Essential Git

Jorge Escobar

fromzerø_

Essential Git

Why spend hours learning all Git and Github? Learn the parts professional developers use and get on with your coding

Jorge Escobar

This book is available at http://leanpub.com/essential-git

This version was published on 2024-10-21

Contents

Introduction

Welcome to "Essential Git - Third Edition". My name is Jorge Escobar and I have been managing development teams that use Git on a daily basis for many years and I know first hand how important it is to be fluent in this fundamental tool.

I also happen to have started my online instructor career teaching this course. And after thousands of students enrolled, I have decided to make this second edition that improves and expands the material on this wonderful piece of software.

Git has a lot of quirks and it does have a wide range of features, but to be honest, most professional developers only use a subset of the tool in their day to day job. So the goal of this course is that you understand from the very beginning and without any previous experience the essential parts of Git that you will use managing software development in any professional tech company.

We will begin our journey by introducing the concept of version control and why it's important; we will then install Git, go over the File Status Lifecycle, get comfortable through hands-on exercises with our first project and finally learn all about branching and remote repositories. At the end of the course we will look at some of the different workflows used today and finally we will all work together in a real worldwide open source project called "The Global Restaurant Guide".

So let's start learning the "Essential" Git!

What is Git?

Git is a version control system, or VCS, that allows developers to improve the way they work on their code as well as collaborate

easier with other developers in software development projects. Since its inception in 2005, Git has become an essential tool for anyone wanting to work in the computer software industry.

What is Version Control?

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later. Every time the developer finishes his work and gets to a new version, he notifies or "commits" this version to the Version Control software, and each version is kept within the system. Additionally, other developers working on this project can get notified and they incorporate the changes made by other developers into their own codebase.

Here are some advantages to using version control:

Serves as a backup method

Every time you make a change to a file, a new version is saved, but the previous version is kept, so that you can go back to that or any previous versions if, for example, you have an error you want to undo. Additionally, it will transmit these changes to a remote server on another computer or to a service like Github, which means that if something happened to your computer, your work will be saved elsewhere.

Allows a historic revision of the project

With version control you can easily see what the project looked like at any specific time and how it organically grows. You can also see things like how each developer has contributed to the project and what decisions have been made in the past.

Enables collaboration between project members

Version control systems make it easy for the changes being made to the project to be disseminated across the team without having to have everyone in sync or announcing new changes. This makes version control specially well suited for geographically distributed teams to work seamlessly.

Types of version control systems

Historically, there have been three main types of version control systems: - Local (LVCS) - Centralized (CVCS) - Distributed (DVCS)

In *local* version control systems (rcs[1], is an example) a computer has all the files and all the versions of those files locally. Of course a big problem with this setup is that you can't share your work easily and if your computer or hard drive fails, you lose all the work.

[1] https://www.gnu.org/software/rcs/

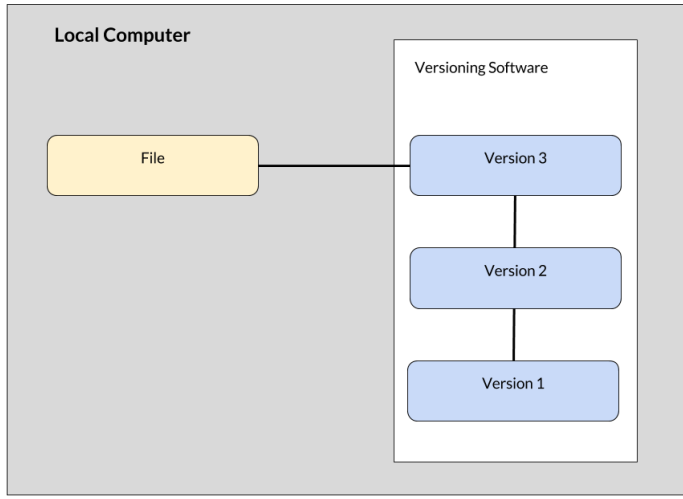

Figure 1.1

In a *centralized* version control system (CVS[2], Subversion[3]) there is a central server that holds all the files and its versions and then computers connect to this centralized server and commit the changes to their files to this server. The problem is that if that server fails or there is a network connectivity problem between the computers and the central version control server, you are basically stuck and can't do any commits or receive any new changes until the servers comes back online.

[2] http://www.nongnu.org/cvs/
[3] https://subversion.apache.org/

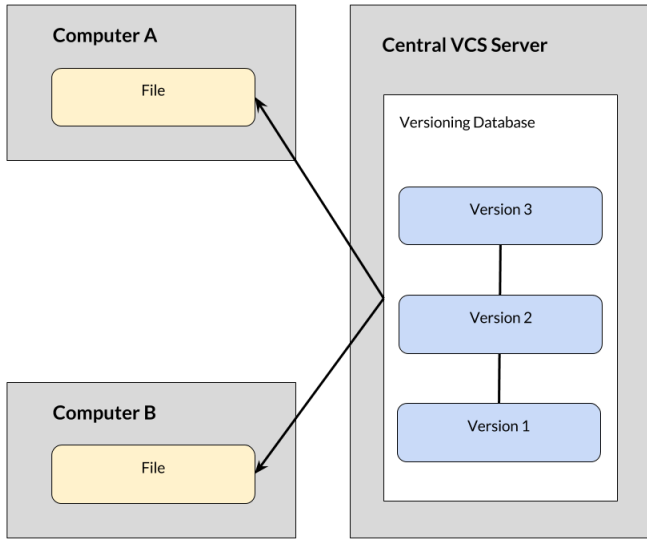

Figure 1.2

In a *distributed* version control system (Git[4], Mercurial[5]) each computer has a full history of all the versions of the files and each person that works in this project has a local list of changes, so there's no issues with backing up or with connectivity. If you're not connected to the internet, you can still work. Additionally you can have a server that works as the main repository, and all computers can connect to this server, but they also can connect with each other.

[4] https://git-scm.com/
[5] https://www.mercurial-scm.org/

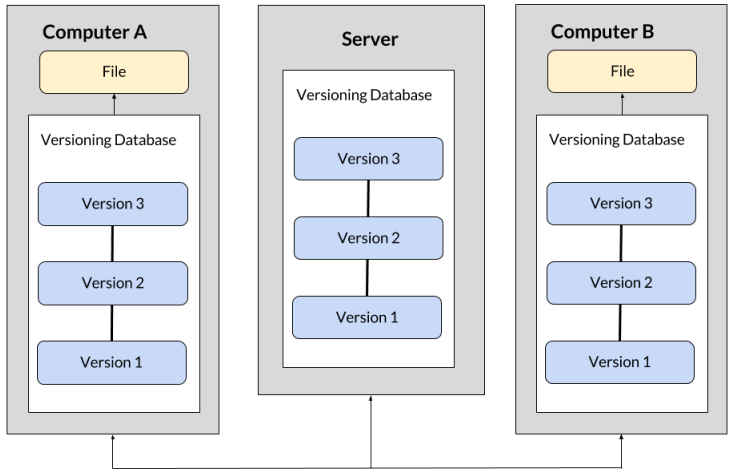

Figure 1.3

Is Git better than other similar tools?

After working with many types of version control systems through the years, there are many reasons I believe Git is better than other version control systems, but here's a few key ones:

- Git follows the distributed model, which as we saw earlier is the best model for a VCS, since we don't need an internet connection to continue working with it or even have a centralized server available.
- Git offers better control over history, thanks to its ability to change commits that happened in the past, and we'll see in the course how to do that.

- Git manages branching better than other distributed version control systems. Branches are an important part of the development workflow and we will be looking at branches in this course.
- Git allows you to temporarily save your work using Git stash. This is a timesaver when you want to just store a snapshot of your work to move to other branches or pull changes to work on. Then you can return to your work easily.