Git & GitHub – Complete Documentation

▼ 1. Basic Concepts

▼ What is Software?

- Software is a set of instructions (code) given to a computer to perform a task.
- **Examples**:
    - WhatsApp
    - Instagram
    - Website
    - Mobile App
- These software applications are created using **source code**.

▼ What is Source Code?

- A source code is the code written by developers using programming languages like:
    - HTML
    - CSS
    - JavaScript
    - Java
    - Python
    - etc.

▼ What problem occurs while writing code?

- When developers work on code:
    - Code changes every day
    - New features are added

    - Bugs are fixed
    - Many developers work together
- This creates problems like:
    - ❌Old Code lost
    - ❌New code overwritten
    - ❌No history
    - ❌Hard to track changes

👉 To solve this problem, we use a **Version Control System**.

▼ 2. Introduction to Version Control

▼ What is a `Version`?

- A version means a saved copy of your project at a specific time.
- Example:
    - Project_v1
    - Project_v2
    - Project_v3
- Each version stores a snapshot of the project at a specific point in time..

▼ What is a `Version Control System` (VCS)?

- A **Version Control System** is a tool that helps us:
    - Save different versions of code
    - Track changes
    - Go back to the old code
    - Work with team members
- Simply → It keeps a record of our code changes.

▼ Types of Version Control Systems

- There are mainly 3 types of version control systems available:

1. **Local VCS**
    - Code stored only in one system
    - Not safe
2. **Centralized VCS**
    - One main server
    - Internet required
3. **Distributed VCS**
    - Every developer has a full copy
    - Most secure

- Here, 👉 **Git is Distributed VCS.**

▼ 3. What is `Git`?

- Git is a software tool.
- **Git** is a **distributed version control system** is used to:
    - Track file changes
    - Maintain project history
    - Work with multiple developers
    - Manage source code efficiently

▼ Who created Git?

- Git was created by `Linus Torvalds` in **2005**. (Creator of Linux)

▼ Why Git is important?

- Because Git helps to:
    - Recover deleted code
    - Track who changed what
    - Avoid code loss
    - Work offline
    - Maintain clean history

▼ How Git Works?

- Git works in 3 main areas:

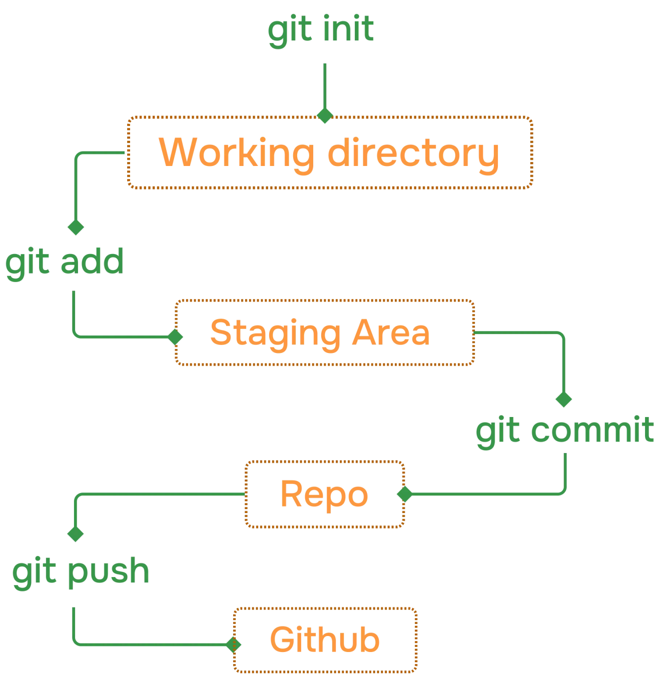
  1. **Working Directory**
     - Actual folder where you write code
     - Actual project files
     - Example:

     ```
     index.html
     style.css
     app.js
     ```

  2. **Staging Area**
     - Temporary Area
     - Files kept before final saving
     - Like “review area”
     - Meaning: *I want to save these changes.*

  3. **Local Repository**
     - Permanent storage
     - Code is officially saved
     - Contains history

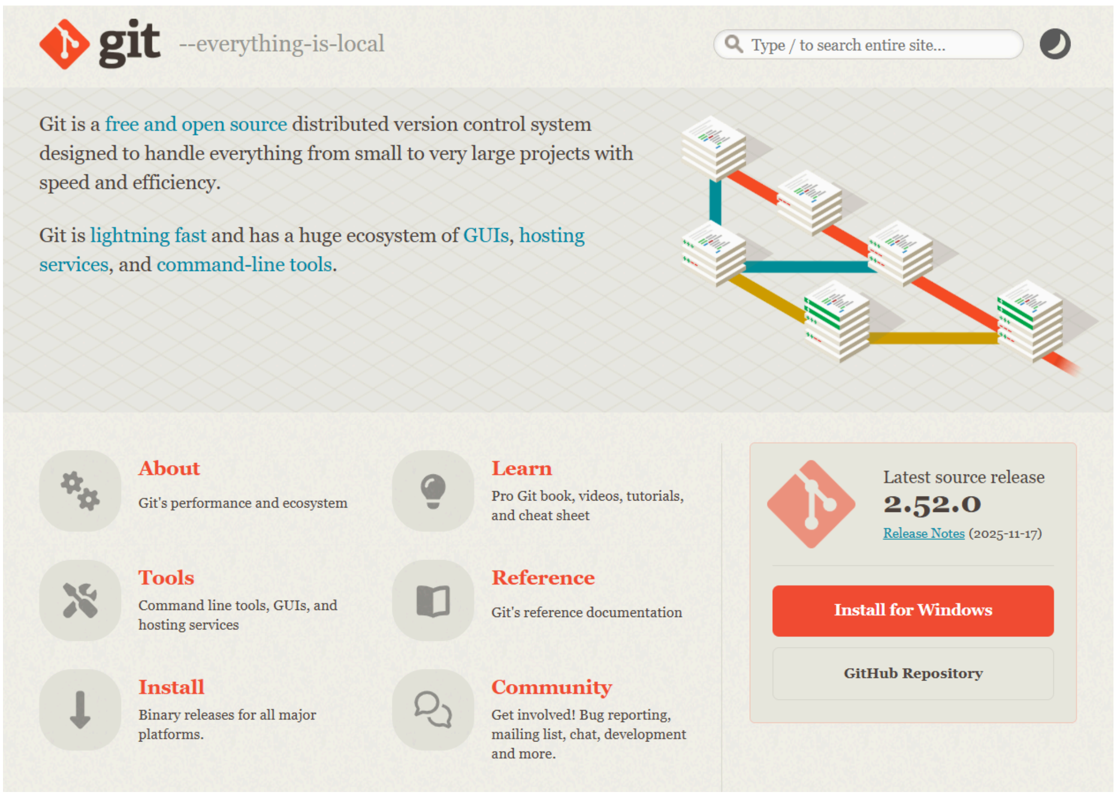
▼ Git Workflow



▼ 4. Installation and Setup (Git)

▼ Install Git

- Download from: https://git-scm.com



git --everything-is-local

Type / to search entire site...

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is lightning fast and has a huge ecosystem of GUIs, hosting services, and command-line tools.

About Git's performance and ecosystem

Learn Pro Git book, videos, tutorials, and cheat sheet

Tools Command line tools, GUIs, and hosting services

Reference Git's reference documentation

Install Binary releases for all major platforms.

Community Get involved! Bug reporting, mailing list, chat, development and more.

Latest source release
2.52.0
Release Notes (2025-11-17)

Install for Windows

GitHub Repository

- Install and verify:

```
git --version
```

▼ Git Configuration

Tell Git who you are:

```
git config --global user.email "yourmail@gmail.com"
git config --global user.name "Your Name"
```

Git uses this information in commits.

Check config:

```
git config --list
```

▼ 5. Git Repository

▼ What is a repository?

- A repository (repo) is a place (folder) where Git Stores:
    - Project files
    - Git tracks changes
    - Branch Information

▼ Initialize Repository / Create Repository

- Command:

```
git init
```

- Creates: This creates hidden folder:

```
.git/
```

- This folder controls Git.

▼ 6. Basic Git Commands

▼ 1. Check Status

- Command:

```
git status
```

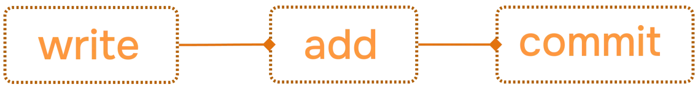
- Shows:
  - Modified files
  - Staged files
  - Untracked files

▼ 2. Add Files

```
git add <file1.js> <file2.js>
git add .
```

▼ 3. Commit changes

- Commit = snapshot of code.



```
git commit -m "message"
```

▼ 4. View commit history

```
git log
git log --oneline
```

- Shows:
  - Commit history

    - Date
    - Author

▼ Git File States

State	Meaning
Untracked (U)	New file
Modified (M)	File changed
Staged	Ready to commit
Committed	Saved in repo

▼ Undo Commands (Very Important)

▼ Undo staged file

```
git restore --staged file.js
```

▼ Undo file changes

```
git restore file.js
```

▼ Change last commit message

```
git commit --amend
```

▼ 7. Branching in Git

▼ What is `Branch`?

- A **branch** is an independent line of development.
- OR
- A **branch** is a separate copy of code.
- Main branch: (Default Branch)

```
main
```

- The `main` branch usually contains **stable / production-ready code**.
- Development happens in feature branches and is merged into `main`.

▼ Create branch

```
git branch <branch-name>
```

▼ Switch branch

```
git checkout <branch-name>
```

▼ Create & switch

```
git checkout -b <branch-name>
```

▼ 8. Git Merging

▼ What is `Merge`?

- Combine changes from one branch into another branch.

```
git merge <branch-name>
```

▼ Merge Conflicts

Occurs when:

- Same file
- Same line
- Different changes

Git will ask developer to resolve manually.

▼ Rename a branch

- You can rename a branch using the following command:

```
git branch -m <old-branch-name> <new-branch-name>
```

▼ Delete a branch

- You can delete a branch using the following command:

```
git branch -d <branch-name>
```

▼ List all branches

- You can list all branches using the following command:

```
git branch
```

▼ 9. What is `GitHub`?

- **GitHub** is a **cloud-based platform** used to:
    - Store Git repositories online
    - Share code
    - Work with teams
    - Manage projects
    - Perform CI/CD
- Basically GitHub stores the code online.

▼ 10. Connecting Git to GitHub

Steps:

1. Create repository on GitHub
2. Copy repository URL
3. Add remote

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

- Check:

```
git remote -v
```

11. Push & Pull

What is `push`?

- The **push** means send your code to the remote repository.
- When you write code and commit it **locally**, no one else can see it until you **push** it.
- Command:

```
git push -u origin main

//OR

git push
```

- Meaning:
  - `origin` → remote repository
  - `main` → branch name

Example:

You added a new feature:

```
git add .
git commit -m "Added login page"
git push origin main
```

✅ Now your code is visible on GitHub.

What is `pull`?

- The **pull** means get a latest code from remote repository.
- If someone else updated the project (or you updated from another system), you need to **pull** the changes.

- Command:

```
git pull origin main
```

- What happens internally?

```
git pull = git fetch + git merge
```

- **fetch** → downloads changes
- **merge** → merges into your branch

▼ 12. Clone Repository

- `Clone` ⇒ Copy the entire project from remote to your system.
- When you clone a repository, Git downloads:
  - All source code
  - All commits history
  - All branches
  - Remote connection (origin)
- from **GitHub** → **your local computer**.
- Git Clone Flow:

```
Remote Repository (GitHub)
           ↓
     git clone
           ↓
Local Repository (Your system)
```

- Command:

```
git clone <repository-url>
```

- Example:

```
git clone https://github.com/user/project.git
```

After this:

- A new folder is created
- Git is already initialized
- `origin` remote is already connected

✅ You can directly start working.

▼ 13. Fork & Pull Request

▼ What is `Fork`?

- Fork means create a copy of someone else's repository into your own GitHub account.
- You **cannot directly push** to someone else's repository.
- So GitHub gives you **Fork**.

▼ What happens when you fork?

```
Original Repo (Owner)
          ↓
        Fork
          ↓
Your GitHub Account (Your Repo)
```

✅ You now have:

- Your own copy
- Full permission
- Same code & history

▼ Example:

- Repository:

```
github.com/reactjs/react
```

- You click **Fork →**
- Now you get:

```
github.com/your-username/react
```

You can:

- clone it
- edit code
- commit
- push changes

▼ What is `Pull Request` (PR)?

- A Pull Request means request repository owner to merge your changes.
- After making changes in your forked repo, you **cannot merge directly** into the original repo.
- So you send a **Pull Request** (PR).
- Flow:

```
Your Forked Repo
       ↓
   Pull Request
       ↓
Original Repository
```

- You are saying: *Please review my changes and merge them into your project.*

▼ Complete Fork + PR Workflow

```
1️⃣ Fork repository
2️⃣ Clone fork to local system
3️⃣ Create new branch
4️⃣ Make changes
```

```
5️⃣ Commit changes
6️⃣ Push to your fork
7️⃣ Create Pull Request
```

Used heavily in:

- Open-source projects

▼ 14. Git Ignore

▼ What is `.gitignore`?

- **.gitignore** is a file used to tell Git which files or folders **should NOT be tracked or pushed** to the repository.
- In simple word: ***Git, please ignore these files***.

▼ Why do we need `.gitignore`?

Some files should **never go to GitHub**, like:

- ❌ `node_modules/`
- ❌ `.env` (API keys, passwords)
- ❌ build files
- ❌ system files
- ❌ logs

These files:

- are large
- are auto-generated
- may contain secrets

▼ Example `.gitignore` file

```
node_modules/
.env
dist/
build/
*.log
```

Meaning:

- `node_modules/` → ignore entire folder
- `.env` → ignore environment file
- `.log` → ignore all log files

▼ Important Rule (Very common mistake)

- `.gitignore` works **only for untracked files**
- If a file is already tracked:

```
git add .env
git commit -m "added env"
```

- Then adding `.env` in `.gitignore` will NOT work ❌
- **Solution**:

```
git rm --cached .env
```

- Then commit again.

▼ 15. Real-Time Git Workflow (Company Level)

```
1. Clone repo
2. Create branch
3. Write code
4. Commit changes
5. Push branch
6. Create Pull Request
7. Code review
8. Merge to main
9. Deploy
```

▼ Git & GitHub – General Interview Questions with Answers

1. What is Git?

Answer:

Git is a version control tool used to track changes in source code and manage project history.

---

2. Why do we use Git?

Answer:

We use Git to:

- Track code changes
- Save project versions
- Recover old code
- Work with multiple developers

---

3. What is Version Control System?

Answer:

A Version Control System is a system that keeps record of changes made to files over time.

---

4. What is GitHub?

Answer:

GitHub is an online platform used to store Git repositories and collaborate with team members.

---

5. Difference between Git and GitHub?

Git	GitHub
Tool	Website

Git	GitHub
Works locally	Works online
Manages versions	Stores repositories

6. What is a repository?

Answer:

A repository is a folder where Git stores project files and their history.

7. What is a commit?

Answer:

A commit is a snapshot of the project that saves changes permanently in Git.

8. What is commit message?

Answer:

A commit message describes what changes were made in the commit.

Example:

```
git commit -m"Login page added"
```

9. What is staging area?

Answer:

The staging area is a temporary place where files are kept before committing.

20. What is working directory?

Answer:

The working directory is the folder where we write and modify code.

11. What is git add?

Answer:

`git add` moves files from working directory to staging area.

12. What is git status?

Answer:

`git status` shows the current state of files like modified, staged, or untracked.

13. What is git log?

Answer:

`git log` shows the commit history of the project.

14. What is branch?

Answer:

A branch is a separate line of development used to work on new features safely.

15. What is main branch?

Answer:

The main branch is the default branch that contains stable production code.

16. What is merge?

Answer:

Merge means combining code from one branch into another branch.

17. What is merge conflict?

Answer:

Merge conflict occurs when two developers change the same line of code differently.

18. What is HEAD in Git?

Answer:

HEAD points to the current branch or latest commit you are working on.

19. What is git clone?

Answer:

`git clone` downloads a complete repository from GitHub to local system.

20. What is git push?

Answer:

`git push` uploads local commits to GitHub.

21. What is git pull?

Answer:

`git pull` downloads the latest code from GitHub to local system.

22. What is git fetch?

Answer:

`git fetch` only downloads changes but does not merge them.

23. Difference between git pull and git fetch?

git pull	git fetch
Downloads + merges	Only downloads
Updates working code	Does not update code

24. What is remote repository?

Answer:

A remote repository is a repository stored on GitHub.

25. What is origin?

Answer:

Origin is the default name of the remote GitHub repository.

---

26. What is .gitignore?

Answer:

`.gitignore` is a file used to tell Git which files should not be tracked.

---

27. Why node_modules is ignored?

Answer:

Because it is very large and can be installed again using npm install.

---

28. What is fork?

Answer:

Fork creates a copy of someone else’s repository in our GitHub account.

---

29. What is pull request?

Answer:

A pull request is a request sent to the repository owner to merge our changes.

---

30. What is stash?

Answer:

Git stash temporarily saves changes without committing them.

---

31. What is rebase?

Answer:

Rebase is used to move or combine commits to maintain clean history.

---

32. What is tag in Git?

Answer:

Tags are used to mark specific versions like v1.0 or v2.0.

33. Can we use Git without GitHub?

Answer:

Yes, Git can be used locally without GitHub.

34. Is Git required for frontend developers?

Answer:

Yes, Git is required for all developers including frontend, backend, and full-stack.

35. What happens if we don’t use Git?

Answer:

We may lose code, cannot track changes, and teamwork becomes difficult.