**GIT AND GITHUB**

Git is a popular version control system. It was created by Linus Torvalds in 2005, and has been maintained by Junio Hamano since then.

It is used for:

* Tracking code changes or history of code
* Tracking who made changes
* Coding collaboration

**GitHub: The Collaboration Platform**

GitHub is the largest host of source code in the world, and has been owned by Microsoft since 2018.

GIT WORKFLOW

**Creating Folder using command promt**

mkdir foldername  makes a new directory.

cd foldername changes the current working directory.

**Note:** If you already have a folder/directory you would like to use for Git:

Navigate to it in command line, or open it in your file explorer, right-click and select "Git Bash here"

**1)Initialize Git Repository:**

To start using Git for your project, you need to initialize a Git repository, which track the changes of the project. Git creates a hidden folder to keep track of changes.

git init : the above command will create an empty git repos

**2)Create Files:** Once you have initialized the Git repository, you can start creating files or making changes to existing files in your project.

**touch filename.txt**

**what to insert in the file above created :**

**vi filename.txt:**

* **To enter insert mode:**
* Press the **i** key. This is the most common way to insert text before the current cursor position.
* **To exit insert mode and return to command mode, press the Escape (Esc) key.**
* **Exiting** vi **with Saving Changes:**
* **Press Esc:**This key takes you out of insert mode (where you can type text) and puts you in command mode.
* **Type :wq:** This command stands for "write and quit." It saves your changes to the file and exits vi.
* **Press**Enter**:** This executes the **:wq** command.
* **Exiting** vi **Without Saving Changes:**
* **Press**Esc**:** As before, this exits insert mode.
* **Type :q!:** This command stands for "quit without saving!" It exits vi and discards any unsaved changes.
* **Press**Enter**:** This executes the **:q**! command.

**what to view the content of the file :**

**cat filename.txt:**

**3)Stage Changes:** After making changes to your files, you need to stage them for committing. This means telling Git which changes you want to include in your next commit. You can stage individual files or all files using the following commands:

**git add filename.txt (adds single file)**

**git add . (adds all files) or** git add --all

The staging area, also referred to as the index in Git terminology, acts as a temporary holding zone for changes you plan to include in your next commit. It's like a staging area for a play before the final curtain rises.

Here's a breakdown of its role in the Git workflow:

**1. Working Directory:**

* This is your local workspace where you make changes to files in your project. These changes are initially untracked by Git.

**2. Staging Area:**

* Imagine this as a selection area. You use the git add command to mark specific files or changes within files that you want to capture in your next commit. These marked changes are now "staged."

**4)Commit Changes:** Once you have staged your changes, you can create a new commit. A commit is like a snapshot of your project at a specific point in time, with a unique identifier and a commit message describing the changes you made. To create a commit, use the following command:

**git commit -m “commit message”**

**5)View Status:** This command shows the current status of the working directory. It displays information about modified, staged, and untracked files.

**git status**

**6)View commit history:** This command displays the commit history for the current branch. It shows a list of all previous commits, their commit messages, authors, dates, and commit IDs.

**git log**

**7)Delete file from git history:**

**Delete the file from your working directory but keep it in Git history (soft delete):**

**git rm filename.ext**

**Permanently delete the file from both your working directory and Git history (hard delete):**

**git rm -f filename.ext**

**8)Hold files: uses when doesn’t want to commit but want to store the working file**

**git stash**

This command temporarily saves changes that you don't want to commit immediately. Stashing is useful when you need to switch branches or temporarily set aside your changes.

**Here's how** git stash **works:**

1. **Uncommitted Changes:** You've made modifications to files in your working directory, but haven't yet added them to a commit using git add.
2. git stash**:** When you run this command (optionally with a descriptive message), Git:
   * Takes a snapshot of your current working directory state, including both modified and staged changes.
   * Stores this snapshot as a "stash."
   * Reverts your working directory to a clean state, matching the latest committed version.

**Benefits of Using** git stash**:**

* **Clean Working Directory:** You can quickly switch branches or work on another task without your uncommitted changes cluttering your workspace.
* **Preserving Work:** Your uncommitted changes are safely stored in the stash, ready to be reapplied later.
* **Flexible Workflow:** Stashing allows you to work on different parts of your project independently, even if they involve changes to the same files.

**Common** git stash **Commands:**

* git stash: Creates a new stash (equivalent to git stash push).
* git stash save "message": Creates a new stash with a descriptive message.
* git stash list: Displays a list of your stashes with their corresponding messages.
* git stash show stash@{0}: Shows the details of a specific stash (replace 0 with the stash index).
* git stash apply: Applies the latest stash to your working directory, but doesn't remove it from the stash list.
* git stash pop: Applies the latest stash and removes it from the list.

**There are two main ways to paste text into Git Bash:**

**Method 1: Using the Mouse (Right-Click Menu)**

1. **Copy your text:** Copy the text you want to paste from another source (like a website or another application). You can use the standard Ctrl + C shortcut on your keyboard.
2. **Right-click in Git Bash:** Right-click anywhere within the Git Bash terminal window.
3. **Select "Paste":** In the context menu that appears, choose the "Paste" option. This will paste the copied text at the current cursor position.

**Extra:**

# Create a new directory for your project

mkdir my-project

cd my-project

# Initialize a new Git repository

git init

# Create a new file

touch README.md

# Add the file to the staging area

git add README.md

# Create your first commit

git commit -m "Initial commit"