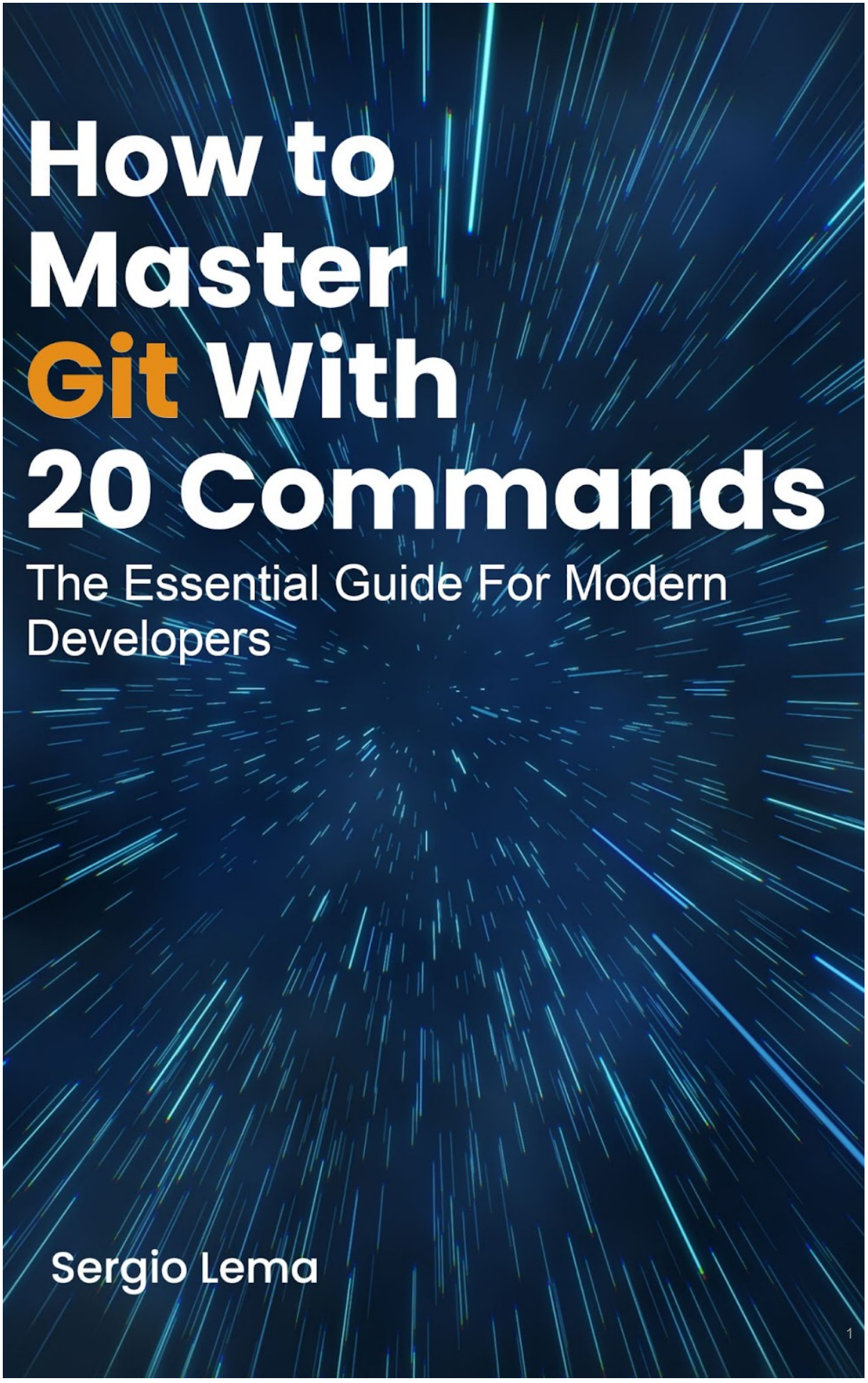
How to
Master
Git With
20 Commands
The Essential Guide For Modern
Developers
Sergio Lema

To my brother Cristian

Table of Contents

Copyright

How to Master Git With 20 Commands
The Essential Guide for Modern Developers
by Sergio Lema

Edition: v1.1 (2023-04)

The information presented here is for informational and educational purposes only. It does not construe any kind of advice. You assume the sole responsibility of relying on this information at your own risk. No liability is assumed for losses or damages due to the information provided. You are responsible for your own choices, actions, and results.

1 - Introduction

I already have a lot of problems with the project and I don't want more with Git.

Git is here to help with my day-to-day.

Git has a lot of commands and options. Still, I only use about 20.

Mastering those 20 commands makes my project life a lot easier.

In the first week, when I was hired as a senior engineer, they asked me to develop a single feature. As always, I did it in a few separate commits, in my separate branch.

When creating the Merge Request, they asked me to rebase and squash all my commits into a single one.

Google helped me to find out what a rebase and squash was… I had a simple experience with Git. Creating some branches, commits and merges. The easiest path.

After performing the rebase, I tried to push but Git warned me with an error message: my current repository is not up to date with the remote repository. And that I should pull from the remote repository.

Long story short, 3 hours later, I ended up with a Merge Request with 40 commits, asking my boss how to do so. Ashamed that a senior engineer wasn't able to run a rebase.

Instead of being angry, he teaches me. Here started my comprehension of Git.

Not everybody has the chance to have a boss who teaches you kindly some core concepts of Git. From here, I've been more and more comfortable with Git. Being able to teach newcomers the main concepts about Git. Deliver training courses about Git in many companies.

I don't want this book to be so rich to cover all the concepts of Git. I want this book to handle the least to be comfortable. I want this book to handle the least to not be afraid in the day-to-day. I want this book to cover the main problems I've also faced in my career.

And all of this can be achieved with about 20 commands.

- Guidelines to create a good commit message
- Guidelines to create a clean Git history
- What's the difference between merge and rebase
- When to use a merge and when to use a rebase
- Guidelines to manage many branches assigned to many environments
- A global configuration to share with all the team members

What's Git?

Git was created to manage a set of files. Git will store only the history of all the modifications made from file to file. It won't store the complete project per modification.

Git was first built to allow a team to share their project modifications as easy as possible.

Linus Torvalds, the creator of Linux, created Git around 2005, when the actual Version Control System (VCS) they used to store the Linux kernel was no longer a solution for their needs. So, the team developed their own VCS. Since then, Git has become one of the biggest VCSs used around the world, with a community growing every year.

Audience

The audience for this book is mid-level developers who already know the basics of Git and/or versioning tools, but want to improve their knowledge by better understanding how Git works.

Book Organization

This book aims to provide knowledge on how Git behaves. Letting the user understand how each of the listed commands works and being more comfortable with Git.

As there isn't a single way to perform an action, I don't want to show the best way to perform each action. Instead, I will show how to navigate in the Git history and some basic commands more than enough to perform every action needed.

The following book uses Git version 2.33. It doesn't cover the Git installation. And it uses the terminal to run all the commands.

The book has the following parts:

- In the first part, Zones, I cover how a file is managed by Git;
- In the Commits part, I describe how to create, edit and undo commits;
- In the Branching part, I describe what are the branches and how to interact with them;
- In the Search part, I describe how to look for a specific change or commit;
- In the Remote Repository part, I handle the interaction with other developers through a remote repository;
- In the Advanced part, I briefly describe some important concepts about Git and its environment.

Convention Used

`git status` This is a command that can be run in a terminal.

feat This is the name of a branch used in a Git command or a special file.

💡 Tip 💡 This is useful information that helped me to understand a complicated concept or to perform a complicated action easily.

🚨 Warning 🚨 This is a warning message about an action that has some risk, such as losing data.

Contact

You can address any comments and questions concerning the book to the author at book@sergiolema.dev.

2 - Zones

Git identifies a file in different zones depending on my actions. I first must know the definition of those zones, why my file is in a specific zone and how I can move a file from one zone to another.

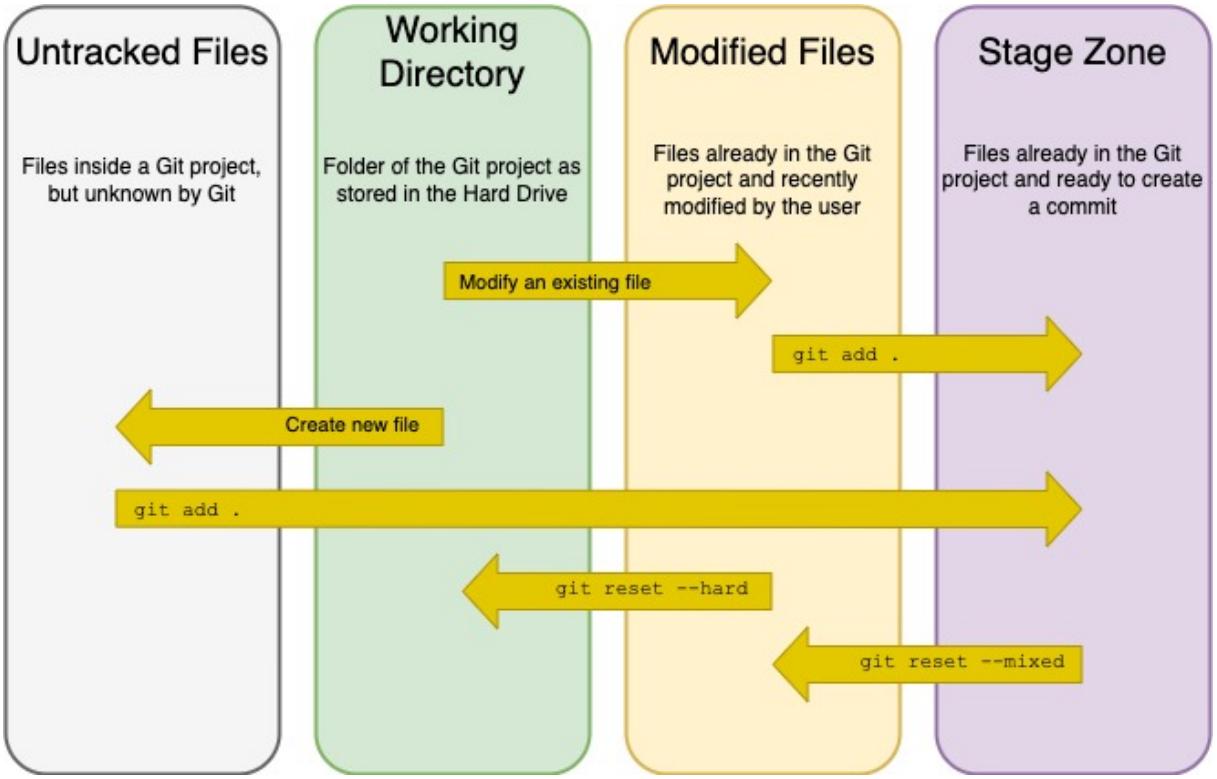

Figure 2-1: Different Zones of Git

Working Directory

A Git project is always stored in a folder, in a root folder. The working directory is the actual state of the hard drive for the folder of my Git project.

When I switch to a branch, my working directory will change. When modifying a file, I'm modifying the working directory.

Stage

Once I've done all the modifications I want in my working directory, I must add the modifications to the stage. Those modifications will compose the further commit.

Untracked Files

When I modify existing files in my project, Git will see the modifications. Then I can add them to the stage if I want.

But if those are new files, Git won't know what to do with them. Track them or ignore them? Those files will remain as untracked files until I tell Git what to do.

I have two options:

- I can add them to the stage, telling Git to track them;

- or I can also add them to the *gitignore* file to ignore them. I will explain later more about the *gitignore* file.

💡 Tip 💡

Status

A file can be in many zones. With `git status`, I can see all the zones and which files are on each zone.

From there, `git status` shows me which command to use to move a file to another zone: to the stage, ignore it, or reset the modifications.

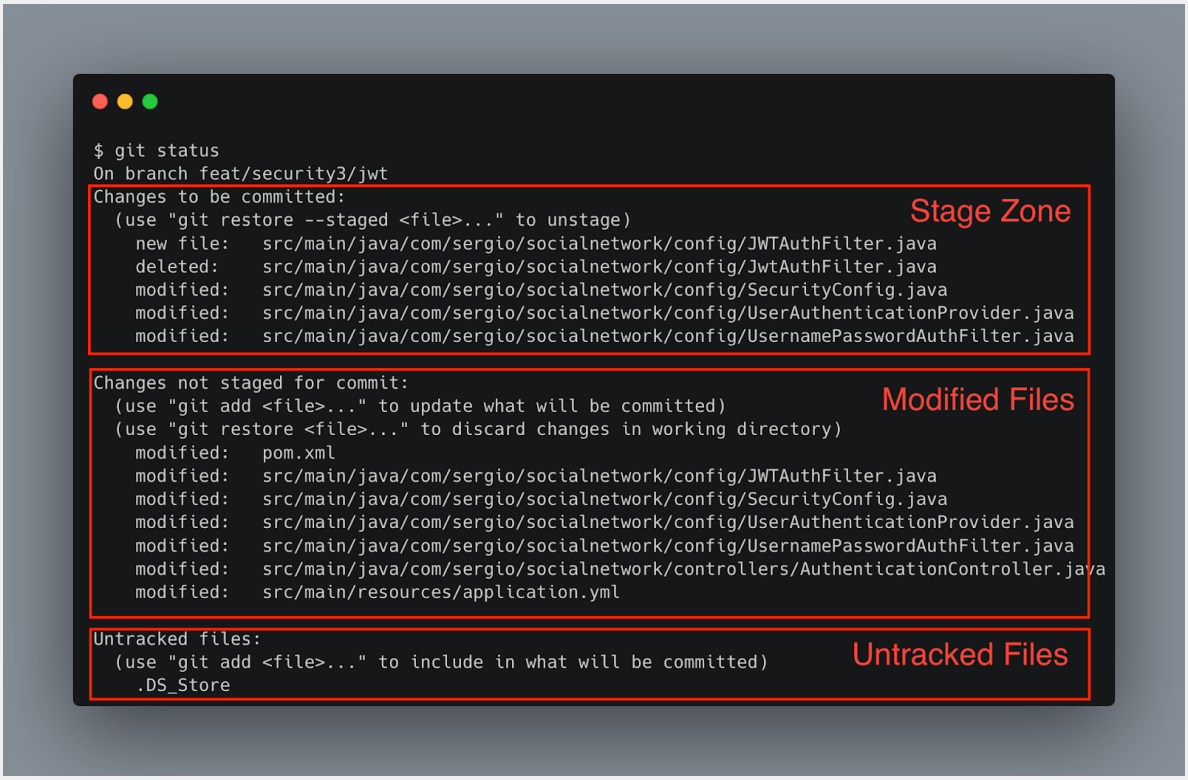

Figure 2-2: Status command displaying the files in the different zones