Git-Github

Version Control Systems
A Version Control System (VCS) is a tool used in software development and collaborative projects to track and manage changes to source code, documents, and other files. Whether you are working alone or in a team, version control helps ensure your work is safe, organized, and easy to collaborate on. It allows developers to:

- Record and track every update to the codebase
- Collaborate on code without overwriting each other's work
- Revert to earlier states of the project if needed
- Maintain a detailed and structured history of the project’s evolution

Types of Version Control Systems

There are three main types of Version Control Systems:

1. Local Version Control Systems (Local VCS)

A Local Version Control System operates entirely on your personal machine without any connection to a remote repository. All changes and version history are stored in a local database on your computer.

Key Characteristics

- No internet or server dependency.
- Useful for individual projects.
- Limited to single-user environments.

2. Centralized Version Control Systems

In a Centralized Version Control System, all the files and their version history are stored in a single central server. This server acts as the main source for the entire project. Developers connect to this server to access or modify files but they do not maintain a full local copy of the project, instead they work with the most recent versions pulled from the server.

3. Distributed Version Control Systems** (Ex: Git is an example of this)

Distributed version control systems contain multiple repositories. Each user has their own repository and working copy. Just committing your changes will not give others access to your changes. This is because commit will reflect those changes in your local repository and you need to push them in order to make them visible on the central repository.

Git Introduction
Git is a version control system that helps you keep track of all the changes made to your code. It is like a time machine for your code. Whenever you make changes to a file, Git can store those changes, so you can go back to previous versions if something goes wrong.

Use of Git

Git brings numerous advantages to developers:

1. **Collaboration**: Git enables multiple developers to work on the same project simultaneously. Changes can be merged seamlessly, and conflicts can be resolved easily.
2. **History Tracking**: Every change is recorded, allowing you to revert to previous versions of your code if something goes wrong.
3. **Branching and Merging**: Git allows you to create branches for new features or experiments without affecting the main codebase. Once the feature is ready, it can be merged back into the main branch.
4. **Distributed Development**: Each developer has a complete copy of the repository, including its history. This decentralization enhances collaboration and backup capabilities

Core Concepts of Git

1. Repositories

A **repository** (or repo) is a storage space where your project files and their history are kept. There are two types of repositories in Git:

- **Local Repository**: A copy of the project on your local machine.
- **Remote Repository**: A version of the project hosted on a server, often on platforms like GitHub, GitLab, or Bitbucket.

2. Commits

A ****commit**** is a snapshot of your project at a specific point in time. Each commit has a unique identifier (hash) and includes a message describing the changes made. Commits allow you to track and review the history of your project.

3. Branches

Branches allow developers to work on separate tasks without affecting the main codebase. Common branch types include:

- ****Main (or Master) Branch****: The stable version of the project, usually production-ready.
- ****Feature Branch****: Used for developing new features or bug fixes.

4. Merging

****Merging**** is the process of integrating changes from one branch into another. It allows you to combine the work done in different branches and resolve any conflicts that arise.

5. Cloning

****Cloning**** a repository means creating a local copy of a remote repository. This copy includes all files, branches, and commit history.

6. Pull and Push

- ****Pull****: Fetches updates from the remote repository and integrates them into your local repository.
- ****Push****: Sends your local changes to the remote repository, making them available to others.

Introduction to Github
Collaboration and version control are important for software development. ****GitHub**** has become an important platform for developers, enabling seamless teamwork and efficient project management.

- GitHub is a web-based platform that uses Git, a version control system, to help developers manage and track changes in their code.
- It allows multiple people to collaborate on a project, track revisions, and contribute to code from anywhere in the world.
- GitHub offers both free and paid plans, catering to individuals and large organizations alike

Git Configuration & Setup

Here are Git configuration and setup commands:

Commands	Description
git config --global user.name "Your Name"	Set your Git username globally for all repositories.
git config --global user.email "youremail@example.com"	Set your Git email address globally.
git config --global color.ui auto	Set to display colored output in the terminal for better readability.
git config --global alias.	Create a custom alias for a Git command to save time.
git config --list	List all Git configuration settings (global, system, local).
git config --get	Retrieve the value of a specific configuration key (e.g., `user.name`).
git help	Display the main help documentation, showing a list of commonly used Git commands.

Initializing a Repository

Here are the Git initializing a repository commands:

Commands	Description
git init	Initializes a new Git repository in the current directory.
git init	Creates a new Git repository in the specified directory.
git clone <repository_url>	Clone a repository from a remote server to your local machine.
git clone --branch <branch_name> <repository_url>	Clones a specific branch from a remote repository.

Basic Git Commands

Here are some basic Git commands:

Commands	Description
git add	Adds a specific file to the staging area.
git add . or git add --all	Adds all modified and new files to the staging area.

Commands	Description
git status	Shows the current state of your repository, including tracked and untracked files, modified files, and branch information.
git status --ignored	Displays ignored files in addition to the regular status output.
git diff	Shows the changes between the working directory and the staging area (index).
git diff	Displays the differences between two commits.
git diff --staged or git diff --cached	Displays the changes between the staging area (index) and the last commit.
git diff HEAD	Display the difference between the current directory and the last commit
git commit	Creates a new commit with the changes in the staging area and opens the default text editor for adding a commit message.
git commit -m "" or git commit --message ""	Creates a new commit with the changes in the staging area and specifies the commit message inline.
git commit -a or git commit --all	Commits all modified and deleted files in the repository without explicitly using git add to stage the changes.
git notes add	Creates a new note and associates it with an object (commit, tag, etc.).
git restore	Restores the file in the working directory to its state in the last commit.
git reset	Moves the branch pointer to a specified commit, resetting the staging area and the working directory to match the specified commit.
git reset --soft	Moves the branch pointer to a specified commit, preserving the changes in the staging area and the working directory.
git reset --hard	Moves the branch pointer to a specified commit, discarding all changes in the staging area and the working directory, effectively resetting the repository to the specified commit.
git rm	Removes a file from both the working directory and the repository, staging the deletion.
git mv	Moves or renames a file or directory in your Git repository.

Branching and Merging

Here are some Git branching and merging commands:

Commands	Description
git branch	Lists all branches in the repository.
git branch	Creates a new branch with the specified name.

Commands	Description
git branch -d	Deletes the specified branch.
git branch -a	Lists all local and remote branches.
git branch -r	Lists all remote branches.
git checkout	Switches to the specified branch.
git checkout -b	Creates a new branch and switches to it.
git checkout --	Discards changes made to the specified file and revert it to the version in the last commit.
git merge	Merges the specified branch into the current branch.
git log	Displays the commit history of the current branch.
git log <branch-d	Displays the commit history of the specified branch.
git log --follow	Displays the commit history of a file, including its renames.
git log --all	Displays the commit history of all branches.
git stash	Stashes the changes in the working directory, allowing you to switch to a different branch or commit without committing the changes.
git stash list	Lists all stashes in the repository.
git stash pop	Applies and removes the most recent stash from the stash list.
git stash drop	Removes the most recent stash from the stash list.
git tag	Lists all tags in the repository.
git tag	Creates a lightweight tag at the current commit.
git tag	Creates a lightweight tag at the specified commit.
git tag -a -m ""	Creates an annotated tag at the current commit with a custom message.

Remote Repositories

Here are some Git remote repositories commands:

Commands	Description
git fetch	Retrieves change from a remote repository, including new branches and commit.
git fetch	Retrieves change from the specified remote repository.
git fetch --prune	Removes any remote-tracking branches that no longer exist on the remote repository.
git pull	Fetches changes from the remote repository and merges them into the current branch.

Commands	Description
git pull	Fetches changes from the specified remote repository and merges them into the current branch.
git pull --rebase	Fetches changes from the remote repository and rebases the current branch onto the updated branch.
git push	Pushes local commits to the remote repository.
git push	Pushes local commits to the specified remote repository.
git push	Pushes local commits to the specified branch of the remote repository.
git push --all	Pushes all branches to the remote repository.
git remote	Lists all remote repositories.
git remote add	Adds a new remote repository with the specified name and URL.
git remote rm	Remove a connection to a remote repository (e.g., `origin`).
git remote rename <old_name> <new_name>	Rename an existing remote connection.

Git Comparison

Here are some Git comparison commands:

Commands	Description
git show	Shows the details of a specific commit, including its changes.
git show	Shows the details of the specified commit, including its changes.

Git Logging and Reviewing Commands

Here are some Git Logging and Reviewing commands:

Command	**Description**
`git log`	Displays the commit history of the current branch.
`git log --oneline`	Shows commits in a compact format (1 line per commit).
`git log --graph`	Displays an ASCII graph of the branch history alongside log output.
`git log --all`	Shows commit logs for all branches.
`git log --author="Name"`	Shows commits made by a specific author.
`git log --since="2 weeks ago"`	Filters commits made in the last 2 weeks.

Command	**Description**
`git log --until="2024-12-31"`	Shows commits made ****before**** a specific date.
`git log <file>`	Shows the commit history of a specific file.
`git log --follow <file>`	Tracks file history including renames.
`git show`	Displays the full details of a specific commit (diff + metadata).
`git show <commit>`	Shows information about the given commit hash.
`git blame <file>`	Shows which commit last modified each line of a file.
`git diff`	Compares working directory and staging area (unstaged changes).
`git diff --staged`	Compares staged changes with the last commit.
`git diff <commit1> <commit2>`	Shows the difference between two commits.

Git Managing History

Here are some Git managing history commands:

Commands	Description
git revert	Creates a new commit that undoes the changes introduced by the specified commit.
git revert --no-commit	Undoes the changes introduced by the specified commit, but does not create a new commit.
git rebase	Reapplies commits on the current branch onto the tip of the specified branch.

Git Reflog – Recovering Lost Commits

Here are some commands for recovering lost commits:

Command	**Description**
`git reflog`	Show history of HEAD changes (including resets, rebases, checkouts)
`git checkout HEAD@{n}`	Restore a previous HEAD state
`git reset --hard HEAD@{n}`	Hard reset to a previous state