Module 6: Git and GitHub

Abhijit Lalasaheb Zende

1. L1 – Create local Git repository and demonstrate all git reset options and revert. Compare the differences

Ans.

(*** Note: Screen shots attached to end of each question ***)

Step 1: Initialize a Local Git Repository

1. Open your terminal or command prompt.
2. Create a new directory:
   `mkdir git-assignment`
3. Navigate to the directory:
   `cd git-assignment`
4. Initialize a Git repository:
   `git init`

Step 2: Create file and make initial commits

1. Create a file:
   `echo "First version" > file.txt`
2. Add and commit:
   ```
   git add file.txt
   git commit -m "Initial commit"
   ```
3. Make further changes:
   ```
   echo "Second version" >> file.txt
   git commit -am "Second commit"
   echo "Third version" >> file.txt
   git commit -am "Third commit"
   ```

Step 3: Demonstrate Git Reset Options

1. Soft Reset:

`git reset --soft HEAD~1`
Changes are staged, but the commit is undone.

2. Mixed Reset (default):
   `git reset HEAD~1`
   Unstages changes but keeps them in the working directory.

3. Hard Reset:
   `git reset --hard HEAD~1`
   Deletes all changes permanently.

Step 4: Demonstrate Git reveret

1. Revert the last commit:
   `git revert HEAD`
   This creates a new commit that reverses changes.

Comparison Between git reset and git revert:

The primary difference between git reset and git revert lies in how they handle the history of commits.

- **git reset** modifies the commit history by moving the HEAD pointer back to a specific commit, essentially erasing any commits made after that point. This is useful when you want to discard unwanted changes entirely, but it can be risky, especially if the changes have been pushed to a shared repository, as it rewrites history and can cause issues for collaborators.
- **git revert**, on the other hand, is a safer option as it creates a new commit that undoes the changes introduced by a specific commit without altering the commit history. This means that git revert is often preferable when working in a team, as it maintains a transparent and traceable history of all changes. Unlike git reset, it does not rewrite history, making it ideal for public branches.

In summary, use git reset when you want a clean slate and are working locally, while git revert is better for maintaining a clear and collaborative commit history.

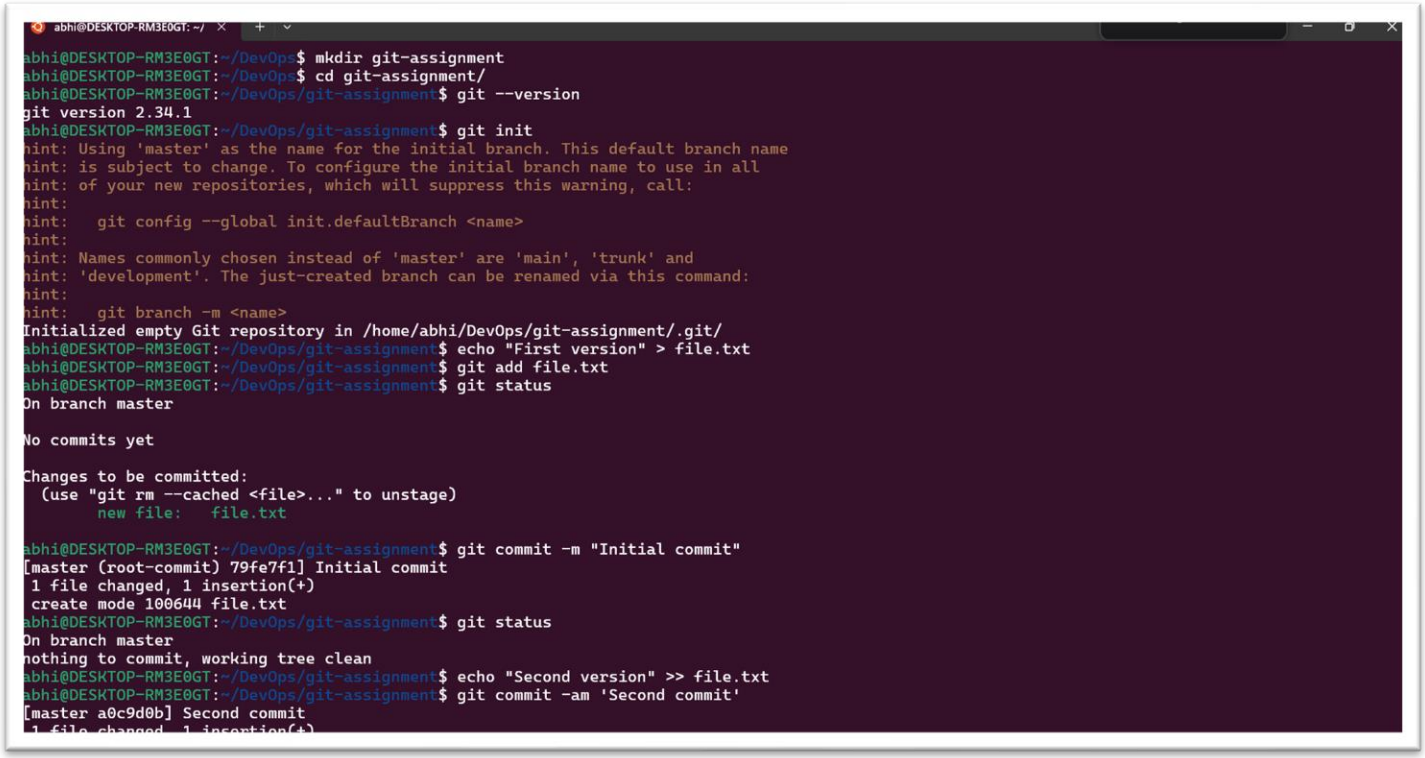

Fig. 1.01: Initializing local git repository

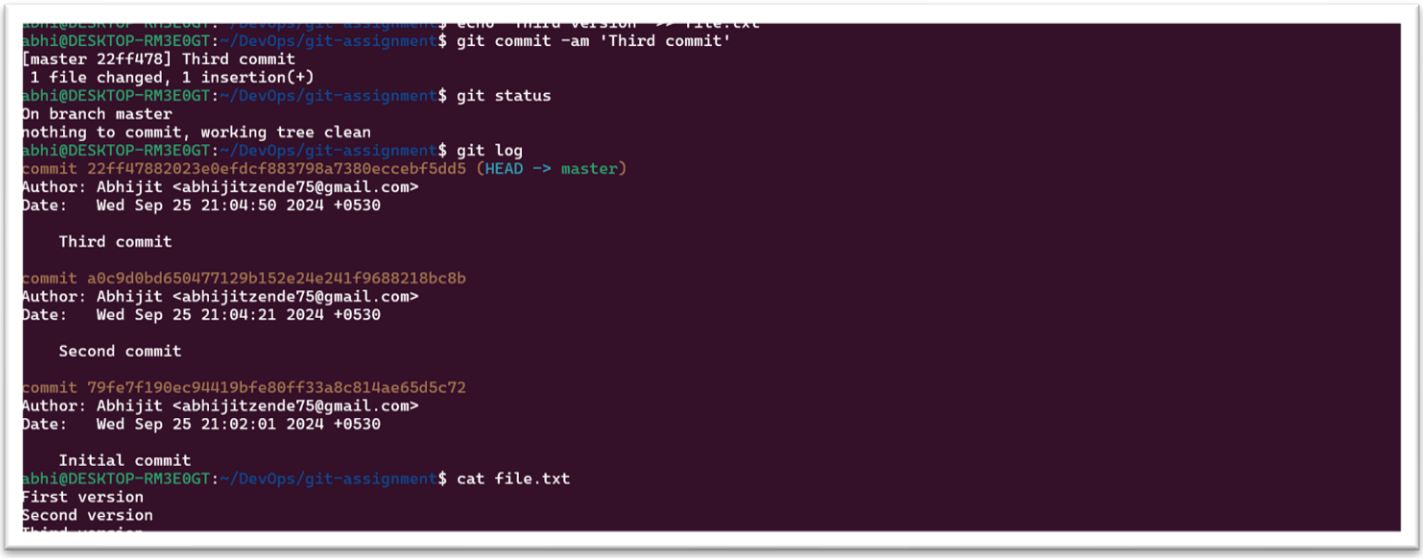

Fig 1.02: Knowing the status of git and file

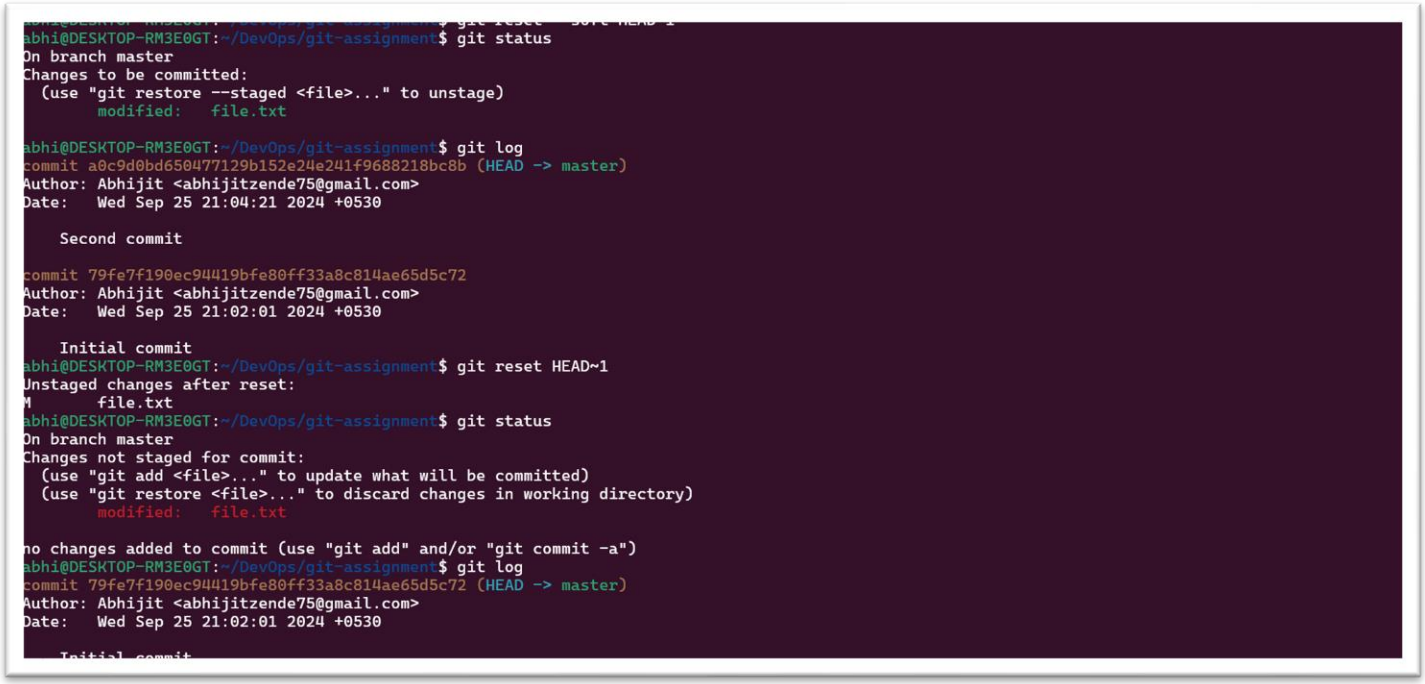

Fig. 1.03: Demonstration of soft and mixed git reset

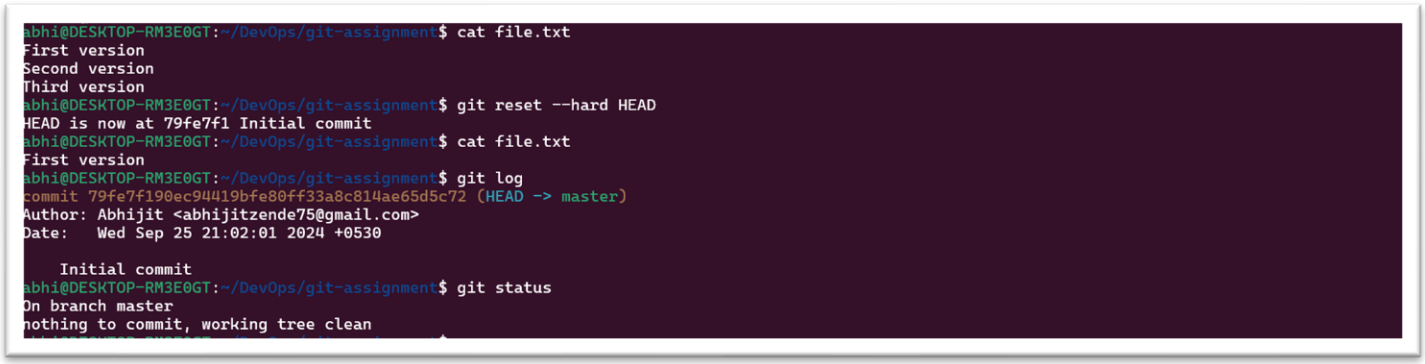

Fig. 1.04: Demonstration of hard git reset

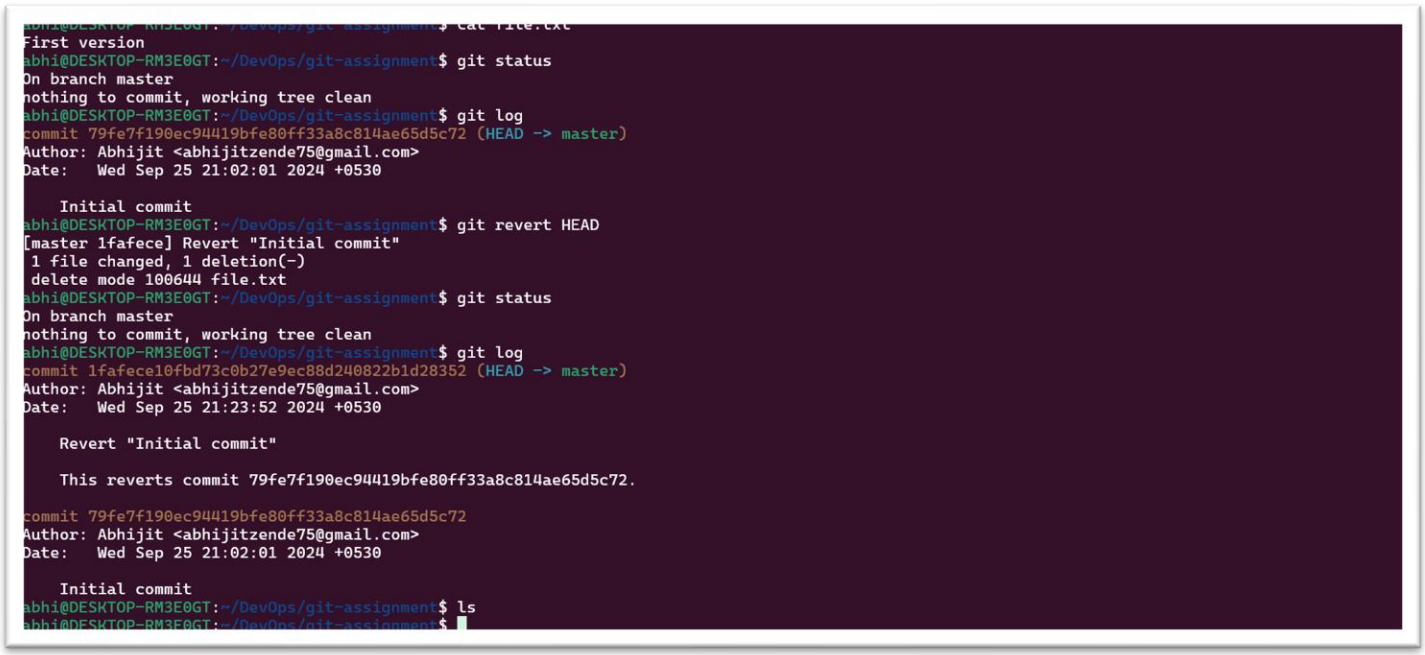

Fig. 1.05: Demonstration of git revert

2. L2 - Create Local git repository and demonstrate git merge and Merge Conflicts with the steps to resolve merge conflicts

Step 1: Create a New Repository and Branches

1. Initialize a repository and create branches:
   ```
   git checkout -b branch1
   echo "Line 1" > conflict.txt
   git add conflict.txt
   git commit -m "Commit on branch1"
   ```
2. Create another branch:
   ```
   git checkout master
   git checkout -b branch2
   echo "Different line 1" > conflict.txt
   git add conflict.txt
   git commit -m "Commit on branch2"
   ```

Step 2: Merge Branch and Create a Conflict

1. Merge branch into branch2:
   ```
   git checkout branch2
   git merge branch1
   ```
   You will see a conflict message.

Step 3: Resolve the Merge Conflict

1. Open conflict.txt in an editor.
2. Modify to resolve the conflict.
3. Mark the conflict resolved:
   ```
   git add conflict.txt
   git commit -m "Resolved merge conflict"
   ```

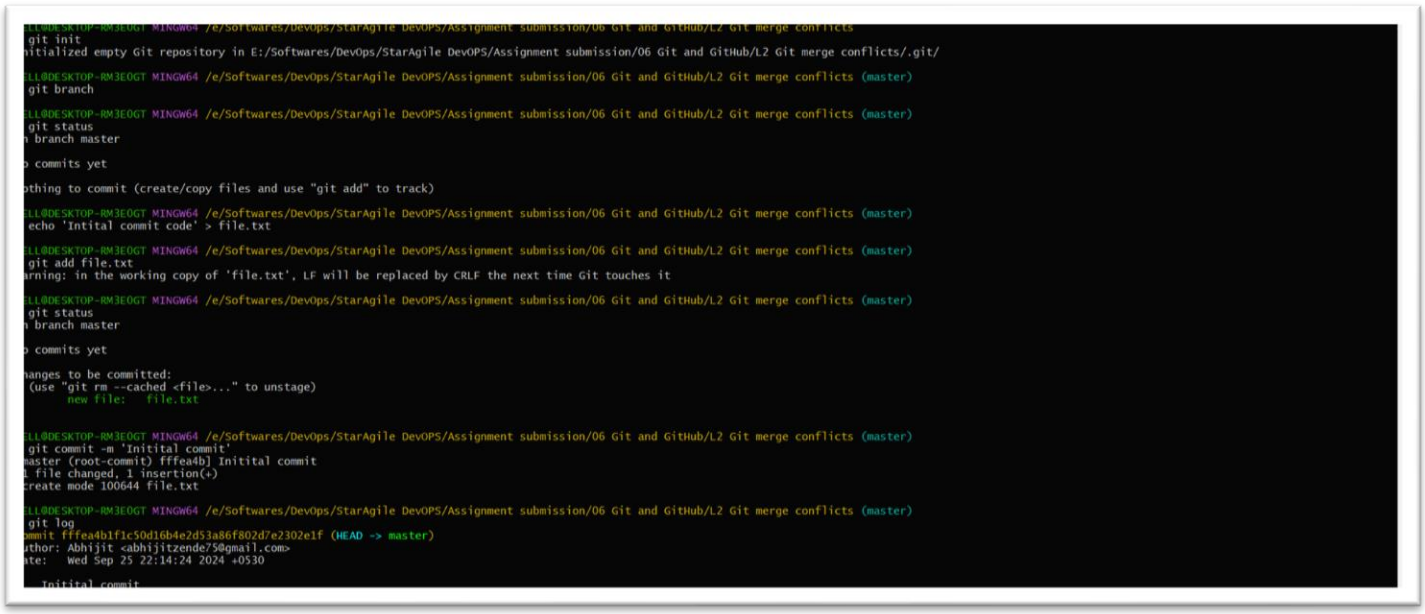

Fig. 2.01: Initial commit

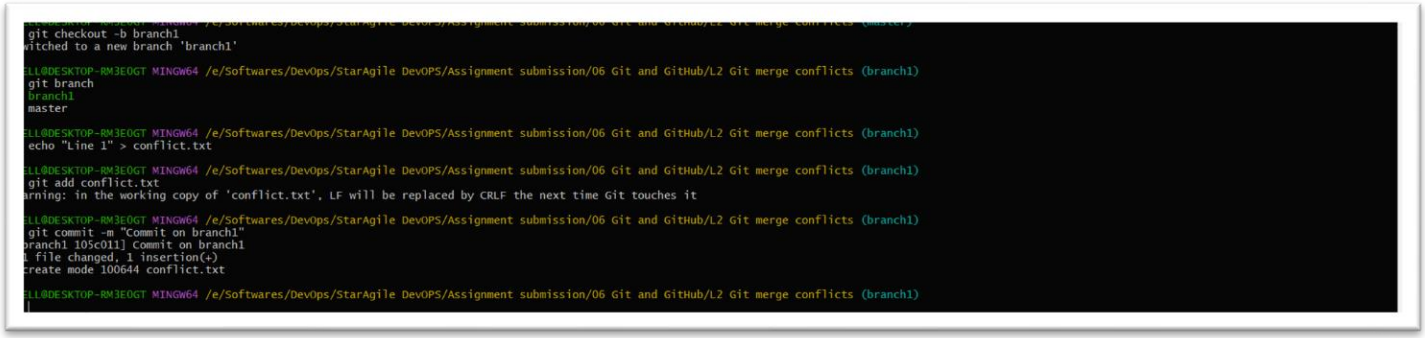

Fig. 2.02: Checking out to new branch 1

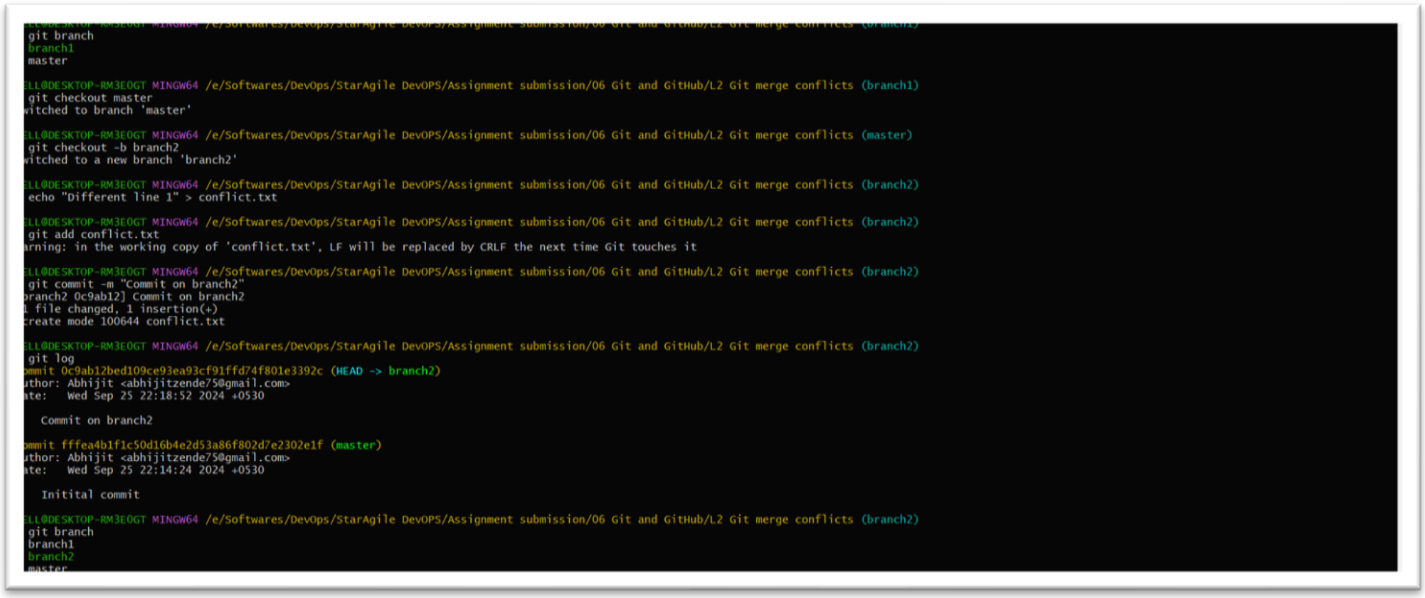

Fig. 2.03: Checking out to branch 2

Fig. 2.04: Merge conflict

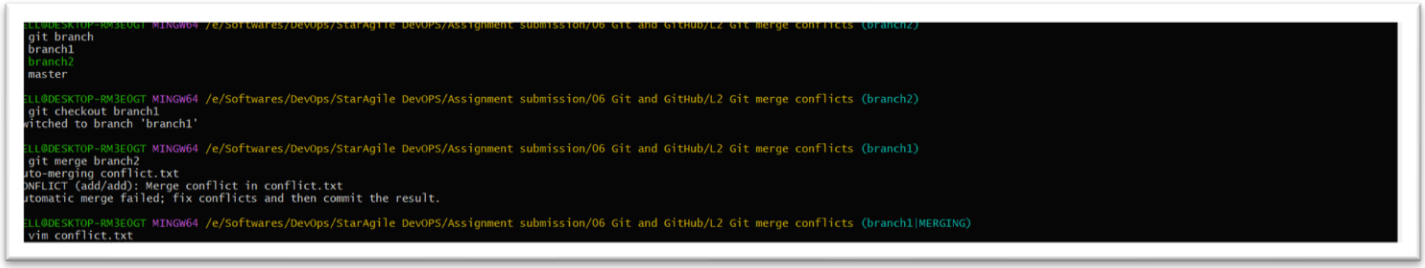

Fig. 2.05: Opening Vim text editor for resolving merge conflict

Fig. 2.06: Merge conflict viewed from vim text editor

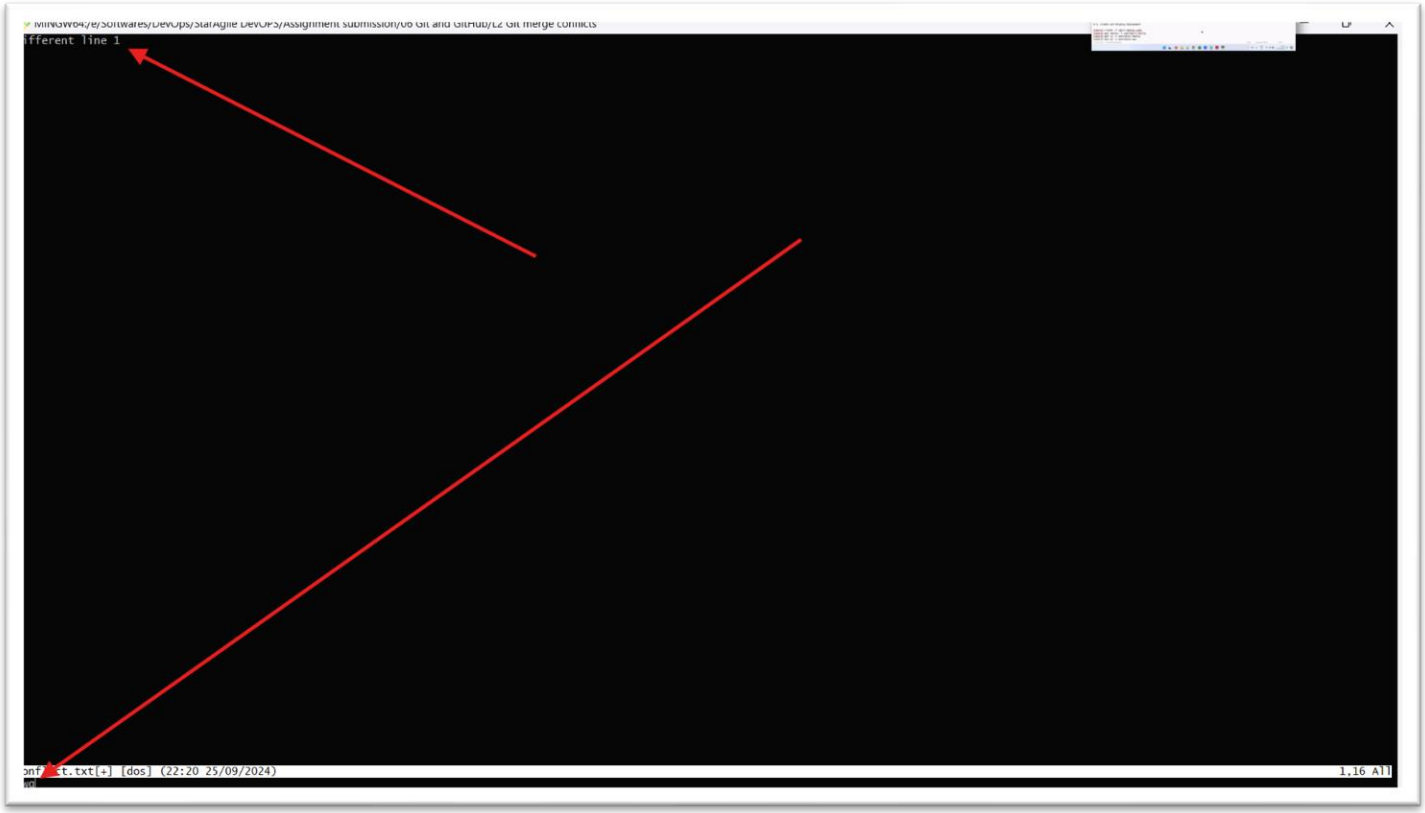

Fig. 2.07: Vim text editor view after resolving merge conflict

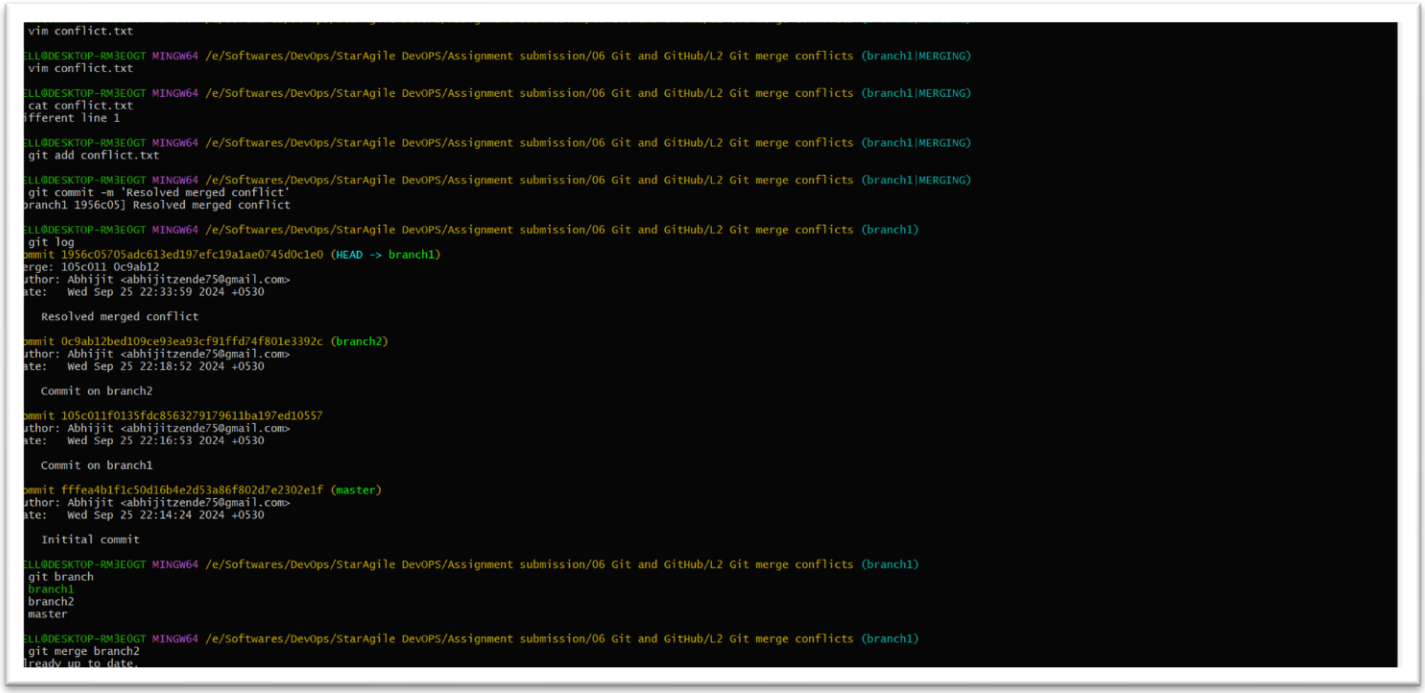

Fig. 2.08: Merge conflict resolved

3. L3 - Using Local and Remote git repositories demonstrate git pull and git fetch. Compare the differences

Step 1: Set Up a Remote Repository

1. Create a new repository on GitHub (e.g., git-demo).

Step 2: Connect Local to Remote:

1. Initialize git repository and add remote
   `git remote add origin https://github.com/yourusername/git-demo.git`

Step 3: Demonstrate git fetch

1. Fetch changes:
   `git fetch origin`
   Changes are downloaded but not merged.

Step 4: Demonstrate git pull

1. Pull changes:
   `git pull origin master`

Comparison:

git fetch only downloads, while git pull downloads and merges.

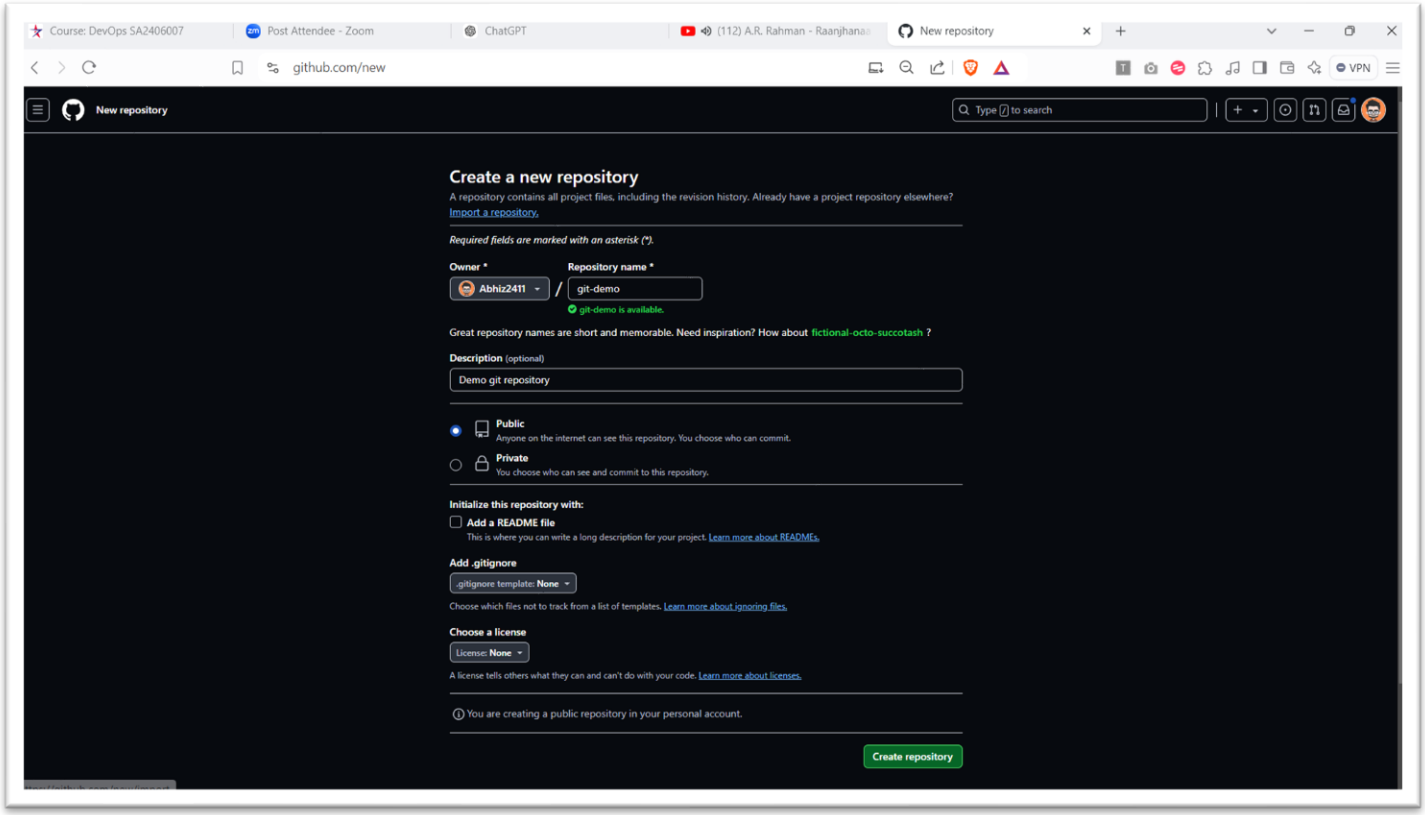

Fig. 3.01: Creating a GitHub repository

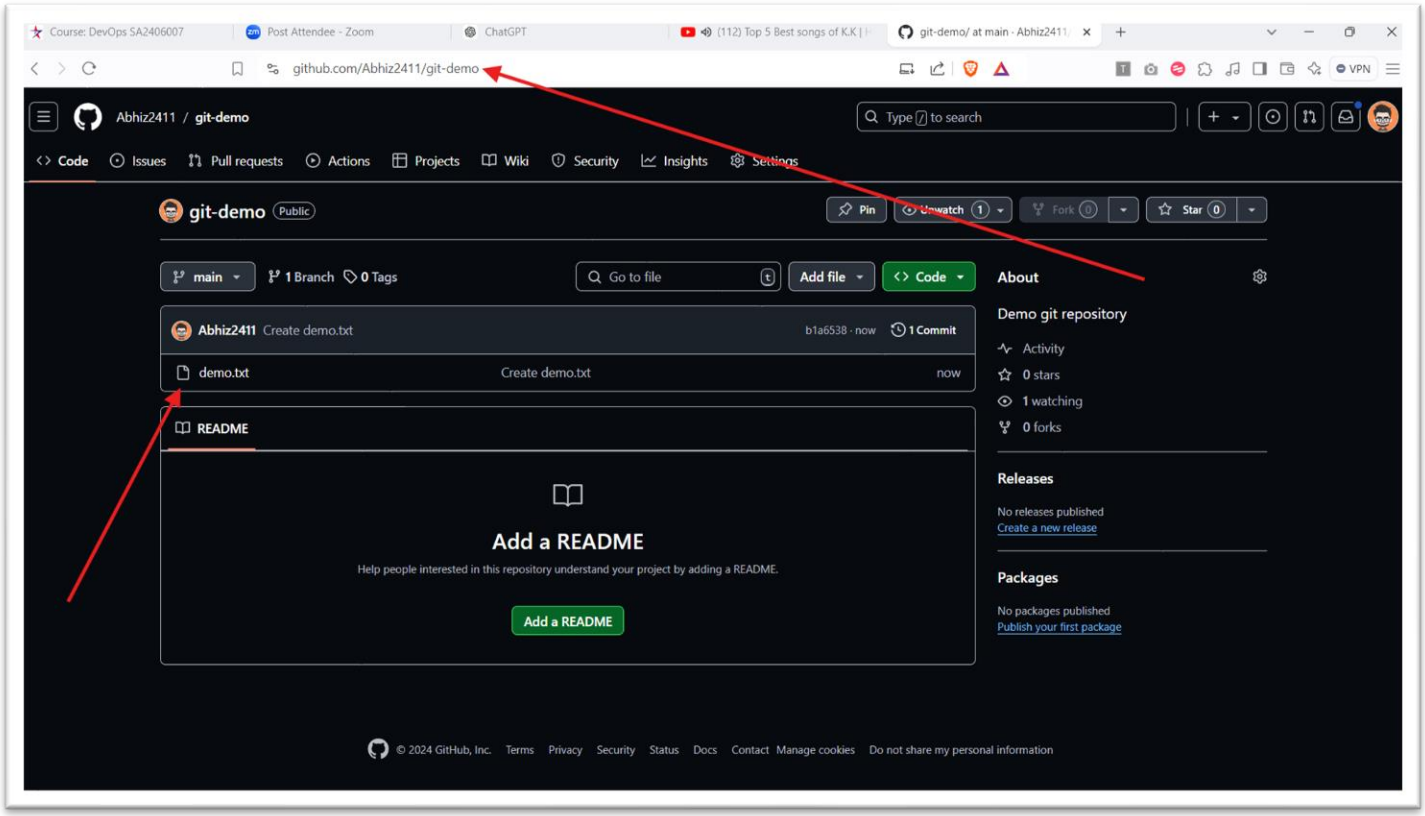

Fig. 3.02: Upload a file to the repo

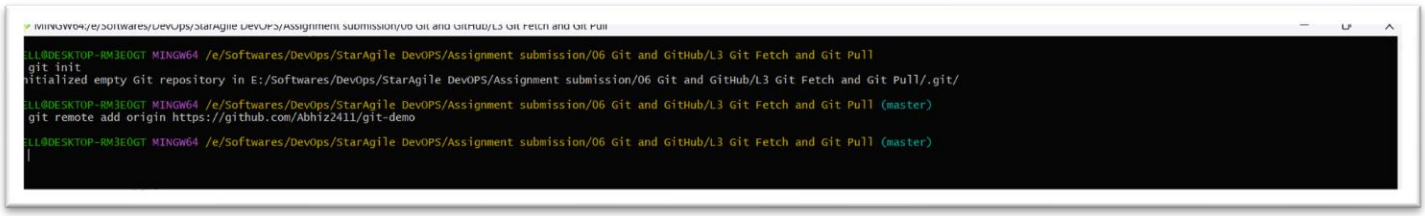

Fig. 3.03: Add remote origin to the repo

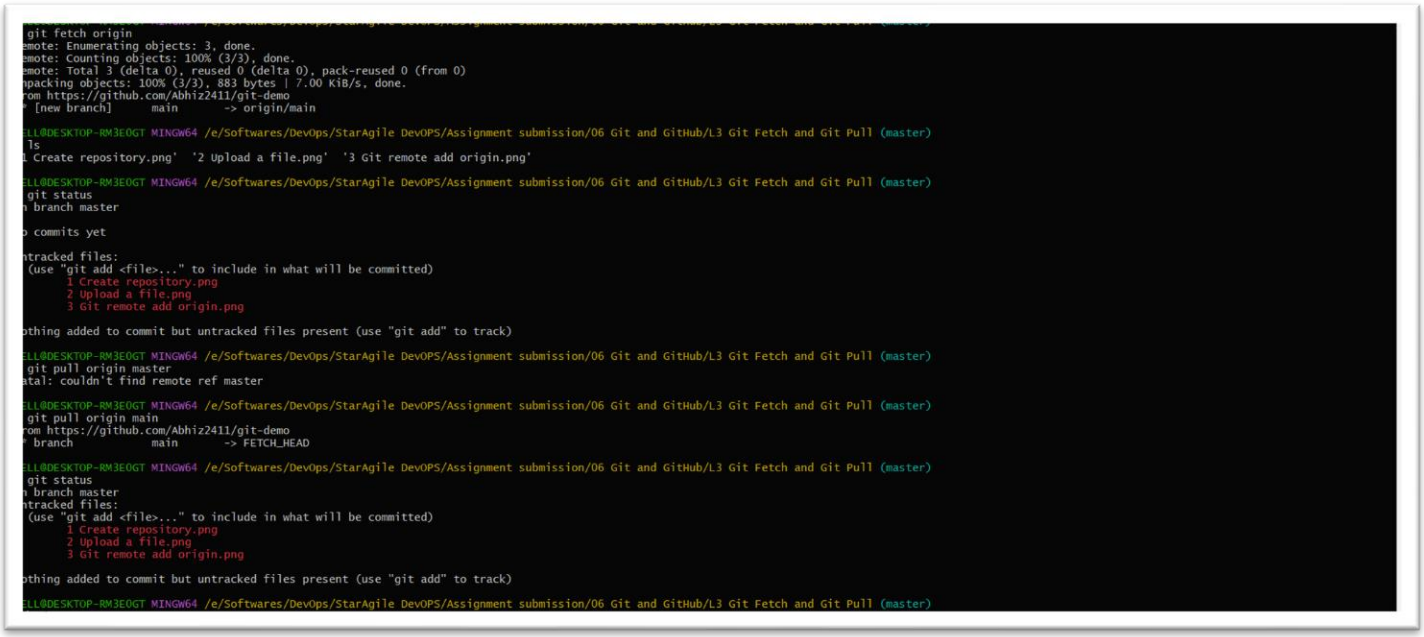

Fig. 3.04: Demonstrating git fetch and git pull

4. L4 - Clone GitHub repository using Visual Studio Code IDE

Step 1: Create a repository if not already exists

Step 2: Download and Install VS Code if not already installed

Step 3: Clone the Repository Using VS Code

1. Open VS Code and go to View > Command Palette.
2. Select Git: Clone and paste the repository URL from GitHub.
3. Select Git: Clone and paste the repository URL from GitHub.

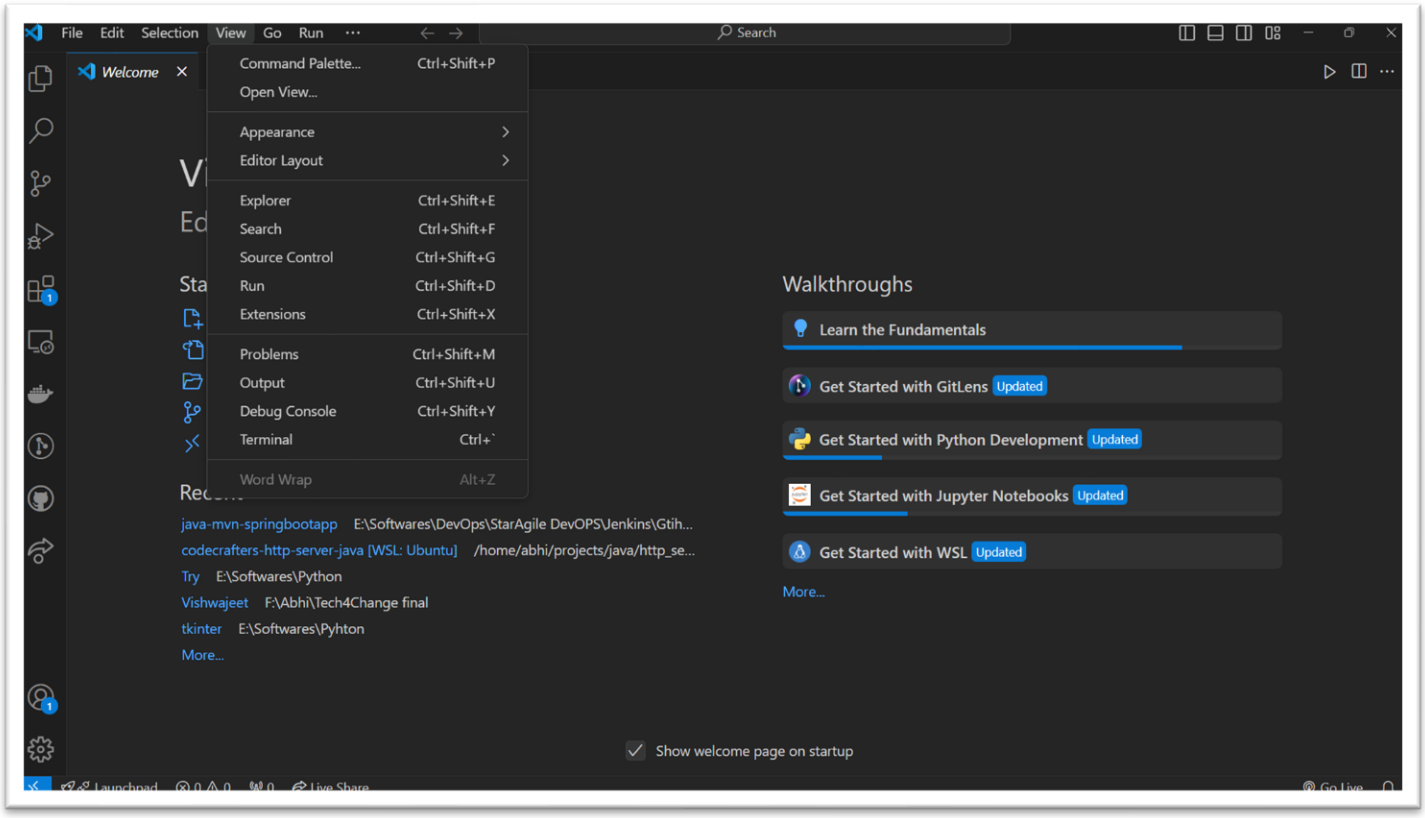

Fig. 4.01: View command palette of VS code

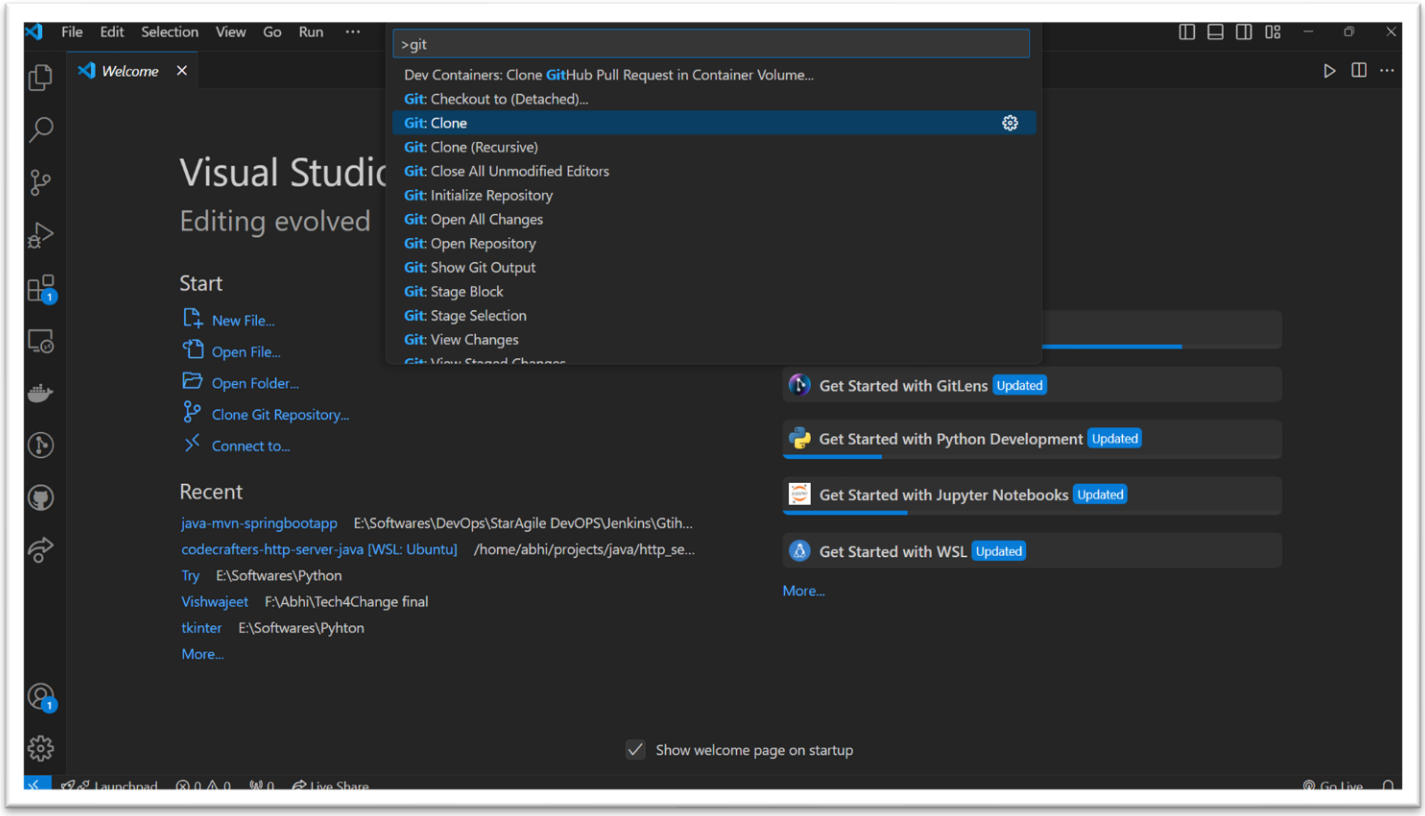

Fig. 4.02: Git clone option of VS code

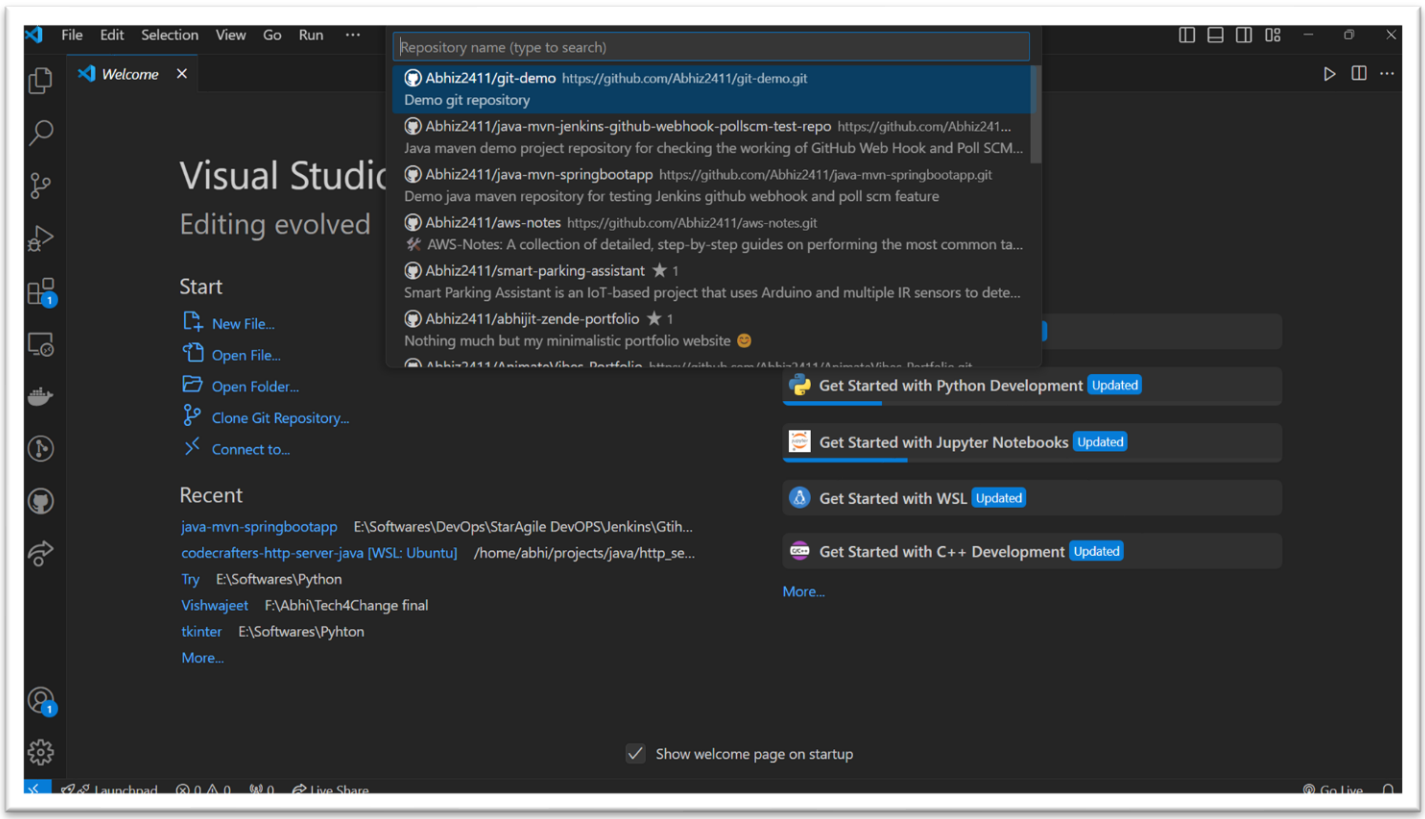

Fig. 4.03: Select the repository name to clone

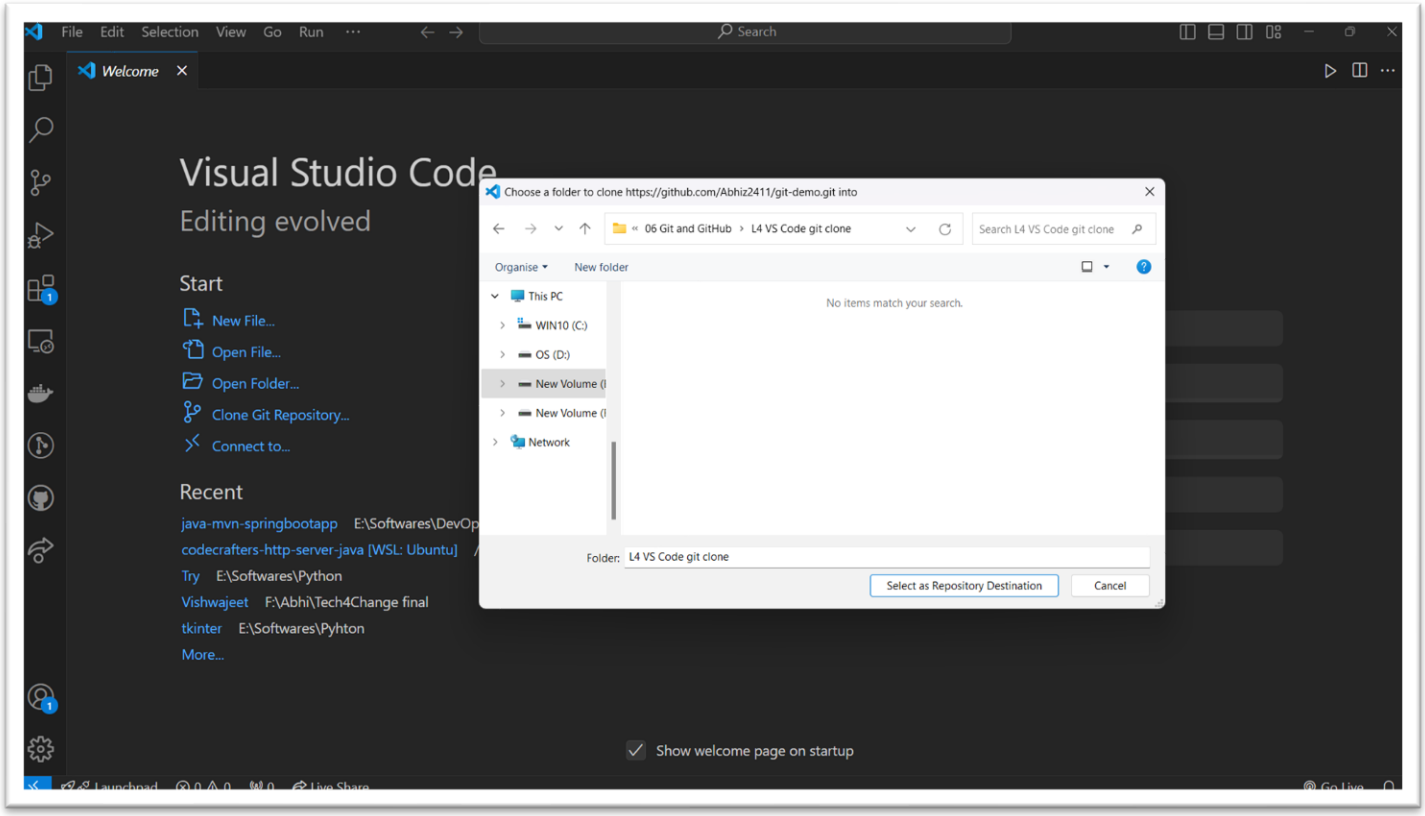

Fig. 4.04: Select the location to clone the repository

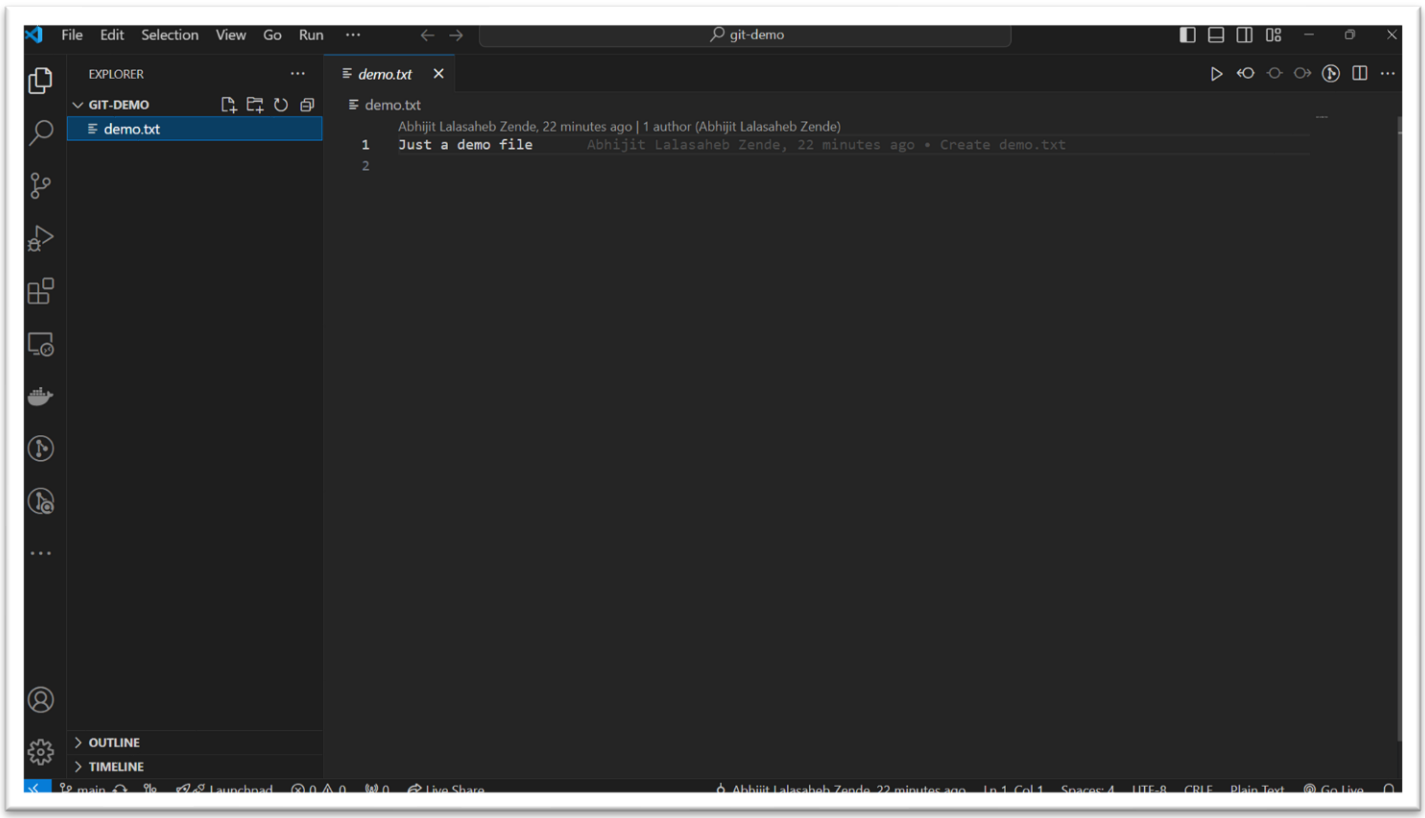

Fig. 4.05: After successful cloning of the repository

5. L5 - Push the incremental changes to GitHub Repository through Visual Studio Code IDE

Step 1: Create a repository if not already exists

Step 2: Download and Install VS Code if not already installed

Step 3: Clone the Repository Using VS Code

1. Open VS Code and go to View > Command Palette.
2. Select Git: Clone and paste the repository URL from GitHub.
3. Select Git: Clone and paste the repository URL from GitHub.

Step 4: Make Changes in VS Code

1. Modify any file or create a new one.

Step 5: Commit Changes

1. Open-Source Control (Ctrl + Shift + G), stage changes, and add a commit message.
2. Click the checkmark to commit.

Step 6: Push Changes to GitHub

1. Click the three dots in Source Control > Push.

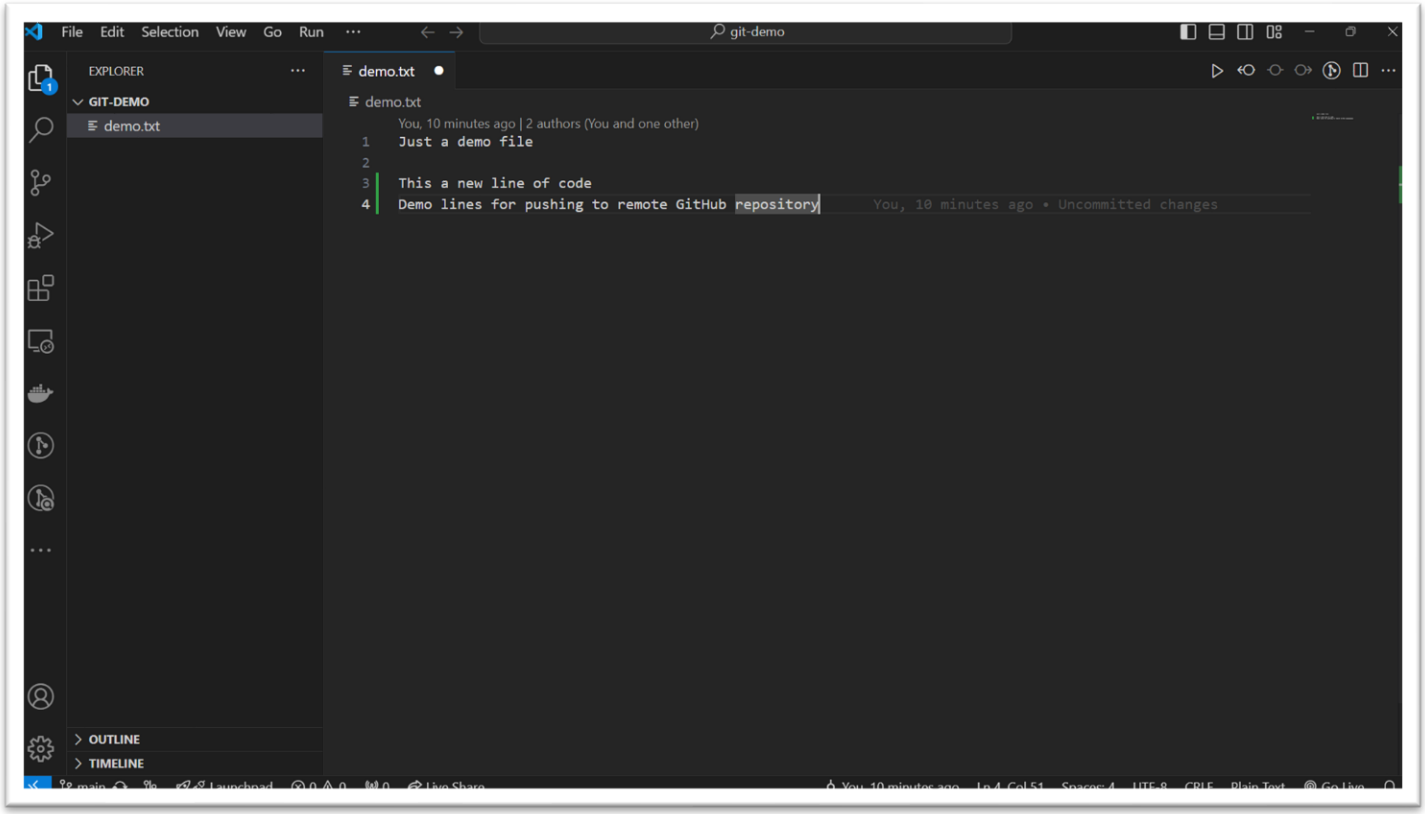

Fig. 5.01: Code in VS code

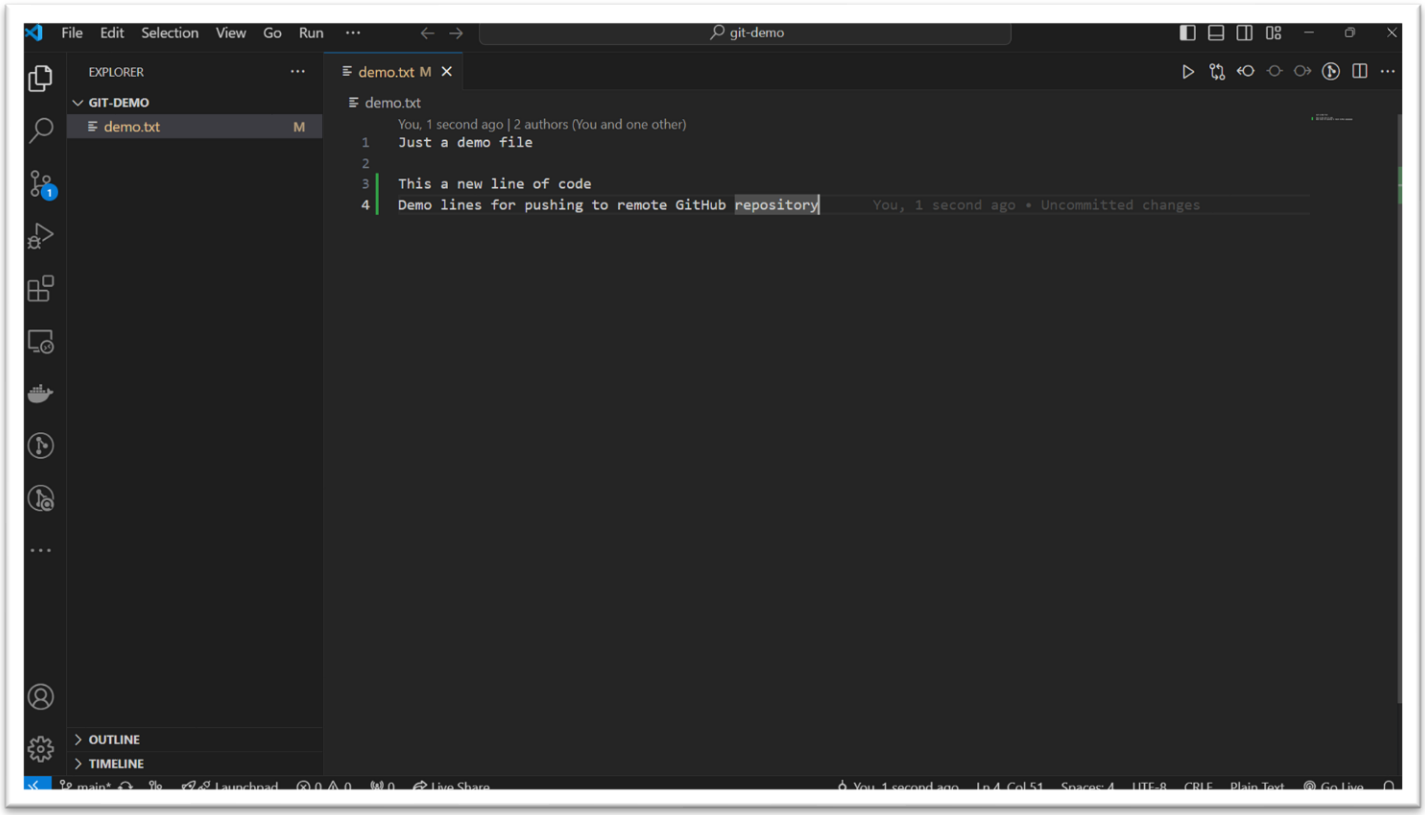

Fig. 5.02: Save file

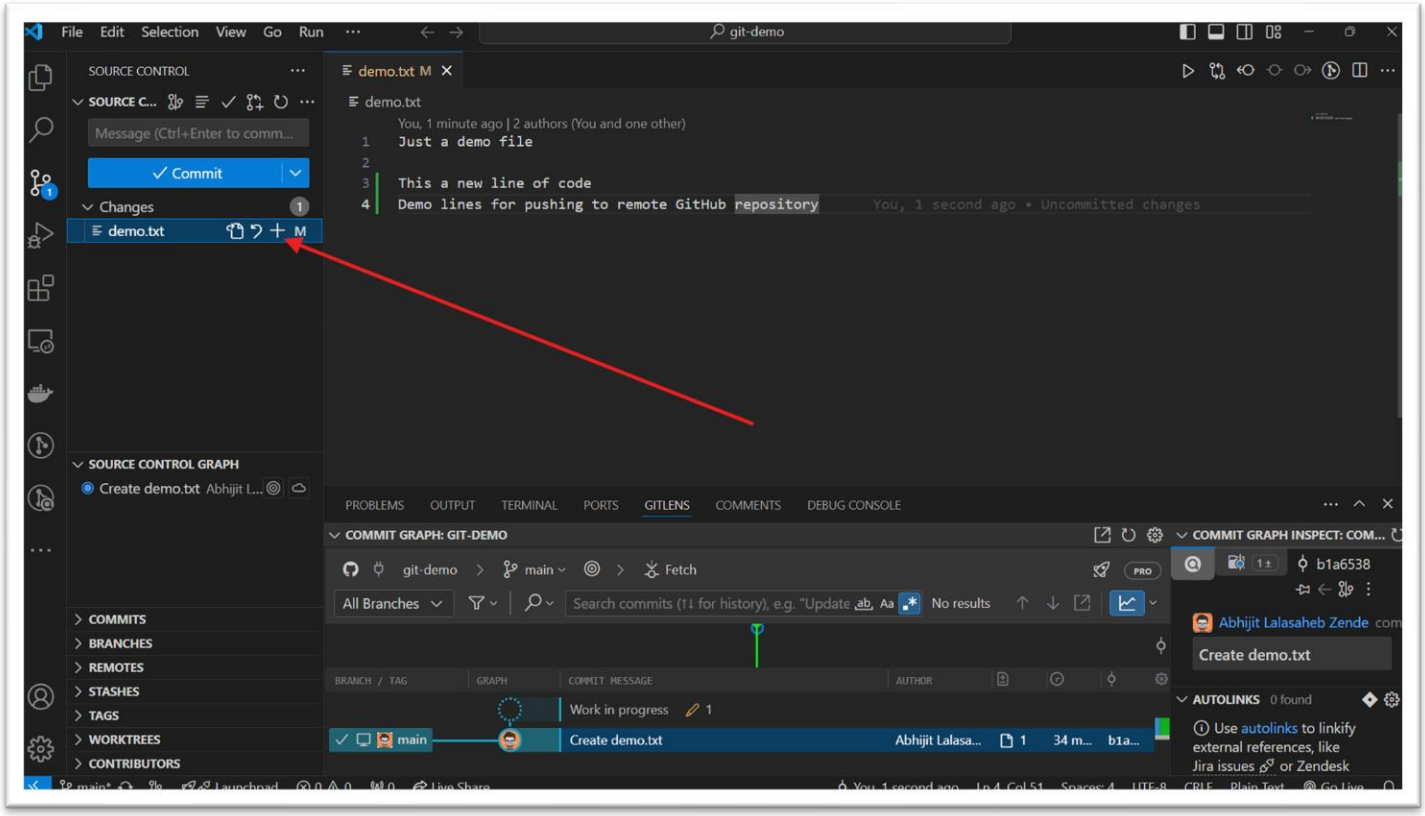

Fig. 5.03: Stage the changes

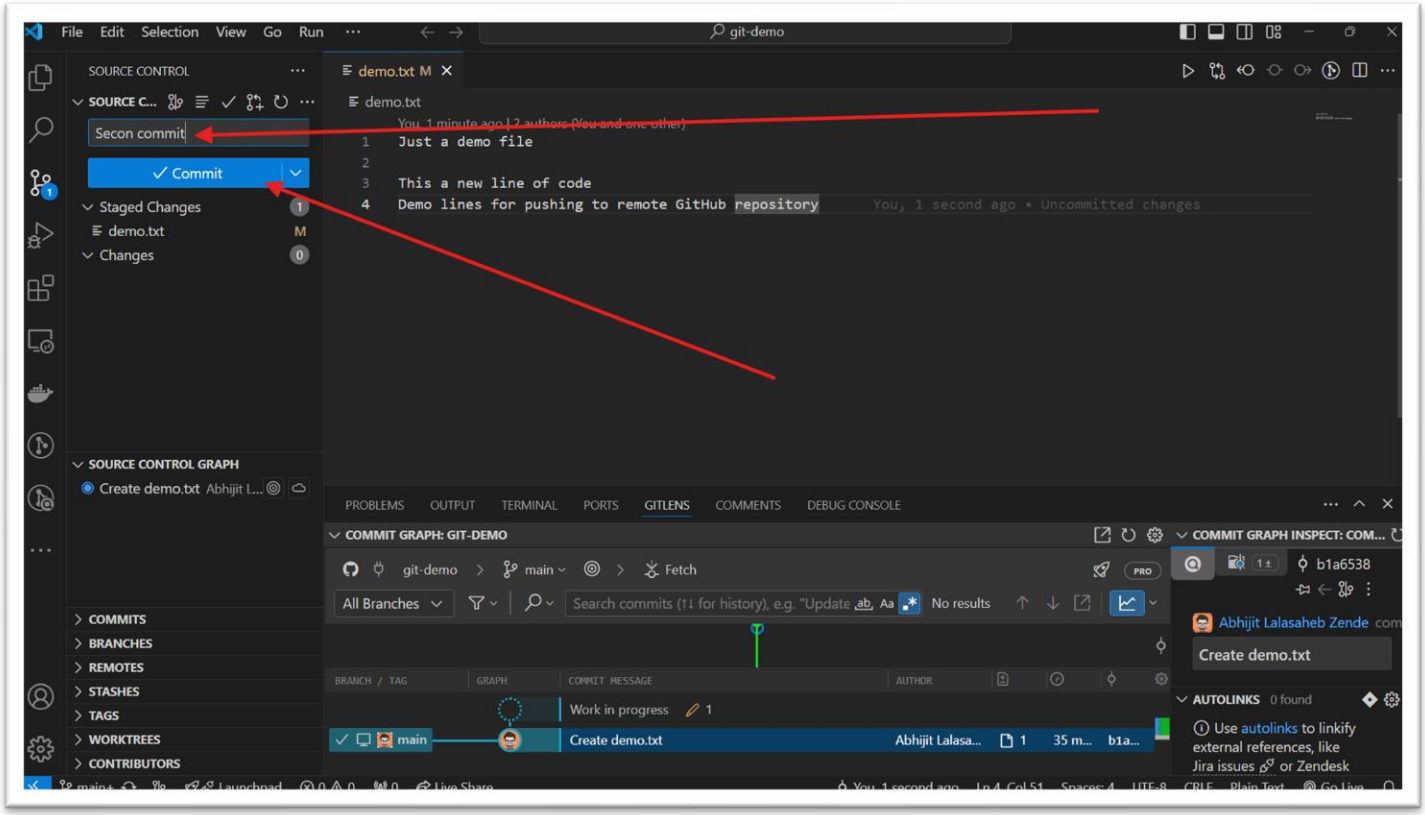

Fig. 5.04: Commit the changes

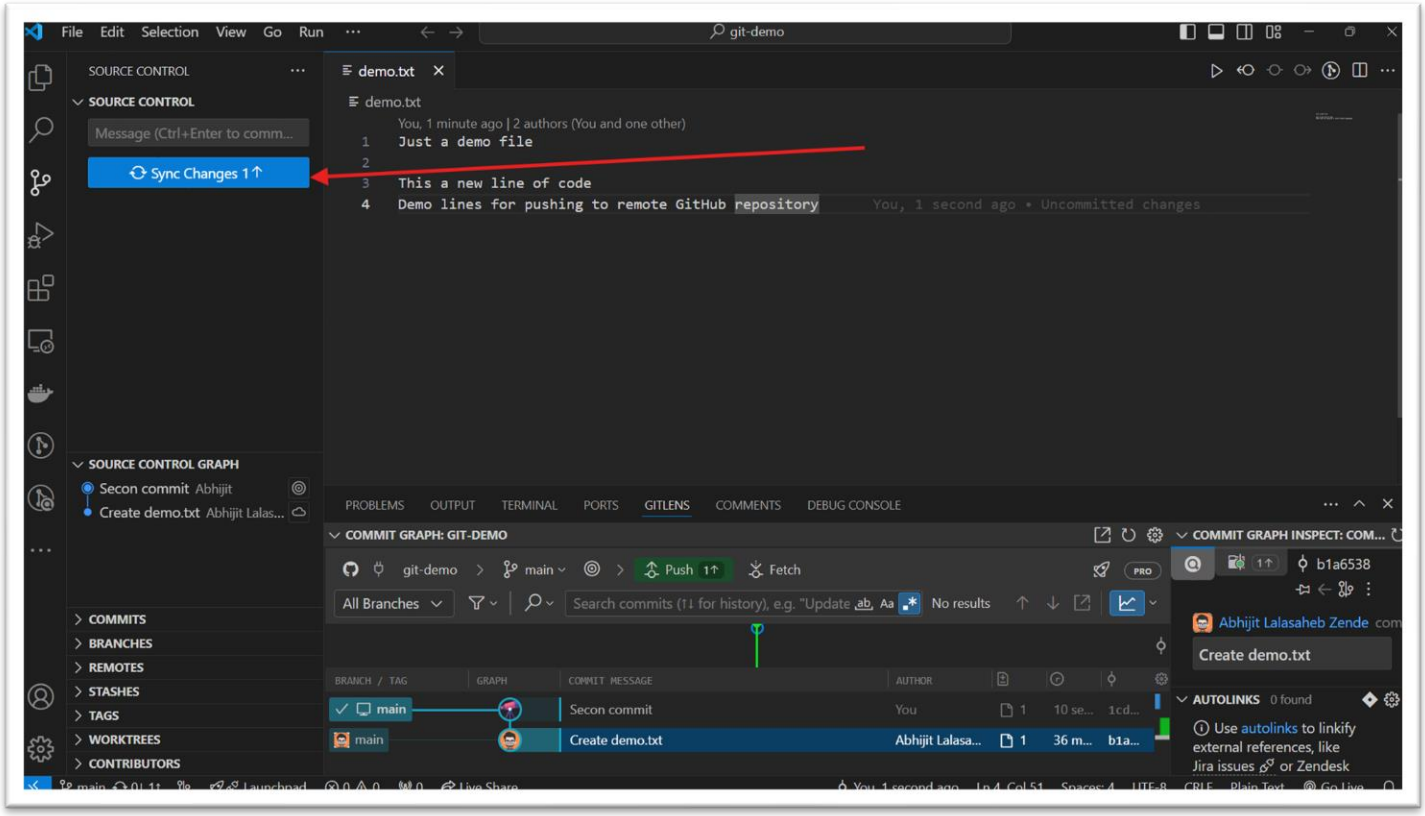

Fig. 5.05: Sync the changes

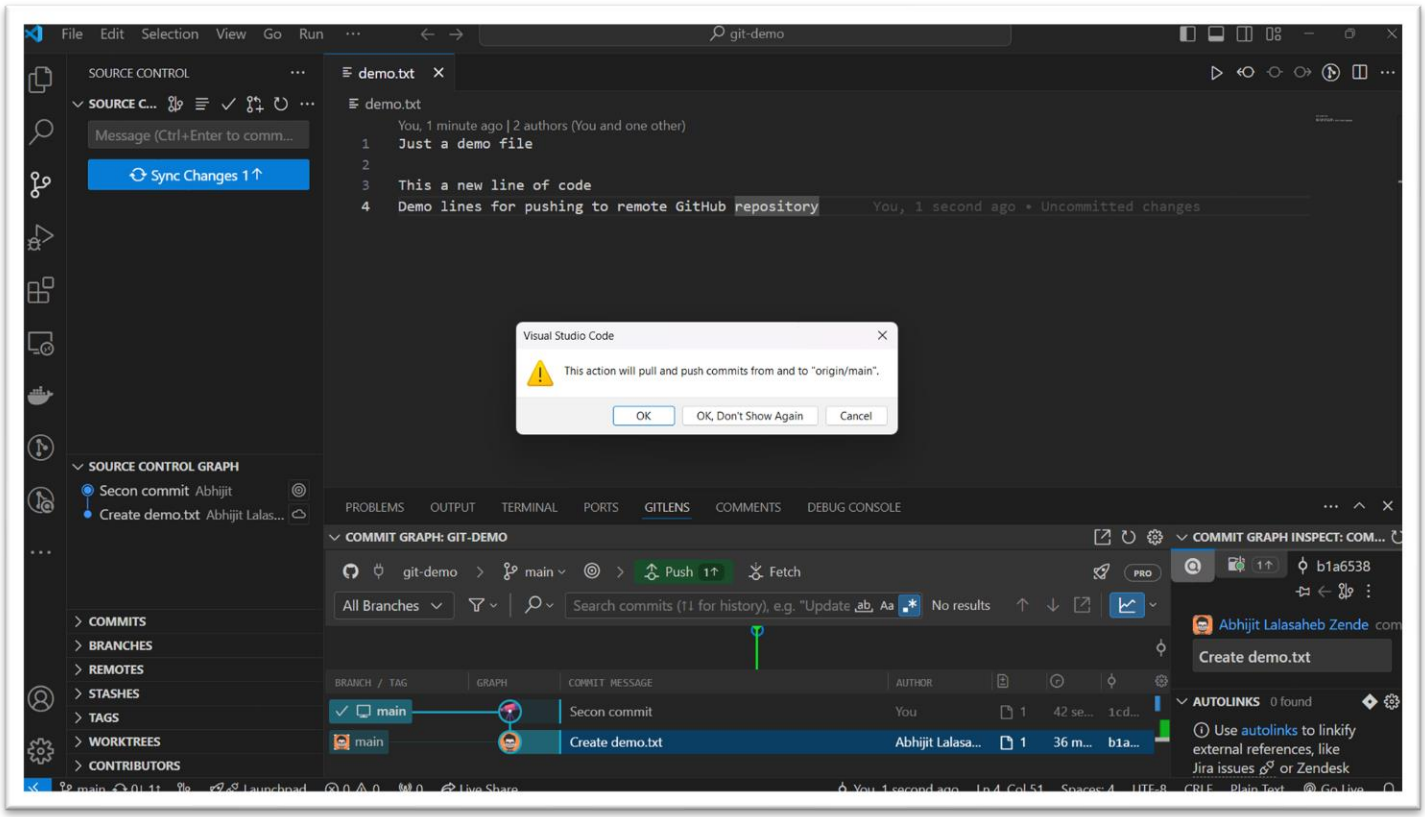

Fig. 5.06: Accept

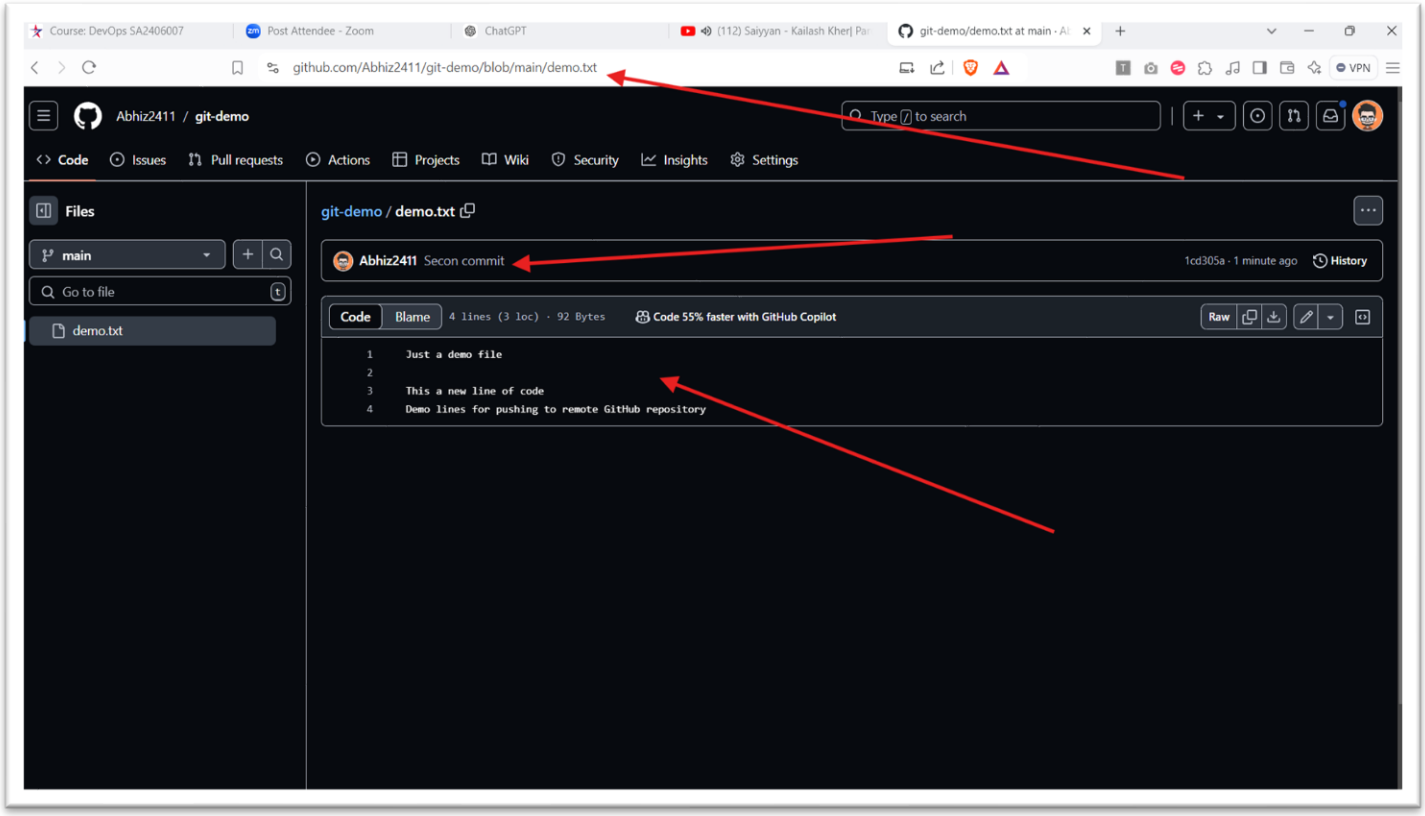

Fig. 5.07: Changes has been done to remote repository