

**Submitted**

**By:**

Aayush Bansal

2110990026

G01

**Submitted**

**To:**

Dr. Monit Kapoor

Subject

Name:

**Source**

**Code**

**Management**

Subject

Code:

**CS181**

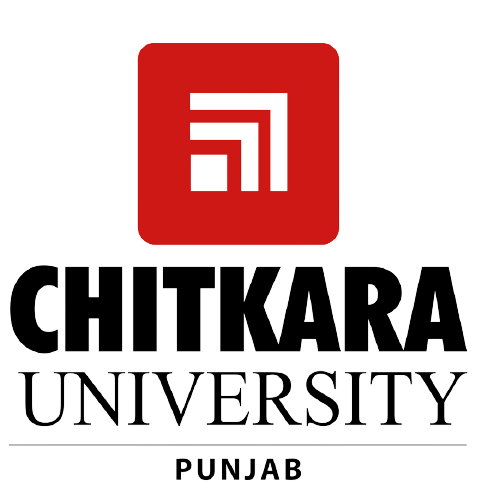
Cluster:

**Beta**

Department:

**DCSE**

**# List of Programs #**



|  |  |  |
| --- | --- | --- |
| **S. No** | **Program Title** | **Page No.** |
| 1. | Add collaborators on Github Repo | 1 |
| 2. | Fork and Commit | 5 |
| 3. | Merge and Resolve Conflicts | 9 |
| 4. | Reset and Revert | 12 |



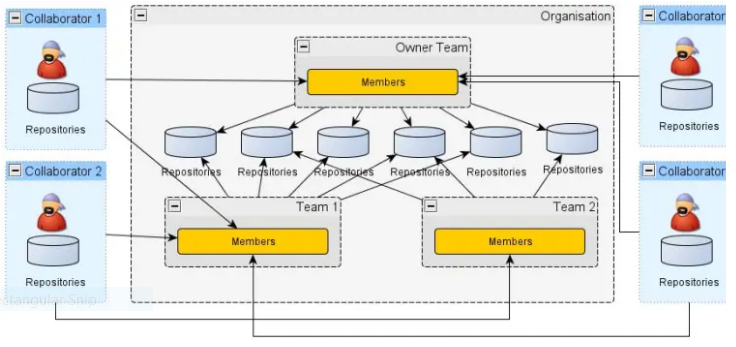
### Experiment No. 06

# **Aim:** Add collaborators on GitHub Repo

Create a New Repository.

A repository is like a folder for your project. You can have any number of public and private repositories in your user account. Repositories can contain folders and files, images, videos, spreadsheets, and data sets, as well as the revision history for all files in the repository. For more information, see "About repositories."

When you create a new repository, you should initialize the repository with a README file to let people know about your project. For more information, see "Creating a new repository."



In GitHub, We can invite other GitHub users to become collaborators to our private repositories(which expires after 7 days if not accepted, restoring any unclaimed licenses). Being a collaborator, of a personal repository you can pull (read) the contents of the repository and push (write) changes to the repository. You can add unlimited collaborators on public and private repositories(with some per day limit restrictions). But, in a private repository, the owner of the repo can only grant write-access to the collaborators, and they can't have the read-only access.

GitHub also restricts the number of collaborators we can invite within a period of 24 hours. If we exceed the limit, then either we have to wait for 24-hours or we can also create an organization to collaborate with more people.

**Actions that can be Performed By Collaborators**

Collaborators can perform a number of actions into someone else's personal repositories, they have gained access to Some of them are,

* Create, merge, and close pull requests in the repository
* Publish, view, install the packages
* Fork the repositories
* Make the changes on the repositories as suggested by the Pull requests.
* Mark issues or pull requests as duplicate
* Create, edit, and delete any comments on commits, pull requests, and issues in the repository
* Removing themselves as collaborators on the repositories.
* Manage releases in the repositories.

Now, let's see how can we invite collaborators to our repositories.

### Inviting Collaborators to your personal repositories

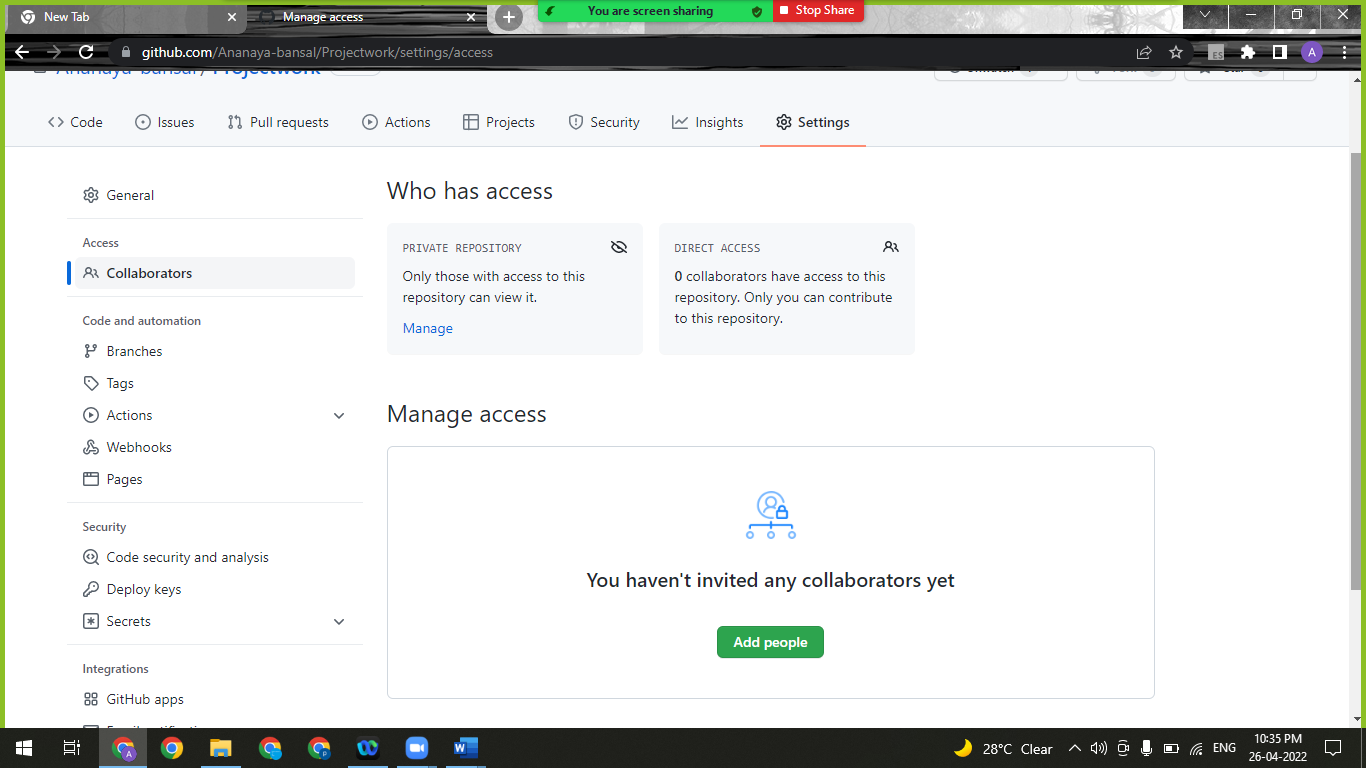
Follow the steps below to invite collaborators to your own repository(public or private).

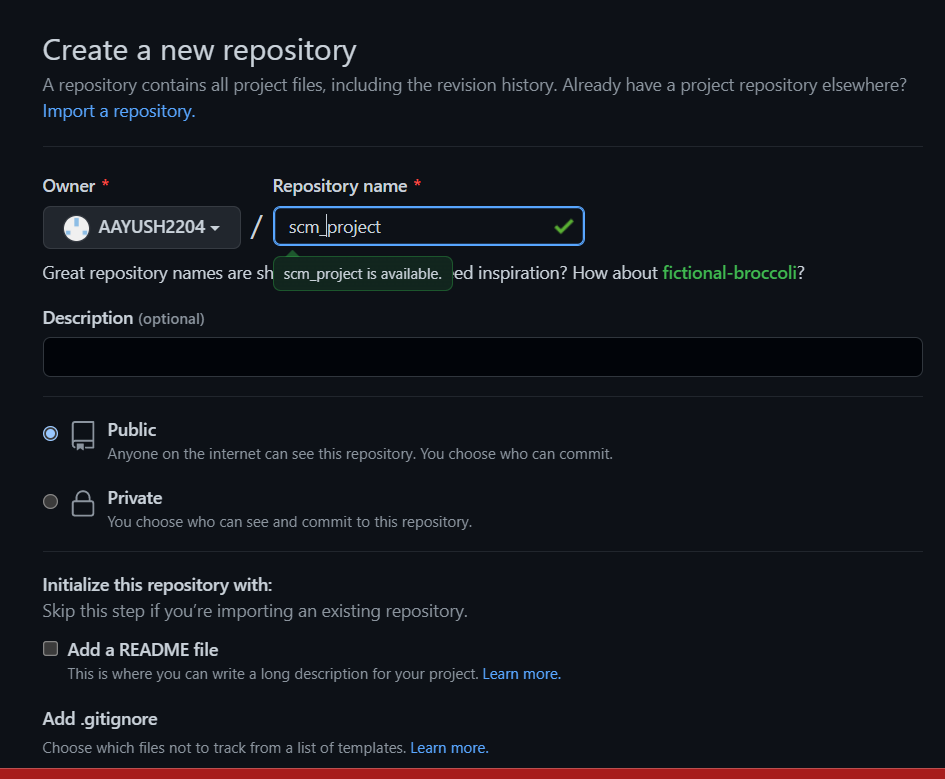
**Step 1:** Get the usernames of the GitHub users you will be adding as collaborators. In case, they are not on GitHub, ask them to sign in to GitHub.

**Step 2:**Go to your repository( intended to add collaborators)

**Step 3:**Click into the Settings.

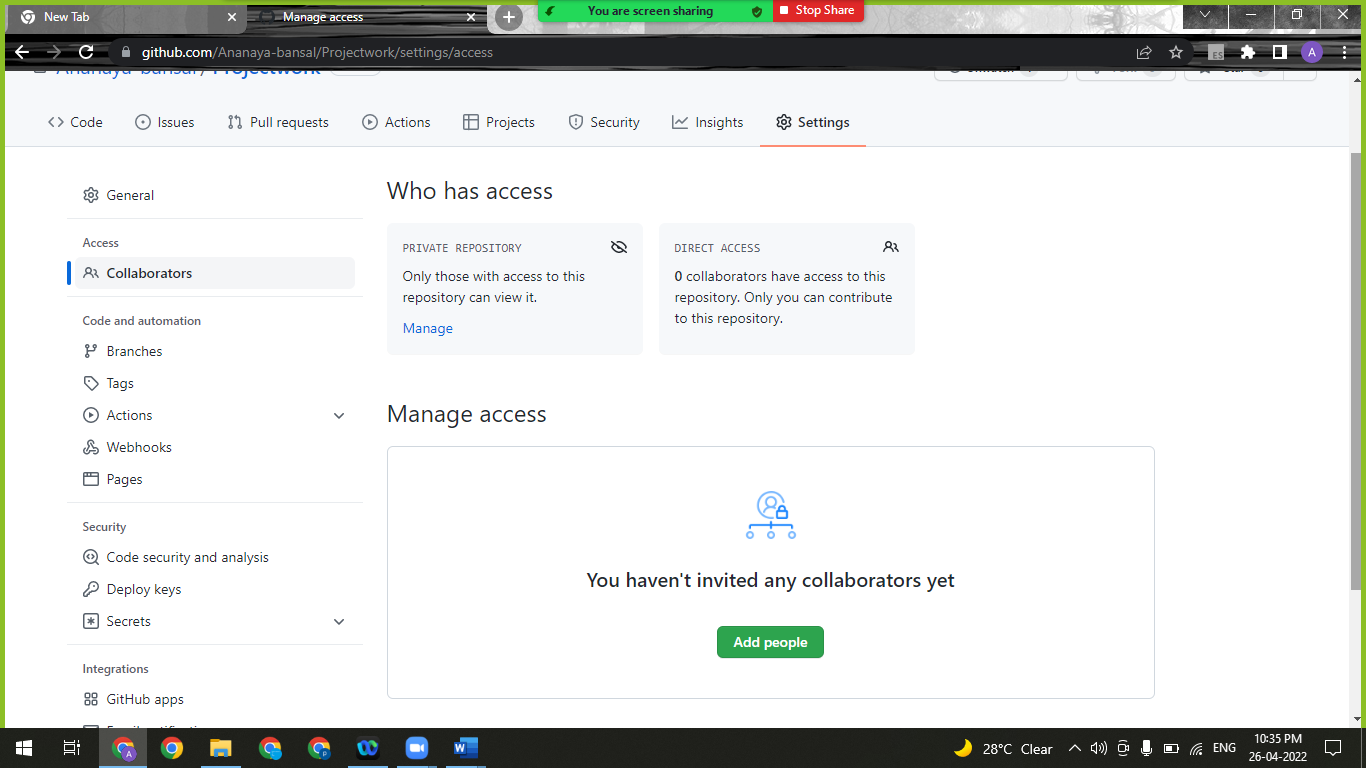
**Step 4:**A settings page will appear. Here, into the left-sidebar click into the Collaborators.



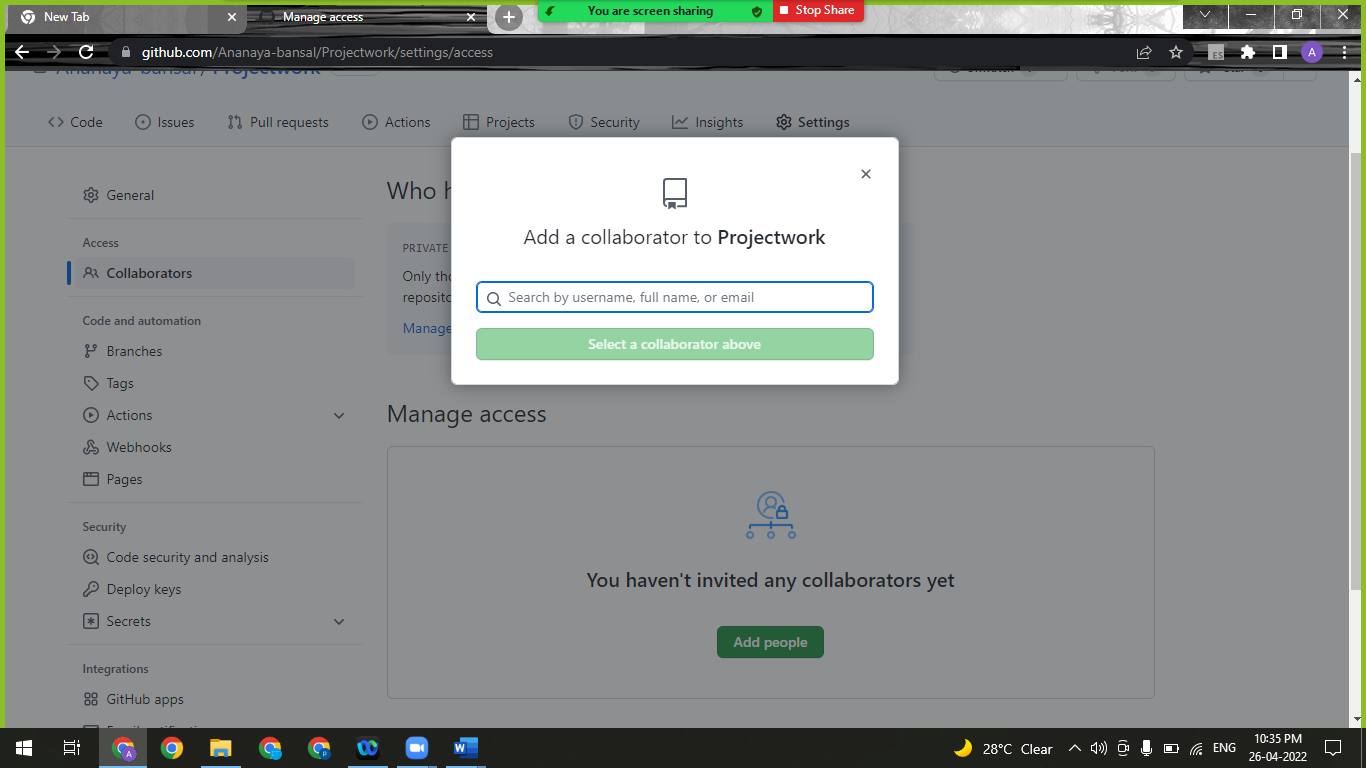


**Step 5:**Then a confirm password page may appear, enter your password for the confirmation.

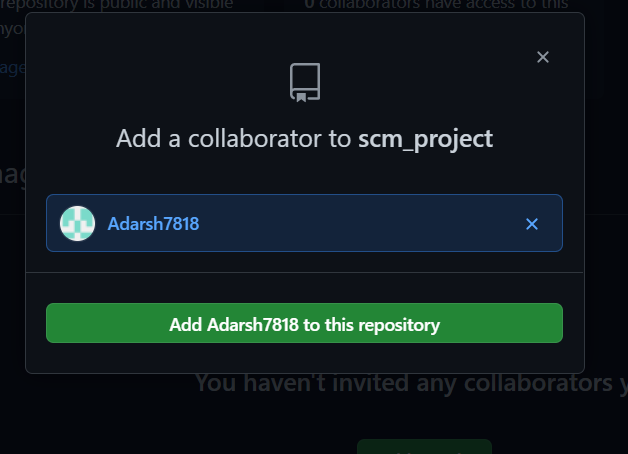
**Step 6:** Next, click into Add People.



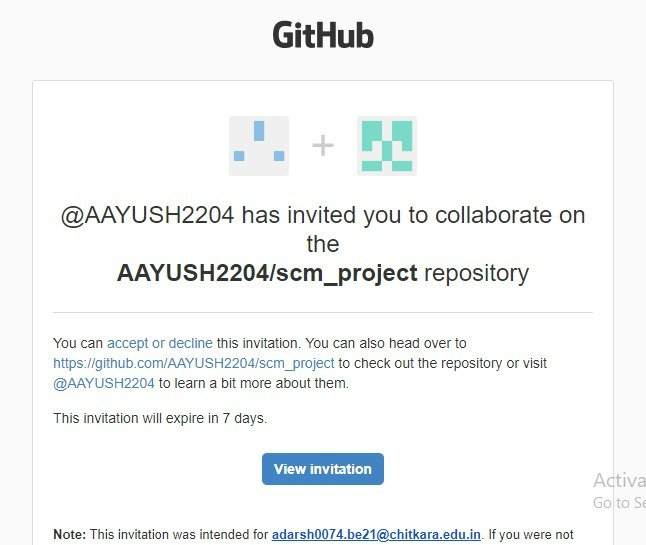
**Step 7:**Then a search field will appear, where you can enter the username of the ones you want to add as collaborator.



**Step 8:**After selecting the people, add them as collaborator.



**Step 9:** After sending the request for collaboration to that person whom we wanted as our collaborator for our project .



**Step 10:** After Clicking the View invitation, He/she will be redirected to the GitHub Page for accepting the invitation sent by the team leader.



### Experiment No. 07

## **Aim:** Fork and Commit

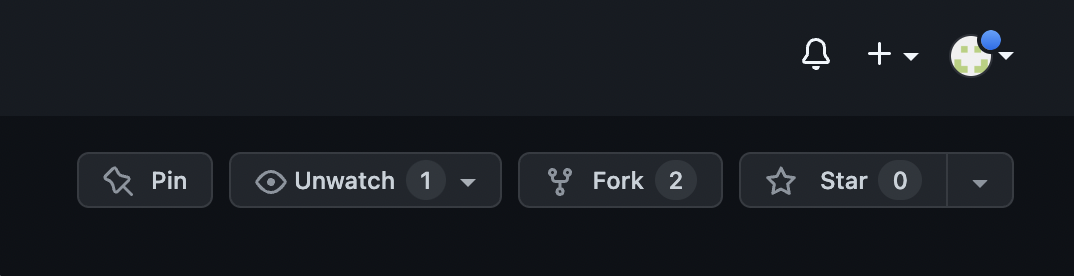
**What is Forking a Repository mean and Why it is used?**

**Forking a repository** means creating a copy of the repo. When you fork a repo, you create your own copy of the repository on your GitHub account. This is done for the following reasons:

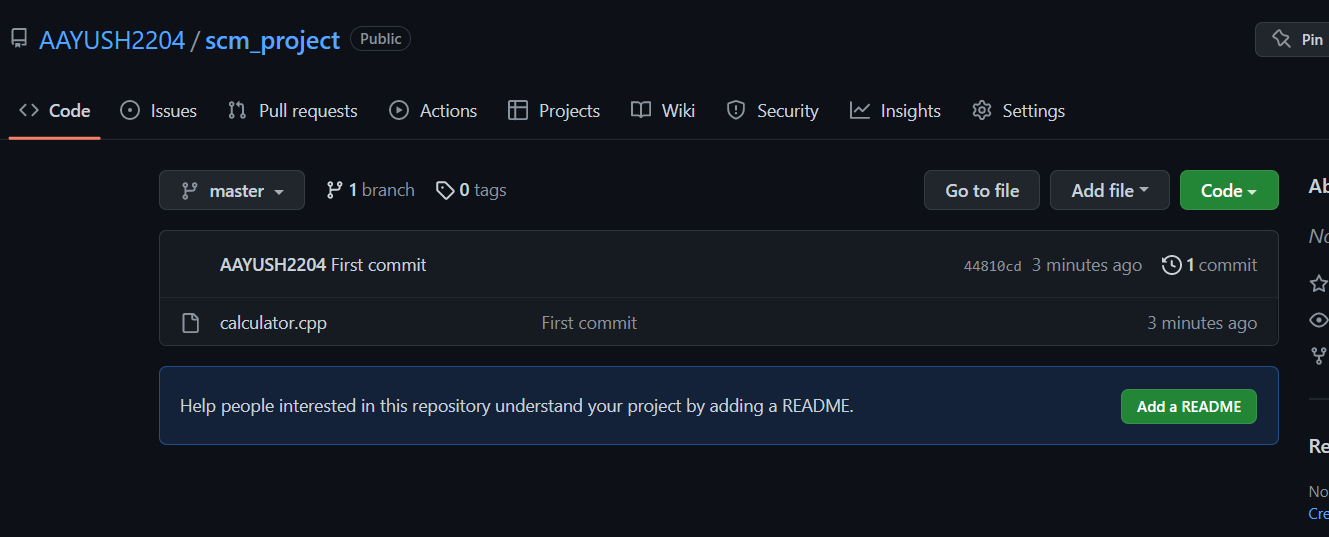
1. You have your own copy of the project on which you may test your own changes without changing the original project.
2. This helps the maintainer of the project to better check the changes you made to the project and has the power to either accept, reject or suggest something.
3. When you clone an Open Source project, which isn't yours, you don't have the right to push code directly into the project.

For these reasons, you are always suggested to FORK. Let's have a screenshot walkthrough of the whole process. When getting started with a contribution to Open Source Project, you have been advised to first FORK the repository(repo). But what is a fork?

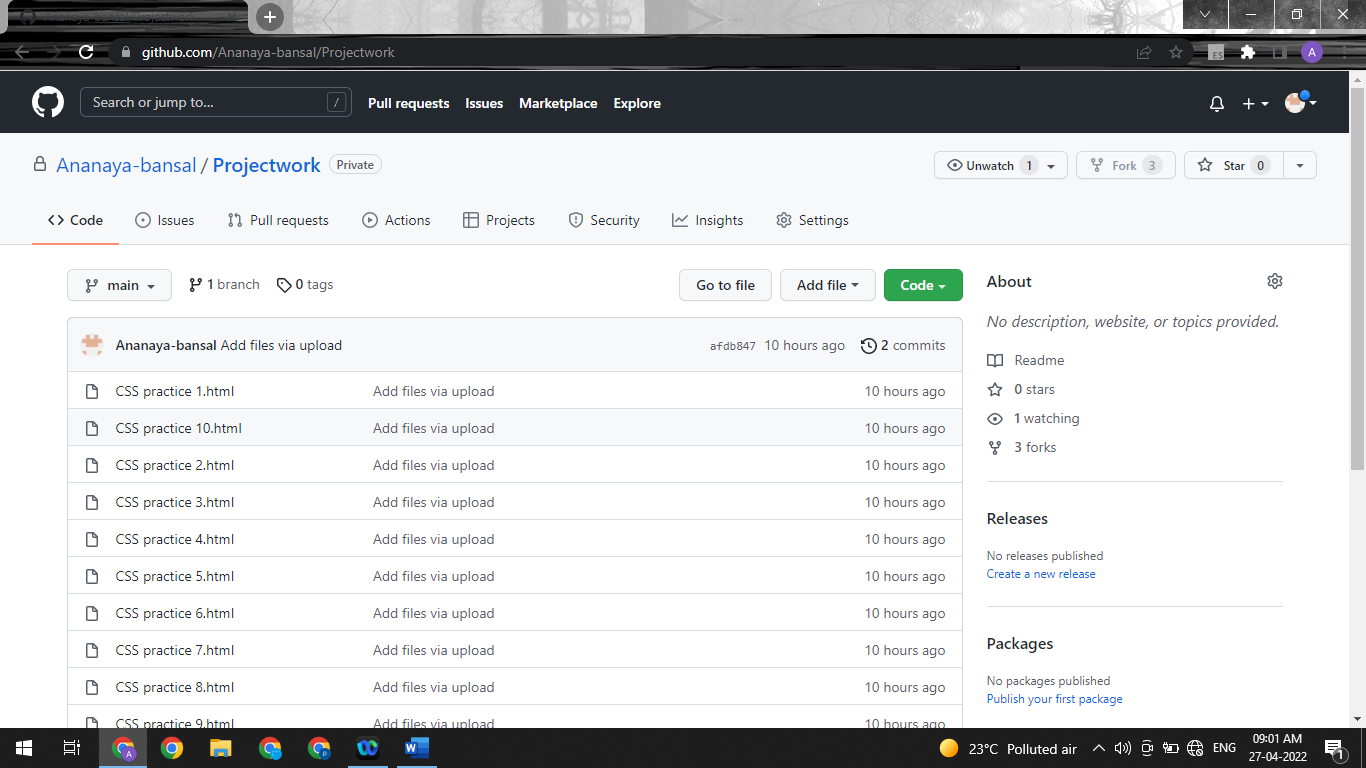
You must have seen this icon on every repository in the top right corner. Now, this button is used to Fork the repo. But again, what is a fork or forking a repository in GitHub as shown in the below media as follows:



**Step 1:** Go to **scm\_project** official repository.



**Step 2:** Find the Fork button on the top right corner.



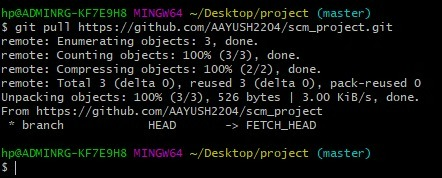
**Step 3:** Click on **Fork**.

You will see this screen.

Now you will have a copy of the repo you have forked from other user. Now you can do any modification you want without making changes to main source code.

Now type https://github.com/AAYUSH2204/scm\_project.git on CLI.

Git pull <url> --> This command is used to fetch the remote repo or to clone the repo.



Now Open the file make changes in it and commit it and push it to remote.



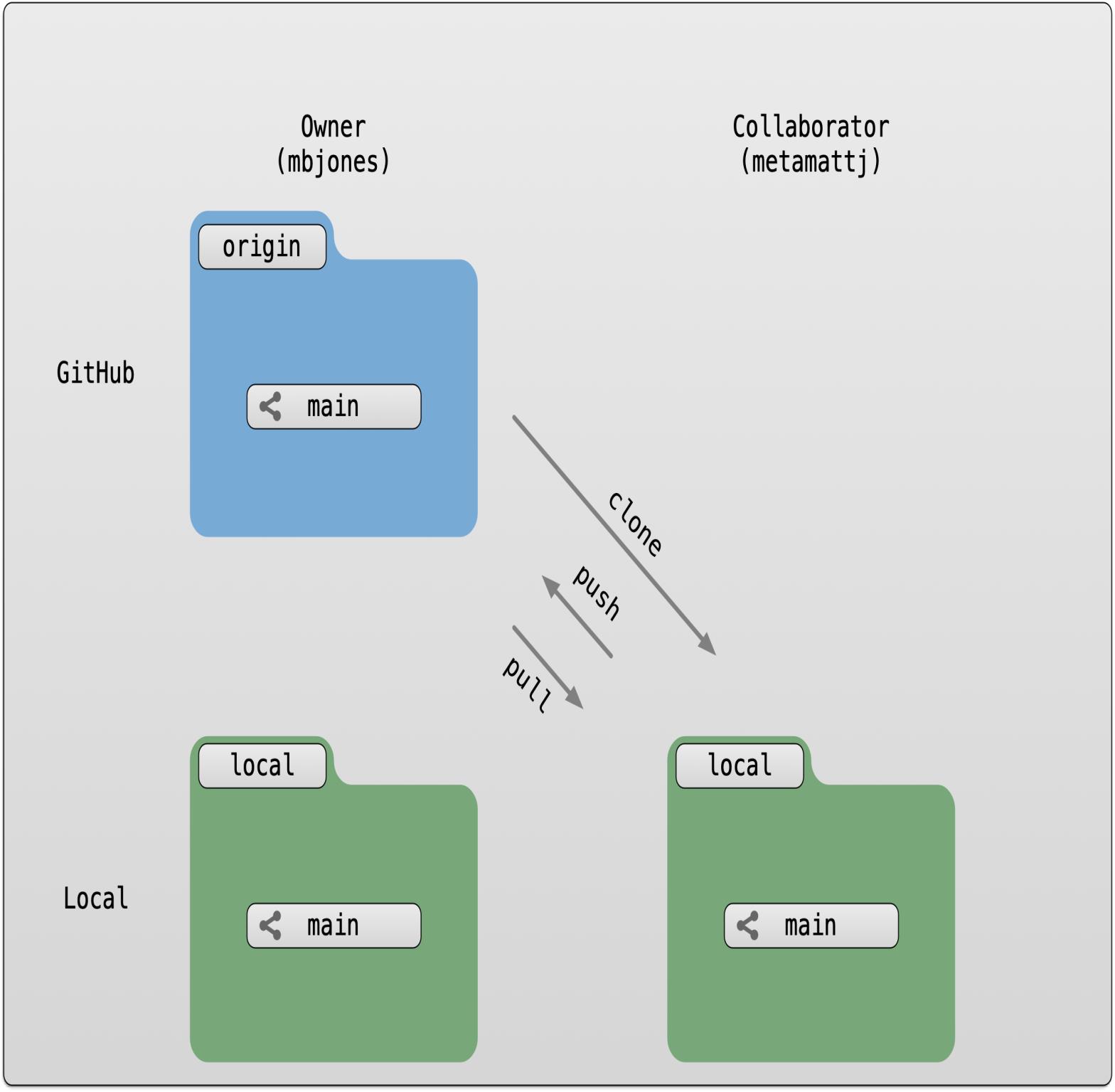
### Experiment No. 08

**Aim**: Merge and Resolve Conflicts

Git is a great tool for working on your own, but even better for working with friends and colleagues. Git allows you to work with confidence on your own local copy of files with the confidence that you will be able to successfully synchronize your changes with the changes made by others.

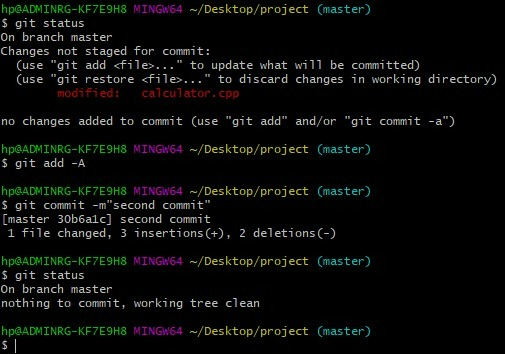
The simplest way to collaborate with Git is to use a shared repository on a hosting service such as GitHub, and use this shared repository as the mechanism to move changes from one collaborator to another. While there are other more advanced ways to sync git repositories, this “hub and spoke” model works really well due to its simplicity.

In this model, the collaborator will clone a copy of the owner’s repository from GitHub, and the owner will grant them collaborator status, enabling the collaborator to directly pull and push from the owner’s GitHub repository.

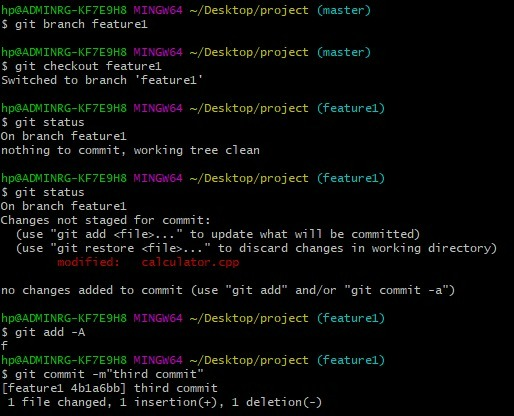


1. Do changes in master branch and commit those change. And checkout to different branch and again do changes and commit it. Now checkout to master branch and merge that branch in master.

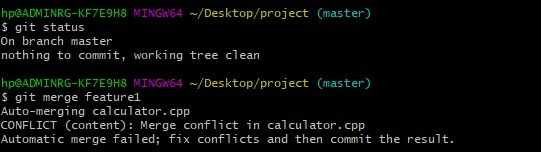
**COMMIT IN MASTER BRANCH**



**COMMIT IN FEATURE-1 BRANCH**

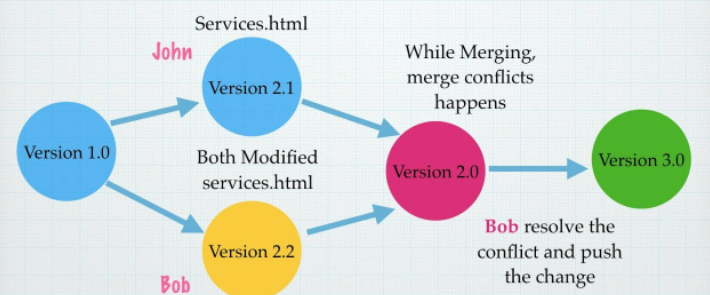


Now try to merge it will give Conflicts Error.

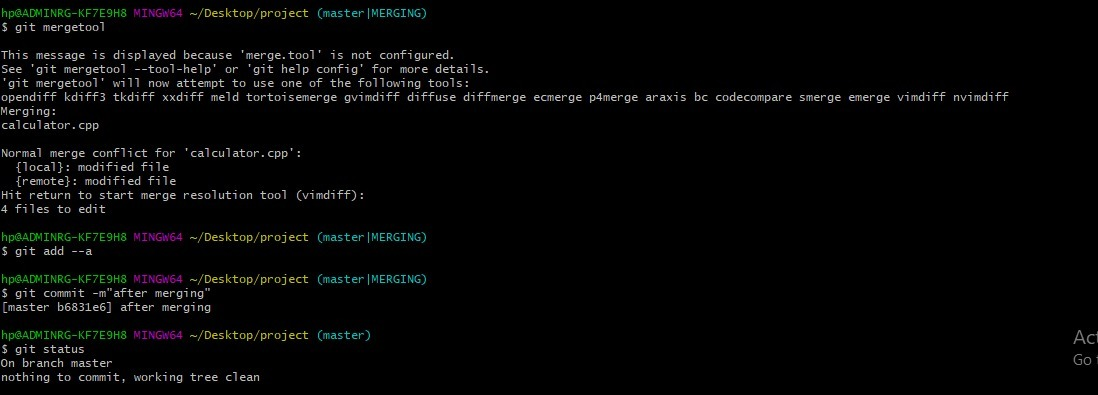


Use Command “git mergetool” to solve the conflict.

**git -mergetool** – Run merge conflict resolution tools to resolve merge conflicts.



Press “I” to insert, after insertion. Press “:wq”. The merge conflict is solved and our Feature-1 branch is merged to master branch.





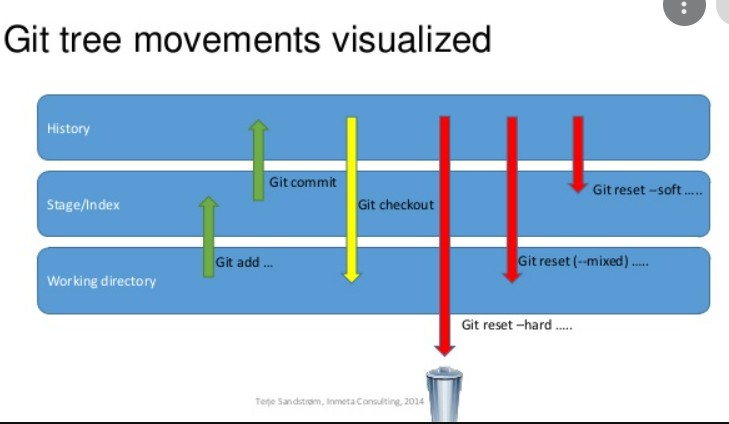
### Experiment No. 09

## **Aim:** Reset and Revert

While Working with Git in certain situations we want to undo changes in the working area or index area, sometimes remove commits locally or remotely and we need to reverse those changes. There are 3 different ways in which we can undo the changes in our repository, these are ***git reset***, ***git checkout***, and ***git revert***.

**1. git reset**

git reset is used when we want to unstage a file and bring our changes back to the working directory. git reset can also be used to remove commits from the local repository.



git reset HEAD <filename>

Whenever we unstage a file, all the changes are kept in the working area.

We are back to the working directory, where our changes are present but the file is now unstaged. Now there are also some commits that we don't want to get committed and we want to remove them from our local repository. To see how to remove the commit from our local repository let's stage and commit the changes that we just did and then remove that commit.

We have 2 commits now, with the latest being the Added Hello World commit which we are going to remove. The command that we would be using now is -

git reset HEAD~1

Points to be noted -

* HEAD~1 here means that we are going to remove the topmost commit or the latest commit that we have done.
* We cannot remove a specific commit with the help of git reset , for ex : we cannot say that we want to remove the second commit or the third commit , we can only remove latest commit or latest 2 commits ... latest N commits.(HEAD~n) [n here means n recent commits that needs to be deleted].

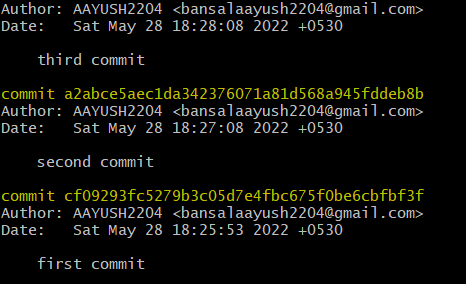
After using the above command we can see that our commit is being deleted and also our file is again unstaged and is back to the working directory. There are different ways in which git reset can actually keep your changes.

* ***git reset --soft HEAD~1*** *-* This command will remove the commit but would not unstage a file. Our changes still would be in the staging area.
* ***git reset --mixed HEAD~1*** or ***git reset HEAD~1 -*** This is the default command that we have used in the above example which removes the commit as well as unstages the file and our changes are stored in the working directory.
* ***git reset --hard HEAD~1 -*** This command removes the commit as well as the changes from your working directory. This command can also be called destructive command as we would not be able to get back the changes so be careful while using this command.

Points to keep in mind while using git reset command -

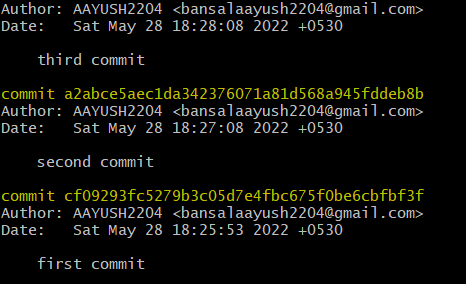
* If our commits are not published to remote repository , then we can use git reset.
* Use git reset only for removing commits that are present in our local directory and not in remote directory.
* We cannot remove a specific commit with the help of git reset , for ex : we cannot say that we want to remove the second commit or the third commit , we can only remove latest commit or latest 2 commits ... latest N commits.(HEAD~n) [n here means n recent commits that needs to be deleted].

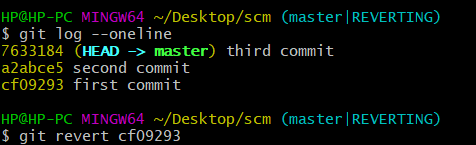
We just discussed above that the git reset command cannot be used to delete commits from the remote repository, then how do we remove the unwanted commits from the remote repository The command that we use here is –



**2. git revert**

git revert is used to remove the commits from the remote repository. Since now our changes are in the working directory, let's add those changes to the staging area and commit them.





**Difference Table**

|  |  |  |
| --- | --- | --- |
| **git checkout** | **git reset** | **git revert** |
| Discards the changes in the working repository. | Unstages a file and bring our changes back to the working directory | Removes the commits from the remote repository. |
| Used in the local repository. | Used in local repository | Used in the remote repository |
| Does not make any changes to the commit history. | Alters the existing commit history | Adds a new commit to the existing commit history. |
| Moves HEAD pointer to a specific commit. | Discards the uncommitted changes. | Rollbacks the changes which we have committed. |
| Can be used to manipulate commits or files. | Can be used to manipulate commits or files. | Does not manipulate your commits or files. |

