28 June 2025
Imran Nawar

Version Control with Git and GitHub

Overview

Version Control

Version Control is a way to track and manage changes in your code over time. It helps individuals and teams collaborate efficiently and safely on software projects.

- Think of it as a time machine for your code, you can go back to older versions if something breaks.

Git

Git is a version control system.

- It tracks changes to your files and allows you to move between different versions of your code.
- Code is very important. Sometimes, your mental level is so good that you write a clever algorithm, and later, you might not even understand it. So it's important to save your code properly.
- Git ensures that your code is never lost and can always be recovered or shared with others.
- Git is the tool (a set of commands) that helps you manage and share your code.

Git also supports:

- **Local versioning** (on your own machine)
- **Remote versioning** (using services like GitHub, GitLab, Bitbucket)

GitHub

GitHub is one of the most popular platforms to host Git repositories online.

- It allows you to store, share, and collaborate on code securely.
- You "push" your code from your local machine to GitHub so it's backed up and available from anywhere.
- GitHub is widely used in the software industry and open-source community.

Getting Started:

1. **Install Git**
   - Installing Git gives you access to the terminal where you can run Git commands like `ls`, `cd`, `clear`, and more.
2. **Create a GitHub Account**
   - Go to github.com, sign up using your name and email, and you're ready to host your code remotely.

What is a Repository?

A **repository** (or "repo") is a folder that Git is tracking.

- It contains all your code and a complete history of every change you have made, every saved state or "commit."
- You can have:
  - A **local repository** (on your machine)
  - A **remote repository** (on GitHub or other hosting platforms)
- Repositories make it easier to manage different versions of a project, collaborate with others, and roll back changes if needed.

Creating Repositories

There are two main ways to create a Git repository:

1. **Locally using Git:**
   ```
   git init
   ```
   This command creates a new Git repository in your current directory.

2. **On GitHub:**
   - Click **"New Repository"**
   - Name your repository
   - Choose public or private
   - (Optionally) add a README, .gitignore, or license
   - Click **"Create Repository"**

Some Common Git Commands (Basics)

These are the most common Git commands you'll use regularly:

- `clone` → (First time) Download the full code from the remote repo to your computer
- `pull` → Get the latest changes from GitHub to your computer
- `add` → Select (stage) the changes/files you want to save in the next commit
- `commit` → Mark your changes as final on your computer (locally)
- `push` → Upload your committed changes from your computer to GitHub

Rule:

Always do a `git pull` before a `git push` to avoid conflicts.

Basic Git Workflow

```
git add .             # Stage changes
git commit -m "msg"  # Save changes locally
git push origin main # Push to GitHub
```

.gitignore File

- The `.gitignore` file tells Git which files or folders to skip.
- Common examples to ignore:
  - `__pycache__/`
  - `.env`
  - `node_modules/`

This keeps the repo clean and secure from unnecessary or sensitive files.

---

requirements.txt

- **requirements.txt** is a file that lists all the dependencies a project needs, making it easy for others to install them using a single command.

File and Folder Navigation

- `pwd` → Show your current working directory (Present Working Directory)
- `cd ..` → Go back to the previous directory (Parent folder)

Working with Git in VS Code (Easy Way)

- You can use Git **without typing commands**, using VS Code's built-in Git interface.
- Make your changes, click on the **Source Control** icon, and then click **commit**, and finally **push**.
- The **GitLens** extension is highly recommended, it shows detailed Git history and who made each line of code.

Branch

- A **branch** is like a separate copy of your codebase where you can safely make changes.
- Developers create a new branch (from `main`) to work independently.
- When the feature is ready, a **Pull Request (PR)** is submitted to merge it into `main`.
- After merging, the branch can be deleted.
- Every GitHub repo starts with a default branch, usually called `main`.

Good Practices

- Keep most of your projects **public**, unless they are client-sensitive.
- Use **lowercase names** for files and folders.
- Avoid using **spaces** in names.
- Once you're used to using the **keyboard**, you'll hate switching to the **mouse** 😊

For Linux Terminal Users

If you're using a Linux terminal or git bash:

- **Copy** → `Ctrl + Insert`
- **Paste** → `Shift + Insert`

(Instead of the usual `Ctrl + C` and `Ctrl + V`)

Contributing to Open Source Projects

Contributing to open source can be a great way to learn and build your portfolio.

A. Setting up the Project

Step 1: Fork the project

- Go to the repository on GitHub and click **"Fork"** to create your own copy.

Step 2: Clone your forked project

- Open Git Bash in your chosen folder, then run:

```
git clone https://github.com/<your-account-username>/<your-forked-project>.git
```

Step 3: Move into the project folder

```
cd <your-forked-project-folder>
```

Step 4: Add a reference to the original repository

- This is to keep your fork up to date with the main project:

```
git remote add upstream https://github.com/<author-account-username>/<original-project>.git
```

Step 5: View your remote connections

```
git remote -v
```

Step 6: Pull latest changes from the original repository

```
git pull upstream main
```

B. Contributing to the Project

Step 1: Create a new branch for your changes:

- (Do not name it `main`, `master`, or something generic.)

```
git checkout -b <your_branch_name>
```

Step 2: Make your changes to the code base

Step 3: Track your changes

```
git add .
```

Step 4: Commit your changes with a message

```
git commit -m “<your commit message>”
```

Step 5: Push your changes to your fork

```
git push -u origin <your_branch_name>
```

Step 6: Create a Pull Request (PR)

- Go to GitHub → your forked repository → click **"Compare & pull request"**

Once submitted, the maintainers of the original project will review your PR and, if accepted, merge it into the main codebase.

```
Thank You!
```