PRACTICAL-6

AIM: Add collaborators on GitHub repository.

- Create a new repo on GitHub and add a collaborator.
- Commit some changes in repo and show Git log.

Theory:

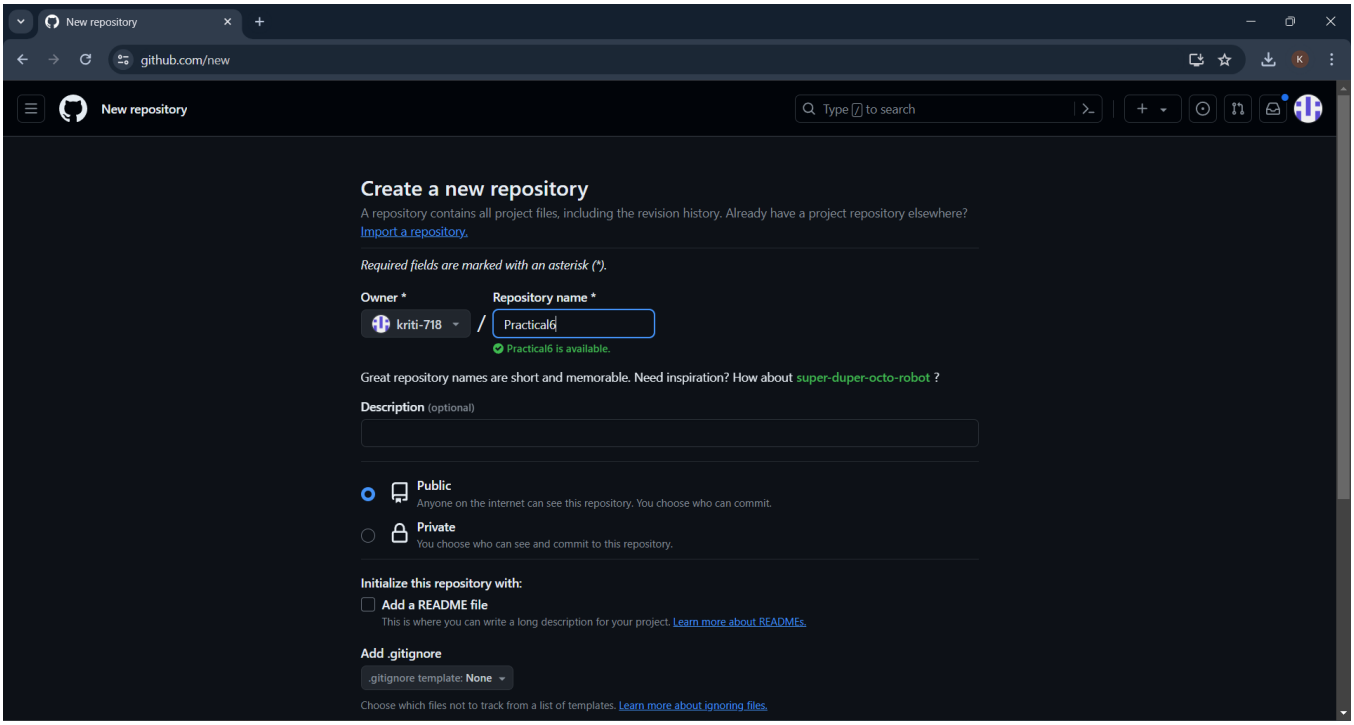
In your public repository in GitHub, no one has the permission to push code into your repository. They can only read the repository. In order to allow other individuals to make changes to your repository, you need to invite them to collaborate to the project.

Understanding Collaboration Roles:

- **Read:** Permission to view the repository's contents, including files, discussions, and commits.
- **Write:** Allows collaborators to make changes to existing files and create new ones. This encompasses pushing commits, creating branches, and raising pull requests.
- **Admin:** Grants extensive control, encompassing all permissions of the "Read" and "Write" roles, along with the ability to manage collaborator access (adding, removing, and modifying permissions).

Steps to invite other team members to collaborate with your repository:

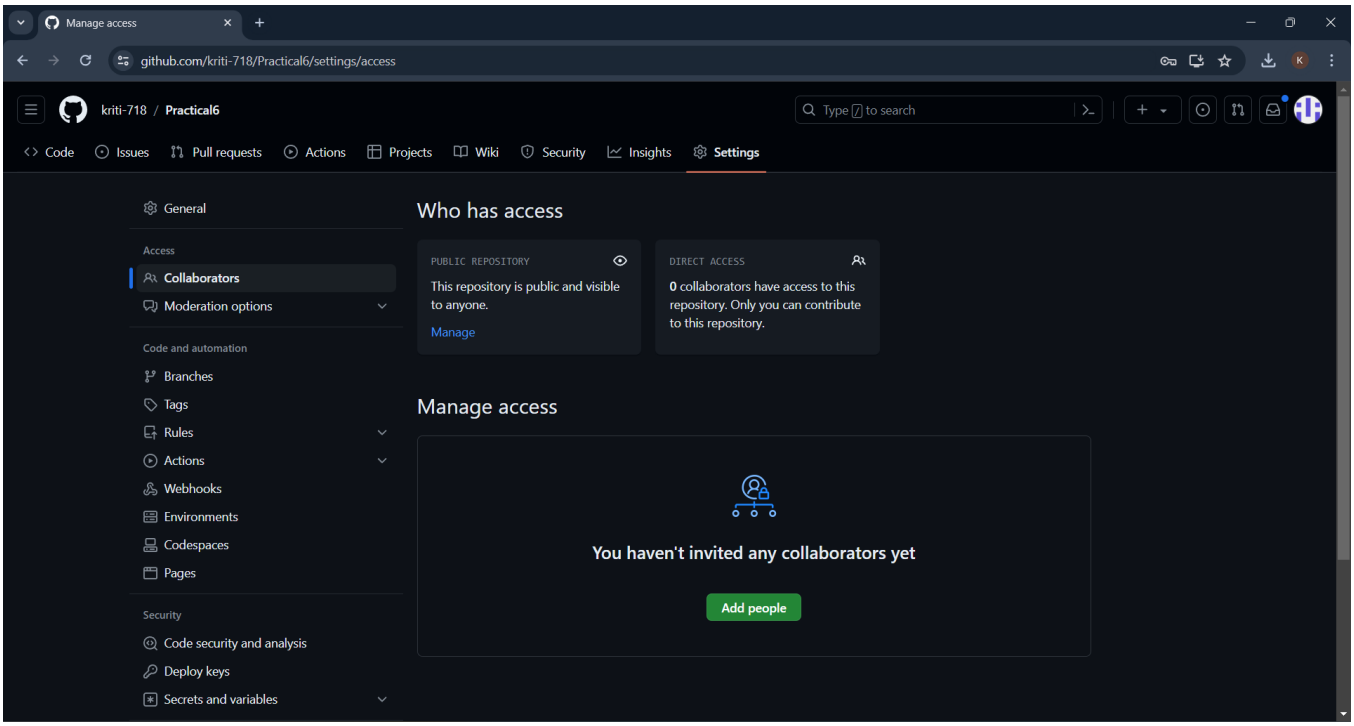
1. Firstly, create a repository on GitHub



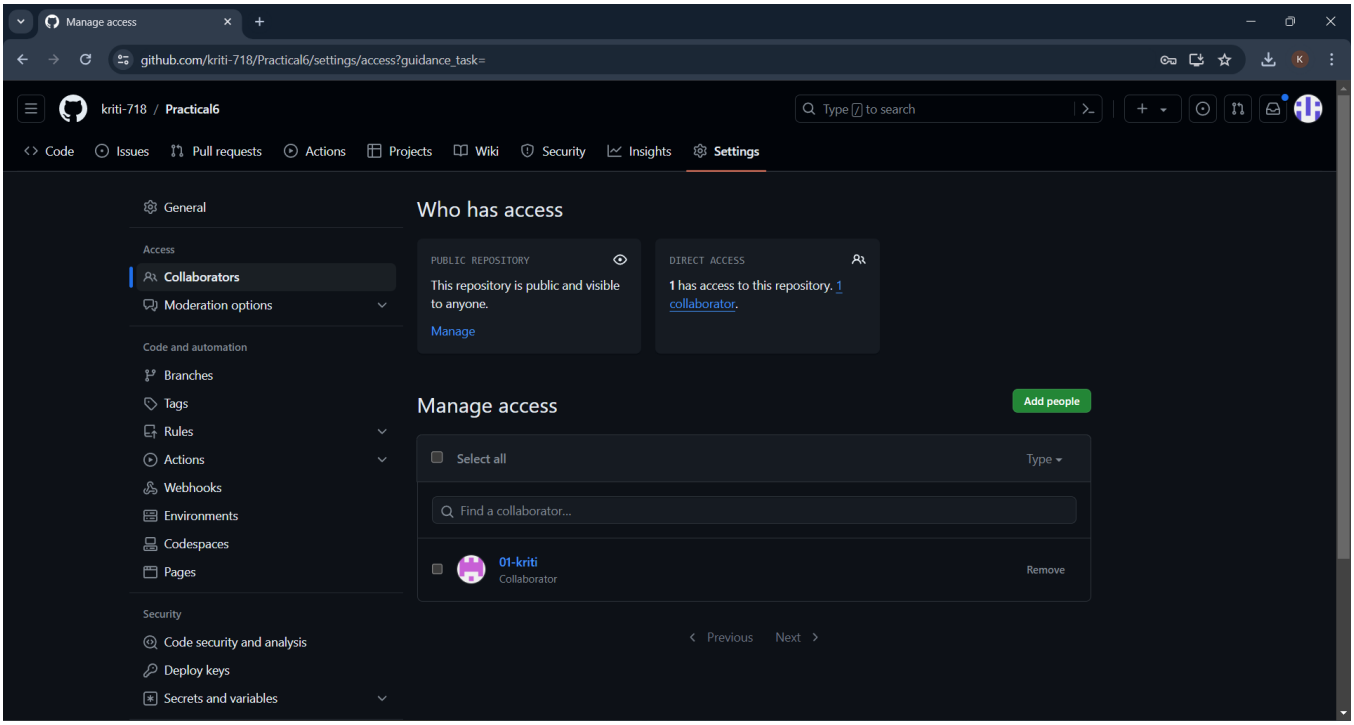
2. Create a new repo or push an existing one using these commands.

```
PS C:\Users\kriti\OneDrive\Desktop\SCM\Practical6> git branch -M main
PS C:\Users\kriti\OneDrive\Desktop\SCM\Practical6> git remote add origin git@github.com:kriti-718/Practical6.git
PS C:\Users\kriti\OneDrive\Desktop\SCM\Practical6> git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 244 bytes | 244.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:kriti-718/Practical6.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
PS C:\Users\kriti\OneDrive\Desktop\SCM\Practical6>
```

3. Now, invite your team member for collaboration on the repo and work on this project in collaboration.



4. GitHub provides us feature to manage our collaborators on our GitHub repository.



5. Now, see how beautifully Git tell us about changes and who have done these changes by running the command “git log”.

```
PS C:\Users\kriti\OneDrive\Desktop\SCM\Practical6> git log
commit 40d89278b049bc3527fcdd7ac796383aa35fa04a (HEAD -> main, origin/main)
Author: 01-kriti <154600653+01-kriti@users.noreply.github.com>
Date:   Fri Mar 29 19:14:25 2024 +0530

    Update Practical6

commit aa0e51d9667d9fe46397c0642568e43ed2e02512
Author: kriti-718 <156739389+kriti-718@users.noreply.github.com>
Date:   Fri Mar 29 19:14:00 2024 +0530

    Update Practical6

commit 96aa7fdd9a9fe3bc4c23a76a194572c599180014
Author: Kriti <krit0781.be23@chitkara.edu.in>
Date:   Fri Mar 29 18:58:07 2024 +0530

    First Commit
```

PRACTICAL-7

AIM: Fork and commit

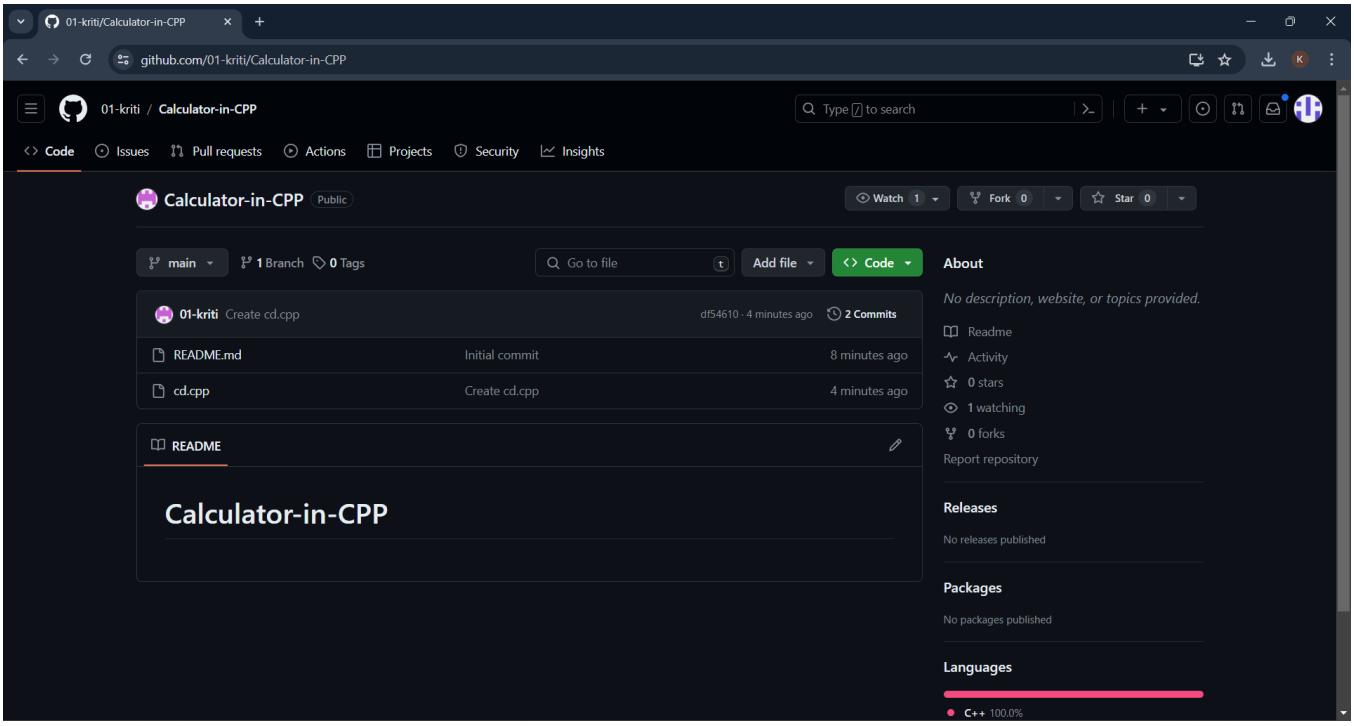
- Create a fork of repository and make open pull request.

Theory:

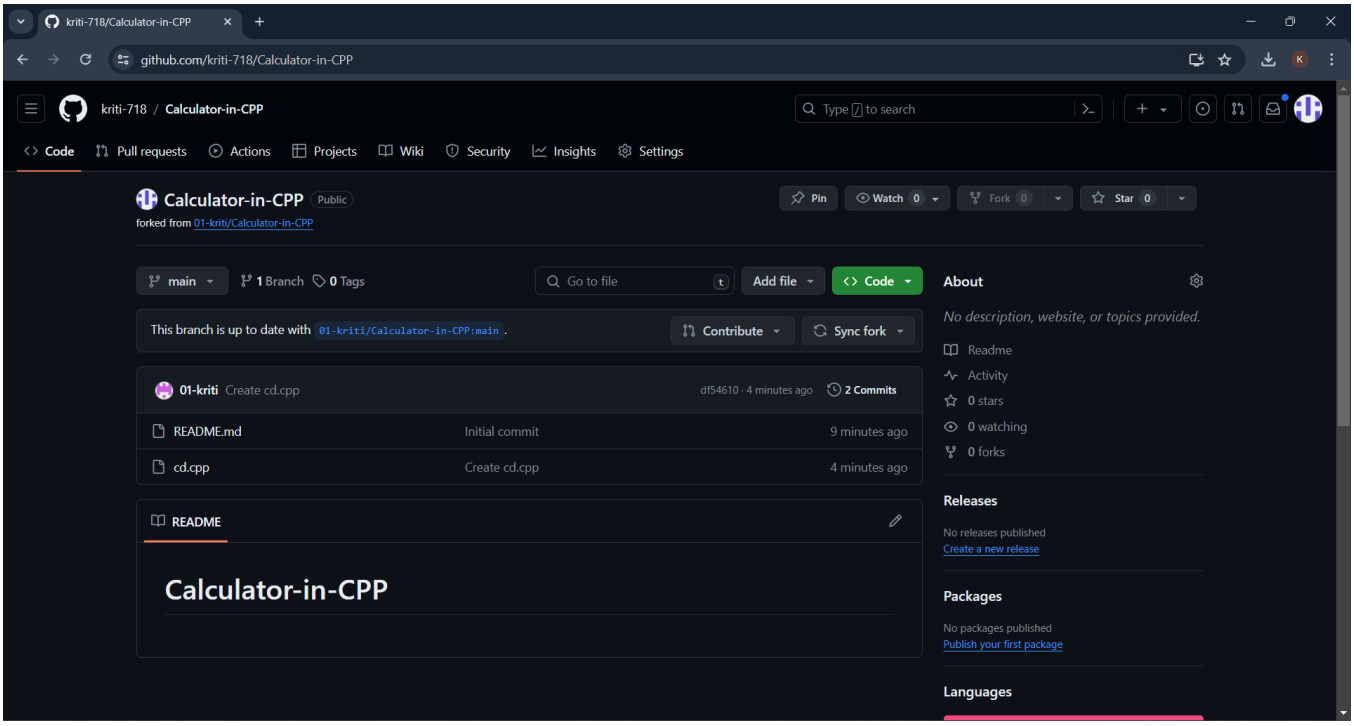
A fork is a new repository that shares code and visibility settings with the original "upstream" repository. Forks are often used to iterate on ideas or changes before they are proposed back to the upstream repository, such as in open-source projects or when a user does not have write access to the upstream repository.

Steps of creating "Fork" of repository in GitHub:

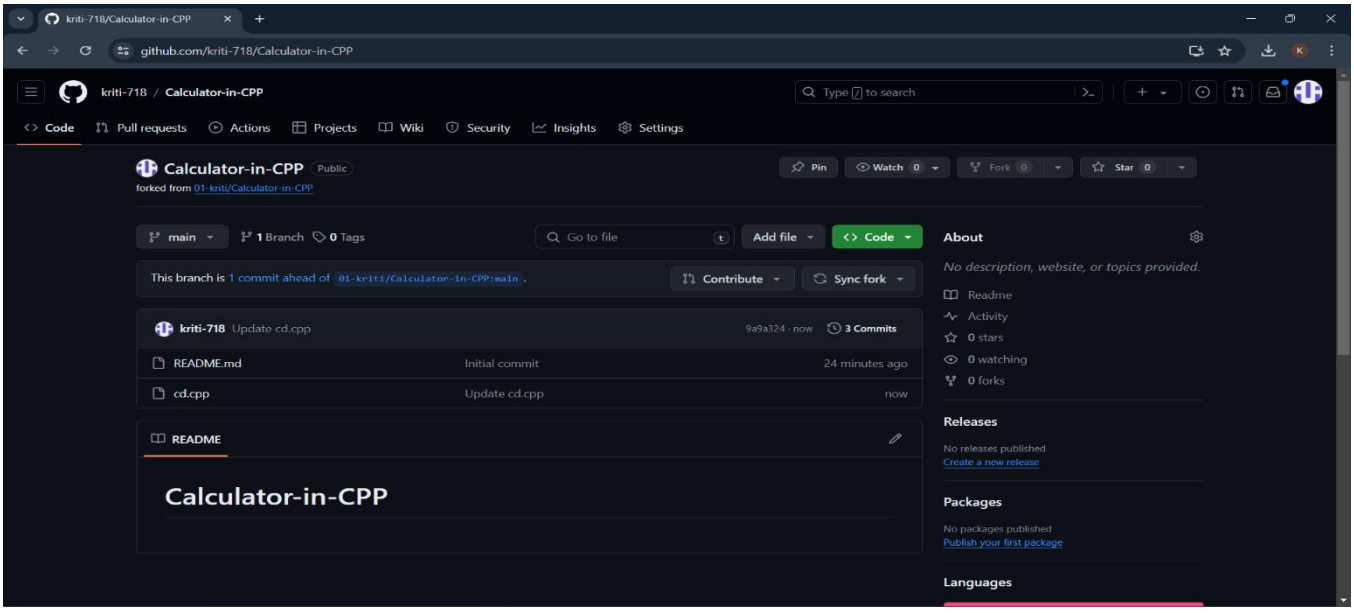
1. Firstly, open the repository which you wanted to create "Fork" by clicking the fork option given on top-right corner as shown in picture given below.



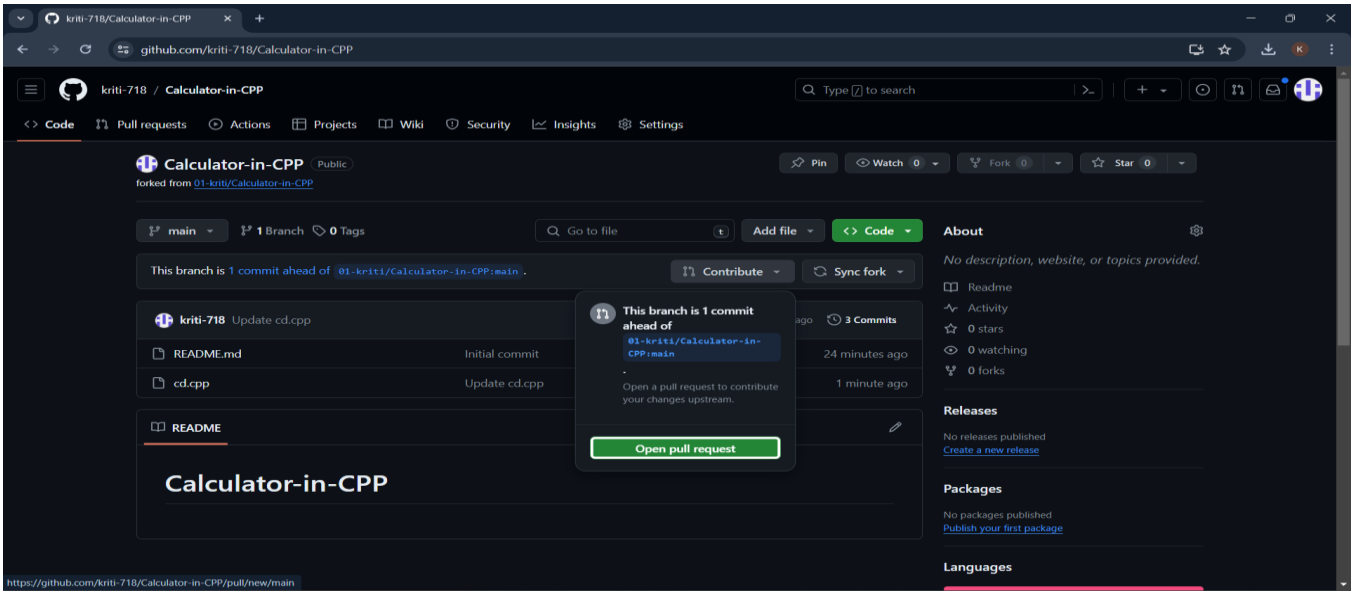
2. After creating “Fork”, GitHub platform is designed in this way it will automatically create a link with original repository and tells user the fork repo is ‘up to date’ or it is ‘ahead’ or it is ‘behind’ of original repository.



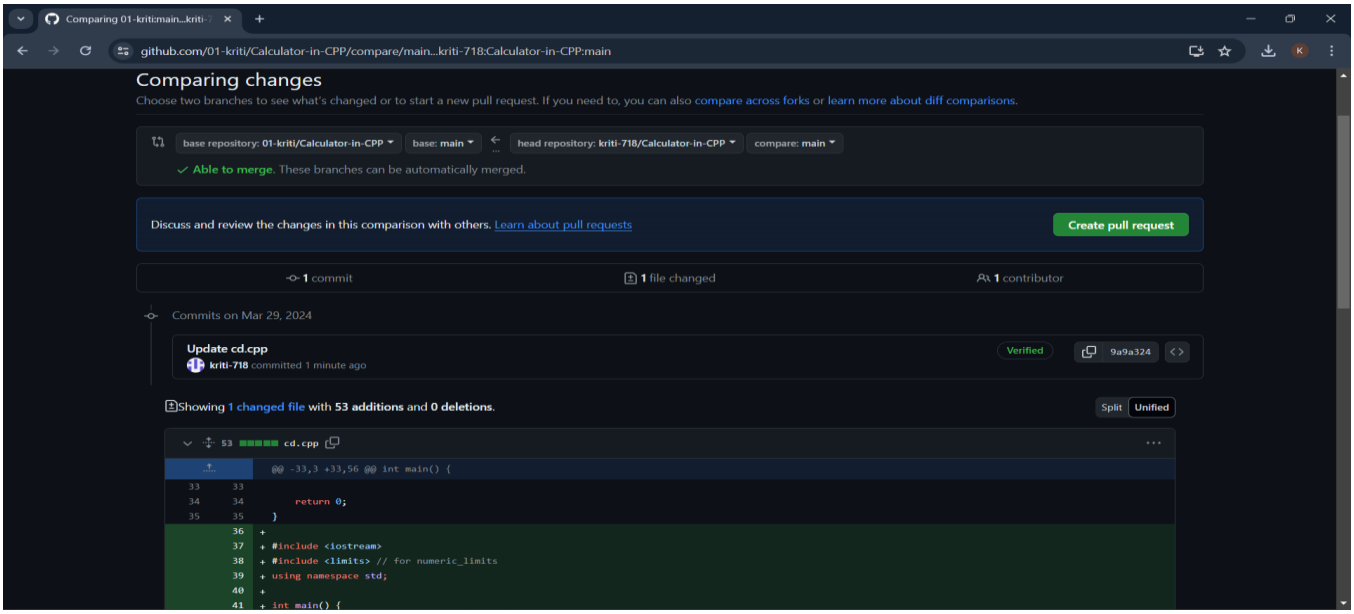
3. When user make some changes in “Fork” repo GitHub tells user that now this repo isahead of original repo to make this change on original repository user have to create the open pull request and ask the owner of repository.



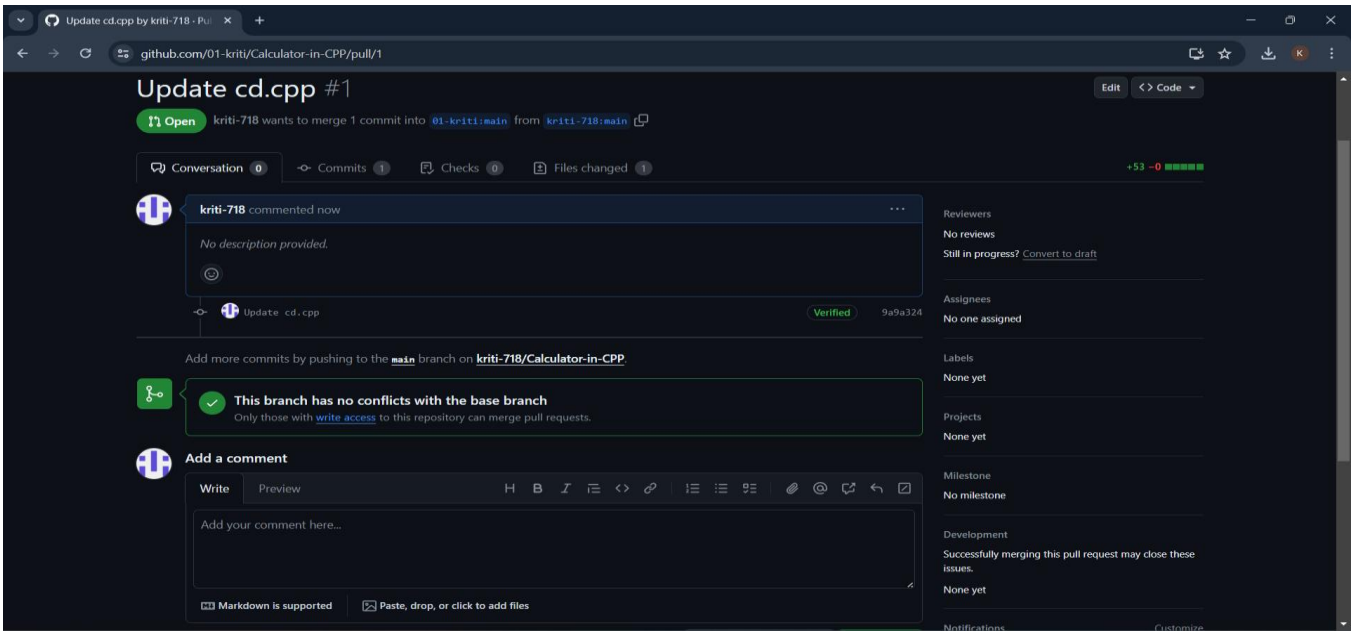
4. To create open pull request click on the contribute option as shown in picture given below.



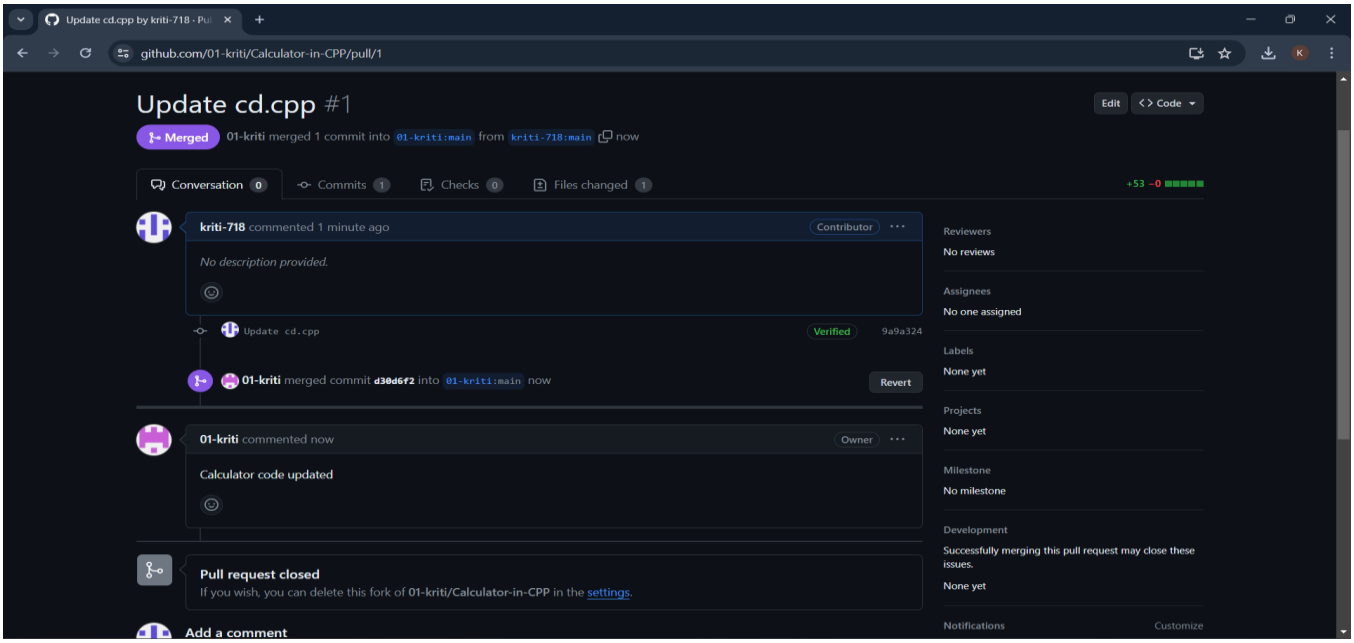
5. Before creating open pull request GitHub compare all the changes, so that user can confirm their change. It will highlights the changes what user have make changes on the repo after confirming user can proceed and create open pull request.



6. Now, user have to wait for confirmation from owner if he accepts user's changes it will automatically reflected on original repository.



7. When owner will accept user's open-pull request then request will automatically closed as shown in picture given below.



PRACTICAL-8

AIM: Merge & resolve conflicts created due to own activity & collaborator activity.

Theory:

Working with version control systems such as Git, most merge conflicts resolve automatically. However, there are situations where git merge is unable to resolve an issue.

Some examples of merge conflicts include:

- Changing the same lines of code in a file.
- Removal of files while changes happen in another place.

Since the problem happens locally and the rest of the project members are unaware of the issue, resolving the conflict is of high priority and requires an immediate fix.

Types Of Git Merge Conflicts:

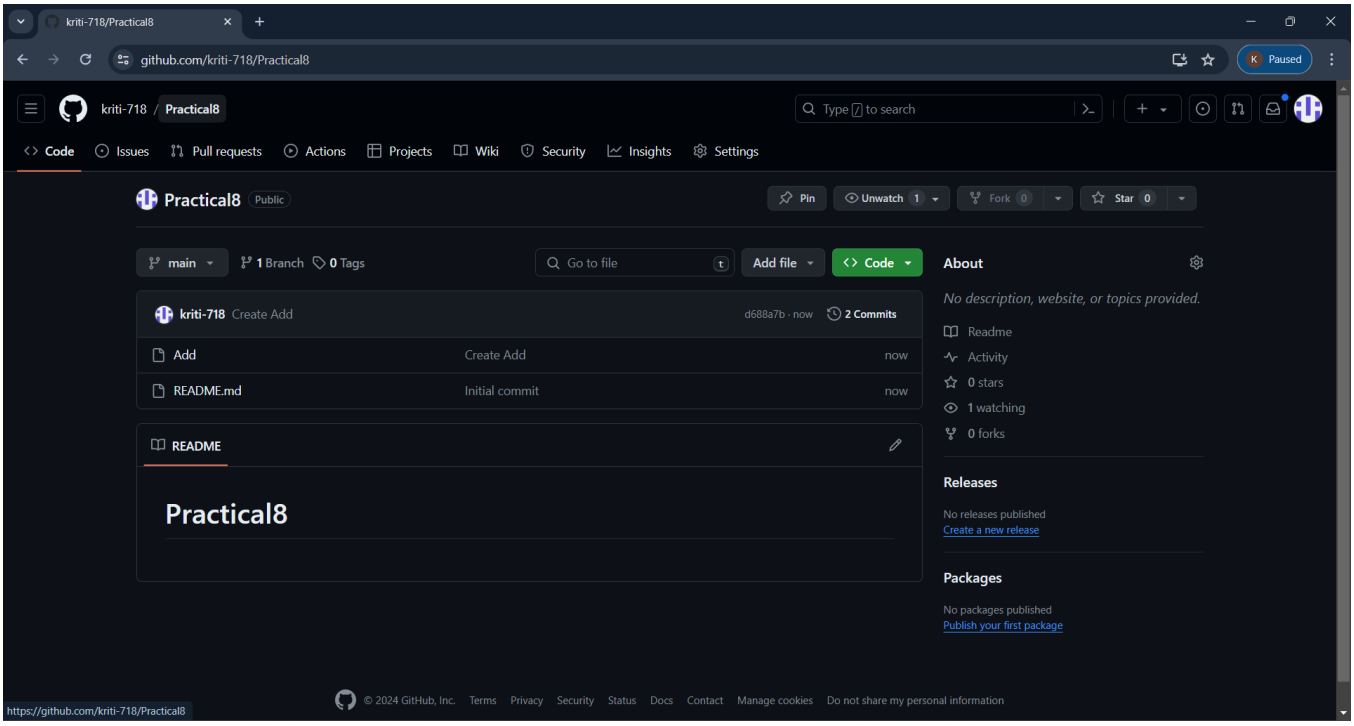
The general types of merge conflicts depend on when the issue appears.

The conflicts happen either:

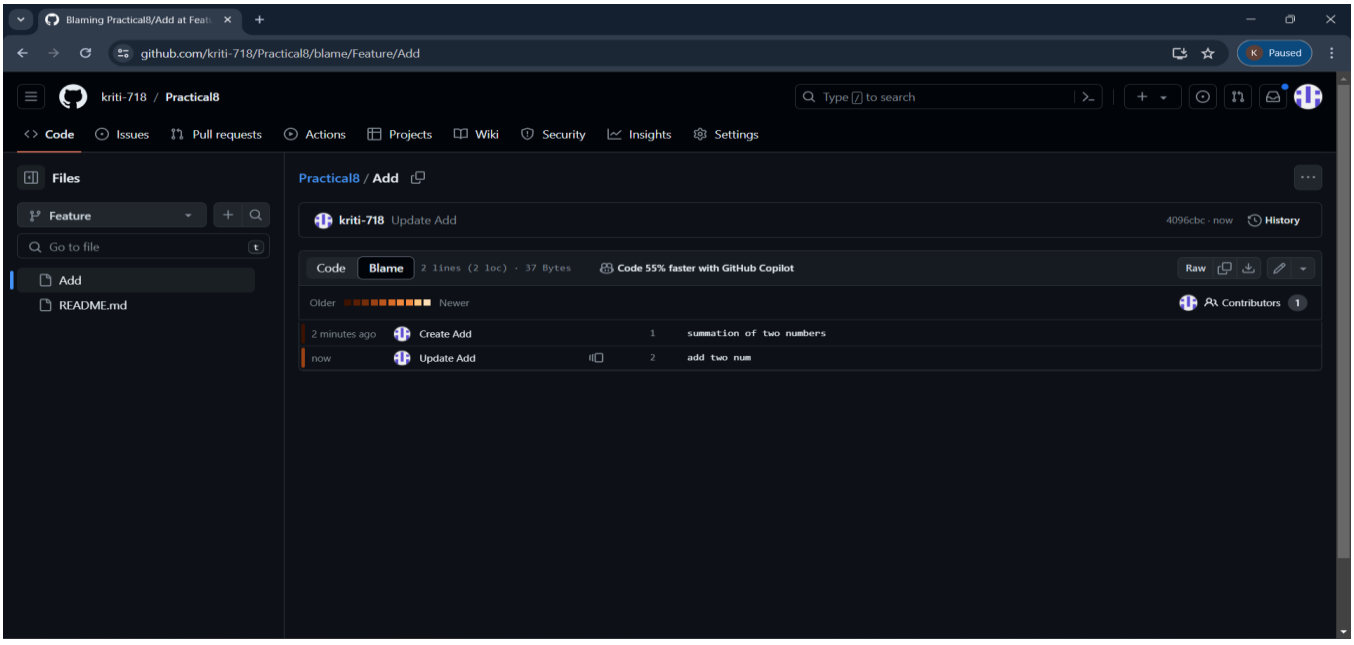
- **Before merging**, indicating there are local changes not up to date. The conflict error message appears before the merge starts to avoid issues.
- **During the merge**, indicating an overwrite issue. The error message appears and stops the merging process to avoid overwriting changes.

- **Steps for resolving conflicts created due to own activity in repository:**

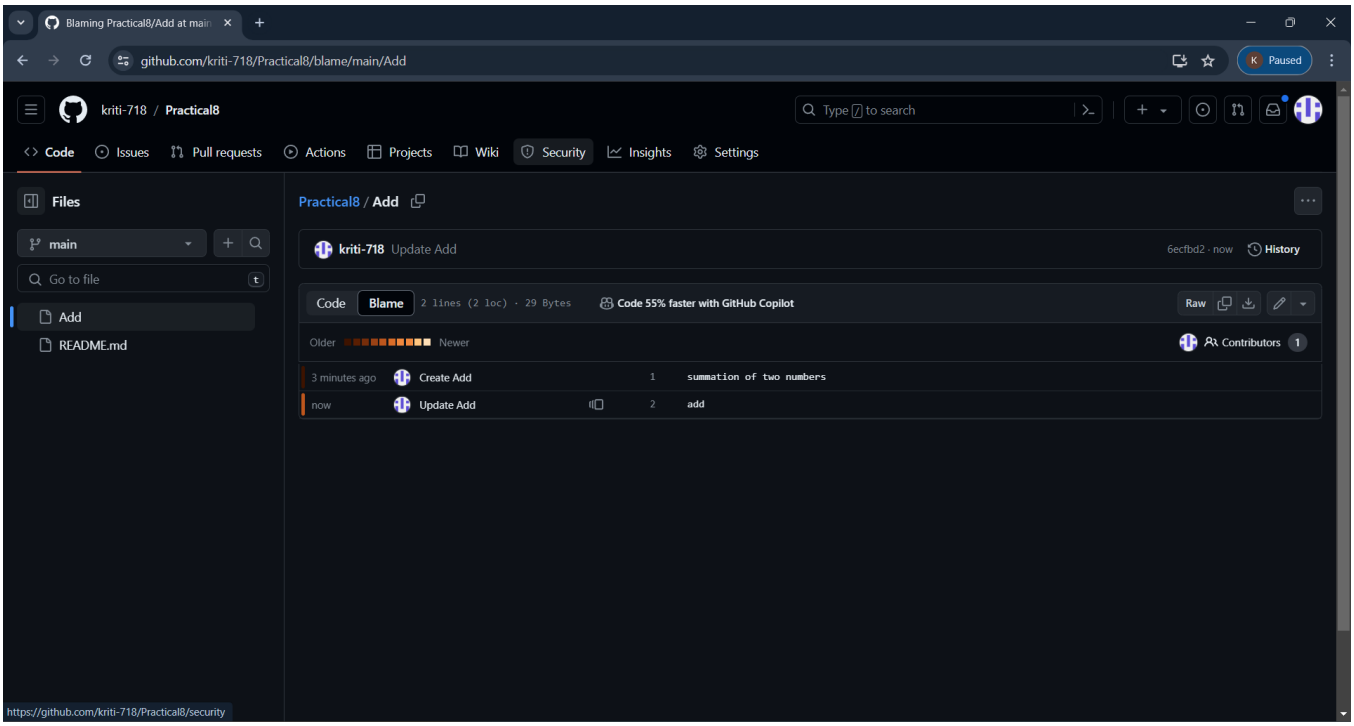
1. Firstly, created a new repository and add some file in the repository



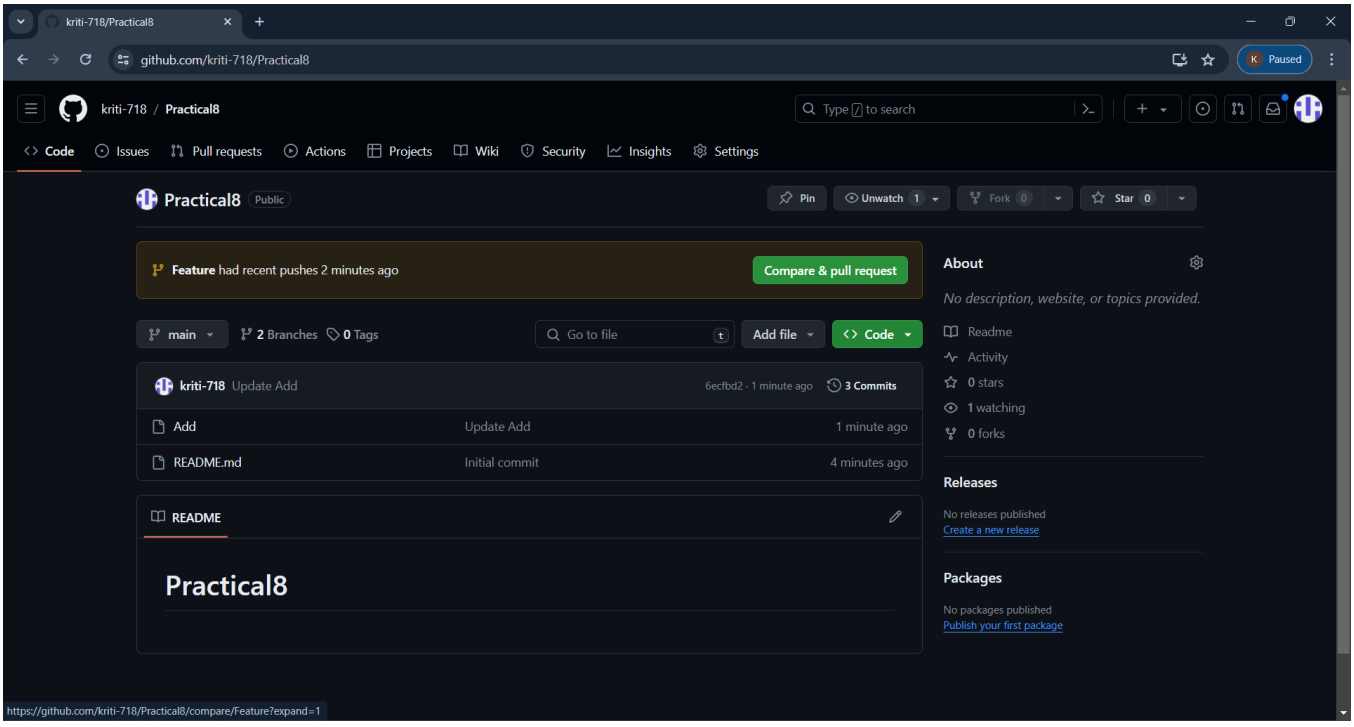
2. Now, make a new branch and make some changes in the file.



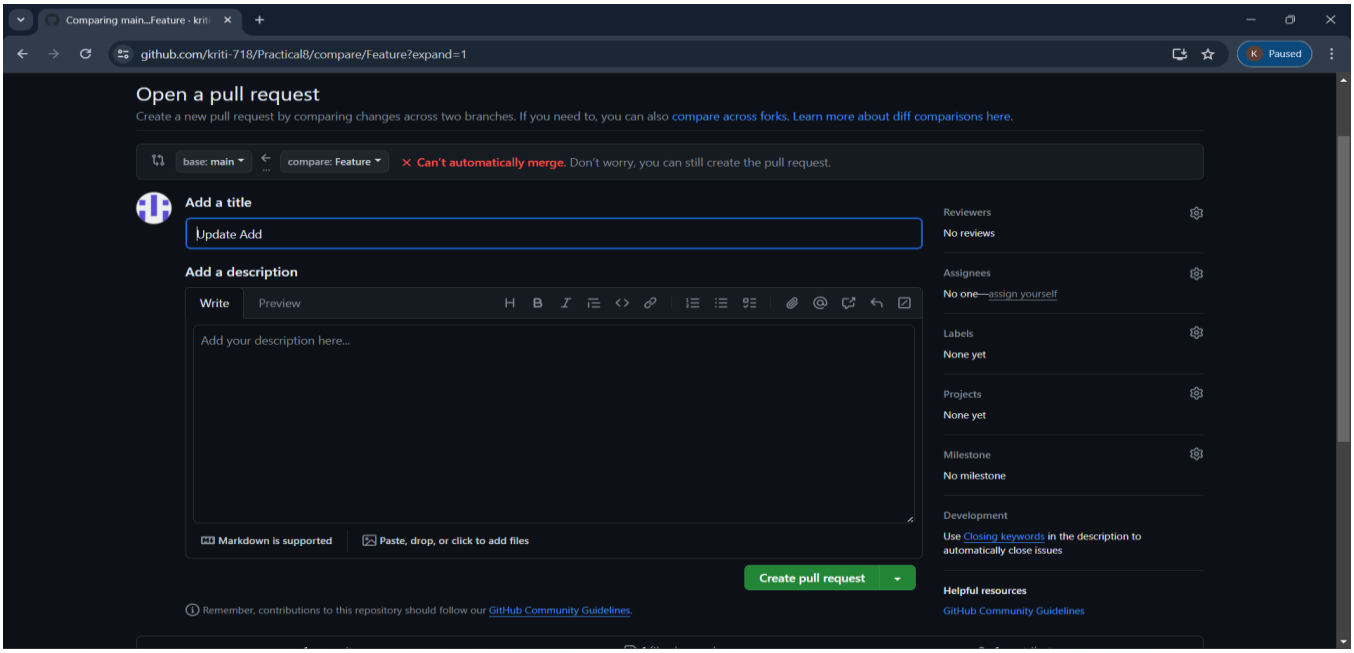
3. Now, switch back to the main branch and again make some changes in the same file.



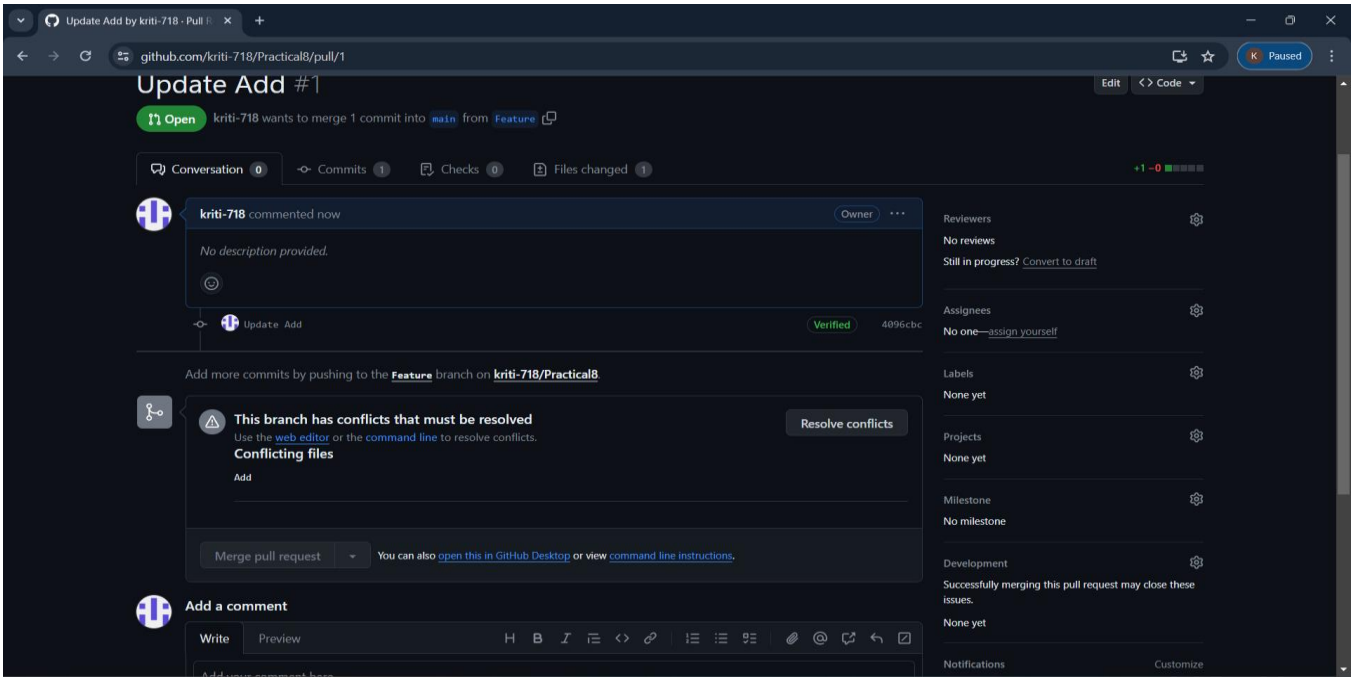
4. Now, try to merge these two branches by creating a pull request.



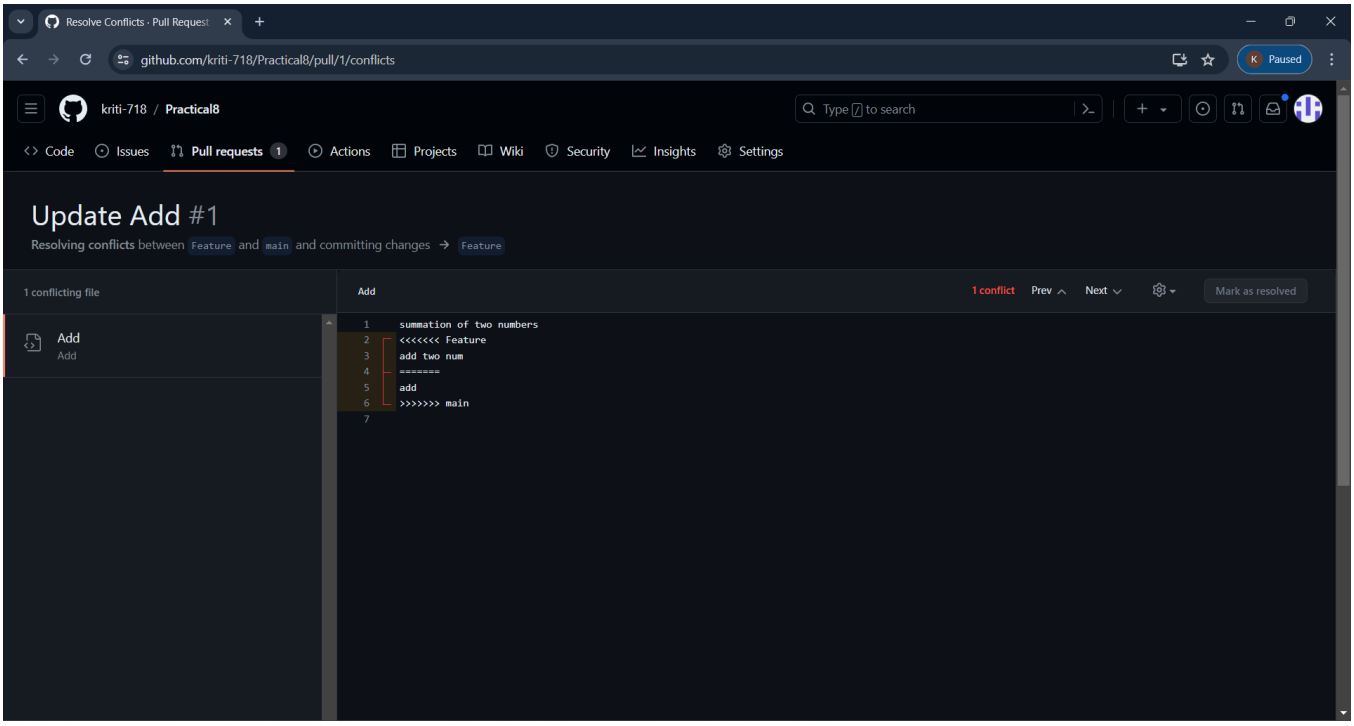
5. When we tried to merge these branches it displays error that it cant merge these branches automatically. So we will try to merge these branches manually by creating a pull request.



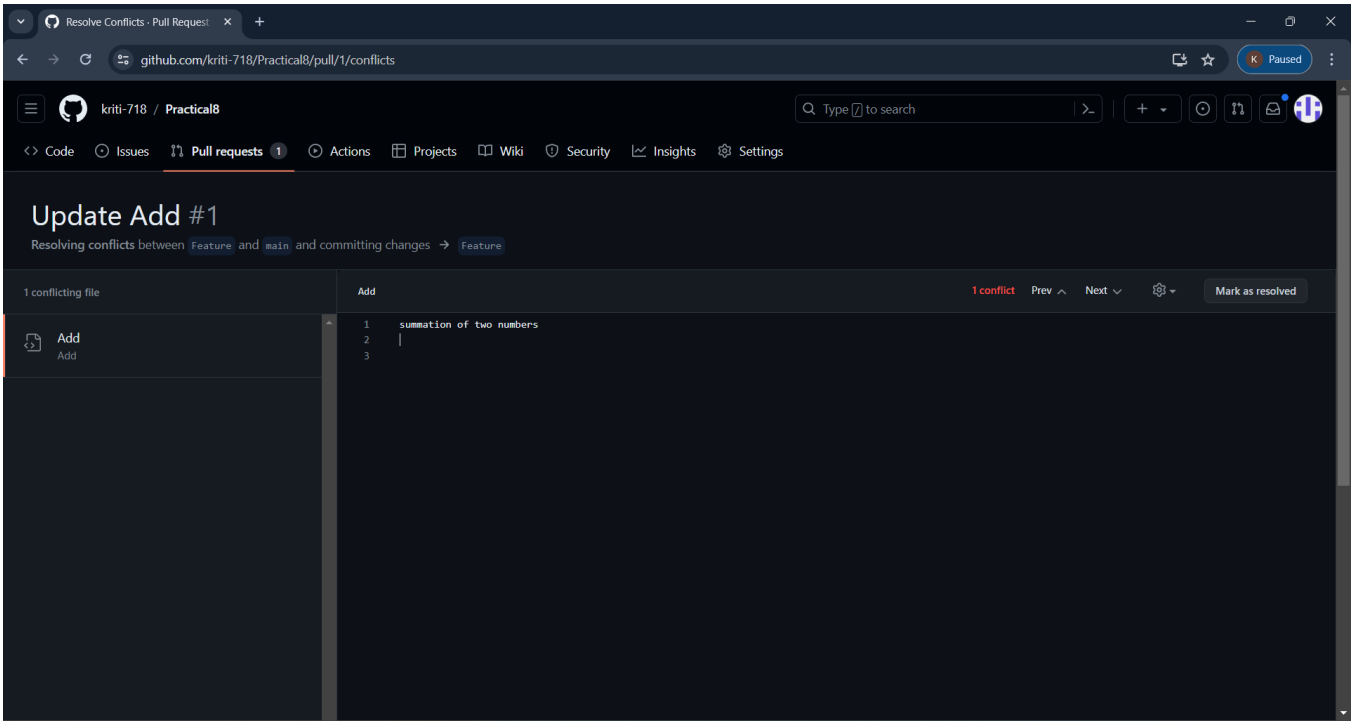
6. This shows an error that there is some conflict in the merging of the two branches.



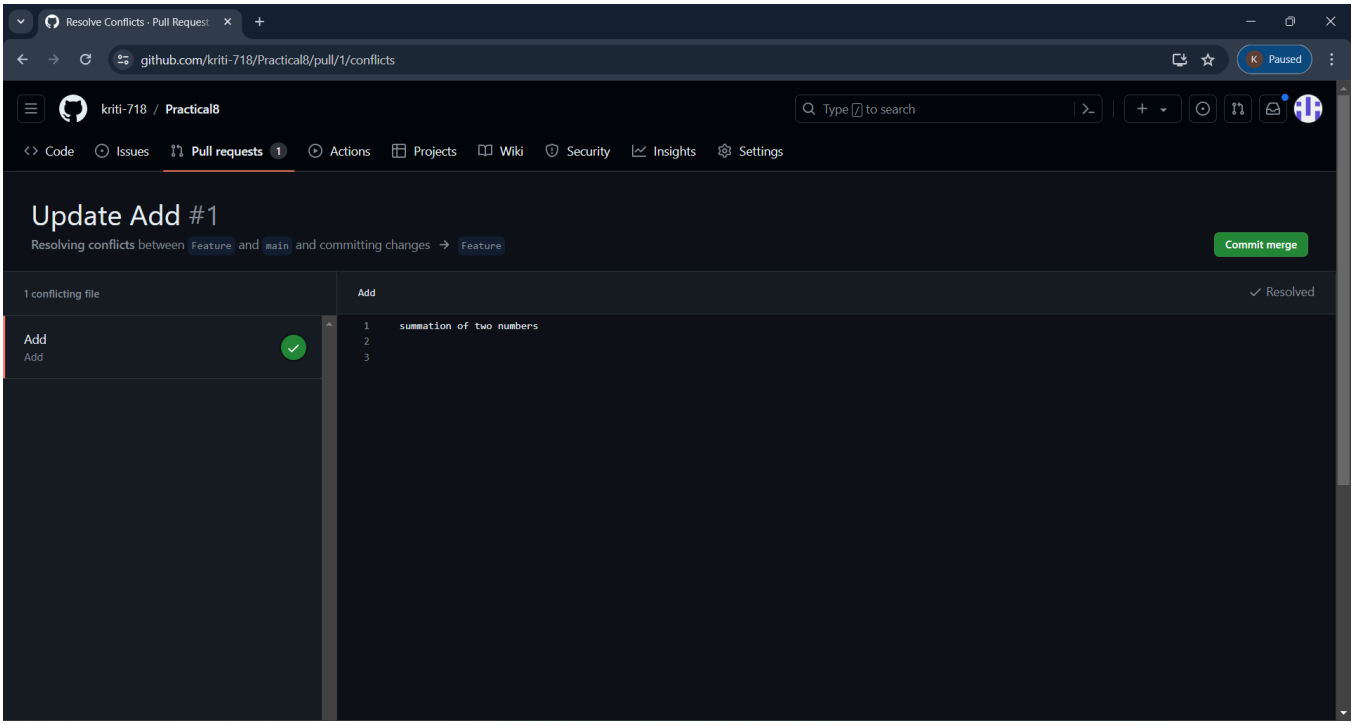
7. Now click on resolve conflicts option, this will suggest up some techniques to resolve the conflicts.



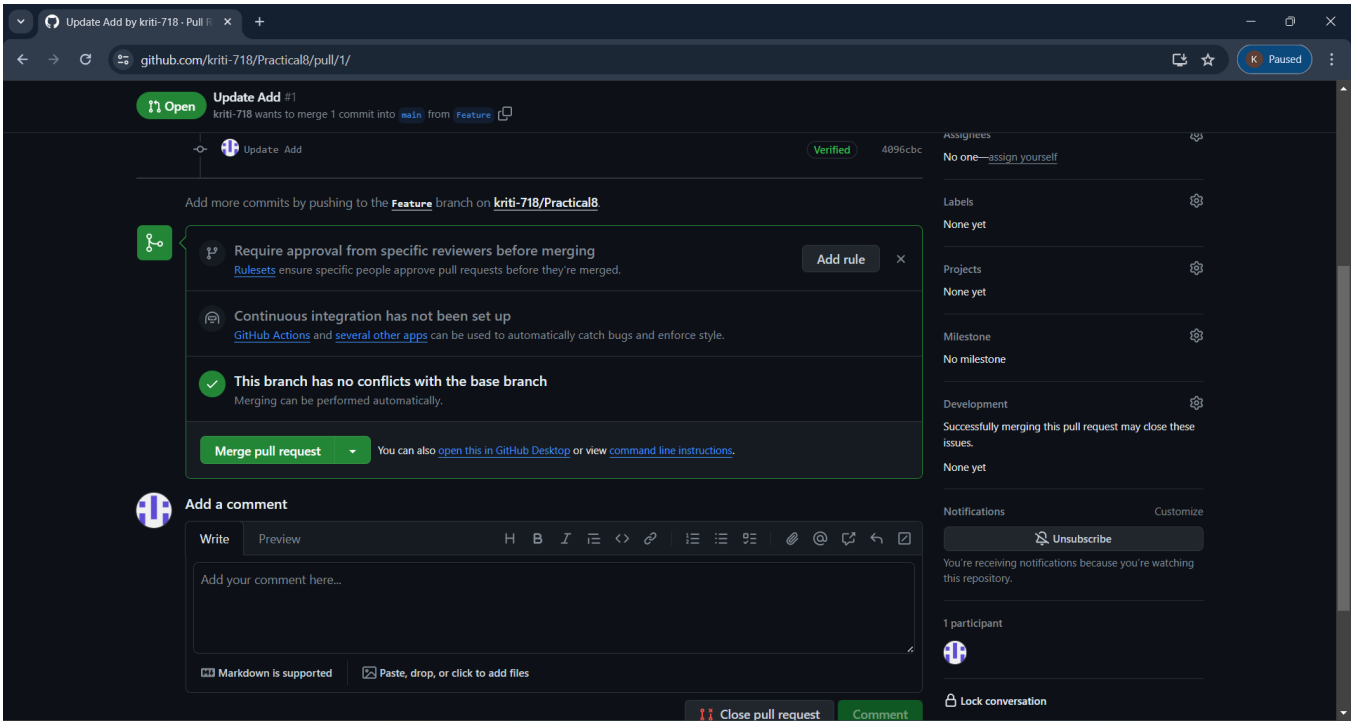
8. Resolve the conflict by changing the content of the files. And click on mark as resolved.



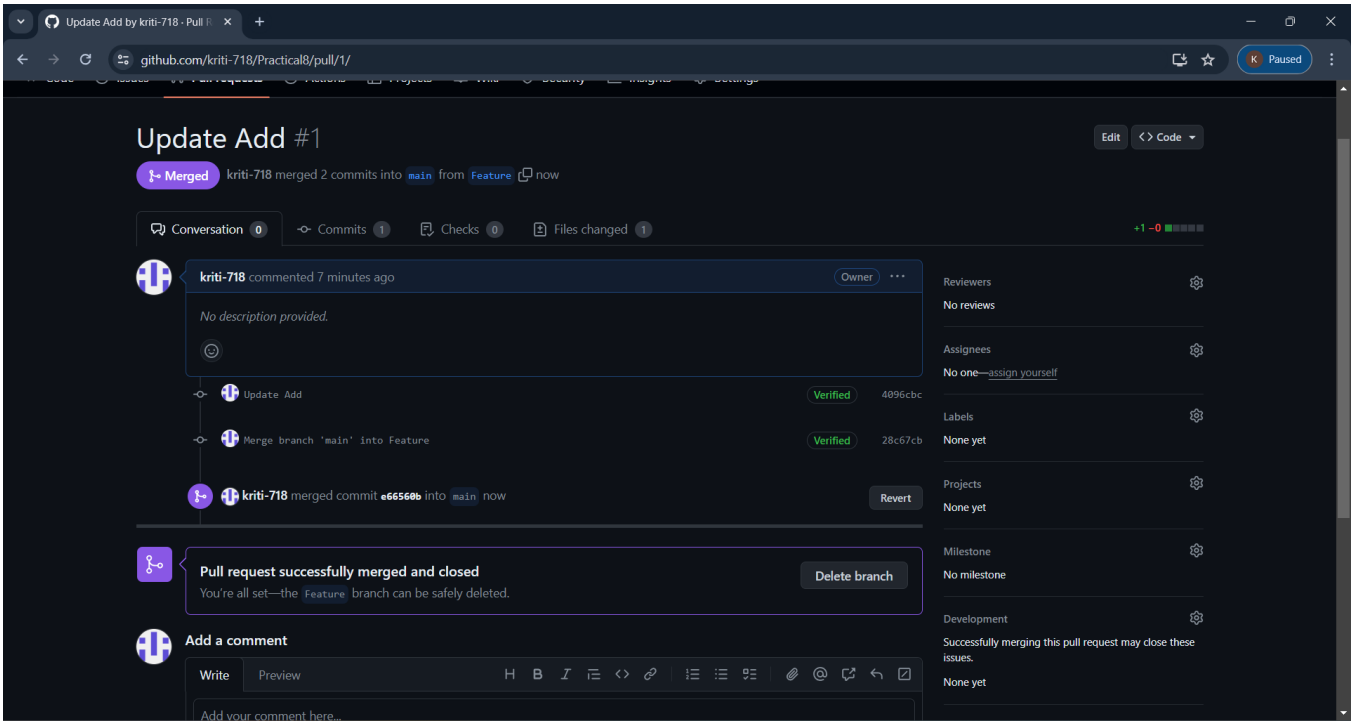
9. Click on commit merge.



10. It can be seen that the conflict has been resolved and branches can be merged easily now.



11. Now, the branches are merged successfully and there is no conflict . We have successfully resolved the own activity conflict error.



PRACTICAL-9

AIM: Git Reset & Git Revert.

Theory:

- The **git reset** command allows you to RESET your current head to a specified state. You can reset the state of specific files as well as an entire branch. This is useful if you haven't pushed your commit up to GitHub or another remote repository yet.
- Both the **git revert** and git reset commands undo previous commits. But if you've already pushed your commit to a remote repository, it is recommended that you do not use git reset since it rewrites the history of commits. This can make working on a repository with other developers and maintaining a consistent history of commits very difficult. Instead, it is better to use **git revert**, which undoes the

changes made by a previous commit by creating an entirely new commit, all without altering the history of commits.

Some Git Reset & Git Revert commands:

1. Firstly, we will run command **"git reset HEAD -- "*filename*"".** This command helps user to un-staged the file which is staged last time

```
PS C:\Users\kriti\OneDrive\Desktop\SCM\Practical9> git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        text.txt

nothing added to commit but untracked files present (use "git add" to track)
PS C:\Users\kriti\OneDrive\Desktop\SCM\Practical9> git add text.txt
PS C:\Users\kriti\OneDrive\Desktop\SCM\Practical9> git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   text.txt

PS C:\Users\kriti\OneDrive\Desktop\SCM\Practical9> git reset HEAD -- "text.txt"
PS C:\Users\kriti\OneDrive\Desktop\SCM\Practical9> git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        text.txt

nothing added to commit but untracked files present (use "git add" to track)
```

2. **"git revert HEAD",** it will washout the last commit and

file get automatically committed and user no need to stage the file again or commit any change.

```
PS C:\Users\kriti\OneDrive\Desktop\SCM\Practical9> git revert HEAD
[main d253dca] Revert "Update scm"
 1 file changed, 0 insertions(+), 0 deletions(-)
PS C:\Users\kriti\OneDrive\Desktop\SCM\Practical9> git log
commit d253dca52457788bebf72abf5f61f85c9103e3b0 (HEAD -> main)
Author: Kriti <krit0781.be23@chitkara.edu.in>
Date:   Sun Mar 31 10:45:08 2024 +0530

    Revert "Update scm"

    This reverts commit 4aafd638895e16388a674c2af95039c71a640eca.

commit 4aafd638895e16388a674c2af95039c71a640eca (origin/main)
Author: Kriti <krit0781.be23@chitkara.edu.in>
Date:   Sun Mar 31 10:38:42 2024 +0530

    Update scm
```

3. **git restore --staged “filename””,** command is used to unstage changes that you have previously added to the staging area. When you make changes to files in your Git repository and want to commit those changes, you first add them to the staging area using git add. However, if you accidentally added a file or changes to a file that you didn't intend to include in the next commit, you can use git restore --staged <filename> to unstage those changes.

```
PS C:\Users\kriti\OneDrive\Desktop\SCM\Practical9> git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   text.txt

PS C:\Users\kriti\OneDrive\Desktop\SCM\Practical9> git restore --staged text.txt
PS C:\Users\kriti\OneDrive\Desktop\SCM\Practical9> git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        text.txt

nothing added to commit but untracked files present (use "git add" to track)
```

4. **“git reset --soft HEAD~ “**, it does not reset the index file or working tree, but resets HEAD to commit. Changes all files to "Changes to be committed". Basically, it uncommit changes, changes are left staged.

```
PS C:\Users\kriti\OneDrive\Desktop\SCM\Practical9> git reset --soft HEAD~1
PS C:\Users\kriti\OneDrive\Desktop\SCM\Practical9> git status
On branch main
Your branch is behind 'origin/main' by 1 commit, and can be fast-forwarded.
  (use "git pull" to update your local branch)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   test.txt
        new file:   text.txt
```

5. **“git reset --mixed HEAD”**, resets the index but not the working tree and reports whathas not been updated. This command (uncommit + un-stage) changes, changes are left in working tree.

```
PS C:\Users\kriti\OneDrive\Desktop\SCM\Practical9> git reset --mixed HEAD~2
Unstaged changes after reset:
M       scm
PS C:\Users\kriti\OneDrive\Desktop\SCM\Practical9> git status
On branch main
Your branch is behind 'origin/main' by 2 commits, and can be fast-forwarded.
  (use "git pull" to update your local branch)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   scm

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        test.txt
        text.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

6. **“git reset –hard HEAD”**, resets the index and working tree. Any changes to tracked files in the working tree since commit are discarded. uncommit + unstage + delete changes, nothing left.

```
PS C:\Users\kriti\OneDrive\Desktop\SCM\Practical9> git log --pretty=oneline
182b0daf3cb1554121a3c3db83cb6171f62ac3fb (HEAD -> main) Updated
4aafd638895e16388a674c2af95039c71a640eca Update scm
b53c52716afb1312deb625326405bae50eafed84 Create scm
987d96b211bafa485aa2cf99ea397fe927ea6a12 Create README.md
PS C:\Users\kriti\OneDrive\Desktop\SCM\Practical9> git reset --hard HEAD~1
HEAD is now at 4aafd63 Update scm
PS C:\Users\kriti\OneDrive\Desktop\SCM\Practical9> git log --pretty=oneline
4aafd638895e16388a674c2af95039c71a640eca (HEAD -> main) Update scm
b53c52716afb1312deb625326405bae50eafed84 Create scm
987d96b211bafa485aa2cf99ea397fe927ea6a12 Create README.md
```

7. **“git mv <previous file name> <new file name>”,** it is used to change a flle name,thefile with changed name goes to staging area then we need to commit to apply changes.

```
PS C:\Users\kriti\OneDrive\Desktop\SCM\Practical9> git mv "kriti.txt" "Game1.txt"
PS C:\Users\kriti\OneDrive\Desktop\SCM\Practical9> git status
On branch main
Your branch and 'origin/main' have diverged,
and have 1 and 2 different commits each, respectively.
  (use "git pull" if you want to integrate the remote branch with yours)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        renamed:    kriti.txt -> Game1.txt
```

8. **“git rm <filename>”,it is used to delete a file,** the deleted file is already in staging area then we need to commit the changes to remove file completely.

```
PS C:\Users\kriti\OneDrive\Desktop\SCM\Practical9> git rm "ls.txt"
rm 'ls.txt'
PS C:\Users\kriti\OneDrive\Desktop\SCM\Practical9> git status
On branch main
Your branch and 'origin/main' have diverged,
and have 2 and 2 different commits each, respectively.
  (use "git pull" if you want to integrate the remote branch with yours)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    ls.txt
```