*Git/GitHub*

**Git** :- (Open Source) it is a software and can be installed in your computer.

1.Git is a version control system that allows you to track changes to your files and collaborate with others.

2.Consider version control as a checkpoint in game. You can move to any time in the game and you can always go back to the previous checkpoint. This is the same concept in software development.

3.It is used to manage the history of your code and to merge changes from different branches.

4.This command will display the version of git installed on your system:

git -- version

**GitHub :-**

1.GitHub is a web-based hosting service for Git repositories.

2.GitHub is an online platform that allows you to store and share your code with others.

git status : The git status command is used to show the current status of your working directory and staging area in a Git repository. It provides information about which files are staged for commit, which files have been modified, and which files are untracked.

-> GitHub has a lot of settings that you can change. You can change your username, email, and other settings. Whenever you checkpoint your changes, git will add some information about your such as your username and email to the commit. There is a git config file that stores all the settings that you have changed. You can make settings like what editor you would like to use etc.

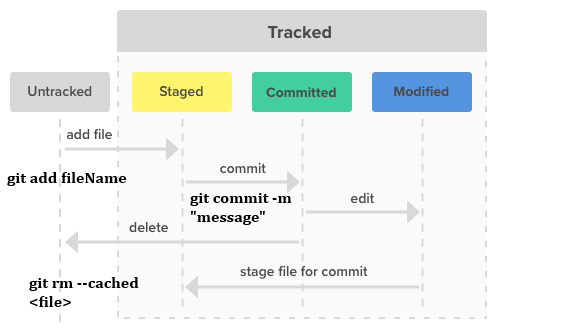
{

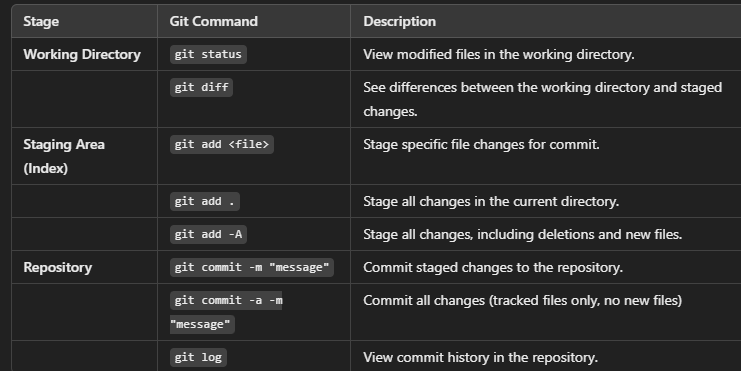
-> git config --global user.email "your-email@example.com"

-> git config --global user.name "Your Name"

}

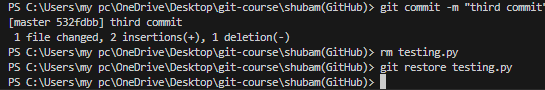
The git add command is used to **stage** changes in your working directory, preparing them to be included in the next commit. It doesn't actually commit the changes, but rather tells Git which changes you want to keep track of.

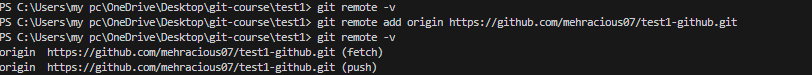


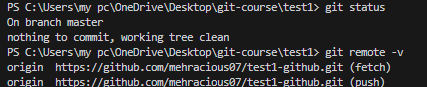


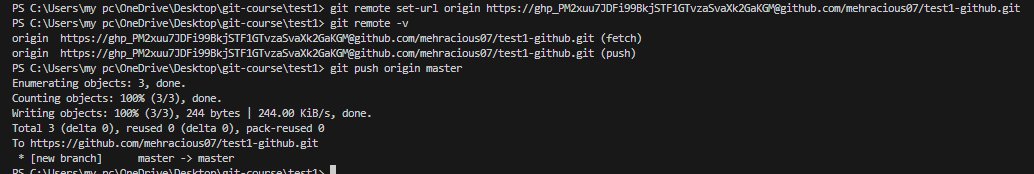
Token: ghp\_PM2xuu7JDFi99BkjSTF1GTvzaSvaXk2GaKGM

**How to push files to github repo**









git add <file> # Stages a specific file

git add . # Stages all changed files in the current directory and subdirectories

git add -A # Stages all changes, including file deletions

git add -u # Stages changes to tracked files (modifications and deletions, not new files)

Unstage:

git reset <file\_name>

git reset (for all file)

***Fork and Cloning :-***

 Forking a repository means creating a copy of someone else’s repository under your GitHub account. It allows you to freely make changes without affecting the original repository.

 Forking is usually done when you want to contribute to a project but you don't have direct write access to the original repository. After forking, you can make changes to your forked copy, and then you can submit those changes back to the original repository via a Pull Request (PR).

 **Cloning** is the process of creating a local copy of a repository (either your own or someone else's) on your computer. This is how you actually download the repository and begin working on it locally.

 After cloning, you can work with the files locally, make commits, and then push your changes back to the remote repository (e.g., GitHub).

**Fork + Clone Workflow (Typical Scenario for Contributions):**

1. **Fork** the repository on GitHub (if you want to contribute to someone else’s project).
2. **Clone** the forked repository to your local machine using the git clone command.
3. Make changes to the files locally.
4. **Commit** your changes.

*git add .*

*git commit -m "Your commit message"*

1. **Push** your changes to your forked repository on GitHub.

*git push origin master*

1. On GitHub, create a **Pull Request** (PR) from your forked repository to the original repository. This is where you propose your changes to be merged into the original project.

**Summary:**

* **Fork** is creating a copy of a repository in your GitHub account (you do this on GitHub itself).
* **Clone** is creating a local copy of the repository on your machine so you can work with it using Git
* The commands git pull and git push are two fundamental Git operations, but they serve different purposes. Let's break them down:

**git pull**

* **What it does**: Fetches the changes from the remote repository (e.g., GitHub) and merges them into your local repository. This updates your local working directory with any changes made by others or changes you pushed from other machines.
* **When to use it**: You use git pull when you want to update your local repository with the latest changes from the remote repository (e.g., if someone else has pushed changes that you want to get).
* **Command**:

git pull origin main

* + origin is the default name for the remote repository (you can replace it if your remote has a different name).
  + main is the branch you are pulling from (replace it with the branch you want to pull from, if different).
* **How it works**:
  + **Fetch**: Git first retrieves the changes from the remote repository.
  + **Merge**: Git then merges the fetched changes into your current branch.
* **Example scenario**: If you're working on a team project, someone else might push their changes to GitHub. You would use git pull to fetch those changes and incorporate them into your local branch.

**git push**

* **What it does**: Pushes your local changes (commits) to the remote repository, making them available to others.
* **When to use it**: You use git push when you want to upload your local commits to the remote repository (e.g., GitHub), so others can see and access your changes.
* **Command**:

bash

Copy

git push origin main

* + origin is the name of the remote repository (by default, it is origin).
  + main is the branch you're pushing to. If you're working on another branch, replace main with the appropriate branch name.
* **How it works**:
  + **Upload**: Git pushes your local commits to the remote repository.
  + **Sync**: The remote repository is now updated with your local changes, and others can see your updates.
* **Example scenario**: After you make changes, commit them locally, and want to share those changes with your team or update the central repository, you use git push to send your changes to GitHub.

**Key Differences:**

| **Command** | **Purpose** | **Direction** |
| --- | --- | --- |
| git pull | Fetches changes from the remote repository and merges them into your local repository. | Remote to Local |
| git push | Pushes your local commits to the remote repository. | Local to Remote |

**Workflow Example:**

1. **Start** by making sure your local repository is up to date:

bash

Copy

git pull origin main

This gets any new changes that might have been made by others.

1. **Make your changes** locally and commit them:

bash

Copy

git add .

git commit -m "Description of the changes"

1. **Push** your changes to the remote repository:

bash

Copy

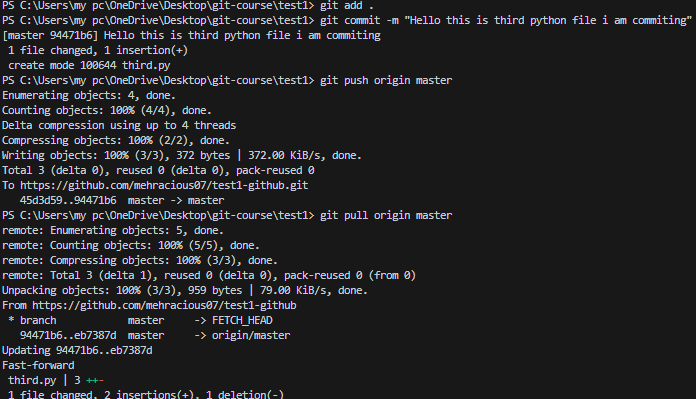
git push origin main

**What happens if you forget to git pull before git push?**

* If someone else has pushed changes to the repository after you last pulled, and you don't pull the changes before pushing, you might run into conflicts or errors. Git will prevent the push to avoid overwriting changes that others have made.
* If this happens, Git will give you a message saying that your local branch is behind the remote branch, and you will need to first pull, resolve any merge conflicts, and then push again.

**Conclusion:**

* Use git pull to get the latest changes from the remote repository to your local machine.
* Use git push to send your local changes to the remote repository.



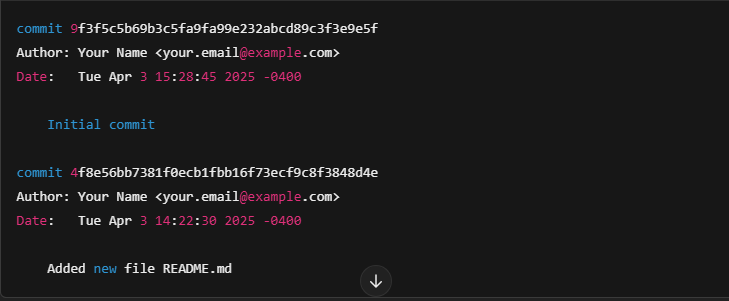
**git log**

The git log command in Git allows you to view the commit history of your repository. It shows a list of all the commits made in the repository, along with details like the commit ID, author, date, and commit message.

**Basic Command:**

git log

**Output:**

** Key Elements in git log Output:**

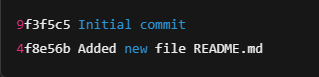
* **Commit Hash**: The long string (e.g., 9f3f5c5b69b3c5fa9fa99e232abcd89c3f3e9e5f) is a unique identifier for each commit.
* **Author**: The person who made the commit.
* **Date**: When the commit was made.
* **Commit Message**: A brief description of the changes made in that commit.

**Common Options with git log:**

1. **--oneline**: Displays each commit on a single line, showing the commit hash (shortened) and the commit message.

git log --oneline

Output:



**Git log --graph**: Visualizes the commit history as a graph, making it easier to see branching and merging.

git log --graph

1. **--all**: Shows the commit history for all branches, not just the current one.

git log --all

1. **--author="name"**: Filters the commits to show only those made by a specific author.

git log --author="Your Name"

1. **--since="date"** or **--until="date"**: Show commits within a specific date range.

git log --since="2025-04-01"

git log --until="2025-04-02"

1. **--stat**: Shows a summary of changes made in each commit, including the number of lines added or deleted.

git log --stat

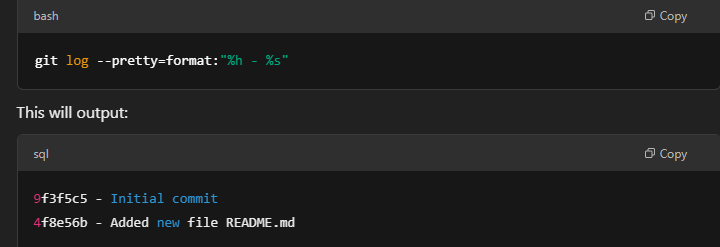
1. **-p** or **--patch**: Shows the diff (changes) introduced by each commit.

git log -p

1. **--pretty=format:"<format>"**: Allows custom formatting of the log output. For example, to display commit hash and message:

git log --pretty=format:"%h - %s"

This will output:



**Example: Viewing the Commit History in a More Readable Way**

To see a condensed view of the commit history with a graph, oneline, and colored output, you can use:

git log --oneline --graph --decorate --all

This shows:

* The commit hash in a shortened form.
* A graphical representation of the branch structure.
* The branch and tag names where the commits are located.

**Navigating git log Output:**

* **Scroll**: You can scroll through the commit history if it’s long. Use the arrow keys or Space to go down, b to go up.
* **Exit**: Press q to quit and return to the terminal.

**Example Workflow:**

1. You make a commit, then check your commit history:

bash

Copy

git log --oneline

1. You see your recent commit, then want to see the diff of that commit:

bash

Copy

git log -p

1. You want to check all commits made by a specific author:

bash

Copy

git log --author="Your Name"

Let me know if you need more details on git log or any other Git commands!

**You said:**

* how to exit log screen
* **ChatGPT said:**
* To exit the git log screen, which uses a pager (like less), you just need to press the following key:
* **Press q**
* **q** stands for "quit," and pressing it will exit the log view and return you to the terminal prompt.
* This works because git log uses a pager (usually less) to display the log history when the output is long and requires scrolling.
* Let me know if you need more help with this!