Subject Name: **Source Code Management**

Subject Code: **22CS003** Session: **2023-24** epartment: **DCSE**



**Submitted By:**

Harsh Singla 2310990765-First Year G10-A

Submitted To:

Dr. Deepak Thakur Assistant Professor | Head Academic Operations Chitkara University, Punj

**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:**

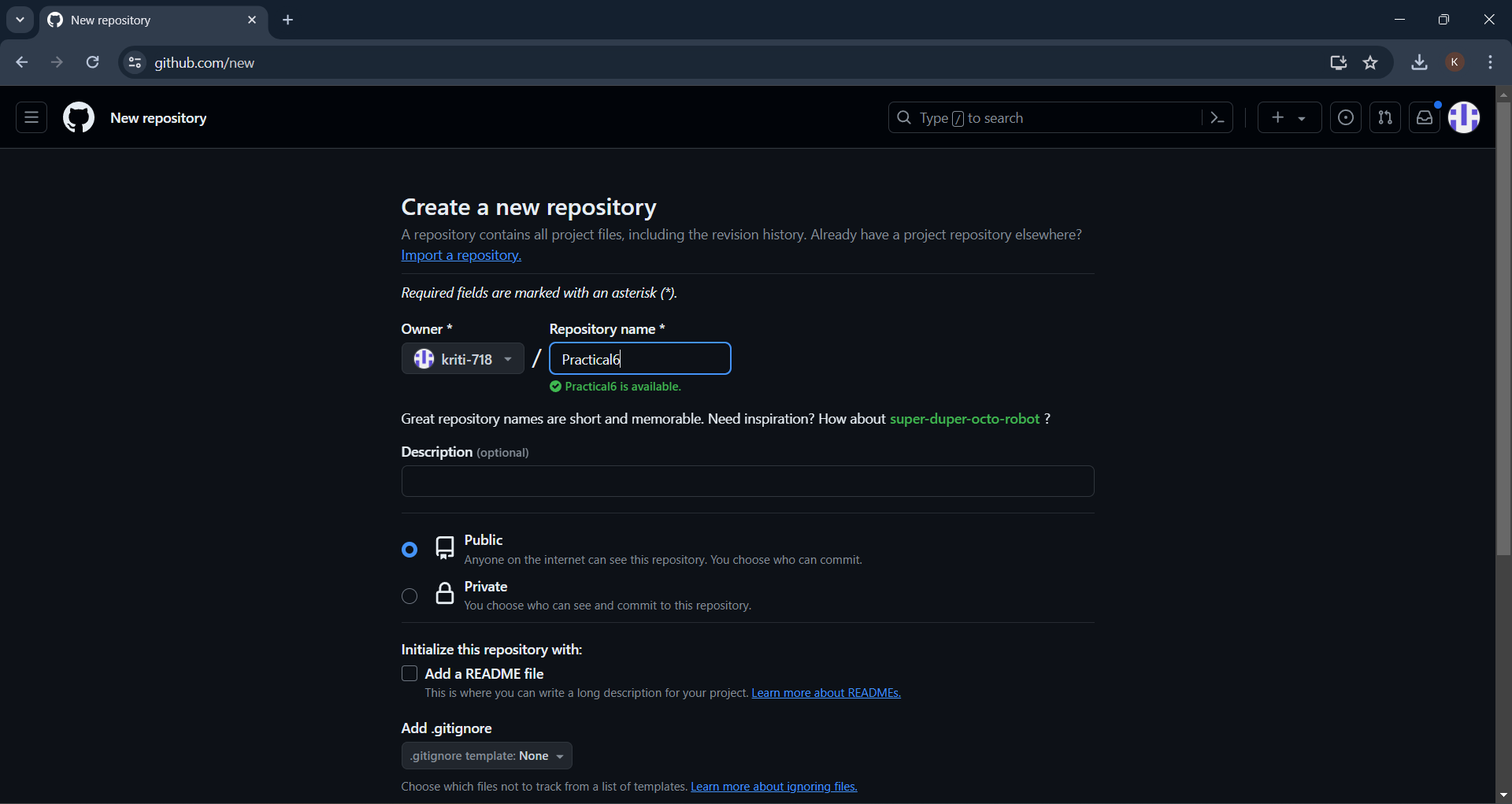
In your public repository in GitHub, no one has the permission to push code into your repository. They can only read the repository. In order to allow other individuals to make changes to your repository, you need to invite them to collaborate to the project.

## Understanding Collaboration Roles:

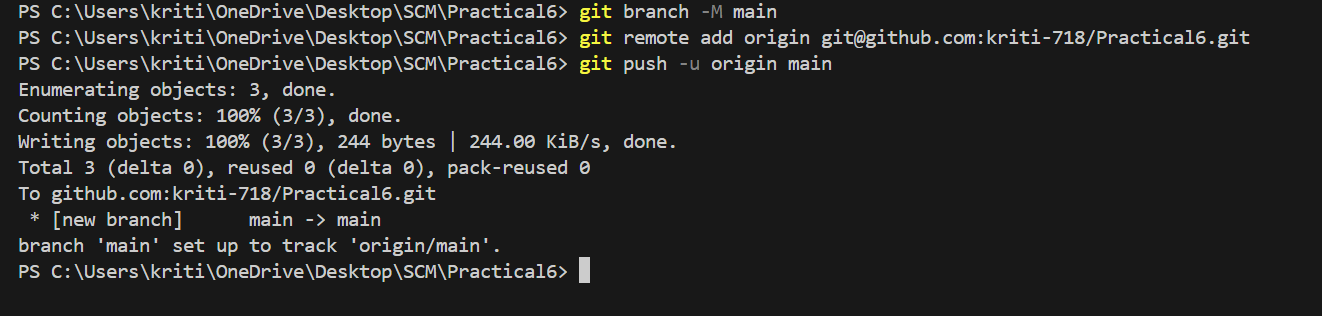
* **Read:** Permission to view the repository's contents, including files, discussions, and commits.
* **Write:** Allows collaborators to make changes to existing files and create new ones. This encompasses pushing commits, creating branches, and raising pull requests.
* **Admin:** Grants extensive control, 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:

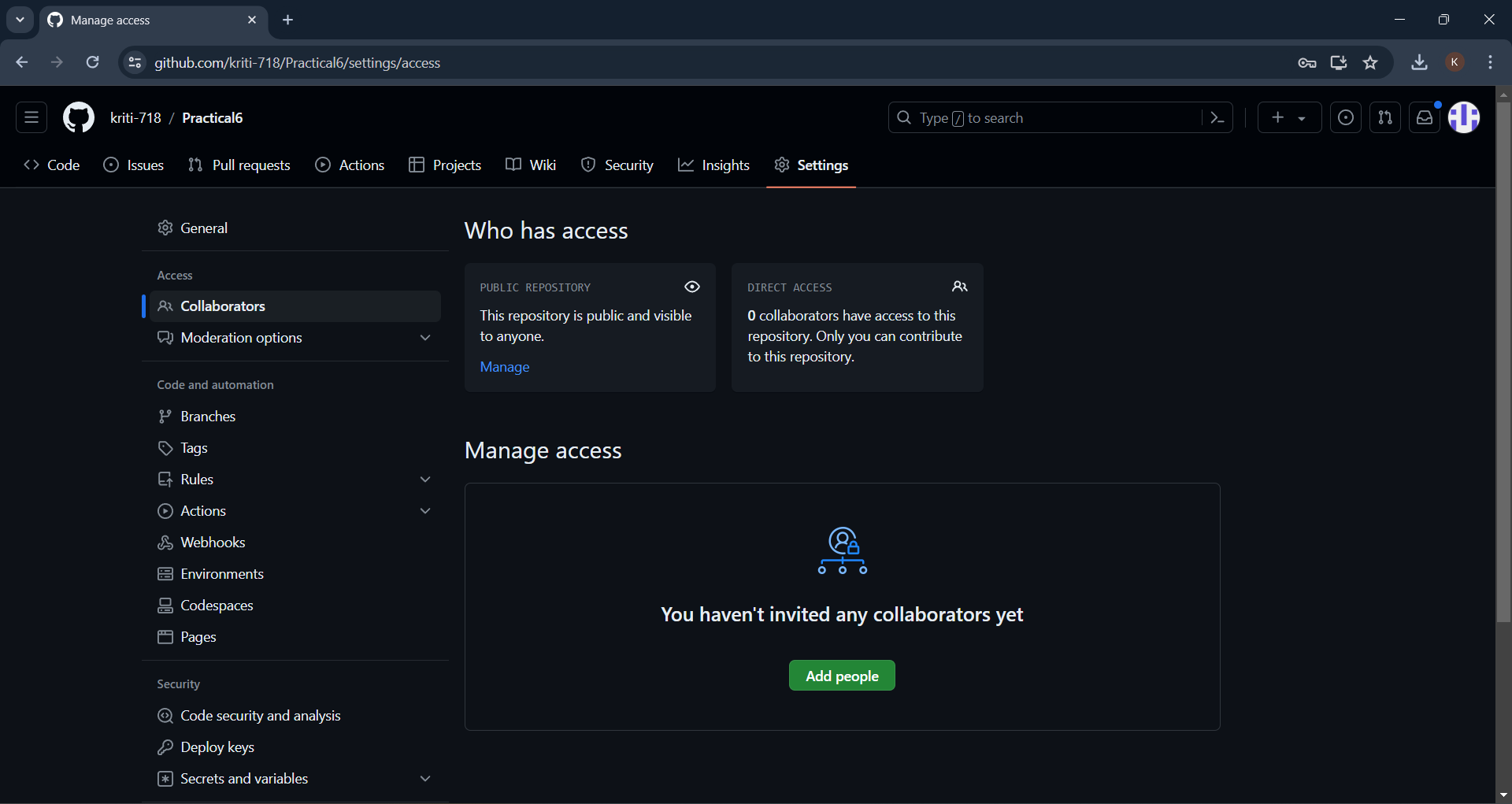
1. Firstly, create a repository on GitHub.



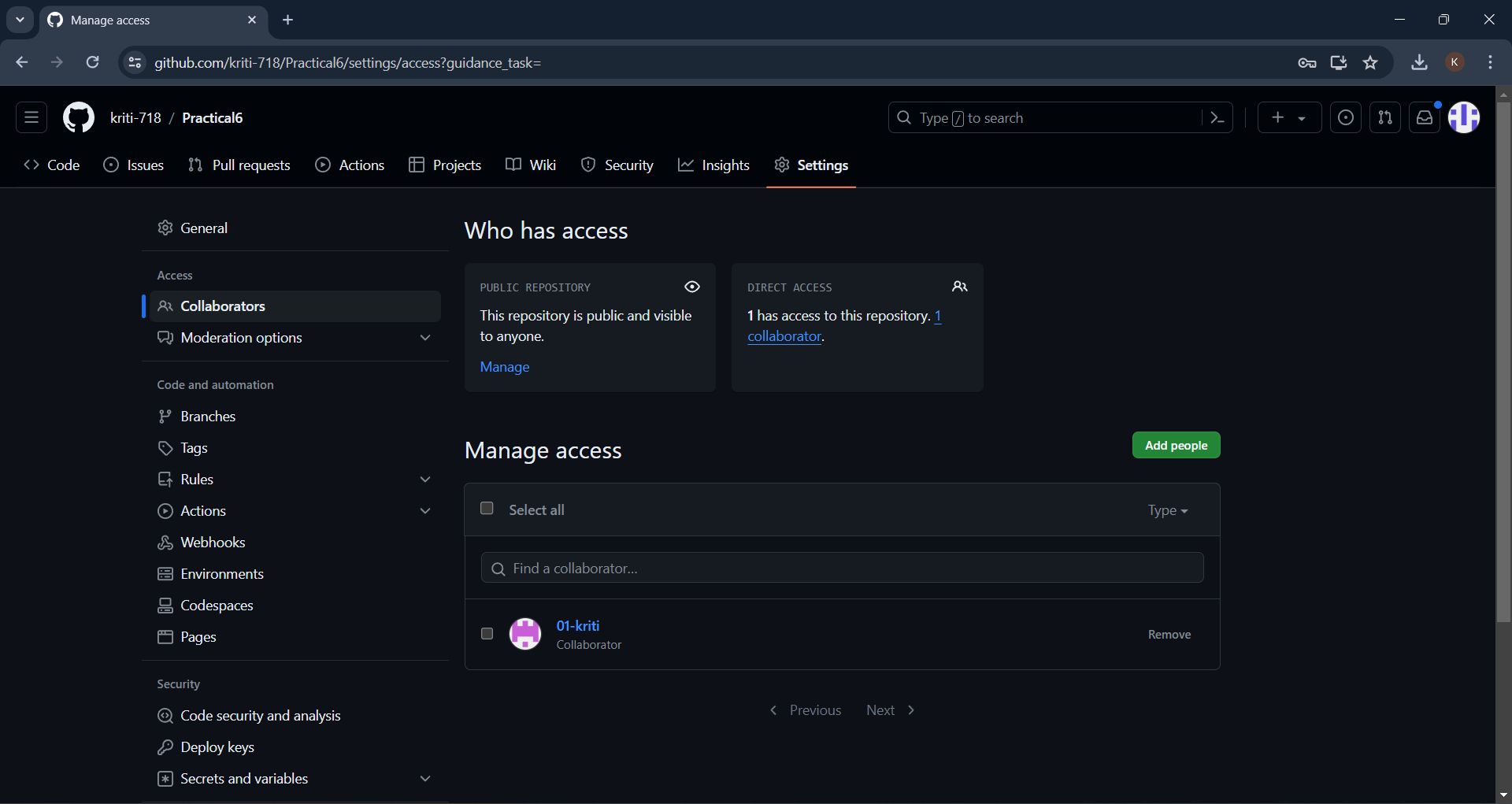
1. Create a new repo or push an existing one using these commands.



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

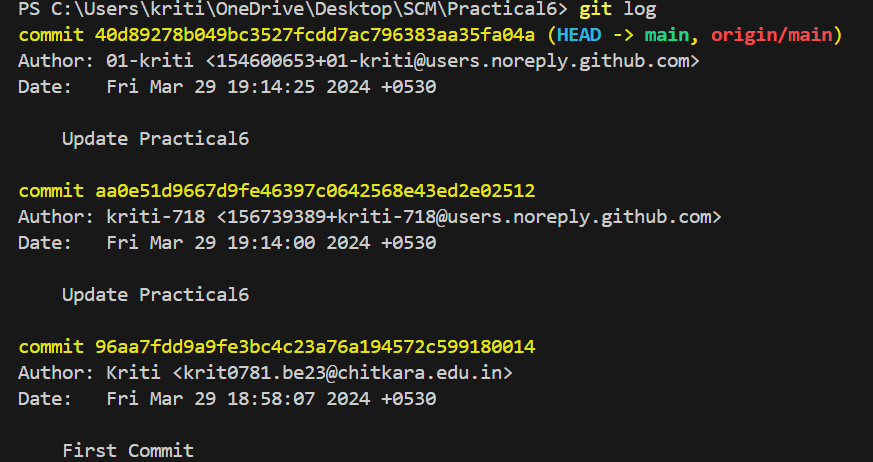


1. GitHub provides us feature to manage our collaborators on our GitHub repository.



1. Now, see how beautifully Git tell us about changes and who have done these changes

by running the command “git log”.



# PRACTICAL-7

**AIM: Fork and commit**

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

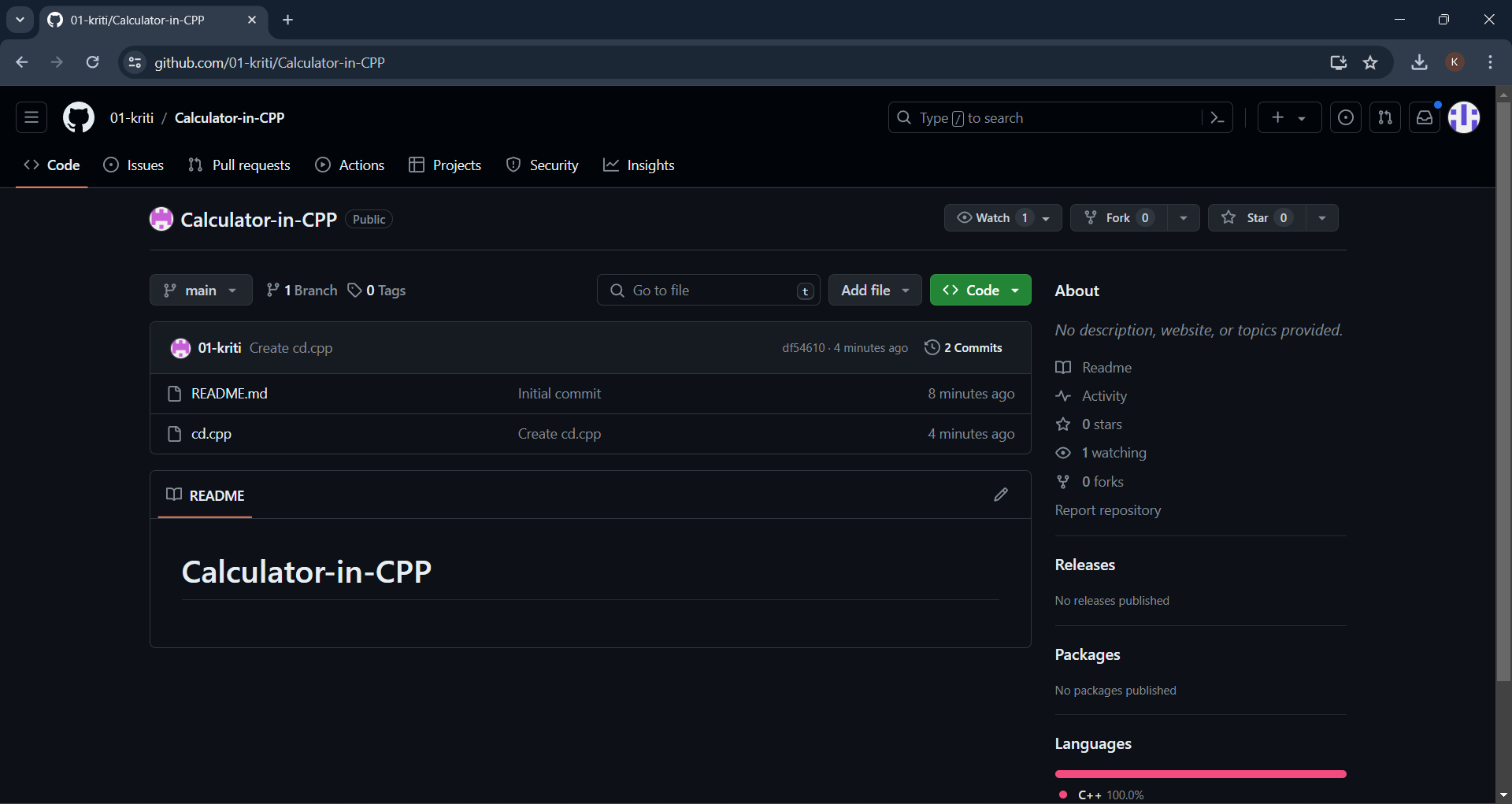
**Theory:**

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.

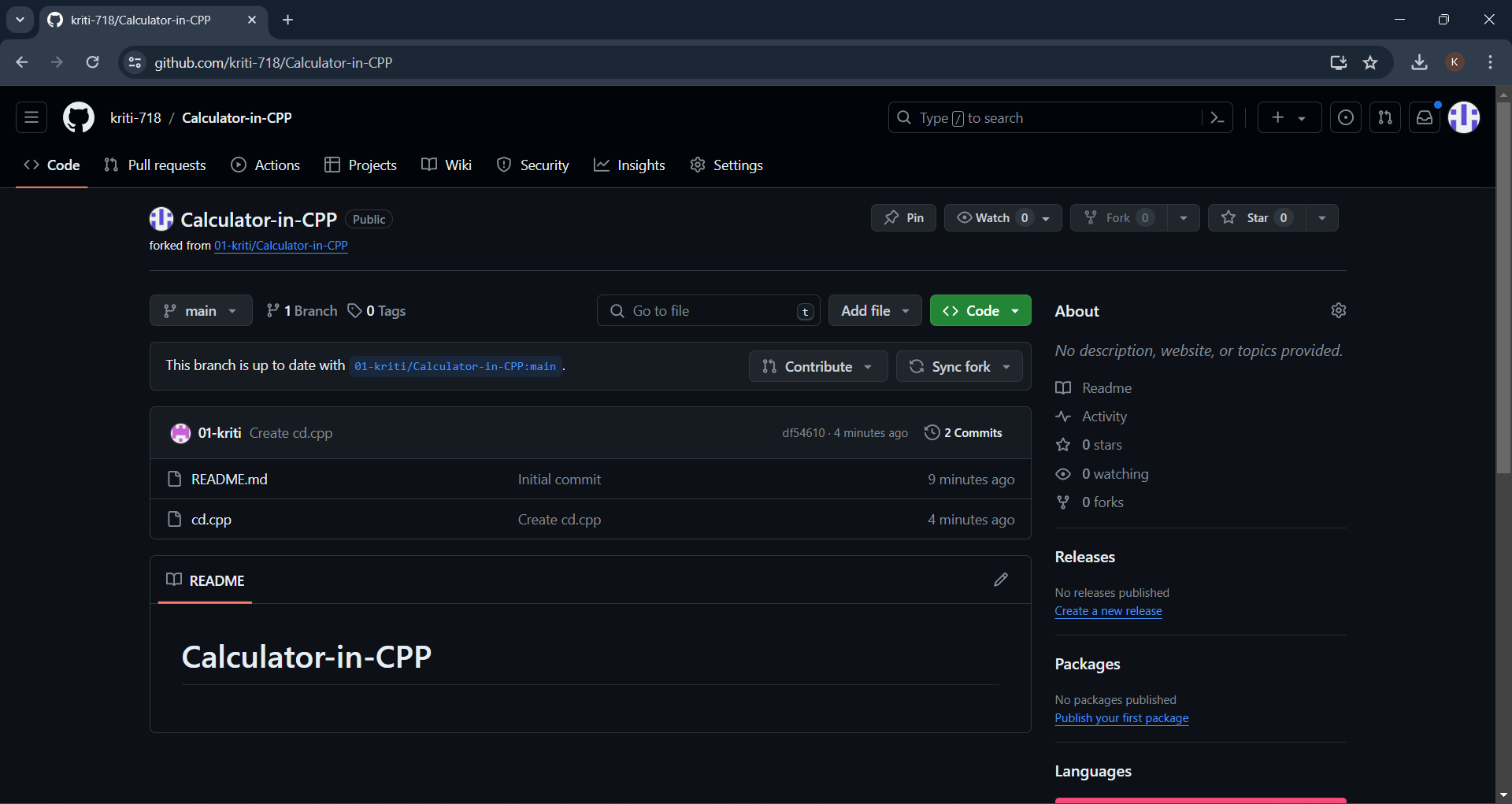
## Steps of creating “Fork” of repository in GitHub:

1. Firstly, open the repository which you wanted to create “Fork” by clicking the fork

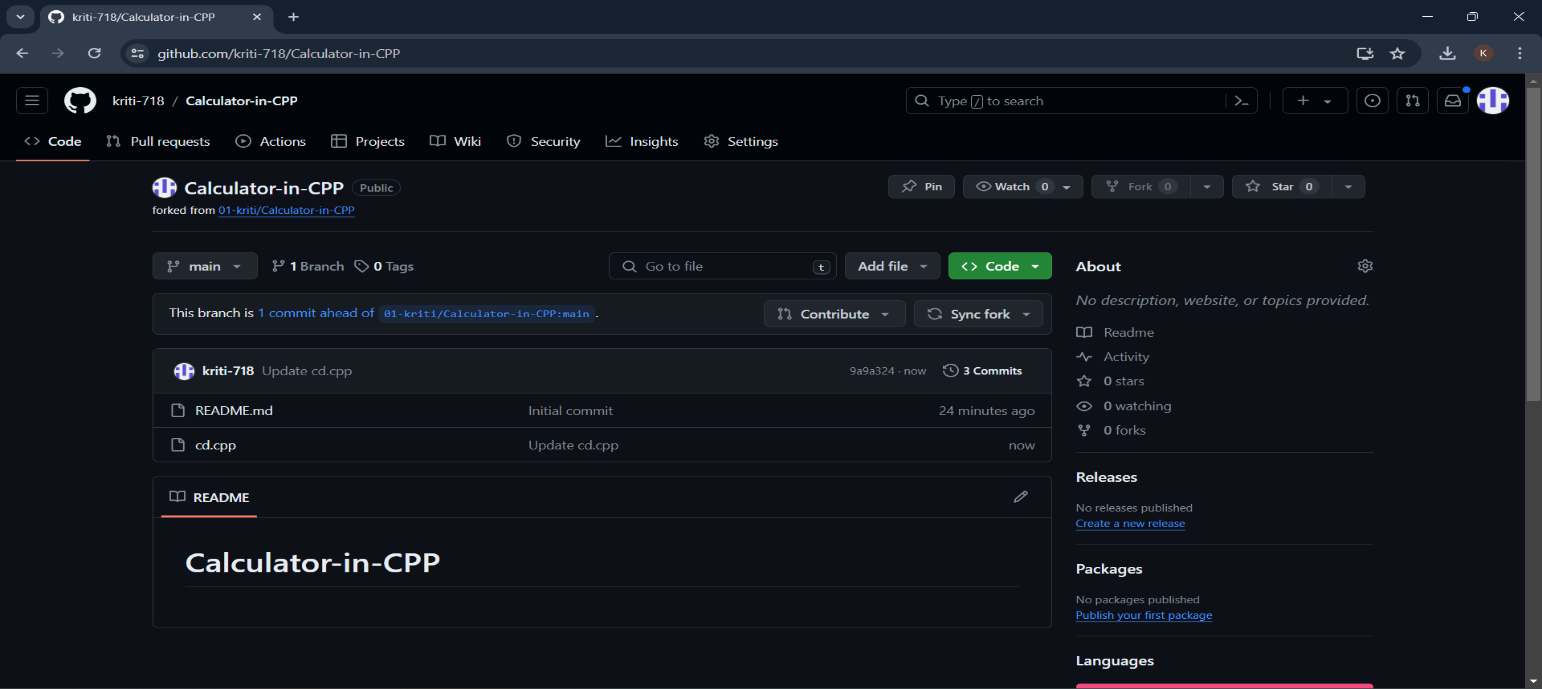
option given on top-right corner as shown in picture given below.



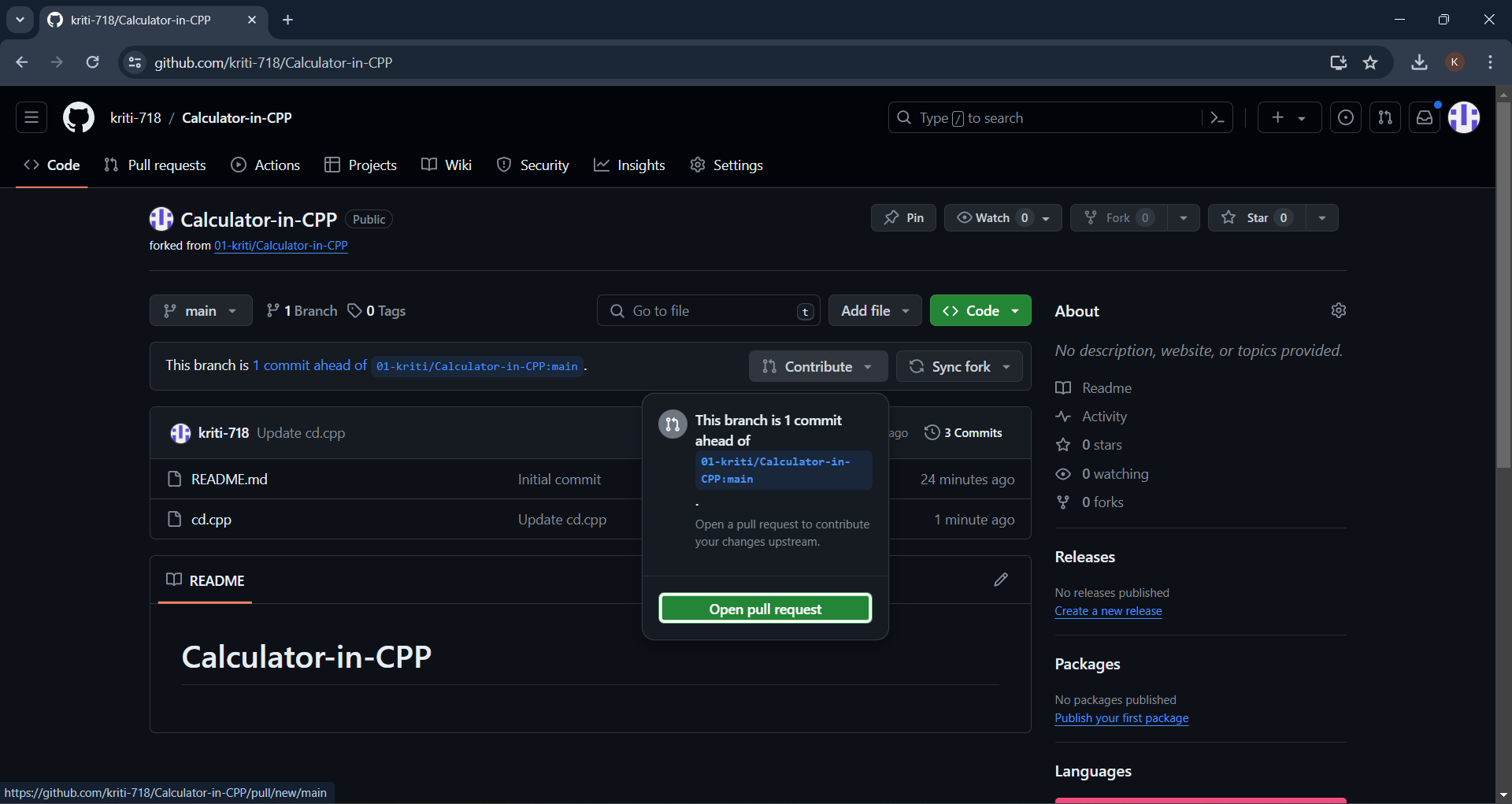
1. After creating “Fork”, GitHub platform is designed in this way it will automatically create a link with original repository and tells user the fork repo is ‘up to date’ or it is ‘ahead’ or it is ‘behind’ of original repository.



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



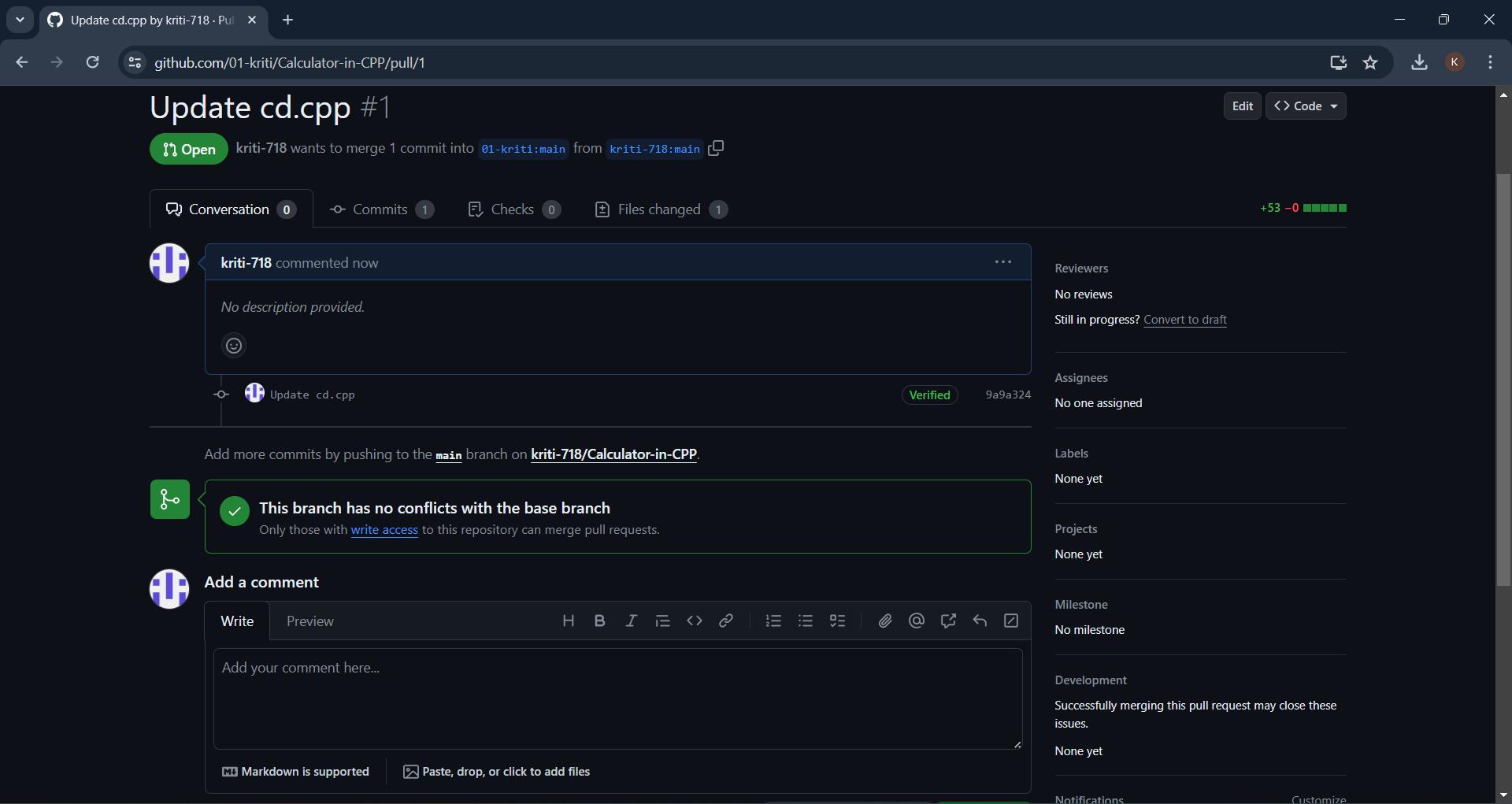
1. To create open pull request click on the contribute option as shown in picture given below.



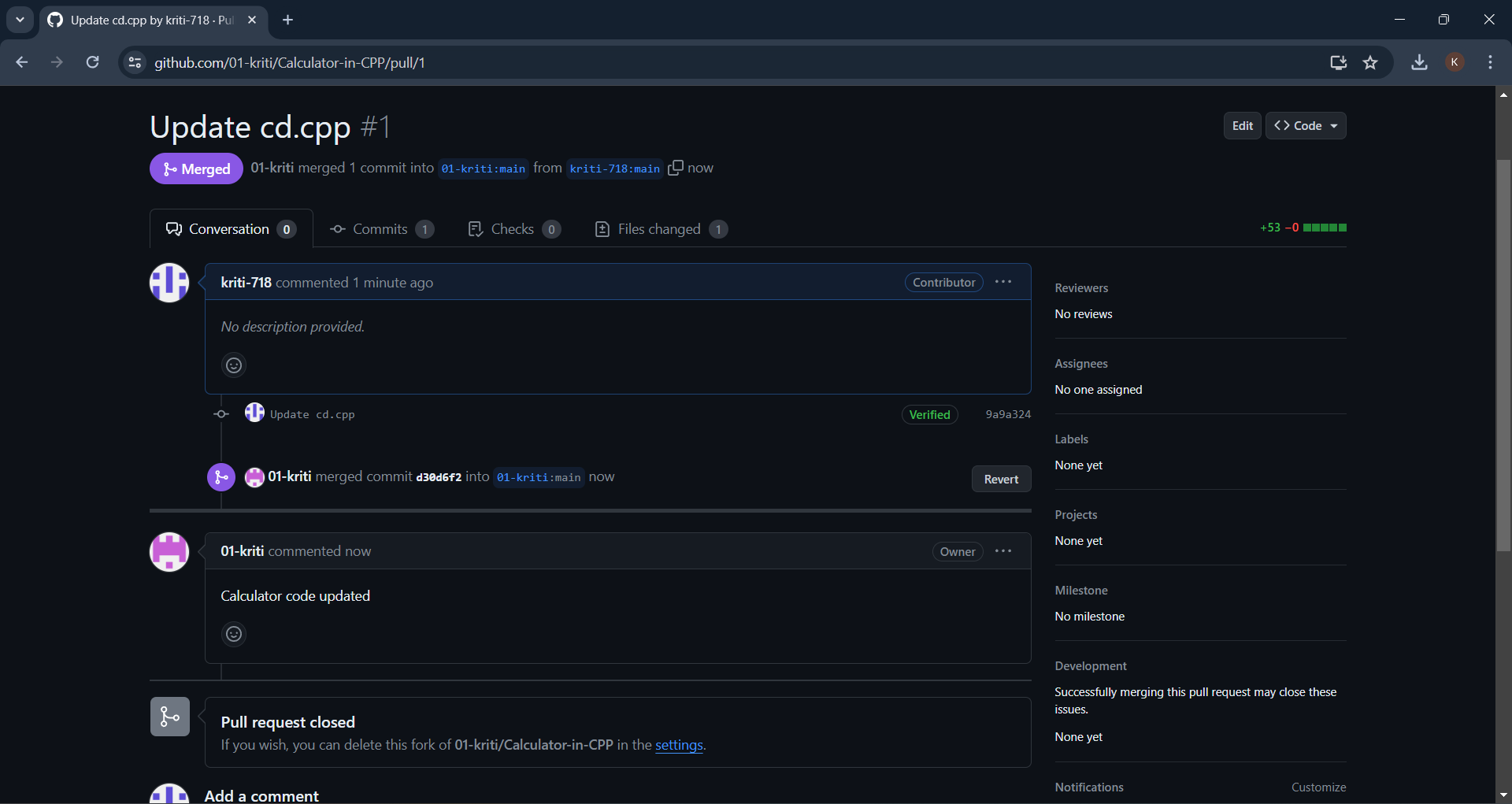
1. Before creating open pull request GitHub compare all the changes, so that user can confirm their change. It will highlights the changes what user have make changes on the repo after confirming user can proceed and create open pull request.



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



1. 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.

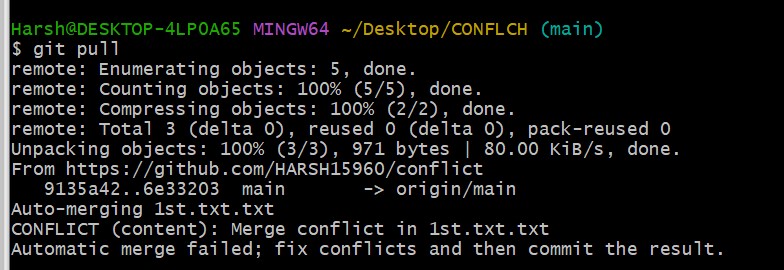


## Steps for resolving conflicts created due to own activity in repository:

1. Firstly, created a new repository and add some file in the repo and make some changes in it and push it on the cloud with the help of git command “git push”.(i.e.; as we have done these all work in above experiments or if you haven’t done this before, please complete above experiments then proceed to experiment 08)
2. Now, unfortunately user have make the changes on his/her local machine and also on the cloud in the same repo and same file, when user trying to push local change to cloud the git will rejected it (i.e.; as shown in picture given below).



* + 1. So, to resolve this error user have to run a command “git pull” (i.e.; as git also giving hint to run git pull command), when the user run a git pull command conflict error will come. The error message prints the information about where the conflict is present.

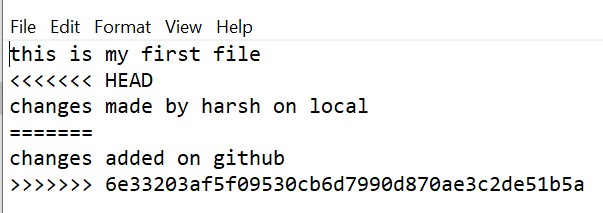


* + 1. Now, Check the file in which git shows error message and look at the contents where the merge conflict happened. Git automatically adds three indicators alongside the conflicting lines of code:

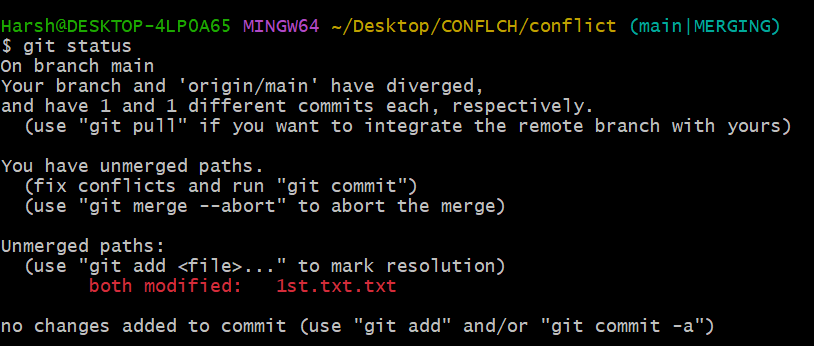
**<<<<<<<** (seven "less than" characters) followed by **HEAD**, which is an alias for the current branch. The symbols indicate the beginning of the edits within this section.

**=======** (seven "equals sign" characters), which show the end of the revisions within the current branch and the beginning of the edits within a new one.

**>>>>>>>** (seven "greater than" characters) followed by the branch where the attempted merge happened. He added symbols indicate the ending of the edits within the conflicting branch.



1. Now, what user can edit this text easily and keep the text what he/she wants in the file, user can edit the file by simply open the file or user can directly use editor tools on git such as vim/vi, nano and etc. After that edit the change and run command “git status” it will show *“Your branch and 'origin/master' have diverged, and have 2 and 1 different commits each, respectively.”* Commit this change and the user can easily push this change to cloud.



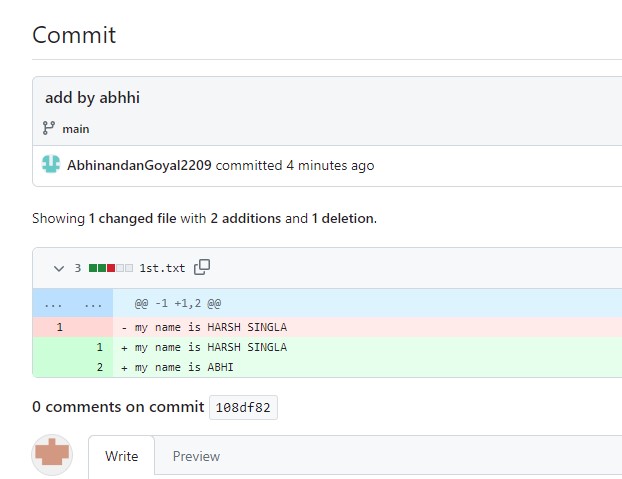
1. Now, run git push and you can see no conflict error will come. We have successfully resolved the own activity conflict error.

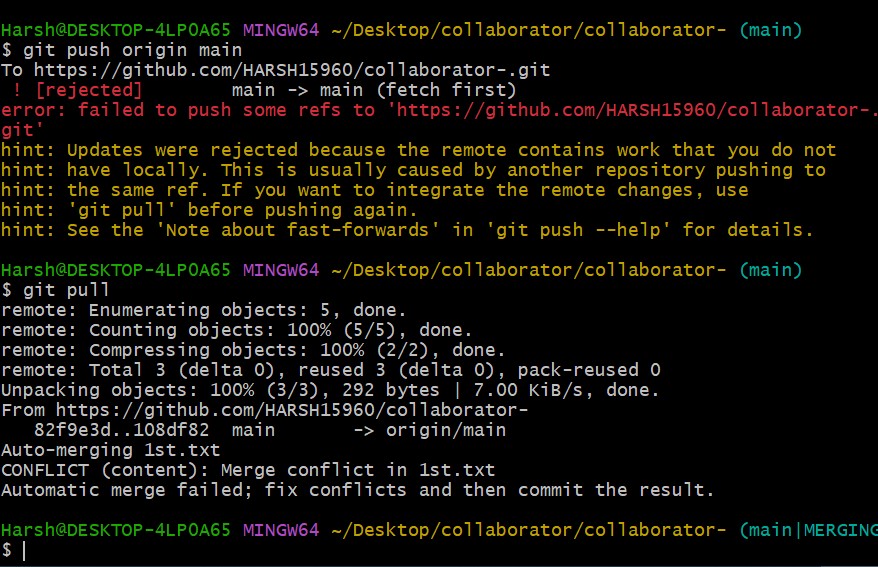


## Steps to resolve conflicts created due to collaborators activity:

* + 1. This is very similar to own activity conflict the main difference in this is conflict occur due to the collaborator activity, if user make some changes in local machine and collaborator also make changes on cloud in the same repository and same file then conflict error will occur.

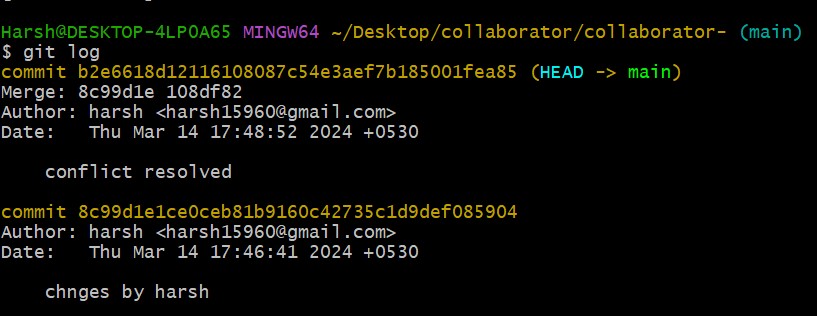
In the given picture you can easily observe that user make changes on his/her local machine and collaborator also make changes in the same file. Now if user try to push his/her work to cloud conflict will occur.





* + 1. To resolve this conflict follow the same steps which are done to resolve own activity conflict, use editor tools or just simply edits directly form file.

Here, we successfully resolved the collaborator activity conflict.



# PRACTICAL-9

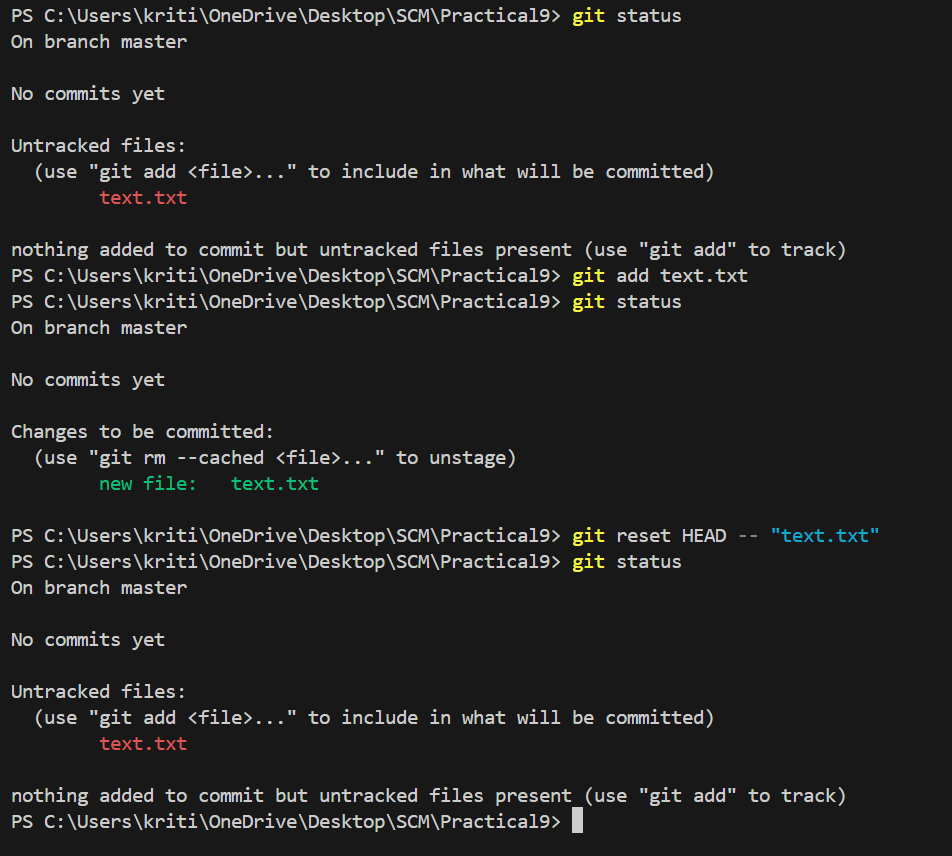
**AIM: Git Reset & Git Revert.**

## Theory:

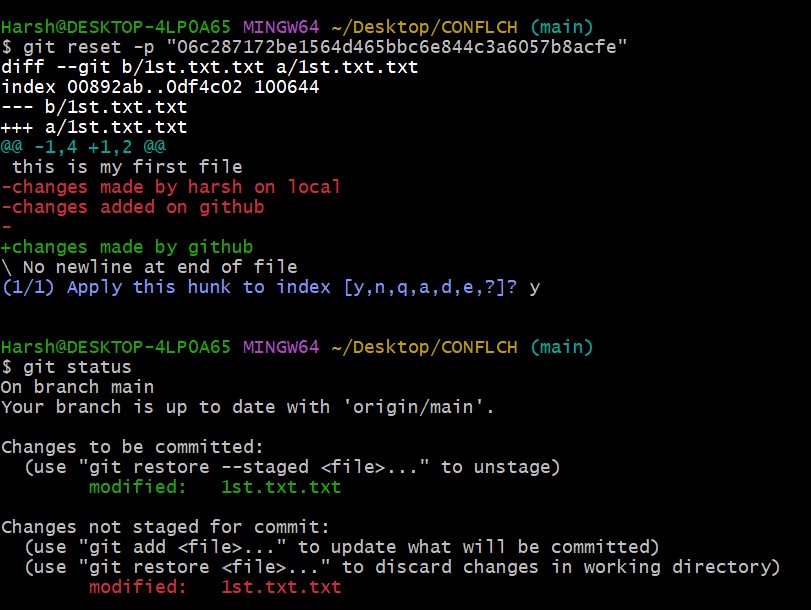
* The **git reset** command allows you to RESET your current head to a specified state. You can reset the state of specific files as well as an entire branch. This is useful if you haven't pushed your commit up to GitHub or another remote repository yet.
* Both the **git revert** and git reset commands 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:

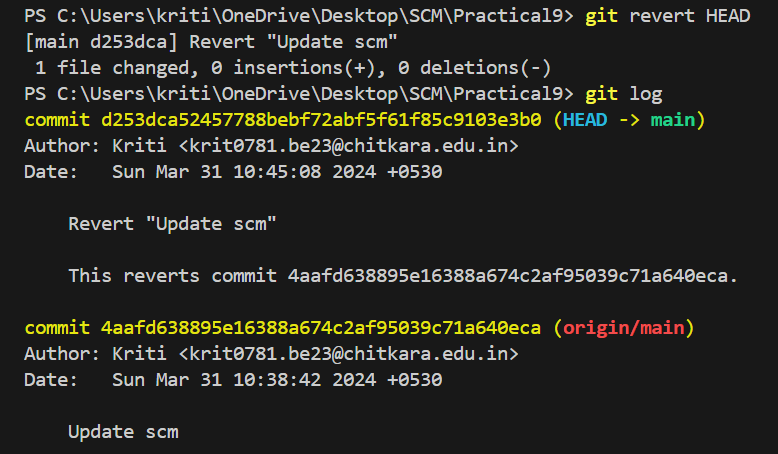
1. Firstly, we will run command **“git reset HEAD -- “*filename*””.** This command helps user to un-staged the file which is staged last time.



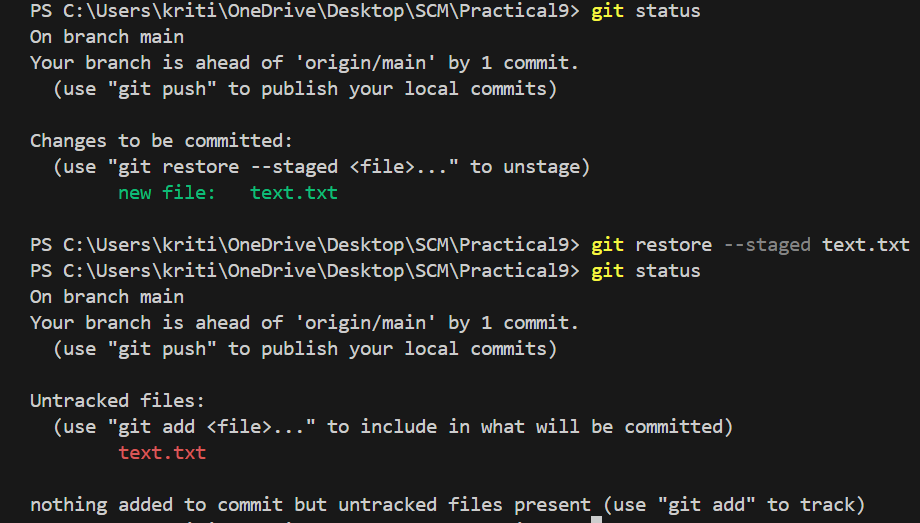
1. **“git reset -p “*checksum*””**, this command keeps everything till checksum provided by user & un-stage all the commits after that checksum that user provided to git.



1. **“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.



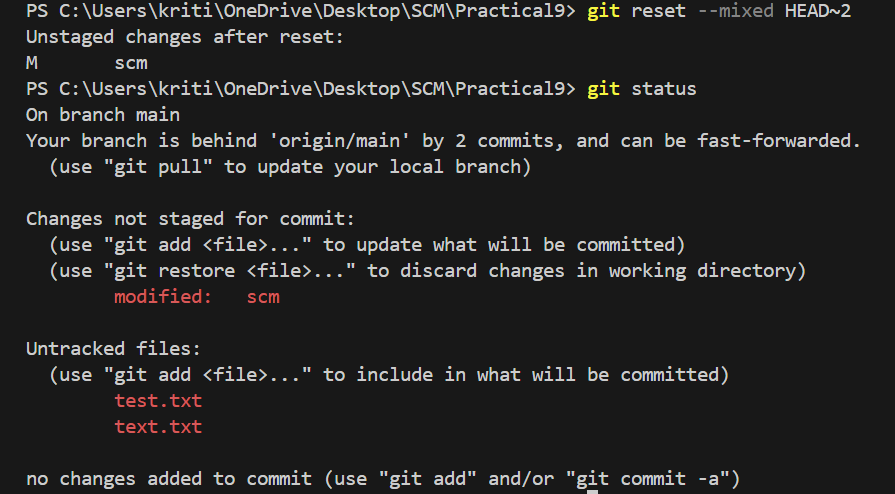
1. **“git restore --staged “filename””,** command is used to unstage changes that you have previously added to the staging area. When you make changes to files in your Git repository and want to commit those changes, you first add them to the staging area using git add. However, if you accidentally added 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.



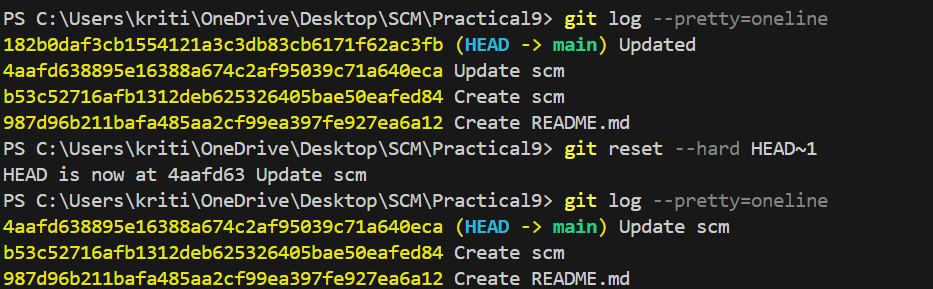
1. **“git reset --soft HEAD~ “**, it does not reset the index file or working tree, but resets HEAD to commit. Changes all files to "Changes to be committed". Basically, it uncommit changes, changes are left staged.



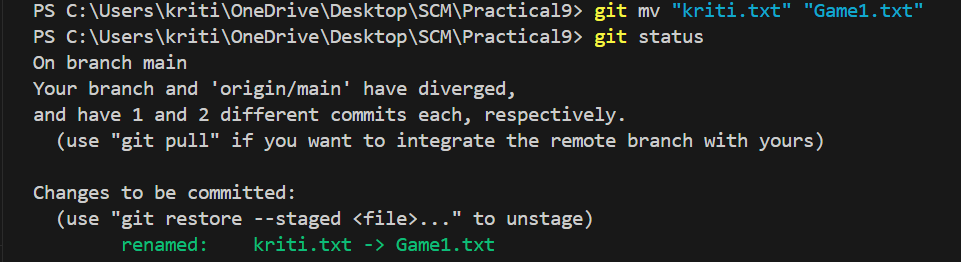
1. **“git reset --mixed HEAD”**, resets the index but not the working tree and reports what has not been updated. This command (uncommit + un-stage) changes, changes are left in working tree.



1. **“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.



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



1. **“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.

