COMP1531

👯‍♀️ Coding Together - Git - Solo Usage

Lecture 1.2

Author(s): Hayden Smith

(Download as PDF)

🤔 The Problem

To effectively work on large projects in groups of engineers we need a more complex method of managing our code, that incorporates:

- **Version control**: Tracks changes to our code over time in a detailed and systematic way (like a logbook and/or time machine)
- **Concurrent programming**: Effectively allows multiple people to work on the same files or series of files and seamlessly integrate changes together (like google docs but for code).

Programs like "Dropbox" or "Onedrive" maintain a history of files and allow syncing between multiple sources. Other tools like Google Docs allow for version control and collaboration. However, they are too simple for our needs or not specific enough to programming.

A Popular Solution: Git

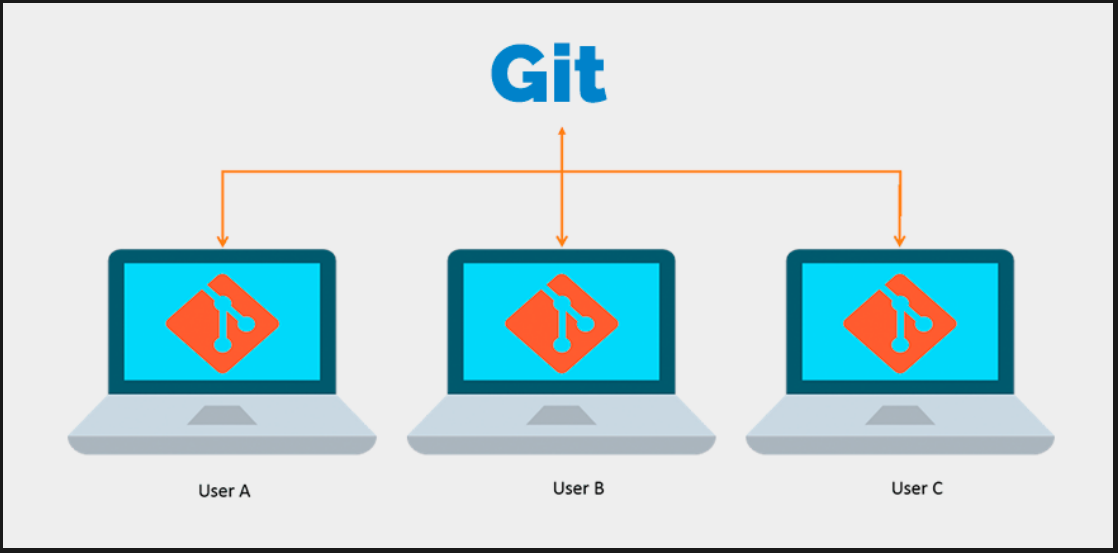
Git is a version control tool that enables people to work concurrently on the same codebase. Git is a **program** just like **gedit**, **vscode**, **gcc**, etc.

Git is built for programmers and designed for managing code across lots of people with a detailed history.

There are other solutions to the same problem, but git continues to be popular in both industry and open source work.

💡 A Popular Solution: Git

Git is a distributed version control software. Whilst many users share work via a central cloud, each user has a full copy of the work and therefore each user has a full backup of the work.



source

 A Popular Solution: Git

Git is just a command line program. However, due to it's popularity web services came to lift that not old act as the central server for the code, but also offer a GUI to help manage some of git's functionality.

There are 3 major git software tools that implement the git language via an easy-to-use web app.

They all do basically the same thing. Just like how Chrome, Safari, Firefox are different ways to interacting with the exact same internet. In this course we will be using Gitlab.

Learning Git

We're going to learn git in 3 key stages:

1. Version control on a single machine (today)
2. Version control across multiple of your machines (today)
3. Version control across a team of engineers (next lecture)

These will be practical demos. If you want to follow a written guide, then please checkout Atlassian's git guide.

Learning Git

Single Machine

Stage 1. Version control on a single machine

getting setup	status of work	doing work
• SSH Keys • `git clone`	• `git status` • `git log`	• `git add` • `git diff` • `git commit` • `git push`

Learning Git

Multiple Of Your Machines

Stage 2. Version control across multiple of your machines

multiple machines

- `git pull`
- merge conflicts

🥳 Git Commands Summary

The following commands are what we learn before we worry about "branches" next.

Command	Description	Example
git clone	Clones from a cloud repository to a local repository	`git clone`
git status	Tells you information about the "state" of your repo	`git status`
git log	Gives you a commit history of commits made	`git log`
git add	Adds a particular untracked file to your repo ready for commit, or stages a tracked file ready for commit	`git add --all` `git add file.py`
git diff	Shows the difference between the last commit and the work you've done since then	`git diff`
git commit	Commits changes ("takes a snapshot") of your work	`git commit -m "Message name"`
git push	Syncs the commit history locally with the commit history on the cloud	`git push` `git push origin master`
git pull	Syncs the commit history on the cloud with the commit history locally	`git pull` `git pull origin master`

👂 Feedback

Or go to the form here.