**Git and github**

## ****What is Git?****

* **Git** is a **version control system (VCS)**.
* It keeps track of changes in files over time.
* It allows multiple people to work on the same project without overwriting each other’s changes.
* It helps you manage code history, go back to previous versions, create branches for new features, and merge them.

### ****Why use Git?****

### 1️⃣ ****Version Control – Track Changes Over Time****

* Git allows you to **record every change** made to your code or files.
* You can go back to any previous version if something breaks or you want to compare changes.
* Example: If you accidentally delete a function, you can recover it easily.

### 2️⃣ ****Collaboration – Work with Others Seamlessly****

* Multiple people can work on the same project at the same time.
* Git helps merge changes and resolve conflicts when two people edit the same file.
* Example: A team of developers working on different features can integrate their work without overwriting each other’s changes.

### 3️⃣ ****Branching – Work on Features Independently****

* You can create a **branch** to work on new features or fix bugs without affecting the main codebase.
* Once the work is complete and tested, you can merge it back to the main branch.
* Example: You can work on a new feature while the main app stays stable.

### 4️⃣ ****Backup – Keep Your Work Safe****

* Git keeps a history of your work, so if your local files are lost or corrupted, you can recover them from the remote repository (like GitHub or GitLab).

### 5️⃣ ****Documentation – See What, Who, and Why****

* Git logs show you:
* **What** was changed
* **Who** changed it
* **Why** it was changed (through commit messages)
* This makes debugging and understanding code easier.

### 6️⃣ ****Integration with Tools****

* Git integrates with popular platforms like:
* GitHub, GitLab, Bitbucket → for sharing and managing code
* CI/CD pipelines → for automated testing and deployment
* This helps teams deliver high-quality code faster.

### ✅ Example Use Cases:

✔ A developer working alone – uses Git to keep track of their own changes.  
✔ A team working on a project – uses Git to coordinate and avoid conflicts.  
✔ Open-source projects – Git makes it easy for developers from all over the world to contribute.

### How to use Git?

* Install Git on your computer.
* Initialize a project with git init.
* Add files with git add.
* Save changes with git commit.
* Push and pull changes to/from remote repositories.

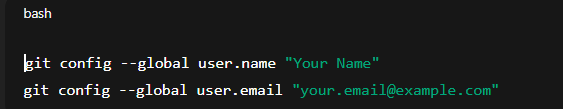
## ✅ ****Basic Git Setup and Workflow****

### 1️⃣ Install Git

* Download and install Git from [https://git-scm.com/](https://git-scm.com/" \t "_new).
* After installation, open the terminal (or Git Bash on Windows).

### 2️⃣ Configure Git

* Set your name and email (used in commits):



* You can check your configuration with:

git config --list

### 3️⃣ Create a New Repository

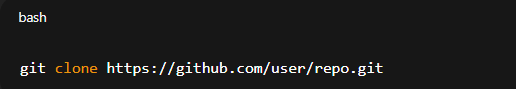
* A repository is where Git tracks changes.

#### Option 1: Create a new folder and initialize it with Git



* This starts tracking this folder.

#### Option 2: Clone an existing repository



* This downloads the repo to your local machine.

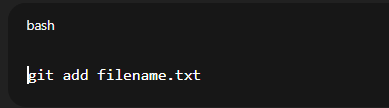
### 4.Check the Status

* Check which files are new, modified, or staged:

git status

### 5️⃣ Stage Files (Add them to be committed)

Add one file:



* Add all files in the directory:

git add .

### 6️⃣ Commit Files

* Save your changes with a message explaining what you've done:

git commit -m "Add initial project files"

### 7️⃣ View Commit History

git log

* Press q to exit.

### 8️⃣ Working with Branches

#### Check current branch:

git branch

#### Create a new branch:

git branch new-feature

#### Switch to the new branch:

git checkout new-feature

* Or create and switch in one step:

git checkout -b new-feature

### 9️⃣ Merge Branches

* After finishing work on a branch, you can merge it back into the main branch.

git checkout main

git merge new-feature

### 10️⃣ Undo Changes

#### Unstage a file:

git reset filename.txt

#### Discard changes to a file:

git checkout -- filename.txt

#### Undo last commit (keep changes staged):

git reset --soft HEAD~1

### 11️⃣ Working with Remote Repository (like GitHub)

#### Add a remote URL:

git remote add origin https://github.com/user/repo.git

#### Push your commits to the remote:

git push -u origin main

#### Pull changes from remote:

git pull origin main

## ✅ ****Summary Workflow****

git init → Initialize a new repository.

git status → See changes.

git add . → Stage changes.

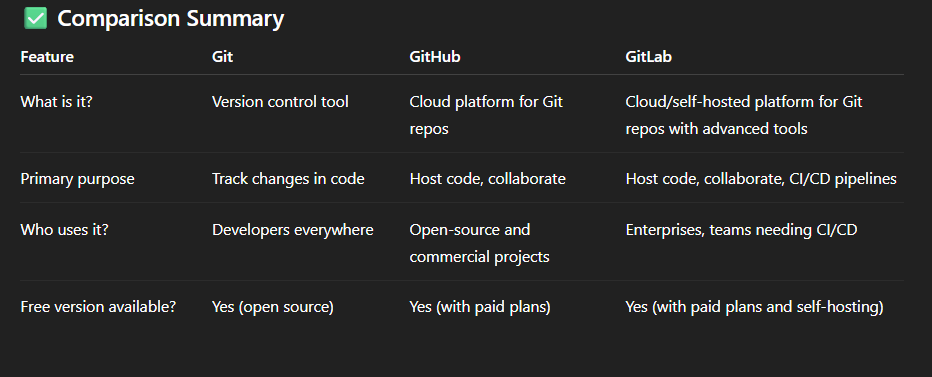
git commit -m "message" → Save changes.

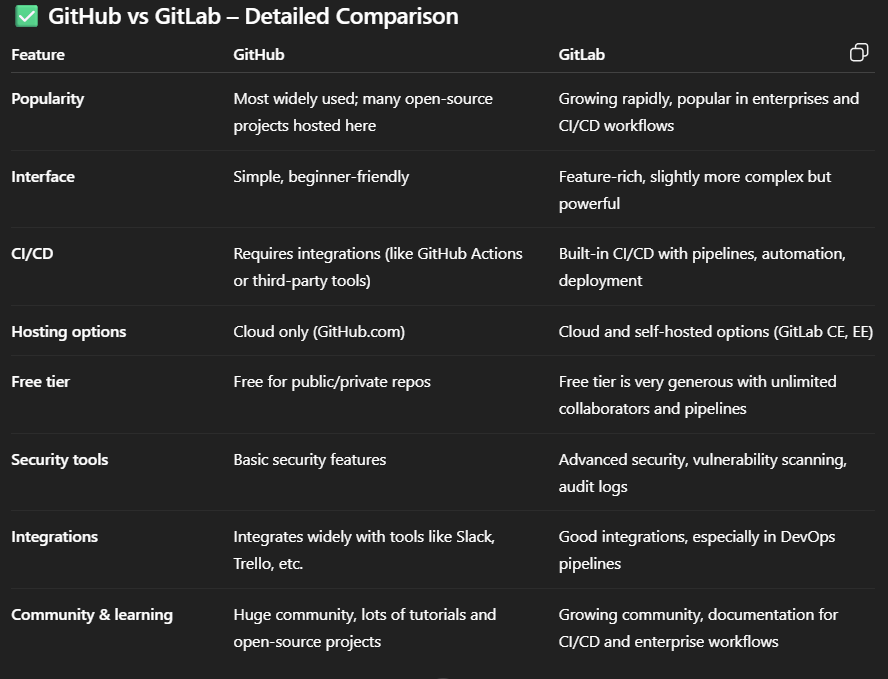
git push → Send changes to remote.

git pull → Update local copy from remote.

git branch → Manage branches.

git merge → Combine changes from branches.





# 📌 ****Basic Git Questions****

1. **What is Git?**  
   → Git is a **distributed version control system** used to track changes in code and collaborate with multiple developers.
2. **What is the difference between Git and GitHub?**

Git → Tool installed on your machine (VCS).

GitHub → Cloud platform to host Git repositories.

· **Git**: A **version control system (VCS)** → used to track changes in your code.

· **GitHub**: A **remote cloud platform** for hosting Git repositories → makes collaboration, sharing, and automation easier.

1. **What is a repository in Git?**  
   → A project folder tracked by Git, containing .git hidden folder with commit history.
2. **How do you check Git version?**  
   → git --version
3. **What is the difference between local and remote repositories?**

Local repo → On your machine

Remote repo → Hosted on GitHub, GitLab, Bitbucket, etc.

1. **What is the difference between** git clone **and** git fork**?**

Clone → Copies repo to local machine.

Fork → Creates a copy of someone else’s repo under your GitHub account.

**Essential Git Commands**

1. **How do you initialize a Git repository**

→ git init

1. **How do you check the status of your repo?**  
   → git status
2. **What’s the difference between** git add . **and** git commit**?**

git add . → Stages files

git commit → Saves snapshot with a message

1. **How do you push code to GitHub?**

git add .

git commit -m "message"

git push origin main

1. **How do you pull latest changes from GitHub?**  
   → git pull origin main
2. **What is the difference between** git fetch **and** git pull**?**

git fetch → Downloads changes (does not merge).

git pull → Downloads + merges changes.

📌 **Branching & Merging**

1. **What is a branch in Git?**  
   → A pointer to a specific commit, allows working on features in isolation.
2. **How do you create and switch to a branch?**  
   → git checkout -b feature-branch
3. **What is the difference between** git merge **and** git rebase**?**

Merge → Keeps history as is, creates a new merge commit.

Rebase → Rewrites history, applies commits on top of another branch.

1. **How do you delete a branch?**

Local: git branch -d branch-name

Remote: git push origin --delete branch-name

1. **What is a conflict in Git? How do you resolve it?**  
   → A conflict happens when two developers modify the same line in a file.  
   → Resolve manually in the file, then git add + git commit.

**Advanced Git / GitHub**

1. **What is the difference between Git and SVN?**

Git → Distributed, every developer has full history.

SVN → Centralized.

1. **What are Git hooks?**  
   → Scripts that run automatically at certain events (like before commit or push).
2. **What is the difference between** git reset**,** git revert**, and** git checkout**?**

reset → Moves HEAD to a previous commit (can discard changes).

revert → Creates a new commit that undoes changes.

checkout → Switches branches or restores files.

1. **What is the difference between** git stash **and** git commit**?**

stash → Saves changes temporarily (not in history).

commit → Saves permanently in history.

1. **What is a pull request (PR)?**  
   → A request to merge changes from one branch into another in GitHub.
2. **How do you fork and contribute to an open-source project?**

* Fork the repo
* Clone locally
* Create a branch
* Commit changes
* Push to your fork
* Open Pull Request on GitHub

1. **What is GitHub Actions?**  
   → A CI/CD service to automate testing, building, and deployment.
2. **What are tags in Git?**  
   → A snapshot for marking releases (e.g., v1.0).  
   Command: git tag v1.0 and git push origin v1.0.

📌 **Scenario-Based Questions**

1. What would you do if you accidentally committed a password or secret in Git?

Remove file → git rm --cached filename

Use git filter-branch or git filter-repo to rewrite history.

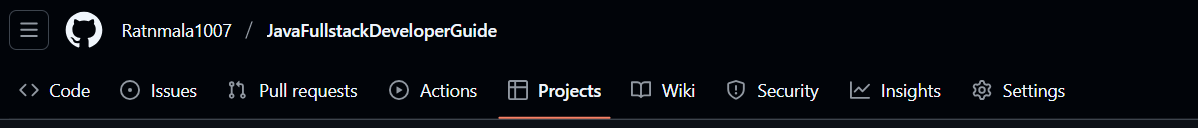
1. How would you recover a deleted branch?

Find commit hash → git reflog

Checkout → git checkout -b branch-name commit-hash

1. How do you handle large files in GitHub?  
   → Use **Git LFS (Large File Storage)**.
2. If your git push fails due to conflicts, what would you do?

* git pull origin main
* Resolve conflicts
* git push origin main
* Difference between SSH and HTTPS in GitHub? Which is better?
* HTTPS → Easy, but needs credentials.
* SSH → More secure, uses keys.  
  → SSH is generally preferred.
* **GitHub repository navigation bar**



Here’s what each of those **tags (tabs)** means 👇

### ****1️⃣ Code****

* This is the **main tab** where your project’s files live.
* You can:
* View your folder structure, source code, notes, etc.
* Upload new files or edit existing ones.
* Clone or download the repository.

💡 Think of this as your **file explorer** for the repo.

### ****2️⃣ Issues****

* Used for **bug tracking or feature requests**.
* Anyone (or just you, if it’s private) can create an issue to report a problem or suggest improvement.

💡 Example:  
“Need to fix broken image link in README.md.”

### ****3️⃣ Pull Requests****

* Used when multiple people work together.
* A **pull request (PR)** is a proposal to merge code from one branch into another.

💡 Example:  
If you make changes in a dev branch, you create a PR to merge into main.

### ****4️⃣ Actions****

* Used for **automation and CI/CD (Continuous Integration/Deployment)**.
* You can create workflows that automatically:
* Run tests when code is pushed
* Deploy your app to a server
* Format code, check syntax, etc.

💡 Example:  
Automatically build and test your Spring Boot project after every commit.

### ****5️⃣ Projects****

* A **Kanban-style board** to manage tasks, to-dos, and progress.
* You can create columns like “To Do,” “In Progress,” “Done” and move cards between them.

💡 Example:  
You could track your learning topics for Java Fullstack:

To Do: Spring Boot SecurityIn Progress: HibernateDone: Core Java

### ****6️⃣ Wiki****

* Used for creating **detailed documentation** or guides.
* Similar to Markdown files, but separate from your code.
* You can organize pages and subpages like a mini website.

💡 Example:  
You could write a “Java Fullstack Developer Guide” wiki with topics for each framework.

### ****7️⃣ Security****

* Shows **security alerts** about vulnerabilities in your code or dependencies.
* You can enable Dependabot to automatically detect and fix insecure dependencies.

### ****8️⃣ Insights****

* Provides **analytics** and activity stats\*\* about your repository:
* Number of commits
* Contributors
* Traffic (views and clones)
* Community engagement

💡 Great for tracking repo growth and contribution history.

### ****9️⃣ Settings****

* Used to manage **everything about your repo**:
* Rename repository
* Change visibility (Public/Private)
* Manage branches
* Enable/disable features (Issues, Wiki, Projects, etc.)
* Delete the repo

### ****Summary Table****

| **Tab** | **Purpose** |
| --- | --- |
| **Code** | View and edit repository files |
| **Issues** | Track bugs, tasks, ideas |
| **Pull Requests** | Propose and review code changes |
| **Actions** | Automate workflows (build, deploy, test) |
| **Projects** | Manage tasks in Kanban board style |
| **Wiki** | Create documentation pages |
| **Security** | Manage vulnerabilities and dependencies |
| **Insights** | View repo statistics and activity |
| **Settings** | Configure and manage repo options |