Git

1. What is Global information tracker (Git)

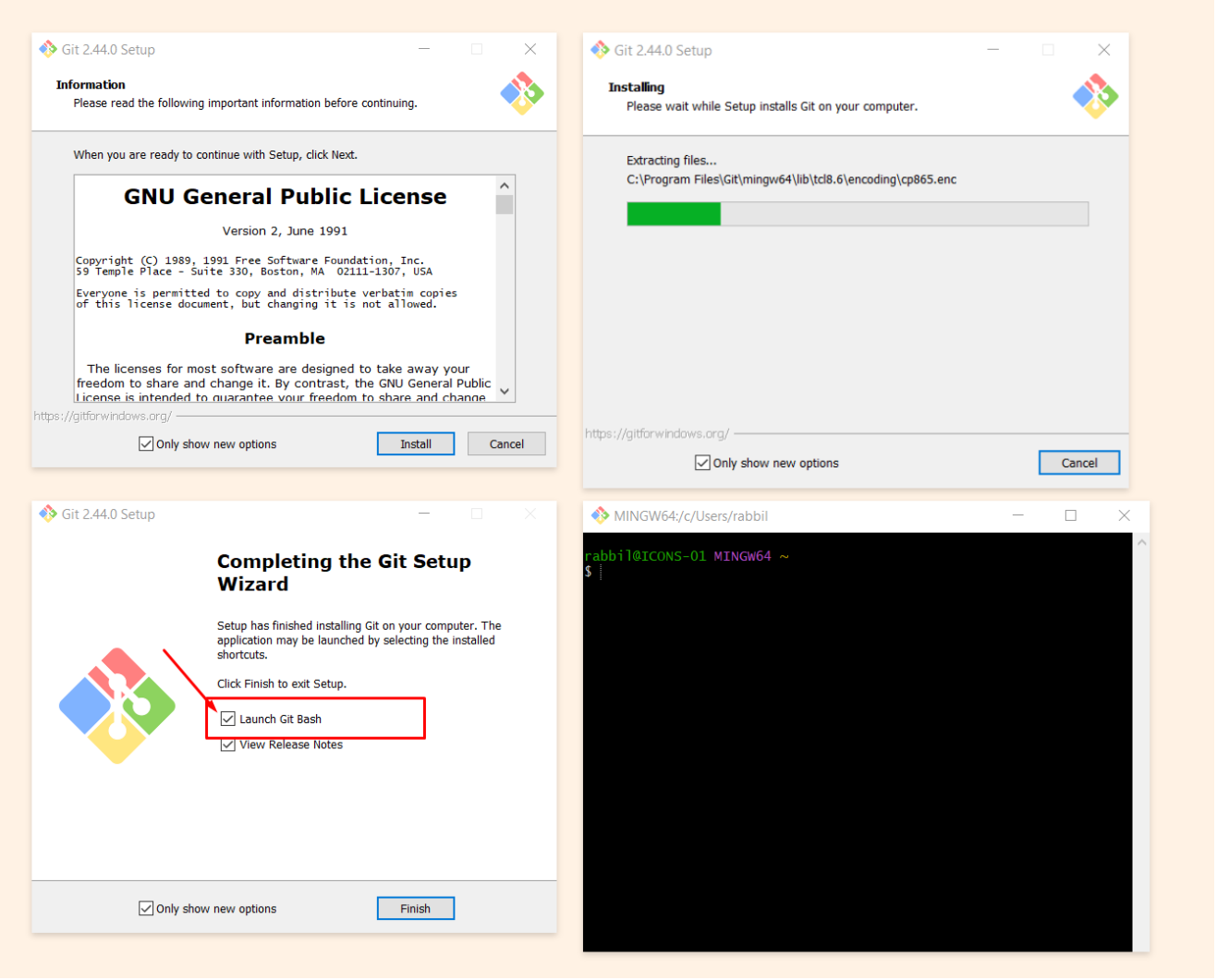
Git is a tool that helps developers work together on projects by keeping track of changes to files. It makes teamwork smoother and allows for better organization of code.

2. Use Case

1. **Version Control**: Git helps track changes made to files over time, allowing developers to revert to previous versions if needed, ensuring code stability and history tracking.
2. **Collaboration**: Git enables multiple developers to work on the same project simultaneously, managing conflicts and merging changes seamlessly, facilitating teamwork.
3. **Backup and Recovery**: Git serves as a backup system, storing project code on remote repositories like GitHub, GitLab, or Bitbucket, ensuring data safety and providing a platform for disaster recovery.

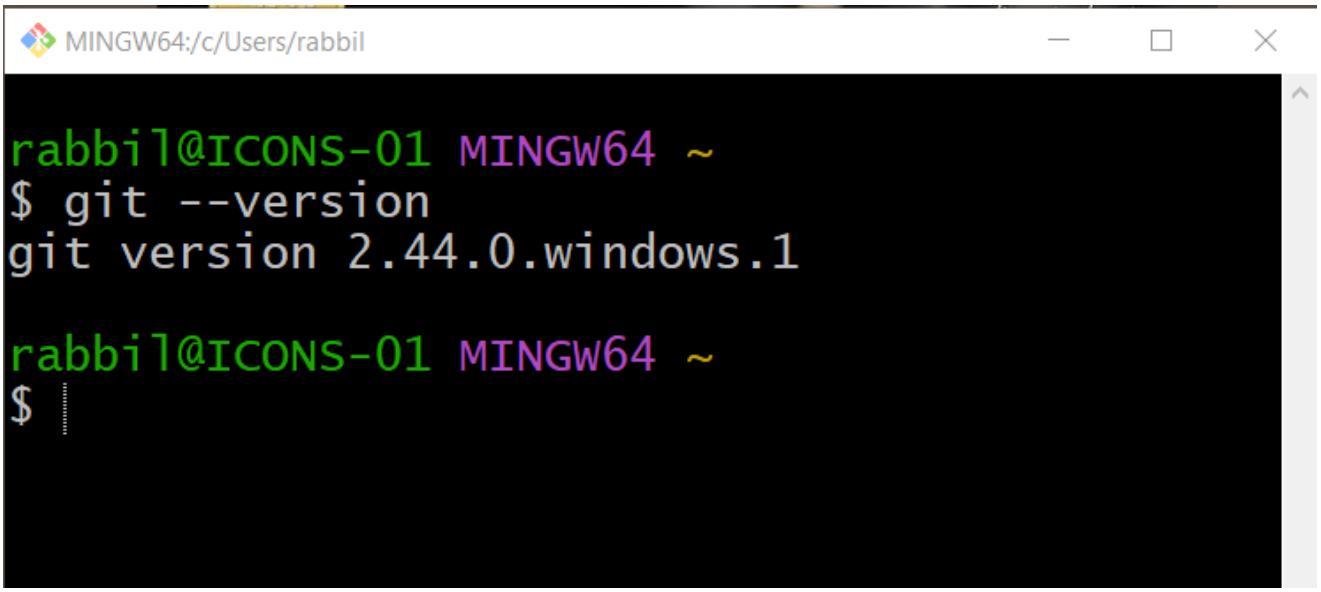
3. Git Download & Install

Download Link https://www.git-scm.com/



4. Your First Git Command

```
git --version
```

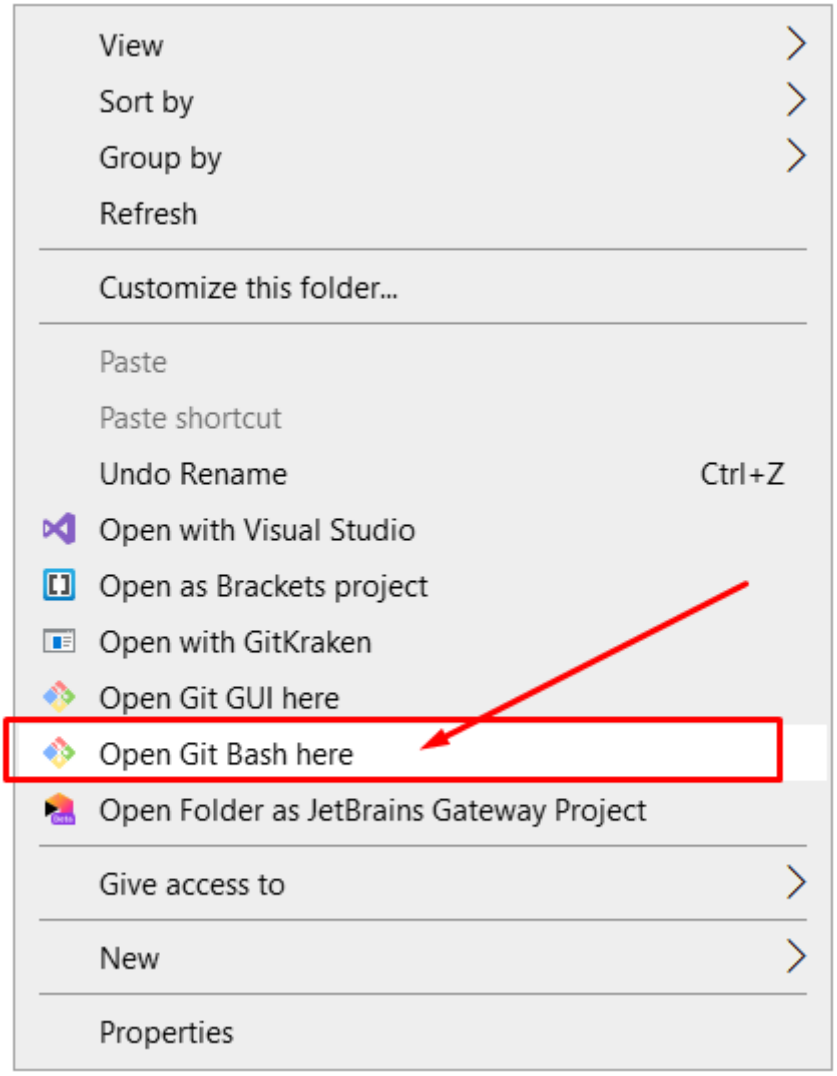


'

5. Configure Git that who your are ?

```
git config --global user.name "rabbil"
git config --global user.email "mrrabbilhasan@gmail.com"
```

6. Create your first local git repository

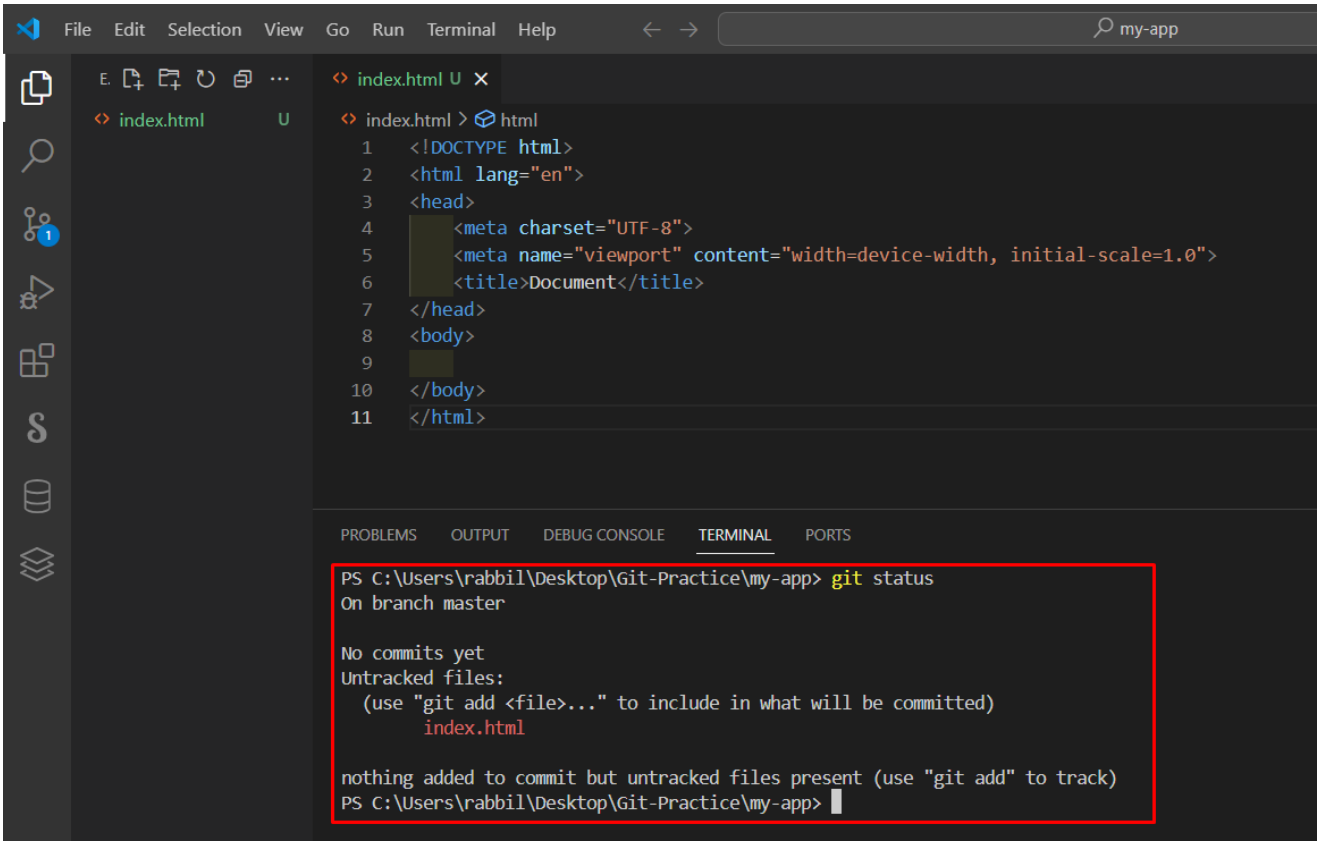


```
mkdir my-app
cd my-app
git init
```

7. Lets create your first coding project & start version controlling

⬛ STEP-01: Understanding Git Status Tracked & Untracked Files

- Open `my-app` folder in vs code
- Create simple html file `index.html`
- Lets check your project git status `git status`

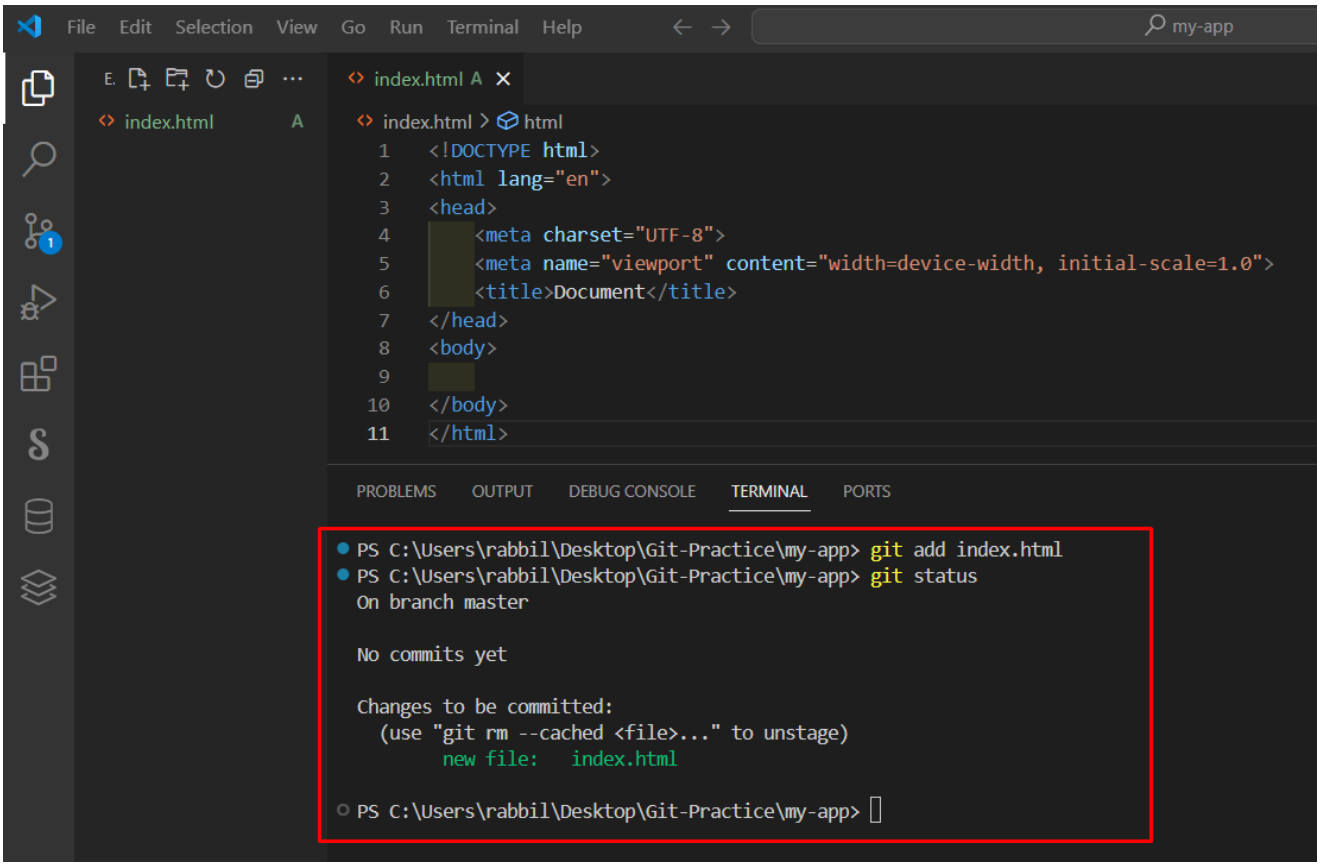


- Files in your Git repository folder can be in one of **2 states**
- **Tracked** - files that Git knows about and are added to the repository
- **Untracked** - files that are in your working directory, but not added to the repository
- When you first add files to an empty repository, they are all untracked
- To get Git to track them, you need to add them

⬛ STEP-02: Understanding Git Staging

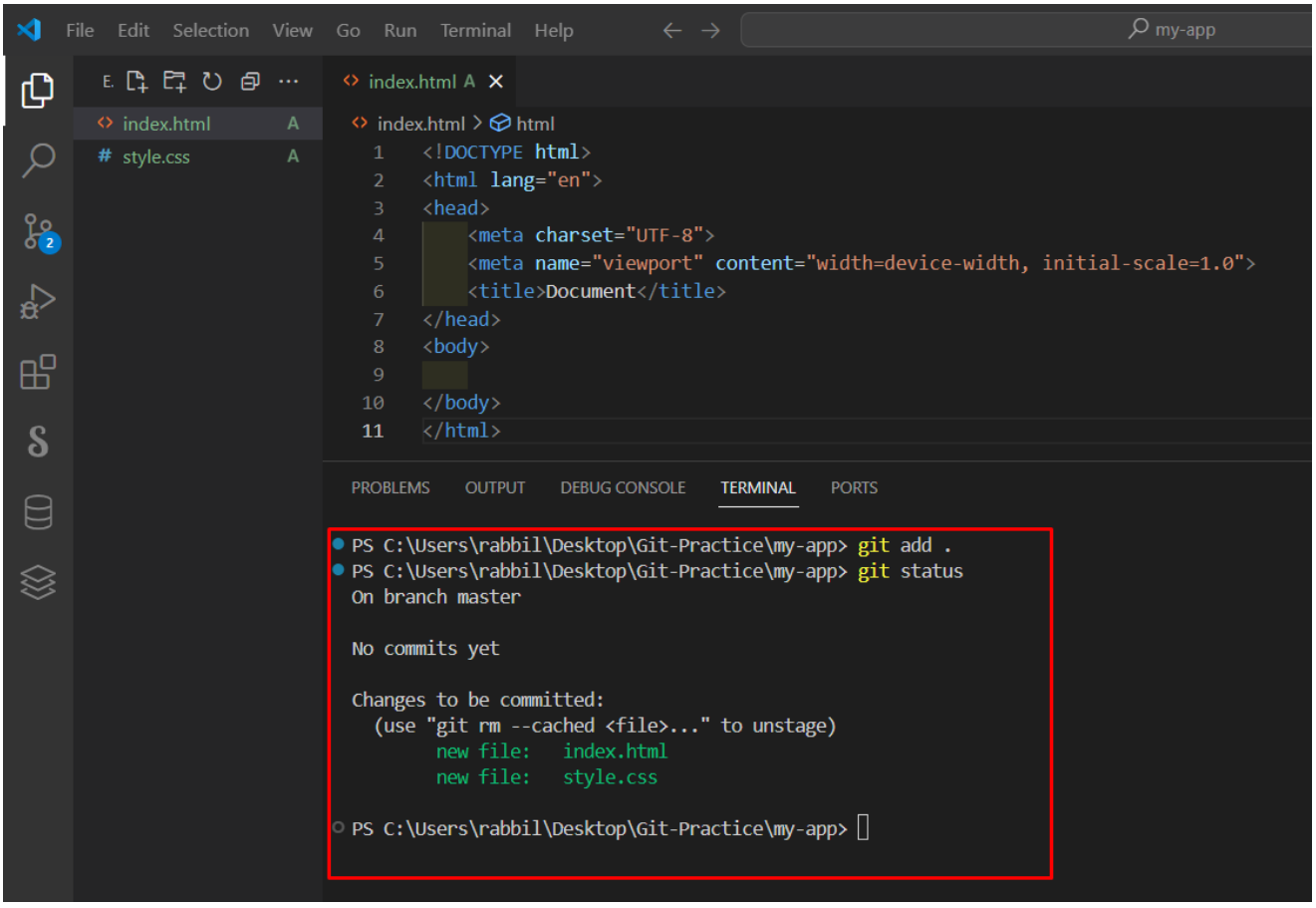
- To add specific files or directories, you would typically follow the `git add` command with the names of the files or directories you want to add.

```
git add index.html
```



- If you want to add all changes in the current directory and its subdirectories, you can use

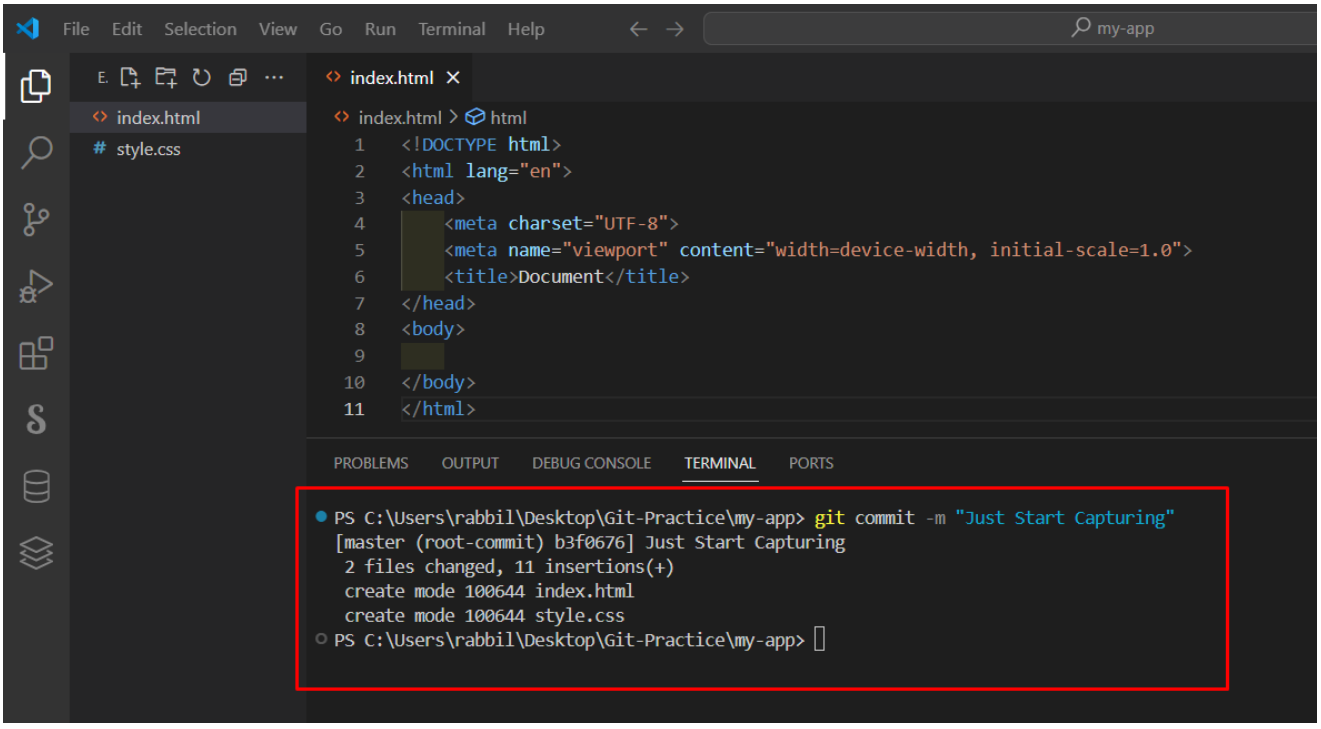
```
git add .
// or
git add --all
```



■ STEP-03: Understanding Git Commit

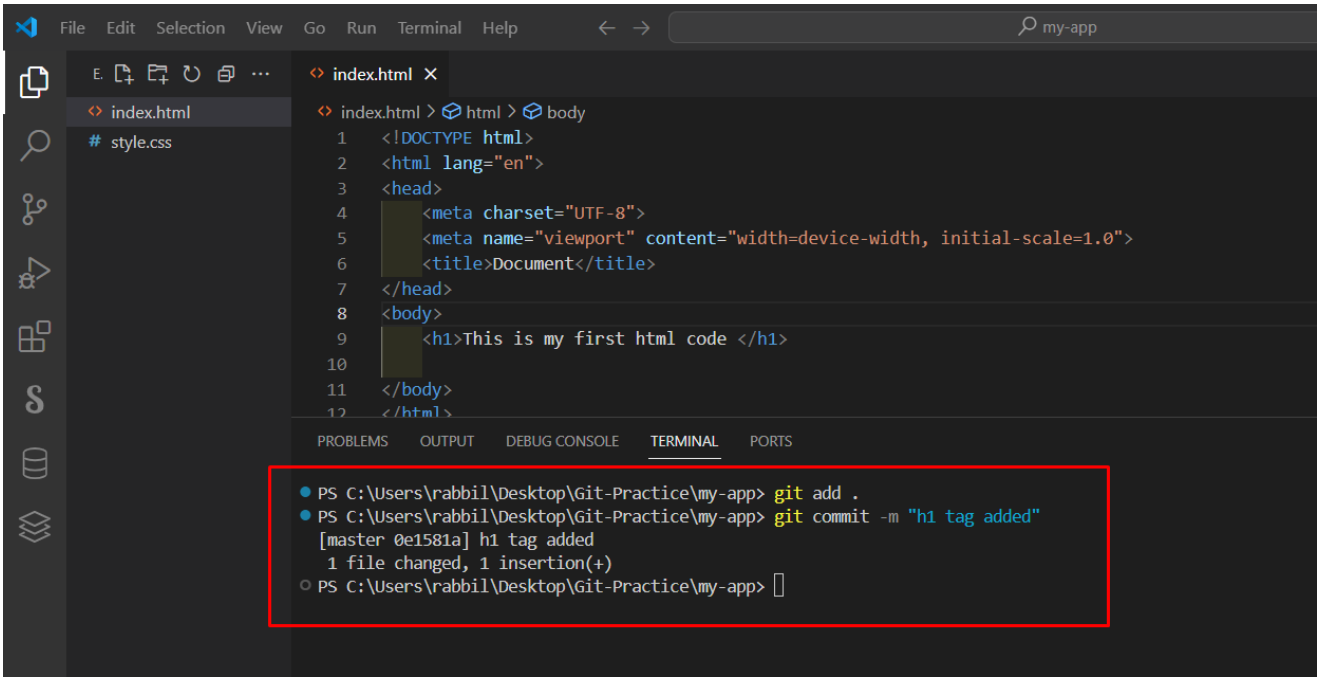
- A Git commit is like taking a photo of your project at a specific moment. It captures all the changes you've made, who made them, and a short message explaining what was done. It's like saving a checkpoint in your project's history.

```
git commit -m "Just Start Capturing"
```



- Make some changes capture again

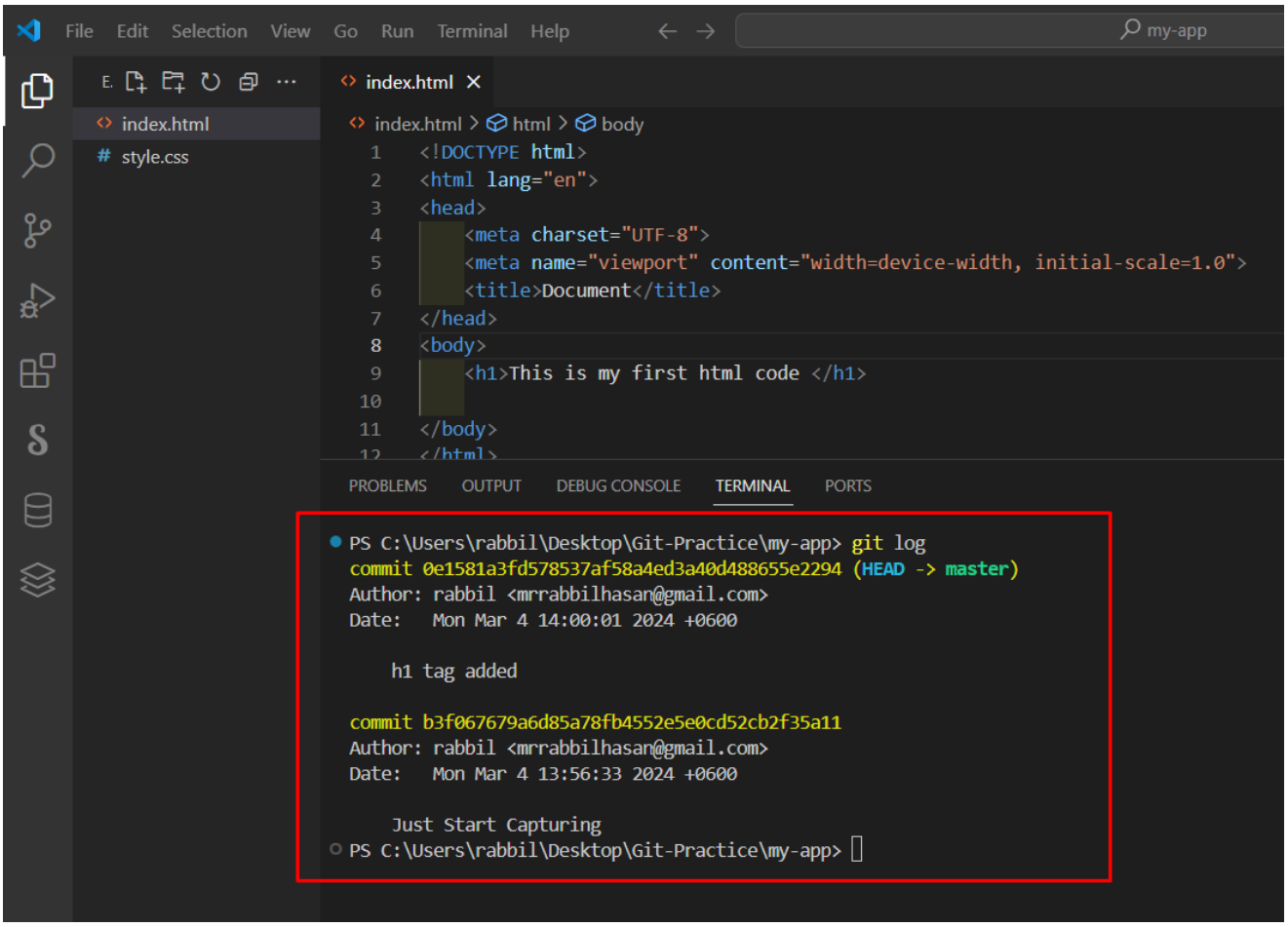
```
git add .
git commit -m "h1 tag added"
```



■ STEP-04: Understanding Git Log

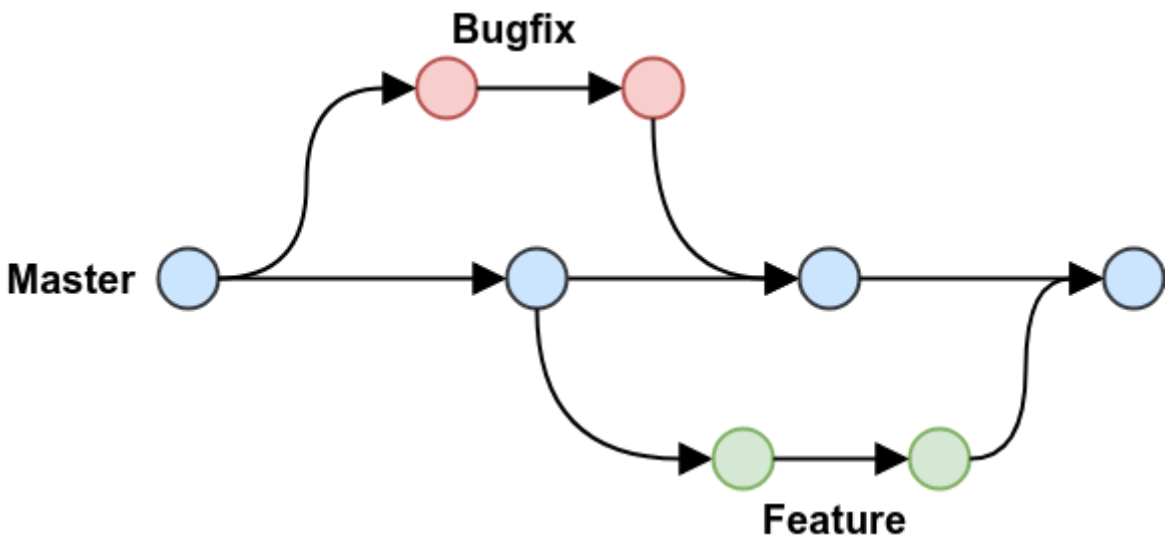
Command used to view the history of captures in a Git repository.

```
git log
```



8. Lets Understand Branch

Branch is a new/separate version of the main repository.

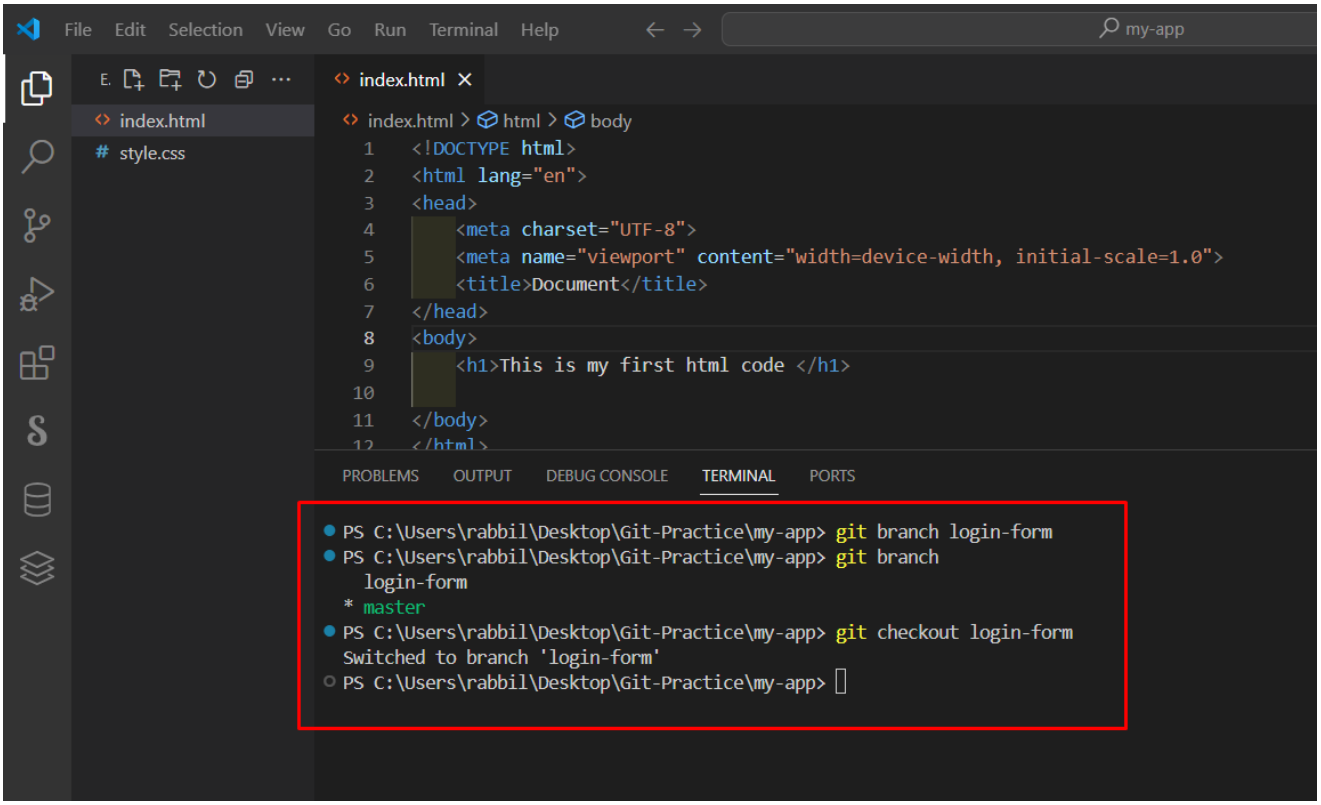


1. **Feature Development**: Creating a branch to work on a new feature or enhancement without disrupting the main codebase.
2. **Bug Fixes**: Isolating bug fixes in separate branches to ensure they can be tested and deployed independently.
3. **Experimentation**: Trying out new ideas or approaches without altering the main project until they're proven successful.
4. **Versioning**: Maintaining different versions of the project for different purposes (e.g., stable releases, development versions

9. Playing with branch

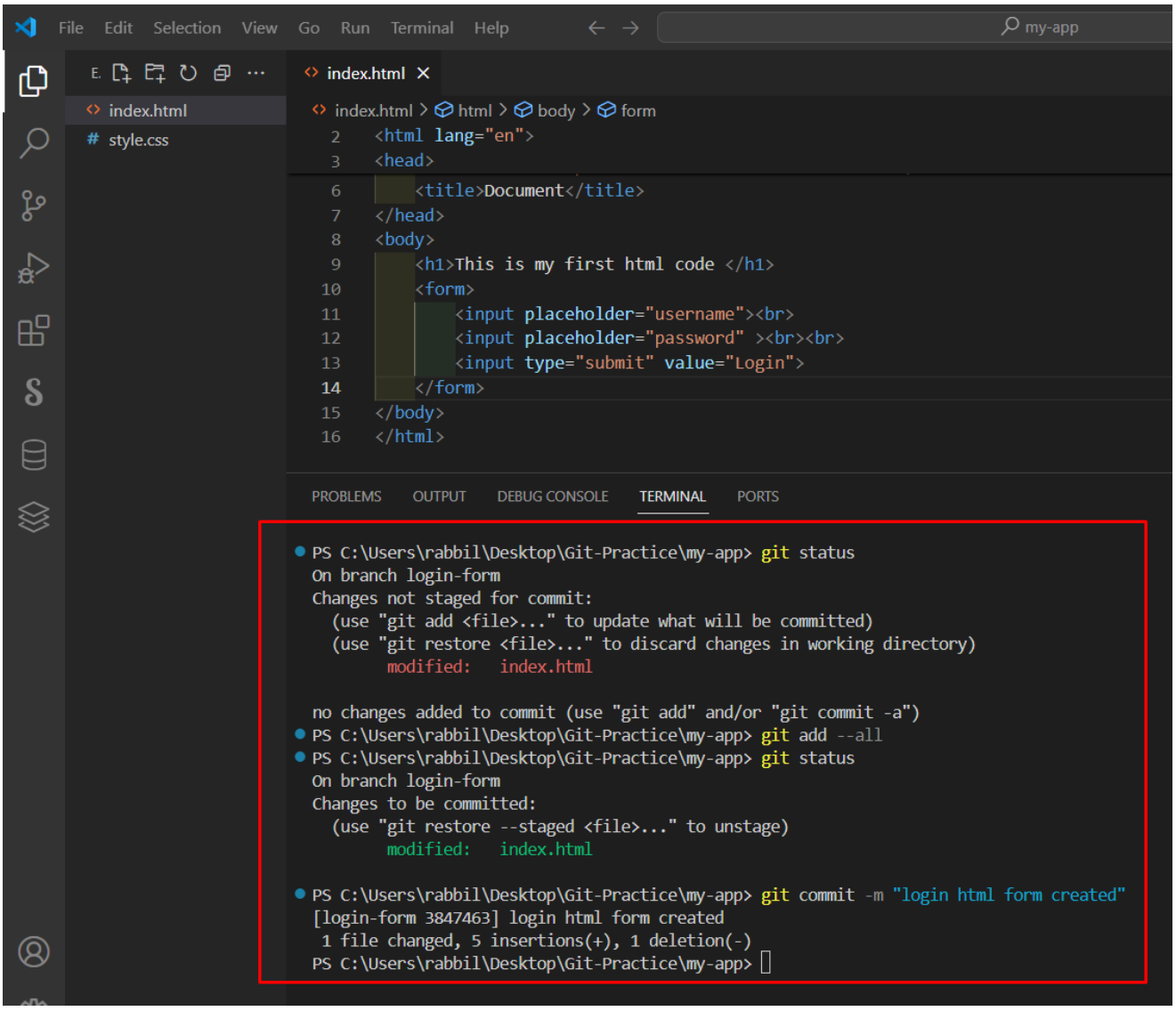
■ **STEP-01: Create new branch & confirm that is created & switching to new branch**

```
git branch login-form
git branch
git checkout login-form
```



■ **STEP-02: Say I added a login form feature inside `index.html`

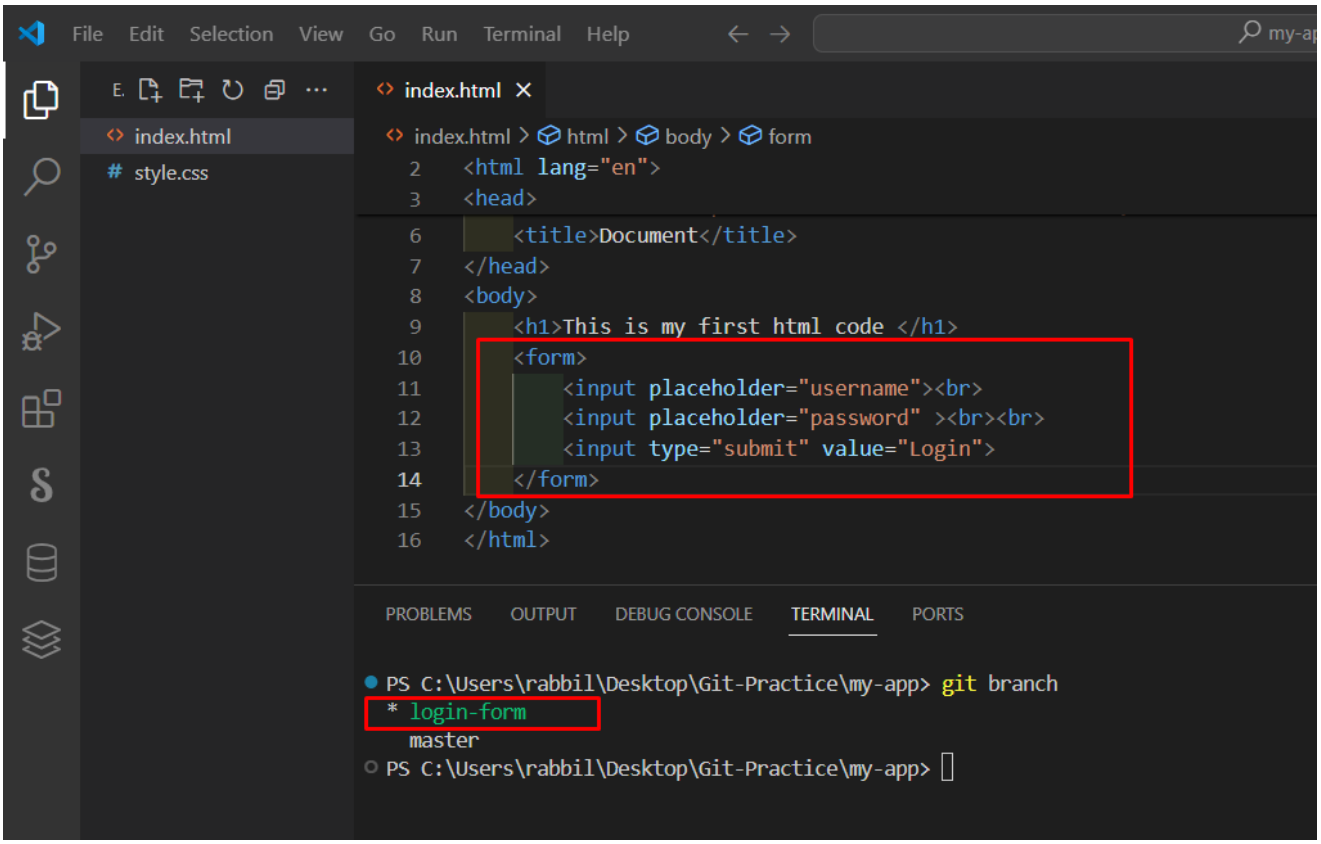
```
git status
git add --all
git status
git commit -m "login html form created"
```



■ **STEP-03: Now discover `index.html` at `main` branch & `login-form` branch

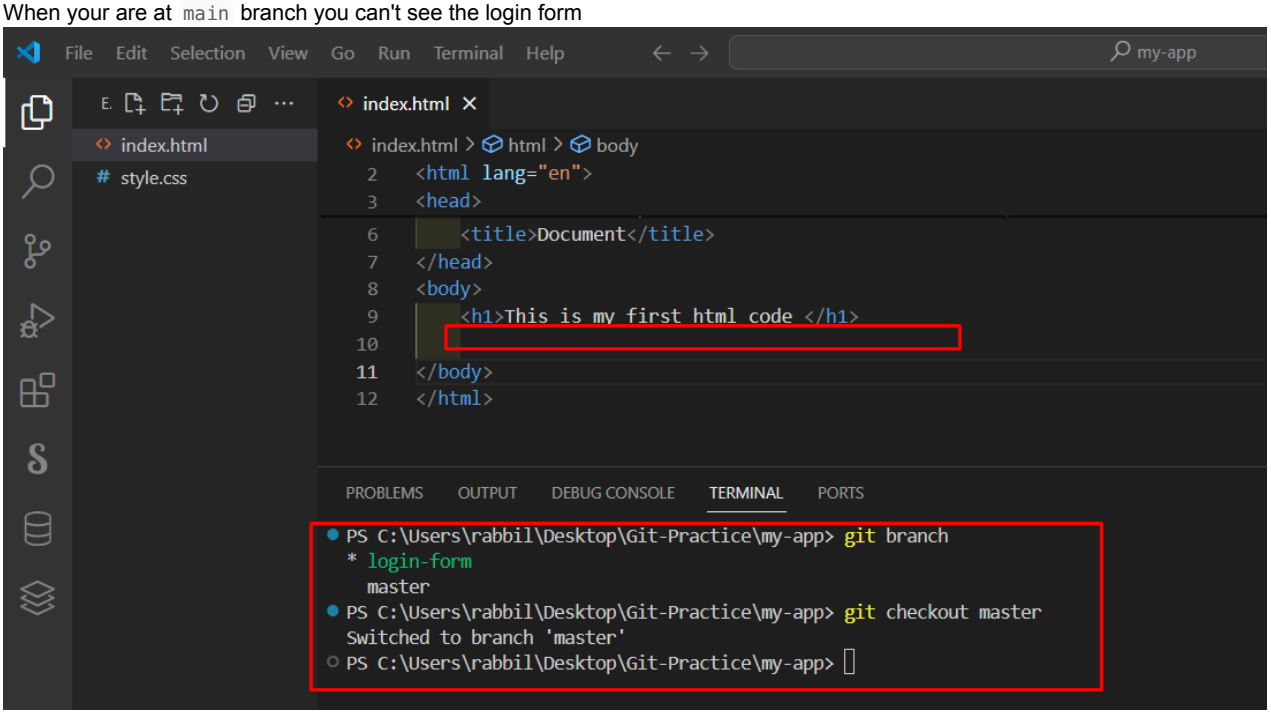
```
git branch
```

- When your are at `login-form` branch you can see the login form



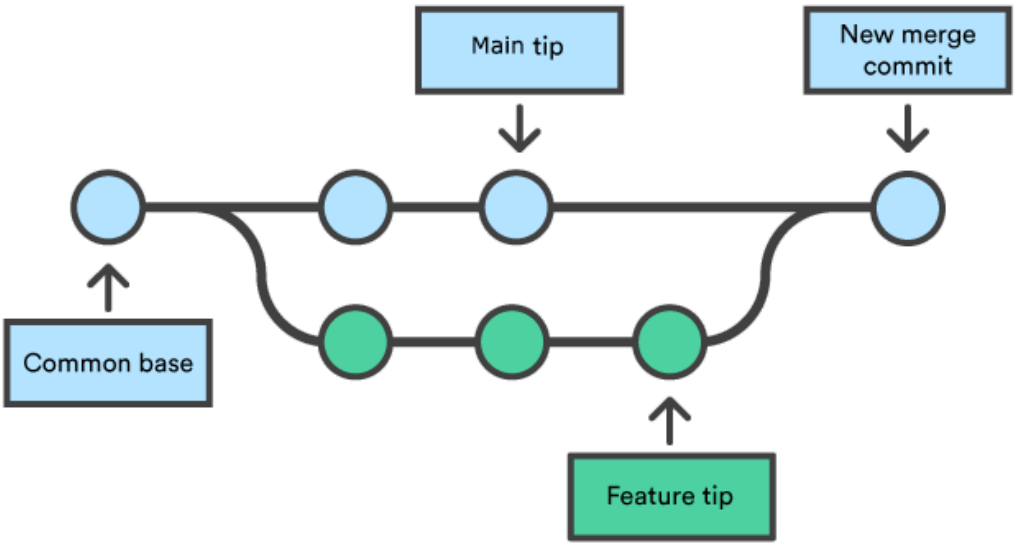

```
git checkout master
```

- When your are at `main` branch you can't see the login form



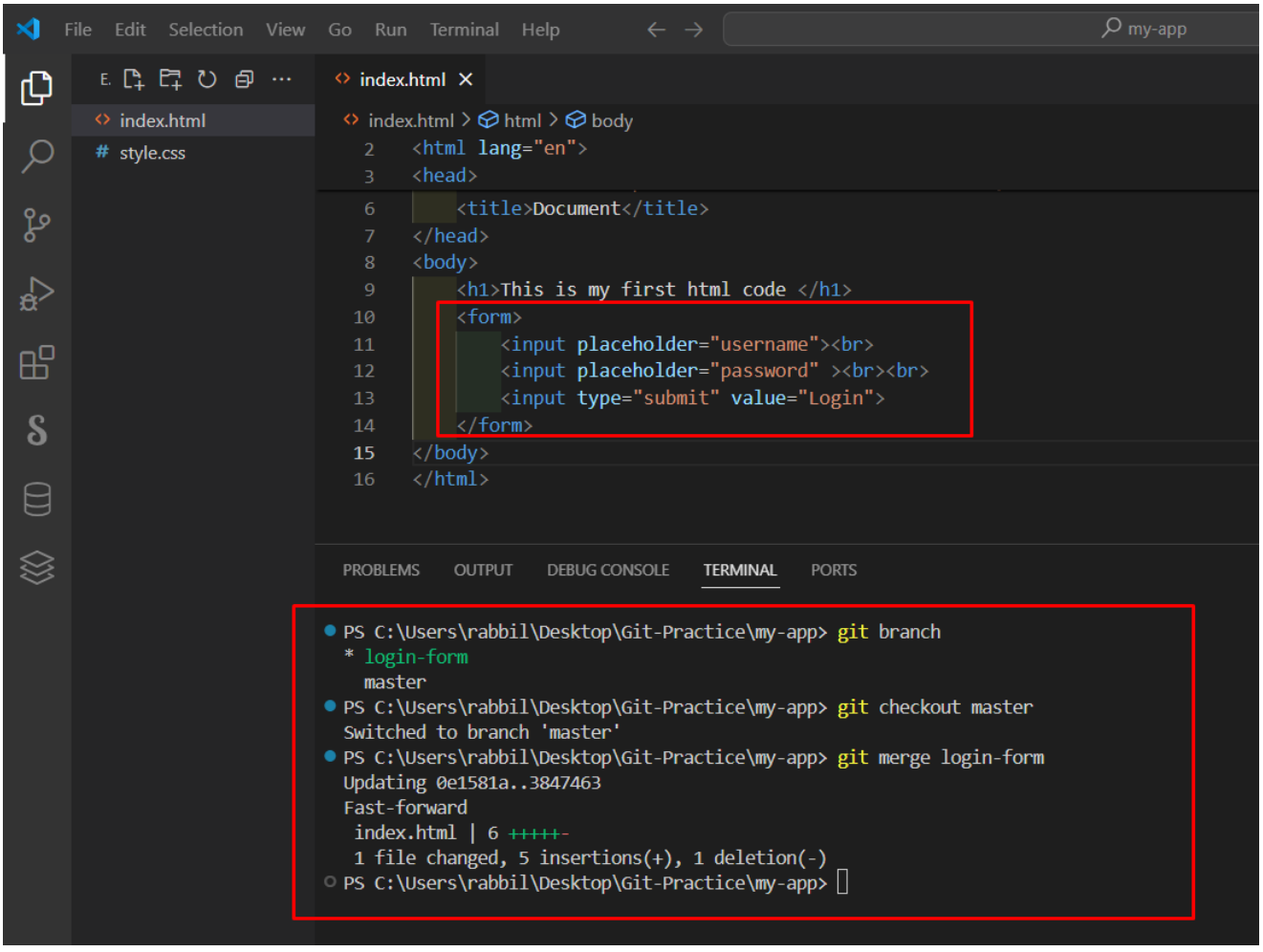
10. Git Branch Merge

Git branch merge refers to the process of combining the changes from one branch into another. This is typically done to incorporate the changes made in a feature branch (or any other branch) back into the main branch of the repository, such as `master` or `main`.



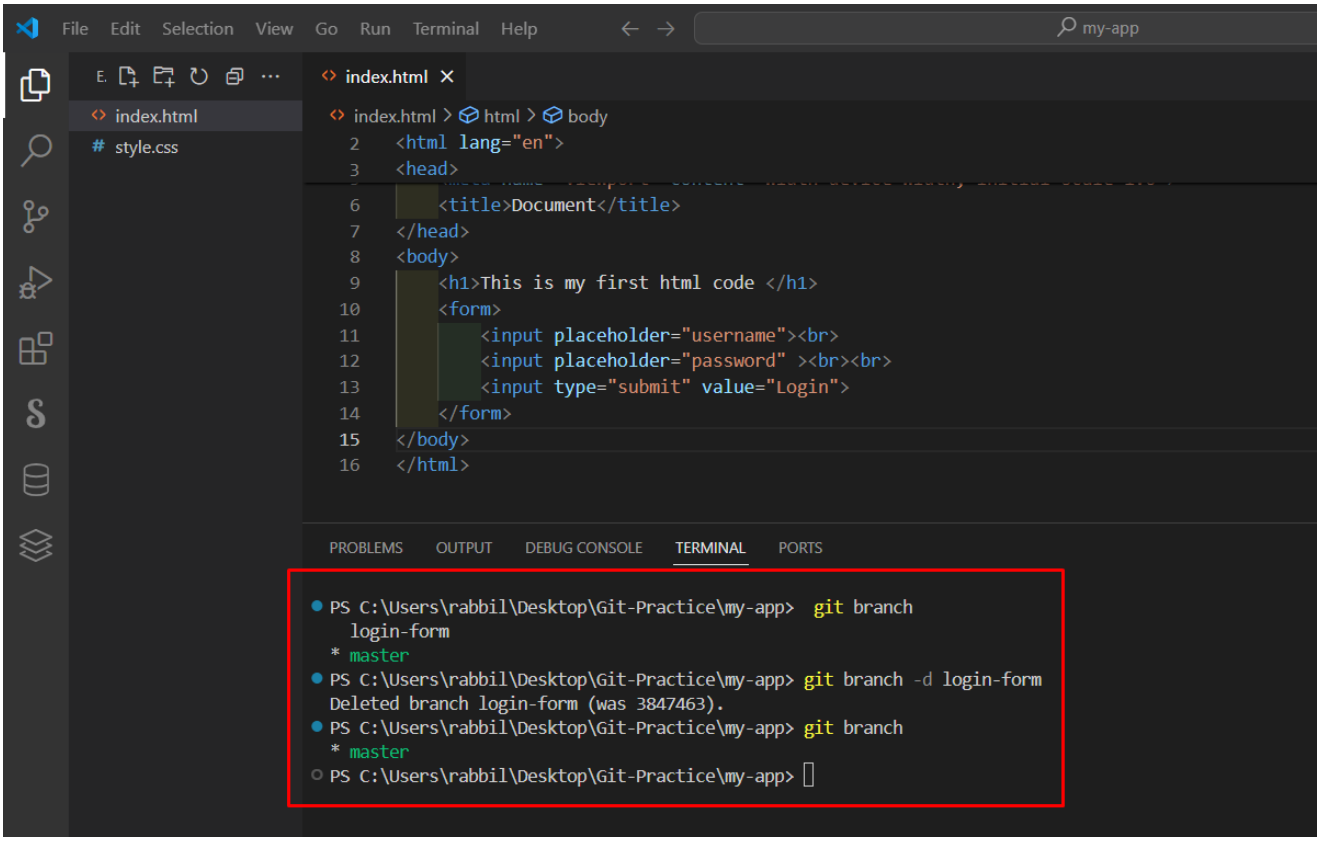
STEP: 01 Let merge the login from html codes from `login-form` branch to `main` branch

```
git branch
git checkout master
git merge login-form
```



STEP: 02 Delete the `login-form` branch after merge

```
git branch
git branch -d login-form
git branch
```



Git Ignore and .gitignore

`.gitignore` is a file used in Git repositories to specify intentionally untracked files and directories that Git should ignore.

- Log files
- Vendor files
- Node modules
- Temporary files
- Hidden files

```
touch .gitignore
```

Rules for .gitignore

- **File and directory matching**: You can specify individual files, directories, or patterns using wildcard characters.
  - Example: `logs/` ignores the `logs` directory.
  - Example: `*.log` ignores all files with the `.log` extension.
  - Example: `build/` ignores all directories named `build`.
- **Negation**: You can use negation to exclude specific files or directories that would otherwise match a pattern.
  - Example: `!important.log` ignores all `.log` files except `important.log`
- **Comments**: Lines starting with `#` are considered comments and are ignored.
  - Example: `# Ignore compiled files` is a comment.
- **Ignore a specific file**: If you want to ignore a file named `example.txt`, just add this line to your `.gitignore` file:
  - Example: write `example.txt` inside