LAB 2 – Version control, Git and Github

Version control

Version control - also known as source control or revision control - is an important software development practice for tracking and managing changes made to code and other files. It is closely related to source code management.

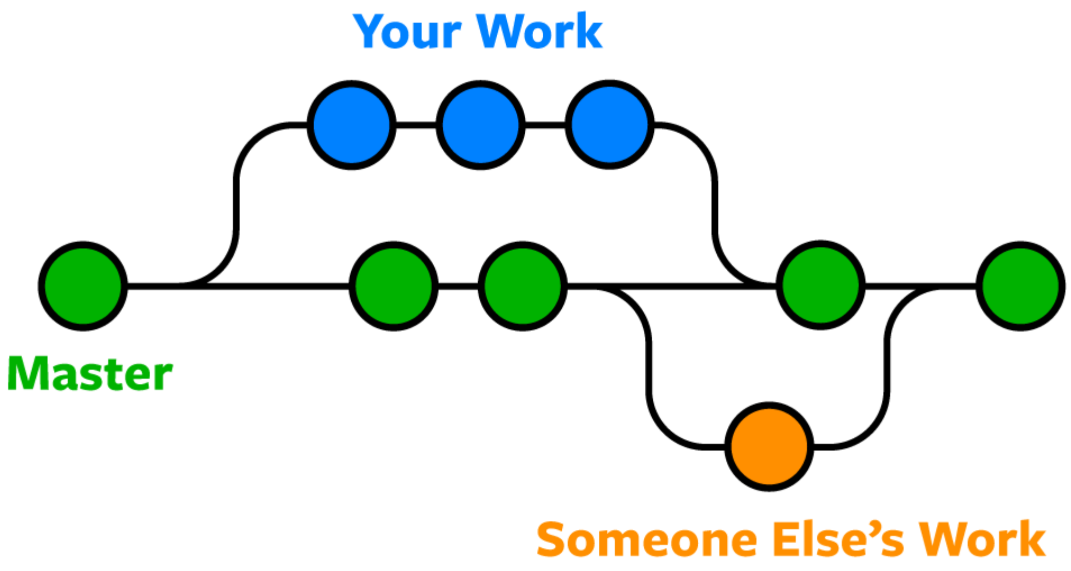
With version control, every change made to the code base is tracked. This allows software developers to see the entire history of who changed what at any given time — and roll back from the current version to an earlier version if they need to. It also creates a single source of truth.

Version control (or source control or revision control) serves as a safety net to protect the source code from irreparable harm, giving the development team the freedom to experiment without fear of causing damage or creating code conflicts.

If developers code concurrently and create incompatible changes, version control identifies the problem areas so that team members can quickly revert changes to a previous version, compare changes, or identify who committed the problem code through the revision history. With a version control system (VCS), a software team can solve an issue before progressing further into a project. Through code reviews, software teams can analyze earlier versions to understand the changes made to the code over time.

What is Git?

Git is the most commonly used version control system. Git tracks the changes you make to files, so you have a record of what has been done, and you can revert to specific versions should you ever need to. Git also makes collaboration easier, allowing changes by multiple people to all be merged into one source.



Git is software that runs locally. Your files and their history are stored on your computer. You can also use online hosts (such as GitHub or Bitbucket) to store a copy of the files and their revision history. Having a centrally located place where you can upload your changes and download changes from others, enable you to collaborate more easily with other developers. Git can automatically merge the changes, so two people can even work on different parts of the same file and later merge those changes without loosing each other's work!

Git Repositories

A Git **repository** (or **repo** for short) contains all of the project files and the entire revision history. You'll take an ordinary folder of files (such as a website's root folder), and tell Git to make it a repository. This creates a **.git** subfolder, which contains all of the Git metadata for tracking changes.

On Unix-based operating systems such as macOS, files and folders that start with a period (.) are hidden, so you will not see the **.git** folder in the macOS Finder unless you show hidden files, but it's there! You might be able to see it in some code editors.

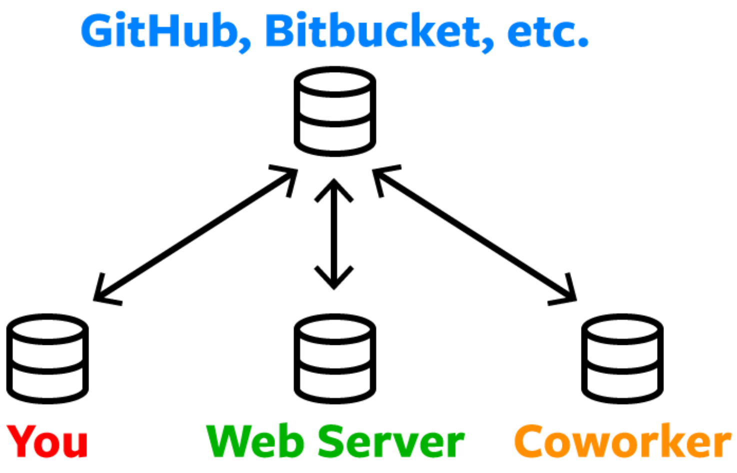
Stage & Commit Files

Think of Git as keeping a list of changes to files. So how do we tell Git to record our changes? Each recorded change to a file or set of files is called a commit.

Before we make a commit, we must tell Git what files we want to commit. This is called staging and uses the **add** command. Why must we do this? Why can't we just commit the file directly? Let's say you're working on a two files, but only one of them is ready to commit. You don't want to be forced to commit both files, just the one that's ready. That's where Git's **add** command comes in. We add files to a staging area, and then we commit the files that have been staged.

Remote Repositories (on GitHub & Bitbucket)

Storing a copy of your Git repo with an online host (such as GitHub or Bitbucket) gives you a centrally located place where you can upload your changes and download changes from others, letting you collaborate more easily with other developers. After you have a remote repository set up, you upload (push) your files and revision history to it. After someone else makes changes to a remote repo, you can download (pull) their changes into your local repo.



Branches & Merging

Git lets you branch out from the original code base. This lets you more easily work with other developers, and gives you a lot of flexibility in your workflow.

Here's an example of how Git branches are useful. Let's say you need to work on a new feature for a website. You create a new branch and start working. You haven't finished your new feature, but you get a request to make a rush change that needs to go live on the site today. You switch back to the master branch, make the change, and push it live. Then you can switch back to your new feature branch and finish your work. When you're done, you merge the new feature branch into the master branch and both the new feature and rush change are kept!

When you merge two branches (or merge a local and remote branch) you can sometimes get a conflict. For example, you and another developer unknowingly both work on the same part of a file. The other developer pushes their changes to the remote repo. When you then pull them to your local repo you'll get a merge conflict. Luckily Git has a way to handle conflicts, so you can see both sets of changes and decide which you want to keep.

Pull Requests

Pull requests are a way to discuss changes before merging them into your codebase. Let's say you're managing a project. A developer makes changes on a new branch and would like to merge that branch into the master. They can create a pull request to notify you to review their code. You can discuss the changes, and decide if you want to merge it or not

How to install Git

In order to use Git, you have to install it on your computer. To do this, you can download the latest version on the official website. You can download for your operating system from the options given.

Exercise

Please go over the entire tutorial

https://www.w3schools.com/git/

Using VS Code Integrated Source Control Management

Visual Studio Code has integrated source control management (SCM) and includes Git support out-of-the-box.

1. Create a new folder, open VS code and initialize git

https://code.visualstudio.com/docs/sourcecontrol/overview