

**ROCKABLE\***

# **GETTING GOOD WITH GIT**

**Includes  
Screen  
Casts!**



**Andrew Burgess**



**ROCKABLE\***

[Rockablepress.com](http://Rockablepress.com)  
[Envato.com](http://Envato.com)

© Rockable Press 2010

All rights reserved. No part of this publication may be reproduced or redistributed in any form without the prior written permission of the publishers.

<b>Introduction to Git</b>	<b>6</b>
<i>What is Git?</i>	6
<i>Why should I use a Version Control System?</i>	8
<i>Where did Git come from?</i>	9
<i>Why not use another Source Code Manager?</i>	10
<i>Summary</i>	11
<b>Commands</b>	<b>13</b>
<i>A Warning</i>	13
<i>Another Warning</i>	14
<i>Opening the Command Line</i>	14
<i>What You're Looking At</i>	15
<i>Commands</i>	16
<i>Advanced Command Line Skills</i>	23
<i>Summary</i>	24
<b>Configuration</b>	<b>26</b>
<i>Installing Git</i>	26
<i>Configuring Git</i>	31
<i>Using Git</i>	32
<i>Referencing Commits: Git Treeishes</i>	49
<i>Summary</i>	50
<b>Beyond the Basics</b>	<b>52</b>
<i>Git Add Revisited</i>	52
<i>Git Commit Revisited</i>	62
<i>Git Reset</i>	64
<i>Git Checkout Revisited</i>	65
<i>Git Diff</i>	65
<i>Git Stash</i>	67
<i>Working with Remote Repositories</i>	68
<i>Git Rebase</i>	75
<i>Summary</i>	77

<b>GitHub</b>	<b>79</b>
<i>What is GitHub?</i>	79
<i>Tour of the Tools</i>	82
<i>Creating a GitHub Repository</i>	87
<i>Forking a Repository</i>	91
<i>Summary</i>	94
<b>Appendix A: More Git Resources</b>	<b>96</b>
<b>Appendix B: Git GUIs</b>	<b>98</b>
<b>About The Author</b>	<b>99</b>
<b>Your Screencasts</b>	<b>100</b>



# Introduction to Git

So, you want to learn about Git? Then you've come to the right place. In this ebook, I'll be guiding you through the sometimes-confusing waters of using Git to manage your development projects. But before we get into that, let's take a step back and discover what exactly Git is, and why you would want to use it.

## What is Git?

Git is a source code manager (SCM). More accurately, it's a distributed version control system. But what exactly does that mean? Let's break it down.

## Git is Development Software

First off, Git is not a language, concept, or best practice. It's a program, a piece of software that you should use in your development, just like your text editor or FTP client. So what is it for? Git manages your source code... but what does that mean?

## Git is for Versioning

The idea behind Git (and the other source code managers, which we'll talk about soon) is that it's a smart idea to keep snapshots of your coding projects. This means that, as you code, at intervals you declare a point in the history of the project. Think of a

**ROCK\***  
**SCREENCAST**

see:  
*git\_chapter\_1.mp4*



timeline, where all the markers represent milestones in your coding. Where you put these markers is up to you, but it's wise to put them in when you've completed something. For example, you might make a mark after each feature you've implemented. That way your project timeline might look something like this:

- mark 1 — started the project, added the CSS and JS files
- mark 2 — built the main structure of the website
- mark 3 — added the navigation

I hope you're warming up to the idea of code snapshots. To steal an example from Tom Preston-Werner (a prominent member of the Git community) it's like taking a photo of your child every year, to track her growth. So if you're grokking the concept of timeline-markers for code, I think you'll see where Git fits in. If you wanted to make these snapshots manually, you'd have to copy your project directory and rename it every time. Git does this, and much more, for you.

## Git is Distributed

That's why Git is called a version control system. But it's also distributed. Distributed, in version control systems, means that the code repositories (that's what your project folders are officially called) don't need to have a single home. With some systems, the main repository is kept on a server, and you as the programmer just get a copy of the most recent version of the project to work from. Then, when you want to create a new snapshot, you have to send that snapshot back to the server.

Let's back up a second: version control systems are made so that more than one person can work on the same project at once. Each programmer gets the code, works on it, and takes snapshots

(which are properly called *commits*). Then, they can share their commits with each other.

So, you make a commit and send it back to the server. This is NOT a distributed system. It required access to the server to make a commit or get updates from others (who have sent their changes to the server). Git is just the opposite. Almost everything you do with Git is done on your own machine. Of course, you can still share your repository with others, but not in the non-distributed way: you can send commits directly to other computers, and get commits back from them. Also, every copy of the repository includes the full commit history with Git. That's not true of some non-distributed (or centralized) systems, where you get only the latest commit from the server. The distributed way is safer, in that every copy of the repository is a full copy, so there should be no great loss if one dies. With centralized systems, if the server goes down, the commit history goes with it.

## Why should I use a Version Control System?

We already saw that a system like Git makes it easy to create a code timeline, but why would you want to do that? And are there other benefits? Let's discuss some reasons why you should use a source code manager.

### Freedom to Play

When you're using a version control system (VCS) and making commits regularly, you don't have to worry about breaking anything. If you really mess something up, just roll back to the latest commit and keep going.

## Freedom to Branch

We'll talk more about branching in Chapter 3, but it basically lets you take your project in two or more directions within a single repository (trust me, it will feel magical). This is useful in situations where you're building version two of a project, but need to supply bug fixes for version one. Or, maybe you want to implement a rather challenging feature. It would definitely be a good move to give it a branch of its own. Without branching, you'd have to duplicate your project directory each time you wanted to try something new: not fun.

## Freedom to Share

Using a VCS makes it really easy to share your project with others, and let them help you develop it. Without a VCS, you'd have to copy, compare, and integrate their changes all by hand. Again, not fun.

## Where did Git come from?

Time for a history lesson: Git was created by Linus Torvalds, the creator of the Linux kernel. Actually, Linus built Git for managing the code for the Linux kernel. He looked at the available SCMs but came to the conclusion that none of them were fast enough. So, he built his own. You can have the peace of mind that Git will be able to handle your projects with blazing speed and — unless you're building an operating system — super-high efficiency.

## Why not use another Source Code Manager?

Hopefully by now you're convinced that it's a good idea to use a Version Control System. But why Git? There are several other options out there, the main ones being:

- Subversion
- Mercurial
- Perforce
- Bazaar

What's wrong with these? Really, there's nothing inherently wrong with them, but here are a few reasons why you might prefer Git:

- Git is fast
- Git is easy to learn
- Git offers a staging area
- GitHub is available for sharing

There are other reasons, but they won't make much sense if you're not familiar with Git. You can read about these reasons and others in detail at [WhyGitsBetterThanX.com](http://WhyGitsBetterThanX.com). Once you've gotten the hang of Git, go back and review that site. When I did, I was surprised to see what other SCMs were missing!

## **Summary**

In this chapter, you've learned what Git is, where it came from, and why you should use it. Now, let's dive into the terminal and get our feet wet with a few commands!

2

# Commands

By default, Git is a completely command line-based program, so if you want to use it well, you should be familiar with the command line. If your command line-fu isn't quite so polished, this chapter will help you get up to speed.

## A Warning

Before we go anywhere near the command line, you need to know something: 99% of everything you do on the command line is irreversible. This is especially true of deleting things. When you remove files or folders from the command line, they don't go to the trash, the recycle bin, or any other kind of halfway house. They're gone forever (barring some kind of hardware level recovery... and even that won't always work). Therefore, it's smart to double-check your commands as you're getting used to the command line.

That said, working on the command line can be much more efficient and (if you're geeky like me) very fun. Don't get scared off by the cryptic nature of it. Take things slowly, and it'll be worth your time.

Also, don't worry about hurting anything today. All but one of the commands we're about to discuss are perfectly harmless.

There's one more thing to note: if a command was executed successfully, it usually won't give you any feedback. It may seem a bit confusing, but get used to the fact that if the command

**ROCK\***  
**SCREENCAST**

see:  
*git\_chapter\_2.mp4*



# About The Author



Andrew Burgess is a Canadian web developer and is the Associate Editor at [Nettuts+](#), where he has published numerous popular tutorials and screencasts. He's also a web development reviewer on Envato's [Tuts+ Marketplace](#), and as a web developer, he specializes in JavaScript and Ruby. Andrew lives with his family in Oshawa, Canada.

Check out Andrew's personal site at: <http://andrewburgess.ca/>

Or follow him on Twitter:  
[@Andrew8088](#)

# Your Screencasts

Use the links below to download your series of screencasts.

In *Getting Good with Git*, Nettuts+ Associate Editor Andrew Burgess will guide you through the sometimes-scary waters of source code management with Git, the fast version control system.

Git's speed, efficiency, and ease-of-use have made it the popular choice in the world of source code managers. And with a service like GitHub available for sharing your code, there's no question about whether learning Git is worth your time!

In this book, Andrew Burgess will take you from knowing nothing about source code management to being able to use Git proficiently. You'll look at why you should use a version control system, why Git is better than the other options, and how to set up and use Git. This book covers some of the advanced features of Git, and includes an appendix of other resources that will take your Git knowledge to the next level. We'll even get to know GitHub!

This book also includes a screencast series that walks you through most of the concepts that the book teaches. With over two and a half hours of video, you'll get a solid grounding in Git without much effort.

**ROCKABLE** ✨

**Get Good**