What is Git?

Version Control:

A method of keeping track of all the different versions of files as time progresses

Some other types of version control track what is added or removed between snapshots, git captures snapshots of the entire project, not just what’s different.

Allows you to keep multiple versions of the same project at the same time.

Why use Git?

Keeps a complete backup of your project in all different versions.

Easily share projects with multiple people and control what gets overwritten, no “tug of war”

Easily try something out, revert the changes if you don’t like it

Great for open source, modify projects you enjoy

No “right” or “wrong” way to use it, lots of ways to effectively use it, allows projects to be more modular

Usable for all types of files, graphic designer keeping track of all the different iterations of their work, author trying out different ideas for a story

It’s used by a lot of people in various computer science industries, great to be familiar with, great resume material

History

The Linux kernel maintenance distributed patches and archived files from 1991-2002

2002-2005 the Linux developers used a proprietary distributed version control system (BitKeeper)

2005 Git was developed

Components

Working directory: where everything you are currently working on is, the current “snapshot” of the project

Staging area: files ready to be backed up are placed here. When you’ve done some work you might add the files to the staging area in preparation of your project

Repository: the history of your project, contains all snapshots and how they are linked to one another, “timeline”

Basics

Create a repository in the directory with the files you want to store.

Add the files to the staging area of the repository.

Commit the changes to your repository (saving them into the repository)

Branch if you want to, sometimes it's beneficial to have multiple branches in order to keep things separate

Modify files, add them to the staging area, and commit the changes. Repeat this process as necessary.

You may wish to merge branches back into the master branch

Core Concepts

Repository: the history of the project, all snapshots in it

Master branch: all repositories have one, the initial branch from which all others stem

Commit: a stored snapshot of your project, could include additions or deletions or entirely new files

Head: where you are currently in the history, points to the commit the current working directory is based on

Remote Repositories

Adds the benefit of having an offsite backup.

"Hub" where many users can make contributions to a project.

From an open source perspective it's really cool because anyone can make changes to programs they like and potentially have them placed in the main branch

Lots of options when choosing which to use, most provide free public repositories with an option to pay for private ones

Git Flow

"layers" of branch, the top most layer would be what most users see of your application

Contains a development branch where most of the work takes place, pushed to release branch when it's ready for general consumption

GitHub Flow

master branch is the main branch. Features stem from and merge into the master branch

Common Commands

git init	Initialize a new git repository in the current location
git add <filename>	Add the file to the staging area
git commit -m “commit message”	Commit the files in the staging area
git status	See what files are staged to be merged
git checkout –b <branchname>	Create a new branch with a name
git merge <branchname>	Merge a branch into the existing branch
git checkout <branchname>	Switch to a different branch
git pull	Get commits from a remote repository
git push origin <branchname>	Send commits to a remote repository

There are also of ways of using Git with a GUI if you don’t want to use the command line.