Sayapatri Group
Society, IT & Technology

Git and Github

1. Why Git Came? What Problem Did It Solve?

Problem Before Git

Before Git existed, developers used to save files manually in folders like:

- project_v1
- project_final
- project_final_final
- project_new_really_final

This caused major issues:

1. **No proper version history**
   No easy way to see what was changed, when, and by whom.

2. **Collisions when working in teams**
   Two people editing the same file created confusion and overwriting.

3. **No backup of code**
   If your computer crashed, the project was gone.

4. **No way to go back safely**
   If you made a mistake, undoing old versions was extremely difficult.

Why Git Was Created

Git was built to:

- Track every version of your project
- Let many people work together without overwriting
- Allow safe experiments using branches
- Provide full history and rollback options
- Work offline and sync later

2. What Is Git?

Git is a **version control system**.

Meaning:
It saves the history of your project so you can go to any version anytime.

Git works on your **local computer**.

Alternatives to Git

Some version control systems before or alongside Git:

- **SVN (Subversion)** – Centralized version control.
- **Mercurial** – Distributed version control like Git.
- **CVS** – Very old, rarely used now.

Why Git won over others?

- **Distributed system**: Every developer has the full repository.
- **Speed**: Operations are local and fast.
- **Branching & merging**: Easy, lightweight branches.
- **Community & support**: Huge ecosystem and integration with GitHub, GitLab, Bitbucket.

3. What Is GitHub?

GitHub is a **website** where you store and manage your Git repositories online.

https://github.com/

Alternatives of github : GitLab, Bitbucket

Git = tool on your computer
GitHub = online storage + collaboration platform

GitHub helps you:

- Store projects online
- Share code
- Collaborate with others
- Show projects to employers
- Work from any device

4. What Is a Repository?

A **repository (repo)** is a project folder tracked by Git.

Inside a repo, Git stores:

- All files
- All versions
- All commit history

A repo can be:

- Local (your computer)
- Remote (GitHub)

Why GitHub Came? What Problem Did It Solve?

Git allowed version control **locally**, but teams needed to:

- Share code easily
- Upload code to the cloud
- Collaborate on the same project
- Review and merge changes

GitHub solved these by providing:

- Remote repositories
- Pull Requests
- Issues
- Collaboration tools

Task: **Make a new Github Repository**

Git Configuration

Step 1: Install Git

- Windows: Use Git for Windows
- Mac: `brew install git` (if Homebrew is installed)
- Linux: `sudo apt install git` (Debian/Ubuntu)

After installation, check version:

git --version

This confirms Git is installed.

Step 2: Configure User Details

Git needs your identity to track changes.

git config user.name "Your Name" (user name for current repo only ,useful when you have too many account working in different repo)

Recommended method

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

Explanation:

- `--global` → applies to all repositories on this machine.
- These details show up in commit history.

Check your configuration:

git config --list

You will see something like:

user.name=Your Name
user.email=you@example.com

Start tracking with git

Step 1: Initialize a Repository
git init

- Creates a hidden `.git` folder in your project directory.
- This is the **start of Git tracking**.
- **ls -a** *(to view the hidden git file)*

Concept:

1. `.git` stores all commit history, branches, and configuration for this repo.
2. After this, your files are **untracked** until you add them.

2. Check Status
git status

- Shows the **state of files**:
    - Untracked → files not yet staged.
    - Changes not staged → modified files.
- Helps you see what will be committed.

GIT FILE STATES

1. Untracked

- File exists in working dir but Git doesn't track it.
- Status short form: **U file.txt**

2. Tracked

Tracked files can be in 3 states:

a) Unmodified

- Same as last commit.
- Status: *(none)*

b) Modified

- Changed in working directory.
- Status short form: **M file.txt** (left side in short status)

c) Staged

- Added to staging area using `git add`.
- Status short form: **A** or **M** on right side.

4. Commit Changes

git commit -m "Your commit message"

git commit -am "Your commit message"(Git Add and Commit in the Same Line)

- Moves staged changes to the **Local Repository**.
- Commit message explains what you changed.
- Each commit gets a **unique ID (hash)**.

5. View Commit History

git log
git log –oneline (compact form)

- Shows all commits in the repository.
- Useful to **track changes** and review history.

6. Branching

git branch new-branch
git checkout new-branch

- Branches allow **parallel work**.
- `main` branch is default.
  - git branch
  - git branch -M main
- Flow: `Local Repository → Branch` → work on features independently.

7. Merge Branches

git checkout main
git merge new-branch

- Integrates changes from another branch into **current branch**.
- Conflicts may occur if same lines are edited.

8. Remote Repository

git remote add origin <repo-url>
git push -u origin main

- `remote add` → connects local repo to GitHub (or other remote).
- `push` → sends local commits to remote repository.
- `-u` sets upstream branch for future pushes.

9. Pull Updates from Remote
git pull origin main

- Fetches latest changes and merges them into your branch.
- Keeps your local repo updated.

Flow Diagram

1. Local to Remote (Push)
[Working Directory] --git add--> [Staging Area] --git commit--> [Local Repository] --git push--> [Remote Repository]

2. Remote to Local (Pull / Fetch + Merge)
[Create Remote Repo] --git clone--> [Local Repo + Working Dir] --edit--> git add→[Staging Area] --git commit--> [Local Repo] --git push--> [Remote Repo]

How to Upload Local Project to GitHub

Why push code to GitHub?

- Online backup
- Access from anywhere
- Show projects to others
- Collaboration

Steps

Step 1: Create repo on GitHub
Give name → keep empty (no README).

Step 2: Connect local repo to GitHub

git remote add origin URL_of_your_repo

Step 3: Push code

git branch -M main
git push -u origin main

Cloning a Repository

Why clone?

To download someone's GitHub project onto your computer.

Step

git clone URL

This creates a copy of the repo locally.

Common Problems Beginners Face

1. Not committing before pushing

Fix:

```
git add .
git commit -m "message"
git push
git status
```

2. Repo rejects your push

Usually because branch names differ.
Fix:

```
git branch -M main
git push -u origin main
```