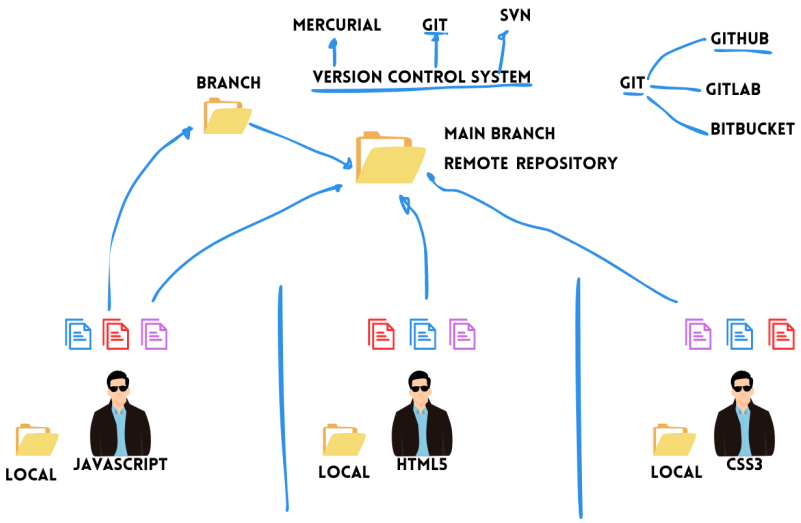
DAY 3 Learning GIT

Git



Git is a distributed version control system (VCS) designed to help software developers track changes in their source code and collaborate on projects. It was created by Linus Torvalds in 2005 and has since become one of the most widely used version control systems in the software development industry.

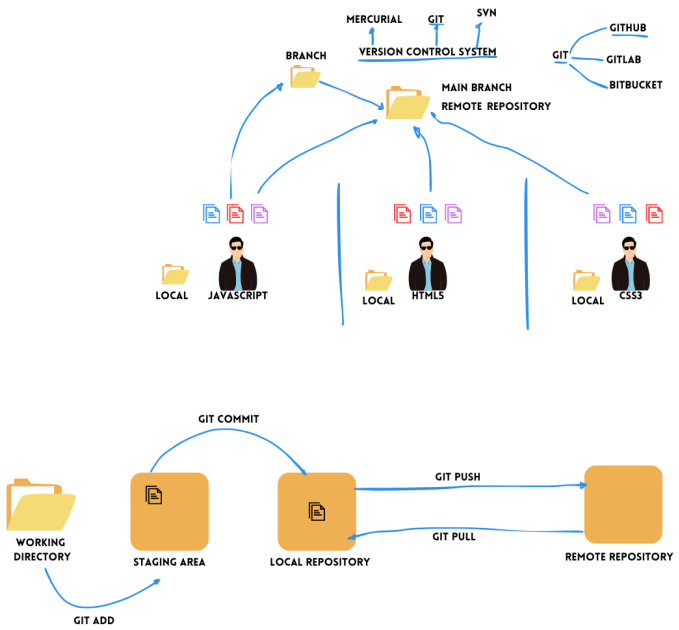
Here's how Git works and what it's used for:

1. Version Control: Git allows developers to track changes to their codebase over time. Every time a change is made, Git records a snapshot of the entire project. This makes it easy to compare different versions of the code, revert to previous states, and understand how the code has evolved.
2. Distributed System: Unlike centralized version control systems, Git is distributed. This means that every developer working on a project has a full copy of the repository on their local machine. This local copy enables developers to work

offline, make changes, commit them, and then later sync those changes with the central repository or other developers' repositories.

3. Branching and Merging: One of Git's powerful features is its ability to create branches. A branch is essentially a separate line of development. This allows developers to work on new features or bug fixes in isolation without affecting the main codebase. Once a feature or fix is complete, the branch can be merged back into the main codebase.
4. Collaboration: Git facilitates collaboration among multiple developers. Each developer can work on their own branch, and when ready, they can merge their changes into a shared branch, such as the "master" branch. Git's branching and merging capabilities make it possible for teams to work on different features concurrently and integrate those features smoothly.
5. History and Accountability: Git maintains a detailed history of all changes made to the codebase. This includes who made each change, when it was made, and the purpose of the change (commit messages). This historical record is valuable for tracking down bugs, understanding the evolution of the project, and reviewing the work of contributors.
6. Open Source and Hosting Platforms: Many open-source projects and commercial software projects host their Git repositories on platforms like GitHub, GitLab, and Bitbucket. These platforms provide additional features such as issue tracking, code review, continuous integration, and more, which enhance the development process.

🎯 Git Areas



👉 Working Directory

The working directory is the area where you make changes to your project files. It is simply the current state of your project on your local machine. You can modify, add, or delete files in this directory as you work on your project.

👉 Staging Area

The staging area acts as a middle ground between the working directory and the repository. It allows you to selectively choose which changes you want to include in the next commit.

Before committing your changes, you add (or stage) specific files or changes to the staging area. When you make changes to your files, Git recognizes these changes in the working directory. By explicitly adding these changes to the staging area, you are telling Git that you want to include them in the next commit.

The staging area allows you to review and organize your changes before making them a permanent part of the repository. You can add or remove changes from the staging area as needed until you are satisfied with the snapshot you want to commit.

👉 Repository

The repository is where Git permanently stores the committed snapshots of your project. When you make a commit, Git takes the contents of the staging area and creates a new snapshot, which includes all the changes you have staged. These snapshots are stored in the repository along with a unique identifier called a commit hash.

The repository contains the complete history of all commits made to the project. It enables you to access previous versions of your files, view the commit history, and revert to earlier states if needed.

Commits in the repository are organized in a directed acyclic graph (DAG) structure, where each commit references its parent commit(s). This structure enables Git to track the entire history of the project, including branching and merging.

🎯 Git Operations

1. clone: Cloning a repository means creating a local copy of a remote Git repository. This is typically the first step when you want to start working with a project. To clone a repository, you use the `git clone` command followed by the URL of the remote repository:
2. add: After cloning a repository or making changes to files in your local repository, you need to stage the changes you want to commit. Staging is done using the `git add` command. This command adds the specified file(s) to the staging area, indicating that you want to include these changes in the next commit.
3. commit: A commit in Git represents a snapshot of your code at a specific point in time. After adding changes to the staging area, you create a commit using the `git commit` command. This command creates a commit with the changes you've staged. The commit message should describe the purpose of the changes made in this commit.
4. push: Once you've made commits in your local repository, you might want to share those changes with the remote repository so that others can access them. To do this, you use the `git push` command. This command sends your local commits to the remote repository on the specified branch. The remote repository gets updated with your changes.

5. pull: When working collaboratively, it's important to keep your local repository up-to-date with the latest changes from the remote repository. The `git pull` command allows you to fetch the latest changes from the remote repository and merge them into your local branch.This command fetches changes from the remote repository and automatically merges them into your local branch. If conflicts arise, you'll need to resolve them manually before proceeding.