🚀 Git & GitHub Fundamentals

Master Version Control & Collaboration

PART 1: Repository Setup & Basic Commands

Step 1: Create a Folder & Initialize Repo

```
mkdir git-demo
cd git-demo
git init
```

Step 2: Create a File

```
echo "Hello Git" > file1.txt
```

Step 3: Check Status

```
git status
```

- **Step 4: Add File to Staging Area**
- `git add file1.txt`
-
- `git add .`
-

- **Step 5: Commit Changes**
- `git commit -m "Initial commit"`
-
- **Step 6: View Commit History**
- `git log`
-

(short version)

- `git log --oneline`

Open a Directory Using Git Bash / Terminal

- Step 1: Open Git Bash
- Right-click inside the folder → **Open Git Bash here**
- Step 2: Navigate to directory
- `cd path/to/your/directory`
- **Example:**
- `cd /c/Users/Navaneeth/Documents/project`

View File Content Using Git Command (Read-Only)

- If you just want to **see the content** of a file:
- **For text files (.java, .txt, .md, .json)**
- `git show HEAD:filename`
-
- **Example:**
- `git show HEAD:README.md`
-

Or simpler (not Git-specific):

- `cat filename`
- **Example:**
- `cat Student.java`

Open File in Default Editor (Git Bash)

- **Open file using OS default application**
- `start filename`
- **Example:**
- `start index.html`

Open File Using VS Code (Most Common)

- If VS Code is installed:
- Open a specific file
- `code filename`
- **Example:**
- `code StudentController.java`
- To Open entire folder
- `code .`

Open File Using Notepad (Windows)

- `notepad filename`
- **Example:**
- `notepad notes.txt`

Aspect	Git	GitHub
What it is	**Version Control Tool**	**Hosting Platform / Service**
Type	Software / Tool	Web-based platform
Purpose	Tracks code changes locally	Stores code online & enables collaboration
Works Offline?	✅ Yes	❌ No (needs internet)
Installation	Installed on your system	No installation (web-based)
Created By	Linus Torvalds	Microsoft (originally GitHub Inc.)
Used For	Commit, branch, merge, history	Pull requests, code review, CI/CD
Example	`git commit`	Create Pull Request

PART 2: Working with Remote Repositories (GitHub)

- **Step 1: Create Repository on GitHub**
- Login → New Repository
- Name: `git-demo`
- Do NOT add README initially
- **Step 2: Link Local Repo to Remote**
- `git remote add origin https://github.com/username/git-demo.git`
-
- To Check use the below:
- `git remote -v`

- **Step 3: Push Code to GitHub**
- `git branch -M main`
- `git push -u origin main`
-

- **Step 4: Pull Changes from Remote**
- `git pull origin main`
-

PART 3: Git Branching (VERY IMPORTANT)

- What is a Branch in Git?
- A **branch** is like a **separate working line** of your project.
- You can work on a new feature
- Without disturbing the main working code
- Once work is correct → merge it back

- **Real-World Analogy**

Imagine:

- **Main branch** = Final notebook submitted to teacher
- **Feature branch** = Rough notebook where you try new ideas
- You experiment in the rough notebook.
  Once everything is correct, you copy it into the final notebook.
- That's exactly what Git branching does.

Create a New Branch

- Command:
- `git branch feature-login`
- 🔍 **What this command does**
- Creates a new branch named `feature-login`
- It is copied from the **current branch**
- **No switching happens yet**
- You are still on the same branch after this command.
- 🧠 **Internally:**
- Git creates:
- A pointer called `feature-login`
- Pointing to the current commit

- 📌 **Important Note**
- Creating a branch ≠ moving to that branch.

Switch to Branch

- **Command:**
- `git checkout feature-login`
-
- 🔍 What this does
- Switches your working directory to `feature-login`
- Your HEAD now points to this branch
- Any changes you make now belong to `feature-login`

- **Real-World Analogy**
- You close the **final notebook**
  You open the rough notebook
  Now you write freely without fear.
- Verify current branch:
- `git branch`
-
- The active branch will have *

Create & Switch in ONE Command (Shortcut)

- Command:
- `git checkout -b feature-login`

What this does

- Creates the branch
- Immediately switches to it
- This is a **shortcut** for:
- `git branch feature-login`
- `git checkout feature-login`

Make Changes & Commit (Inside Feature Branch)

- **Command:**
- `echo "Login feature" >> file1.txt`
-

What this does

- Adds the text `Login feature` to `file1.txt`
- `>>` means **append**, not overwrite

- Check status:
- `git status`
-
- Stage changes:
- `git add .`
-
- Adds all modified files to staging
- Prepares them for commit

- Commit changes:
- `git commit -m "Added login feature"`
-
- 🔍 What happens here?
- Git creates a **new commit**
- Commit exists only in `feature-login`
- `main` branch is still unchanged

- 💡 Important Concept
- Branches are **independent timelines**.
- Main branch does NOT see these changes yet.

Merge Branch to Main

- Once the feature is tested and correct

- **Step 1:** Switch back to main branch
- `git checkout main`
-
- You are now back in the **final notebook**.

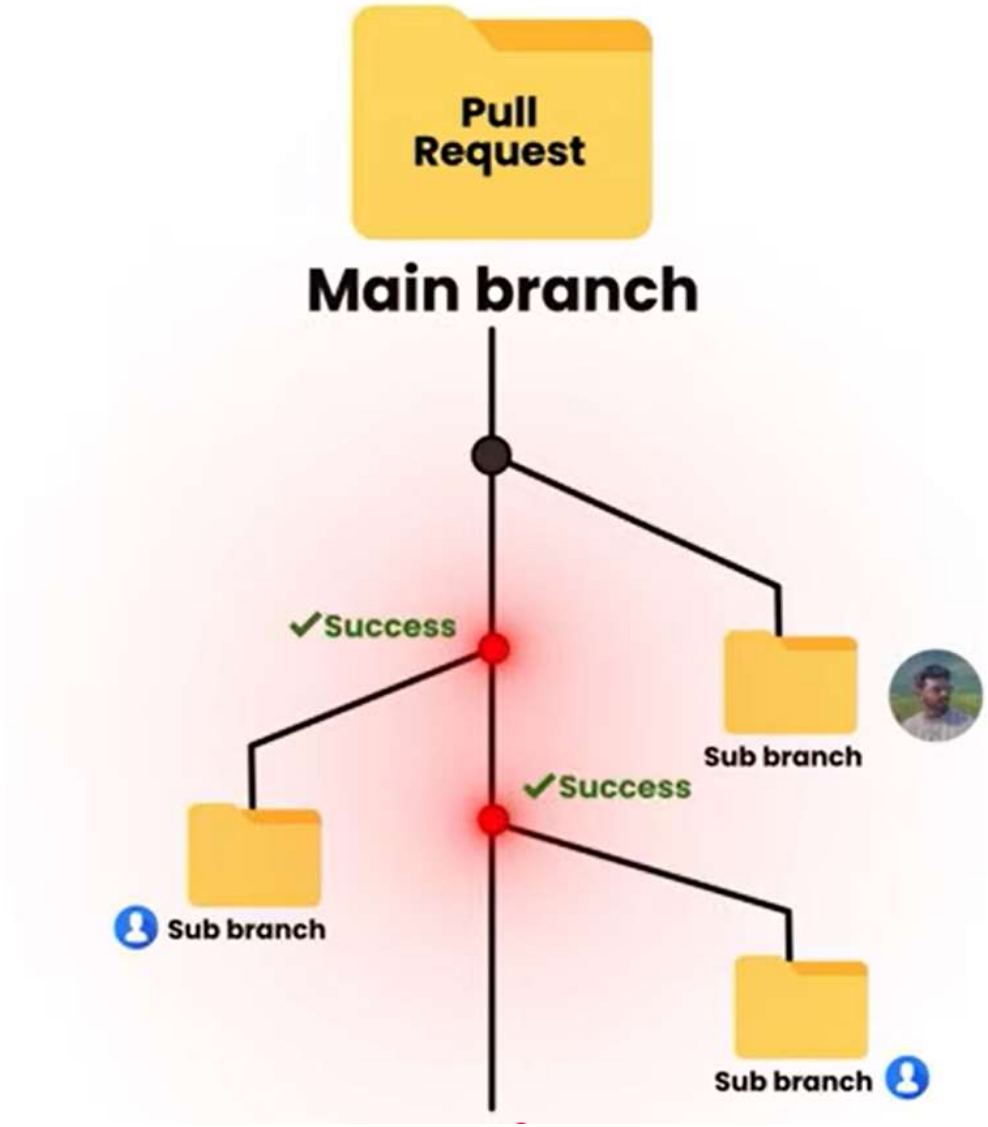
- **Step 2:** Merge feature branch
- `git merge feature-login`
-
- 🔍 What this does
- Takes changes from `feature-login`
- Adds them into `main`
- Creates a **merge commit** (if needed)

- 🧠 Real-World Analogy
- You copy all correct content from rough notebook
  Into the final notebook
  Now final notebook has new feature.

What Happens Internally During Merge?

- **Case 1: Simple merge (Fast-forward)**
- No conflicts
- Git moves main pointer forward
- **Case 2: Conflict merge**
- Same file changed in both branches
- Git asks you to resolve conflicts manually

Visual Representation



Summary

Command	Meaning
`git branch feature-login`	Create branch
`git checkout feature-login`	Switch branch
`git checkout -b feature-login`	Create + switch
`git commit`	Save changes
`git merge`	Combine branches

PART 6: GitHub Workflow (Collaboration)

Why Do We Need a GitHub Workflow?

- In real projects:
- Multiple developers work on the **same codebase**
- Everyone cannot directly change the **main branch**
- Changes must be **reviewed, discussed, and approved**
- 👉 GitHub workflow ensures:
- Safe collaboration
- Code quality
- No accidental breakage

🧠 Real-World Analogy (Very Important)

Imagine:

- **Main repository** = University syllabus document
- **Contributors** = Faculty members
- Everyone cannot directly edit the final syllabus
- They suggest changes → review → approval → merge
- GitHub workflow works the **same way**.

Fork & Clone (First Step of Collaboration)

- 🔷 What is Forking?
- **Forking** means:
- Creating **your own copy** of someone else's repository
- Stored under **your GitHub account**
- Original repo remains unchanged

Why Fork?

- Because:
- You don't have write permission to original repo
- You need a safe space to experiment

- 🍴 How to Fork (On GitHub)
- Open the repository on GitHub
- Click **Fork** (top-right)
- GitHub creates a copy under:
- https://github.com/your-username/repo-name

🏠 Real-World Analogy

Teacher gives notes (read-only).
You photocopy them and write your own additions.
That photocopy = **Fork**.

Clone the Forked Repository

- Command:
- `git clone https://github.com/username/repo-name.git`
-
- 🔍 **What this does**
- Downloads the repository from GitHub
- Creates a local copy on your laptop
- Sets `origin` → your forked repo

- 🧠 **Behind the scenes:**
- Remote called `origin` is created
- Code is available locally for editing
- 🏗️ **Analogy**
- You bring the photocopied notes home to study and write on.

Create Branch & Push Changes

- Why Create a Branch?
- Never work directly on `main`
- Feature development should be isolated
- Easier review and rollback
- **Step 1: Create and switch branch**
- `git checkout -b new-feature`
-
- ✓ Creates a branch
  ✓ Switches to it
  ✓ Safe workspace

- **Step 2: Make changes & commit**
- `git add .`
- `git commit -m "Added new feature"`
- **Step 3: Push branch to GitHub**
- `git push origin new-feature`

- 🔍 What happens?
- Your branch is uploaded to **your fork**
- GitHub now knows this branch exists
- Ready for Pull Request

- **Analogy**
- You wrote improvements in your photocopy
  Now you show it to the teacher.

Pull Request (PR) – The Heart of Collaboration

- ❓ What is a Pull Request?
- A **Pull Request (PR)** means:
- “Please pull my changes into your main repository.”
- 📌 Important Clarification
- You are NOT directly merging
- You are **requesting** the project owner to merge

- 🧭 How to Create PR on GitHub
- Go to your forked repo
- Click **Compare & Pull Request**
  (or New Pull Request)
- Select:
  - Base repo → original repo
  - Compare repo → your fork & branch
- Add:
  - Title
  - Description
- Click **Create Pull Request**

- 🔹 Real-World Analogy

You submit:

- Assignment
- With explanation
  Teacher reviews and decides.

Code Review Process

- Once PR is submitted, reviewers examine your code.
- ✓ **Reviewer Can:**
- 1 Comment
- Suggest improvements
- Ask questions
- Point issues
- **Example**:
- “Please rename this variable.

- 2 **Request Changes**
- PR cannot be merged yet
- You must fix issues and push again
- 3 **Approve**
- Code is correct
- Ready to merge

- Important
- When you **fix code**:
- Commit changes
- Push to **same branch**
- PR updates automatically

Approve & Merge

- Once approved:
- ✓ Maintainer clicks **Merge Pull Request**
  ✓ Changes are merged into main branch
  ✓ Feature becomes part of project

- 🏁 **Final Result**
- Original repository updated
- Your contribution accepted
- You become a contributor 🎉

Complete GitHub Collaboration Flow (End-to-End)

Fork repo

↓

Clone fork

↓

Create feature branch

↓

Commit changes

↓

Push branch

↓

Create Pull Request

↓

Code Review

↓

Approve & Merge

PRACTICE ACTIVITY

Mini Assignment

- Create repo
- Create 2 branches
- Introduce conflict
- Resolve conflict
- Raise Pull Request