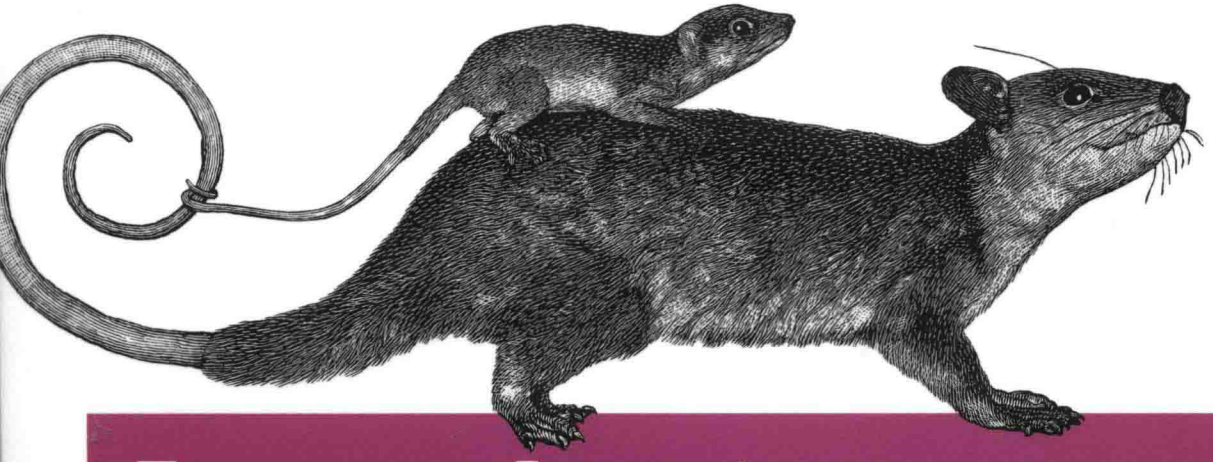
O'REILLY®



Introducing GitHub

A NON-TECHNICAL GUIDE

Peter Bell & Brent Beer

Introducing GitHub

A Non-Technical Guide

Peter Bell and Brent Beer

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo O'REILLY®

Introducing GitHub
by Peter Bell and Brent Beer

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://safaribooksonline.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editor: Meghan Blanchette
Production Editor: Melanie Yarbrough
Copyeditor: Sonia Saruba
Proofreader: Sharon Wilkey
Indexer: Judy McConville
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Rebecca Demarest

November 2014: First Edition

Revision History for the First Edition
2014-11-07: First Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781491949740* for release details.

978-1-491-94974-0

[LSI]

Preface

GitHub is changing the way that software gets built. Conceived originally as a way to make it easier for developers to contribute to open source projects, GitHub is rapidly becoming the default platform for software development. More than just a tool for storing source code, GitHub provides a range of powerful tools for specifying, discussing, and reviewing software.

Who This Book Is For

If you are working with developers on a software project, this book is for you, whether you are a:

- *Business stakeholder* who wants to have a sense of how your project is going
- *Product or project manager* who needs to ensure that software is delivered on time and within budget
- *Designer* who needs to deliver anything from mockups to HTML/CSS for a project
- *Copywriter* who's adding marketing copy or other content to a site or an app
- *Lawyer* who's reviewing the legal implications of a project or writing the terms and conditions or privacy policy
- *Team member* who needs to review, comment on, and/or contribute to the project
- *Developer* who is new to using GitHub and wants to learn how to collaborate using GitHub in a team

If you need to view the progress of a piece of software while it's being developed, if you would like to be able to comment on the progress, and if you'd like to have the option of contributing changes to the project, this book will show you how to effectively collaborate with a software development team by using GitHub.

Beyond Software

While GitHub is still primarily used to collaborate on the development of software, it's also a great way for a team to collaborate on a wide range of projects. From the authoring of books (like this one) and the distribution of models for 3D printing to the crafting of legislation, whenever you have a team of people collaborating on a collection of documents, you should consider using GitHub to manage the process. Our examples will assume that you're working on software because that is currently the most common use case, but this book is the perfect guide to collaborating via GitHub —whatever kind of project you're working on.

Who This Book Is Not For

This book is designed to teach the core skills required to collaborate effectively using GitHub. If you are already familiar with forking, cloning, and using feature branches and pull requests for collaboration, you probably won't learn that much.

Equally, if you are looking for an in-depth introduction to the Git version control system, this is not the book that you are looking for. This book covers just enough Git to do the job of introducing GitHub, but it's not a comprehensive introduction to Git. For that you should read the excellent *Version Control with Git* by Jon Loeliger and Matthew McCullough (O'Reilly, 2012).

How to Use This Book

We've deliberately made this book as concise as possible. You should be able to read it pretty quickly. If you want to gain the confidence that comes from really understanding what GitHub is about and how to use it, try to read the book from start to finish.

However, we know that you're busy. If you're in a rush, start by skimming the first chapter. Chapter 1 gives you a brief introduction to Git, GitHub, and some key terms that you'll need to understand to make sense of the rest of the book. Then feel free to just jump into whatever chapters you need. We've tried to write the book so that each chapter runs you through specific workflows, so you should be able to read just the chapter you need to complete a particular task.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width bold`
: Shows commands or other text that should be typed literally by the user.

`Constant width italic`
: Shows text that should be replaced with user-supplied values or by values determined by context.

This element signifies a tip or suggestion.

This element signifies a general note.

This element indicates a warning or caution.

Safari® Books Online

Safari Books Online is an on-demand digital library that delivers expert content in both book and video form from the world's leading authors in technology and business.

Technology professionals, software developers, web designers, and business and creative professionals use Safari Books Online as their primary resource for research, problem solving, learning, and certification training.

Safari Books Online offers a range of plans and pricing for enterprise, government, education, and individuals.

Members have access to thousands of books, training videos, and prepublication manuscripts in one fully searchable database from publishers like O'Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kauf-

mann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology, and hundreds more. For more information about Safari Books Online, please visit us online.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *http://bit.ly/intro-github*.

To comment or ask technical questions about this book, send email to *bookquestions@oreilly.com*.

For more information about our books, courses, conferences, and news, see our website at *http://www.oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Acknowledgments

Peter: I would like to thank my wife for her tireless support of the time and effort required to write this book—and the many other projects that keep me away from her more than I'd like that don't have acknowledgments sections! I would also like to thank my Mum for always going above and beyond to give me the support I needed to always follow my dreams—even under often difficult circumstances.

Brent: I'd like to thank my Mom for her constant encouragement for reading, without which I may never have found a love for it. And also my Dad. Without him letting me watch him work on our computer, entertaining me with the Oscar the Grouch trash can utility on our Macintosh, and encouraging me to learn how to program, I would not be in the field I am today.

We would both like to thank the inspiring Matthew and Jordan McCullough and the rest of the GitHub team for their feedback on this book and their ideas and support

over the years. Much of the best content here came from them. We'd also like to thank the amazing Meg Blanchette at O'Reilly, without whom this book would never have been conceived, written, or delivered—thanks so much, Meg!

Table of Contents

CHAPTER 1

Introduction

In this chapter we'll start by introducing Git and GitHub. What are they, what is the difference between them, and why would you want to use them? We'll then introduce some other common terms that you'll often hear mentioned when people are discussing GitHub. That way you'll be able to understand and participate in discussions about your projects more easily.

What Is Git?

Git is a version control system. A *version control system* is a piece of software designed to keep track of the changes made to files over time. More specifically, Git is a *distributed* version control system, which means that everyone working with a project in Git has a copy of the full history of the project, not just the current state of the files.

What Is GitHub?

GitHub is a website where you can upload a copy of your Git repository. It allows you to collaborate much more easily with other people on a project. It does that by providing a centralized location to share the repository, a web-based interface to view it, and features like *forking*, *pull requests*, *issues*, and *wikis*, which allow you to specify, discuss, and review changes with your team more effectively.

Why Use Git?

Even if you're working on your own, if you are editing text files, there are a number of benefits to using Git. Those benefits include the following:

The ability to undo changes
: If you make a mistake, you can go back to a previous point in time to recover an earlier version of your work.

A complete history of all the changes
: If you ever want to see what your project looked like a day, week, month, or year ago, you can *check out* a previous version of the project to see exactly what the state of the files was back then.

Documentation of why changes were made
: Often it's hard to remember *why* a change was made. With *commit messages* in Git, it's easy to document for future reference why you're making a change.

The confidence to change anything
: Because it's easy to recover a previous version of your project, you can have the confidence to make any changes you want. If they don't work out, you can always get back to an earlier version of your work.

Multiple streams of history
: You can create different *branches* of history to experiment with different changes to your content or to build out different features independently. You can then *merge* those back into the main project history (the *master branch*) once they're done, or delete them if they end up not working out.

Working on a team, you get an even wider range of benefits when using Git to keep track of your changes. Some of the key benefits of Git when working with a team are:

The ability to resolve conflicts
: With Git, multiple people can work on the same file at the same time. Usually Git will be able to merge the changes automatically. If it can't, it'll show you what the conflicts are and will make it easy for you to resolve them.

Independent streams of history
: Different people on the project can work on different *branches*, allowing you to work on separate features independently and then merge the features when they're done.

Why Use GitHub?

GitHub is much more than just a place to store your Git repositories. It provides a number of additional benefits, including the ability to do the following:

Document requirements
: Using *Issues*, you can either document bugs or specify new features that you'd like to have your team develop.

Collaborate on independent streams of history
: Using branches and *pull requests*, you can collaborate on different branches or features.

Review work in progress

By looking at a list of pull requests, you can see all of the different features that are currently being worked on, and by clicking any given pull request, you can see the latest changes as well as all of the discussions about the changes.

See team progress

Skimming the *pulse* or looking through the *commit history* allows you to see what the team has been working on.

Key Concepts

There are a number of key concepts that you'll need to understand to work effectively with Git and GitHub. Here is a list of some of the most common terms with a short description of each and an example of how they might be used in conversation:

Commit

Whenever you save your changes to one or more files to history in Git, you create a new commit. *Example usage: "Let's commit these changes and push them up to GitHub."*

Commit message

Every time you make a commit, you need to supply a message that describes *why* the change was made. That commit message is invaluable when trying to understand later why a certain change was implemented. *Example usage: "Make sure to include Susan's comment about the new SEC guidelines in the commit message."*

Branch

An independent series of commits off to one side that you can use to try out an experiment or create a new feature. *Example usage: "Let's create a branch to implement the new search functionality."*

Master branch (master)

Whenever you create a new Git project, there is a default branch created that is called *master*. This is the branch that your work should end up on eventually once it's ready to push to production. *Example usage: "Remember never to commit directly to master."*

Feature (or topic) branch

Whenever you're building a new piece of functionality, you'll create a branch to work on it. That's called a *feature branch. Example usage: "We've got way too many feature branches. Let's focus on getting one or two of these finished and into production."*

Release branch

If you have a manual QA process or have to support old versions of your software for your customers, you might need a release branch as a place to make any necessary fixes or updates. There is no technical difference between a feature or release branch, but the distinction is useful when talking about a project with your team. *Example usage: "We've got to fix the security bug on all of our supported release branches."*

Merge

This is a way to take completed work from one branch and incorporate it into another branch. Most commonly you'll merge a feature branch into the master branch. *Example usage: "Great job on the 'my account' feature. Could you merge it into master so we can push it to production?"*

Tag

A reference to a specific historic commit. Most often used to document production releases so you know exactly which versions of the code went into production and when. *Example usage: "Let's tag this release and push it to production."*

Check out

To go to a different version of the project's history to see the files as of that point in time. Most commonly you'll check out a branch to see all of the work that has been done on it, but any commit can be checked out. *Example usage: "Could you check out the last release tag? There's a bug in production that I need you to replicate and fix."*

Pull request

Originally, a pull request was used to request that someone else review the work you completed on a branch and then merge it into master. Now, pull requests are often used earlier in the process to start a discussion about a possible feature. *Example usage: "Go create a pull request for the new voting feature so we can see what the rest of the team thinks about it."*

Issue

GitHub has a feature called Issues that can be used to discuss features, track bugs, or both. *Example usage: "You're right, the login doesn't work on an iPhone. Could you create an issue on GitHub documenting the steps to replicate the bug?"*

Wiki

Originally developed by Ward Cunningham, wikis are a lightweight way of creating web pages with simple links between them. GitHub projects often use wikis for documentation. *Example usage: "Could you add a page to the wiki to explain how to configure the project to run on multiple servers?"*

Clone

Often you'll want to download a copy of a project from GitHub so you can work on it locally. The process of copying the repository to your computer is called *cloning*. *Example usage: "Could you clone the repo, fix the bug, and then push the fix back up to GitHub later tonight?"*

Fork

Sometimes you don't have the necessary permission to make changes directly to a project. Perhaps it's an open source project written by people you don't know or it's a project written by another group at your company that you don't work with much. If you want to submit changes to such a project, first you need to make a copy of the project under *your* user account on GitHub. That process is called *forking* the reposi-