Git and Github

Initial Setup (First Time)

1. cd path/to/your-project # Go to your project folder
2. git init # Initialize Git in the folder
3. git remote add origin https://github.com/your-username/your-repo-name.git # Link to GitHub repo
4. git add . # Stage all files for commit
5. git commit -m "Initial commit" # Create the first commit with a message
6. git branch // cheek the local reposity branch name (master/main)

if you push the code in github main branch then we need a main branch in our local repo if this is not present then use git checkout -b main also if you want to push the code in gihub main branch without criating new local branch use git push origin master:main if you work only main branch then you can rename your master branch to main branch using git branch -m master main

git push -u origin main # Push changes to GitHub

Future Updates

git add . # Stage changes

git commit -m "Your message" # Commit changes

git push # Push to GitHub

**Checking Git Status**

* To see what’s happening in your local repo (e.g., which files have been modified),
* This command will show you if there are any changes, staged files, or untracked files.

git status

**Seeing Commit History**

* You can view the commit history of your local repository by running:
* This will show you a list of commits you’ve made in this local repository.

git log

--------------------------------------------------------------------------------------------------------------------------------------

git commit

in simpal word sappose you make some chanches in your file mean local file then its no record anyware and this is temparie mens you make again chanches then previous changes are vanished then this isssu solve the commit commamd

when you make some chanch then add this file mean git add then your next command is git commit -m "Initial commit" this command are save all chanches in your local repository . so you are parmantly save it after some time you realize or need this cod you can retived this code issily

When you make edits to files on your laptop, those changes are only stored temporarily on your system. If something goes wrong or if you want to go back to an earlier version, there’s no record of previous versions unless you use version control. **Git** creates these “snapshots” (commits)

**Make Changes**: You edit files or create new files in your project folder on your laptop. These changes are not automatically saved in your Git repository.

**Stage Changes**: When you're ready to save your changes in Git, you use the command git add <file> (or git add . to add all changes). This moves the changes into the **staging area**, which is a temporary space where Git prepares your changes for the next snapshot.

**Commit Changes**: After staging your changes, you run the command git commit -m "Your commit message". This command takes a snapshot of the current state of all the staged files and saves it in your **local Git repository**. The commit becomes a permanent record of your changes.

**Push Changes**: If you want to share these changes with others or back them up to GitHub, you would then use the command git push. This sends your local commits to the remote repository (like GitHub).

Once you commit your changes in Git, you create a history of your project, which allows you to access previous versions of your files. Here’s how it works:

1. **History of Commits**: Each time you run git commit, Git saves a snapshot of your project at that point in time. This includes all the changes you staged. Each commit has a unique identifier (a hash) that you can use to reference that specific version.
2. **Accessing Previous Versions**:

* You can view the commit history using the command git log. This will show you a list of all commits made in the repository, along with their commit messages, dates, and unique identifiers.
* If you want to revert to a previous version of a file or the entire project, you can use commands like git checkout <commit\_hash> to switch to that version temporarily or create a new branch from it

1. **Reverting Changes**:

* If you want to undo changes after a commit, you can use git revert <commit\_hash>, which creates a new commit that undoes the changes made in the specified commit.

git clone

git clone https://github.com/username/repository.git

1 **Creating a Local Copy**: When you use git clone, Git downloads the entire repository, including all files, history, branches, and commits, to your local machine.

2 **Setting Up a Remote Connection**: After cloning, your local copy is set up with a connection to the original remote repository. This means you can easily pull updates and push your changes back to the remote repository.

When you **clone** a Git repository, you get the entire history of that repository, including all commits, branches, and files. However, the behavior you see when you open the cloned files depends on how you're looking at them. Let’s clarify a few key points:

**1. What Happens When You Clone**

* **Complete Repository**: When you use the command git clone <repo-url>, Git creates a complete copy of the repository on your local machine. This includes:
  + All branches
  + All commits
  + All tags
  + All files in their current state at the time of cloning
* **Working Directory**: The **working directory** (the folder where the files are located) will display the latest state of the default branch (often main or master). This means you see the latest files as they were at the last commit on that branch.

**2. Viewing Commits and History**

* **Last Commit**: The files in your working directory represent the last committed state of the default branch. So, when you open the files, you are looking at the most recent version of the files.
* **Viewing Previous Commits**: To see the entire history of the project (including previous versions of files), you need to use Git commands:
* **View Commit History**: You can run the following command in the terminal:

git log

This will show you a list of all commits made to the repository, including their messages, dates, and authors.

**Check Out Previous Commits**: If you want to view files as they were in a previous commit, you can check out that specific commit:

git checkout <commit-hash>

Replace <commit-hash> with the hash of the commit you want to view. Be careful with this command, as it will change your working directory to reflect that commit.

**Summary**

* After cloning, you see the latest state of files because that’s how the working directory reflects the default branch.
* The entire history is still present in the repository; it’s just not immediately visible in your working directory.
* You can explore all previous commits and the entire history of changes made in the repository using Git commands like git log and git checkout

**Clone to a Specific Directory**: If you want to clone the repository into a specific directory(folder) instead of using the default name, you can specify the directory name at the end:

git clone <repository-url> <directory-name>

git clone https://github.com/username/repository.git my-project

**Shallow Clone**: If you only want the latest version without the entire history, you can use the --depth option:

git clone --depth n <repository-url>

1 **Limited History**: You only get the latest n commits (where n is specified by the --depth option). For instance, --depth 1 means you get only the latest commit.

2 **Smaller Size**: Since it includes fewer commits, a shallow clone is usually much smaller in size compared to a full clone.

3 **Usage**: Useful when you only need the current state of the repository without its history. This is often used in scenarios like CI/CD pipelines, where you just need the latest version of the code.

Git pull

The git pull command is used to **update your local repository** with the latest changes from a remote repository

Git add file name

git commit -m “your messege”

git pull origin main

**Process of git pull**

1. **Fetch Updates**:
   * The git pull command first runs git fetch, which retrieves the latest commits and data from the remote repository without merging them into your local branch.
   * It downloads all new changes from the specified branch on the remote repository to your local repository, but it does not change your working files yet.
2. **Merge Updates**:
   * After fetching, git pull automatically tries to merge the fetched changes into your current branch using git merge.
   * If the changes do not conflict with your local changes, the merge occurs automatically, and you have the latest version of the code.
3. **Handle Merge Conflicts** (if necessary):
   * If there are changes in the same lines of code both in your local branch and the fetched branch, Git cannot automatically merge them. This situation is known as a **merge conflict**.
   * You will need to manually resolve the conflicts by editing the affected files. Git marks the conflicting areas in the file with markers like <<<<<<<, =======, and >>>>>>>.
   * After resolving conflicts, you stage the resolved files with git add <file> and then commit the changes using git commit

Supoose your local file contain some code but this code are not present in github repo then you use git pull so your local file and repo are updated but this extra line which is not present in github repo this line still present in your local file .

Then if you want to your local file and github file are exactly similar yhe after pull use this command

git reset --hard origin/main