

head tags (HTML), 216

hidden files, 33–34

hierarchies

- files, 61 (*see also* directories)
- navigating, 65–70

highlighting syntax, 111–113, 153–155, 170, 171

history, version control, 198, 199. *See also* version control

holy wars, 88

Homebrew package manager, 316–317

home directories, 62

HTML (Hypertext Markup Language), 81

- About pages, 243
- adding tags, 210–215, 214
- anchor tags, 246
- confirming characters, 292–294
- converting Markdown, 113
- editing, 148
- opening, 212
- structures, 216–220
- tags, 236
- templates, 283, 284
- top-level headings, 211, 213
- viewing, 213

HTTP (HyperText Transfer Protocol), 226  
 Hypertext Markup Language. *See* HTML  
 (Hypertext Markup Language)  
 hypertext references, 246  
 HyperText Transfer Protocol. *See* HTTP  
 (HyperText Transfer Protocol)

**I**

Identity and Access Management (IAM), 309  
 IDEs (integrated development environments),  
   82, 83, 107, 200, 201  
   cloud IDEs, 307–312  
   native OS setup, 312–322  
   options, 306–307  
   starting, 307, 308, 309  
 ignoring files, 241–243  
 images  
   adding, 235–240, 271, 272  
   sources, 237  
   tags, 236  
 indenting, 156–164  
 initializing  
   **rbenv**, 318  
   repositories (repo), 203  
 inserting text, 90, 91. *See also* adding  
 insertion mode, 87  
 installing  
   Git, 200–202  
   Homebrew, 316–317  
   iTerm, 313, 314  
   **rbenv**, 317  
   Ruby, 316, 317–320, 319  
   shell commands (Atom), 107  
   Xcode command-line tools, 314  
 integrated development environment. *See* IDE  
 interfaces  
   CLIs (command-line interfaces), 5  
   man pages, 16  
 iOS, 5  
 italicized text, 245  
 iTerm, installing, 313, 314

**J**

jargon, learning, 25–26  
 Jargon File, 88  
 Jumpcut, 128–132

**K**

keyboards  
   Cut/Copy/Paste menu items, 127, 128  
   shortcuts, 23, 134, 138, 144  
   symbols, 21, 104  
 kill command, 167

**L**

labels, 146  
 languages  
   Python, 159 (*see also* Python)  
   Ruby, 152 (*see also* Ruby;  
     source code)  
 large files, editing, 97–101  
 launching. *See* starting  
 less commands, 51  
 less program, 16, 49–52, 100  
 limits, 80–column, 164–165  
 lines  
   adding, 161  
   commenting out, 156–164  
   deleting, 98, 279  
   selecting multiple, 124  
   selecting single, 123–124  
   word wrap, 108, 109, 110, 111  
 links  
   adding to About pages, 246  
   anchor tags, 246  
   image directories, 239  
   pasting, 130  
 Linux, 5, 83  
   Git (*see* Git)  
   running terminals, 7  
   setup, 321–322  
   terminal icons, 7  
 listing  
   directories, 30  
   files, 29–34  
 logs  
   Git, 275  
   viewing SHAs (Secure Hash Algorithms),  
     255  
 long files, downloading, 96  
 lr command, 94  
 ls command, 29–34

**M**

## macOS

- Git (*see* Git)
- Homebrew, 316–317
- menu items, 13
- Ruby, 317–320
- running terminals, 6–7
- setup, 313–321
- switching to Bash, 35–36
- terminal alerts, 6
- Xcode command-line tools, 314

main branch, branching, 244

man command, running, 16

man man, running, 17–18

man pages, 15–20

## Markdown

- autocomplete, 145, 146
- files, 106, 107, 112
- previewing, 113–114, 115, 231
- selecting links, 129
- tab triggers, 147

match groups, 184, 187

## matching

- regular expressions, 186
- text patterns, 184

mathematics, 18–19

## menus

- Cut/Copy/Paste menu items, 127–132
- Find, 138
- finding using, 139

merging, 243–251

- conflicts, 269–282
- non-conflicting changes, 270–276

## messages

- commit, 206, 219 (*see also* committing)
- rejection, 278

meta, 287, 288

metasyntactic variables, 32

meta usage, 237

Microsoft Windows, 6, 83. *See also* Windows

- Git (*see* Git)
- setup, 322

Microsoft Word, editing text in, 82

Minimum Viable Vim editor, 84–89. *See also* text editors

mixing tabs, 157

mkdir command, 64–66

## models

- collaboration, 259 (*see also* collaboration)
- editors, 86, 87

modern text editors, 103–104

- Cut/Copy/Paste menu items, 127–132
- deleting text, 132–135
- finding text, 138–143
- highlighting syntax, 111–113
- navigating, 117–119
- previewing markdown, 113–114
- replacing text, 138–143
- saving files, 135–138
- selecting text, 119–127
- selecting text editors, 104–106
- starting, 106–114
- undoing text, 132–135

## modes

- insertion, 87
- Vim editor, 87

## modifying

- collaboration, 265 (*see also* collaboration)
- conflicting changes, 276–280
- conflicts, 269–282
- Cut/Copy/Paste menu items, 127–132
- directories, 65, 67, 204
- files, 25–26, 208
- ls command, 33
- non-conflicting changes, 270–276
- system files, 61
- windows, 116

more program, 49–52

Most Important Vim Command, 87, 89, 92

## mouse

- clicking/dragging, 123
- moving, 120 (*see also* selecting text)

moving. *See also* navigating; selecting

- Cut/Copy/Paste menu items, 127–132
- directories, 70–74
- downloading files, 43–46
- to end of files, 121
- files, 35–39
- goto line number, 164

- mouse (*see also* selecting text)
- windows, 264
- multiple files, opening, 178
- multiple lines, selecting, 124
- multiple links, pasting, 130
- multiple panes
  - opening files in, 182
  - splitting text editors into, 179–181
- mv command, 36, 39

## N

- naming
  - branch names (Git), 202, 203, 243
  - directories, 70–74
- native systems, 307
- navigating
  - commands, 99–100
  - directories, 66–70
  - filesystems, 108, 109, 110
  - fuzzy opening, 176–179
  - goto line number, 164
  - modern text editors, 117–119
- new work environments, creating, 311
- non-ASCII characters, 287
- non-conflicting changes, 270–276
- normal mode, Vim editors, 87

## O

- online regex (regular expressions) builders, 55, 56
- open command, 68
- opening. *See also* starting
  - About pages, 245
  - files, 118
  - fuzzy opening, 106, 176–179, 245
  - HTML files, 212
  - multiple files, 178
  - in multiple panes, 182
  - projects, 175
  - tags (HTML), 213
- operating systems, 5, 6
  - running terminals, 6–10
  - setup (*see* setup)
- operators
  - append (>), 27
  - redirect (>:), 27

- options, 8
  - development environments, 306–307
- OSs (operating systems), 307
  - native OS setup, 312–322

## P

- package control, 191
- paragraphs (HTML), adding, 216
- passwords, GitHub, 226
- pasting
  - Cut/Copy/Paste menu items, 127–132
  - documents multiple times, 133
  - Jumpcut, 128–132
  - links, 130
- paths
  - absolute, 61
  - relative, 71
  - system, 169
- patterns
  - commands, 11
  - ls command, 32
  - text, 184
- personal access tokens, 225
- pipes, 47–48, 53
- plain text, editing, 79. *See also* text editors
- previewing, 116
  - files, 230
  - Markdown, 113–114, 115, 231
- printing, 21
  - characters, 10
- processes, 167
  - grepping, 52–58 (*see also* grepping)
- profiles, Bash, 95
- programming languages
  - Python, 159 (*see also* Python)
  - Ruby, 152 (*see also* Ruby; source code)
- programs. *See also* applications; utilities
  - bash (Bourne Again SHell), 70
  - curl, 43, 44
  - Jumpcut, 128–132
  - less, 49–52, 100
  - more, 49–52
  - Rails Tutorial (Atom), 176
  - source code, 152–166 (*see also* source code)
  - version control, 197

- projects. *See also* files
    - copying, 221 (*see also* GitHub)
    - editing, 175–188 (*see also* editing; text editors)
    - fuzzy opening, 176–179
    - global find and replace, 181–187
    - opening, 175
    - shipping, 228
    - splitting into multiple panes, 179–181
    - version control (*see* version control)
  - prompts, 8
    - customizing, 138
    - defaults, 138
    - defining, 9
  - protocols, HTTP (HyperText Transfer Protocol), 226
  - pulling, 260–266
    - conflicts, 269–282
    - non-conflicting changes, 270–276
  - pull requests, 259
  - pushing, 222–224, 260–261, 260–266
    - branches, 283–292
    - rejections, 273
  - pwd command, 65
  - Python, 159
- Q**
- quitting files, 91–96
- R**
- Rails Tutorial (Atom), 176
  - rbenv
    - initializing, 318
    - installing, 317
  - README files, adding, 227–233
  - reconfirming fixes, 291. *See also* confirming; conflicts
  - recovery, errors, 252–258
  - redirecting
    - commands, 47
    - files, 26–29
  - redirect (>:) operator, 27
  - Redo command, 134
  - references, hypertext, 246
  - regexes (regular expressions), 55, 184
    - matching, 186
    - searching, 53
  - rehash command, 320
  - rejection messages, 278
  - relative paths, 71
  - remote origin, 223, 286
  - remote repositories, 221, 222–227. *See also* GitHub
  - renaming. *See also* naming
    - directories, 70–74
    - files, 35–39
  - repeating previous commands, 44–45
  - replacing
    - global find and replace, 181–187
    - regexes (regular expressions), 185, 186
    - text, 138–143
  - repositories, 200
    - adding README files, 227–233
    - adding to GitHub, 221, 222–227 (*see also* GitHub)
    - collaboration, 259 (*see also* collaboration)
    - committing, 204–207, 210
    - initializing, 203–204
    - main, 244
    - pushing, 222–224, 260–261
    - upstream, 286
  - requests, pull, 259
  - restarting shell programs, 320. *See also* starting results
    - finding, 183
    - pipes, 53
    - searching, 54, 183
  - rights, commit, 259
  - rm command, 38
  - rmdir command, 71, 72
  - rm -rf command, 72
  - roles of Unix, 6
  - root directories, 61, 62
  - Ruby
    - commenting out, 155–156
    - editing projects, 175–188
    - indenting/dedenting, 156–164
    - installing, 316, 317–320, 319
    - programs in Atom, 154
    - writing source code, 152–166 (*see also* source code)

- Ruby on Rails, 242, 308
- running
  - 1s command, 31
  - Vim editor, 86
- S**
- Save command, 135
- saving files, 91–96, 135–138
- scripts
  - ekill (escalating kill), 167, 168, 173
  - writing executable, 166–175
- scrollbars. *See also* navigating
  - Sublime Text, 119
- searching
  - executable scripts, 169
  - files, 73
  - global find and replace, 181–187
  - grepping, 52
  - regexes (regular expressions), 53, 185, 186
  - results, 54, 183
  - strings, 50
  - Sublime Text editor, 190
- Secure Hash Algorithms. *See* SHAs (Secure Hash Algorithms)
- secure shell (ssh) command, 6
- security
  - credentials, 226
  - GitHub, 225
- selecting
  - Atom, 105
  - Cut/Copy/Paste menu items, 127–132
  - entire documents, 124–125, 126
  - multiple lines, 124
  - single lines, 123–124
  - single words, 122–123
  - Sublime Text editor, 104–105
  - text, 119–127
  - text editors, 104–106
  - VSCoDe, 105
- self-closing tags, 236
- settings. *See* configuring
- setup. *See also* configuring; formatting
  - Git, 200–202
  - global configuration settings, 201
  - Linux, 321–322
  - macOS, 313–321
  - Microsoft Windows, 322
  - native OS setup, 312–322
- SHAs (Secure Hash Algorithms), 207, 255
- shells, 15
  - Bash, 70
  - installing commands (Atom), 107
  - restarting, 320
  - scripts, 166 (*see also* scripts)
  - switching macOS to Bash, 35–36
  - tab completion, 37–38
- shipping projects, 228
- shortcuts, keyboards, 23, 134, 138, 144
- signup, GitHub, 221–222
- single lines, selecting, 123–124
- single words, selecting, 122–123
- site templates, 245
- small files, editing, 89–91
- snippets, code, 188
- Softcover, 242
- soft wraps, 265, 266
- source code
  - 80-column limit, 164–165
  - commenting out, 155–156
  - dedenting, 156–164
  - goto line number, 164
  - hello world, 153
  - highlighting syntax, 153–155
  - indenting, 156–164
  - writing, 152–166
- source command, 95
- sources, images, 237
- spaces, viewing, 156
- splitting text editors into multiple panes, 179–181
- Spotlight Search bars (macOS), 6
- ssh (secure shell) command, 6
- star (\*), 32
- starting
  - commands, 10
  - Git, 200–202
  - GitHub, 221–222
  - IDEs (integrated development environments), 307, 308, 309
  - modern text editors, 106–114

- terminals, 6–10
  - Vim editor, 85–89
- statements, define, 149
- status, Git, 205, 206
- Stephenson, Neal, 5
- stopping
  - files, 91–96
  - on whitespace, 122
- strings
  - finding, 140
  - searching, 50
- structures
  - adding (HTML), 216–220
  - directories, 61–64
  - tree, 251
- subdirectories, 72. *See also* directories
- Sublime Text editor, 83, 88, 103. *See also*
  - modern text editors
  - navigating, 117–119
  - package control, 191
  - searching, 189
  - selecting, 104–105
  - selecting documents, 124–125, 126
  - starting, 106–114
- sudo command, 62, 63
- superusers, 61, 63
- support, snippets, 188
- symbols, keyboards, 21, 104
- syncing branches, 251
- syntax
  - highlighting, 111–113, 153–155, 170, 171
  - push, 285
- system directories, 61
- system files, modifying, 61
- system paths, 169

## T

- tabs
  - completion, 37–38
  - emulated, 156
  - mixing, 157
  - terminal windows, 12
  - triggers, 145, 147–152, 149, 150, 151, 153
- tags
  - adding HTML, 210–215, 214
  - anchor, 246
  - formatting, 245
  - HTML (Hypertext Markup Language), 236
  - images, 236
  - self-closing, 236
- tail command, 46–48
- technical sophistication, 83–84, 85, 198, 221, 222, 307
- temp directories, 262
- templates
  - GitHub, 224
  - HTML (Hypertext Markup Language), 283, 284
  - site, 245
- temporary files, 242
- terminal programs, macOS, 313–314
- terminal prompts, defaults, 138
- terminals
  - Linux, 7
  - macOS, 6–7
  - menu items, 13
  - running, 6–10
  - windows, 8–9, 12
  - Windows, 7–8
- terseness, Unix, 56
- text. *See also* files
  - boldface, 79, 81, 245
  - deleting, 132–135
  - finding, 138–143
  - inserting, 90, 91
  - italicized, 245
  - patterns, 184
  - replacing, 138–143
  - saving, 135–138
  - selecting, 119–127
  - single words, 122–123
  - undoing, 132–135
  - viewing, 46 (*see also* viewing)
- text editors, 15, 79–84
  - autocomplete, 145–147
  - comparing to word processors, 81
  - customizing, 188–191
  - deleting content, 96–97
  - editing large files, 97–101

- editing projects, 175–188
- editing small files, 89–91
- global find and replace, 181–187
- macOS, 313–314
- Minimum Viable Vim, 84–89
- model editors, 86, 87
- modern, 103–104 (*see also* modern text editors)
- moving windows, 264
- saving files, 91–96
- searching, 190
- selecting, 104–106
- splitting into multiple panes, 179–181
- tab triggers, 145, 147–152
- Vim editor, 82
- writing executable scripts, 166–175
- writing source code, 152–166
- `text_files` directory, 71
- tokens, personal access, 225
- tools, 3. *See also* commands; utilities
  - editing, 81 (*see also* text editors)
  - find, 68
  - grepping, 52–58
  - man pages, 15–20
  - software development, 306 (*see also* development environments)
  - viewing, 49–52
  - Xcode command-line, 314
- top-level headings (HTML), 211, 213
- Torvalds, Linus, 199
- touch command, 31
- tracking version control, 198. *See also* version control
- tree structures, 251
- triggers, tab, 145, 147–152, 153
- troubleshooting
  - command line, 14–15
  - error recovery, 252–258
- typeface, 80
- types
  - of applications, 306
  - detecting file, 171
  - of directories, 61
- typing. *See* writing

## U

- Ubuntu Servers, 312
- Undo command, 134
- undoing text, 132–135
- Unix, 5
  - commands, 6, 46 (*see also* commands)
  - `diff` command, 28, 29
  - directories, 61–64 (*see also* directories)
  - error recovery, 253
  - formatting commands, 38–39
  - Git (*see* Git)
  - hidden files, 33–34
  - `ls` command, 29–34
  - `mv` command, 36, 39
  - processes, 167
  - `rm` command, 38
  - role of, 6
  - terseness, 38–39, 56
  - text editors, 87 (*see also* text editors)
  - tools, 49–52
  - `touch` command, 31
  - Unix tradition, 82
  - `which` command, 43
- unstaged files, 205
- untracked files, 205, 239
- `unzip` command, 175
- updating, 319
- upgrading, 201, 310, 319
- upstream repositories, 223, 286
- URLs (Uniform Resource Locators), 43, 45, 247, 262, 263
- user directories, 61
- UTF-8, 288
- utilities. *See also* commands
  - editing, 81 (*see also* text editors)
  - find, 68
  - grepping, 52–58
  - viewing, 49–52

## V

- variables
  - environment, 172
  - metasyntactic, 32
- VCS (version control systems), 198. *See also* Git
- version control, 197, 198. *See also* Git



- technical sophistication, 198
- version control systems. *See* VCS
- viewing
  - diff command, 208–209
  - files, 28, 43, 100
  - head command, 46–48
  - HTML (Hypertext Markup Language), 213
  - README files, 230
  - SHAs (Secure Hash Algorithms), 255
  - spaces, 156
  - splitting into multiple panes, 179–181
  - tab triggers, 149
  - tail command, 46–48
  - utilities, 49–52
- vim command, 85, 90
- Vim editor, 82, 84–89
  - modes, 87
  - running, 86
  - starting, 85–89
  - temporary files, 242
- Visual Studio Code. *See* VSCode
- void elements, 236
- VSCode, 103. *See also* modern text editors
  - selecting, 105
  - starting, 106–114

**W**

- web applications, developing, 10
- websites, 292–294. *See also* HTML (Hypertext Markup Language)
- which command, 43, 321
- whitespace, stopping on, 122
- wildcard characters, \* (star), 32

- windows
  - finding/replacing, 139
  - modifying, 116
  - moving, 264
  - terminals, 8–9, 12
- Windows. *See* Microsoft Windows
- wordcount, 47–48
- word processors, 87
  - comparing to text editors, 81
- words, selecting single, 122–123
- word wrap, 108, 109, 110, 111, 112, 113, 114
- workflow, 235
  - adding images, 235–240
  - branching, 243–251
  - collaboration, 259 (*see also* collaboration)
  - error recovery, 252–258
  - ignoring files, 241–243
- World Wide Web, 6
- writing
  - 80-column limit, 164–165
  - commands, 10–15
  - commenting out, 155–156
  - dedenting, 156–164
  - executable scripts, 166–175
  - goto line number, 164
  - highlighting syntax, 153–155
  - indenting, 156–164
  - source code, 152–166
- WYSIWYG (What You See Is What You Get), 81

**X**

- Xcode command-line tools, 314
- x command, 96, 97