Painless Git

A Sane Person's Guide to Distributed Development

by Nate Dickson

Painless Git

A Sane Person's Guide to Distributed Development

Nate Dickson

This book is for sale at http://leanpub.com/painless_git

This version was published on 2020-06-02

Tweet This Book!

To Libbie, of course.

And also Linus Torvalds.

Contents

Introduction

What is git, and Why do I Need It?

If you've started dabbling in open source software, you've come up against links to sites like Github and Bitbucket. Places where people can work on code together, and you get to use their software free of charge. It's pretty great. But you may be wondering *how* that collaborative development works. How can a lot of people all work on the same code at the same time? If you've poked around these sites, you've seen things like "branches" and "cloning" and "pull requests" mentioned a bunch of times [1].

All of these terms are part of *git*. Git is a *version control system*, which is a fancy way of saying it lets people make changes to the code and then helps them merge all their differences together. Open source depends on teamwork, and for that matter so does any software project these days.

But even if you're working on a project solo, git can offer you great benefits. If you've ever made a change to your code that seemed *absolutely brilliant* at 2:23 am and was revealed to be *absolutely terrible* at 10:45 the next morning, git is how you, pardon me, "git"[^That is the last time I make that pun. I promise.] back to your functional code. If you are diligent about keeping git up to date on what you're doing, git will have your back and make sure you don't get too lost. In short, git is your safety net.

What This Book Aims to be

This book has a few simple goals: It should be:

Short

You have better things to do than read about git. You need to be using it to create something. I'll keep this brief so you can get back to what you're doing.

Cheap

Git is free software, and there are a *lot* of free resources to help you learn git out on the web. I believe that I've got something here that will be worth the money, but I also understand that a starting developer isn't usually rolling in cash. Unless you already have angel funding, in which case I hate you[2].

So *Painless Git* is priced to be easy to add to your arsenal, making git easy to add to your toolset.

[1]Maybe you've even read that if you use git, you can end up in a "detached head state," which seems needlessly risky.
[2]Just kidding, I love you. Please make me the CIO of your new company.

Easy to Understand

I have been through just about every online git tutorial and reference in my years learning and teaching git. And while many of them are packed with useful information, it's often information that is only useful *after* you're somewhat familiar with git. *Painless Git* will help you get over the initial hurdles and into a place where you can start reading about `reflog` and git's internal object representation and make sense of it.

Friendly!

The *Painless* books weren't named arbitrarily. Learning git doesn't have to be a struggle against monster odds. I'm coming into this book with the belief that you are intelligent, energetic, and in many ways smarter than I am. I like git, and I want you to have a good time with it as well.

Opinionated (But in a Good Way)

I'm gonna keep it light, but I've got *opinions* people. I like to flatter myself into believing that my views are born of long experience with git and are based on my innate knowledge of the *right* way to do things. But some of them are just opinions. I will do my best to present my ideas, and the possible opposing opinions, and let you make up your mind.

Lighthearted, but Serious

I'll keep the tone happy, but at the end of the day if I don't teach you anything my light tone isn't doing any good. All the advice I give in here is advice I follow on my development projects. I'm not going to steer you wrong, not even to prove a point. If I have to tell you a wrong way of doing things, I will mark it and call it out, so you don't mix it into your workflow.

What this book is *not*

This book isn't:

The End-all-be-all reference on git

I use and teach git; I didn't write it. Furthermore, it's not my goal to explain to you how git does internal file representation. I hope I get you to the point where you can understand the deeper mysteries of git[3], but I'm not going to be teaching those mysteries here.

[3]And maybe even to the point where you *want* to learn them. Git is fascinating, inside and out.

A List of Tips

A lot of books are presented in "cookbook" format, where you can pick and choose which tips you want or need. And that's awesome! I love those books. But what I'm presenting here is more of a guided course on git, with later chapters building on what you learned in earlier ones. If you're already somewhat experienced with git, or if you've already read the book, you can start to jump around. But I would recommend you go straight through the first time.

Who Am I?

I'm Nate, nice to meet you.

Beyond that, I'm the author of *Painless Vim* and *Painless Tmux*. I'm a professional developer and for the past few years have been a developer trainer in addition to my coding duties. When I'm not at work, I'm a blogger, MBA student, author, and daddy. If you want to get a better idea of who I am, you can check out my personal blog, my short stories blog, or look at my Twitter or Github accounts. If you like Doctor Who reposts you can check out my Tumblr.

References

References available upon request.

No, just kidding, that's a little resumé humor for you.

I'll be referencing some sites and books and whatnot. The joy of the modern age is that you don't have to learn in a vacuum. The git documentation from git-scm.com, in particular, is *excellent*, once you've got your feet wet.

Hey, Are There Any Conventions Used In this Book?

There sure are! If you've read either of my other two *Painless* Books, you can probably skip this section.

The first convention is the word "git." As is common with the names of executables, the official name is all lower case. If it's at the beginning of a sentence, I'll write it with an initial capital, because that's how English works, my friend.

Code will appear in a `monospaced font,` and longer code blocks will appear

```
In a big block
Of a monospaced font.
```

On the rare occasions where I need to refer to modifier keys they will appear with angle brackets around the name: `<shift>`, `<enter>`, `<ctrl>` and so forth. If you need to press a modifier key and another key at the same time I'll type them right next to each other, like this: `<ctrl>c`. This only applies to modifier keys. If I put `rm` that doesn't mean hit both keys at the same time, it means type an `r` and then an `m` like you were typing words. I realize I'm being pedantic here, but hey, it's worth getting all this out in the open. I don't want there to be any secrets between us, dear reader.

There are a few Leanpub-enabled section types that I'll use in the book.

> Asides are used to present lengthy, rambling sections of text that should help illuminate the main text of the chapter. They get this fancy gray box.

Warnings!

I will use warnings sparingly, but they are useful. In general, I try to never present you with a way of doing things that I wouldn't do myself, but sometimes I have to put in "what not to do" examples. Warnings are for that sort of thing.

Exercises!

Exercises are short, "Now you do it" activities. Have you ever read a tech book before, where they're like "you don't have to do the exercises, but you won't get nearly as much out of the book if you don't!" Have you read one of those books? Okay good, I'll skip the lecture then.

And there are a bunch of others. For the most part, they should be fairly obvious.

Let's "Git" Started!

That's the last time I will make the "get/git" pun in this book unless autocorrect messes me up.

How This Book is Structured

A brief guide to *Painless Git*.

If you've flipped through the table of contents a bit, you've no doubt seen that the text is split into three parts: Beginning, Refinement, and Sophistication. By the end of part 1, you should be able to confidently add things to your git repository, making some branches, fixing some small mistakes, and sharing your work with others. For a lot of people, this is as far as they go.

But when you've done this much, you're only a few steps away from a much bigger and better world!

In the "Refinement" section we talk about how to move from "working with git" to "making git work for you". This section is all about practices and habits you can adopt to smooth out your git workflow and catch a few extra bugs while you're working with git.

Git lends itself well to a very structured, predictable workflow. If we impose just a little bit of structure on ourselves, git will work so much faster and easier. We'll talk about some good habits to get into, take a break and talk about a neat git trick, then discuss issues of style and talk about keeping git clean and happy.

Finally, as a transition into the final section, we'll discuss the distributed nature of git and why it matters.

And then we get to the "Sophistication" section. We're going to delve into the `.git` directory and discover that working with git's guts is pretty painless after all. We'll start looking at hooks as a basis for automating even more of git's interactions with the wider world, and discuss how teams can use git's power to speed through code reviews. We'll dive into some of the more esoteric parts of git and, if we're up to it, we'll even try to figure out what the "reflog" is and why we should care.

What Are Interludes?

Learning a new system like git can be difficult, and sometimes it feels like you're getting everything thrown at you all at the same time. So from time to time in the text, I like to include *Interludes*. These are chapters that take a break from teaching you the main topics and cover something git-adjacent. The interludes aren't *exactly* optional; if you skip them, you'll still end up with reasonably solid git skills.

I intend that when you get to an interlude, you spend some time looking back over what you've learned so far and take a breather. Reaching an interlude is a good sign that you've reached the end of a section or topic, and now it's time to play a little and solidify your knowledge.

Welcome to the Sample!

Hi there!

This is just a *small sample* of Painless Git, but this little sample contains some of the really good stuff. At present the Table of Contents *looks* like it has the whole book, but that’s just Leanpub playing with you, showing you all the sections that are included in the full text. If you like the sample why not pick up the whole book?

Part I: Beginning

A Brief History of Git. The Briefest!

Okay, I know how annoying it is, you want to get into a new technology and the author is like "hey let's talk about a bunch of stuff that happened back when large men with bushy beards and suspenders ruled the programming landscape!" So I won't spend too long on this, but it's worth it to get a high-level overview of where we are and what the world of git looks like. Along the way I'll give you some reasons you really *really* want to use git.

The Linux Kernel

Git was created because SVN[4] was having a hard time keeping up with the Linux Kernel. Linux is the most popular open source project of all time. It's everywhere. If you have any mobile device [5] you have a small Linux device on your person. If you have been to a website, it's probably running on Linux. Basically, any time you need a small, free operating system for a computer-like thing, you probably have Linux running somewhere.

Anyway, with thousands of contributors and millions upon millions of lines of code, Linux[6] was pushing the very limits of what SVN could do. SVN is a *centralized* version control system. When you want to save your code, you put that code on the central server. When you want to retrieve code, you get it from the central server. It was getting to where the Linux Foundation was spending almost as much time managing svn as they did writing actual code.

So Linus Torvalds[7], the father of Linux, did something about it. Specifically, he hacked up ruby scripts to tie a bunch of Linux/Unix[8] commands together. The actual beating heart of git is fascinating, but outside the scope of this book, and way outside the scope of this chapter, which I promised would be brief.

Anyway

The open source world fell in love with git because even in its early days it solved a lot of the problems people had with SVN. But git really took off with the launch of Github, a site that allowed you to host a git repository for free as long as it was open to the public[9] . Github made it easy to do

[4]Subversion, or "SVN" or "svn", is a version control system that has been around for many years. They designed it to be better than the then-current "CVS" ("Content Versioning System"), and was therefore a "subversion" of the norm, and was also better at dealing with sub-versions of a code base. Programmer humor.

[5]other than an iPhone, which is using Unix.

[6]Specifically the Linux Kernel, or the "heart" of the operating system.

[7]And many others. Open Source software is usually a collaborative effort. But the story sounds better if one person heroically did it all himself.

[8]While Linux and Unix are very different operating systems with very different capabilities, they both adhere to the "Posix" standard, meaning you can be fairly certain that anything you can run on Unix will run on Linux and vice versa. There are other Posix based operating systems, but we don't care. Some people will just say `*nix` to refer to anything that is Posix compatible, but I will not. All you need to remember is that your server is probably running Linux and your Mac is running Unix under the hood. And Windows is in its own little non-Posix world.

[9]Github now offers *private* repositories to individuals with free accounts. This is a major change that has actually already been made by every other git hosting service, but it's nice to see Github join in.

all the things that open source projects like to do: collaborate on a project, get into arguments, split a code base into two competing projects, descend into flame wars about which fork of the project is better, and then either abandon them both or merge them back into one project. But I kid.

Except not really. You can do all those things from within Github. If you want to work on someone's git project, it's easy to make your own copy of the code, make a bunch of changes, and ask the maintainers of the original project to pull your changes into their project[10] .

So, with the advent of Github git's visibility to the rest of the world soared. Developers worldwide used it, and the folks at Github were savvy enough to offer paid plans that let you host private repositories for your proprietary projects. Git became the *de facto* standard for version control, and while there are still SVN and even CVS based projects[11] out there in slow-moving industries like academia and banking, most new projects will be shared in a git repository. So if you're just getting into version control git is what you want to learn.

There. We did it in less than a thousand words[12] and didn't put a single concrete date in the chapter anywhere. That's it for the history of git, now let's use it!

[10]And *that*, dear reader, is why it's called a "Pull request", not a "Merge Request", which would make infinitely more sense. But we'll get to that later.

[11]And others like Mercurial and Perforce and Team Foundation Server etc etc *ad infinitum ad nauseaum*. Look, even Microsoft is moving away from Team Foundation Server to git. By which I mean, "Microsoft now owns Github". Let us take it as read that there are other options, but in this book we're focusing on git.

[12]Including footnotes!

Installing Git

This is the shortest chapter. Depending on your OS you may already have git installed. Here's how to find out:

On Linux or Unix[13] just open a terminal and type `git --version`. You should get something like `git version 2.13.2` back. Congratulations, you have git installed.

On Windows you probably don't have it installed by default, but that's not a problem, because there's an amazing project out there called Git for Windows that installs git and makes working with it lyrically, beautifully simple. It adds git to your path, so you can run git commands from the command prompt or even from PowerShell. It also gives you an implementation of the ‘ bash‘ shell, so you can run every command in this book without modification.

Keeping Git up to Date

On Linux updating git should happen automatically via your package manager. On macOS it generally only gets updated annually, when the operating system is updated. But you can use Homebrew to update git more frequently.

At present, if you're using Git for Windows you'll need to check for updates manually. I generally check for updates monthly when I'm working on a Windows machine.

[13]Remember, Mac OS is technically a Unix operating system.

First Steps

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Creating a New Repository

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Quick Detour: Tell Git A Little About Yourself

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Okay, Back to Your First Commit

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Fear of Commitment

Now it's time for us to look at some of those things we blithely skipped over in the last chapter. We're going to take a look at commits in git, try to make sense of the mysterious heart of distributed version control. First we're going to look at *how* you make commits in git, and then we'll talk about *why* you do it that way.

The Three Step Dance

Let's say you're a developer. You have a git repository with some code in it. You've made some changes to your files and you want to save them on your team's shared git repository. The steps are:

1. **Stage** the files for commit.
2. **Commit** the files, writing a nice commit message.
3. **Push** your changes to the team repository.

If you've used a system like SVN in the past, it will feel like git adds some extra steps there. After all, the SVN workflow is this:

1. **Check-in** your code, writing a nice commit message.
2. There is no step 2.

What gives? Why am I teaching you about a system that takes *three times* as many steps to do the same thing? There are many reasons, but the simplest of them is that those two extra steps increase git's flexibility exponentially. We'll cover that in more detail later. For now just remember the three steps:

1. **Stage**
2. **Commit**
3. **Push**

We're going to talk about those steps in order.

Staging Files

Lets go back to our sample directory and create a few new files. You can call them whatever you like; I know better than to try and stop you. Once you've added your new files type `git add --all` and hit enter. You have just told git to start watching those new files you added. Well, technically you told it to add *every file* in the directory, but it's not dumb. It's not going to double down on files it already knows about.

The other thing you did is you *added* all changes to git's *staging area*. "Staging" in git terminology means getting things ready for the commit. When you *stage* a file, you tell git to take a snapshot of that file and hold onto that snapshot for a bit, you're probably going to commit it soon.

In practice things are often staged for about as long as it takes you to write a commit message, but when we get into the Refinement and Sophistication phases of the book we'll talk more about the power of the staging area in the hands of a careful developer.

What it Means to Commit

The usual definition of a "commit" in git is "a snapshot of the repository at a point in time". Which is *almost* true. We've just seen one exception to that rule: you can stage some files for commit and leave others out.

Remember that a "working copy" is a directory that git is watching for you. But just because git is watching a directory for you doesn't mean you've told it what to *do* about those files. Remember when we first added `README.txt` to our test directory? Git called it an "untracked file". If we had tried to commit right then it would have complained, telling us there wasn't anything for it to commit.

Let's refine our definition of a commit, then, and say that a commit is "a snapshot of all the *tracked* files in a repository at a point in time". Which, again, is *almost* true. I'm not teasing this out to annoy you, I promise. I'm going somewhere with this.

Remember when we were in that weird place, where git said that `README.txt` was both "staged for commit" and "not staged" at the same time? If we had committed right then git would have stored the empty file version of `README.txt` instead of the one with your pithy prose in it. Your deathless text would still be in the working copy, but you hadn't *staged* the changed version of the file. So at that moment in time you had the *staged* version of `README.txt` which was empty, and a *changed* version of `README.txt` that had some snarky comments in it. As far as git is concerned, those are two different files[14]

So this is going to be our final refinement of our understanding of a commit:

[14]Ugh, again, I'm simplifying. The way humans think about files and the way git thinks about files bear only a passing resemblance to one another. If you want to skip *way ahead* in the book you can read the Interlude called "Commits Revisited" near the end of the book.

> A commit is a snapshot of all tracked changes to all tracked files in a working copy at a point in time.

And that is the as-close-to-actually-true-as-makes-no-odds definition of what a commit is. And now that you know that, squirrel it away in the back of your mind, and when someone asks you what a git commit is just say "A snapshot of the repository at a point in time." It's far easier to say.

The Anatomy of a Commit

When we committed earlier, we saw a few interesting things that I said we'd talk about later. That later is now. Let's make another commit to show off a bit more of the information git gives you.

So, open our readme file again and add a few more lines of text. *Stage* the changes by typing `git add --all` and commit by typing `git commit -m "Added some more text"`. You should have gotten another commit message from git that looks something like this:

```
[master 88ed320] Added some more text
 1 file changed, 1 insertion(+)
```

Let's look at what git is telling us. The first line tells us what *branch* we are on (master) and the *commit hash* of the thing we just checked in. We are *absolutely* not going to worry about how git arrives at the commit hash. All we need to know is that there is a reasonably good chance that each commit *in the world* will have a unique hash. Which means we can use it as a unique identifier later, when we want to do fancy things with commits. You don't need to worry about that now, for now just know that the commit hash is this commit's "name" inside of git, and when you know a commit's name you can do dark magicks.

Degrees of Uniqueness

a git commit hash is a GUID, or a Globally Unique Identifier. There has been some discussion about whether or not it counts as a UUID (Universally Unique Identifier), but that conversation is not very interesting. The laws of probability state that a number as complex and large as a commit hash *shouldn't* collide with (meaning, "be the same as") another commit hash in the lifetime of the universe. That said, it happens. This is because probability likes to laugh at us.

The second line tells us some pretty obvious things. We changed one file, and we added one line to that file. The math git shows in these messages is always *linewise*, git really does not care how many characters were on a line. For example, say that we added the following text to our readme:

```
When I am learning
I often sum up in verse
Short, sharp, crystalline.
_
```

And then committed it. Git would tell us

```
    1 file changed, 4 insertions(+)
```

And it *wouldn't even care how many syllables are on each line.* All it cares about is how many lines were added, deleted, or modified. As you start doing more interesting things in git those messages will start getting longer and more interesting as well.

Pushing Git

Here's a point about git that probably doesn't matter to you all that much yet, but it's *very important*: Git is incredibly egalitarian. When you start sharing your repository with others git doesn't care who created the "original" repository; once a repo is on someone's machine they can do whatever they like with it. Tools like GitHub and Bitbucket have added authentication and permissions and controls and whatnot to git, but deep down, git sees all copies of a repo as equally valid.

From your point of view, there are two types of repositories: local and remote. You generally have one local repository, the one inside your working directory. But the interesting news is that you can have as many remote repositories as you like, scattered all over the world. There was a brief period of time where I had two instances of a repo on the *same machine*, each of them considering the other one a "remote" repository. Yes, I had a good reason for doing that. No, it definitely didn't count as "painless".

We're not going to push our changes to a remote repository just now, because we'd need to do a fair bit of setup first and this chapter is already getting a bit long. We'll cover that step in the chapter on Remotes. For now, though, we're going to talk about how git differs from everything else out there. Let's talk about branching.

Branching

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Branching: Split Apart

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Merging: Come Together

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Branch Practice

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Learn Git Branching

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Extra Credit

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Commits and Their Parents

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Terminology

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Use Your `HEAD`

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Configuring and Ignoring

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Configuration: Here and Everywhere

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Ignore this text. Fnord.

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Git, Ignore!

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

`.gitignore` is Bliss

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Personalized Ignorance

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Reaching Out: Working with Git Remotes

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Team Git Terminology

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Who Does Git Know?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Two Remotes, No Waiting

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Playing in the Sandbox

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

The Github Option!

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

The Non-Github Option

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Interlude: Oops! I Broke Git!

What Not To Do

Reset or Revert

Git Reset: For Problems That Haven’t Been Committed

Git Revert: For Undoing Public Mistakes

But What About Merges? Can you Revert Those?

Fix a Merge Conflict

Burn It Down. Burn It All Down.

Conflict Resolution

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Artisanal, Hand-Crafted Fixes

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

How Real People Resolve Git Conflicts

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

In Either Case

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Solo Git

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

It's Just Not That Hard

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

It's a Free Backup System

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Asking For Help Just Got Simpler

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

You Can Try New Stuff

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Moving On

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Part II: Refinement

Good Git Habits

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Commit Messages: Say Something Worth Saying

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Writing Commit Messages

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Branch Names

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Small Commits

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

So How Often is “All the Time” In Practice?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Worry About Merges, Not Commits

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Branching Out

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Workflow 0: No Branching

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Workflow 1: Informal Branching

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Workflow 2: Autonomous Feature Branching

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Workflow 3: Formal Feature Branching

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Workflow 4: Git Flow

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Good Git Hygiene

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Delete Old Branches

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Prune Frequently

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Taking out the Trash: Garbage Collection in git

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Why Would I Ever Need Aggressive Cleaning?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Interlude A: Tools

One of the advantages of working with git is that there are a *lot* of tools out there to help you work with git quickly and efficiently. Don't get me wrong, you can profitably do everything you need to do from the command line, and it'll work just great. But you don't need to feel constrained by the command line. Sometimes it's easier to get a view of what git is doing using newfangled *graphics* and even *graphical user interfaces*. I've written two books about command line tools, so you won't find a bigger fan of terminal funtimes than me. But there are times where it's really nice to just right click on the `test` branch and choose "fast-forward" instead of trying to remember the exact grammar of the command to update a branch I'm not on (`git fetch origin test:test` if you're interested).

It's important to know how to do things by yourself, so you aren't dependent on your tools. But think of what a good git GUI represents: a team of developers decided to write an app that comprises all the best practices they have found. As they learn more, they improve that GUI. They are spending all day thinking about the best way to do things in git *so you don't have to!* This requires a level of trust, and a willingness to get over "not invented here", but if you can do that you'll find that your productivity increases.

There are three main types of tools I would like to focus on:

1. Git GUIs
2. Diff/Merge Tools
3. IDEs and Editors [15]
4. The Command Line!

I'm going to provide some features I look for in each class of tool, and a couple of examples of tools you might want to use. But in the end, it doesn't really matter *which* tool you use, what matters is that once you choose a tool you spend the time it takes to get good at it, to be able to take full advantage of it. Once you've chosen a tool, you should commit to stick with it.

What If I Find A Better Tool?

It happens so innocently. You're working along, using the set of tools you have committed to, and you see something; maybe it's an ad on a website, maybe it's a Stack Overflow answer that links to a tool's website. And you're just looking, at least, at first. But…that new tool is shiny, good looking,

[15]The difference between an "Editor" and an "IDE" is rapidly shrinking. It used to be easy to say "Vim is an editor, Visual Studio is an IDE". But now there's Visual Studio Code, which is almost perfectly balanced between the two worlds. It also used to be easy to say "if it just edits text it's an editor, if it also compiles your code its an IDE". But now you've got plugins for all the editors that allow you to run all your compilations and preview the output. Basically I'm including both terms right now so that nobody gets mad at me for calling their favorite editor an IDE and vice versa.

and does all kinds of things that your current tool can't do. What do you do?

There's two sides to this one. You shouldn't feel locked into a tool that is no longer meeting your needs, or that has been eclipsed in functionality by something else. On the other hand, you need to be aware of the costs and benefits of changing tools.

The benefits are usually obvious; that's the thing that makes you want to switch in the first place. Maybe the new GUI makes nicer git graphs or makes it easier to manage branches graphically. But it can be hard to quantify how useful those benefits are. Will prettier graphs make you faster? Perhaps; a picture is worth a thousand words, and a good picture is worth a well chosen thousand words. Also, don't discount your intuition here; a tool that fits your way of working will increase your productivity, because it'll work with you instead of making you adapt to fit it.

But there are costs involved. As you switch to a new tool you will need to spend time figuring it out, configuring its settings, working out how it works with the rest of your tool stack, etc. And you *must* spend this time. Half understanding the tool you use is doing yourself a disservice.

So it looks like we are wasting time, sometimes. We spend this time getting good at one tool, then a while later we abandon it in favor of a new, better tool, and spend time getting good at that new tool. How do we balance this learning time with our need to actually, you know, *get things done*?

My answer is more of a guideline: **At least six months**. If you've committed to a specific GUI or IDE or diff tool, commit to use it for at least six months. If it takes you a month to get up to full capacity then you've got at least five months working at full capacity and a 1:5 ratio isn't terrible. Like I said, this is only a guideline; if I had given up on vim after only six months I would have missed out on it entirely. Some tools have a steeper learning curve than others. And sometimes it's obvious that a tool isn't all you thought it was going to be after only a week. Then maybe you should abandon it early. Just be aware of your behavior, make sure you're doing what you do in favor of greater productivity and not just bouncing between tools because learning a new tool is fun.

Git GUIs: Pretty Commit Trees!

The first tool most new git users seek out is a *Graphical User Interface*[16]. And with good reason. There are a lot of things that you *can* do by hand, but it's so much faster to let a GUI deal with the syntax while you just push buttons.

But how do you know which tools are good and which aren't worth your time? Here's a list of what I look for in a git GUI, followed by a list of a few that I like.

A GUI's job is to make easy things easier. If you need to do some deep forensics in git you probably won't do that in your GUI. On the other hand, most GUIs do a good job of correctly formatting your commit message, showing a brief synopsis of what changed in each commit, and most interestingly, They make it easy to browse the *commit tree*.

[16]If you're reading a book about git you probably already know what a GUI is. But I'm in "educational book" mode, I can't help myself. If I'm not careful I'll probably slip up and accidentally throw in the useless factoid that a "nibble" is four bytes....dang it.

The good news here is that *really excellent* git GUIs are **everywhere**. Most IDEs and programmer-centric editors have at least a decent one built in. But we're going to look at standalone git GUIs.

SourceTree
: SourceTree is made by Atlassian, is cross-platform, and is free. It's very good. When I'm working with new git users I always recommend they download SourceTree immediately. Unsurprisingly, SourceTree integrates very well with Bitbucket, Atlassian's git hosting service.

Tower
: This is the one I actually use. Tower Git is a very refined and powerful git GUI that gets frequent updates and tends to do what I want a GUI to do without me working too hard to configure it. It's not free, and has recently switched to a yearly subscription model. I'm still trying to decide if I want to pay for it yearly or if I'm going to move to a free option.

GitKraken
: Besides the punny name, GitKraken offers a visual approach to git management. Many operations are handled via drag and drop. There's a built-in code editor, a built-in diff/merge tool…this is a very complete package, but also comes at an annual price. If you're interested in staying as far as possible from the command line GitKraken is probably the tool for you.

Diff/Merge Tools: Seeing What Changed, Fixing What Broke

There are two things you're going to want to know about when you're working in git: what changed in a commit? This information is requisite enough that it's on display everywhere. You can use your git server software to look over commits. Your IDE probably has a diff tool built in, your toaster probably does too, if you have a toaster that also does software development. IoT and whatnot.

But a good Diff tool isn't a good Merge tool, although people often confuse the two. One is a lens, the other is a scalpel. The difference is often in the number of panes.

When you're looking over a set of changes in a web UI for a git server it often looks like a set of lines in red, followed by a set of lines in green. What this is telling you is that the file used to have these lines, now it has these lines. Neat, yeah? Except imagine reading a book like that:

> And the wizard said, "Hello! We'll be leaving shortly."
> But in an old version she said "Hey there! We're about to get going." Then they went to the mystical castle in the midst of the Dragonwood, which used to be called the Wood of the Wyrm. The Wizard turned her wise hazel eyes to the frog, and asked "have you been lying to me?" Except the frog used to be a salamander."

That's no way to live. So for a *diff* tool we want something with two panes: one that shows how things used to be, and one that shows how things are now. This way we can follow the logic in the old version, then read the new version and make sense of that as well.

But what about Merge tools? For those we want *three* panes. And this is why diff and merge tools are different. A diff tool shows us what changed, an Merge tool goes a step further and lets us make changes. The most common use case for a merge tool is the dreaded *merge conflict*. We talked about this in the ConflictResolution chapter, but without talking about specific tools.

As I said then, most diff tools are also merge tools. So here's a list of a few good once to check out.

Kaleidoscope

: I can never spell that word. But if you're using Mac OS this is far and away my favorite diff/merge tool. It integrates with most git GUIs, and can be set as your primary diff tool from the command line as well. Kaleidoscope gives you two panes to compare, three to merge, and has all the features I want: free-hand editing of the final center column, "first left then right" style resolution management, and a clean, simple, and elegant UI. It's not free, but I've been using Kaleidoscope for almost eight years since I purchased my license, so I feel like I got my money's worth.
Beyond Compare

: If you want a more platform agnostic diff/merge tool Beyond Compare is my go-to. It has all the basic features, and has the best interface I've ever seen for comparing entire directories of files. Also not free, Beyond Compare is very reasonably priced.
Meld

:If you're looking for a free, cross-platform tool, Meld is your jam. It's a little quirkier than the others, but I have friends who swear by it. The UI is totally serviceable but I don't find it as comfortable or easy as the other two.

The Command Line!

Seriously, get comfortable with git on the command line. That's where all the power is. As you work with git directly you can start to understand how it works, and stop being afraid of it. Every major OS comes with a good *terminal* program; even Windows 10 is getting one this year. They're only 25 years late to the party! But I kid.

If you're on Linux you owe it to yourself to be very comfortable on the command line. And go farther than that. Choose a favorite shell (ZSH is a nice, comfortable alternative to bash, and makes you sound like a total hacker even though it really is almost identical. FISH is interesting, modern, and really pretty cool). Pick a favorite open-source terminal emulator instead of the default one. (I currently like ITerm2 on Mac OS, and Prompt on iOS.)

Stashes: Quick, Hide Your Code!

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

“Yeah, Let Me Take a Look.”

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

“Awwwww, I forgot to Check Out!”

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

How to Actually Use Git Stash

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Refined

Part III: Sophistication

Use Sparingly: Git Commands You Should Use Less

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Amend: When What You Said Isn't What You Meant

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

The Trap!

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Squash: Pretending History Didn't Happen

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

The Trap!

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Rebase: Moving History

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Cherry-Pick: Moving One Commit to a New Branch

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

The Trap!

The Cases for and Against Bubbles

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

The Case for Rebase

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Keeping the Mystery out of Your History

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

The Verdict!

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Delving Into `.git`

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

The Directories

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Interlude: Selective Staging

Hooks

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Why would I want to do that?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Interlude: Commits Revisited

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

First: Files in a Commit

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Second: Git Doesn’t Actually Store Files

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Third: Git Compresses

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Getting the Elephant out of the Repo: Git's Large File System

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Binary vs Text Files

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

So Let's Put Large Files Somewhere else!

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Then Follow The Directions!

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

This Is All Easier if You Start Earlier

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Shhh! We Are Talking About SSH!

Now that you've spent some time with git and git servers, you've probably noticed something. When you clone a repository you have a choice: you can use HTTPS or SSH. Up to this point you've probably only used HTTPS; indeed all my examples have been using that protocol. So why is that other one there? Why do you have this choice? And why would you ever use an SSH connection?

There's not a simple answer to that question. Setting up an HTTPS connection is instant and effortless, whereas setting up an SSH connection requires effort and time. On the other hand, SSH is more secure, although for most people, most of the time, HTTPS is quite secure enough [17]. An SSL connection can transfer files faster but again, HTTPS is quite fast enough.

The point is that HTTPS is *fine*. It's just great. If you read two more paragraphs into this chapter and decide SSH is not for you that's not a problem at all.

But where HTTPS is fine, SSH is *right*. Initially git handled all remote operations over SSH connections; there was no other alternative. I'm gonna lay out a few advantages to SSH over HTTPS, and if you decide you want these in your life you can go on to the second half of the chapter, where we'll talk about configuring git to use SSH.

First off, What is SSH?

SSH is a *networking protocol*, much like HTTPS or FTP. It's a way to communicate between two computers. Developers and operations personnel love SSH because its main purpose is to give you access to the command line on another machine in a secure way. The name "SSH" stands for "Secure Shell"; A "shell" is a program that lets you issue commands to the kernel, or "command line" as we often call it, somewhat inaccurately. If you don't know anything about *nix shells you're probably using bash (the "Bourne Again Shell"), although these days zsh and fish are growing in popularity.

The good news is you don't need to worry about any of that. When we talk about SSH in the context of git, we can just say "SSH is a very secure way to access a remote repository" and leave it at that.

What are the Benefits of Using SSH?

Excellent question! Here goes:

[17] In all honesty someone trying to compromise your repository will probably find their way in by phishing or hacking your password long before they crack the SSL encryption on an HTTPS connection.

SSH is Highly Secure

When you connect to your remote git server via HTTPS, it's almost exactly like connecting to your bank's website. Your computer and the remote computer create a secure connection using *cryptography*. Then your computer sends your username and password (also cryptographically secured) to the remote computer. The remote computer says "yep, that's someone I know, and that's the right password! You can do git stuff now!" And then it does the git stuff you told it to do.

But there are still some weaknesses in HTTPS security. When you connect to a site via HTTPS the site sends you a certificate that is used to make security magic happen. The problem is that you have to trust that the certificate came from the site you want to visit, and not some scary third party that wants to harvest your cookies and turn your computer into a botnet drone of evil. To get around this HTTPS certificates have to be *signed* by someone who we can all mutually trust. To get your certificate signed by one of the big companies can cost thousands of dollars and take days and days of lawyers and signatures and stuff. So a lot of smaller certificate signers have popped up…and the circle begins again. HTTPS is great, if everyone is who they say they are and everyone trusts everyone to tell the truth about who they trust.

When you connect to your remote git server via SSH, your username and password are *not* sent as part of the connection. Instead you generate a *key pair*, which we will discuss in a moment. You share one of the keys with the remote server in some way, which, again, we will discuss in a moment. And you keep the other key secret on your local machine. Using these two keys, SSH can verify that you are who you say you are using *magic* [18]. If someone intercepts the connection they don't get *either* of your keys, they get a nonsense token that is useless without your keys. You don't need any third party to help you verify who anyone is; your key pair is the verification.

SSH is Fast

SSH was created to let you work on a remote machine as if you were physically sitting in front of it. File transfers happen using a (comparatively) fast protocol compared to HTTPS. Instead of opening a new network connection for each file, SSH operates over a single connection, held open for as long as required. Mind you, git is being developed by very intelligent people, so the HTTPS connection method is getting better at maximizing the capabilities of the HTTPS protocol, but you're still going to get a little more speed out of SSH.

Okay, Good Enough! How do I use it?

Here are the steps required to set up SSH:

1. Generate your RSA key pair
2. Install your *public* key on the remote server

[18]Okay it's math, not magic. It's very fancy and confusing and **not** painless you can look it up if you like that sort of thing.

3. Configure local git to use SSH
4. Change your repos to use SSH instead of HTTPS

Easy right?

Well…kinda? We'll go through this process together. The problem here is that the steps are *slightly* different depending on your OS of choice. So that's always fun. Okay, let's make some key pairs!

Generating a Key Pair!

Okay, here we're in the clear…mostly. Linux and Mac OS both ship with the default `keygen` tool that makes this easy. Windows doesn't ship with it, but you can use it via Git for Windows, the amazing package I told you about clear back in the Installing Git chapter. So: if you're using Mac OS or Linux, open a terminal window. If you're using Windows, right click in Explorer and click "git bash here", which has the effect of, well, opening a terminal window.

There. Now our steps are mostly the same.

In that terminal window type the following:

```
ssh-keygen -C you@email.com
```

Only, you know, with your email address.

First it will ask if you want to name this key, or use the default of `id_rsa`. You can name it whatever you like, or just stick with `id_rsa`. But what if if you're going to connect to more than one account on a single service? If you have multiple Bitbucket accounts you're going to need multiple keys, so Bitbucket knows which account you're signing into. So on my laptop I have the following keys:

- `laptop-work-bitbucket`
- `laptop-writing-bitbucket`
- `GitHub-personal`
- `GitHub-work`

Full Path Required

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Installing the Public Key on your Favorite Git Server

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Local SSH Configuration

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Added Awesome

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Extra Awesome Non-Git Use of SSH

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

But We Can Make It Even Easier!

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Who Am Us, Anyway?

Play It Again, Git: Using Git Rerere to Stop Repeating Yourself

It's Ref-log, not Re-flog

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Show

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Expire

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Delete

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

And That's Pretty Much It!

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

The End!

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Appendices

Appendix B: Glossary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Appendix C: Terminal Velocity: Getting Up to Speed on the Command Line

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Tell Me What to Do: The Command

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Tell Me How to Do It: The Flags

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Tell Me What to Do It to: The Arguments

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Putting it All Together

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

How do You Remember All This???

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Commands and Command Suites

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Unix-y Stuff

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Executables don’t have extensions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Hidden Configuration Files

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Appendix D: Commit or Commit-ish

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Specific References

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Relative References

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Appendix E: SubGit

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Step 0: Bookmark the SubGit Documentation Page

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Step 1: Gather Information

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Credentials: How to get to Subversion And Who To Be

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Authors: Who Has Worked in Your Subversion Repository?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Branches to Branches: What special instructions do you need to give SubGit?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Initial Config

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

Test! And Test Again!

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/painless_git.

About This Book

About the Text

Painless Git is by and © 2018-2020 Nathanial Ellsworth Dickson.

But you can call me Nate.

It was written using an unwieldy combination of (in no particular order):

- Marked
- monodraw
- MultiMarkdown Composer
- MindNode
- Bear
- Scrivener
- vim (of course)
- TextSoap

It was written on a MacBook Pro.

The Painless Git Repository is on Bitbucket.

About the Cover

The cover features an illustration by Retro Clip Art. Used with permission.
The typefaces used on the cover are:

- **Matchbook Serif** (*title*)
- **Aleo Light** (*subtitle*)
- **Caviar Dreams** (*by line*)

The cover was designed by Nate Dickson using Adobe Illustrator and Pixelmator

Special Thanks

I would like to thank, as always, my dear Libbie, for her support, kind words, encouragement, proofreading, spell checking, questions, and love. I'm the luckiest husband ever.