**Git And GitHub**

**Git**

* Git is a version control system that allows developers to keep track of changes in the source code during software development.
* It was created by Linus Torvalds in 2005,Git became an essential tool for developer worldwide.
* Git is 2.45.1 is the latest version of GIT, released on May 2024.
* Git is used to collaborate with others and manage different versions of their projects efficiently.

**Benefits of Git**

* **Version Control:** Git helps in tracking changes, you can change to the previous state if something goes wrong.
* **Collaboration:** Git helps multiple developers to work on a single project without interfering with each other's work.
* **Backup:** Your whole project is saved on the Git repository. so it provides a backup of all versions(no accidental losses or mistakes)
* **Branching & Merging:** Git can create a separate copy of your main project (or clone copy)so you can try out new features or fix bug without affecting the main project.
* **Once you are satisfied with the changes you can merge your copy into the main project**
* **Industry Standard:**  Git is widely used in many software industries and it is an essential skill.
* **Offline work:** Git helps to commit changes and work on project even without an internet connection.

**Repository in Git**

* A repository is a storage location or storage space for your code and its history. You can think of a repository as a container that holds all your code.
* It contains all the project files and the entire revision history
* There are 2 types of repository - Local repso & Remote repso
* **Local repository:** These are present on your local machine you can create, modify &manage your local repository without needing an internet connection.
* **Remote repository:** Hosted on a remote server such as Github & allows you to share your work with others & collaborate on projects

**Check your git version**

To check your git version, you can run the following command:

Terminal window

**git --version**

This command will display the version of git installed on your system. Git is a very stable software and doesn’t get any breaking changes in majority of the cases, at least in my experience.

**Configuration in Git**

* To let git know who you are.

git config --global user. name “ Simran”

git config --global user.email”[simran2@email.com](mailto:simran2@email.com)”

* Use global to set the username & email.

**Git Status**

`git status` as checking the progress of your homework. When you use `git status`, you're asking Git, "What’s going on with my project right now?" It will show you:

1. \*\*What files have been changed\*\*: This is like seeing which parts of your homework you’ve updated.

2. \*\*What files are ready to be saved\*\*: These changes you’ve made are ready to be turned in.

3. \*\*What files still need to be saved\*\*: These are changes you’ve made but haven't marked as ready yet.

So, `git status` helps you keep track of what’s been done and what’s still left to do.

**Creating Git Folder**

* mkdir myproject – makes a new directory
* cd myproject –changes the current working directory.

**Initialize Git**

* git init –git knows the folder you initiated it on. git creates a hidden folder to keep of changes
* It is used to start a new git repository
* ‘git init’ creates a hidden ‘.git’directory in your project.

**Command in git**

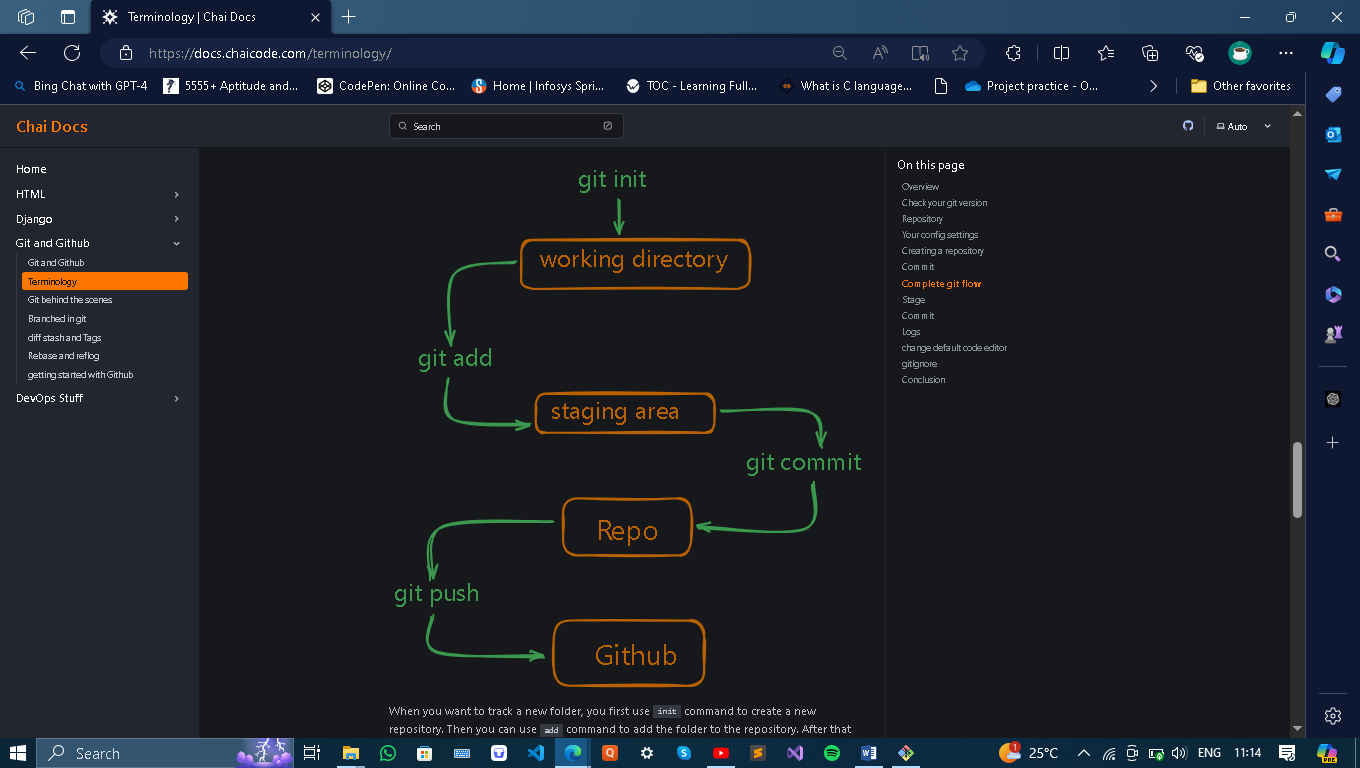
* ls- will list the file in the directory
* git add – add the file
* git add –all or git add –A : adds multiple files to repos

**Git Commit**

Commit is a way to save your changes to your repository. It is a way to record your changes and make them permanent. You can think of a commit as a snapshot of your code at a particular point in time. When you commit your changes, you are telling git to save them in a permanent way. This way, you can always go back to that point in time and see what you changed.

git commit –m “ Message”

**Complete git flow**

****

**Logs**

**git log**

This command will show you the history of your repository. It will show you all the commits that were made to the repository. You can use the --oneline flag to show only the commit message. This will make the output more compact and easier to read.

**Gitignore**

**Gitignore is a file that tells git which files and folders to ignore. It is a way to prevent git from tracking certain file or folders. You can create a git ignore file and add a list and folder to ignore by using the following command**

**Git Snapshots**

A git snapshot is a point in time in the history of your code. It represents a specific version of your code, including all the files and folders that were present at that time. Each snapshot is identified by a unique hash code, which is a string of characters that represents the contents of the snapshot.

A snapshot is not an image, it’s just a representation of the code at a specific point in time. Snapshot is a loose term that is used when git stores information about the code in a locally stored key-value based database. Everything is stored as an object and each object is identified by a unique hash code.

**3 Musketeers of git**

**The three musketeers of git are:**

* **Commit Object**
* **Tree Object**
* **Blob Object**

**Commit Object**

Each commit in the project is stored in .git folder in the form of a commit object. A commit object contains the following information**:**

**Tree Object**

Parent Commit Object

Author

Committer

Commit Message

**Tree Object**

Tree Object is a container for all the files and folders in the project. It contains the following information:

**File Mode**

**File Name**

**File Hash**

**Parent Tree Object**

Everything is stored as key-value pairs in the tree object. The key is the file name and the value is the file hash.

**Blob Object**

Blob Object is present in the tree object and contains the actual file content. This is the place where the file content is stored.

**Branching and Merging**

### 1. **Clone the Repository:**

To work with a repository from GitHub, you need to clone it to your local machine first. Use the following command:

git clone https://github.com/Simrannayak647/Learn-Git.git

### 2. **Navigate to the Cloned Repository:**

After cloning, you can navigate into the directory created by Git:

cd Learn-Git

### **1. Make Sure Your Local Repository is Up to Date**

Since you already cloned the repository and are on the main branch, ensure your local repository is up to date with the remote repository:

**git pull**

### **2. Create a New Branch**

Create a new branch where you’ll make changes. For example, you might create a branch named feature-branch:

**git branch feature-branch**

### **3. Switch to the New Branch**

Move to the new branch to work on it:

**git checkout feature-branch**

Alternatively, you can create and switch to a new branch in one command:

**git checkout -b feature-branch**

### **4. Make Changes to Files**

Edit or add files in your repository as needed. For example, you might modify file1 or file2.

### **5. Stage Your Changes**

After making changes, stage the files for commit:

**git add file1 file2**

Or if you want to stage all changes:

**git add .**

### **6. Commit Your Changes**

Commit the staged changes with a descriptive message:

**git commit -m "Made changes to file1 and file2"**

### **7. Push Your Branch to GitHub**

Push the new branch and its commits to GitHub:

**git push origin feature-branch**

### **8. Create a Pull Request on GitHub**

Go to your repository on GitHub. You should see an option to create a Pull Request (PR) for the feature branch. Click on it and follow the instructions to open a PR to merge feature-branch into main.

### **9. Merge the Pull Request**

After reviewing, merge the pull request on GitHub. This will integrate the changes from the feature-branch into the main branch.

### **10. Update Your Local** main **Branch**

Switch back to the main branch and pull the latest changes:

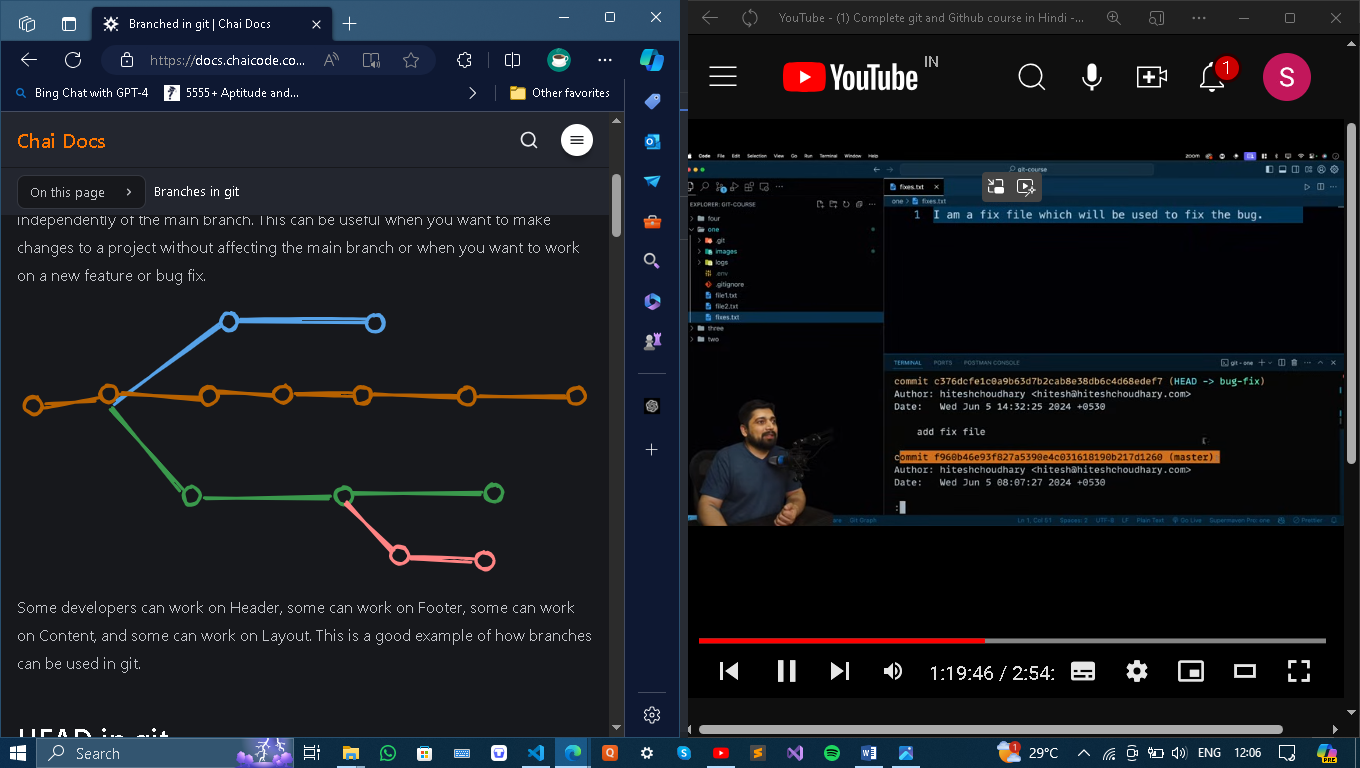
**git checkout main**

**git pull**

**Branched in git**

Branches in git

Branches are a way to work on different versions of a project at the same time. They allow you to create a separate line of development that can be worked on independently of the main branch. This can be useful when you want to make changes to a project without affecting the main branch or when you want to work on a new feature or bug fix.



Some developers can work on Header, some can work on Footer, some can work on Content, and some can work on Layout. This is a good example of how branches can be used in git.

**HEAD in git**

The HEAD is a pointer to the current branch you are working on. It points to the latest commit in the current branch. When you create a new branch, it is automatically set as the HEAD of that branch.

The default branch used to be called master, but it is now called main. There is nothing particularly special about main; it is just a convention. Creating a new branch

**To create a new branch, you can use the following command:**

**git branch**

**git branch bug-fix**

**git switch bug-fix**

**git log**

**git switch master**

**git switch -c dark-mode**

**git checkout orange-mode**

Some points to note:

**git branch** - This command lists all the branches in the current repository.

**git branch bug-fix** - This command creates a new branch called bug-fix.

**git switch bug-fix** - This command switches to the bug-fix branch.

**git log** - This command shows the commit history for the current branch.

**git switch master** - This command switches to the master branch.

**git switch -c dark-mode**- This command creates a new branch called dark-mode. the -c flag is used to create a new branch.

**git checkout orange-mode** - This command switches to the orange-mode branch.

Commit before switching to a branch

Go to .git folder and checkout to the HEAD file