Git

Chitram Dasgupta

July 16, 2025

Contents

1 Why Use Git?

- You might have saved project files with names with `project.docx`, `project_version2.docx`, `project_final.docx`, `project_final_final.docx`.
- Without a system to manage changes, this quickly becomes chaotic
- Git addresses this problem.
- Git is a **Version Control System**. This means that it helps developers track changes to their codebase over time.
- Git allows us to:

  1. Track who made what changes and when, which is important for debugging and understanding the history of a project.
  2. Collaborate: Multiple developers can simultaneously work on the same project without interfering with each other's work.

2 The Core Concepts

2.1 Repository

- This is the project folder that git is tracking.
- Git will not automatically track changes. You have to specify that you want to version-control a project with git and also tell it to track specific files.

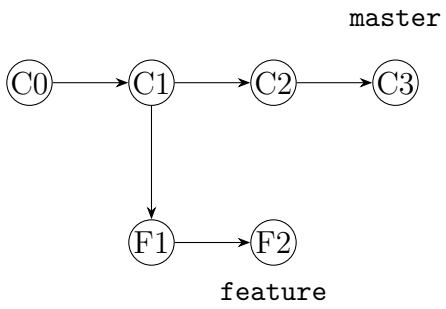
2.2 Commit

- A commit is a snapshot of your project files at a specific point in time.
- This is similar to saving your progress in a videogame. The next time you play the game, you can start from the previous checkpoint instead of starting over.

2.3 The Three States of a File

1. Working Directory: This is where you are currently editing your files.
2. Staging Area (or Index): This is the waiting area where we place the files you want to include in your next commit. This allows you to be selective about which files you want to include in your next commit.
3. Committed: The changes to the file are safely stored in the repository.

2.4 Branch



Here you have two branches: `master` and `feature`, which can evolve independently.

3 Setup

1. Install **git** on your system.
2. Configure your name and email:

```
git config --global user.name "Your Name"

git config --global user.email "your.email@example.com"
```

4 Practical

4.1 Create a New Project

```
mkdir react-bootcamp
cd react-bootcamp
```

4.2 Initialize a Git Repository

```
git init
```

4.3 Add Files to the Staging Area

Create a file called `index.html` and add the basic HTML boilerplate in it.

```
git add index.html
```

4.4 Create a Commit

```
git commit -m "Created a new HTML file"
```

4.5 Create a Commit with Changes

Make some changes to `index.html`.

```
git add index.html
```

```
git commit -m "Added a heading to index.html"
```

4.6 Create a New Branch

Create a new branch:

```
git checkout -b feature
```

4.7 Create a Commit in the New Branch

Create a new file called `style.css` and reference it in your `index.html`

```
git add style.css index.html
```

```
git commit -m "Added style.css"
```

4.8 Create a Commit in the `master` Branch

- First, switch to the `master` branch.

```
git checkout master
```

- Add some changes to `index.html`

```
git add index.html
```

```
git commit -m "Added a paragraph"
```

4.9 Visualize the Diverging Changes

```
git log --all --graph
```

4.10 Merge the two changes

- You have two sets of changes in two branches. You should merge them so that they are present together.
- Make sure that you are on the `master` branch.

```
git merge feature
```

- This will create a *merge* commit with the two changes merged.

5 Summary

Command	Description
git init	Initialize a new Git repository
git add <file>	Add file(s) to the staging area
git commit -m "<message>"	Create a commit with a message
git status	Show the status of working directory and staging area
git diff	Show changes not yet staged or committed
git log	Show the commit history
git checkout <branch>	Switch to an existing branch
git checkout -b <branch>	Create and switch to a new branch
git merge <branch>	Merge the specified branch into the current branch

6 Further Study

6.1 Dealing With Merge Conflicts

6.2 Rebasing

6.3 Working with GitHub