Git/Github

* Git is the most popular version control system(VCS). A version control system records changes made to our code overtime in a special database called Repository. It allows us to look at our project history, who has made what changes, when and why and also revert changes to an earlier state.

Version control can either be centralized or Distributed:
* In a Centralized Version Control Systems (CVCS) all team members connect to a central server to get the latest copy of the code and to share the changes with others. In centralized systems, there is a single central repository where all changes are stored, and developers must connect to it to commit or retrieve changes. Examples - Subversion(SVN), Concurrent Versions System(CVS), Perforce Helix Core(a hybrid model),Team Foundation Version Control (TFVC) and many more.

*In Distributed Version Control Systems (DVCS), each developer has a complete copy of the repository, enabling local commits and offline work.Examples - Mercurial, Bazaar, Fossil and many more

What is Git?
Git is a powerful tool used to manage code changes and collaborate on projects. It’s a Version Control System (VCS) that keeps track of every change made to your files and lets you roll back to earlier versions if needed.

Why Use Git?

- Version Control: Keeps track of all changes made to your code.

- Collaboration: Multiple people can work on the same project without overwriting each other’s work.

- Backup: Ensures your code is safe and recoverable.

- Branching: Experiment with new features without affecting the main code.

What Git Does

1. Tracks Changes: Git keeps a history of all changes you make to files in your project, allowing you to see who changed what and when.

2. Enables Versioning: You can save "snapshots" of your project at different points in time, called commits. This way, you can go back to an earlier version if needed.

3. Supports Collaboration: Git allows multiple people to work on a project at the same time without overwriting each other’s work, making it ideal for team projects.

4. Works Offline: You don’t need an internet connection to work with Git, so you can track changes locally on your computer.

BASIC CONCEPTS IN GIT

• Repository (Repo): This is a storage area for your project and its version history. You initialize a Git repository to start tracking changes.

• Commit: A commit is like a "save point" in your project. It records the current state of your files so you can revert to this point later if needed.

• Branch: A branch is like a separate version of your project where you can experiment with changes without affecting the main project. The default branch is usually called main or master.

• Merge: Merging combines changes from one branch into another, usually bringing new updates into the main branch.

• Remote Repository: You can also push your local Git repository to an online platform like GitHub or GitLab, making it accessible for others to collaborate or view.

TO DOWNLOAD GIT
- Go to https://git-scm.com/ or search download git (your operating system) and download for your operating system.

**Installing git on windows comes with git bash which is encouraged instead of cmd.

TO CONFIRM GIT INSTALLATION RUN:
- git --version

CONFIGURING GIT
- Name
- Email
- default editor
- Line Ending

NAME AND EMAIL
Type the following command on git bash.
1. git config --global user.name "Your Name"
2. git config --global user.email "test@w3schools.com"

DEFAULT EDITOR
- git config --global core.editor "code --wait" (with the wait we are telling the vs code to wait till we close the vs code instance)

TO OPEN DEFAULT EDITOR TO EDIT ALL OUR SETTINGS
- git config --global -e

NB: To confirm that Git is now set to use VS Code as the editor, run:

- git config --global --get core.editor

the output should display:

- code --wait

LINE ENDING
In Git, setting end-of-line (EOL) with core.autocrlf is a way to manage the differences in how operating systems handle newlines in text files. Here’s a breakdown of what this setting does and what the terms mean:
CRLF and LF: Newline Characters

LF (Line Feed): Represented as \n in Unix-based systems (Linux, macOS). This is a single character used to indicate the end of a line.
CRLF (Carriage Return + Line Feed): Represented as \r\n in Windows systems. It’s a two-character sequence used to mark the end of a line.

Different operating systems use different conventions for newlines, which can cause issues when sharing code across platforms.

 -TO HANDLE CRLF
- git config --global core.autocrlf input (for mac and linux)
- git config --global core.autocrlf true (for windows)

 VS code extensions for git
 - gitLens
 - git graph
 - git history
 - git blame

TO GET STARTED
**CHEAT SHEET FOR COMMANDS

1.
initialize a repo in the folder - git init

TO SEE GIT HIDDEN FILE
go to views, options, change folder search options, check hidden files. uncheck hide extension for known file types

2.
- git status - show modified files in working directory, staged for your next commit

 Files in your Git repository folder can be in one of 2 states:
Tracked - files that Git knows about and are added to the repository
Untracked - files that are in your working directory, but not added to the repository

 When you first add files to an empty repository, they are all untracked. To get Git to track them, you need to stage them, or add them to the staging environment.

3.
Staging in Git refers to the process of preparing your changes (like modified or new files) to be included in the next commit. It allows you to choose which changes go into your commit.

The staging area is a space in Git where changes are added before committing. Think of it as a "holding area" where you review and organize your changes before they are officially saved in the project’s history.

TO STAGE A SINGLE FILE
- git add fileName

TO STAGE ALL THE FILES IN WORKING DIRECTORY
- git add .

to check status - git status

4.
to unstage a file
- git restore --staged fileName / older syntax - git reset fileName
- git restore --staged . / git reset . (to unstage all the files in the staging area)

5.
After staging, one needs to commit said work. In Git, a commit is like a snapshot of your project at a specific point in time. It records the changes you’ve made and saves them to the repository with a message describing those

changes.

What Does a Commit Do?

- Captures Changes: A commit saves the changes you’ve staged in the staging area (using git add).

- Creates a New Version: Each commit represents a new version of your project, allowing you to track its history over time.

- Provides a Message: When making a commit, you provide a message that describes the changes you made.

How Git Commit Works:

Staging Changes: Before committing, you need to add changes to the staging area using git add.
Making a Commit: Once your changes are staged, you create a commit to permanently save them with a message that describes what you did.

GIT COMMIT COMMAND
- `git commit -m "Your commit message"`
The -m option allows you to provide a commit message directly in the command.
The commit message should briefly describe what changes were made and why.

6.
Git History: After committing, you can see your changes in the commit history
- `git log`

7.
Amending a Commit: If you forgot to stage a file or need to change the commit message
- `git commit --amend`