Contents

[1. Introduction 2](#_Toc197615519)

[✅ Why Use GitHub? 3](#_Toc197615520)

[Why it’s Useful: 3](#_Toc197615521)

[Summary Table 5](#_Toc197615522)

[2. Install 7](#_Toc197615523)

[3. Learn Basic Git Commands 8](#_Toc197615524)

[4. Create a GitHub repository 9](#_Toc197615525)

[What You Can Do with gh (Examples): 9](#_Toc197615526)

[Common gh Commands: 9](#_Toc197615527)

[5. Connect local repo to GitHub using: 18](#_Toc197615528)

[6. Work with branches: 21](#_Toc197615529)

[Why is main important? 24](#_Toc197615530)

[7. Understand merge conflicts and how to resolve them 25](#_Toc197615531)

[8. Forking and Pull Requests (PRs) 29](#_Toc197615532)

[9. Git Bash 32](#_Toc197615533)

[10. Use .gitignore files to exclude files 35](#_Toc197615534)

[11. Add README.md and LICENSE files 38](#_Toc197615535)

[12. Github Why use Automation testing? 39](#_Toc197615536)

[13. Use GitHub Issues and Projects for task management 40](#_Toc197615537)

[14. GitHub common commands 41](#_Toc197615538)

[15. Simple GitHub Actions 49](#_Toc197615539)

[16. Work on Robot Framework 51](#_Toc197615540)

# 1. Introduction

1. What is Git?

* Git is a version control system.
* It tracks changes in source code during software development.
* It allows multiple developers to work on the same project without overwriting each other’s work.

Git is a version control system (VCS) because it helps developer’s track and mange changes to source code over time. It allows multiple people to collaborate on a project, keep a history of every changes, and enables reverting to earlier version.

Scenario: Website Development for a client (e.g., a Restaurant)

1. without Git

* You create file manually
  + Index.html
  + Style.css
  + Menu.html etc.,
* You save copies like
  + Website\_v1
  + Website final
  + Website\_final\_real
* Team collaboration is hard: If 2 people edit the same file, one person’s changes may get overwritten.
* If you make a mistake (e.g., delete a section), you may have to redo everything manually.
* Client requests old version: You may not have it saved

Problem

* No proper version history
* Easy to overwrite or loss files
* Hard to track who did what
* Collaboration is risky.

2. with Git

* You initialize a git repository: git init
* You commit every changes:
  + Initial structure (git commit –m “Initial”)
  + Add Menu Page (git commit –m “Added”)
  + Style updates (git commit –m “update”)
* Team member use branches to work independently
  + One works on Contact page
  + Another works on gallery page
* You can always revert to previous version if needed.
* You push to github to share with team or client.

Advantaged

* Full version history
* Easy to undo mistakes
* Safe collaboration with branches
* Easy to extract who did what and when
* Backup on GitHub or gitlab

2. What is GitHub?

GitHub is a cloud-based platform that hosts code repositories using Git version control. It allows individuals and teams to collaborate on projects, track changes, and manage software development more efficiently.

Github use deploy the simple projects.

### ✅ Why Use GitHub?

| **Benefit** | **Description** |
| --- | --- |
| 🧠 **Version Control** | Tracks and stores changes in code (via Git). You can roll back to previous versions. |
| 👥 **Collaboration** | Multiple developers can work on the same project using branches and pull requests. |
| ☁️ **Cloud Backup** | Your code is safely stored online and accessible from anywhere. |
| 🛠️ **Project Management** | Use **Issues**, **Projects**, and **Actions** to manage bugs, tasks, and automated workflows. |
| 💼 **Portfolio** | Developers showcase projects on GitHub to employers and clients. |
| 💬 **Community** | Open-source collaboration: contribute to or learn from others’ public projects. |

Example of Github inn Action.

Git init, git add

## Why it’s Useful:

* Code is versioned and backed up
* Everyone sees who changed what and why
* Easy collaboration across time zones and locations

Example: “My Portfolio website”

1. without using GitHub

* Create files on index.html, style.css, about.html
* You make changes and save them manually
* You keep backup version like
  + Index\_v1.html
  + Index\_final.html
  + Index\_final\_final.html
* Your friend sends back edited files with different names: about\_edited\_by\_john.html

Problem

 Messy file versions

 No history of what was changed

 Collaboration is slow and confusing

 No easy way to roll back mistakes

 No automatic backup

2. with Github

* You build the same portfolio, but with Git and Github.
* Create the project and initialize Git
  + Git init
* You commit code with clear message:
  + git commit -m "Added homepage layout"
  + git commit -m "Styled about section"
* You push it to GitHub
  + git remote add origin https://github.com/yourusername/portfolio-site.git
  + git push -u origin main

Benefits

 Clean version history with commit messages

 Collaborators don’t overwrite each other

 Instant backup in the cloud

 Easy to share project link:  
https://github.com/yourusername/portfolio-site

 can deploy using GitHub Pages

### Summary Table

| **Feature** | **Without GitHub** | **With GitHub** |
| --- | --- | --- |
| Version Tracking | Manual (file copies) | Automatic (git commit history) |
| Collaboration | Manual file sharing | Forks, branches, pull requests |
| Backup | Local only | Cloud backup |
| Rollback Changes | Hard (no history) | Easy with git log and git revert |
| Sharing Work | Send ZIP file | Share GitHub link |

Who developed Git and Github?

Git

Created by Linus Torvalds, the creator of the Linus operating system in 2005.

GitHub

Developed by Tom Preston-Werner, Chris Wanstrath, PJ Hyett, and Scott Chacon in 2008.

Current owner Microsoft

Summary:

| **Tool** | **Created By** | **Year** | **Purpose** |
| --- | --- | --- | --- |
| **Git** | Linus Torvalds | 2005 | Fast, distributed version control |
| **GitHub** | GitHub, Inc. (Preston-Werner et al.) | 2008 | Cloud platform for Git collaboration |

Git and Github are same.

No, they are not the same, but they are closely related.

| **Feature** | **Git** |  | **GitHub** |
| --- | --- | --- | --- |

|  |  |  |
| --- | --- | --- |
| **Type** | Tool / System | Web-based service/platform |

|  |  |  |
| --- | --- | --- |
| **Function** | Version control system | Hosting platform for Git repositories |

|  |  |  |
| --- | --- | --- |
| **Runs on** | Your local computer (CLI-based) | The cloud (web interface) |

|  |  |  |
| --- | --- | --- |
| **Created by** | Linus Torvalds (2005) | GitHub, Inc. (2008), acquired by Microsoft |

|  |  |  |
| --- | --- | --- |
| **Used for** | Tracking code changes,  branches | Collaboration, backup, PRs, issue tracking |

Why Git was introduced?

Before git, developers used centralized version control systems like CVS and SVN.Which had major limitation.

* Server no access
* Slow performance for large projects
* Weak support for branching and merging
* Not ideal for open-source, distributed collaboration.

Git Purpose

* Enable fast, distributed and reliable version control
* Support large scale development
* Developer have the full history
* Improve branching, merging and collaboration

Why GitHub was introduced?

Before GitHub hard to use share the code with others.

* No easy hosting
* No friendly UI or collaboration tools.
* Difficult to manage pull requests and issues in teams

GitHub Purpose

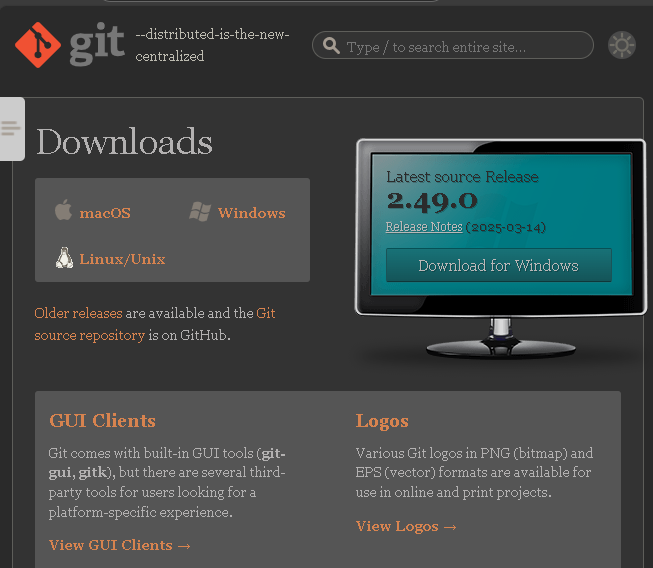
* Make Git easier and more accessible
* Provide cloud hoisting for git repositories
* Offer tool like Pull Requests, Issues, Wikis, Action (CI/CD).

# 2. Install

Part - 1

Download Git

<https://git-scm.com/downloads>



2. Check Git Installation

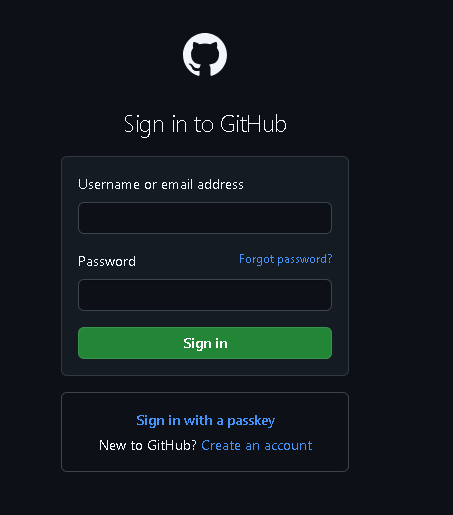
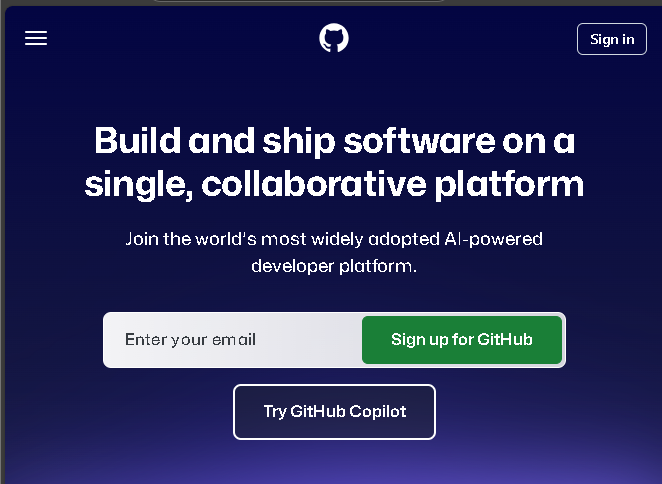
Open Command Prompt (Windows) or Terminal (Mc/Linux) and run

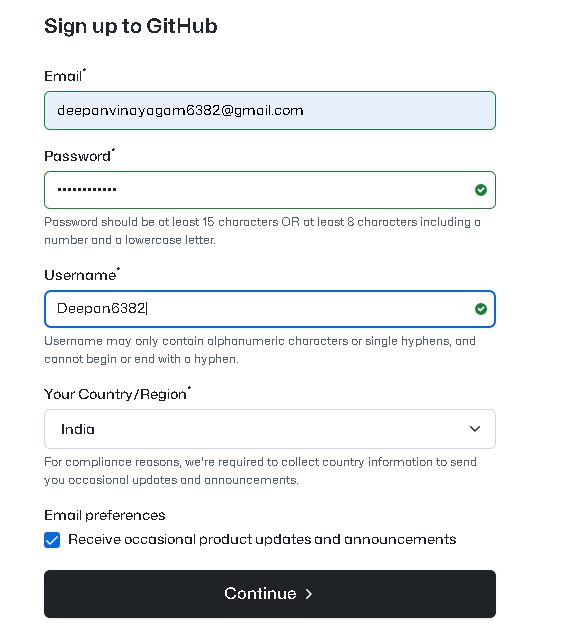


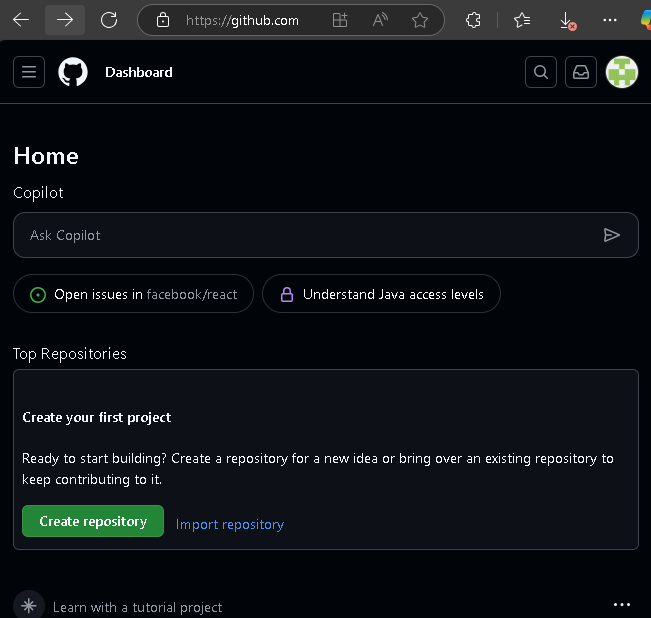
Part 2

1. Crete an account

Go to 👉 <https://github.com/>







# 3. Learn Basic Git Commands

What Learn?

* Creating a git project
* Tracking changes
* Saving version
* Checking status/history
* Pushing to GitHub

# 4. Create a GitHub repository

Method 1

Create a GitHub repository and push a .txt file fully from the terminal. You can do this using the GitHub CLI (gh) tool.

✅ Create a GitHub repository from terminal  
✅ Add a text file  
✅ Push it to GitHub

What is GitHub CLI (gh)?

gh stands GitHub CLI

* g = GitHub
* h = CLI (Command Line Interface)

Using gh allows you to manage Github directly from your terminal without switching to the browser. It saves time and streamlines workflow, especially for developers.

## What You Can Do with gh (Examples):

| **Feature** | **Without CLI (Manual)** | **With CLI (gh)** |
| --- | --- | --- |
| Create a repository | Go to GitHub site manually | gh repo create my-repo |
| Clone a repo | Copy-paste URL | gh repo clone user/repo |
| View issues or pull requests | Open browser tabs | gh issue list, gh pr view |
| Create issues or pull requests | Use web interface | gh issue create, gh pr create |
| Login and authentication | Browser-based only | gh auth login |
| Check repo status or notifications | Manual navigation | gh repo view, gh api |

## Common gh Commands:

gh auth login # Log into GitHub

gh repo create # Create a repo

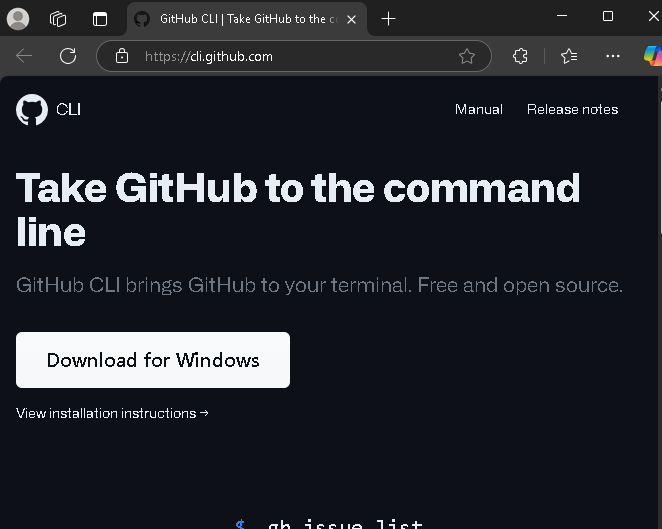
gh repo clone # Clone a repo

gh issue create # Create an issue

gh pr create # Create a pull request

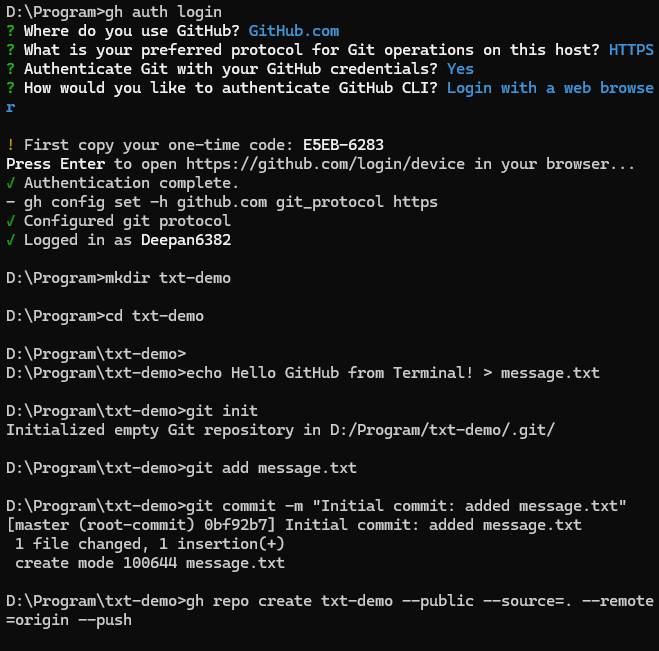
gh repo view # View repo info

Step 1 : Install Github CLI(gh)



Step 2 : Login To GitHub from Terminal

gh auth login



Choose GitHub.com

Choose HTTPS

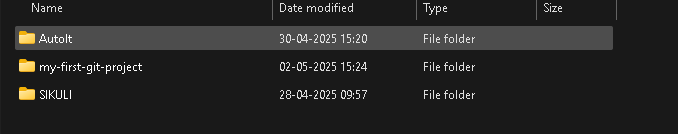
Choose Login with a web browser

Follow the instruction and paste the code in your browser.

It will login your account

Step 3: Create a Local Project Folder

mkdir my-first-git-project

Cd my-first-git-project

Step 4 : Create a Text file

Echo text\_value > text\_file\_name

echo Hello GitHub from Terminal! > message.txt

It is used in the terminal to create a file and write text.

* Echo: A shell command used to display text or output in string.
* Hello GitHub from Terminal! : The actual text (string) you want to write into the file.
* >: Output redirection operators – it sends the output to a file instead of printing to the screen.
* Message.txt: The file name that will be created or overwritten with the text.

type message.txt # Windows

Step 5: Initialize Git and Commit

git init

git add message.txt

git commit -m "Initial commit: added message.txt"

Step 6: Create a Github Repository from terminal

gh repo create txt-demo --public --source=. --remote=origin –push

This is a **one-line GitHub CLI command** that:

✅ Creates a private GitHub repository from your current local folder  
✅ adds a remote called origin  
✅ pushes your local code to the new GitHub repo

🔍 FULL STEP-BY-STEP BREAKDOWN

✅ Step 1: gh repo create

This is the main GitHub CLI command used to **create a new repository** on GitHub from the command line.

* It’s a shortcut to avoid going to GitHub.com and creating a repo manually.
* You must be **authenticated** with gh auth login.

✅ Step 2: --private

This flag makes the new GitHub repository **private**, meaning:

* Only you (and collaborators you invite) can access it.
* It **won’t appear in public search results** or to others without permission.

You can replace it with --public if you want it open to everyone.

✅ Step 3: --source=.

This tells the CLI:

"Use the current folder (.) as the source of the repository."

* The . means "this directory".
* GitHub CLI will check if this folder is already a Git repo (i.e., has a .git folder).
  + If not, it will **initialize Git**, add files, and commit.

This avoids having to run:

bash

Copy code

git init

git add .

git commit -m "Initial commit"

manually.

✅ Step 4: --remote=origin

This sets the remote URL of your local repository to point to the GitHub repo, using the name origin.

* origin is the **default name for the main GitHub remote** in Git projects.
* Behind the scenes, it runs something like:

bash

Copy code

git remote add origin https://github.com/your-username/repo-name.git

✅ Step 5: --push

This tells the GitHub CLI to **automatically push the code to GitHub** after creating the repository and setting the remote.

Equivalent to:

bash

Copy code

git push -u origin main

(if your local branch is main).

✅ What Happens Behind the Scenes (Automatically):

1. Checks if .git folder exists (if not, runs git init)
2. Adds all files: git add .
3. Commits them: git commit -m "Initial commit"
4. Creates GitHub repo (private) using the GitHub API
5. Adds the remote: git remote add origin ...
6. Pushes code: git push -u origin main

🧠 Why use this command?

✅ Pros:

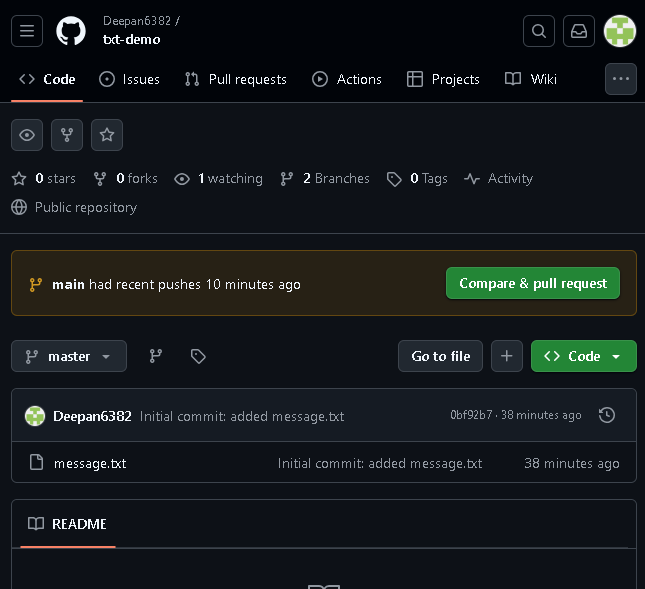
* Saves time — everything done in **one step**
* Perfect for scripting, automation, or daily dev workflow
* Avoids switching between GitHub website and terminal
* Useful for **private projects**, prototypes, or internal tools

❗ Caution:

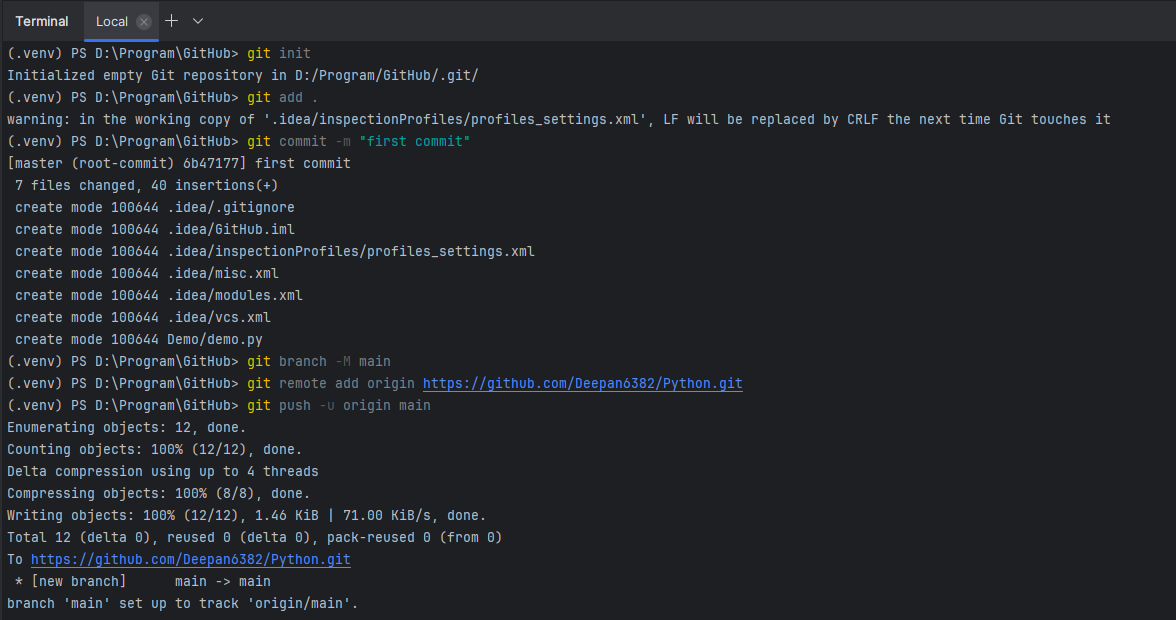
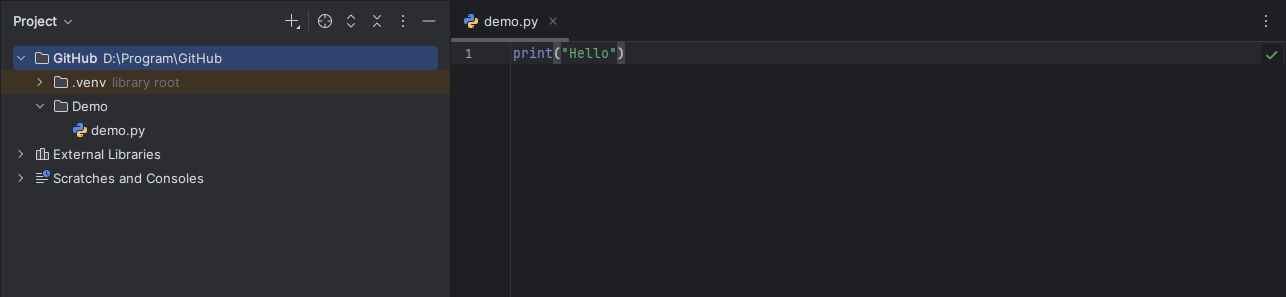
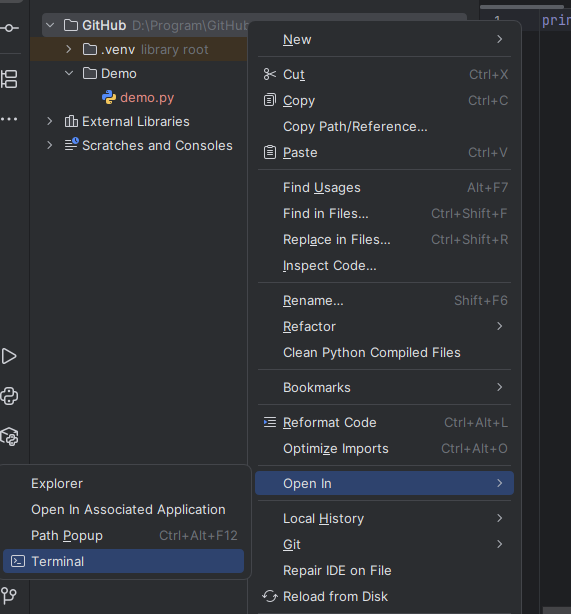
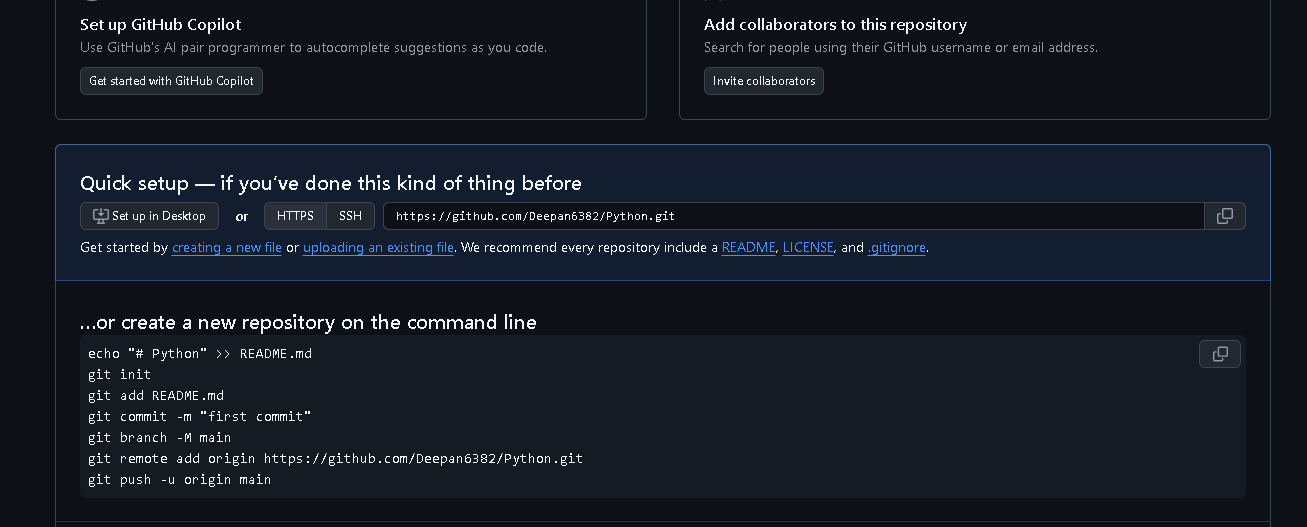
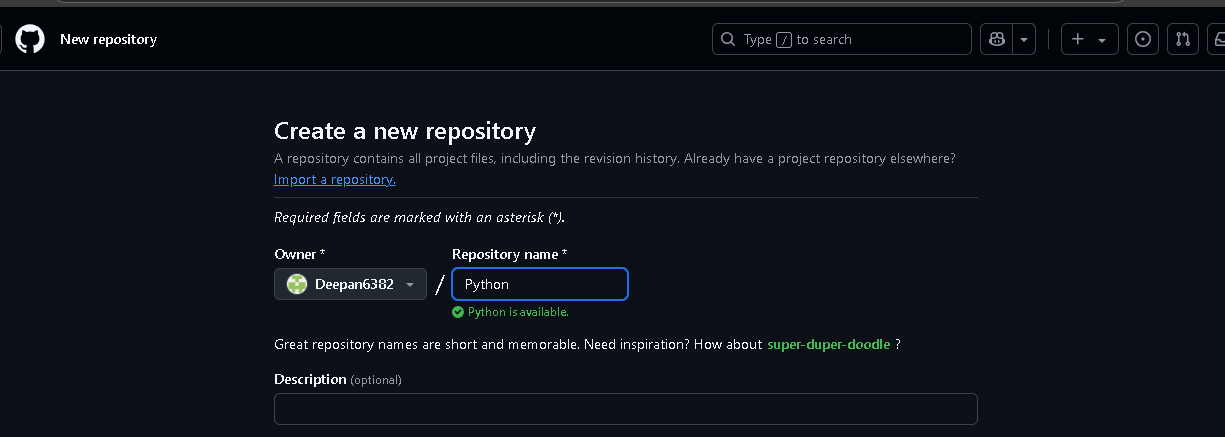
* You must be **authenticated with gh auth login**
* If there's already a remote called origin, it may fail
* Assumes your current branch is main; you may need to rename it

STEP 7: Confirm

Go to https://github.com/Deepan6382/txt-demo in your browser — your .txt file should be there 🎉



Method 2



# 5. Connect local repo to GitHub using:

When you create a git project on your local computer, Git trackes your changes only locally. To share it online, back it up, or collaborate with others, you need to push your project to a remote repository like GitHub.

 ✅ Upload your code to GitHub

 ✅ Share and collaborate with others

 ✅ Backup your project online

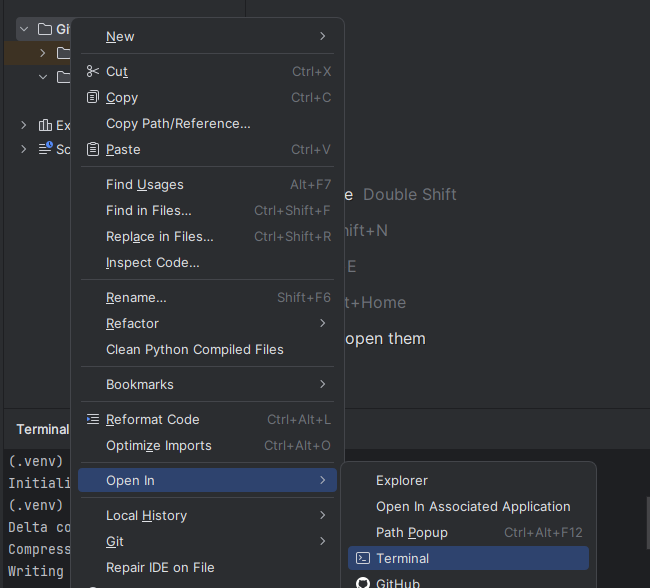
 ✅ Use GitHub tools (issues, PRs, actions, etc.)

Steps to connect Local Git Repo to Github

Step 1: Create a repository on github.

Step 2: cd path/to/your/local-project

Or



Step 3: Initialize Git Locally

git init

Creates a .git folder to start tracking changes

Step 4: Add Files and Commit

Git add .

Git commit – m “Initial commit”

Step 5: Connect to the Github Repository (Remote)

git remote add origin <https://github.com/YourUsername/demo-project.git>

This command connects your local project to the GitHub repository using the URL.

Step 6: Push your code to github

git push –u origin main

push – upload your code

-u – Set this remote as default for future pushes

Origin – The remote name (you can rename it)

Main – the branch you pushing

After This

Git push – to update GitHub

Git pull to get changes from GitHub

✅ Summary

| **Step** | **Purpose** |
| --- | --- |
| git init | Start local repo |
| git add | Stage files |
| git commit | Save snapshot |
| git remote add origin | Link to GitHub |
| git push -u origin main | Upload project |

# 6. Work with branches:

What is a Branch in Git?

A branch in git is a separate version of your project, allowing you to work on different features or fixes without affecting the main code.

By default, your project has a branch called main.

Why Use Branches?

New Features: Build without touching the main code.

Bug Fix: Fix bugs in isolation

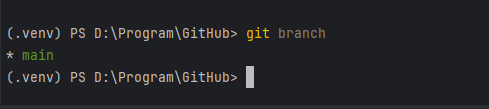
Experiment: Try new ideas safely

Collaboration: Team members can work independently.

Once done, you can merge the branch back into main.

Basic Git Branch commands

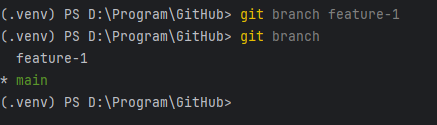
Step 1: Check Current Branch



`Step 2: Create a New Branch

Git branch feature-1

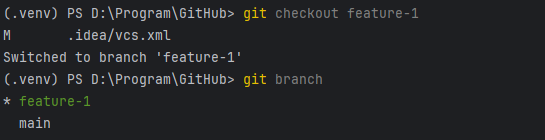
This creates a new branch called feature-1 (but doesn’t switch to it yet)



Step 3: Switch to that branch

Git checkout feature-1

\* change

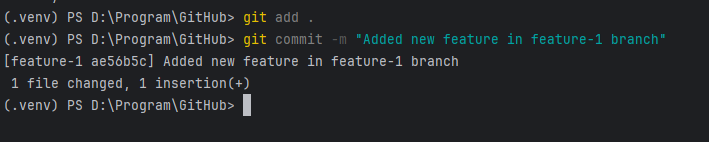


You’re now working on feature-1

Step 4: Work on your feature

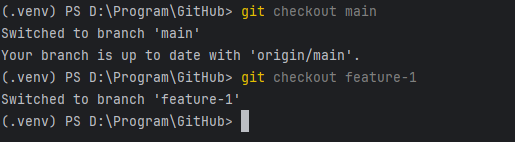
git add .

git commit -m "Added new feature in feature-1 branch"



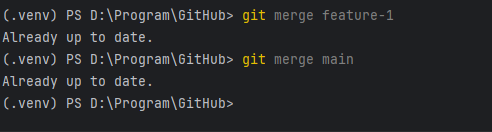
Step 5: Merge Branch into main

Switch to main

Git checkout main

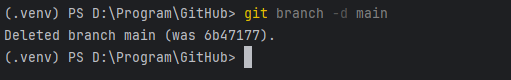
Merge your changes

Git merge feature-1



Step 6: Delete Branch (Optional)

Git branch –d feature-1



| **Command** | **What It Does** |
| --- | --- |

|  |  |
| --- | --- |
| git branch | List all branches |

|  |  |
| --- | --- |
| git branch name | Create new branch |

|  |  |
| --- | --- |
| git checkout name | Switch to branch |

|  |  |
| --- | --- |
| git checkout -b name | Create + switch |

|  |  |
| --- | --- |
| git merge name | Merge into current branch |

|  |  |
| --- | --- |
| git branch -d name | Delete a branch |

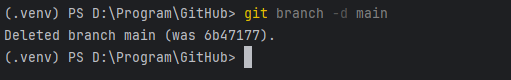
Undo and redo the file

Know the difference

Git branch –d main: Deletes the branch safely (only if it is merged)

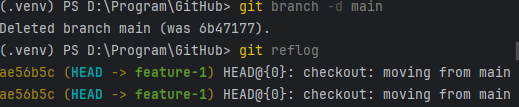
Git branch –D main: Force deletes, even if unmerged

Solution: Undo Deleted Branch



Option 1: Restore with commit Reference (if you remember the last commit)

Git reflog



Main or master in git

Main is the default primary branch in a git repository.

It is the main line of development

In older Git versions, it was called master, but now main is the new standard.

## Why is main important?

| **Purpose** | **Explanation** |
| --- | --- |
| 🧩 Central codebase | The branch where stable and tested code lives |
| 🚀 Deployment-ready | Often used for deploying the live app |
| 🛠️ Merge target | Most feature and bug fix branches merge here |
| 🧼 Always stable | Should always be clean, tested, and working |

# 7. Understand merge conflicts and how to resolve them

A merge conflict occurs when Git can’t automatically merge changes between branches because the same part of the same file was modified in two branches differently.

Example

You and your teammate both edited the same line of code in a file on different branches.

When you try to merge, Git doesn’t know which version to keep.

Why Merge Conflicts Happen?

* Two branches change the same line in a file
* One branch deletes a file the other modified
* File renaming or structure changes occur on both branches.

Simple Example

1. Initialize a git repo

mkdir merge-conflict-demo

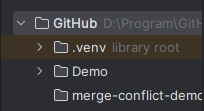
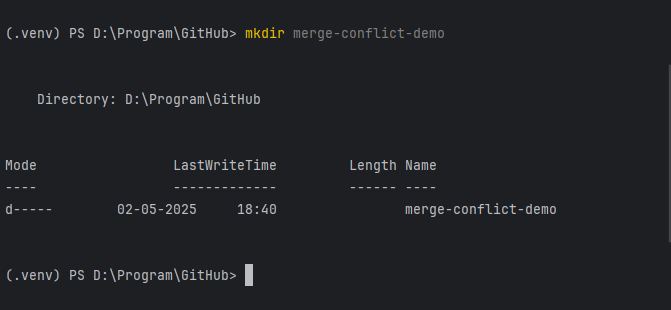
cd merge-conflict-demo

git init

echo "Line 1" > file.txt

git add .

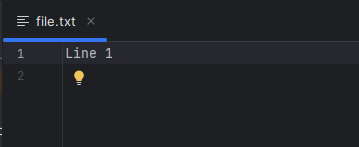
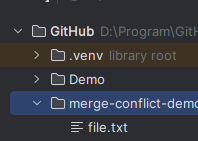
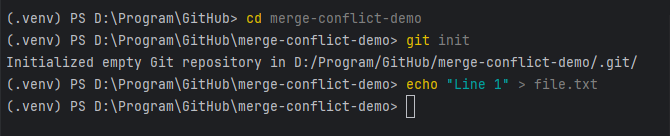
git commit -m "Initial commit"



Cd merge-conflict-demo

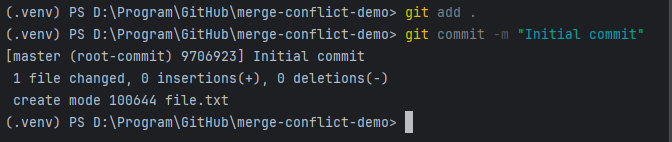
git init

echo "Line 1" > file.txt



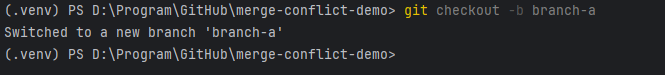
git add .

git commit -m "Initial commit"



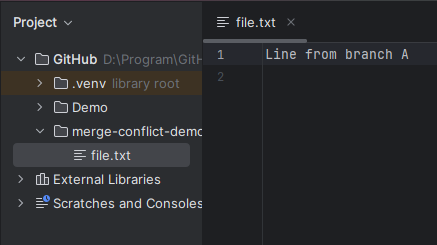
2. Create and switch to a new branch (To edit the line)

git checkout -b branch-a

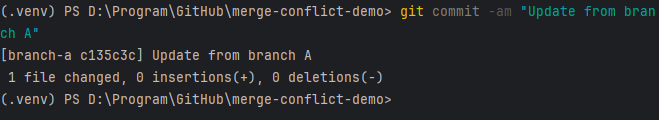


echo "Line from branch A" > file.txt

(Edit the text file)



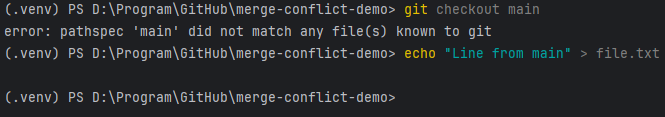
git commit -am "Update from branch A"

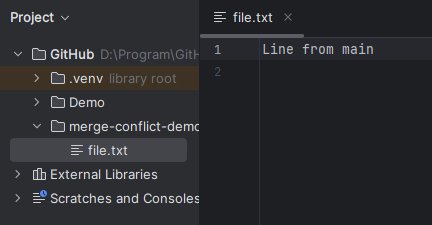


3. Switch back to main and make a different change

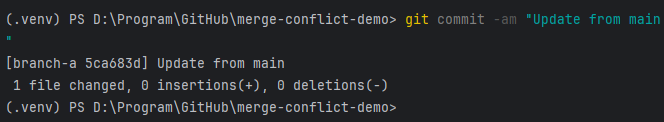
git checkout main

echo "Line from main" > file.txt



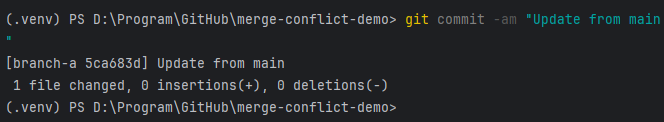


git commit -am "Update from main"



4. Try to merge branch-a into main

git merge branch-a



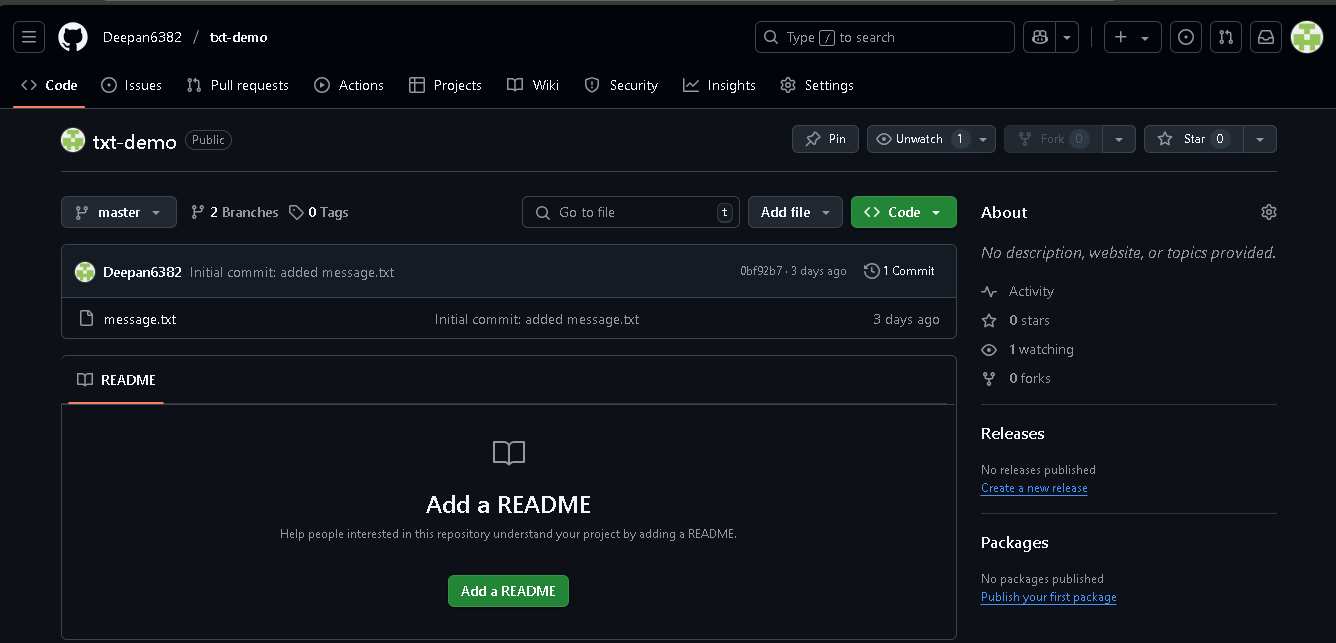
# 8. Forking and Pull Requests (PRs)

Definition

A fork is your own personal copy of someone repository on github.

Why Fork?

* To experiment with changes without affecting the original project.
* To contribute to open-source projects (especially when you don’t have direct write access)
* To build your own variant or customization of a project.

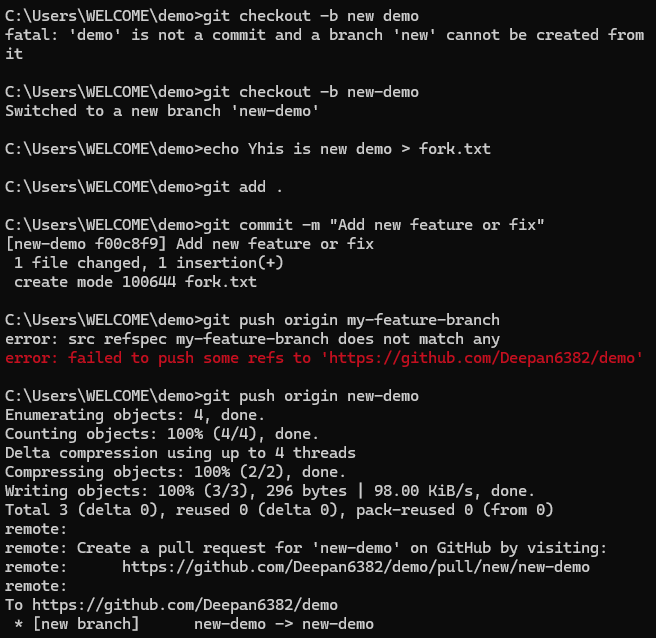
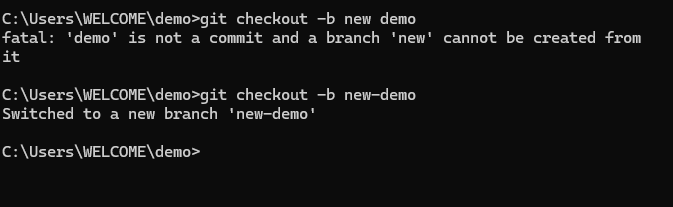
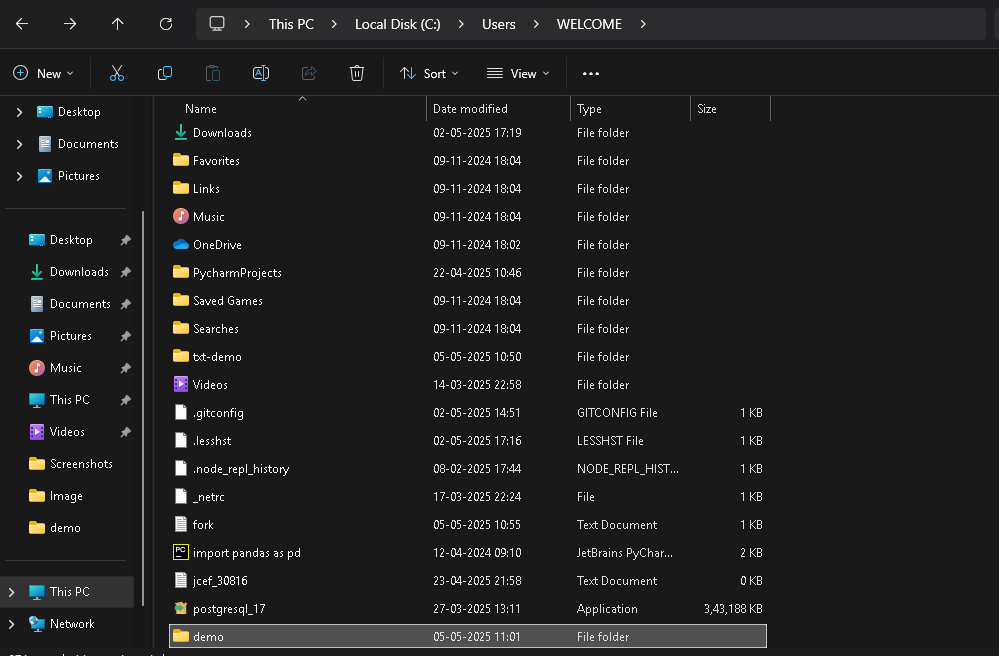
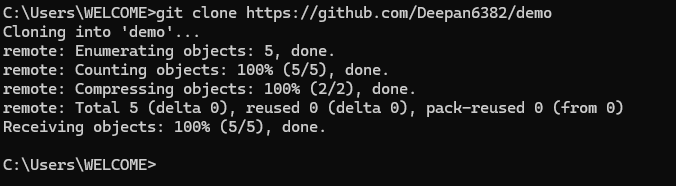


What is a Pull Request (PR)?

A Pull Request is a request to merge changes from one repository/branch into another (usually from your fork in to the original project).

Why use pull Requests?

* To purpose changes or features to someone project.
* To review and discuss code before merging (collaborative development)
* To enable code review, CI/CD checks, and feedback loops.



Step By step

1. Clone Your Repository (Terminal or Git Bash)

git clone https://github.com/your-username/your-repo-name.git

cd your-repo-name

2. Create a New Branch

You create a new branch to safely work on your feature

git checkout -b my-feature-branch

git checkout -b new-topic

3. Make Code Changes

Use any code editor (like VS code) to edit files in the project

4. Stage and Commit the changes

git add .

git commit -m "Add new feature or fix"

5. Push the branch to Github

git push origin my-feature-branch

git push origin new-topic

6. **Create a Pull Request (PR)**

1. Go to your repo on GitHub:  
   https://github.com/your-username/your-repo-name
2. GitHub will show a message like:

"**Compare & pull request**"  
Click that.

1. Review the title and description.
2. Click **“Create Pull Request”**.

7. **Review and Merge (Optional)**

If you're the maintainer:

* Go to the Pull Request tab on GitHub.
* Review changes.
* Click **“Merge Pull Request”** when ready.

Real-World Use Case Example:

You want to contribute to an open-source calculator app:

1. 🔁 **Fork** the original repo into your GitHub account.
2. 👨‍💻 Clone it, make changes (e.g., add a new feature).
3. 🚀 Push the changes to your fork.
4. 🔄 Open a **Pull Request** to the original repository.
5. ✅ The maintainer reviews, discusses, and **merges** it if accepted.

# 9. Git Bash

Git Bash is a Command line tool that lets you use Git on Windows using Unix-style Shell

It is a combination of

Git – Version control system

Bash – A Unix style shell (like Linux, macOS terminal)

It allows you to use Git features like clone, push, pull, commit, etc.

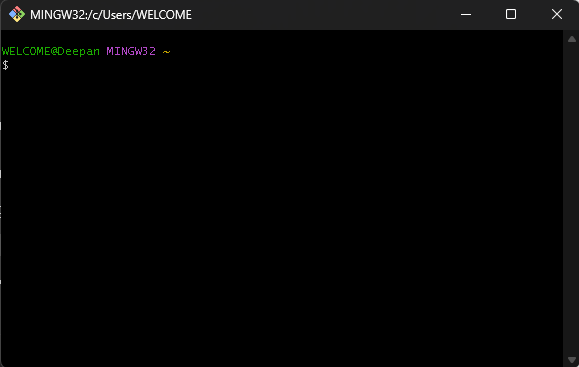
Download Git Bash in browser

1. Click start menu

Type GIT BASH

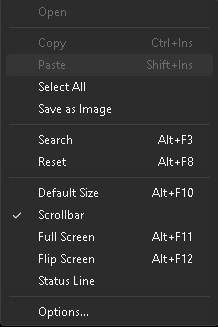
2. Click on Git Bash

It will open a terminal window with a $ prompt



Ctrl + c to copy but to paste into Git Bash Ctrl+V not Work in Git Bash terminal

1. Right Click to Paste



Basic Command in Git Bash

1. Git Configuration (one-time setup)

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

git config --global user.email [your@email.com](mailto:your@email.com)

2. Project Setup & Navigation

| **Command** | **Description** | |
| --- | --- | --- |
| Pwd | | Show current directory (Print Working Directory) |
| Ls | List files in current directory | |
| cd foldername/ | Change into a folder | |
| cd .. | Go back one directory | |
| mkdir new-folder | Make a new folder | |
| Clear | Clear the terminal screen | |

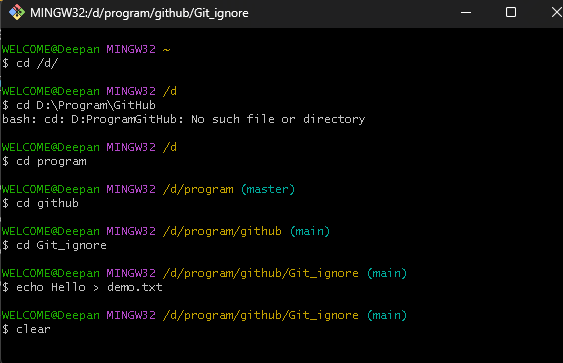
Cd /d/ Change the C drive to D

3. Working with Repositories

| **Command** | **Description** |
| --- | --- |
| git init | Initialize a new Git repo |
| git clone <repo-url> | Clone an existing repo from GitHub |
| git status | See current changes/staging info |
| git add . | Stage all changes |
| git commit -m "Your message" | Commit staged changes |
| git push origin branch-name | Push commits to GitHub |
| git pull | Pull latest changes from remote |
| git branch | Show local branches |
| git checkout -b new-branch | Create and switch to a new branch |
| git checkout branch-name | Switch to an existing branch |
| git merge branch-name | Merge branch into current branch |

4. Undo/ Fix Mistakes

| **Command** | **Description** |
| --- | --- |
| git reset HEAD file | Unstage a file |
| git checkout -- file | Undo local changes to a file |



# 10. Use .gitignore files to exclude files

.gitignore is a specific file used by Git to ignore specific files or folders in your project.

Or

.gitignore file is a plain text file used by Git to tell it which files or directories should be ignored and not tracked or committed to a Git repository. This is useful for excluding temporary files, build artifacts, or sensitive information that you don’t want to upload to your version control system (like Github).

Why use .gitignore?

| **Reason** | **Example** |
| --- | --- |
| ✅ Avoid uploading huge unnecessary files | node\_modules/, build/, dist/ |
| 🔐 Protect sensitive info | .env (contains API keys, passwords) |
| 💻 Avoid machine-specific files | .DS\_Store, Thumbs.db, .vscode/, .idea/ |
| ⚙️ Ignore temporary or generated files | \*.log, \_\_pycache\_\_/, .pyo, .class |

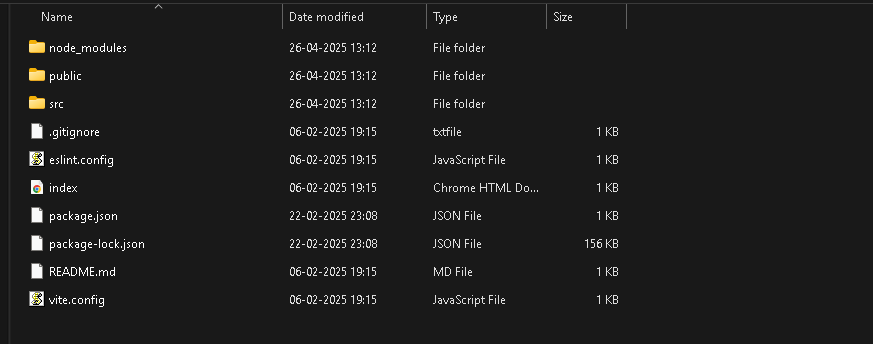
When to Use .gitignore?

Use .gitignore:

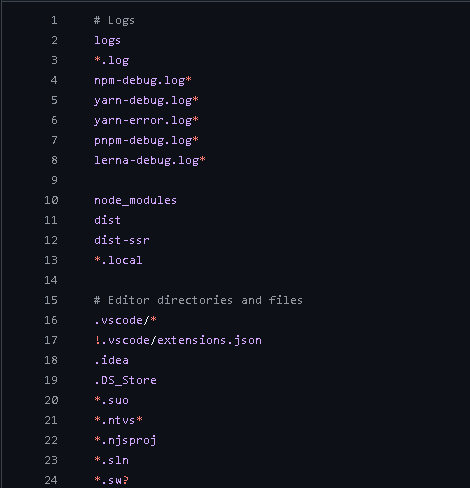
* At the **start of a project**
* Before the first git add .
* When you’re working with:
  + Node.js / React → ignore node\_modules/
  + Python → ignore \_\_pycache\_\_/, env/
  + Java → ignore \*.class, target/
  + Logs or temp files → ignore \*.log, \*.tmp

For Example

F:\Full stack\React.js\Pratice\pratice\MyApp\resurant\_website



Ignore the file



Why node Upload Node\_module Folder

| **❌ Don't Upload node\_modules/** | **✅ Reason** |
| --- | --- |
| It’s **huge** (can be hundreds of MBs) | Increases repo size and slows down GitHub |
| It can be **regenerated** | Use npm install to recreate it from package.json |
| Causes **platform issues** | Some modules behave differently on Windows/Linux |

You Should Know Copying Code from GitHub

Copy the project in a folder (Other person upload the code)

What can go wrong?

| **Problem** | **Solution** |
| --- | --- |
| ❌ node\_modules/ is missing | Run npm install |
| ❌ .env file is missing | Ask the original author or set your own environment variables |
| ❌ Project not running | Check for missing files or wrong commands (check README.md if available) |

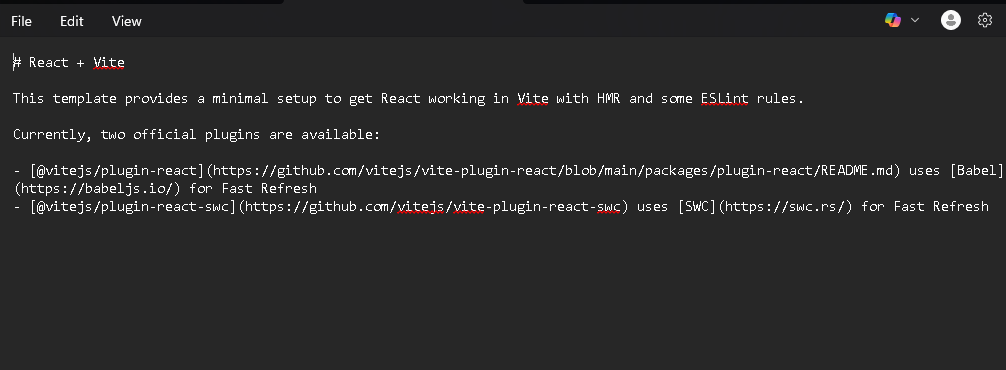
# 11. Add README.md and LICENSE files

What is a README.md file?

A README>md is a markdown file that gives an overview of your project. It is the first thing people see when they visit your GitHub repository.

Why use README.md

| **Purpose** | **Description** |
| --- | --- |
| 📖 Project Info | Describes what your project does and how to use it |
| 🚀 Installation | Tells users how to set up and run the code |
| 🛠️ Usage | Shows examples or demos of the project |
| 🤝 Contribution | Explains how others can help improve the project |
| 📄 Documentation | Links to docs, FAQs, or issues |



License

Declares how other can legally use your code.

# 12. Github Why use Automation testing?

GitHub uses automation testing to improve code quality, speed, and collaboration. When developers push code to GitHub, automation tools like Github Actions, Jenkins can automatically run tests to make sure the code works correctly before merging it.

Benefits of Automation Testing in GitHub

| **Benefit** | **Description** |
| --- | --- |

|  |  |
| --- | --- |
| ⚡ **Fast Feedback** | Tests run automatically on every push, pull request, or commit — no manual steps needed. |

|  |  |
| --- | --- |
| 🔍 **Catch Bugs Early** | Automated tests detect errors as soon as they happen, before the code reaches production. |

|  |  |
| --- | --- |
| 🤝 **Better Collaboration** | Helps multiple team members work safely on the same project without breaking each other's code. |

|  |  |
| --- | --- |
| 🔁 **Repeatable & Reliable** | Same test runs every time — reduces human error. |

|  |  |
| --- | --- |
| 🚀 **CI/CD Integration** | Works with continuous integration pipelines to automatically test and deploy code. |

# 13. Use GitHub Issues and Projects for task management

GitHub issues is a build-in tool for tracking bugs, features request, tasks, or question related to a project. Or

GitHub isn’t just for storing code – it also helps you plan, organize and track work using issues and projects. This is especially useful in automation testing. Where multiple tasks(like writing, fixing or running test scripts) need to be tracked.

GitHub Issues

|  |  |
| --- | --- |
| 🐞 **Issues** | Track bugs, tasks, or feature requests easily |

|  |  |
| --- | --- |
| 📋 **Projects** | Organize work into kanban-style boards (To do / In Progress / Done) |

|  |  |
| --- | --- |
| 🤝 **Collaboration** | Assign team members, link commits, PRs, and updates to each task |

|  |  |
| --- | --- |
| 🔔 **Transparency** | Everyone in the team can see progress, priorities, and workload |

|  |  |
| --- | --- |
| ⏳ **Workflow** | Helps manage agile development (sprints, milestones, status tracking) |

GitHub Projects?

GitHub projects is like a Kanban Board(similar to Trello or Jira) that organizes your issues or tasks into visual column like

* **To Do**
* **In Progress**
* **Done**

📌 **Why Use It:**

* Visual task tracking
* Easy to manage team workflow
* Drag-and-drop Issues between columns
* Supports automation with GitHub Actions

# 14. GitHub common commands

GitHub commands help you work efficiently with Git repositories. These commands are used to manage code version, collaborate and, maintain project workflow.

Check current git branch -a

1. git init

* Initialize a new Git repository in the current directory.
* This command sets up your project to be tracked by git. You only need to use it once when you start working on a project.

Example Project Structure After git init:

my-project/

│

├── .git/ # Git's internal repository

│ ├── config # Git configuration file

│ ├── HEAD # Pointer to the current branch

│ ├── index # Staging area for files

│ └── objects/ # Store for git objects (commits, trees, etc.)

│

├── README.md # Project's README file

├── .gitignore # List of files/folders to exclude from version control

├── app.py # Your project code

└── requirements.txt # Dependencies for your project (if using Python)

2. git clone <repository\_url>

* Creates a copy (clone) of a remote repository on your local machine.
* This is used when you want to download an existing repository from GitHub to start working on it locally.

Example

git clone <https://github.com/username/repository.git>

3. git add <file>

Stages a file or (or multiple files) to be committed to the repository.

Before committing any changes, you need to add files to the staging area.This allows you to choose which files should be committed.

Example

git add index.html

To add files in the directory

git add .

4. git commit –m “message”

Commits the stages changes to the local repository with a message describing the changes.

After adding files, this command permanently records the changes in the project history with a message explaining what was done.

Example

git commit -m "Fixed login page bug"

Meaning of –m:

-m allows you to provide a commit message directly in the command line. The message inside the quotes (“Fixed login page bug”) describe what changes you made in the commit.

5. git status

Display the status of the current repository, showing changes that have been staged, changes that have not been staged, and files that aren’t being tracked.

This command helps you see the state of your files in the repository. It tells you which files are ready to be committed and which ones need to be added.

Example

Git status

6. git push <remote> <branch>

Pushes your local commits to the remote repository (Github) to update the project.

After committing locally, you push the changes to GitHub so that others can see and collaborate on your work.

Example

Git push origin main

6.1. Git push origin main

Pushes the current main branches to the remote repository named origin.

When you already have a tracking relationship set between your local branch and the remote branch.

6.2. git push -u origin main

-u is shorthand for –set-upstream.

It tells Git “Push my local main branch to the remote called origin, and set origin/main as the default upstream branch.”

7. git pull

Fetches the changes from the remote repository and merges them into your local repository.

This command allows you to stay up-to-date with the latest changes made by other contributes. It’s used to sync your local branch with the remote branch.

Syntax

Git pull origin main

8. git branch

Lists all branches in the repository and shows which branch you’re currently on.

This is used to check your current branch and see what other branches are available in the repo

git branch -M main

🔍 What -M means:

-M stands for "move/rename branch".

9. git checkout <branch>

Switches to another branch in your repo

When working on multiple features, you can switch branches to isolate changes and avoid conflicts. It helps you navigate between different parts of your projects.

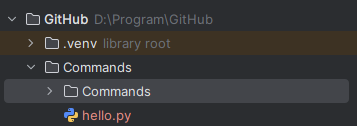
10. git merge <branch>

Merges changes from one branch into the current branch.

When you’ve finished working on a features on a feature or bug fix in a separate branch, you use this to combine those changes back into the main branch (e.g., main or master)

Syntax

Git branch



11. git remote –v

Displays the URL(s) of the remote repository connected to your local repository.

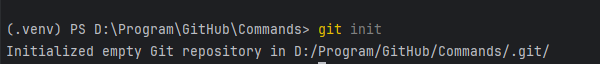
Helps you verify which remote repository you're connected to, particularly useful when working with multiple remotes.

12. git log

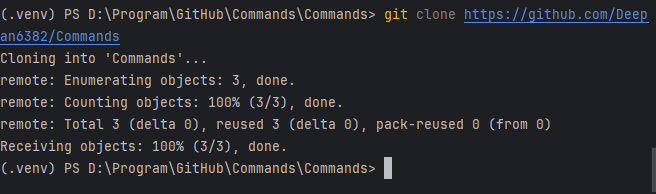
Shows the commit history of the repository.

You can use git log to review the commit history, see who made changes, and view commit messages.

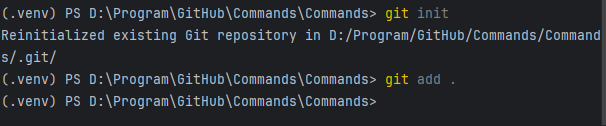
1. git init

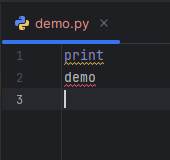
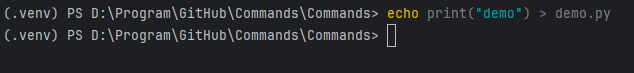


2. git clone <repository\_url>



3. git add <file>

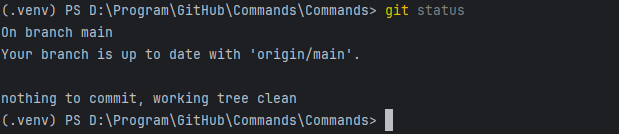




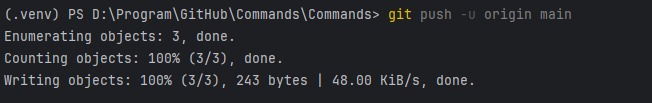
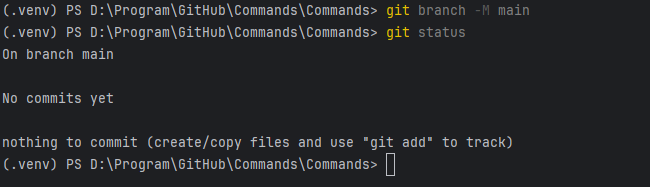
4. git commit –m “<message>”



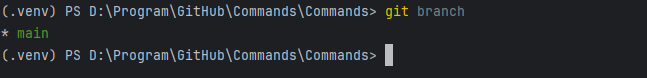
5. git status



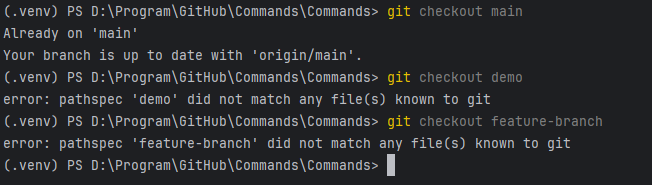
6. git push



8. git branch

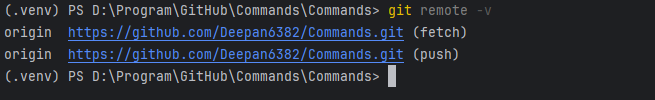


9. git checkout <branch>



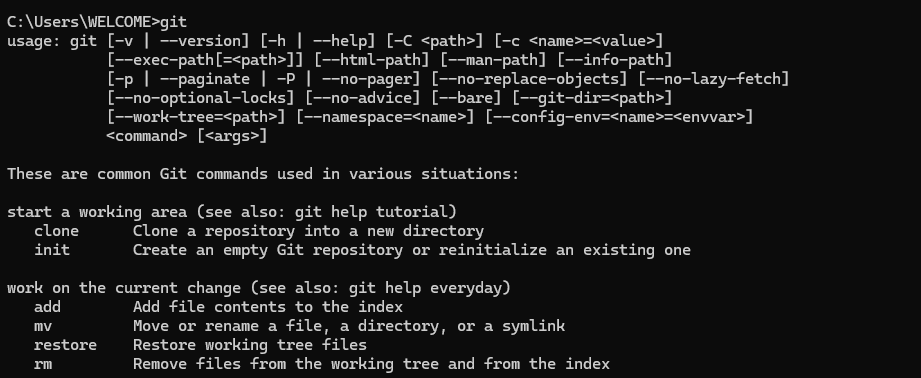
10. git merge <branch>

11. git remote –v



12. git log





1. start a working area (see also: git help tutorial)

clone Clone a repository into a new directory

init Create an empty Git repository or reinitialize an existing one

2. work on the current change (see also: git help everyday)

add Add file contents to the index

mv Move or rename a file, a directory, or a symlink

restore Restore working tree files

rm Remove files from the working tree and from the index

3. Examine the history and state (see also: git help revisions)

bisect Use binary search to find the commit that introduced a bug

diff Show changes between commits, commit and working tree, etc

grep Print lines matching a pattern

log Show commit logs

show Show various types of objects

status Show the working tree status

4. grow, mark and tweak your common history

backfill Download missing objects in a partial clone

branch List, create, or delete branches

commit Record changes to the repository

merge Join two or more development histories together

rebase Reapply commits on top of another base tip

reset Reset current HEAD to the specified state

switch Switch branches

tag Create, list, delete or verify a tag object signed with GPG

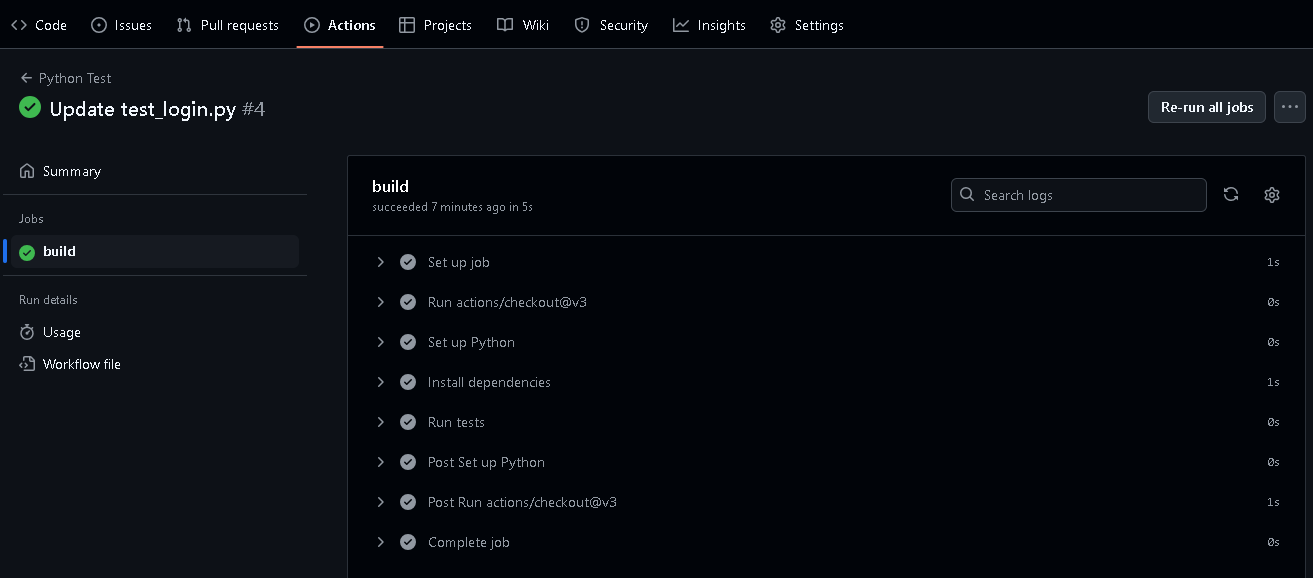
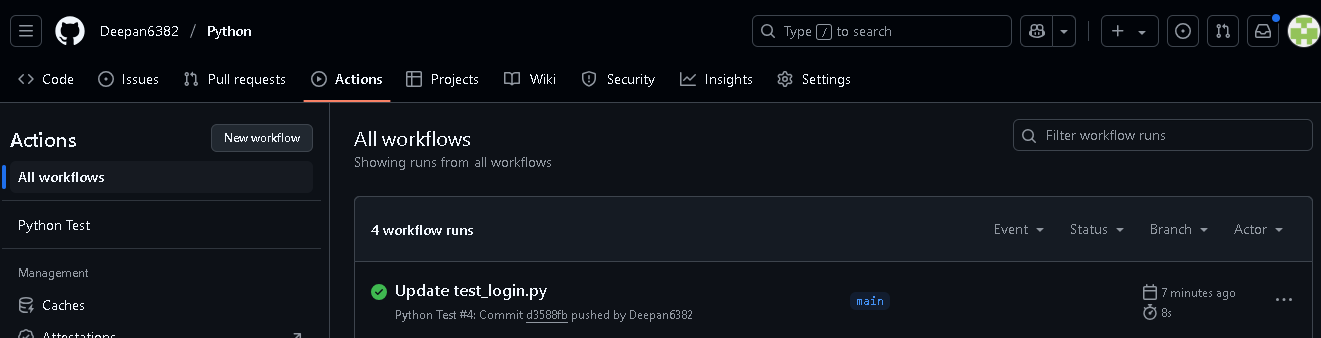
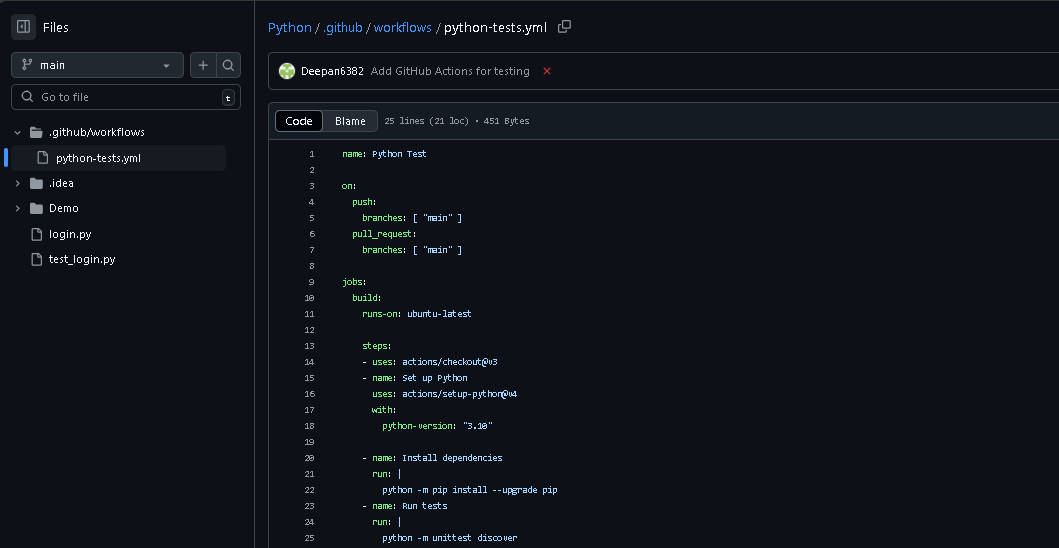
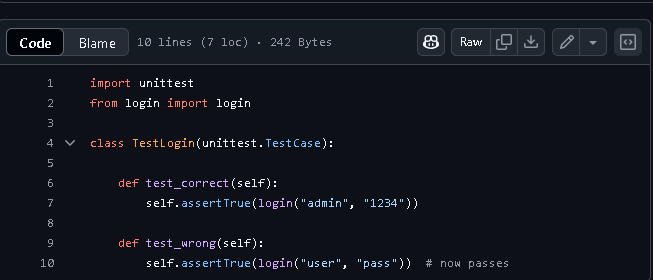
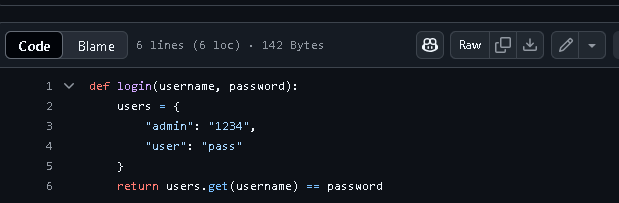
5. collaborate (see also: git help workflows)

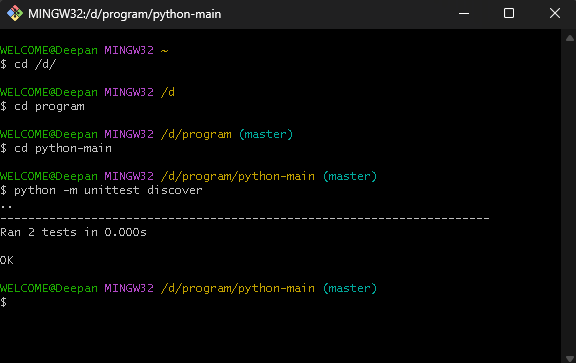
fetch Download objects and refs from another repository

pull Fetch from and integrate with another repository or a local branch

push Update remote refs along with associated objects

# 15. Simple GitHub Actions





# 16. Work on Robot Framework

File & Folder Structure

.

├── tests/

│ └── web\_test.robot ← Robot Framework test script

├── requirements.txt ← Python packages needed for the project

├── ci-cd.sh ← Bash script to automate local test execution

└── .github/

└── workflows/

└── robot\_web.yml ← GitHub Actions CI/CD pipeline configuration

1. web\_test.robot

This is your Robot Framework test case file.

Why use?

It defines **what you want to test** — for example, loading a webpage and checking the title.

2. Requirements.txt

This file lists **Python packages your project needs**.

Why use?

GitHub Actions or your Bash script will use it to install all required libraries in one step.

3. ci-cd.sh

A **Bash shell script** to install dependencies and run Robot Framework tests locally (in your terminal or PyCharm).

Why use?

Instead of manually typing install and run commands each time, this script automates it.

How to run locally in pycharm or terminal or Git bash

Bash ci-cd.sh

4. .github/workflows.robot\_web.yml

A **GitHub Actions workflow file** written in YAML. This is your **CI/CD pipeline definition**.

Why use?

Whenever you **push code to GitHub**, this file tells GitHub what steps to run:

* Install Python
* Install requirements
* Run Robot Framework tests

How it All works together

 You write your test case in web\_test.robot

 You define dependencies in requirements.txt

 You can run the whole pipeline locally via bash ci-cd.sh

 ☁️ When you **push your code to GitHub**, the .github/workflows/robot\_web.yml file tells GitHub to:

* Clone the repo
* Install everything
* Run the tests in a cloud runner (CI/CD)

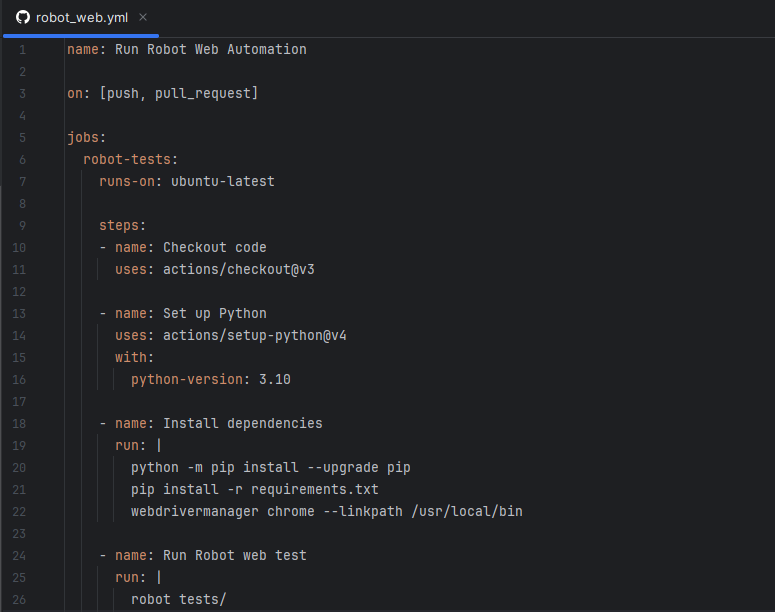
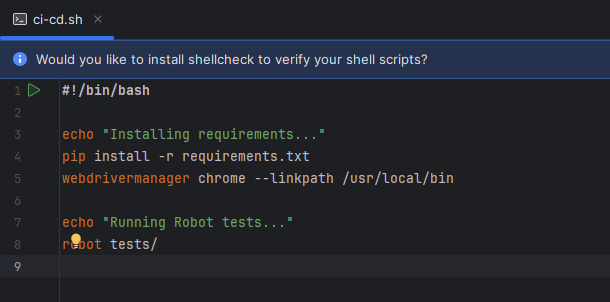
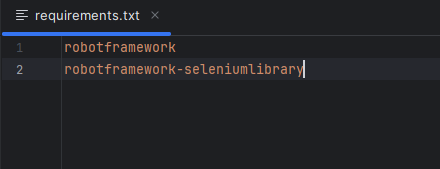
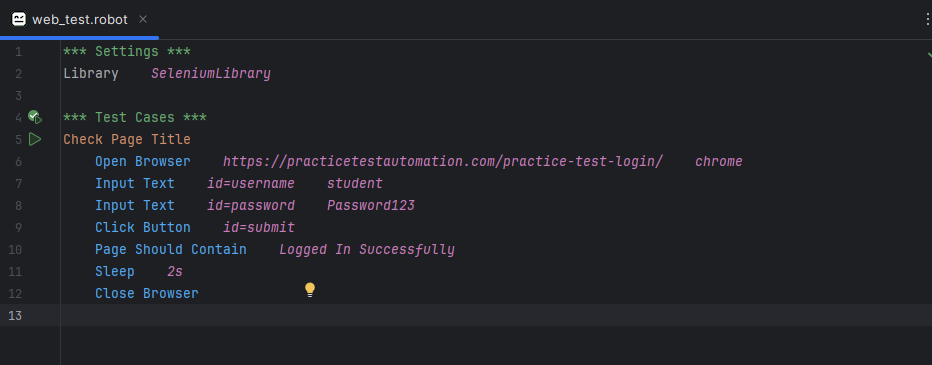
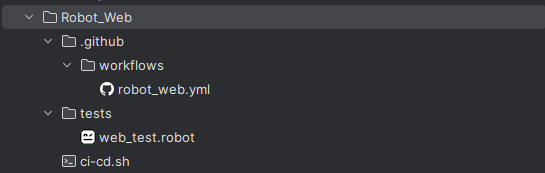
Final Result

 You can test websites automatically

 Your test is repeatable locally **and** in the cloud

 You can track test history in GitHub → **Actions tab**

Using Robot Framework



WELCOME@Deepan MINGW32 /d/program/github/robot (main)

$ cd ..

WELCOME@Deepan MINGW32 /d/program/github (main)

$ cd robot\_web

WELCOME@Deepan MINGW32 /d/program/github/robot\_web (main)

$ bash ci-cd.sh

Installing requirements...

Requirement already satisfied: robotframework in c:\users\welcome\appdata\local\programs\python\python312\lib\site-packages (from -r requirements.txt (line 1)) (7.2.2)

Requirement already satisfied: robotframework-seleniumlibrary in c:\users\welcome\appdata\local\programs\python\python312\lib\site-packages (from -r requirements.txt (line 2)) (6.7.1)

Requirement already satisfied: selenium>=4.3.0 in c:\users\welcome\appdata\local\programs\python\python312\lib\site-packages (from robotframework-seleniumlibrary->-r requirements.txt (line 2)) (4.31.0)

Requirement already satisfied: robotframework-pythonlibcore>=4.4.1 in c:\users\welcome\appdata\local\programs\python\python312\lib\site-packages (from robotframework-seleniumlibrary->-r requirements.txt (line 2)) (4.4.1)

Requirement already satisfied: click>=8.0 in c:\users\welcome\appdata\local\programs\python\python312\lib\site-packages (from robotframework-seleniumlibrary->-r requirements.txt (line 2)) (8.1.7)

Requirement already satisfied: colorama in c:\users\welcome\appdata\local\programs\python\python312\lib\site-packages (from click>=8.0->robotframework-seleniumlibrary->-r requirements.txt (line 2)) (0.4.6)

Requirement already satisfied: urllib3<3,>=1.26 in c:\users\welcome\appdata\local\programs\python\python312\lib\site-packages (from urllib3[socks]<3,>=1.26->selenium>=4.3.0->robotframework-seleniumlibrary->-r requirements.txt (line 2)) (2.2.3)

Requirement already satisfied: trio~=0.17 in c:\users\welcome\appdata\local\programs\python\python312\lib\site-packages (from selenium>=4.3.0->robotframework-seleniumlibrary->-r requirements.txt (line 2)) (0.29.0)

Requirement already satisfied: trio-websocket~=0.9 in c:\users\welcome\appdata\local\programs\python\python312\lib\site-packages (from selenium>=4.3.0->robotframework-seleniumlibrary->-r requirements.txt (line 2)) (0.12.2)

Requirement already satisfied: certifi>=2021.10.8 in c:\users\welcome\appdata\local\programs\python\python312\lib\site-packages (from selenium>=4.3.0->robotframework-seleniumlibrary->-r requirements.txt (line 2)) (2024.8.30)

Requirement already satisfied: typing\_extensions~=4.9 in c:\users\welcome\appdata\local\programs\python\python312\lib\site-packages (from selenium>=4.3.0->robotframework-seleniumlibrary->-r requirements.txt (line 2)) (4.12.2)

Requirement already satisfied: websocket-client~=1.8 in c:\users\welcome\appdata\local\programs\python\python312\lib\site-packages (from selenium>=4.3.0->robotframework-seleniumlibrary->-r requirements.txt (line 2)) (1.8.0)

Requirement already satisfied: attrs>=23.2.0 in c:\users\welcome\appdata\local\programs\python\python312\lib\site-packages (from trio~=0.17->selenium>=4.3.0->robotframework-seleniumlibrary->-r requirements.txt (line 2)) (24.2.0)

Requirement already satisfied: sortedcontainers in c:\users\welcome\appdata\local\programs\python\python312\lib\site-packages (from trio~=0.17->selenium>=4.3.0->robotframework-seleniumlibrary->-r requirements.txt (line 2)) (2.4.0)

Requirement already satisfied: idna in c:\users\welcome\appdata\local\programs\python\python312\lib\site-packages (from trio~=0.17->selenium>=4.3.0->robotframework-seleniumlibrary->-r requirements.txt (line 2)) (3.10)

Requirement already satisfied: outcome in c:\users\welcome\appdata\local\programs\python\python312\lib\site-packages (from trio~=0.17->selenium>=4.3.0->robotframework-seleniumlibrary->-r requirements.txt (line 2)) (1.3.0.post0)

Requirement already satisfied: sniffio>=1.3.0 in c:\users\welcome\appdata\local\programs\python\python312\lib\site-packages (from trio~=0.17->selenium>=4.3.0->robotframework-seleniumlibrary->-r requirements.txt (line 2)) (1.3.1)

Requirement already satisfied: cffi>=1.14 in c:\users\welcome\appdata\local\programs\python\python312\lib\site-packages (from trio~=0.17->selenium>=4.3.0->robotframework-seleniumlibrary->-r requirements.txt (line 2)) (1.17.1)

Requirement already satisfied: wsproto>=0.14 in c:\users\welcome\appdata\local\programs\python\python312\lib\site-packages (from trio-websocket~=0.9->selenium>=4.3.0->robotframework-seleniumlibrary->-r requirements.txt (line 2)) (1.2.0)

Requirement already satisfied: pysocks!=1.5.7,<2.0,>=1.5.6 in c:\users\welcome\appdata\local\programs\python\python312\lib\site-packages (from urllib3[socks]<3,>=1.26->selenium>=4.3.0->robotframework-seleniumlibrary->-r requirements.txt (line 2)) (1.7.1)

Requirement already satisfied: pycparser in c:\users\welcome\appdata\local\programs\python\python312\lib\site-packages (from cffi>=1.14->trio~=0.17->selenium>=4.3.0->robotframework-seleniumlibrary->-r requirements.txt (line 2)) (2.22)

Requirement already satisfied: h11<1,>=0.9.0 in c:\users\welcome\appdata\local\programs\python\python312\lib\site-packages (from wsproto>=0.14->trio-websocket~=0.9->selenium>=4.3.0->robotframework-seleniumlibrary->-r requirements.txt (line 2)) (0.14.0)

ci-cd.sh: line 5: webdrivermanager: command not found

Running Robot tests...

==============================================================================

Tests

==============================================================================

Tests.Web Test

==============================================================================

Check Page Title | PASS |

------------------------------------------------------------------------------

Tests.Web Test | PASS |

1 test, 1 passed, 0 failed

==============================================================================

Tests | PASS |

1 test, 1 passed, 0 failed

==============================================================================

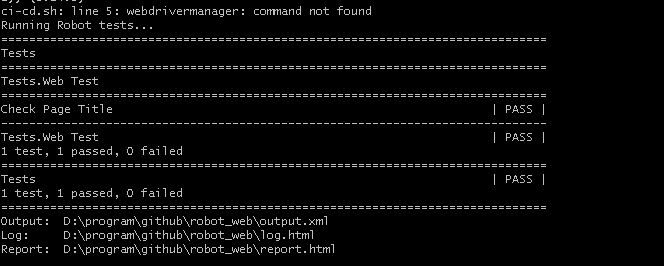
Output: D:\program\github\robot\_web\output.xml

Log: D:\program\github\robot\_web\log.html

Report: D:\program\github\robot\_web\report.html

WELCOME@Deepan MINGW32 /d/program/github/robot\_web (main)

$



git branch branch3

git checkout branch3

git branch

git pull

or

git pull origin main

git push --set-upstream origin branch3

or

git push origin branch3

D:\Program\GitHub>mkdir Second

D:\Program\GitHub>cd second

D:\Program\GitHub\Second>echo This is second message > second.txt

D:\Program\GitHub\Second>git init

Initialized empty Git repository in D:/Program/GitHub/Second/.git/

D:\Program\GitHub\Second>git add second.txt

D:\Program\GitHub\Second>git commit -m "second"

[master (root-commit) 361c8aa] second

1 file changed, 1 insertion(+)

create mode 100644 second.txt

D:\Program\GitHub\Second>gh repo create --public --source=. --remote=origin --push

✓ Created repository Deepan6382/Second on github.com

https://github.com/Deepan6382/Second

✓ Added remote https://github.com/Deepan6382/Second.git

Enumerating objects: 3, done.

Counting objects: 100% (3/3), done.

Writing objects: 100% (3/3), 240 bytes | 60.00 KiB/s, done.

Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)