Git and GitHub:

Git is a distributed version control system that allows multiple developers to work on a project simultaneously without interfering with each other's work. It keeps track of changes made to files and directories over time, enabling collaboration and maintaining a history of modifications.

GitHub is a web-based platform that uses Git for version control. It provides a central repository to host projects and offers features like issue tracking, pull requests, code reviews, and collaboration tools.

Version Control:

Version control is a system that manages changes to files or sets of files over time. It allows you to track revisions, revert to previous versions, and collaborate with others.

Branch:

A parallel version of your project where you can make changes without affecting the main version.

Git Repository:

A Git repository (repo) is a storage space where your project files and their revision history are kept. Repositories can be local to your computer or hosted on a remote server (e.g., GitHub, GitLab, Bitbucket).

Key Features of Git Repository:

Version Control: Tracks changes to files over time, allowing you to revert to previous states.

Branching and Merging: Allows multiple development paths (branches) and the ability to merge them.

Collaboration: Facilitates teamwork by synchronizing changes between different contributors.

pull requests:

Facilitate code collaboration and review. They allow developers to notify team members about changes they've pushed to a branch in a repository.

commit:

A commit in GitHub is a record of changes made to a repository's files. It is an essential part of version control, allowing developers to track and manage changes to their codebase.

staging area:

The staging area (also known as the index or cache) in Git is an intermediate area where changes are gathered before they are committed to the repository. It allows you to prepare and review your changes before finalizing them with a commit.

Tags in github:

Tags in Git are like bookmarks you add to important points in your project’s history. They help you quickly identify and access specific commits that you want to remember, such as major releases or milestones.

git init:

git init is a command used to create a new Git repository. When you run git init in a directory, it initializes a new Git repository in that directory, setting up the necessary files and structures for version control.

git clone:

Git clone is a command used to create a copy of an existing Git repository. It downloads the entire repository, including its history, from a remote source to your local machine.

git add . :

git add . stages all the files in your current directory and its subdirectories, preparing them for the next commit. This means it includes all new, modified, and deleted files in the staging area. When you commit, all these staged changes will be saved.

git fetch:

Purpose: git fetch downloads the latest changes from the remote repository but does not merge them into your local branch.

git pull:

Purpose: git pull is a combination of git fetch followed by git merge. It downloads the latest changes from the remote repository and directly merges them into your current branch.

git stash:

git stash is a command in Git that allows you to temporarily save (or "stash") changes you have made to your working directory and index, so you can switch to another branch or perform other operations without committing those changes. The stashed changes can be reapplied later.

git stash apply:

git stash apply is used to reapply the changes that were previously stashed with git stash. This command restores the stashed changes to your working directory, allowing you to continue working on them.

git rebase:

git rebase is a Git command that allows you to integrate changes from one branch into another. It essentially moves or "replays" your commits from one branch onto another, which can result in a cleaner, more linear commit history.

git merge:

git merge is a Git command used to combine the changes from one branch into another. When you merge branches, Git integrates the commits from the source branch into the target branch, creating a new merge commit that has two parent commits: one from the target branch and one from the source branch. This method preserves the complete history of both branches.

git cherry-pick:

git cherry-pick is a Git command that allows you to apply a specific commit from one branch onto another branch. This can be useful when you want to include a particular change without merging the entire branch.

git log --oneline:

git log --oneline is a Git command used to display the commit history in a compact, one-line-per-commit format. This can make it easier to quickly review the commit history of a repository.

git revert:

git revert is a Git command used to create a new commit that undoes the changes made by a previous commit.

git show:

git show is a Git command used to display detailed information about a specific commit, including the commit message, author, date, and the changes made to the files. This command is useful for reviewing the specifics of a commit.

Command to reset head of the master:

To reset the HEAD of the master branch in Git, you can use the **git reset** command.

Command to delete untracked files:

To delete untracked files in Git, you can use the **git clean** command. This command helps to remove untracked files and directories from your working directory.

Files in GitHub:

Tracked Files: Files that are part of the repository and can be unmodified, modified, or staged.

Untracked Files: Files in the working directory that are not being tracked by Git.

Ignored Files: Files specified in the .gitignore file that Git should ignore.

Command to compare current working directory with the last commit:

To compare the current working directory with the last commit in Git, you can use the **git diff** command. This command shows the differences between your working directory and the latest commit.

Metadata and Data:

In Git, the "thing" that contains both metadata and data, and is often referred to as the heart of Git, is the commit object.

Each commit in Git includes:

Data:The state of the repository at a specific point in time (i.e., a snapshot of the file tree).

The "state of the repository" in Git refers to the complete set of files, their contents, and the directory structure at a specific point in time. It encompasses everything in the working directory and the staging area, providing a snapshot of the entire project.

Metadata:Information such as the commit message, the author, the committer, timestamps, and references to parent commits (for history tracking).

Advantage, Disadvantage and Applications of GitHub:

Advantages: GitHub enhances collaboration, version control, integration, open-source contributions, documentation, and project management.

Disadvantages: Costs for advanced features, learning curve, dependency on external service, file size limits, and security concerns.

Applications: Used in software development, open-source projects, documentation, project management, CI/CD pipelines, and portfolio hosting.

<u>How large files are managed in GitHub:</u>

To manage large files in GitHub, use Git Large File Storage (LFS).Git LFS replaces large files with pointers in your repository and stores the actual files separately.

Git Large File Storage (LFS) stores the actual files on a remote server specifically designed for large file storage, separate from the main Git repository. When you push changes to a Git repository that uses Git LFS, the large files are uploaded to this dedicated storage, while the repository itself co replaces large files with pointers in your repository and stores the actual files separately.

Git Large File Storage (LFS) stores the actual files on a remote server specifically designed for large file storage, separate from the main Git repository. When you push changes to a Git repository that uses Git LFS, the large files are uploaded to this dedicated storage, while the repository itself contains only lightweight pointers to these files.