Git and GitHub

What is Git?

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
- Tracking who made changes
- Coding collaboration

Working with Git

- Initialize Git on a folder, making it a **Repository**
- Git now creates a hidden folder to keep track of changes in that folder
- When a file is changed, added or deleted, it is considered **modified**
- You select the modified files you want to **Stage**
- The **Staged** files are **Committed**, which prompts Git to store a **permanent** snapshot of the files
- Git allows you to see the full history of every commit.
- You can revert back to any previous commit.
- Git does not store a separate copy of every file in every commit, but keeps track of changes made in each commit!

Git Workflow Commands Overview

- Working Directory - Where you make changes
- git add - Stage changes
- git commit - Save changes to your repository
- git push - Share changes with others
- git status - Check what's going on
- Undo/Amend - Fix mistakes (git restore, git reset, git commit --amend)

GitHub

GitHub is a **web-based platform** used by developers to **store, manage, and share code**. It uses **Git**, a version control system, to track changes in code and allow collaboration among multiple people.
If we make changes on our system then we can reflect the changes on github.

- **Creating a GitHub repository**
- **Pushing code to GitHub using Git**

- **GitHub Pages (hosting websites)**
- **Collaborating on GitHub (pull requests, branches, issues)**
- **Setting up Git and GitHub for the first time**
- **GitHub Classroom or event repositories**
- **Creating a README.md or LICENSE file**

Key Git and GitHub Concepts

- **Repository:** A folder where Git tracks your project and its history.
- **Clone:** Make a copy of a remote repository on your computer.
- **Stage:** Tell Git which changes you want to save next.
- **Commit:** Save a snapshot of your staged changes.
- **Branch:** Work on different versions or features at the same time.
- **Merge:** Combine changes from different branches.
- **Pull:** Get the latest changes from a remote repository.
- **Push:** Send your changes to a remote repository.

Complete Guide: Git & GitHub (From Beginning to End)

1. Install Git

- First, install Git on your system.
- Download: https://git-scm.com/downloads

 # Check if Git is installed

Command:-

git –version

2. Set Up Git (First-Time Only)

Open git Bash --→

- git config --global user.name "Your Name"
- git config --global user.email you@example.com
- git config –list

This sets your identity for all your commits.

3. Create a GitHub

Go to https://github.com

Two-Way Workflow of Git & GitHub

1. Local System → GitHub

2. GitHub → Local System

1. Local System → GitHub

All basic commands for Local to remote
git init
git remote add origin <link>
git remote –v
git branch
git branch –m main
git push origin main

You **create a project on your local system** and then **push it to GitHub**.

Step 1: Create local project folder and Go to your project folder(VS CODE)
Mkdir my-project
cd my-project
Note :- Now we need to initialize this local folder as a Git repository.
Step 2 : Check this repo is git repo or local folder
ls –a(mac)
ls –Force(window)
Step 3: Initialize Git(go to folder)
git init
ls –a
Step 4 : Check status
git status
Step 5: Add your files
git add .
Step 6 : Check status
git status
Step 7: Commit your changes
git commit -m "Initial commit"
Step 8: Check status
git status
Step 9: Create a new repo on GitHub (no README)<go to git hub>
Step 10: Connect local to GitHub(vs code terminal)
git remote add origin repo link
To verify remote
git remote –v
Check branch
git branch
To rename branch(master ←→ main)
git branch -m main

Step 11: Push code to GitHub
git push -u origin main

Note :- Now create README.md file in VS code and status check, add, commit, status check, push

2. GitHub → Local System

You **clone an existing GitHub repository** to your **local system** to work on it.

Step 1: Copy the GitHub repo link

- Copy the repository link from GitHub (HTTPS or SSH).
- Open terminal/command prompt where you want the folder.

Check git version

git –version

Step 2: Clone the repo

git clone repo link

Step 3: Move into the project folder

cd repo-name

ls

ls -a

Step 4: Make changes, commit, and push as usual

Note :- Git status >add>commit >push

git status

git add .

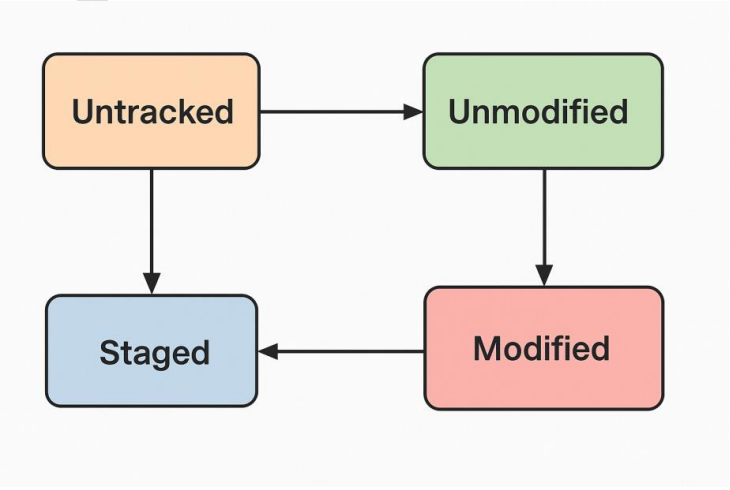
Note :- (add . or add file name)

git commit -m "Your message"

git push origin main

Summary :

GIT Repo Status :-

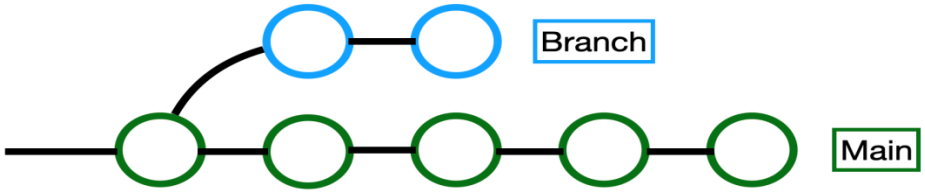


1.	**Untracked**	A file is in the untracked state when it exists in your project directory but is not yet being tracked by Git for version control.	File created but not added
2.	**Modified**	Changed	Change →Modify
3.	**Staged**	File is ready to be committed	Changed →add →modify
4.	**Unmodified**	Unchanged	change →modifeid →add →commit →unchange

GIT BRANCH

- Git branch is used to **manage branches** in a Git repository.
- In Git, a branch is like a separate workspace where you can make changes and try new ideas without affecting the main project. Think of it as a "parallel universe" for your code.

Common Reasons to Create a Branch



- Developing a new feature
- Fixing a bug
- Experimenting with ideas

Use	Command
List all branches	git branch
Create a new branch	git branch new-branch-name
Switch to another branch	git checkout branch-name or git switch branch-name
Create and switch in one command	git checkout -b new-branch-name or git switch -c new-branch-name
Delete a branch	git branch -d branch-name
Force delete (if not merged)	git branch -D branch-name
To rename branch	Git branch –m main

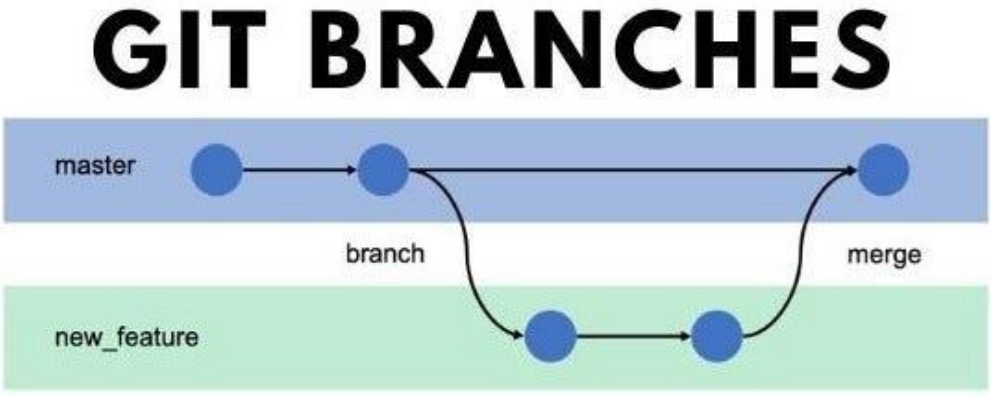
Example:-

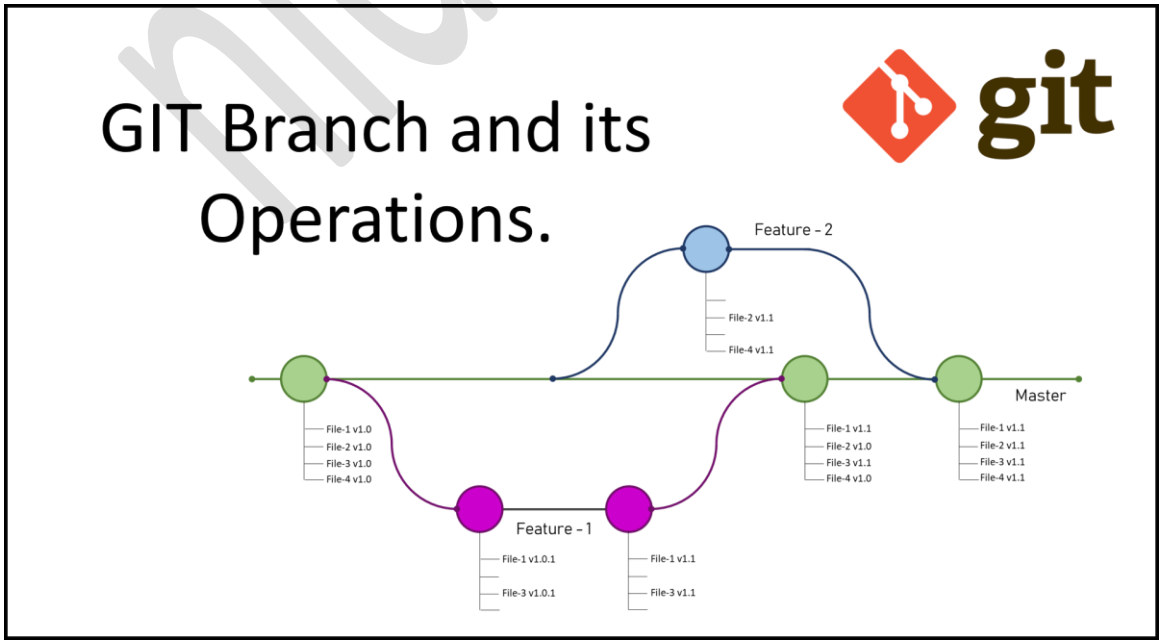
- git branch
- git branch –b features

- git branch
- git branch main
- git branch features
- git branch feature 1
- git checkout main
- git branch –d feature1
- change content of any file
- git status
- git add .
- git commit –m “message”
- git checkout feature 1
- git checkout main
- git push origin feature 1

Git Branch Merge

- Merging in Git means combining the changes from one branch into another.
- This is how you bring your work together after working separately on different features or bug fixes.





Common git merge Options

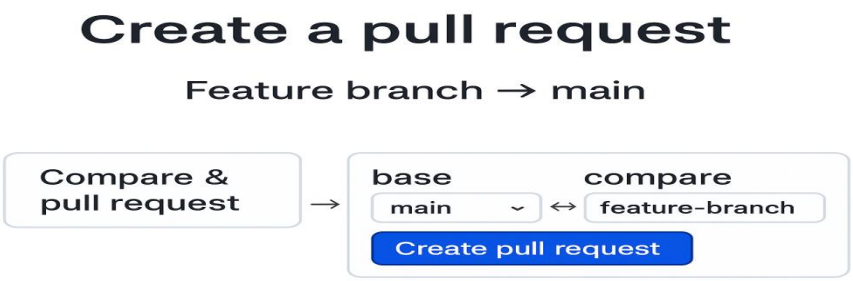
- git merge - Merge a branch into your current branch
- git merge --no-ff - Always create a merge commit
- git merge --squash - Combine changes into a single commit
- git merge --abort - Abort a merge in progress

Two Ways of Merging in Git and GitHub:-

1. Merging Using a **Pull Request** (on GitHub)
2. Resolving Merge Conflicts

1. Merging Using a Pull Request (on GitHub)

A **Pull Request (PR)** is the **recommended way** to merge code when working in teams. Instead of running git merge in your terminal, you request GitHub to do the merging after review.



Steps to Merge with a Pull Request

1. Push Your Feature Branch to GitHub

```
git checkout -b feature-branch
# make changes
git add .
git commit -m "Add feature"
git push origin feature-branch
```

2. Open Pull Request on GitHub

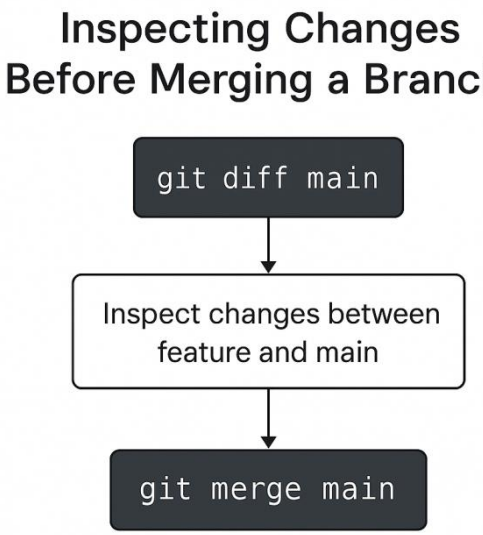
1. Go to your GitHub repository.
2. GitHub will show:
   "Compare & pull request" → Click it.
3. Choose:
   - **Base branch** (e.g., main)
   - **Compare branch** (your feature-branch)
4. Add title, description, and submit the PR.

3. Review & Merge

- Team members can review your PR.
- After approval (Check pull conflict → No conflict → Merge pull request → Pull request successfully merged and closed), click **"Merge pull request"**.
- Confirm the merge and optionally **delete the branch**.

2. Resolving Merge Conflicts

A **merge conflict** happens when Git can't automatically merge changes from two branches (usually because the **same line(s)** in a file were edited differently in each branch).



Feature 1 → add dropdown →add → commit
Main →add button →add →commit

1. git diff main

```
git diff main   #(feature 1)
```

Shows the difference between your current branch and the main branch but does NOT change anything**.**

Purpose:

- Review changes before merging.
- See what's different between branches.

Example:

```
git checkout feature-branch
git diff main
```

Output:

- You'll see code changes made in feature-branch that differ from main.

2. git merge main

Merges the **main** branch into your current branch**.**

Purpose:

- Bring changes from main into your working branch.
- Useful if main has been updated (to avoid conflicts later).

Example:

git checkout feature-branch
git merge main

If there are conflicts: You must resolve them.

ERROR REFLECTS →Choose any one option → add → commit →git diff → go to main → git merge feature 1 → git push

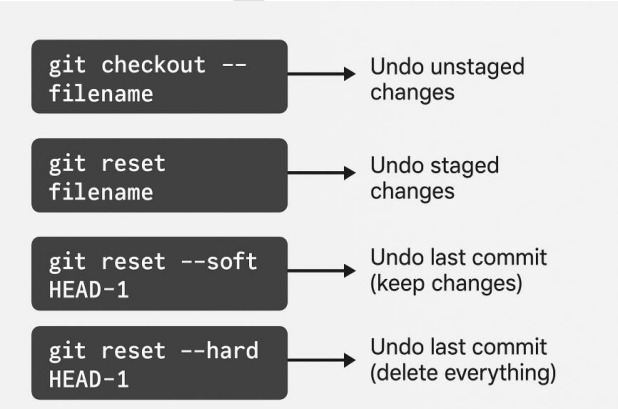
Typical Flow Combining Both:

git checkout feature-branch
git fetch origin
git diff main # Check differences before merging
git merge main # Merge main into feature branch

Then resolve any conflicts → git add . → git commit

Undoing Changes in Git

Git gives you multiple ways to undo changes depending on the **stage of the change**:



1. Undo Unstaged Changes
2. Undo Staged Changes
3. Undo Last Commit (Keep Changes)
4. Undo Last Commit (Unstage Changes)
5. Undo Last Commit (Delete Everything)
6. Undo a File from a Specific Commit

1. Undo Unstaged Changes

Changes made but not added to staging

git checkout – filename

Restores the file to the last committed version (deletes local edits).

2. Undo Staged Changes

Changes added with git add, but not yet committed

git reset filename or git reset HEAD ~1 # one step back

C1 C2 C3 C4 (HEAD)

git log # check log

Unstages the file, but keeps the local edits.

3. Undo Last Commit (Keep Changes)

git reset --soft HEAD~1

Removes the last commit but keeps files staged.

4. Undo Last Commit (Unstage Changes)

git reset --mixed HEAD~1

Removes last commit and unstages files (keeps your code).

5. Undo Last Commit (Delete Everything)

git reset --hard HEAD~1
Deletes the commit and code changes permanently.

6. Undo a File from a Specific Commit (FOR MANY COMMITS OR COMMITTED CHANGES)

git checkout <commit_hash> -- filename
OR
git log →Hash →copy
git reset Hash
git status
git log → History
git reset – hard Hash #both git / vs code remove changes

git reset < commit Hash > git reset – hard < commit Hash >

Fork

A **fork** is a **copy of a repository** that lives on your GitHub account.

It lets you:

- Make changes freely without affecting the original project
- Submit pull requests to suggest changes to the original
- Experiment or build independently

How Forking Works

 You find a repository on GitHub

 Click the **Fork** button (top-right corner)

 GitHub creates a **copy of the repo under your account**

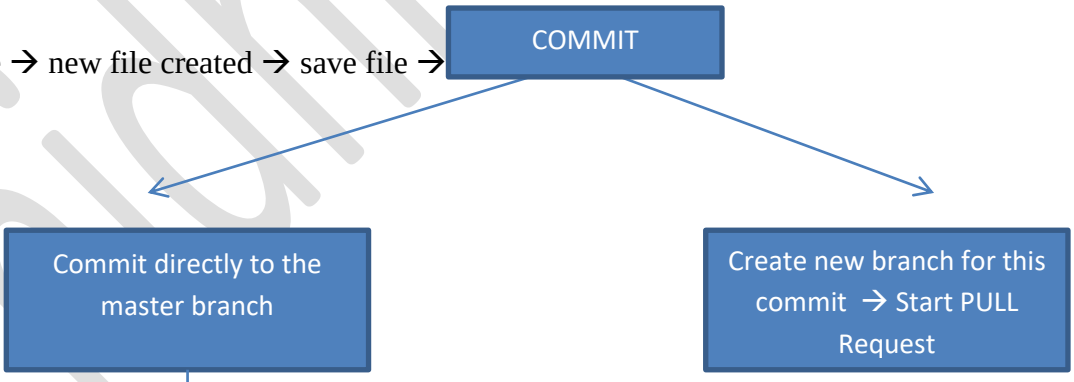
 You clone your fork to your local system:
git clone repo link

 Make changes in your fork

 Push changes and submit a **Pull Request** to the original repo

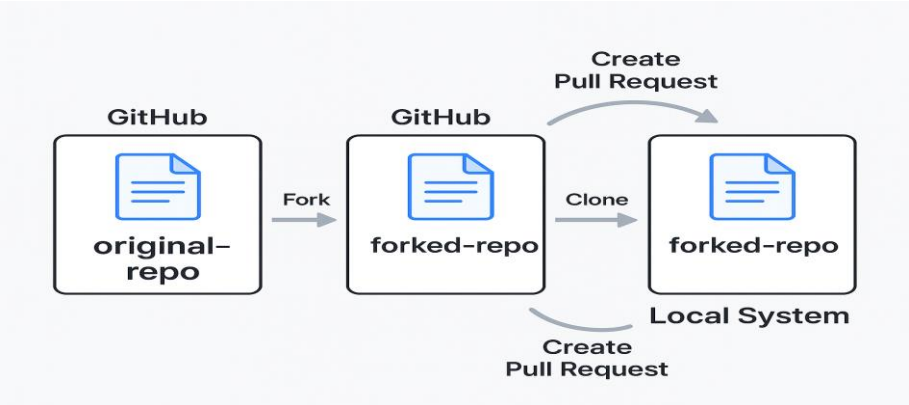
Open other's repo → go to fork → if you want to contribute to that project → as in same master or new branch

Click add file → new file created → save file →



Pull Request →New Pull Request → Click on your change file → Create pull request

Fork vs Clone



Action	Fork	Clone
Purpose	Copy repo to **your GitHub account**	Copy repo to your **local system**
Use Case	Open-source collaboration, submitting PRs	Local development
Scope	Happens on GitHub (remote)	Happens on your computer (local)

Git & GitHub Collaboration :-

GitHub Invitation :- A **GitHub invitation** allows someone to **collaborate** on a repository or join an organization.

1. Invite to Collaborate on a Repository

Steps:-

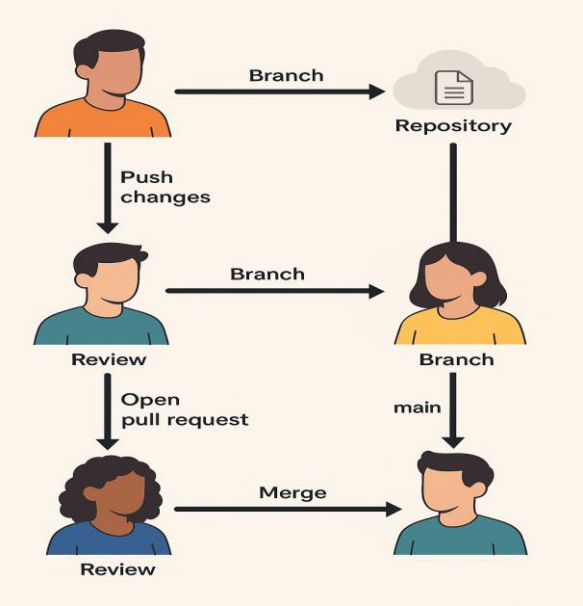
1. Go to your GitHub repo
2. Click **"Settings"** → **"Collaborators"**
3. Click **"Add people"**
4. Enter their GitHub username or email
5. Click **"Invite"**

They will receive an **email** and a **GitHub notification** to accept.

After Accepting:

- The invited user gets **access to the repo**
- They can **clone, push, open pull requests**, etc., based on permission

2. Common Collaboration Workflow



1. Clone the Repository
2. Create a New Branch
3. Make Changes, Then Stage & Commit
4. Push to GitHub
5. Open a Pull Request (on GitHub)
6. Review & Merge

Git Workflow Commands Overview

- Working Directory - Where you make changes
- git add - Stage changes
- git commit - Save changes to your repository
- git push - Share changes with others
- git status - Check what's going on
- Undo/Amend - Fix mistakes (git restore, git reset, git commit --amend)