**Terminal commands :**

- **`ls`**: lists all the files/folders inside the current path

- **`ls -a`**: lists all the files/folders inside the current path, including hidden files

- **`cd ..`**: go back one folder

- **`cd ../..`**: go back two folders

- **`cd ../../<folder name>`**: go back twice, then into a folder

- **`touch <file name>’**: Simple way (Git Bash) to create a file inside the current path

- **‘cat <file name>’**: used to display the contents (even codes) of any file and it works with any .txt, .js, .py, .java, etc. file

- ‘**mv <oldfile.txt> <newfile.txt>’**: renames the file on your system

- **`rm -rf <file name>`**: Deletes the file from the directory

**Observation:**

- Hidden folders/files usually start with a **`.`**

- **`.`** refers to the current directory

- **`..`** refers to the parent directory (one level up)

- That's why **`cd ..`** takes you back - you're literally telling it "go to the parent directory"!

**Git commands :**

- **`git init`**: Creates a new Git repository

- Creates a hidden **`.git`** folder

- Initializes basic Git configuration

- Enables Git tracking for that folder

- Inside the **`.git`** folder:

- **`HEAD`** (tracks current branch)

- **`config`** (repository settings)

- **`objects/`** (stores all your files)

- **`refs/`** (tracks branches)

- **`hooks/`** (custom scripts)

|  |
| --- |
| **Before `git init`**: |
| - Just a normal folder |
| - No version control |
| - Can't track changes |
| - Can't use Git commands |
| **After `git init`:** |
| - Folder becomes a Git repository |
| - Can track changes |
| - Can create branches |
| - Can connect to GitHub |

- **`git status`**: Tracks the history of changes made in the repository. Shows which files have been added, modified, or deleted.

- **‘git add**’: This command is used to stage changes before committing them

- **`git add .`** : Stages ALL changes in the current directory and its subdirectories

- **`git add <filename>`** : Stages changes only for the specified file

|  |
| --- |
| **Example:** |
| - **`git add names.txt`** : Only stages changes in names.txt |
| - Other modified files remain unstaged |

- **‘git restore <file name>/git restore --staged <file name>’**: You can unstage files using this

**NOTE:**

*You can always rely on* ***`git status`*** *to see what's staged (green) and what's not (red)*

**Before moving forward let’s know what's a staging area and a commit?**

**What is Staging Area?**

- Think of the staging area as a "preparation zone" or "packing area"

- It's like a box where you put things before shipping them

- Files you want to commit must go through staging first

**Why use a Staging Area?**

- You modified 3 files: A.txt, B.txt, C.txt

- You want to commit A.txt and B.txt together, but C.txt separately

- Staging lets you:

1. First stage and commit A.txt and B.txt

2. Then stage and commit C.txt later

**What is a Commit?**

- A commit is like taking a snapshot of your project at a specific point in time

- Each commit has:

- A unique ID (hash)

- A message describing what changed

- The actual changes made to files

- Information about when and who made the changes

**“Think of commits like saving a video game:”**

- Each save is a different point in your progress

- You can go back to any save point if needed

- Each save has a description of what you achieved

- You can see your whole journey through all your saves

**Why do we basically make commits?**

- Track History: See what changed, when, and why

- Backup Points: Can return to any previous commit if something breaks

- Collaboration: Other developers can understand your changes

**‘git commit’**: Creates a snapshot of your staged changes

- **`git commit -m "message"`** : Creates a commit with your staged changes

- The -m flag is for adding a commit message

|  |
| --- |
| **Example**: |
| `git commit -m "Added new student names" |

**So the typical workflow looks like:**

- Firstly make some changes to your files

- **`git status`** (check what's changed)

- **`git add`** (stage specific files or all files)

- **`git status`** (verify what's staged)

- **`git commit -m "message"`** (commit the staged changes)

- **‘git log’**: A commit history tracker. It gives you a detailed log of all the commits you've made in your repository, starting from the most recent one.

- ‘**git reset <UID>**’: Removes all commits that came after

|  |
| --- |
| **Case Study:** |
| 1. You create file.txt with content "Hello" (First commit: abc123) |
| 2. You add "World" to file.txt (Second commit: def456) |
| 3. You add "!!!" to file.txt (Third commit: ghi789) |
| When you do: |
| git reset abc123 |
|  |
| **Now what happens:** |
|  |
| The commit history will only show the first commit |
| BUT file.txt will still contain "Hello World!!!" in your folder |
| The changes aren't committed anymore, but the file itself keeps those changes |
| You'll see these changes as "unstaged" if you run git status |

**-‘git stash <file name>’**: Files are REMOVED from your working directory (they disappear from file manager) and the changes are saved in a hidden Git area so you can switch to other work and come back to them later.

-‘**git stash pop**’: Brings back the most recent stash

-‘**git stash clear’**: permanently deletes ALL your stashed changes.

**Now how to push your projects/folders to GitHub?**

First we’ll create a connection between our local project/folder with the remote repository on GitHub.

Create a new repository or open an existing a repository and then copy the URL of the repository present on GitHub. The URL will act as junction with the repository on github with your local project/folder. To attach the URL to your project/ folder use the command.

-**‘git remote add origin <URL>’**: This command establishes a connection between your local Git repository and a remote repository (GitHub).

After the connection has been made whatever changes and whenever the changes were made to our project/folder will reflect on the remote repository.

Since the connection has been made, so whenever we make any changes to our local project/folder we’ll not use the **‘git remote add origin <URL>’** again and again after every commit instead we’ll use the:

**-‘git push origin main’**: