Subject Name: **Source Code Management**

Subject Code: **24CSE0106**

Session: **2024-25**

Department: **CSE**

****

|  |  |  |
| --- | --- | --- |
| **Submitted By:**  Itishjot Singh  2410990366  G5 |  | **Submitted To:**  Dr. Chetna Sharma |

**List of Programs**

|  |  |  |
| --- | --- | --- |
| **S. No** | **Program Title** | **Page No.** |
| 1 | To install and configure Git Client on your local system | 3-5 |
| 2 | Setting up GitHub account ad adding collaborators on GitHub Repository | 6-9 |
| 3 | To merge two branches with a Git repository | 10-13 |
| 4 | To demonstrate push and pull operations in Git | 14-17 |

|  |  |  |
| --- | --- | --- |
|  |  |  |

**Task 1.1**

**Practical 1**

**Aim:** To install and configure Git Client on your local system.

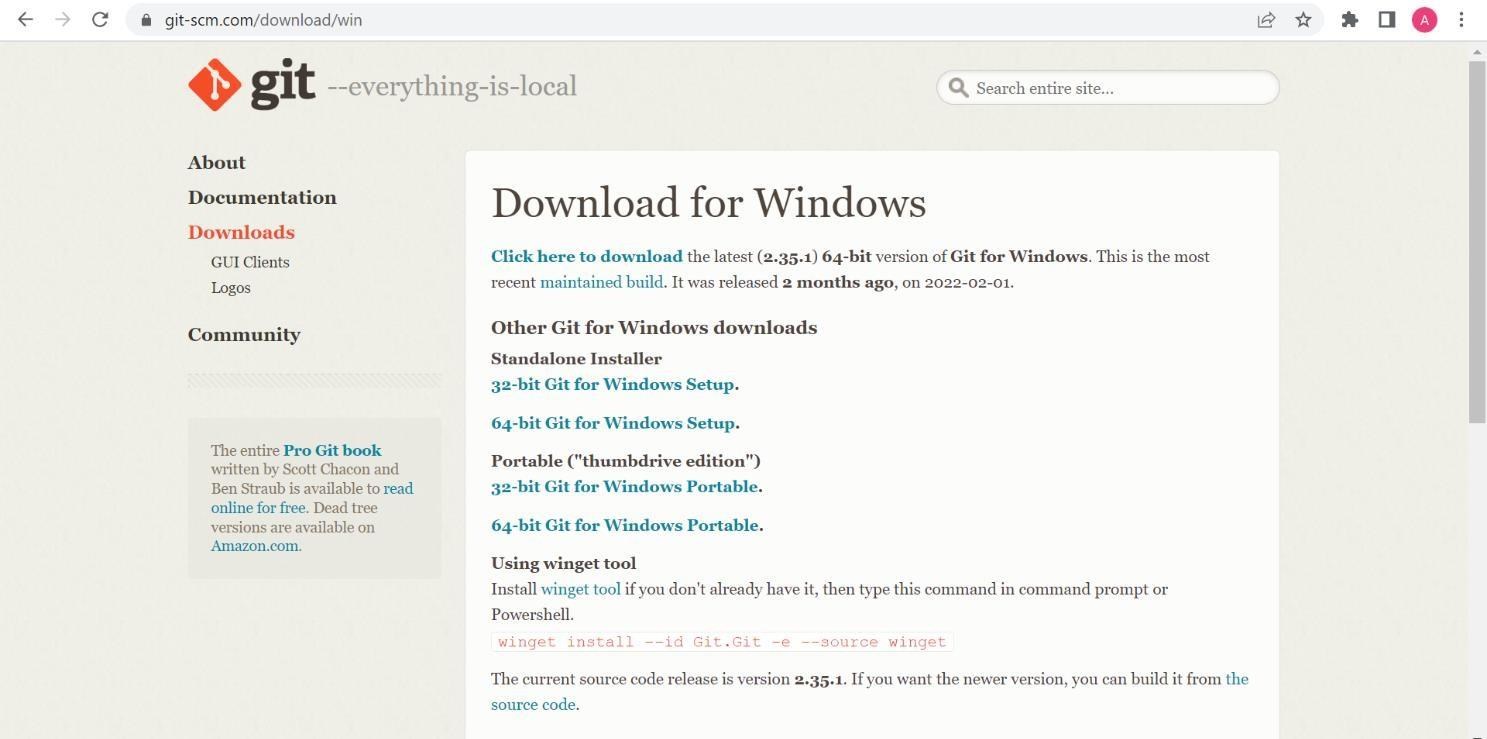
**Theory:**

Git is a distributed version control system used to track changes in source code. This practical focuses on setting up Git on your local system for effective version control.

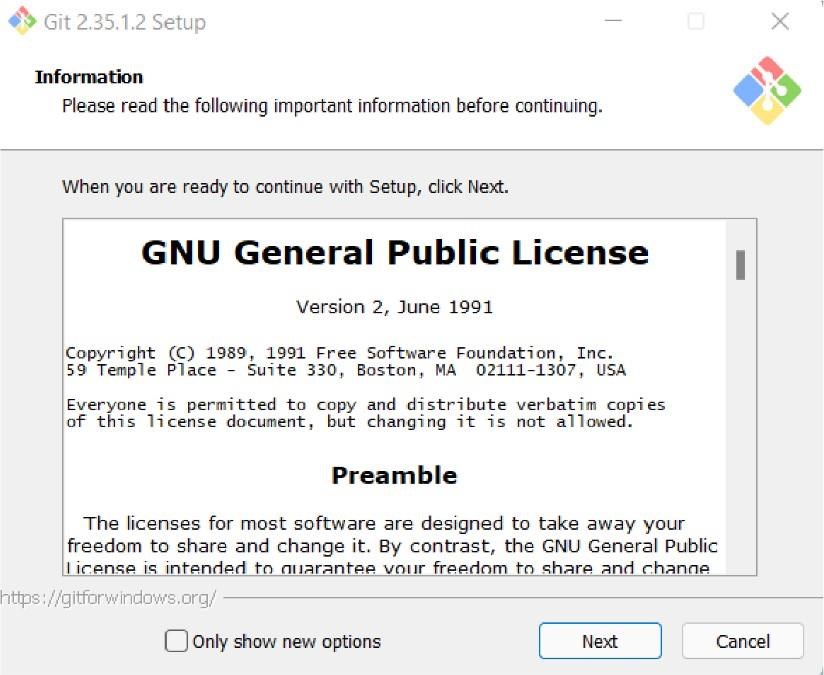
**Procedure:**

○ Download Git from git-scm.com.  
○ Install Git by following the setup wizard.  
○ Open Git Bash and verify installation using the command: git --version.  
○ Configure user details using the commands:  
git config --global user.name "Your Name"  
git config --global user.email "Your Email"

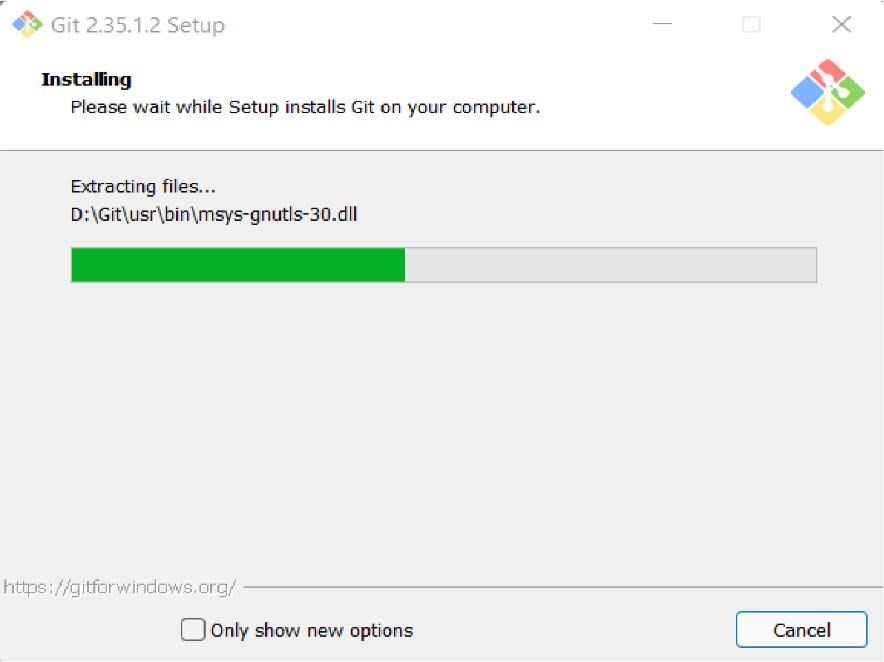
**Snapshots of download:**



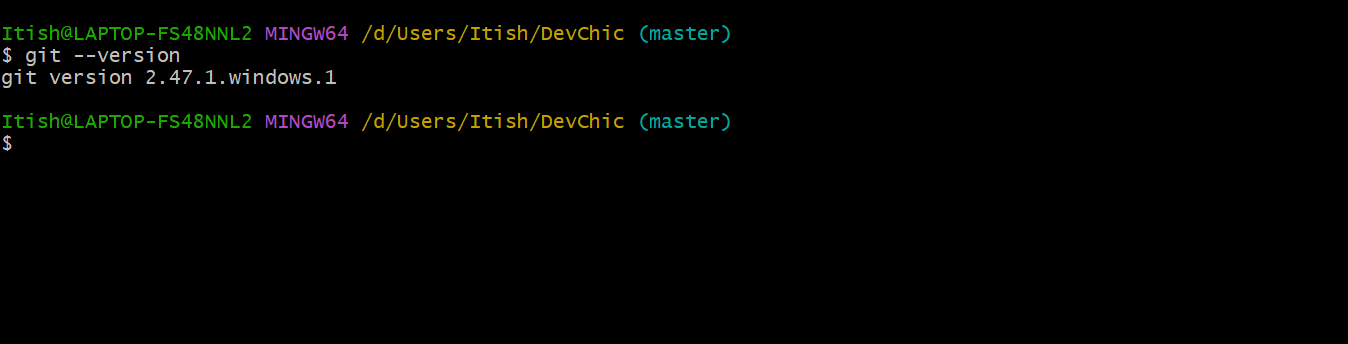
Opted for “64-bit Git for Windows Setup”



Git Setup



Git Installation



Git Bash version

**Practical 2**

**Aim:** Setting up GitHub Account

**Theory:**

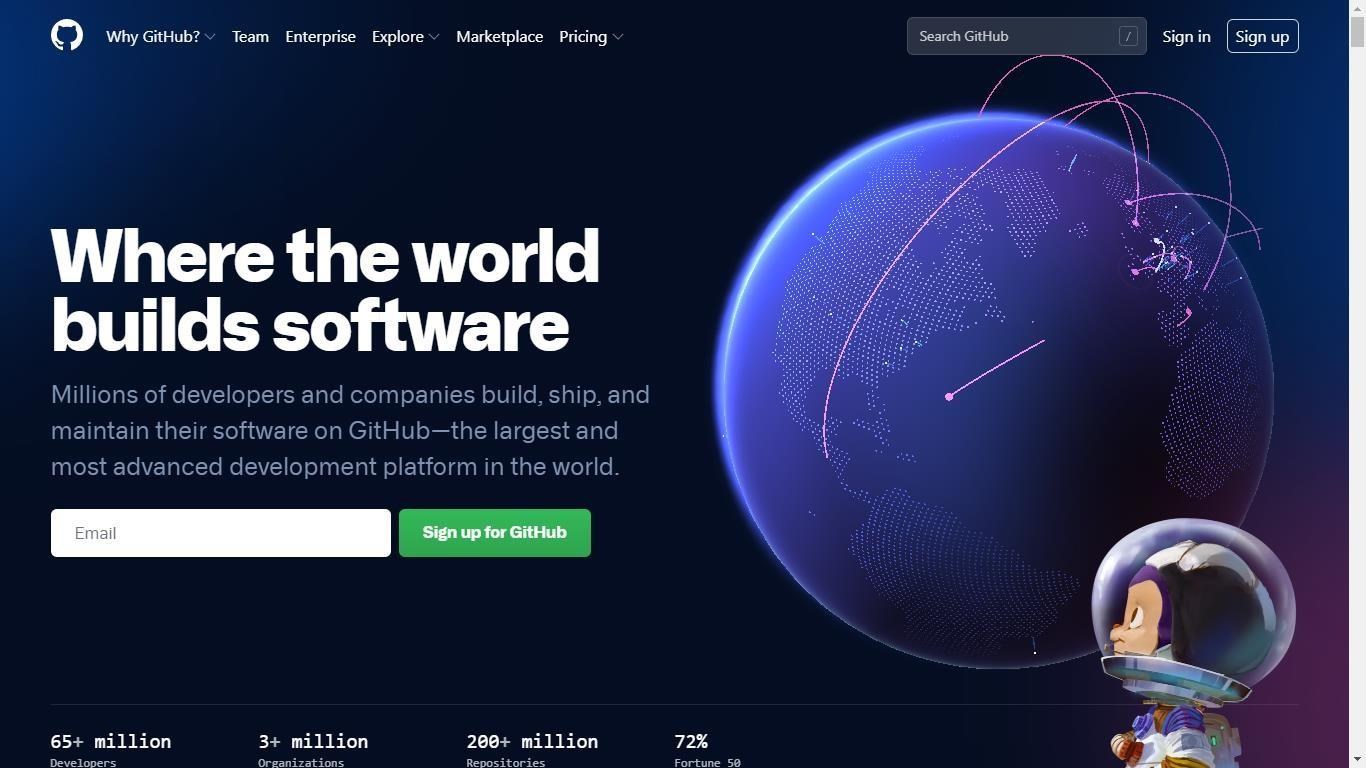
GitHub: GitHub is a website and cloud-based service (client) that helps an individual or developers to store and manage their code. We can also track as well as control changes to our or public code.

Advantages of GitHub: GitHub has a user-friendly interface and is easy to use. We can connect the git-hub and git but using some commands shown below in figure 001. Without GitHub we cannot use Git because it generally requires a host and if we are working for a project, we need to share it will our team members, which can only be done by making a repository. Additionally, anyone can sign up and host a public code repository for free, which makes GitHub especially popular with open-source projects.

**Procedure:**

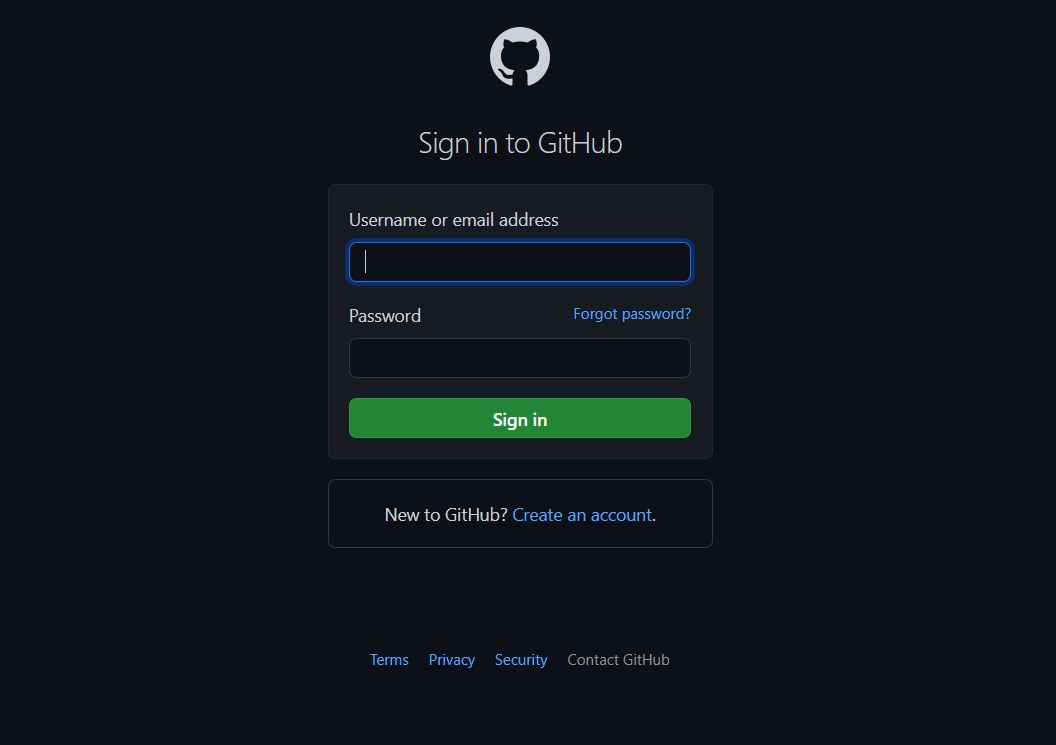
To make an account on GitHub, we search for GitHub on our browser or visit <https://github.com/signup>. Then, we will enter our mail ID and create a username and password for a GitHub account.

**Snapshots:**

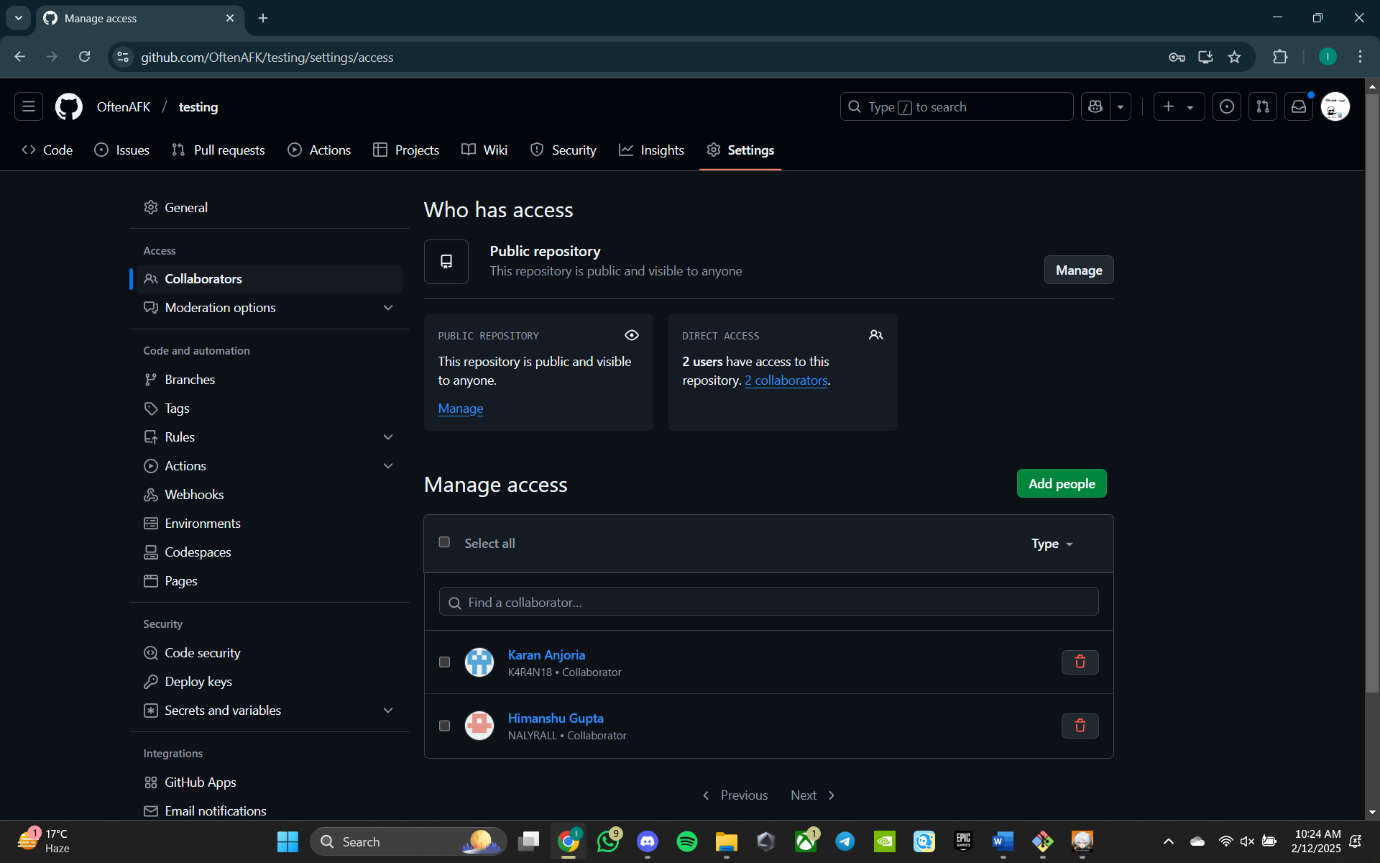


After visiting the link this type of interface will appear, if you already have an account, you can sign in and if not, you can create.

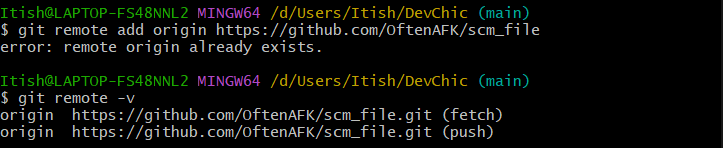
**GitHub Login:**



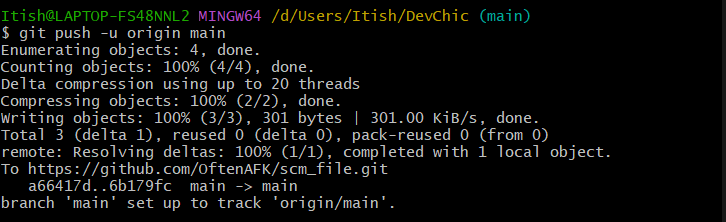
**Adding Collaborators:**



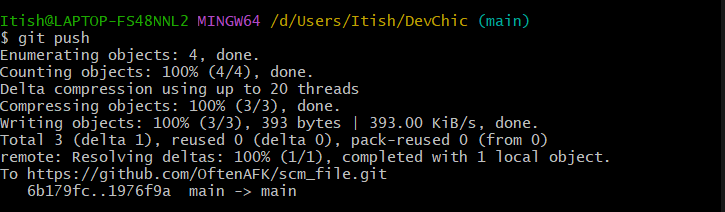
**List of Remote Repositories Associated with Local Git Repository:**

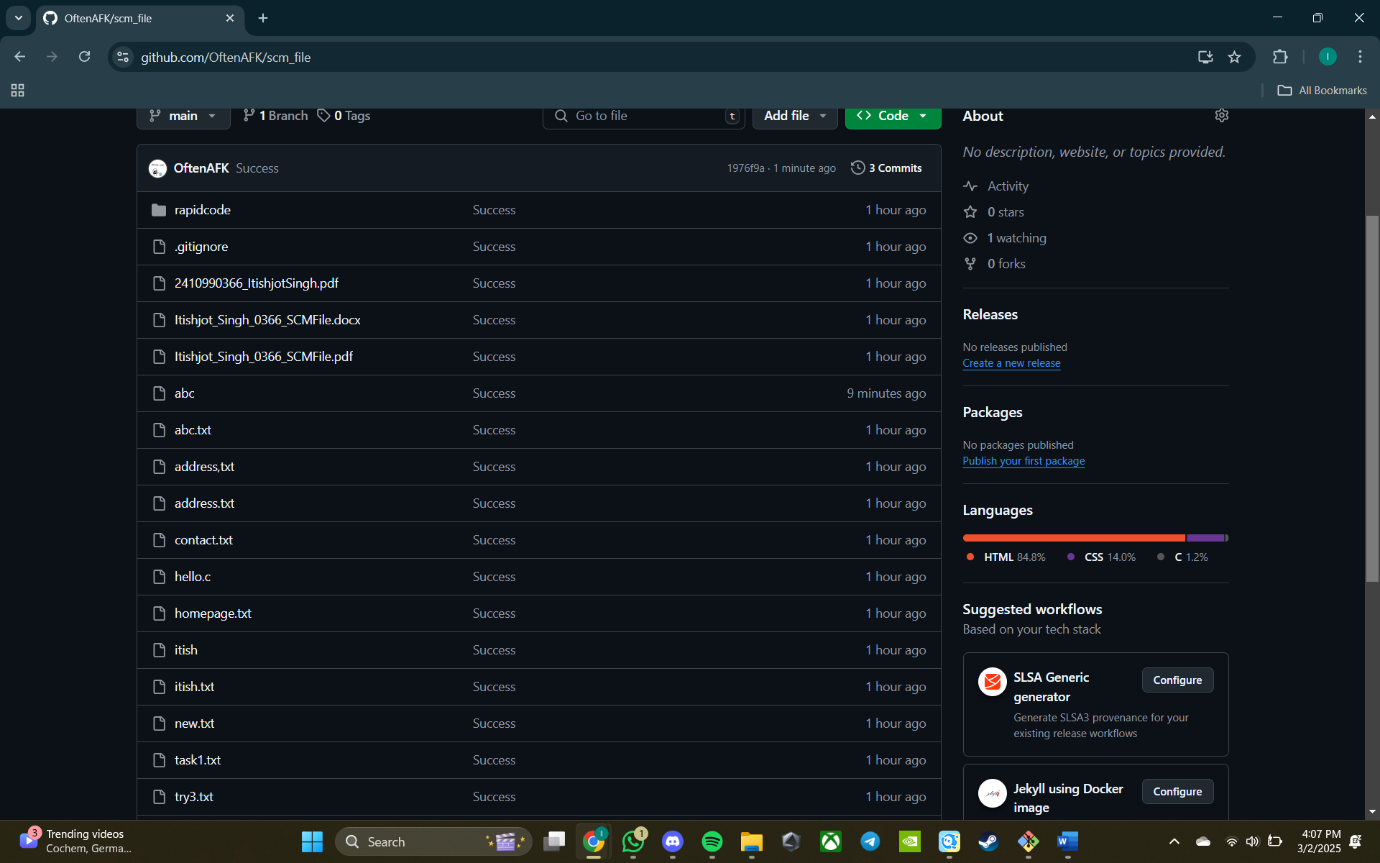


* The command “git remote add origin <link\_of\_repository>” is used to initiate and upstream of data that is to be transmitted to GitHub.
* The command “git remote -v” provides the list of remote repositories associated with our local git repository. Also used for verifying the URLs of your remotes before pushing or pulling.



* The command “git push -u origin main” pushes the data from local repository to the GitHub online repository.
* Using “git push -u origin main” sets the upstream branch for main, making the future git push and git pull commands to be able to function without specifying the remote and branch.





* The file is now available in the online GitHub repository

**Practical 3**

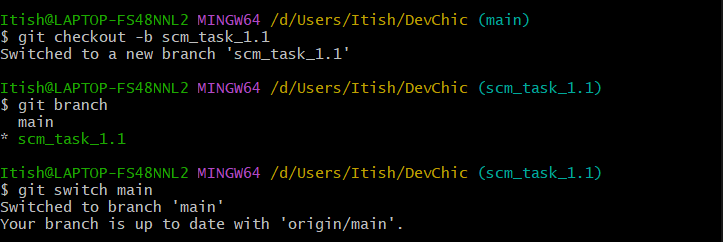
**Aim:**To merge two branches within a Git repository**.**

**Theory:**

 Merging branches in Git allows you to combine changes from one branch into another. It is a fundamental process in collaborative workflows, ensuring all contributions are integrated into a single codebase.

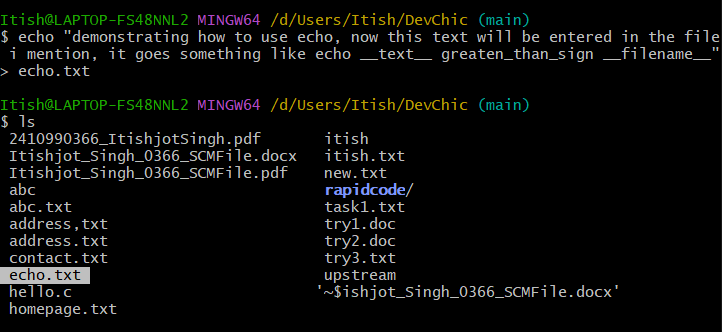
**Procedure:**

**1. Create a new branch and switch to it:**

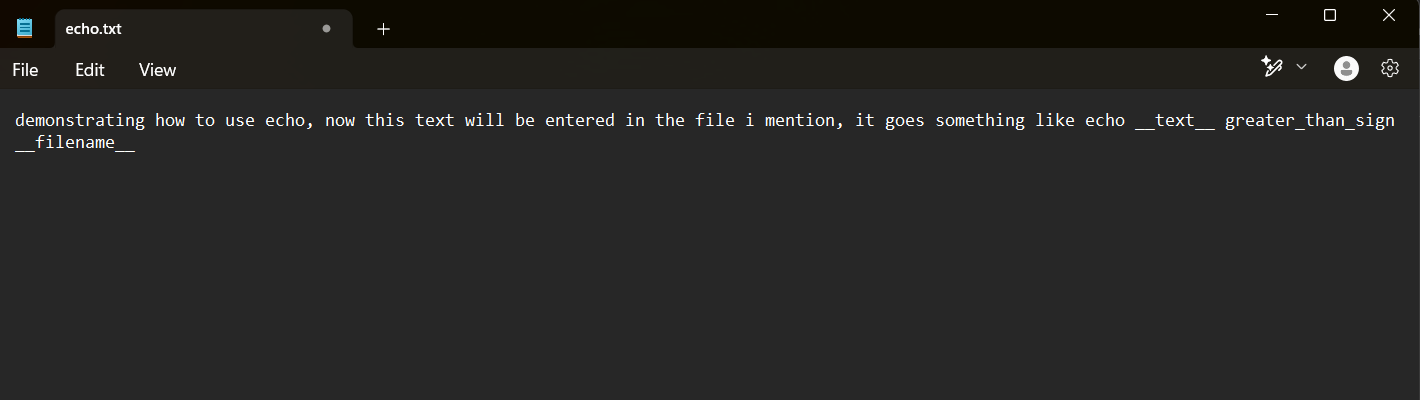


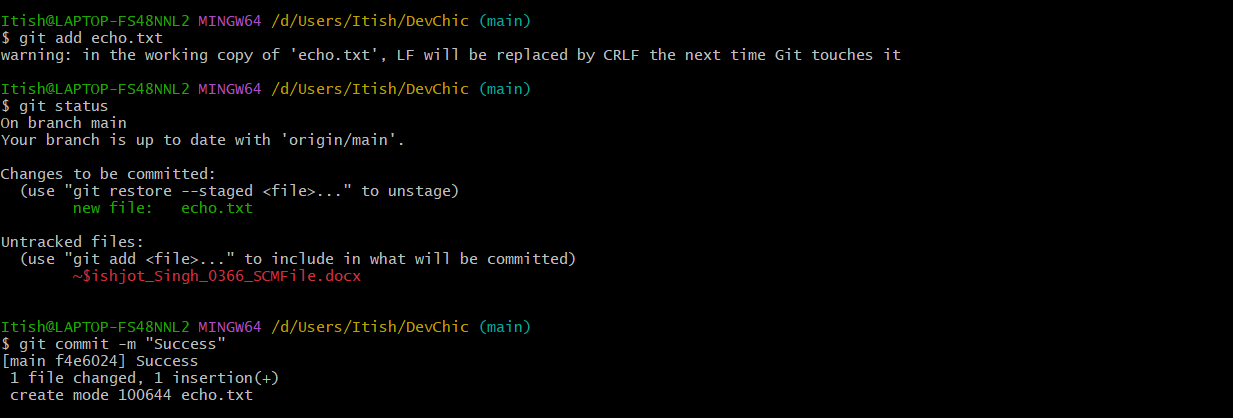
* The command “git checkout -b <branch\_name>” makes a new branch and automatically switches to it.
* Whereas, commands like “git switch <branch\_name>” and “git checkout <branch\_name>” are used to switch between already existing branches.

**2. Make changes to a file in the new branch and commit them:**



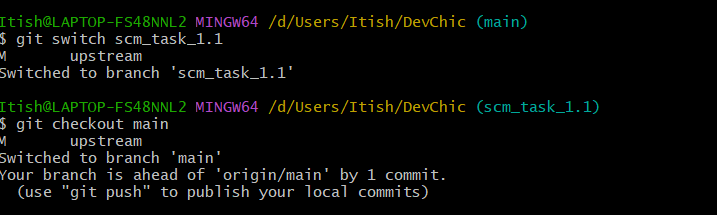
* Using the command “echo -text- > -filename- creates a new file and adds the text that the user typed.





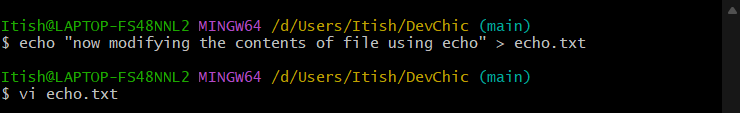
* The new file named echo.txt has now been committed.

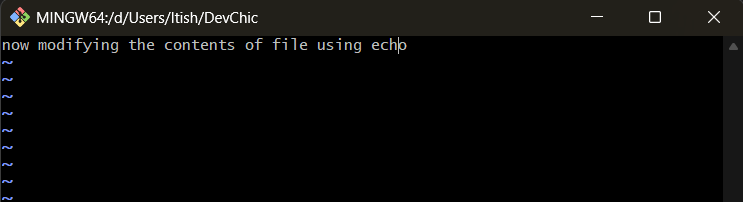
**3. Switch back to the main branch:**



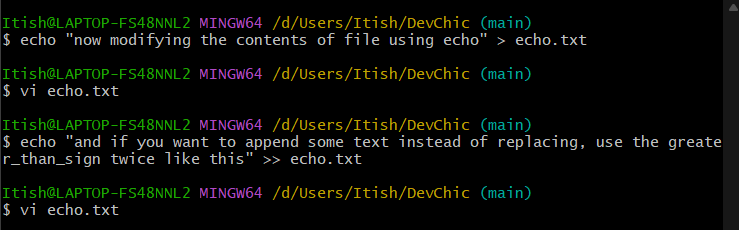
* Using commands like git switch and git checkout to switch between main and the other branches.

**4. Modify another file in the main branch and commit the changes:**

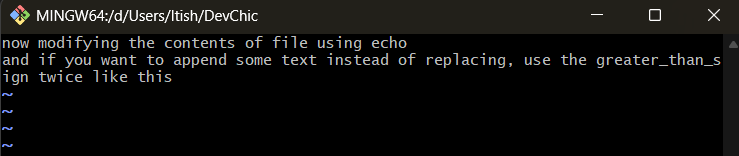




* The contents of the file “echo.txt” have now been modified.

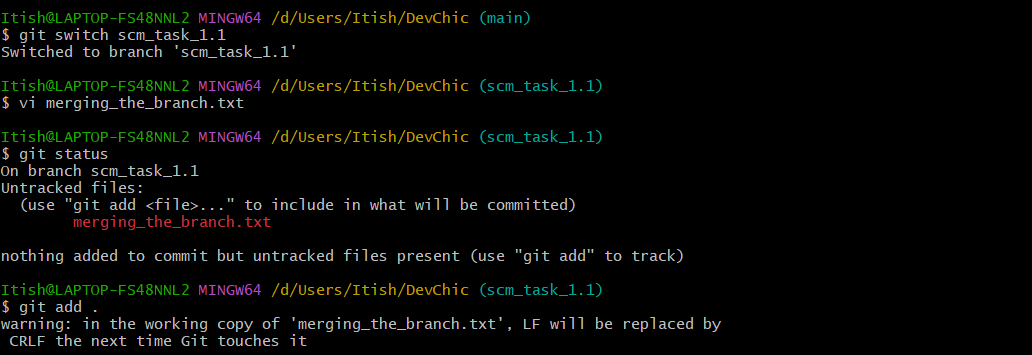


* While editing is a way to do things, appending is an option as well. While appending, instead of using the greater than sign (i.e. “>”) just once, we use twice.

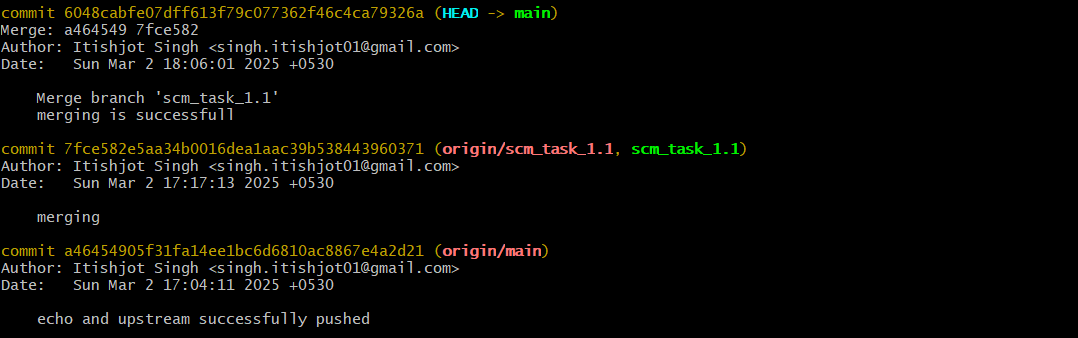


* The contents of echo.txt have now been edited and well as appended by another text.

5. Merge the new branch into the main branch:



* Adding more files to the secondary branch so that it can be merged with main.



* Using the command “git log” we can configure that the branch has successfully been merged with main.

**Practical 4**

**Aim:**

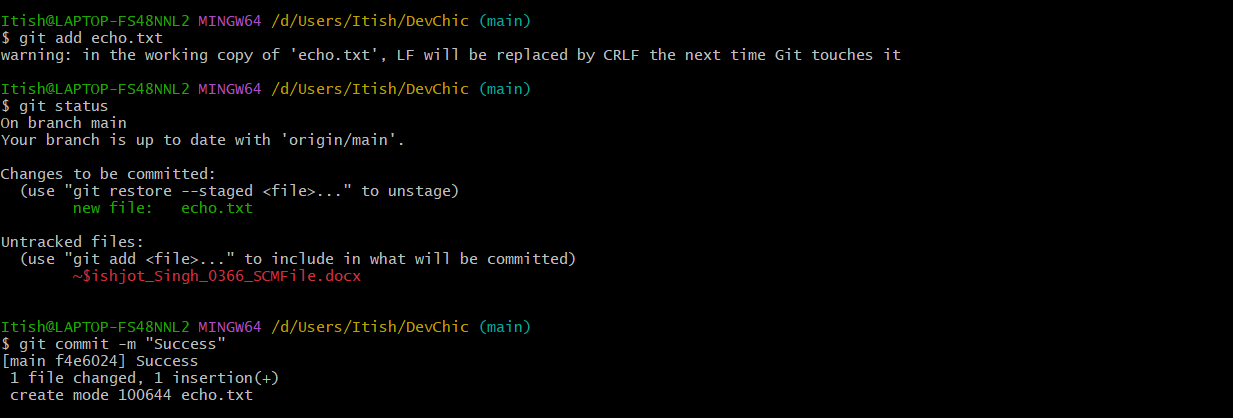
To demonstrate push and pull operations in Git.

**Theory:**

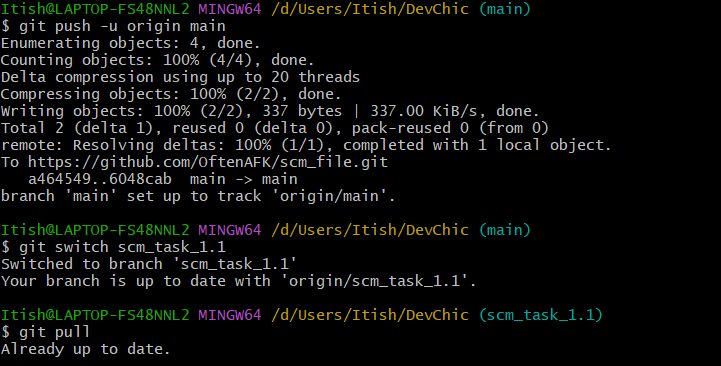
Push transfers committed changes from the local repository to the remote repository, while pull retrieves updates from the remote repository.

**Procedure:**  
○ Make changes in the local repository and commit them.  
○ Push the changes to the remote repository using git push.  
○ Make changes directly on the remote repository (e.g., via GitHub interface).  
○ Pull the changes to the local repository using git pull.

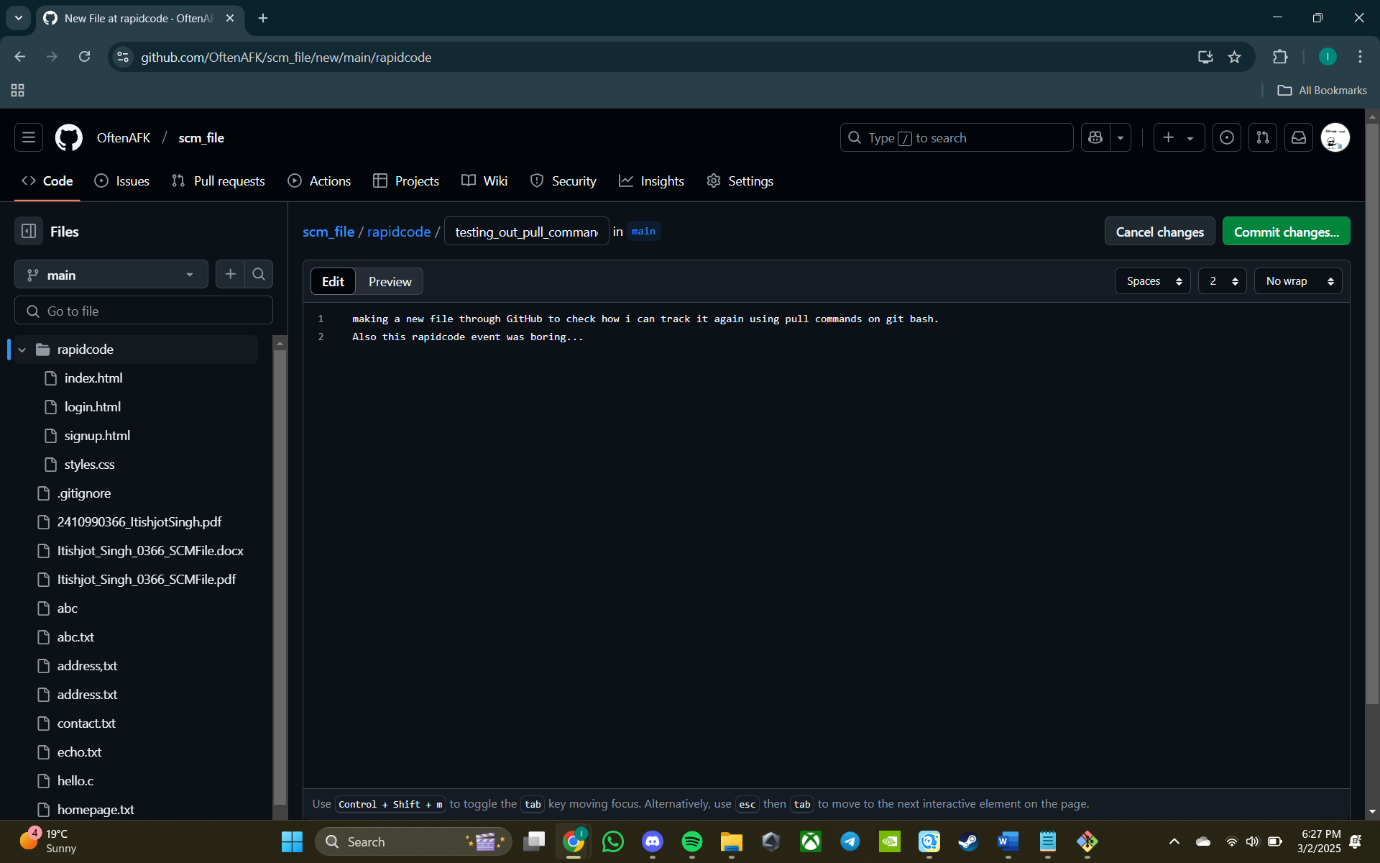
**Tasks:**  
Provide screenshots of the push and pull operations.  
Include the updated commit log.



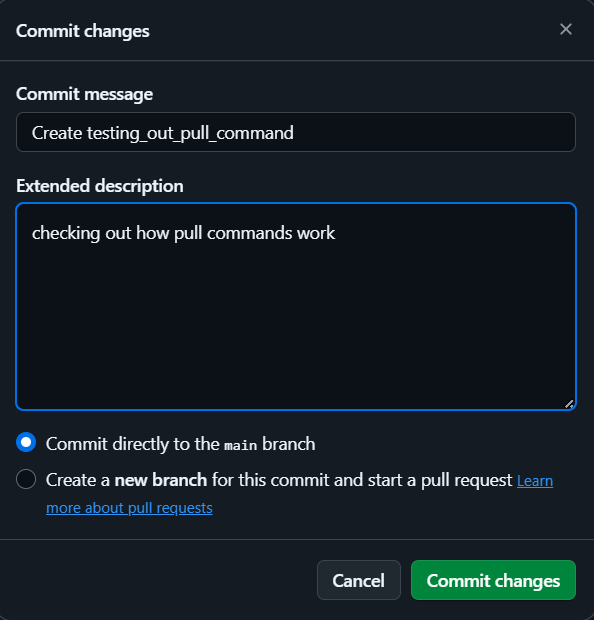
* The file echo.txt was edited once again and after editing, it was pushed to the GitHub repository.



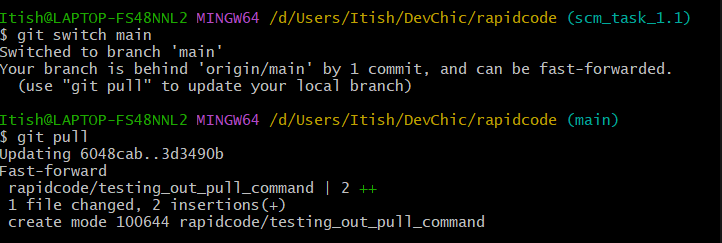
* After using push and pull commands, we are up to date with all the files in the repository.
* Git pull is majorly used to detect what changes have been made by the user as well as other collaborators working in that GitHub repository, since its just the user making changes here, everything is up to date.
* Though git bash won’t detect if the changes are made directly through the GUI of GitHub, and to update the git bash about the changes that were made on GitHub, we use git pull.



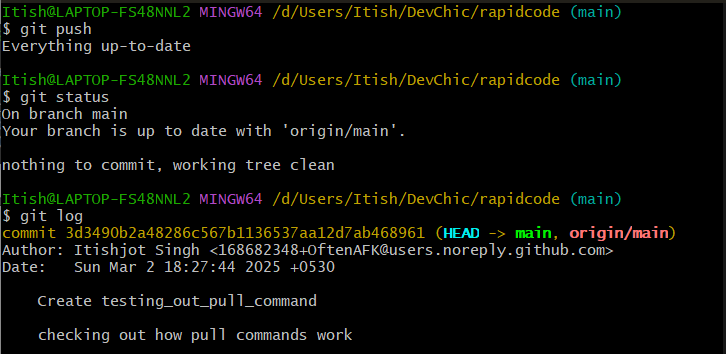
* Making a new file under the rapidcode directory, the file “testing\_out\_pull\_command” was added through GitHub.



* The file “testing\_out\_pull\_command” now being committed through GitHub.

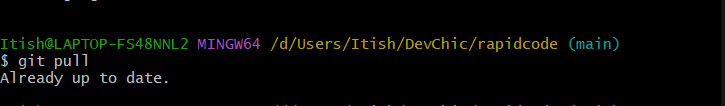


* Now using the command “git pull” we can configure that a change has been made in the rapidcode directory.



* The working tree is clean and we have nothing to commit, hence we are good to go. Now using git log we can check the history and see that git bash is now up to date with the change we made at our GitHub repository.

Therefore:



**Experiment 5**

**Aim:**

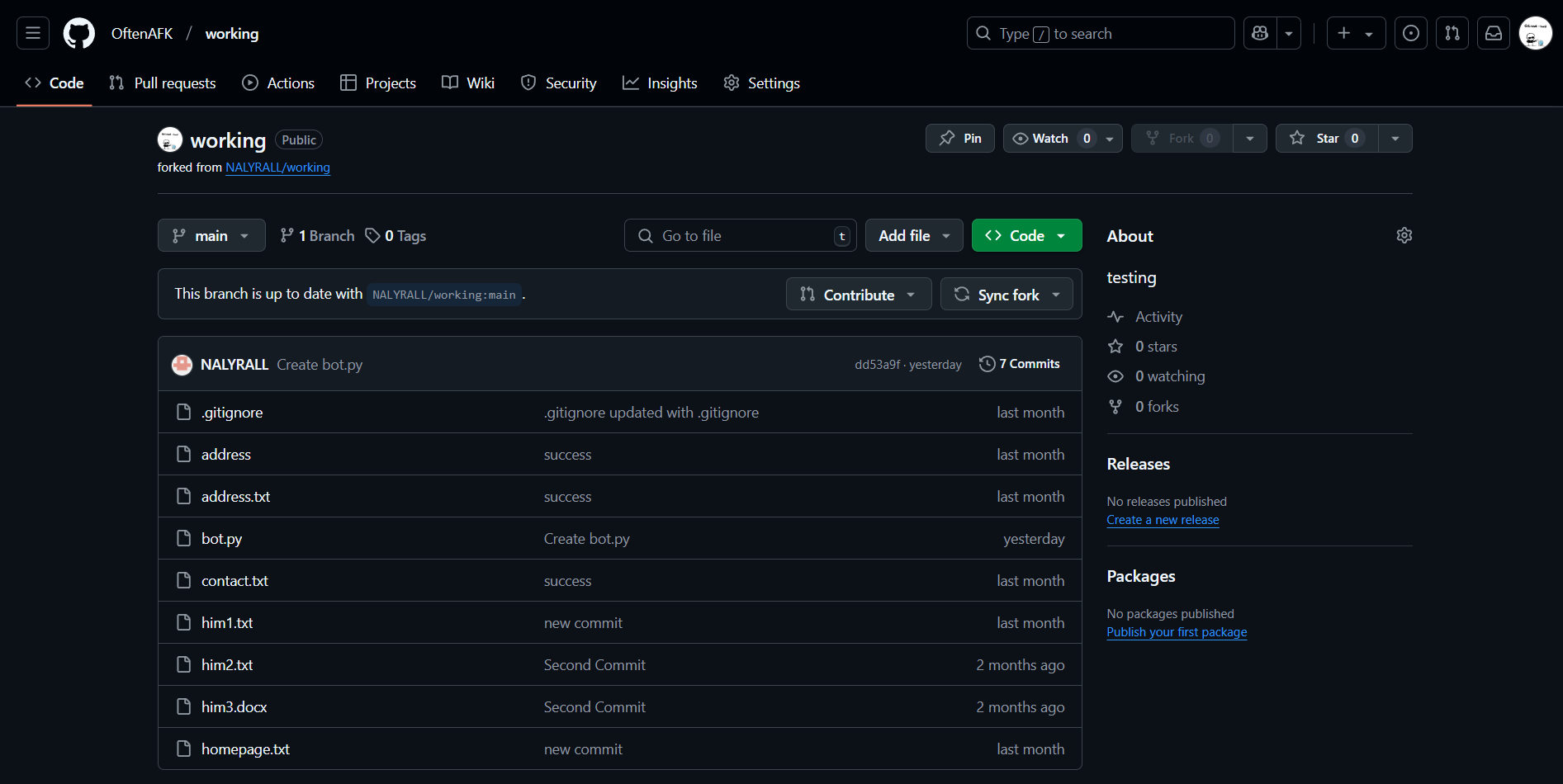
To demonstrate the concepts of forking, cloning, and creating pull requests on GitHub.

**Theory:**

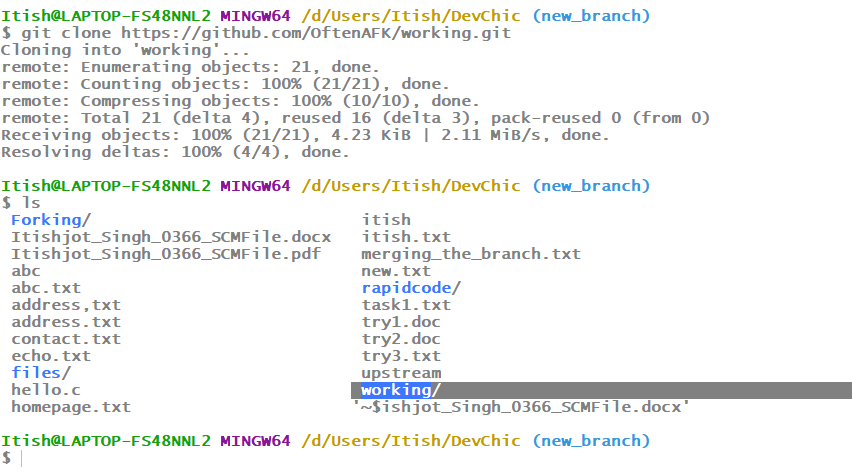
Forking creates a personal copy of a repository, cloning downloads it locally, and pull requests propose changes to the original repository.

**Procedure:**

**•** Fork an existing repository on GitHub.



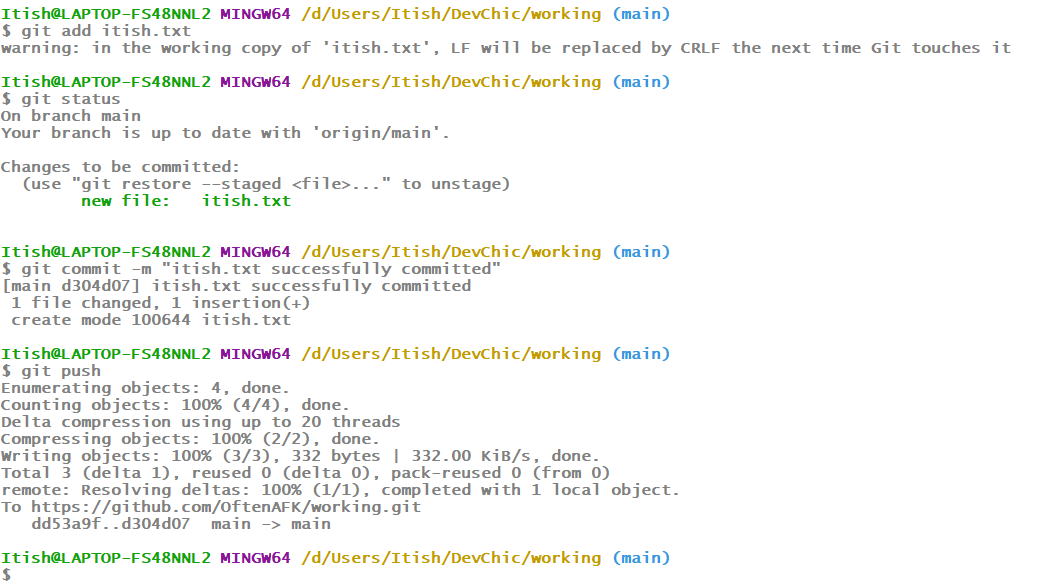
• Clone the forked repository using git clone <URL>.



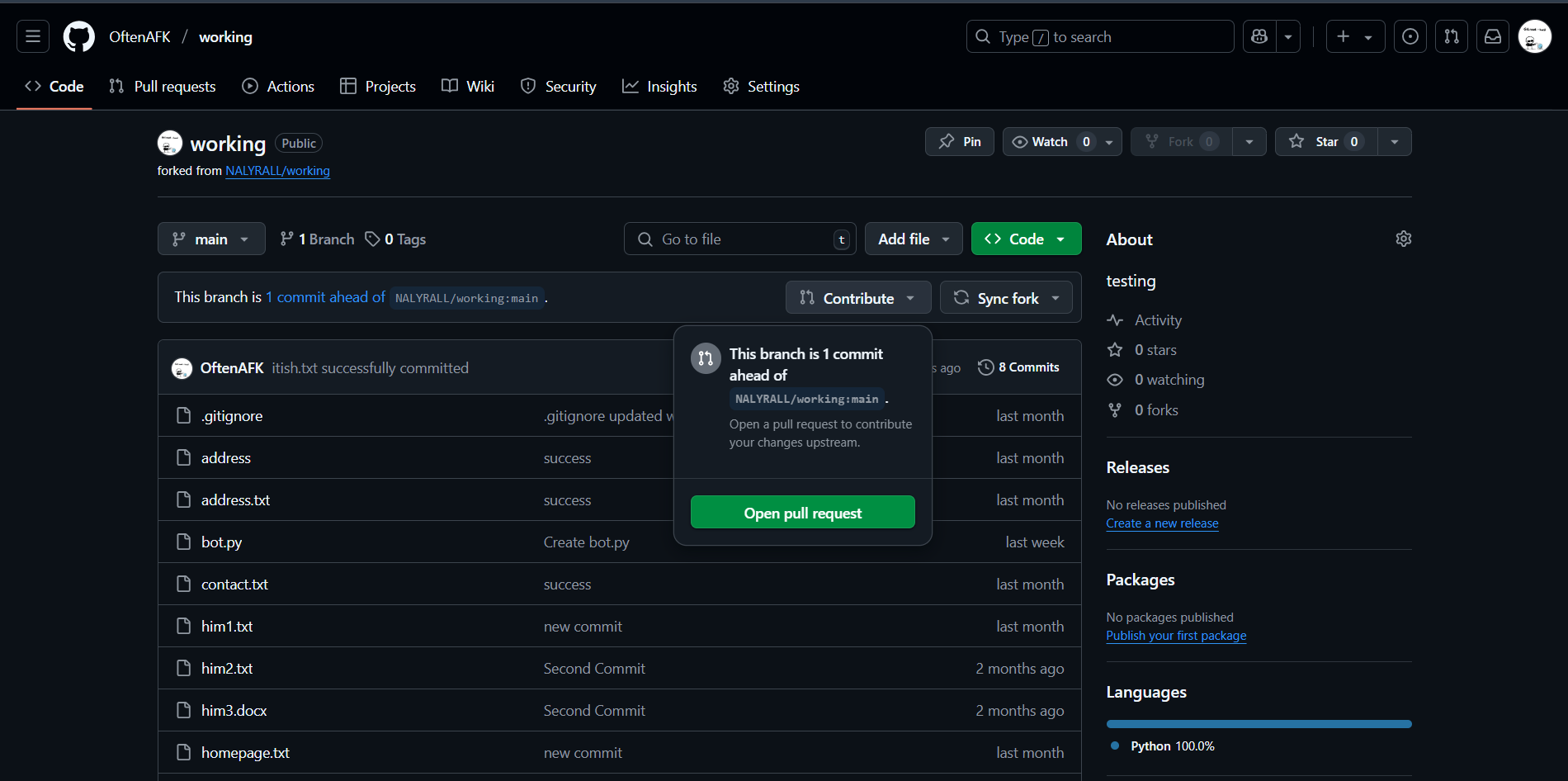
• Make changes in the cloned repository and push them to your forked repository.

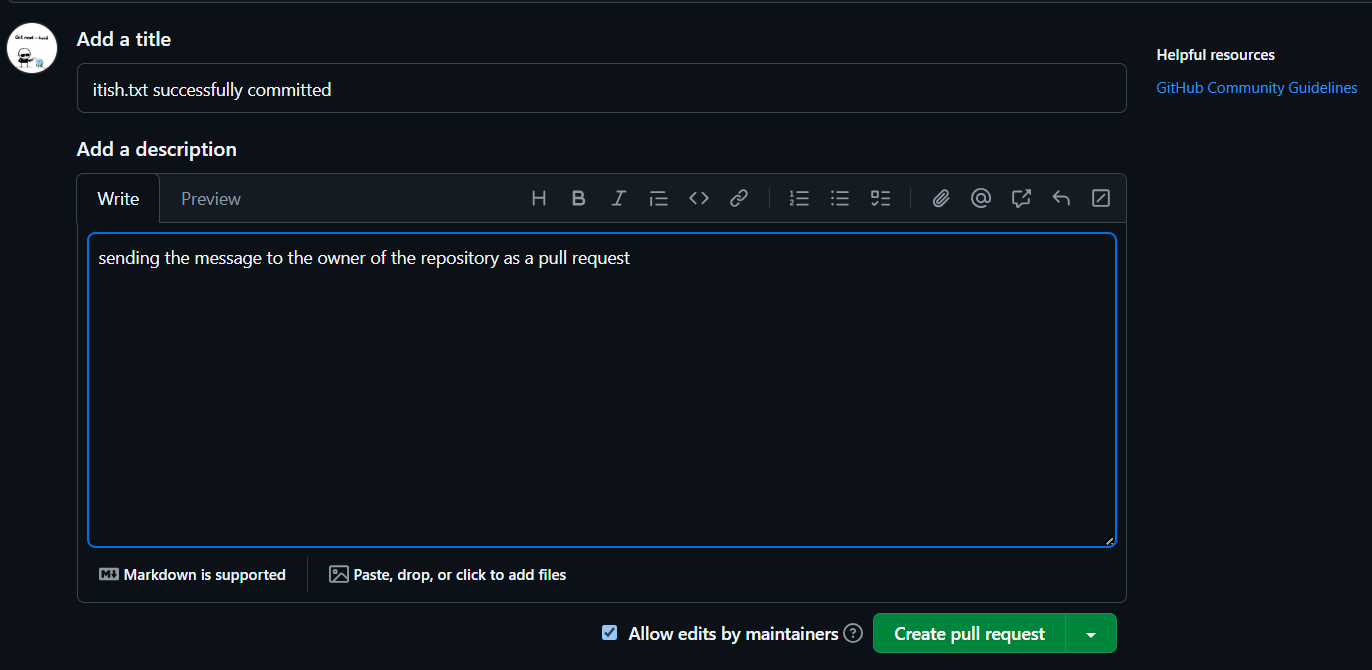


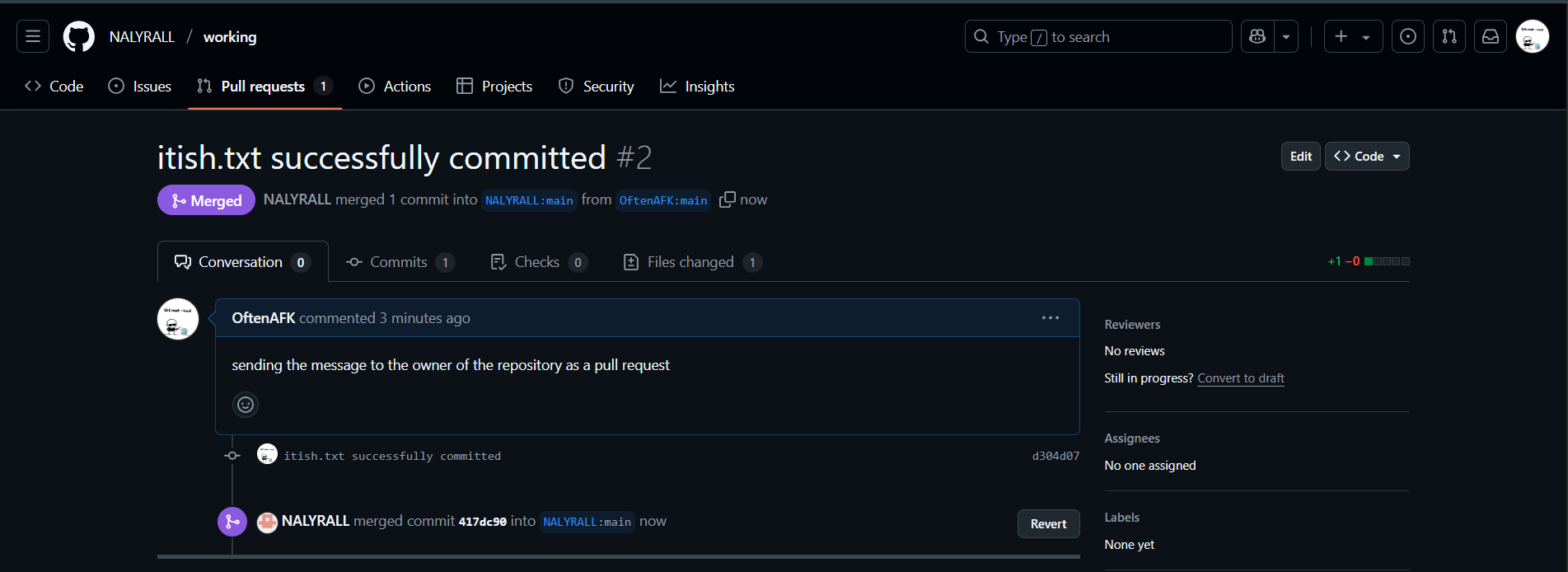
* Adding a new file using echo, we make a new file “itish.txt”. The added file is then to be committed and pushed.



• Create a pull request to propose changes to the original repository.







* Since the cloned repository is one commit ahead of the original repository, creating a pull request we request the owner of the repository to allow modification to the original repository.

**Practical 6**

**Aim:** To manage ignored files and create a structured README file for a repository.

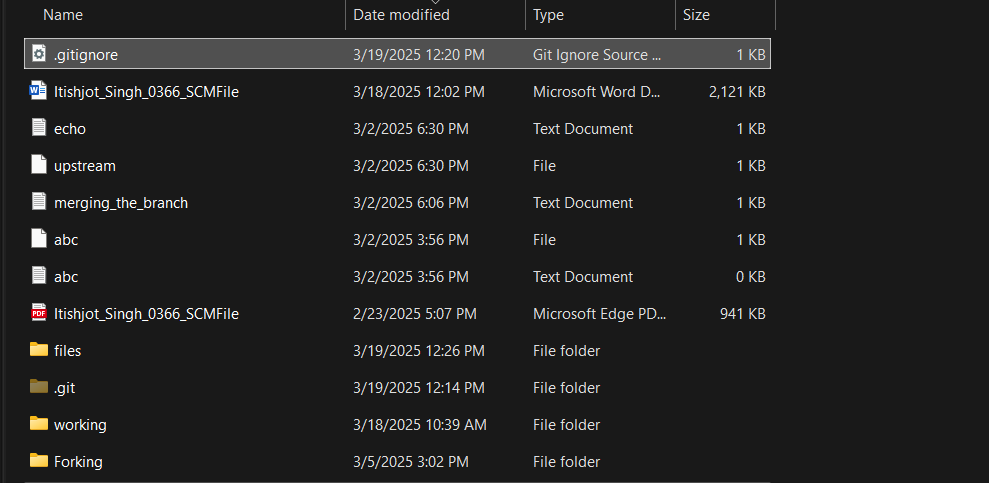
**Theory:**

The .gitignore file specifies files and directories that Git should ignore, ensuring that unnecessary or sensitive files are not included in version control. The README.md file provides an overview of the project, including its purpose, installation instructions, and usage details.

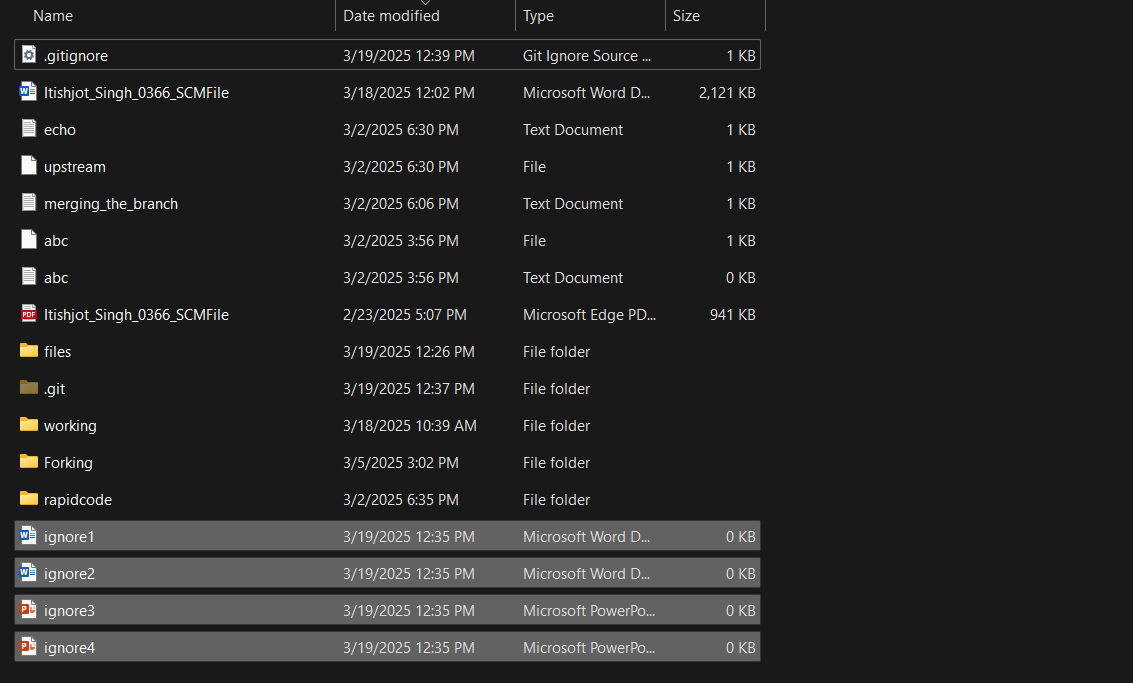
**Instructions:**

1. Create a .gitignore file in your repository.

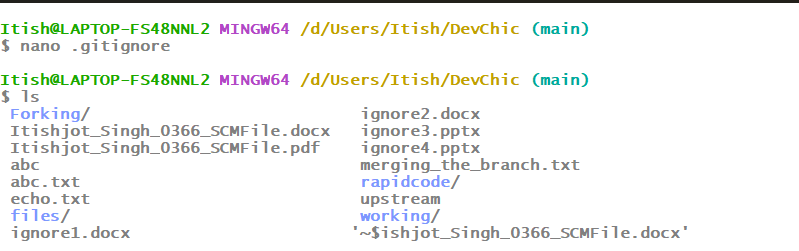


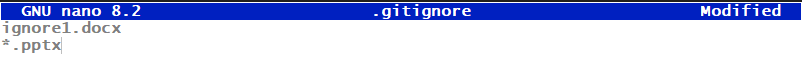


1. Add patterns to the .gitignore file to specify files or directories to ignore.



* 4 new files under the name of ignore are made with different extensions which will then be added to the .gitignore file.





1. Create a README.md file and add the following sections.

* Project Title: Add the name of your project.
* Description: Provide a brief overview of the project.
* Installation Steps: Include detailed steps for setting up the project.
* Usage: Explain how to use the project.

1. Commit the .gitignore and README.md files to the repository.
2. Push the changes to the remote repository on GitHub.
3. Capture the following:

* The content of your .gitignore file.
* A screenshot of the README.md file as it appears in the GitHub repository.

**Tasks:**

* Provide the .gitignore file content.
* Include a screenshot of the README.md file on GitHub.

Practical 7: Scenario-Based Merge Conflict Resolution

● Aim: To simulate and resolve merge conflicts in Git.

● Theory: Simulated scenarios of merge conflicts provide a better understanding of conflict resolution

strategies.

● Procedure:

1. Create a conflict by modifying the same file in two different branches.

2. Merge the branches and encounter the conflict.

3. Use Git tools like git mergetool or manual resolution to resolve the conflict.

4. Commit the resolved file.

● Tasks:

o Provide screenshots showing the conflict and resolution process.

o Include a log of the resolved merge commit.