

# SOURCE CODE MANAGEMENT FILE

**Submitted by:**  
Kartavya Tomar  
2210990484  
Group 24-B

Submission of:  
Task 1.1

**Submitted To:**  
**Dr. Chetna Kaushal**  
Assistant Professor  
Department of Computer  
Science & Engineering  
Chitkara University Institute  
of Engineering and  
Technology  
Rajpura, Punjab

## **Source Code Management File**

Subject Name: **Source Code Management (SCM)**

Subject Code: **22CS003**

Vertical Name: First Year



**Submitted By:**

**Kartavya Tomar**

**2210990484**

**G24-B**



# INDEX

S. NO	Experiment Name	Page No.
1.	Setting up of Git Client	1-3
2.	Setting up GitHub Account	4-5
3.	Generate logs	6-8
4.	Create and visualize branches	9-10
5.	Git lifecycle description	11-12

# Experiment 1

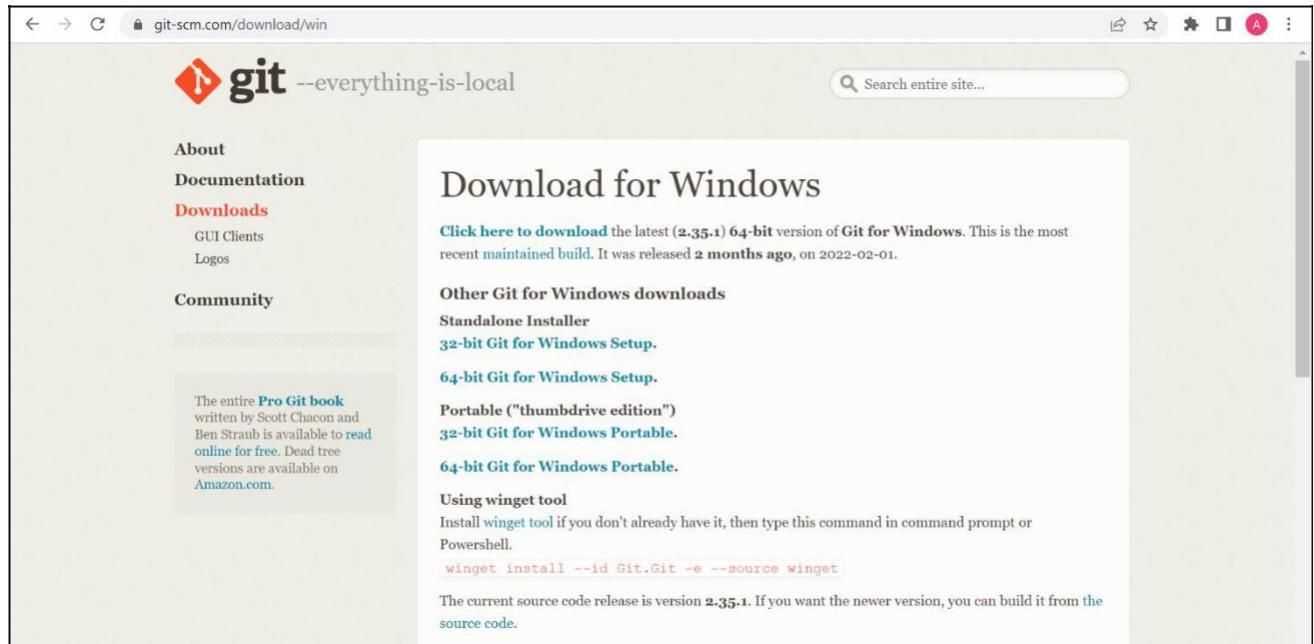
**Aim:** Setting up of Git Client

## Theory:

GIT: It's a Version Control System (VCS). It is a software or we can say a server by which we are able to track all the previous changes in the code. It is basically used for pushing and pulling of code. We can use git and git-hub parallelly to work with multiple members or individually. We can make, edit, recreate, copy or download any code on git hub using git.

**Procedure:** We can install Git on Windows, using the most official build which is available for download on the GIT's official website or by just typing (scm git) on any search engine. We can go on <https://git-scm.com/download/win> and can select the platform and bit-version to download. And after clicking on your desired bit-version or iOS it will start downloading automatically.

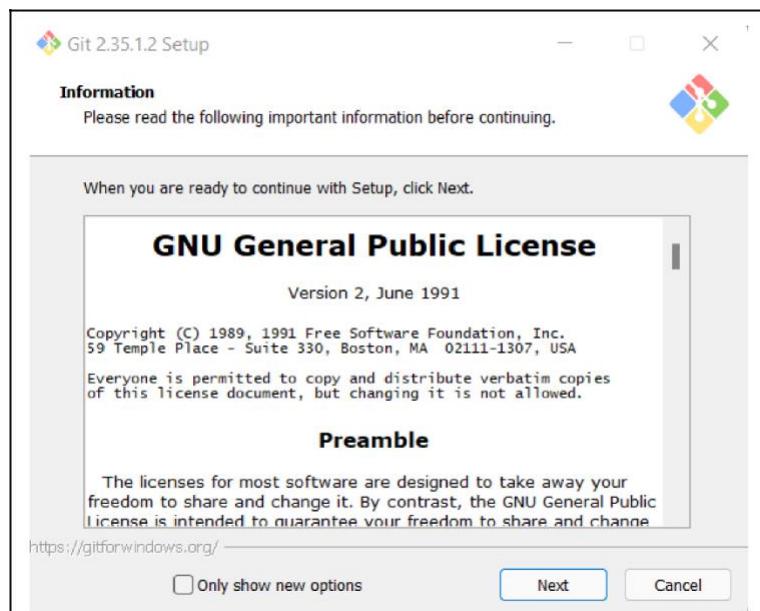
## Snapshots of download:



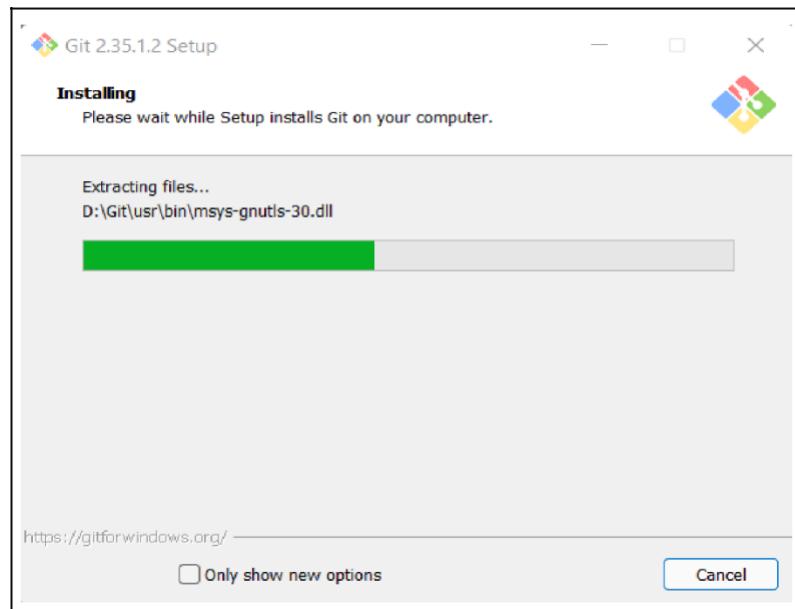
Opted for “64-bit Git for Windows Setup”

Name	Date modified	Type	Size
Git Bash	16-03-2022 08:51	Shortcut	2 KB
Git CMD	16-03-2022 08:51	Shortcut	2 KB
Git FAQs (Frequently Asked Questions)	16-03-2022 08:51	Internet Shortcut	1 KB
Git GUI	16-03-2022 08:51	Shortcut	2 KB
Git Release Notes	16-03-2022 08:51	Shortcut	2 KB

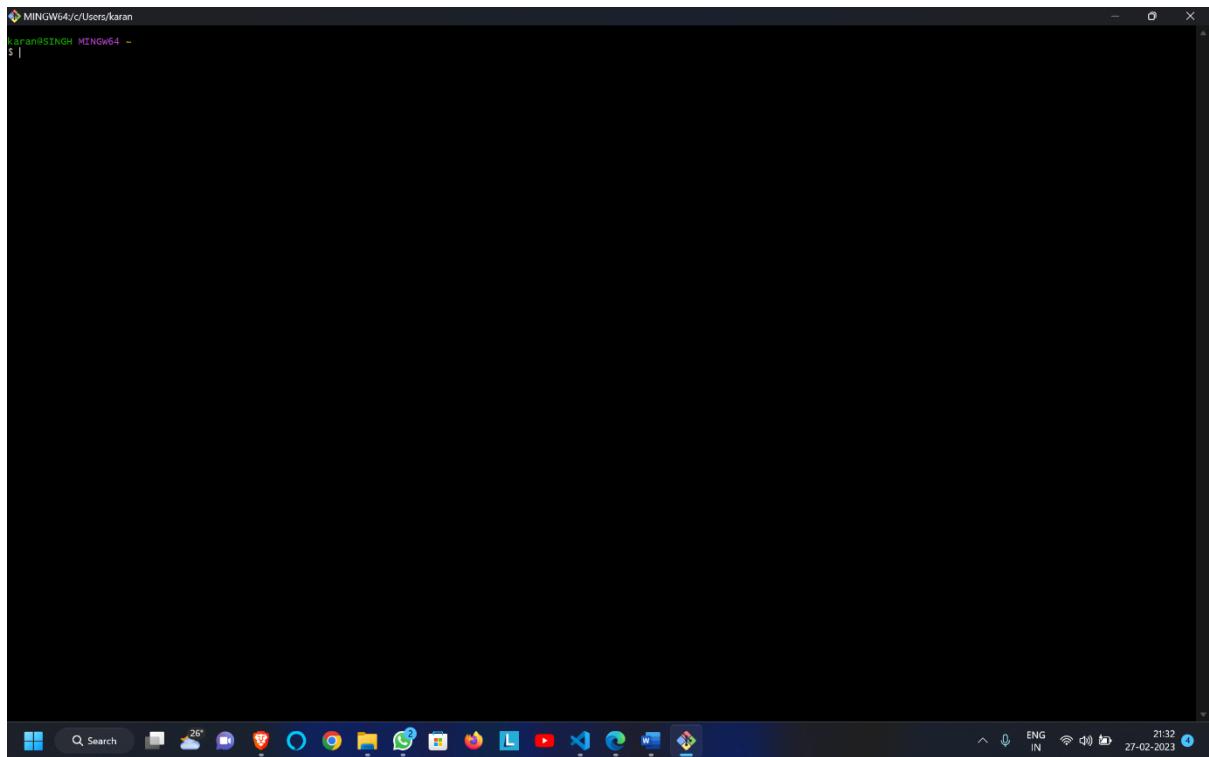
## Git and its files in downloads



## Git Setup



## Git Installation



Git Bash launched

# Experiment 2

**Aim:** Setting up GitHub Account

## Theory:

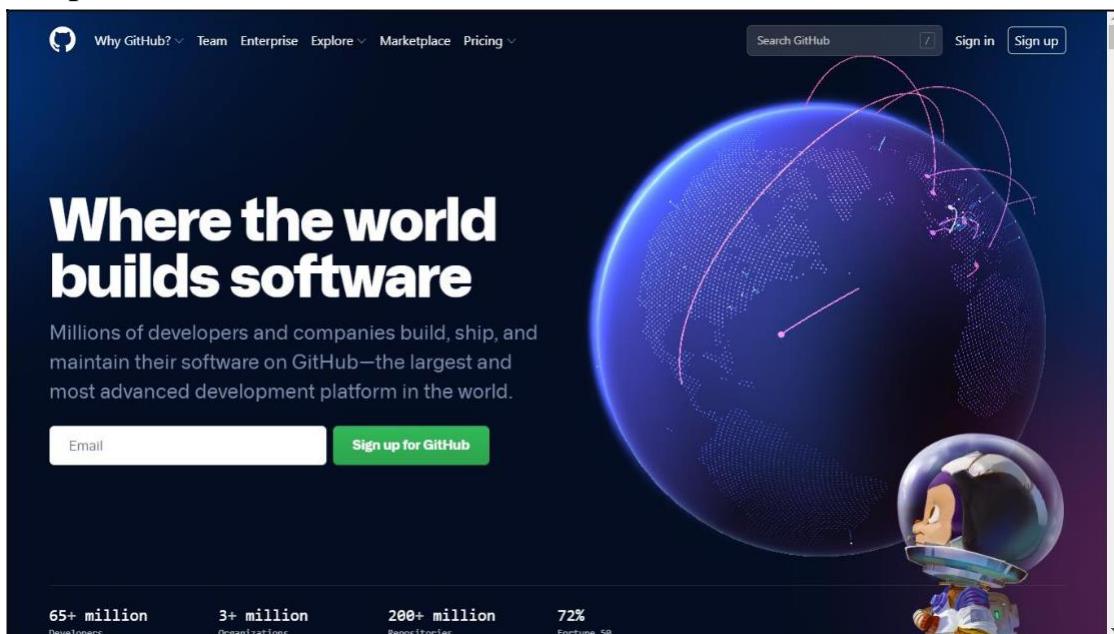
**GitHub:** GitHub is a website and cloud-based service (client) that helps an individual or developers to store and manage their code. We can also track as well as control changes to our or public code.

**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 1. 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.

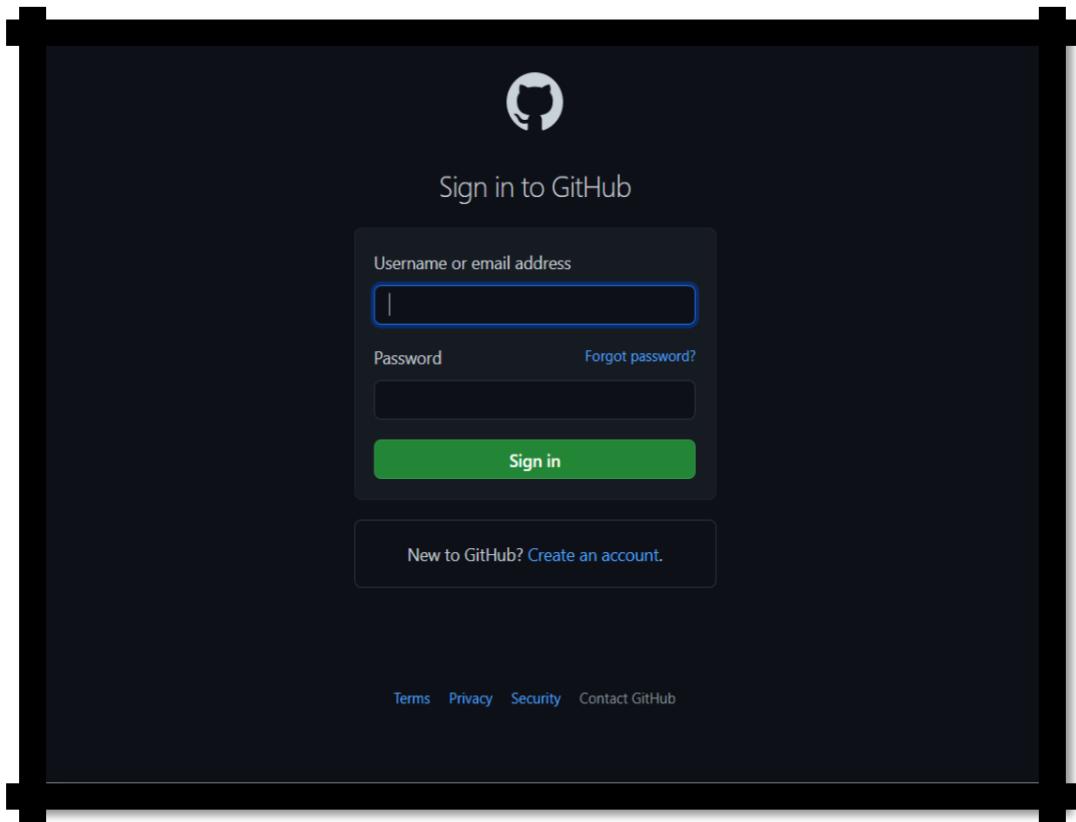
## Procedure:

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.

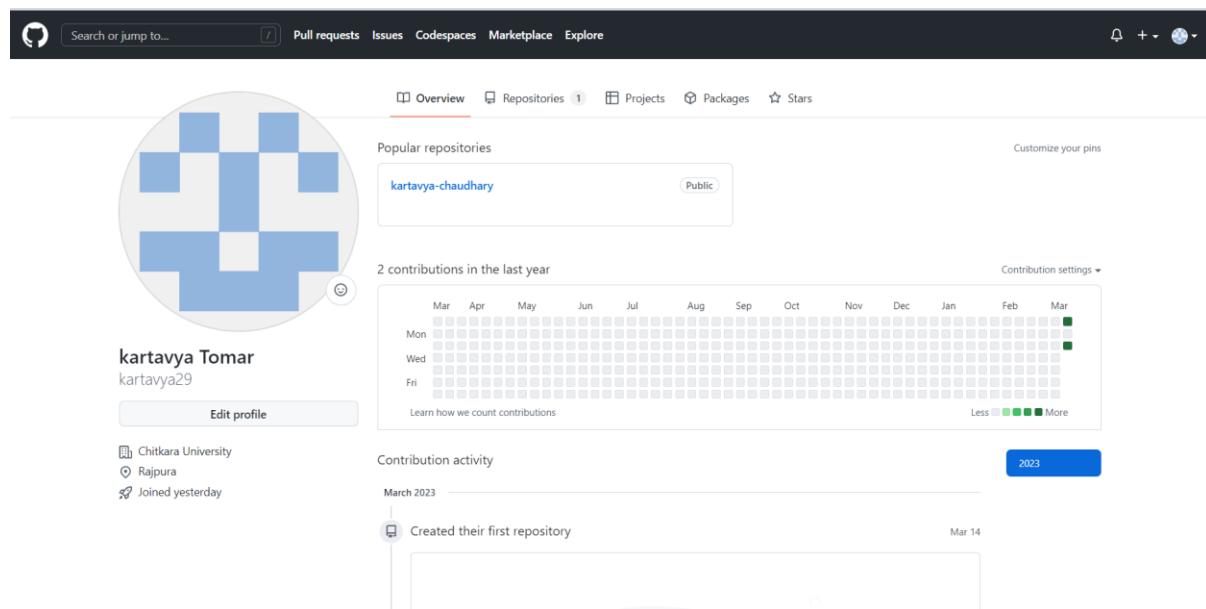
## Snapshots –



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

The image shows a screenshot of a GitHub user profile. At the top, there's a navigation bar with links for "Pull requests", "Issues", "Codespaces", "Marketplace", and "Explore". Below the navigation is a large circular profile picture with a blue and white checkered pattern. The user's name "kartavya Tomar" and handle "kartavya29" are displayed. There's a "Edit profile" button. To the right, there are sections for "Popular repositories" (listing "kartavya-chaudhary" as public), "Contribution settings" (showing a heatmap of activity from March 2023), and "Contribution activity" (listing "Created their first repository" on Mar 14). On the left, there's a sidebar with "Chitkara University", "Rajputra", and "Joined yesterday".

Search or jump to... Pull requests Issues Codespaces Marketplace Explore

kartavya Tomar  
kartavya29

Edit profile

Chitkara University Rajputra Joined yesterday

Overview Repositories 1 Projects Packages Stars

Popular repositories

kartavya-chaudhary Public

Customize your pins

2 contributions in the last year

Contribution settings

Less More

Contribution activity

March 2023

Created their first repository Mar 14

## GitHub Interface

# Experiment 3

**Aim:** Program to Generate log

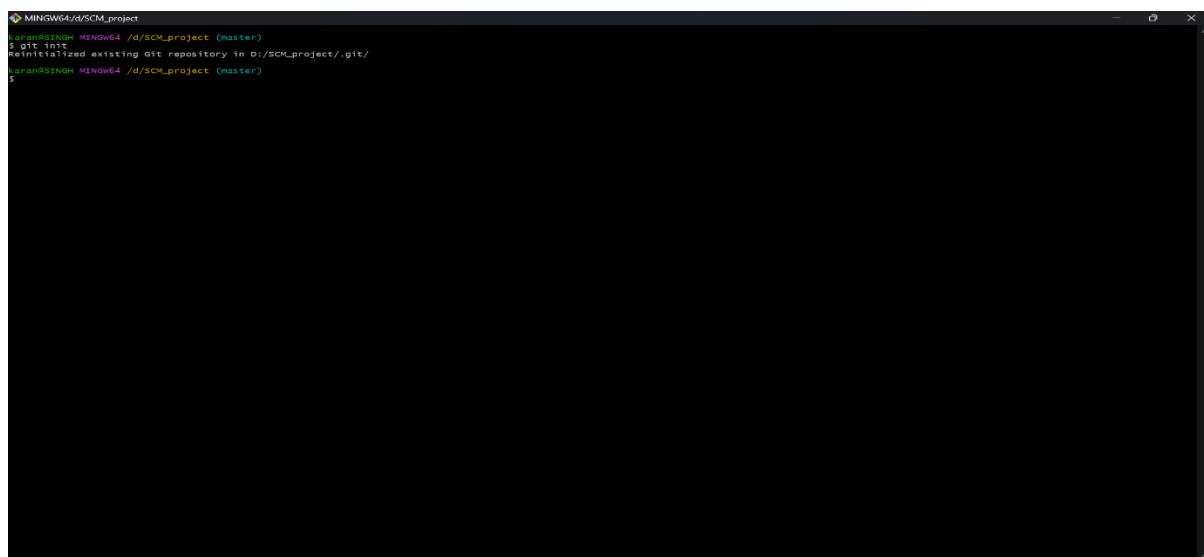
## Theory:

Logs: Logs are nothing but the history which we can see in git by using the code git log.

It contains all the past commits, insertions and deletions in it which we can see any time. Logs helps to check that what were the changes in the code or any other file and by whom. It also contains the number of insertions and deletions including at which time it was changed.

## Procedure:

First of all, create a local repository using Git. For this, you have to make a folder in your device, right click and select “Git Bash Here”. This opens the Git terminal. To create a new local repository, use the command “git init” and it creates a folder “. git”.



```
MINGW64 /d/SCM_project
$ git init
Initialized empty Git repository in d:/SCM_project/.git/
LaranSingh MINGW64 /d/SCM_project (master)
$
```

When we use GIT for the first time, we have to give the user name and email so that if I am going to change in project, it will be visible to all.

For this, we use command:

“git config --global user.name Name”  
“git config --global user.email email”

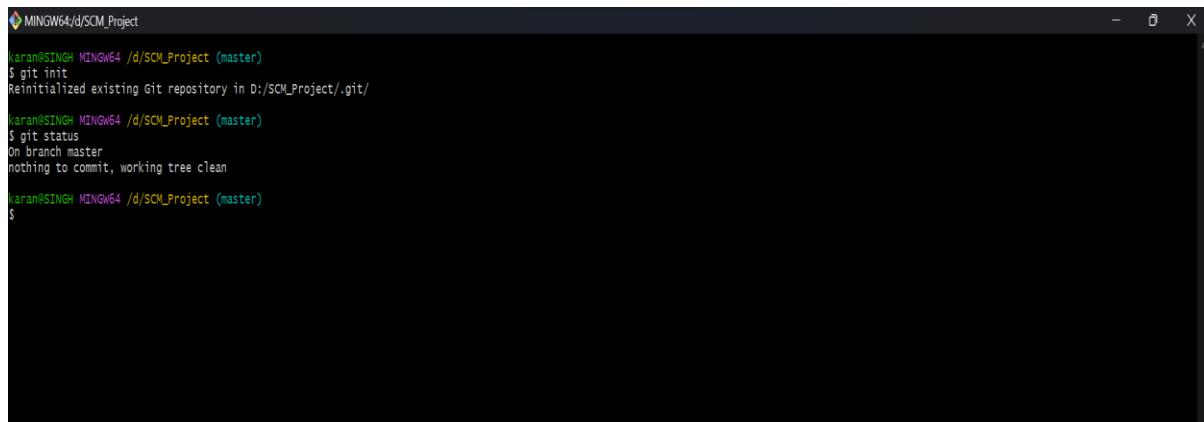
For verifying the user’s name and email, we use:

“git config --global user.name”  
“git config --global user.email”

## Some Important Commands

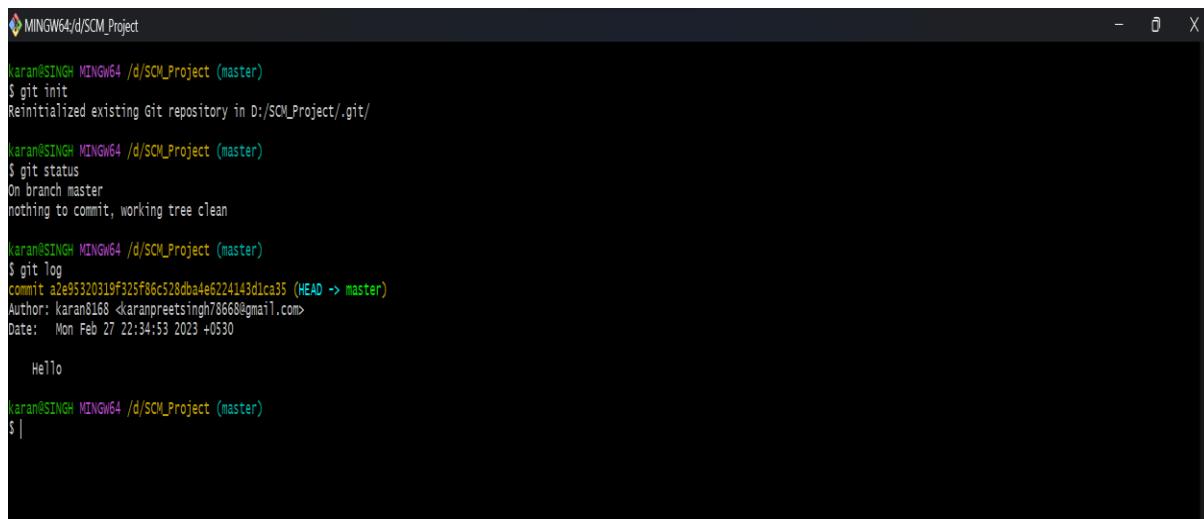
- ls → It gives the file names in the folder.
- ls -lart → Gives the hidden files also.
- git status → Displays the state of the working directory and the staged snapshot.
- touch filename → This command creates a new file in the repository.
- Clear → It clears the terminal.
- rm -rf .git → It removes the repository.
- git log → displays all of the commits in a repository's history
- git diff → It compares my working tree to staging area.

Git status:



```
MINGW64/d/SCM_Project
karan@SINGH MINGW64 /d/SCM_Project (master)
$ git init
Reinitialized existing Git repository in D:/SCM_Project/.git/
karan@SINGH MINGW64 /d/SCM_Project (master)
$ git status
On branch master
nothing to commit, working tree clean
karan@SINGH MINGW64 /d/SCM_Project (master)
$
```

Git log:



```
MINGW64/d/SCM_Project
karan@SINGH MINGW64 /d/SCM_Project (master)
$ git init
Reinitialized existing Git repository in D:/SCM_Project/.git/
karan@SINGH MINGW64 /d/SCM_Project (master)
$ git status
On branch master
nothing to commit, working tree clean
karan@SINGH MINGW64 /d/SCM_Project (master)
$ git log
commit a2e95320319f325f86c528da4e6224143d1ca35 (HEAD -> master)
Author: karan8168 <karandeep.singh78668@gmail.com>
Date:   Mon Feb 27 22:34:53 2023 +0530

    Hello

karan@SINGH MINGW64 /d/SCM_Project (master)
$ |
```

The git log command displays a record of the commits in a Git repository. By default, the git log command displays a commit hash, the commit message and other commit metadata.

# Experiment 4

**Aim:** Create and visualize branches

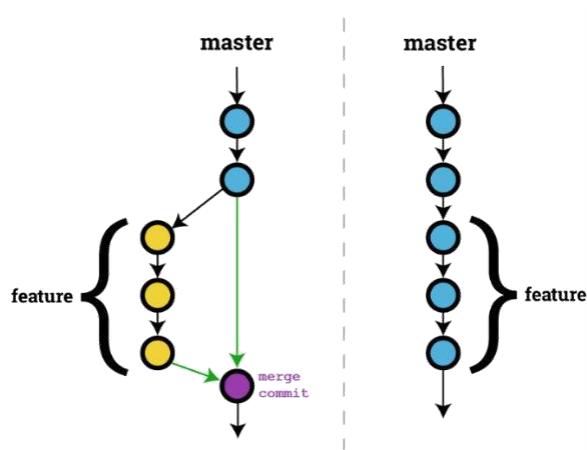
## Theory:

Branching: A branch in Git is an independent line of work (a pointer to a specific commit). It allows users to create a branch from the original code (master branch) and isolate their work. Branches allow you to work on different parts of a project without impacting the main branch.

Create branches: The main branch in git is called as master branch. But we can make branches out of this main master branch. All the files present in master can be shown in branch but the file which are created in branch are not shown in master branch. We can also merge both the parent (master) and child (other branches).

Syntax:

For creating a new branch, git branch name by default is master branch.



**Snapshots –**

```
karan@SINGH MINGW64 /d/SCM_Project (master)
$ git branch
* master

karan@SINGH MINGW64 /d/SCM_Project (master)
$
```

Default branch is master branch

```
MINGW64/d/SCM_Project
$ karan@SINGH MINGW64 ~/d/SCM_Project (master)
$ git branch
* master
karan@SINGH MINGW64 ~/d/SCM_Project (master)
$ git branch feature
karan@SINGH MINGW64 ~/d/SCM_Project (master)
* master
karan@SINGH MINGW64 ~/d/SCM_Project (master)
$
```

## Adding a feature branch

```
MINGW64/d/SCM_Project
$ karan@SINGH MINGW64 ~/d/SCM_Project (master)
$ git branch
* master
karan@SINGH MINGW64 ~/d/SCM_Project (master)
$ git branch feature
karan@SINGH MINGW64 ~/d/SCM_Project (master)
$ git branch
* feature
* master
karan@SINGH MINGW64 ~/d/SCM_Project (master)
$ git checkout feature
Switched to branch 'feature'
karan@SINGH MINGW64 ~/d/SCM_Project (feature)
$ git branch
* feature
* master
karan@SINGH MINGW64 ~/d/SCM_Project (feature)
$
```

## Switching to feature branch

```
MINGW64/d/SCM_Project
$ karan@SINGH MINGW64 ~/d/SCM_Project (master)
$ git branch
* master
karan@SINGH MINGW64 ~/d/SCM_Project (master)
$ git branch feature
karan@SINGH MINGW64 ~/d/SCM_Project (master)
$ git branch
* feature
* master
karan@SINGH MINGW64 ~/d/SCM_Project (master)
$ git checkout feature
Switched to branch 'feature'
karan@SINGH MINGW64 ~/d/SCM_Project (feature)
$ git branch
* feature
* master
karan@SINGH MINGW64 ~/d/SCM_Project (feature)
$ git checkout master
Switched to branch 'master'
karan@SINGH MINGW64 ~/d/SCM_Project (master)
$ git branch
* master
* feature
karan@SINGH MINGW64 ~/d/SCM_Project (master)
$
```

## Switching to master branch

## Checking commits

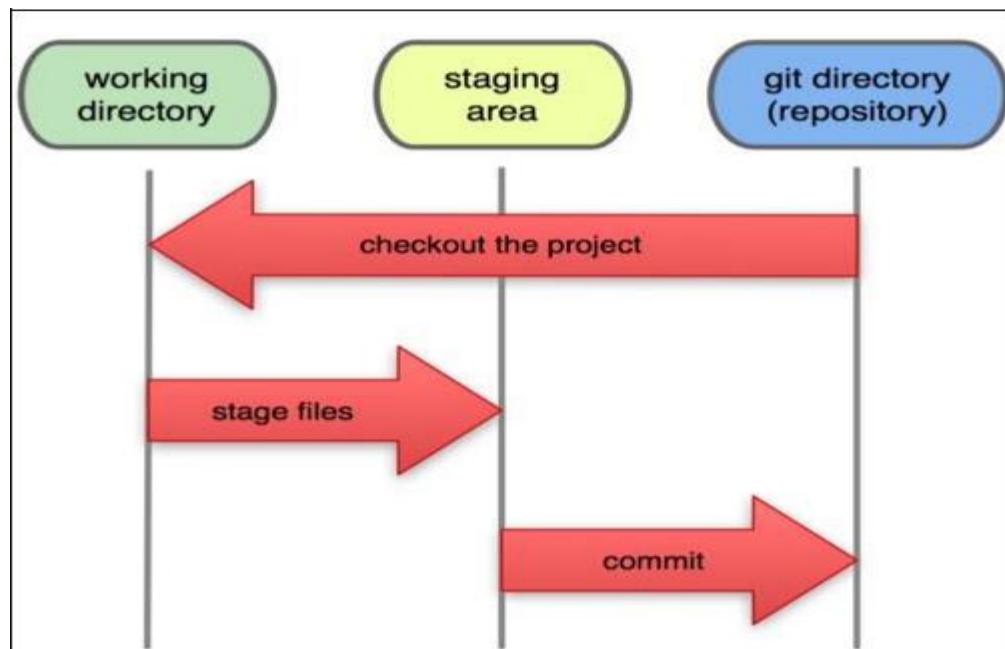
# Experiment 5

**Aim:** Git lifecycle description

## Theory:

Stages in GIT Life Cycle: Files in a Git project have various stages like Creation, Modification, Refactoring, and Deletion and so on. Irrespective of whether this project is tracked by Git or not, these phases are still prevalent. However, when a project is under Git version control system, they are present in three major Git states in addition to these basic ones. Here are the three Git states:

- Working directory
- Staging area
- Git directory (repository)



## Working Directory:

Consider a project residing in your local system. This project may or may not be tracked by Git. In either case, this project directory is called your Working directory.

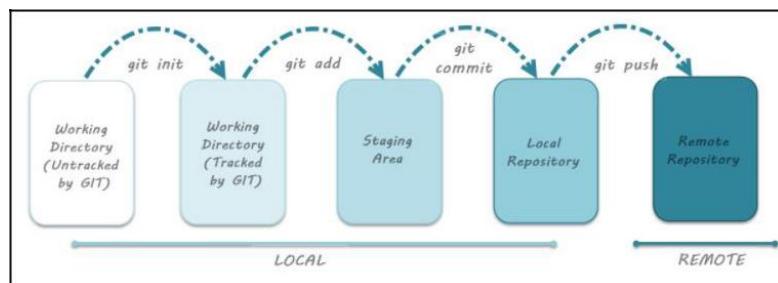
## Staging Area:

Staging area is the playground where you group, add and organize the files to be committed to Git for tracking their versions.

## Git Directory:

Now that the files to be committed are grouped and ready in the staging area, we can commit these files. So, we commit this group of files along with a commit message explaining what is the commit about. Apart from commit message, this step also records the author and time of the commit. Now, a snapshot of the files in the commit is recorded by Git. The information related to this commit is stored in the Git directory.

**Remote Repository:** It means mirror or clone of the local Git repository in GitHub. And pushing means uploading the commits from local Git repository to remote repository hosted in GitHub.



**Subject Name: Source Code Management**

**Subject Code: 22CS003**

**Vertical Name: First Year**

**Department: DCSE**



**CHITKARA  
UNIVERSITY**

**PUNJAB**

**Submitted By:**

Kartavya Tomar

2210990484

G24-B

Submission of Task 1.2

**Submitted To:**

Dr. Chetna Kaushal

Assistant Professor

Department of Computer

Science & Engineering

Chitkara University Institute of

Engineering and Technology

Rajpura, Punjab

**# Index #**

S. No	Program Title	Page No.
1.	Add collaborators on GitHub Repo	16
2.	Fork and commit	21
3.	Merge and Resolve conflicts created due to own activity and collaborators activity.	24
4.	Reset and revert	27



# Experiment 1

**Aim:** Add collaborators on GitHub Repo

## Theory:

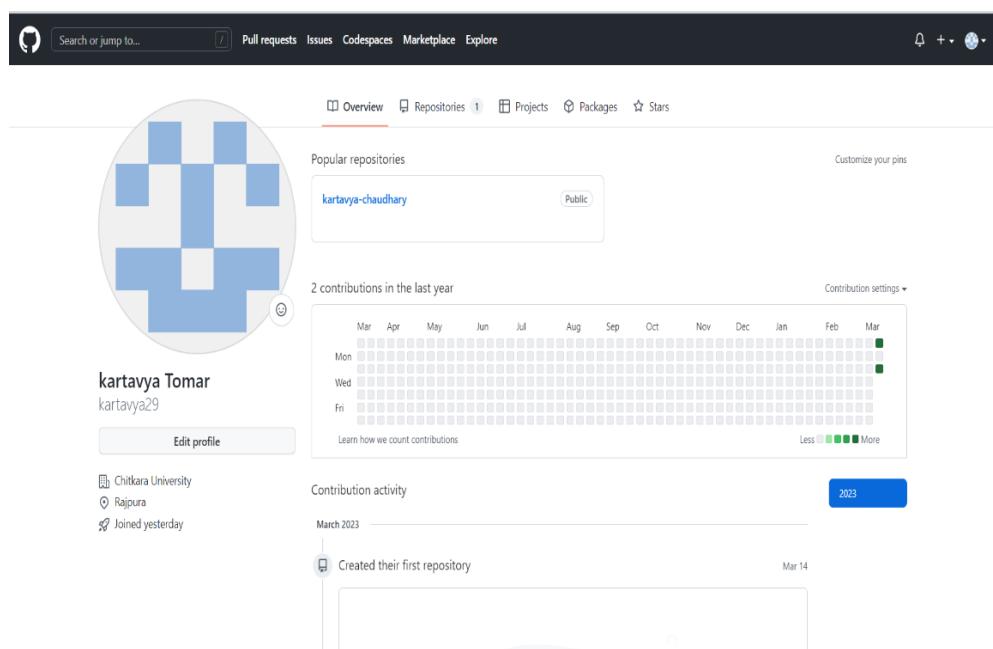
Whenever you make a repository in GitHub, not everyone has the permission to change or push codes into your repository. The users have a read-only access. In order to allow other individuals to make changes to your repository, you need to invite them to collaborate to the project.

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.

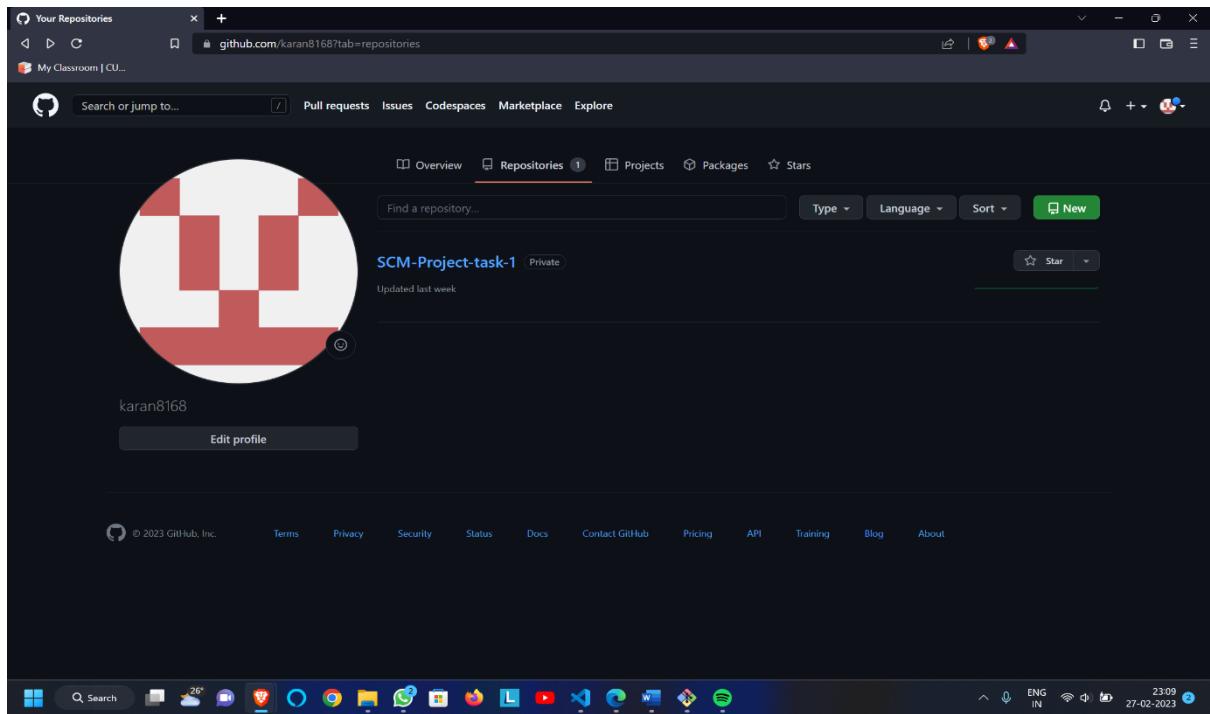
Being a collaborator, the user can create, merge and close pull requests in the repository. They can also remove them as the collaborator.

## Procedure:

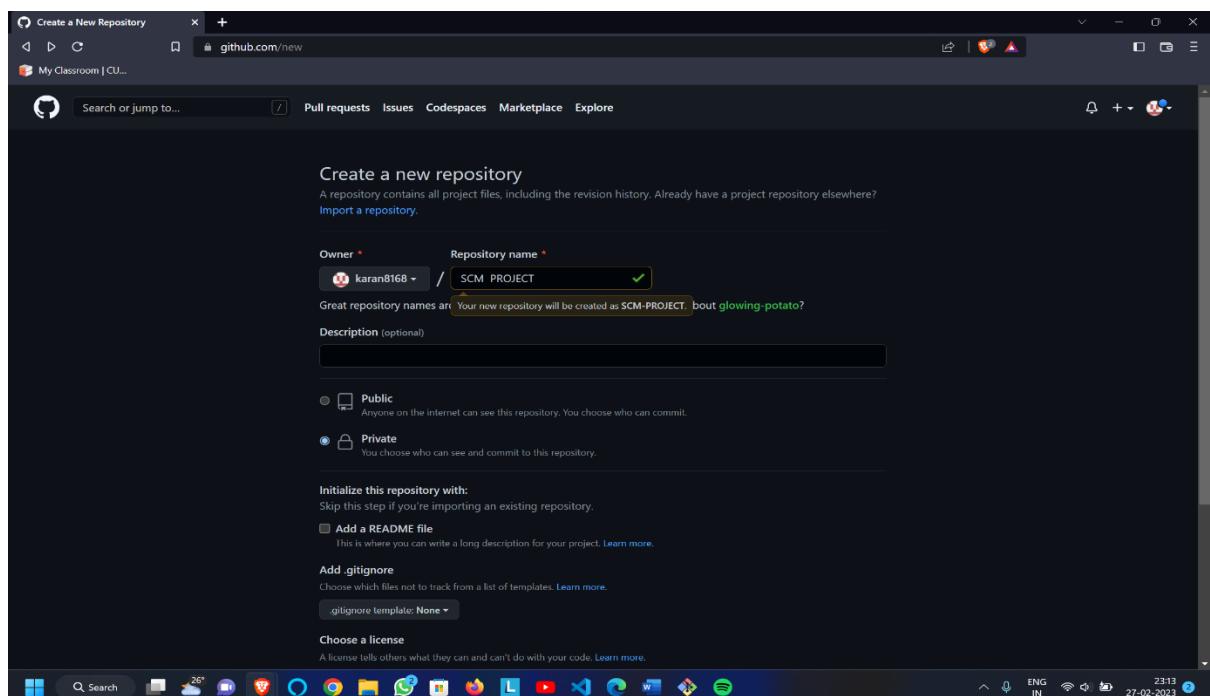
1. Login to your GitHub account and you will land on the homepage as shown below. Click on Repositories option in the menu bar.



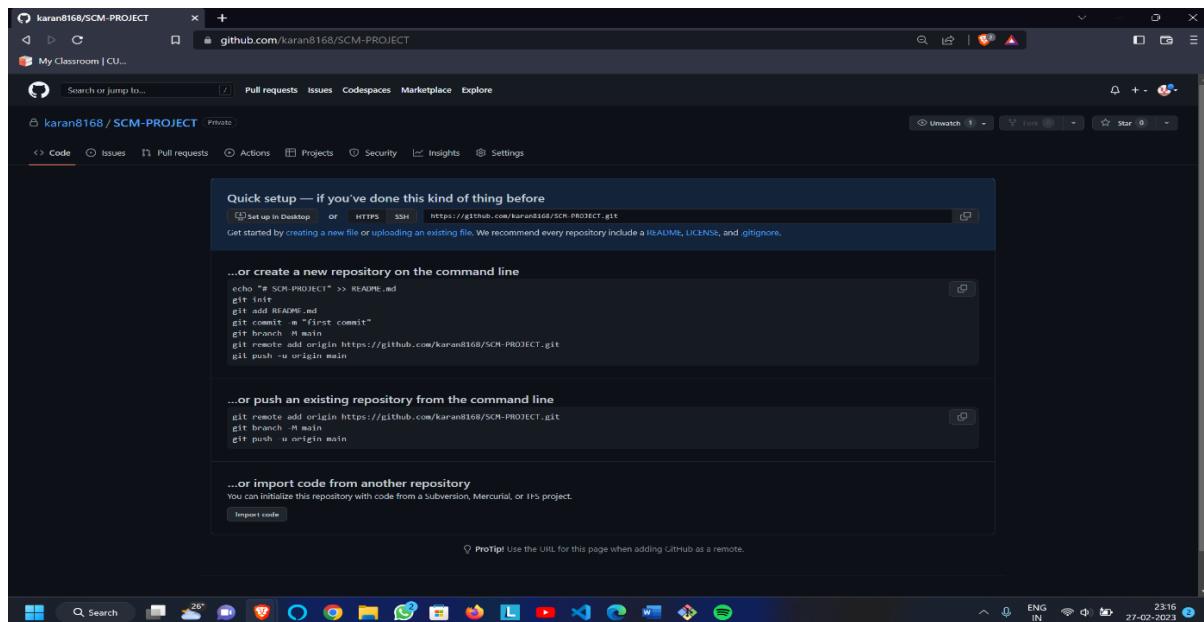
2. Click on the ‘New’ button in the top right corner.



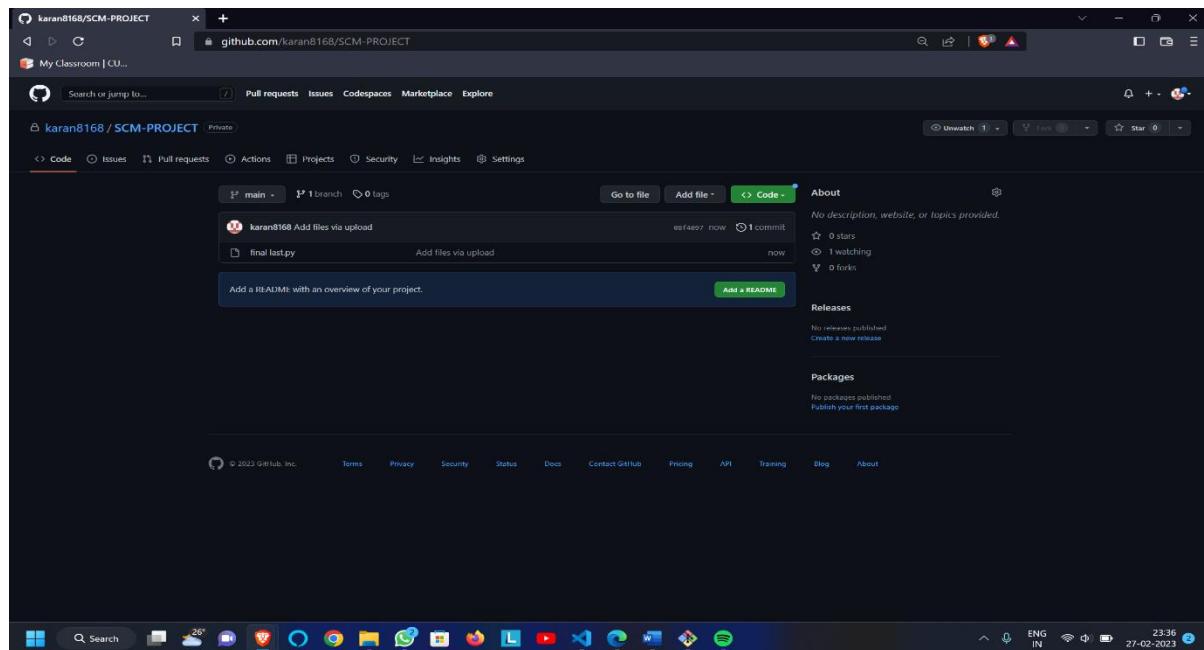
3. Enter the Repository name and add the description of the repository.
4. Select if you want the repository to be public or private.



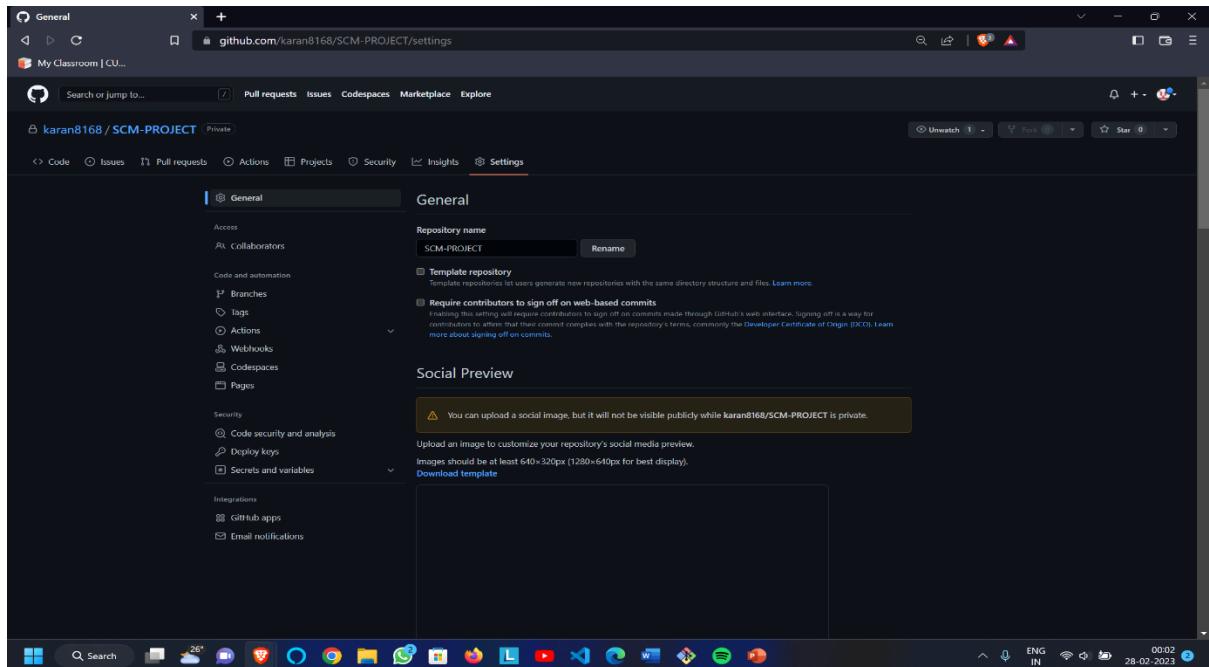
5. If you want to import code from an existing repository select the import code option.



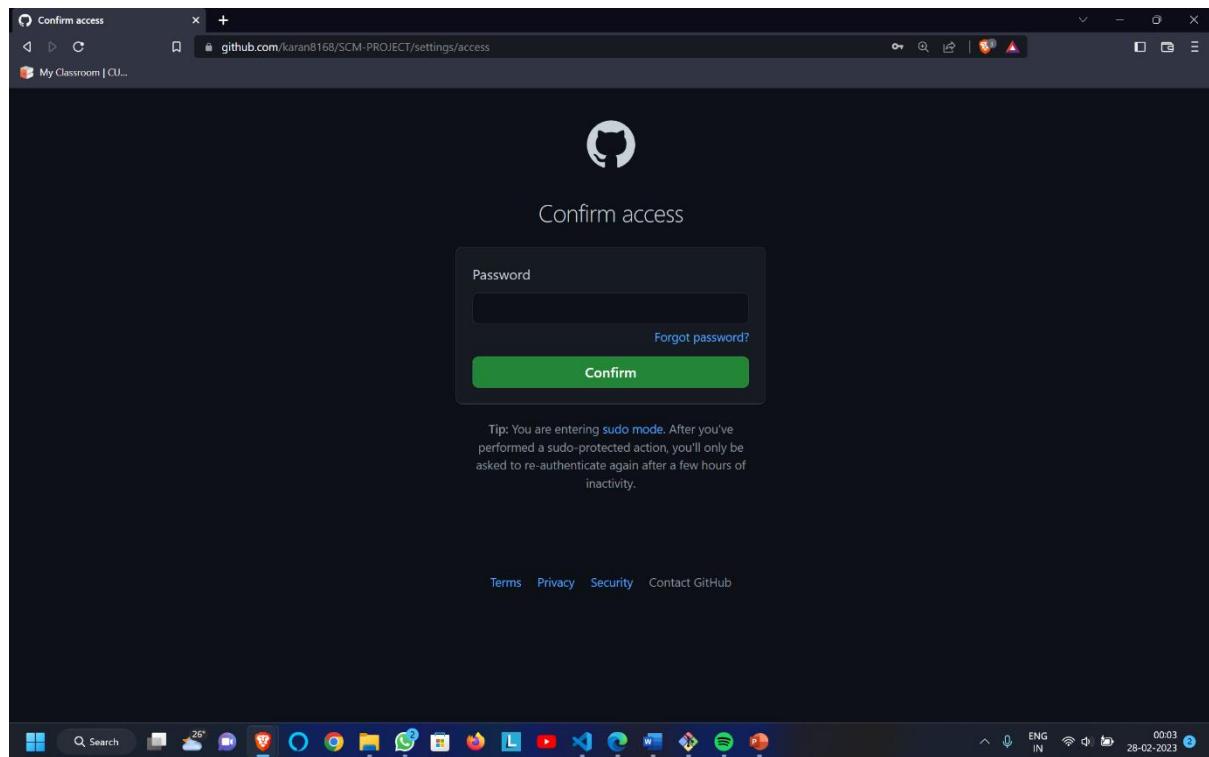
6. Now, you have created your repository successfully.
7. To add collaborators to your repository, open your repository and select settings option in the navigation bar.



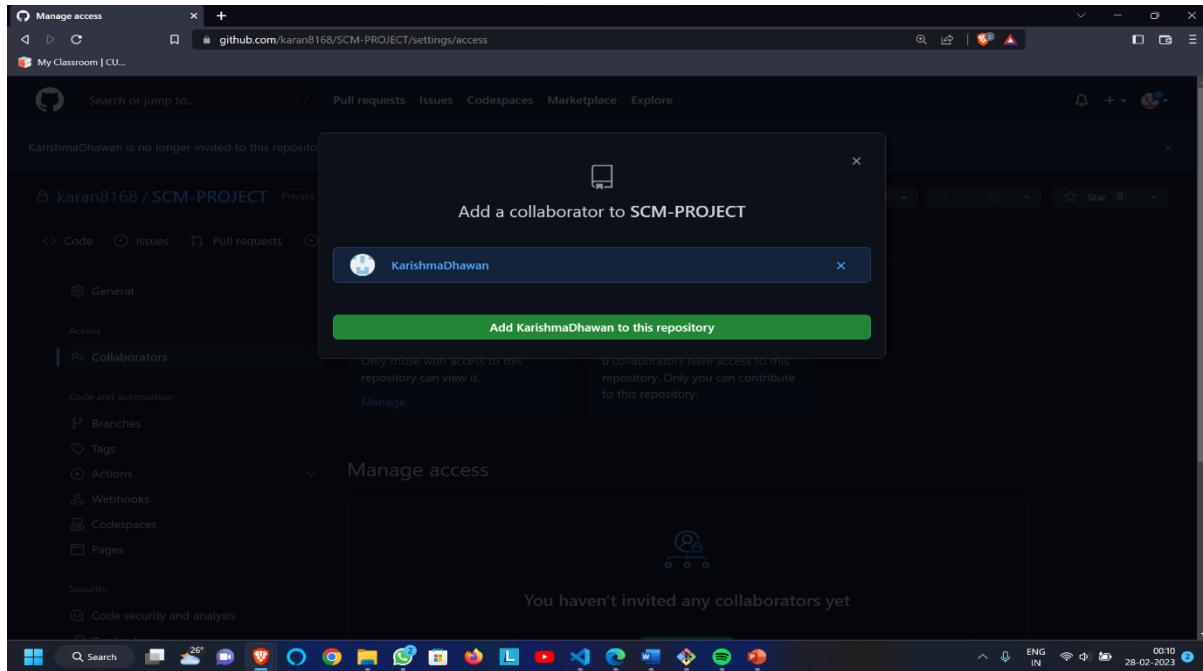
## 8. Click on Collaborators option under the access tab.



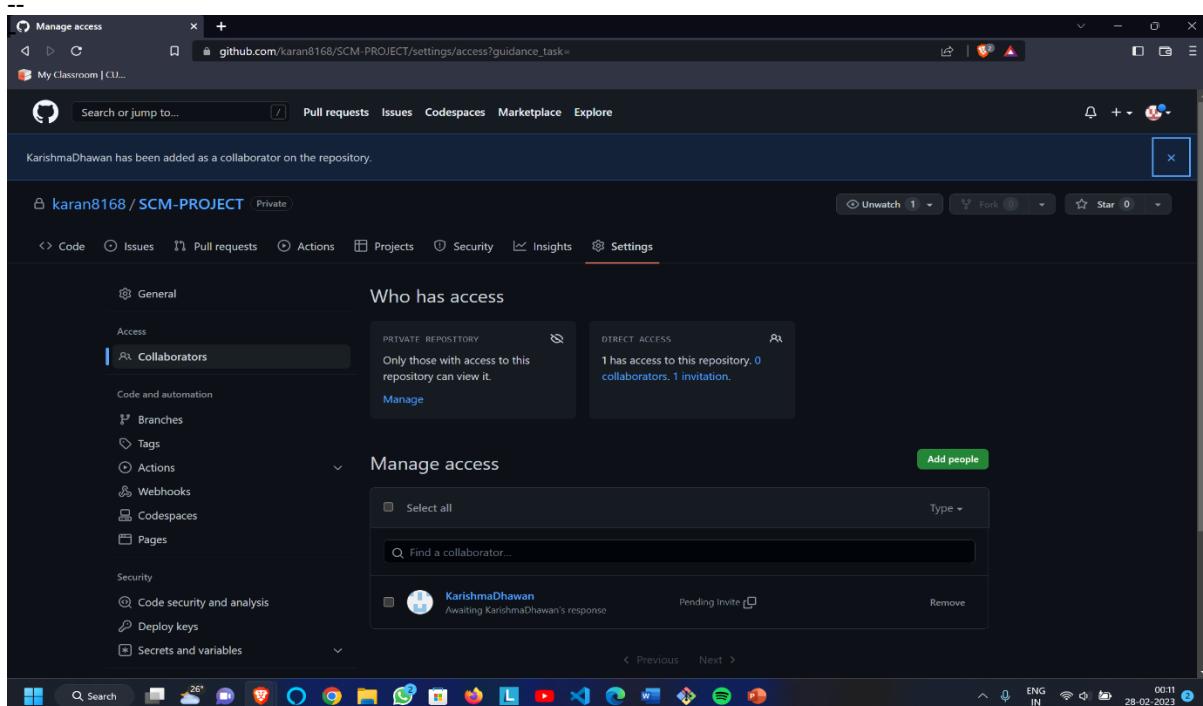
## 9. After clicking on collaborators, GitHub asks you to enter your password to confirm the access to the repository.



10. After entering the password, you can manage access and add/remove team members to your project.
11. To add members, click on the add people option and search the id of your respective team member.



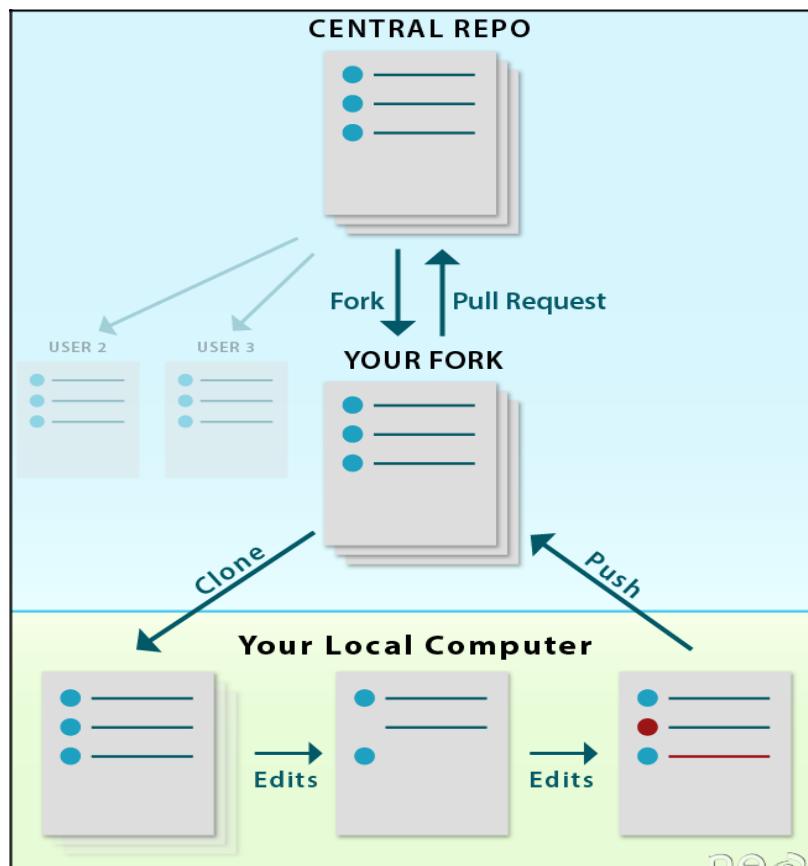
12. To remove any member, click on remove option available in the last column of member's respective row.



# Experiment 7

## Aim: Fork and Commit

**Theory:** A fork is a copy of a repository that you manage. It allows us to freely experiment with the data. After creating a fork, we can make any desired change like adding collaborators, rename files, generate GitHub pages but all these changes won't be reflected in the original repository.



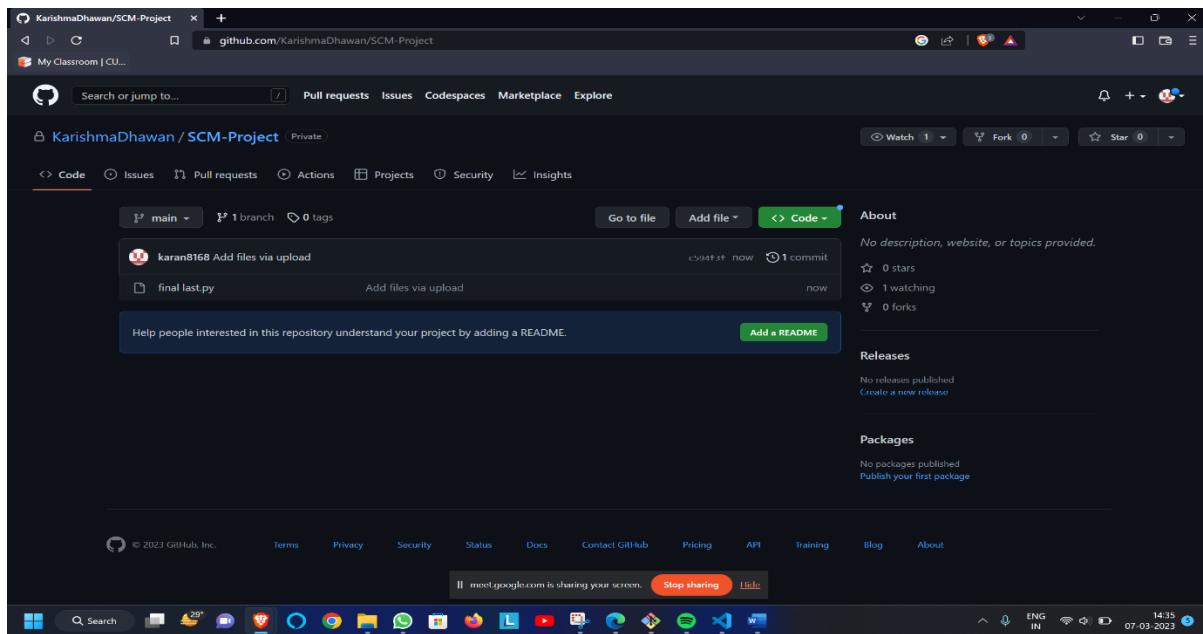
reference for picture: <https://www.earthdatascience.org>

To import the changes into the original repository, the user needs to send a pull request to the maintainer. If the maintainer closes the pull request only then the content can be added to the original repository.

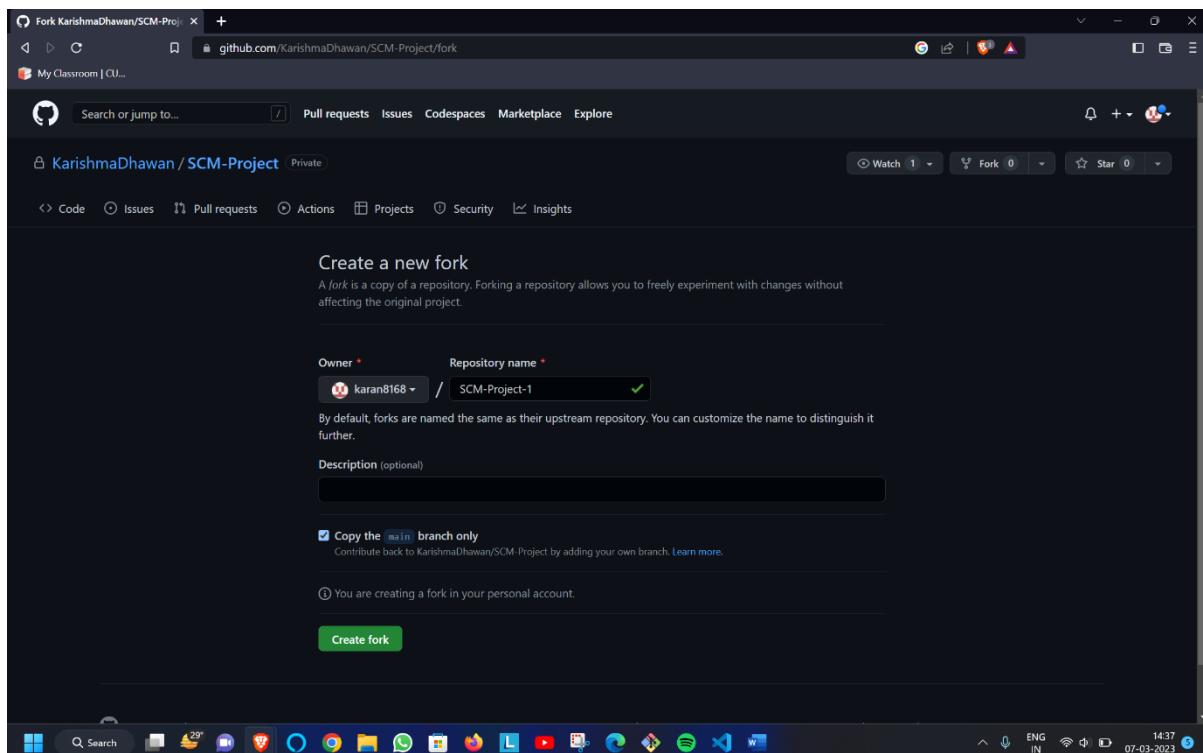
Forking is a better method than directly cloning any repository, as in cloning only the default branch is cloned whereas forking creates a clone of the complete repo and also allows us to push the changes to the main repository by using open and close pull request.

## Procedure:

1. To fork a repository first thing, you need to do is, sign in to your GitHub account and then you come to the repository you want to fork, so here for demo purpose



2. Click on the **Fork** button on right upside corner. Then it will ask to create a new fork, add description if you want and then click on create fork.



3. 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.
4. Now type git clone <https://github.com/karan8168/SCM-Project-1> on git.  
Git clone <URL> --> This command is used to fetch the remote repo or to clone the repo.

```

karan@SINGH MINGW64 /d/SCM_project (master)
$ git clone https://github.com/karan8168/SCM-Project-1
Cloning into 'SCM-Project-1'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
karan@SINGH MINGW64 /d/SCM_project (master)
$ 
```

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

```

karan@SINGH MINGW64 /d/SCM_project (master)
$ git remote add origin https://github.com/karan8168/SCM-Task1-.git
karan@SINGH MINGW64 /d/SCM_project (master)
$ git push -u origin master

Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 253 bytes | 126.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'master' on GitHub by visiting:
remote:   https://github.com/karan8168/SCM-Task1-/pull/new/master
remote:
To https://github.com/karan8168/SCM-Task1-.git
 * [new branch]    master -> master
branch 'master' set up to track 'origin/master'.

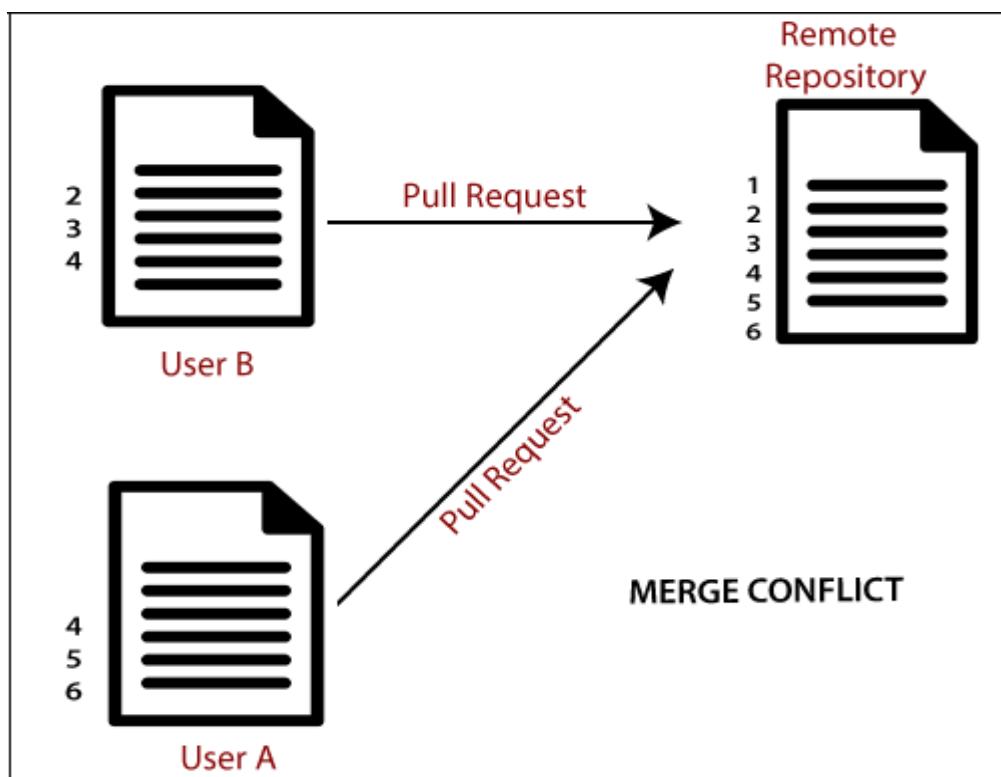
```

# Experiment 8

**Aim:** Merge and Resolve conflicts created due to own activity and collaborators activity.

## Theory:

Version control systems are all about managing contributions between multiple distributed authors (usually developers). Sometimes multiple developers may try to edit the same content. If Developer A tries to edit code that Developer B is editing a conflict may occur.



reference for picture: <https://www.javatpoint.com/git-merge-and-merge-conflict>

If you have a merge conflict on the command line, you cannot push your local changes to GitHub until you resolve the merge conflict locally on your computer.

To alleviate the occurrence of conflicts developers will work in separate isolated branches. If a merge conflict still arises between the compare branch and base branch in your pull request, you can view a list of the files with conflicting changes above the Merge pull request button. The Merge pull request button is deactivated until you've resolved all conflicts between the compare branch and base branch.

## Procedure:

- 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.

The screenshot shows a terminal window with the following content:

```
GNU nano 5.9
#include<iostream>
using namespace std;

//this code will print star pattern
int main(){
    /* Read input as specified in the question.
     * Print output as specified in the question.
     */
    int n;
    cin>>n;
    for(int i=1;i<=n;i++){
        int gap=n-i;
        for(int j=1;j<=gap;j++){
            cout<< " ";
        }
        for(int k=1;k<=i;k++){
            cout<< "*";
        }
        for(int t=1;t<=i-1;t++){
            cout<< "\n";
        }
        cout<< endl;
    }
    return 0;
}
```

At the bottom of the terminal, there is a menu bar with the following options:

- File Name to Write [DOS Format]: starpattern.cpp
- File Help
- File Cancel
- M-D DOS Format
- M-H Mac Format
- M-A Append
- M-P Prepend
- M-B Backup File
- AT Browse

2. Now try to merge it will give Conflicts Error.

The screenshot shows a terminal window with the following command history:

```
karan@KARAN-OptiPlex-5090:~/Desktop$ cd feature
karan@KARAN-OptiPlex-5090:~/Desktop/feature$ git add .
karan@KARAN-OptiPlex-5090:~/Desktop/feature$ git commit -m "Initial commit"
[feature 9770ff] Initial commit
 1 file changed, 1 insertion(+)
 create mode 100644 file.txt
karan@KARAN-OptiPlex-5090:~/Desktop/feature$ git push origin feature
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), done.
Total 3 (delta 0), reused 0 (delta 0)
remote: Resolving deltas: 100% (3/3)
To https://github.com/karan98singh/test.git
 * [new branch] feature -> feature
Branch feature set up to track remote branch feature from origin.
```

Then, the user runs a merge command:

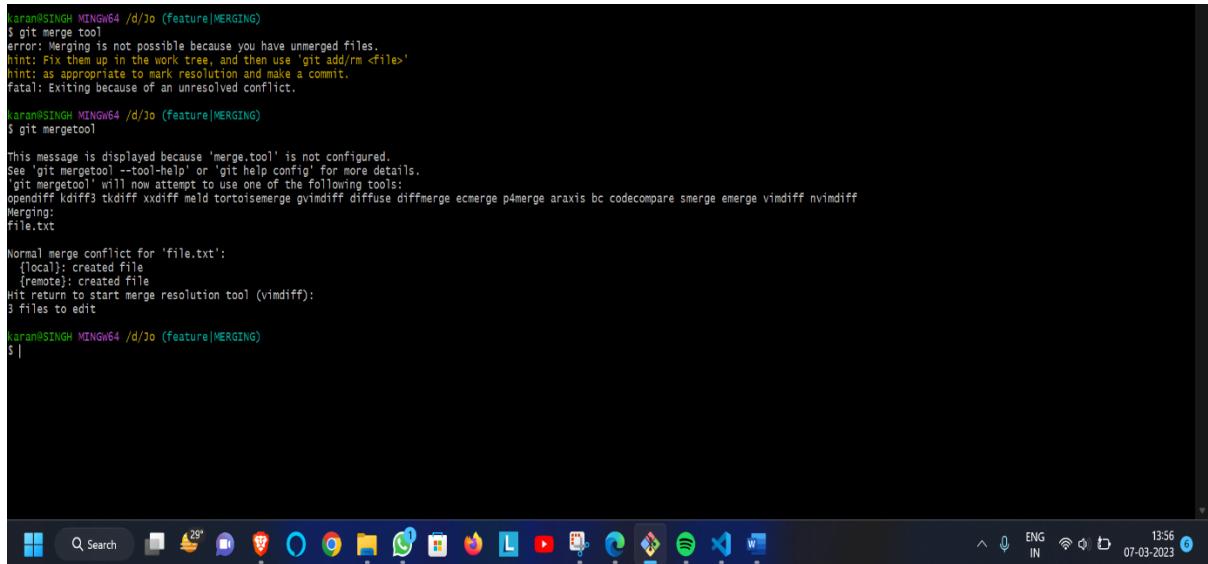
```
karan@KARAN-OptiPlex-5090:~/Desktop$ git merge master
Auto-merging file.txt
CONFLICT (add/add): Merge conflict in file.txt
Automatic merge failed; fix conflicts and then commit the result.
```

Finally, the user attempts to commit the merge:

```
karan@KARAN-OptiPlex-5090:~/Desktop$ git commit -m "Merged master into feature"
[feature 1e0a2d] Merged master into feature
 1 file changed, 1 insertion(+)
 create mode 100644 file.txt
```

3. Use Command “git mergetool” to solve the conflict.

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



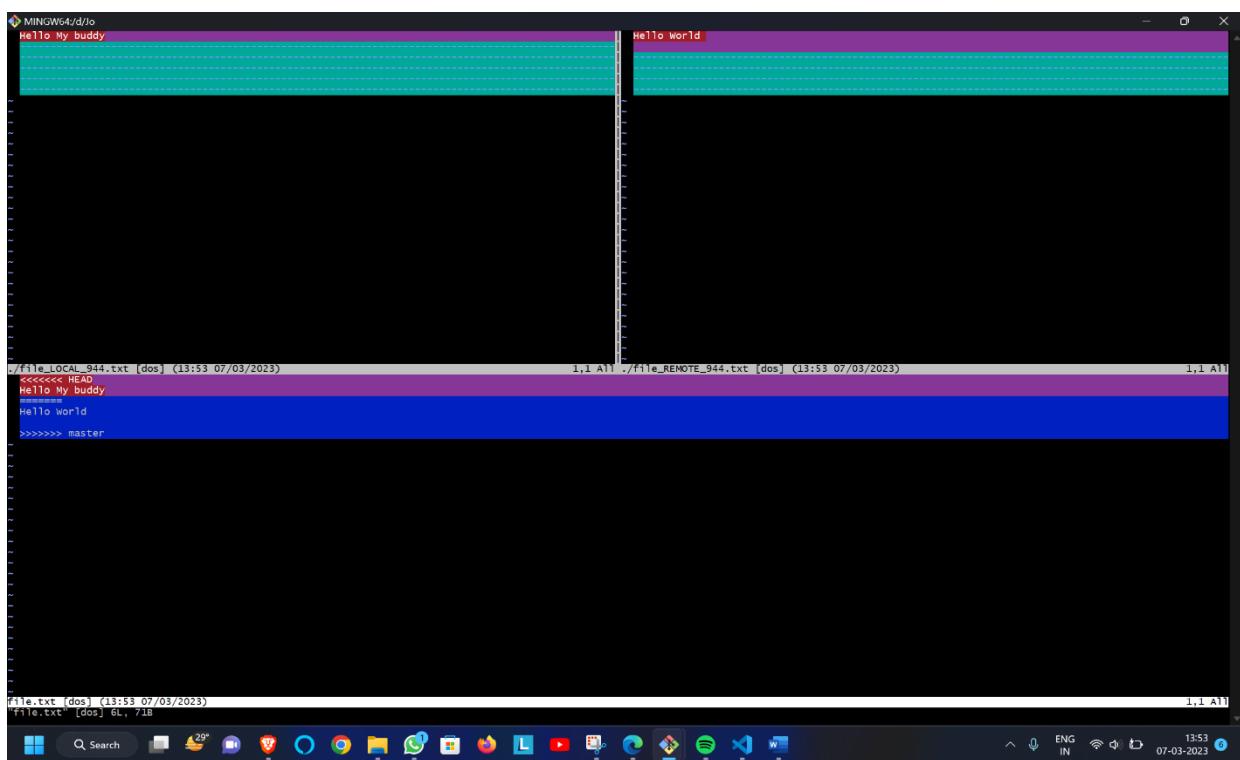
```
karan@SINGH MINGW64 /d/Jo (feature|MERGING)
$ git merge tool
error: Merging is not possible because you have unmerged files.
hint: Fix them up in the work tree, and then use 'git add/rm <file>'.
hint: as appropriate to mark resolution and make a commit.
fatal: Exiting because of an unresolved conflict.

karan@SINGH MINGW64 /d/Jo (feature|MERGING)
$ git mergetool
This message is displayed because 'merge.tool' is not configured.
See 'git mergetool --tool-help' or 'git help config' for more details.
'git mergetool' will now attempt to use one of the following tools:
opendiff kdiff3 tkdiff xxdiff meld tortoisemerge gvdifff diffuse diffmerge ecmerge araxis bc codecompare smerge emerge vimdiff nvimdiff
Merging:
file.txt

Normal merge conflict for 'file.txt':
  (local): created file
  (remote): created file
Hit return to start merge resolution tool (vimdiff):
3 files to edit

karan@SINGH MINGW64 /d/Jo (feature|MERGING)
$ |
```

4. Press “I” to insert, after insertion. Press “: wq”. The merge conflict is solved and feature branch is merged to master branch.



```
MINGW64/d/Jo
Hello My buddy
Hello world

<<<<< HEAD
Hello My buddy
=====
Hello world
>>>>> master

file.LOCAL_944.txt [dos] (13:53 07/03/2023) 1,1 All .file.REMOTE_944.txt [dos] (13:53 07/03/2023) 1,1 All
git merge --no-commit
file.txt [dos] (13:53 07/03/2023) 1,1 All
file.txt [dos] 6L, 71B
```

# Experiment 9

**Aim:** Reset and Revert



## Theory:

Git-revert – Revert some existing commits.

A reset is an operation that takes a specified commit and resets the "three trees" to match the state of the repository at that specified commit. A reset can be invoked in three different modes which correspond to the three trees. In reset, rest of the commits wash out after the mentioned commit. This is a limitation of reset command that we cannot have any random access.

A revert is an operation that takes a specified commit and creates a new commit which inverses the specified commit. git revert can only be run at a commit level scope and has no file level functionality.

These two features justify the Version- controlled feature of the git as we can rollback to any version at any time.

## PROCEDURE:

Firstly, prepare a log of multiple commits to make the reset and revert command function.

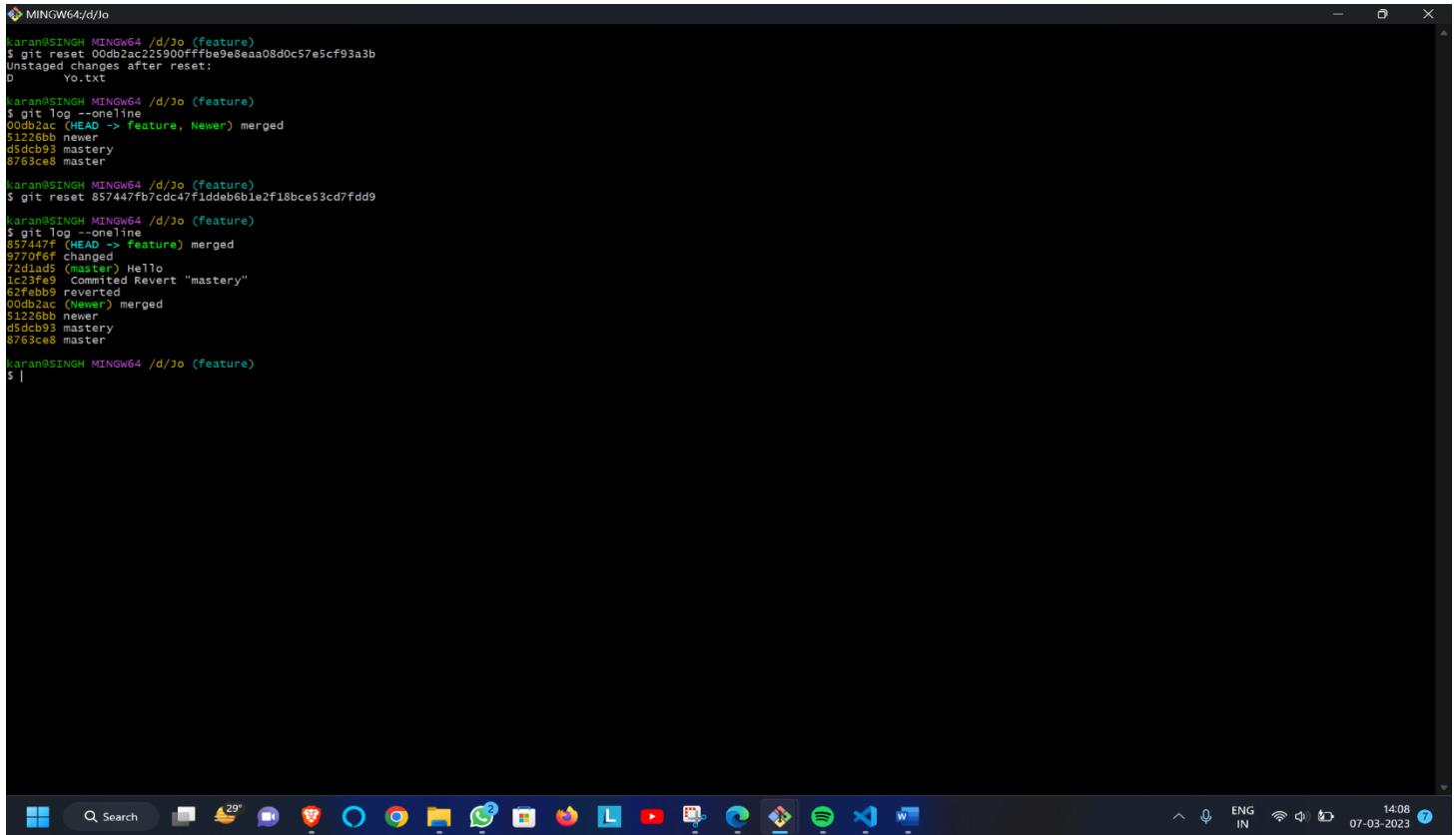
**Reset:** **reset** is the command we use when we want to move the repository back to a previous **commit**, discarding any changes made after that **commit**.

- 1) Create few files, stage them and commit.
- 2) Check the git log
- 3) Pick any commit where you want the repository to rollback. Copy its checksum and paste it in the **\$ git reset checksum** command.

```
MINGW64/d/Jo (feature)
$ git log --oneline
KaranSINGH MINGW64 /d/Jo (feature)
$ git reset 00db2ac2590fffbbe9eaa08d0c57e5cf93a3b
Unstaged changes after reset:
0 Yo.txt
KaranSINGH MINGW64 /d/Jo (feature)
$ git log --oneline
KaranSINGH -> feature, Newer merged
51226bb newer
d5dcbb3 master
8763c68 master
$ |
```

The head is now pointing the commit whose checksum we have provided that means the commits that followed vanished.

- 4) If you want undo this change, you copy the checksum of the commit you want back and run the same command again.



A screenshot of a Windows desktop environment. In the center is a terminal window titled 'MINGW64/d/Jo' with the following text:

```
karan@SINGH MINGW64 /d/Jo (feature)
$ git reset 00db2ac225900fffbbe9e8ea08d0c57e5cf93a3b
Unstaged changes after reset:
  Yo.txt

karan@SINGH MINGW64 /d/Jo (feature)
$ git log --oneline
00db2ac (HEAD --> feature) merged
51226bb newer
d5dcbb9 mastery
8763ce8 master

karan@SINGH MINGW64 /d/Jo (feature)
$ git reset 857447fb7cdc47f1ddeb6bbe2f18bce53cd7fd9
karan@SINGH MINGW64 /d/Jo (feature)
$ git log --oneline
857447f (HEAD --> feature) merged
9770ef6 changed
72d1ad5 (master) Hello
1c23fe9 Committed Revert "mastery"
847447f reverted
00db2ac (feature) merged
51226bb newer
d5dcbb9 mastery
8763ce8 master

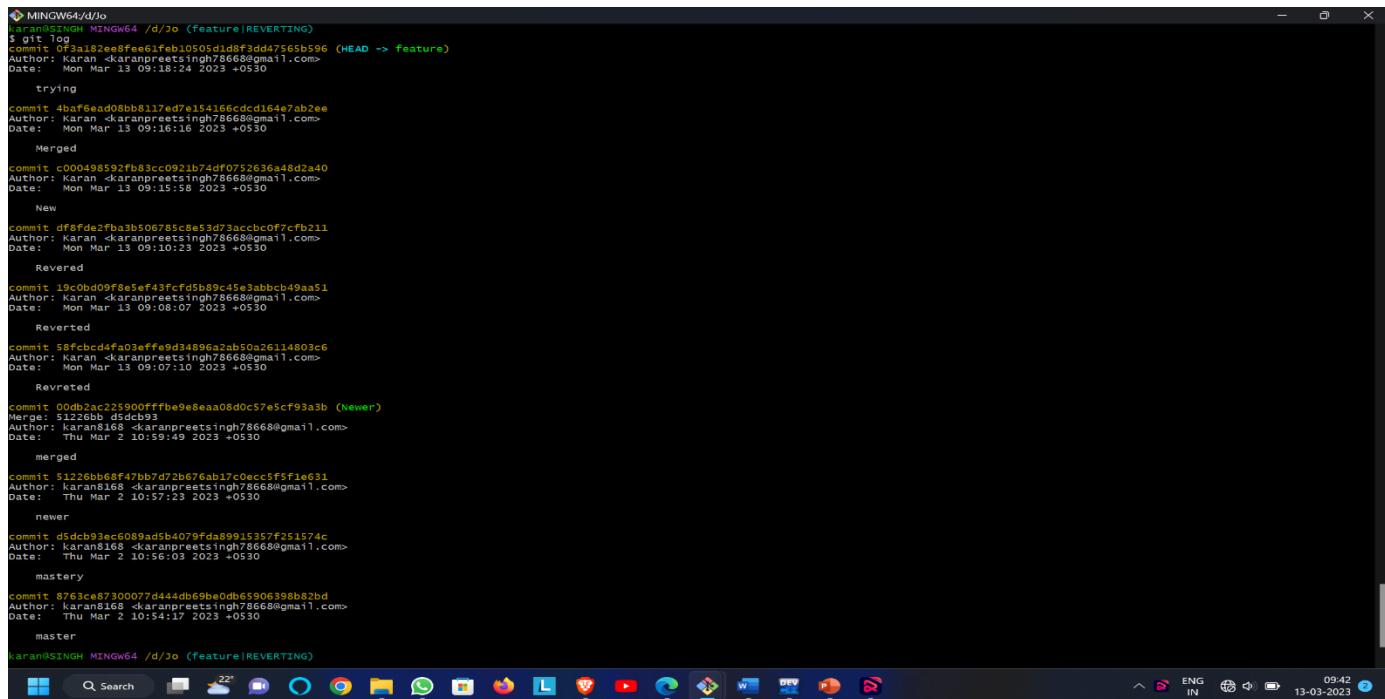
karan@SINGH MINGW64 /d/Jo (feature)
$ |
```

The taskbar at the bottom of the screen shows various pinned application icons, including Microsoft Edge, File Explorer, Task View, and others. The system tray on the right displays the date (07-03-2023), time (14:08), battery level (29%), and network status (ENG IN).

## Revert:

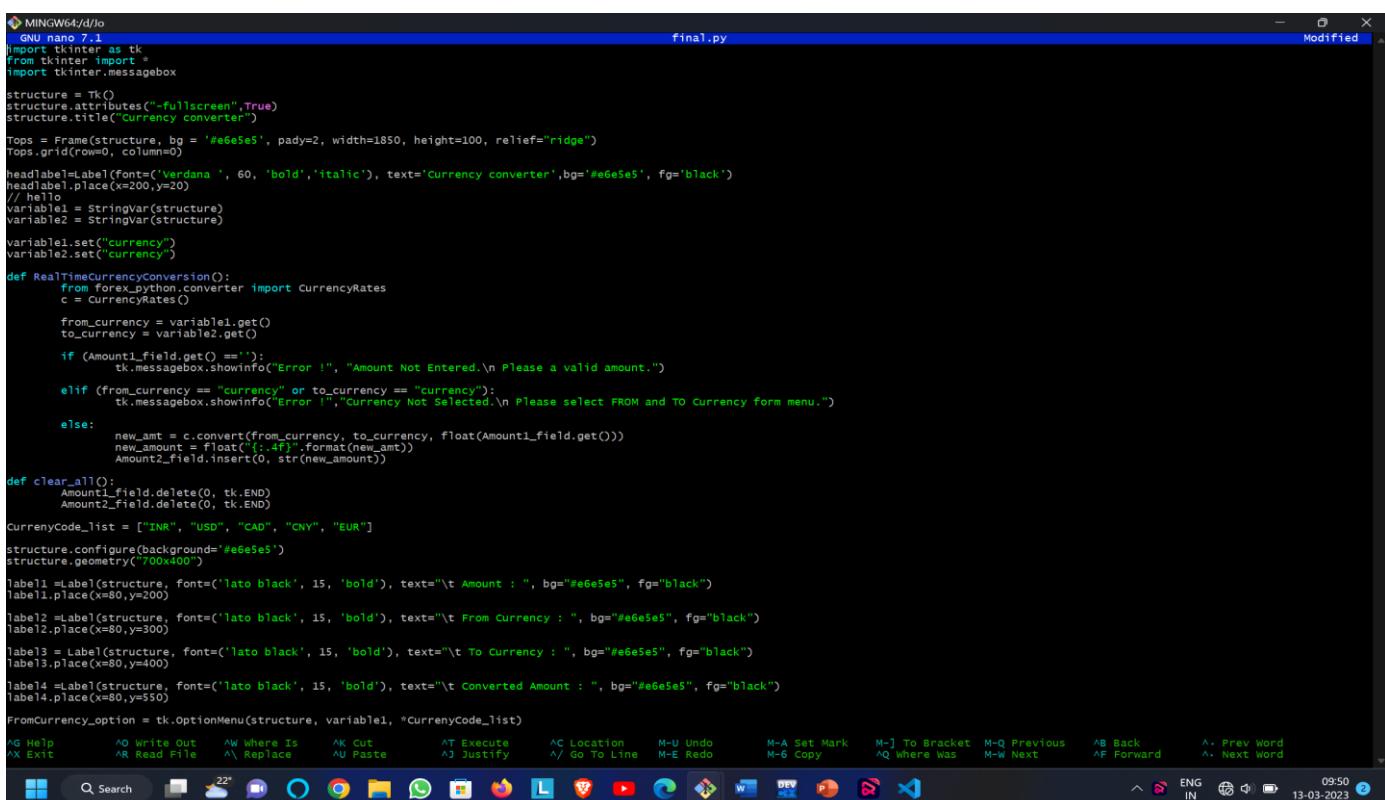
Follow these steps to revert any change:

- 1) Pick the change where you want the project to revert back.  
Copy its checksum and paste it in the revert command.



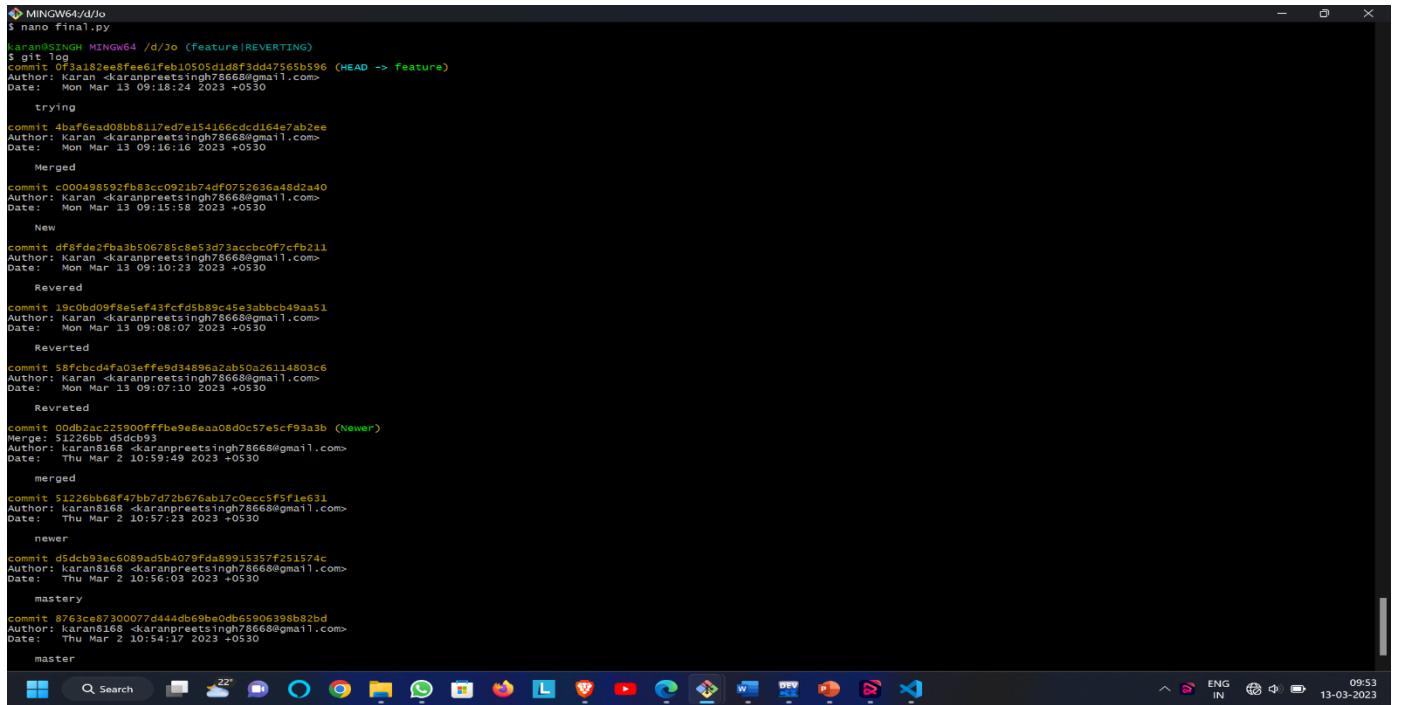
```
MINGW64/d/jo
git push origin feature |REVERTING
$ git log
commit 0f3a18zee8fee6feb10505d1df87565b596 (HEAD -> feature)
Author: Karan <karanpreetsingh7866@gmail.com>
Date: Mon Mar 13 09:18:24 2023 +0530
    trying
commit 4bafe6ad08bb8117ed7e154166cd164e7ab2ee
Author: Karan <karanpreetsingh7866@gmail.com>
Date: Mon Mar 13 09:16:16 2023 +0530
    Merged
commit c00049592fb03cc92ab74df07f5e36a48d2a40
Author: Karan <karanpreetsingh7866@gmail.com>
Date: Mon Mar 13 09:15:58 2023 +0530
    New
commit df8fd2efba3b506785c8e53d73acccbf7cfb211
Author: Karan <karanpreetsingh7866@gmail.com>
Date: Mon Mar 13 09:10:23 2023 +0530
    Reverted
commit 19c0bd09f8a5ef43fcfd5dbb945eabbcb49a51
Author: Karan <karanpreetsingh7866@gmail.com>
Date: Mon Mar 13 09:08:07 2023 +0530
    Reverted
commit 58fcfcda4fa03ffe9d34896a2b50a26114803c6
Author: Karan <karanpreetsingh7866@gmail.com>
Date: Mon Mar 13 09:07:10 2023 +0530
    Reverted
commit 00db2ac225900ffbe9e8ea08d0c57e5cf93a3b (Newer)
Merge: 51226b6 d5dc93
Author: karan8168 <karanpreetsingh7866@gmail.com>
Date: Thu Mar 2 10:59:49 2023 +0530
    merged
commit 51226b6ff47bb7d72b676ab17c0ecc5f5f1e631
Author: karan8168 <karanpreetsingh7866@gmail.com>
Date: Thu Mar 2 10:57:23 2023 +0530
    newer
commit d5dc93ec6089ad5b4079fd89915357f251574c
Author: karan8168 <karanpreetsingh7866@gmail.com>
Date: Thu Mar 2 10:56:03 2023 +0530
    mystery
commit 8763ce87300077d444db9b9ebdb65906398b82bd
Author: karan8168 <karanpreetsingh7866@gmail.com>
Date: Thu Mar 2 10:54:17 2023 +0530
    master
Karan@SINGH MINGW64 /d/jo (feature|REVERTING)
```

- 2) A window will appear. Press 'I' and write the statement you want to be displayed for reverting the change.



```
MINGW64/d/jo
GNU nano 7.1
from tkinter import *
import tkinter.messagebox
structure = Tk()
structure.attributes("-fullscreen",True)
structure.title("Currency converter")
Tops = Frame(structure, bg="#e6e5e5", pady=2, width=1850, height=100, relief="ridge")
Tops.grid(row=0, column=0)
headLabel=Label(font=("verdana ", 60, "bold", "italic"), text='Currency converter',bg="#e6e5e5", fg="black")
headLabel.place(x=200,y=20)
variable1 = StringVar(structure)
variable2 = StringVar(structure)
variable1.set("Currency")
variable2.set("Currency")
def RealTimeCurrencyConversion():
    from forex_python.converter import CurrencyRates
    c = CurrencyRates()
    from_currency = variable1.get()
    to_currency = variable2.get()
    if (Amount1_field.get() == ""):
        tk.messagebox.showinfo("Error !", "Amount Not Entered.\n Please a valid amount.")
    elif (from_currency == "Currency" or to_currency == "Currency"):
        tk.messagebox.showinfo("Error !", "Currency Not Selected.\n Please select FROM and TO currency form menu.")
    else:
        new_amt = c.convert(from_currency, to_currency, float(Amount1_field.get()))
        new_amount = float('%.4f' %new_amt)
        Amount2_field.insert(0, str(new_amount))
def clear_all():
    Amount1_field.delete(0, tk.END)
    Amount2_field.delete(0, tk.END)
CurrencyCode_list = ["INR", "USD", "CAD", "CNY", "EUR"]
structure.configure(background="#e6e5e5")
structure.geometry("700x400")
label1=Label(structure, font="lato black", 15, "bold"), text="\t Amount : ", bg="#e6e5e5", fg="black")
label1.place(x=80,y=200)
label2=Label(structure, font="lato black", 15, "bold"), text="\t From Currency : ", bg="#e6e5e5", fg="black")
label2.place(x=80,y=300)
label3=Label(structure, font="lato black", 15, "bold"), text="\t To Currency : ", bg="#e6e5e5", fg="black")
label3.place(x=80,y=400)
label4=Label(structure, font="lato black", 15, "bold"), text="\t Converted Amount : ", bg="#e6e5e5", fg="black")
label4.place(x=80,y=500)
FromCurrency_option = tk.OptionMenu(structure, variable1, *CurrencyCode_list)
FromCurrency_option.place(x=150,y=150)
ToCurrency_option = tk.OptionMenu(structure, variable2, *CurrencyCode_list)
ToCurrency_option.place(x=350,y=150)
Amount1_field=Entry(structure, width=16, bg="white", fg="black", borderwidth=2)
Amount1_field.place(x=250,y=200)
Amount2_field=Entry(structure, width=16, bg="white", fg="black", borderwidth=2)
Amount2_field.place(x=250,y=300)
structure.mainloop()
```

- 3) After completing press ‘esc’ and write: wq in the terminal
- 4) Check the git log and you will find another commit is added without affecting the rest commits.



```

MINGW64/d/jo
$ nano final.py

karan@SINGH MINGW64 /d/jo (feature |REVERTING)
git: 'final' does not exist.
commit 0f5a182ee8fee61feb10505d1d8f3dd47565b596 (HEAD -> feature)
Author: Karan <karanpreetsingh78668@gmail.com>
Date: Mon Mar 13 09:18:24 2023 +0530

    trying
commit 4baefead08bb3b117ed7e154165c1c1104a7ab2ee
Author: Karan <karanpreetsingh78668@gmail.com>
Date: Mon Mar 13 09:16:16 2023 +0530

    Merged
commit c000498592fb83cc0921b74df0752636a48d2a40
Author: Karan <karanpreetsingh78668@gmail.com>
Date: Mon Mar 13 09:15:58 2023 +0530

    New
commit df8fde2fba3b506785c8e53d73accc0f7cfb211
Author: Karan <karanpreetsingh78668@gmail.com>
Date: Mon Mar 13 09:10:23 2023 +0530

    Reversed
commit 19c0bd09f8a5ef43fcfd5b89c45e3abbcb49aa51
Author: Karan <karanpreetsingh78668@gmail.com>
Date: Mon Mar 13 09:08:07 2023 +0530

    Reverted
commit 58fcbe4fa3effe9d34896a2ab50126114803c6
Author: Karan <karanpreetsingh78668@gmail.com>
Date: Mon Mar 13 09:07:10 2023 +0530

    Revrevert
commit 00dbzac225900fffbe9e8ea08d0c57e5cf93a3b (Newer)
Merge: 5126bbb d5dc93
Author: karans168 <karanpreetsingh78668@gmail.com>
Date: Thu Mar 2 10:59:49 2023 +0530

    merged
commit 5126bbb68f47bb7d7b676ab17c0ecc5f5f1e631
Author: karans168 <karanpreetsingh78668@gmail.com>
Date: Thu Mar 2 10:57:23 2023 +0530

    newer
commit d5dc93ec6089ad5b4079fda89915357f251574c
Author: karans168 <karanpreetsingh78668@gmail.com>
Date: Thu Mar 2 10:56:03 2023 +0530

    mastery
commit 8763ca8720077d444db69be0b5906398b82bd
Author: karans168 <karanpreetsingh78668@gmail.com>
Date: Thu Mar 2 10:54:17 2023 +0530

    master

```

- 5) The change associated to the reverted commit has disappeared.

A Project Report  
on  
**“Task 2”**  
with  
**Source Code Management**  
(22CS003)

**Submitted by:**  
Kartavya Tomar  
Roll No. 2210990484  
Submission of Task 2

**Submitted To:**  
Dr. Chetna Kaushal  
Assistant Professor  
Department of Computer Science  
& Engineering  
Chitkara University Institute  
of Engineering and  
Technology Rajpura, Punjab

Team Member 1 Name: Karanpreet  
Singh

Roll No. 2210990484

Team Member 2 Name: Karishma  
Dhawan

Roll No. 2210990483

Team Member 3 Name: Kartavya  
Tomar

Roll No. 2210990484

Institute/School: - **Chitkara University Institute of Engineering and Technology**

Department Name: - **Department of Computer Science & Engineering**

Program Name: - **Bachelor of Engineering (B.E.), Computer Science& Engineering**

Course Name: - **Source Code Management** Session: **2022-23**

Course Code: - **22CS003** Batch: **2022**

Vertical Name: - **First Year** Group No: - **G24-B**



# **INDEX**

S. NO	Experiment Name	Page No.
1.	Introduction	34
2.	Problem statement	37
3.	Solution	38
4.	Objective	39
5.	Create a distributed Repository and add members in project team	40
6.	Open and close a pull request	46
7.	Create a pull request on a team members repo and close pull requests generated by team members on own Repo as a maintainer	50
8.	Publish and print network graphs	53

## Introduction

### What is GIT and why is it used?

Git is a version control system that is widely used in the programming world. It is used for tracking changes in the source code during software development. It was developed in 2005 by Linus Torvalds, the creator of the Linux operating system kernel.

Git is a speedy and efficient distributed [VCS](#) tool that can handle projects of any size, from small to very large ones. Git provides cheap local branching, convenient staging areas, and multiple workflows. It is free, open-source software that lowers the cost because developers can use Git without paying money. It provides support for non-linear development. Git enables multiple developers or teams to work separately without having an impact on the work of others.

Git is an example of a distributed version control system (DVCS) (hence Distributed Version Control System).



### What is GITHUB?

It is the world's largest open-source software developer community platform where the users upload their projects using the software Git.



### What is the difference between GIT and GITHUB?

## What is Repository?

A repository is a directory or storage space where your projects can live. Sometimes GitHub users shorten this to “repo.” It can be local to a folder on your computer, or it can be a storage space on GitHub or another online host. You can keep code files, text files, image files, you name it, inside a repository.

## What is Version Control System (VCS)?

A version control system is a tool that helps you manage “versions” of your code or changes to your code while working with a team over remote distances. Version control keeps track of every modification in a special kind of database that is accessible to the version control software. Version control software (VCS) helps you revert back to an older version just in case a bug or issue is introduced to the system or fixing a mistake without disrupting the work of other team members.

## Types of VCS

1. Local Version Control System
  2. Centralized Version Control System
  3. Distributed Version Control System
- I. **Local Version Control System:** Local Version Control System is located in your local machine. If the local machine crashes, it would not be possible to retrieve the files, and all the information will be lost. If anything happens to a single version, all the versions made after that will be lost.
- AI. **Centralized Version Control System:** In the Centralized Version Control Systems, there will be a single central server that contains all the files related to the project, and many collaborators checkout files from this single server (you will only have a working copy). The problem with the Centralized
-

Version Control Systems is if the central server crashes, almost everything related to the project will be lost.

- BI. **Distributed Version Control System:** In a distributed version control system, there will be one or more servers and many collaborators similar to the centralized system. But the difference is, not only do they check out the latest version, but each collaborator will have an exact copy of the main repository on their local machines. Each user has their own repository and a working copy. This is very useful because even if the server crashes, we would not lose everything as several copies are residing in several other computers.
-

# **Problem Statement**

At present there are various programs to convert one currency into another. This may result into a code that may be absolute and does not adapt to the changing environments. In order to deal with such a issue a special program is needed that adapts to the changing environment and provide with the absolute results.

# **Solution**

Use of various libraries like forex python to calculate one currency to another is used. This helps to manage currency conversion quite accurately. Not only does this enable accurate conversion but it also enables various buisness to make fast and reliable calculations in a short amount of time

# Objective

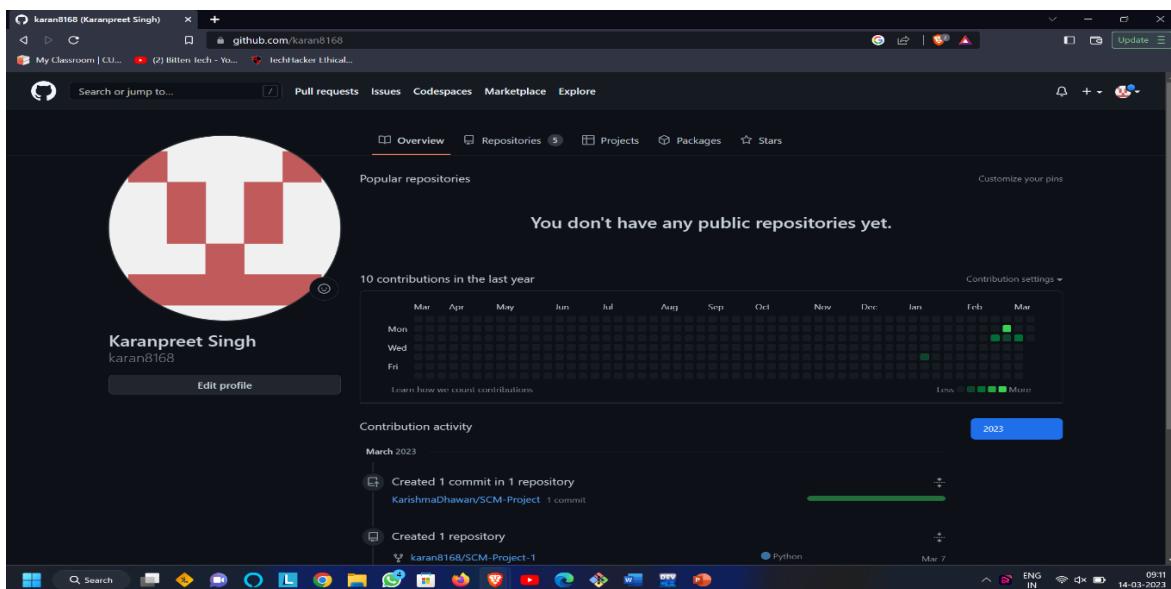
The objective of this project is to associate programming with git because:

1. This is required because the collaboration makes the team work easy.
2. The code becomes manageable and we can build a clean repository.
3. Tracking and resolving of the errors is quite feasible in this process.
4. Moreover, we can make our locally available projects, globally available.

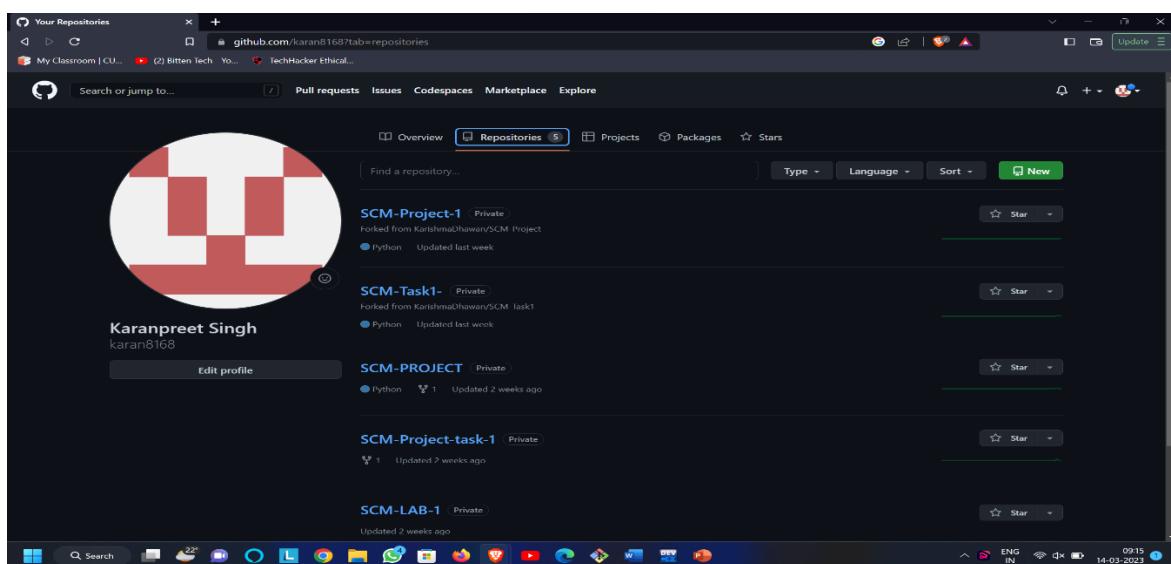
## Experiment No. 01

Aim: Create a distributed Repository and add members in project team

- 1) Login to your GitHub account and you will land on the homepage as shown below. Click on Repositories option in the menu bar.

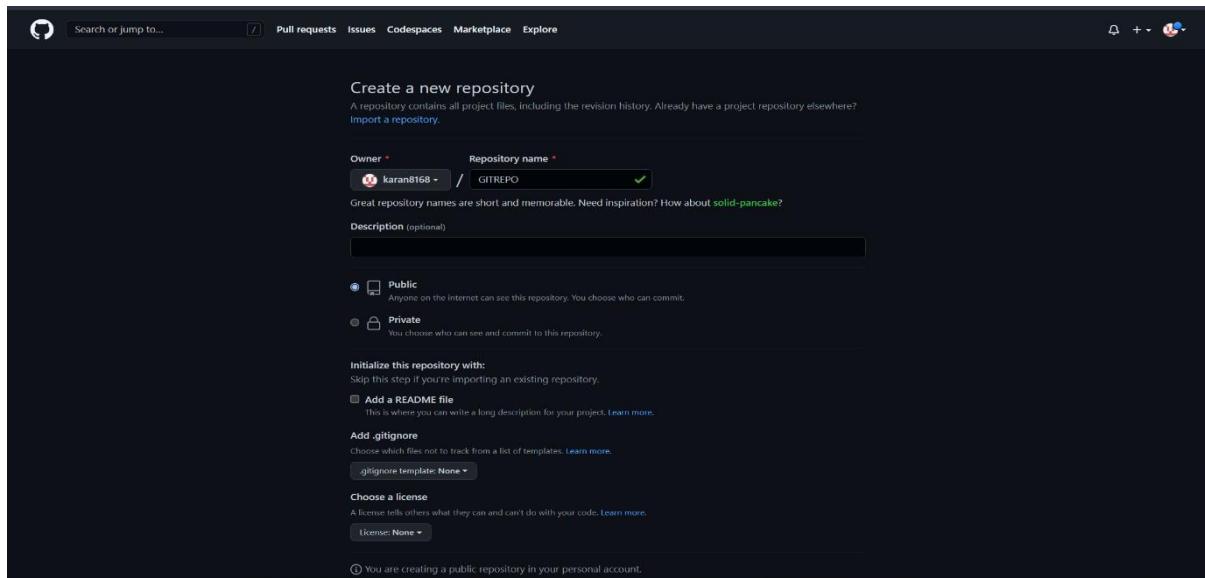


- 2) Click on the ‘New’ button in the top right corner.

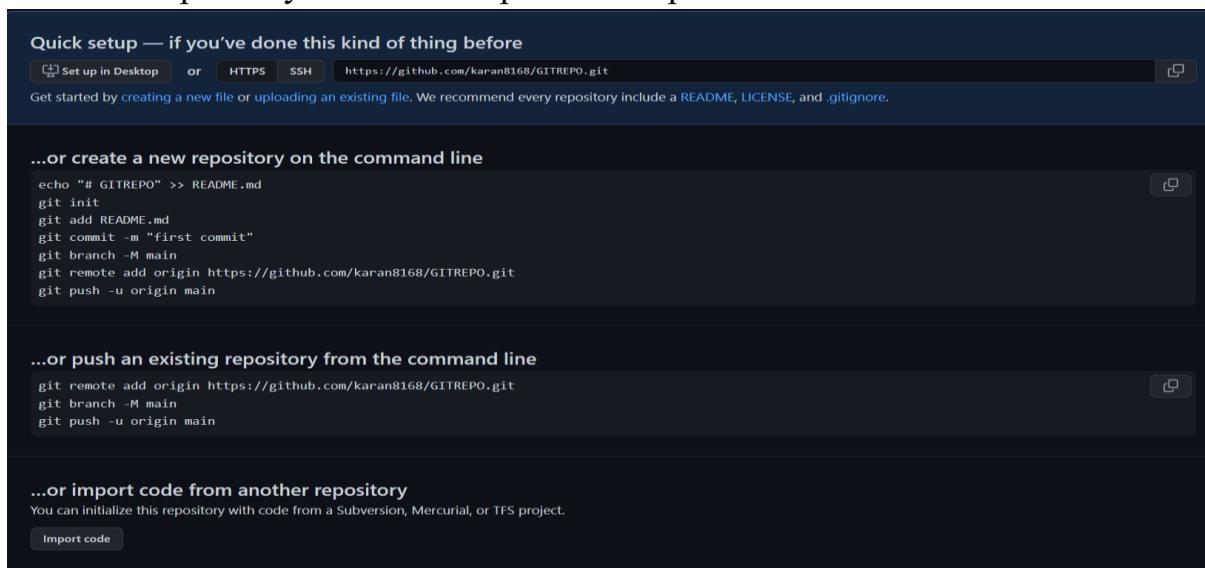


- 3) Enter the Repository name and add the description of the repository.

4) Select if you want the repository to be public or private.

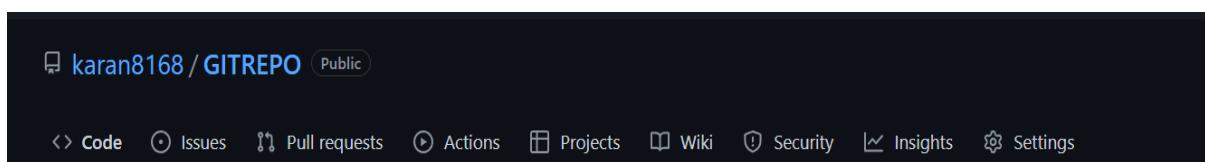


5) If you want to import code from an existing repository select the import code option.

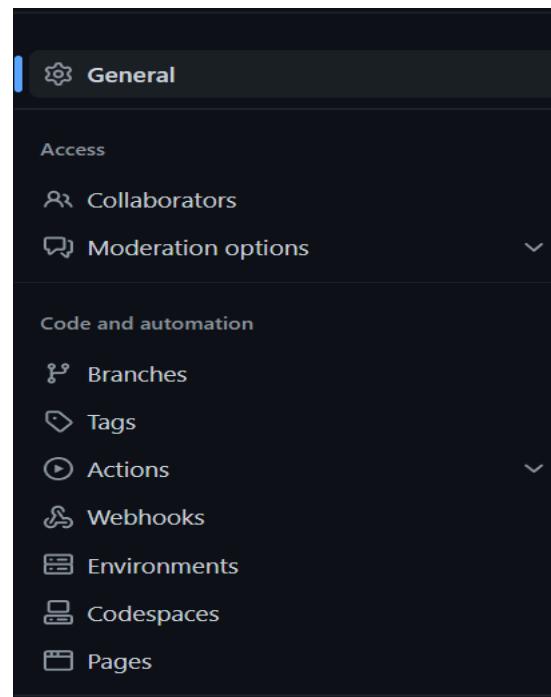


6) Now, you have created your repository successfully.

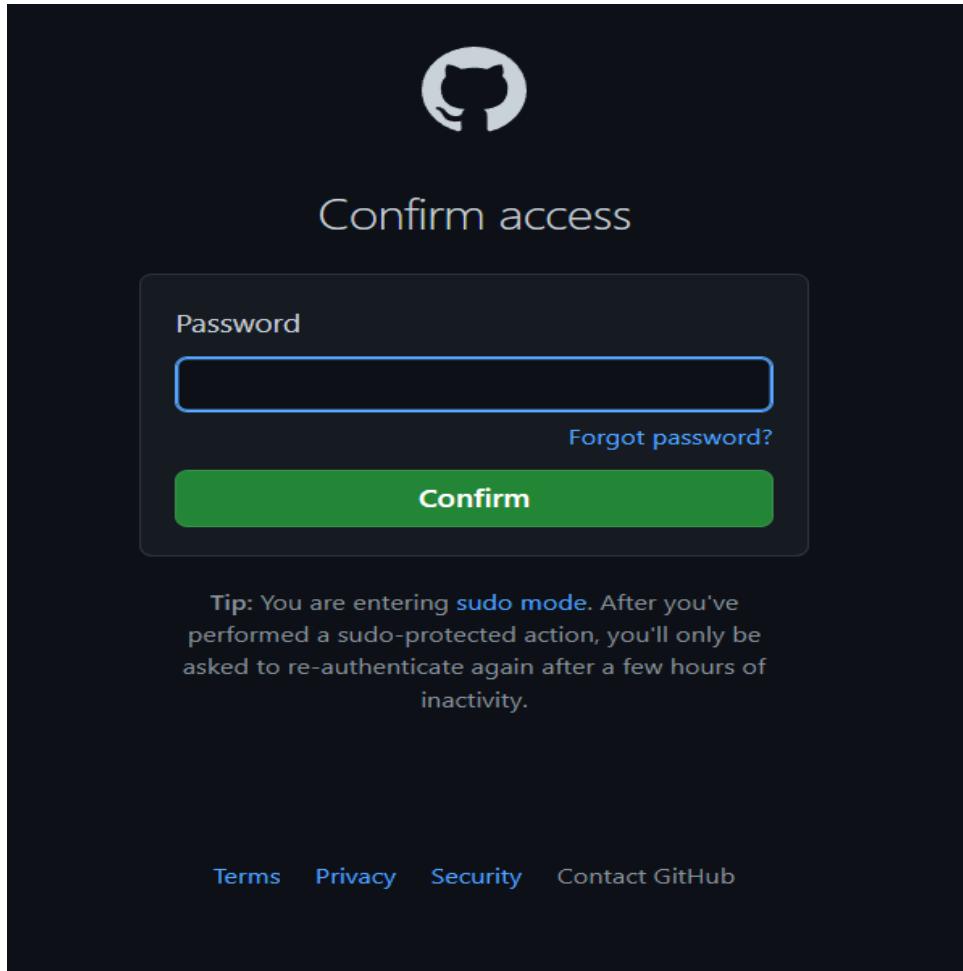
7) To add members to your repository, open your repository and select settings option in the navigation bar.



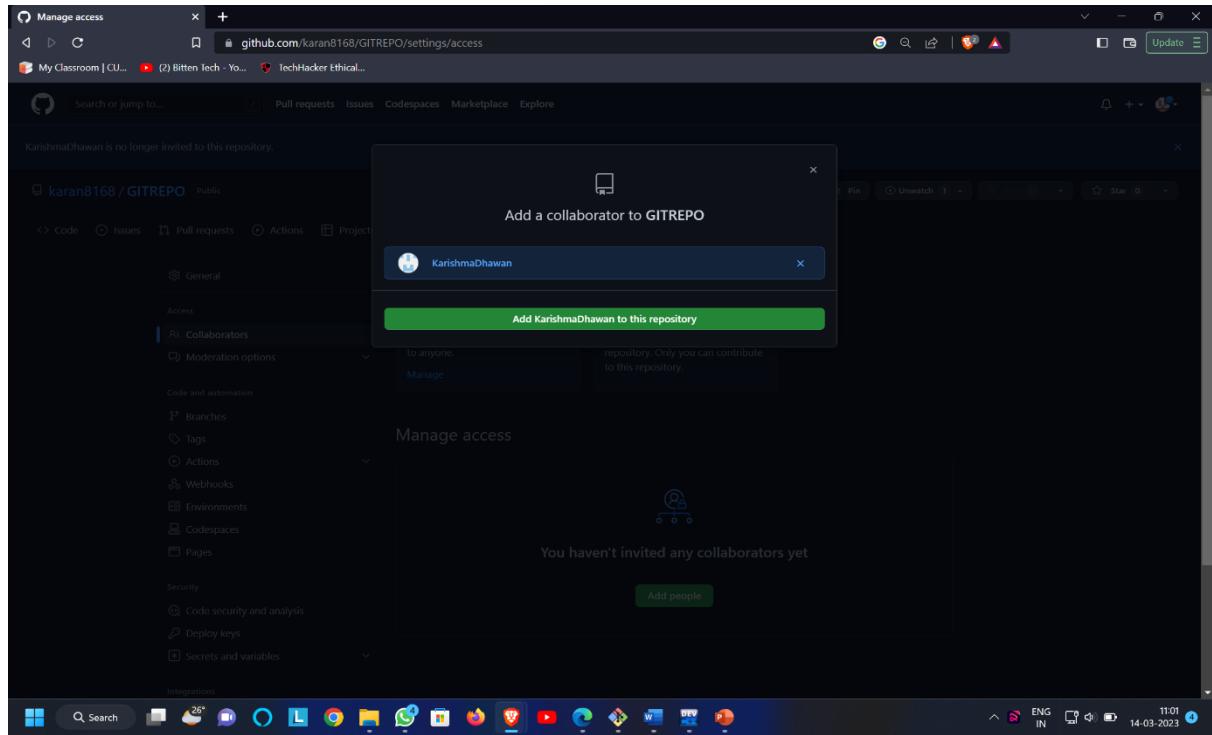
8) Click on Collaborators option under the access tab.



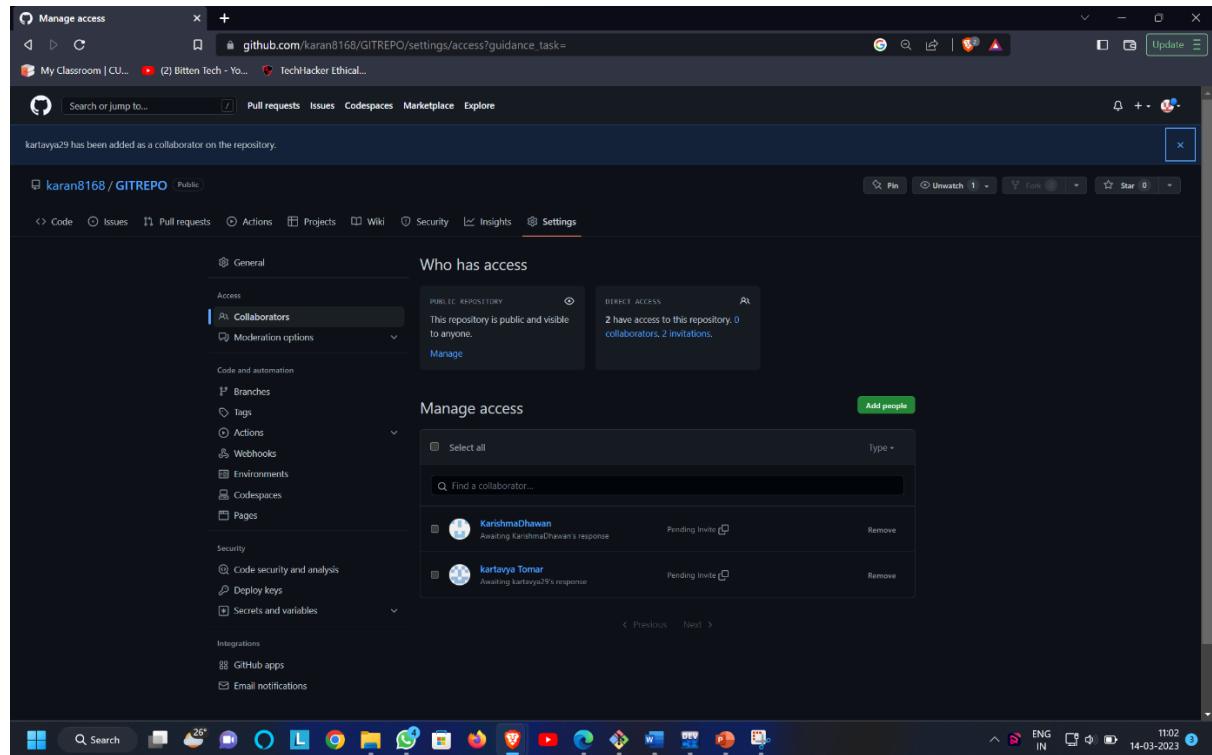
9) After clicking on collaborators GitHub asks you to enter your password to confirm the access to the repository.



- 10) After entering the password, you can manage access and add/remove team members to your project.
- 11) To add members, click on the add people option and search the id of your respective team member.



- 12) To remove any member, click on remove option available in the last column of member's respective row.



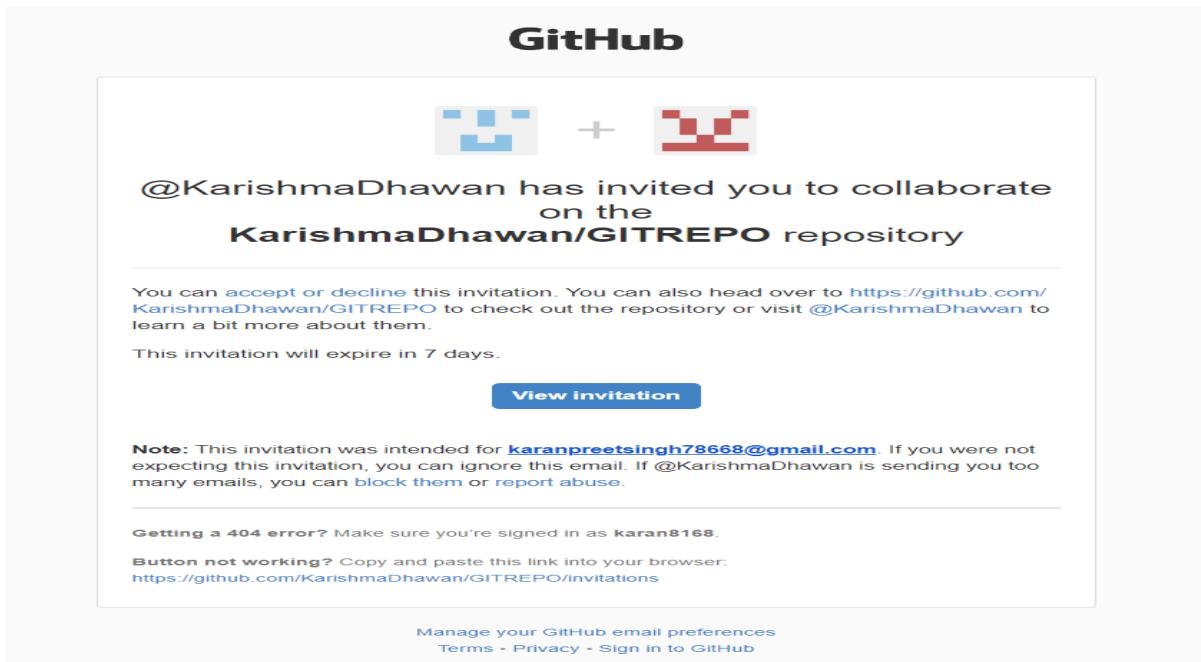
- 13) To accept the invitation from your team member, open your mail registered with GitHub.

The screenshot shows an email from GitHub. At the top is the GitHub logo. Below it are two small square icons: one blue with a white grid pattern and one red with a white 'W' shape. The main text reads: '@KarishmaDhawan has invited you to collaborate on the KarishmaDhawan/GITREPO repository'. A horizontal line follows. Below it, text says: 'You can [accept](#) or [decline](#) this invitation. You can also head over to <https://github.com/KarishmaDhawan/GITREPO> to check out the repository or visit [@KarishmaDhawan](#) to learn a bit more about them.' Another line follows. Then it says: 'This invitation will expire in 7 days.' A blue button labeled 'View invitation' is centered below this. A note below the button states: 'Note: This invitation was intended for [karanpreetsingh78668@gmail.com](mailto:karanpreetsingh78668@gmail.com). If you were not expecting this invitation, you can ignore this email. If @KarishmaDhawan is sending you too many emails, you can [block them](#) or [report abuse](#).' A horizontal line follows. Below this, in smaller text, are links: 'Getting a 404 error? Make sure you're signed in as [karan8168](#)'. 'Button not working? Copy and paste this link into your browser: <https://github.com/KarishmaDhawan/GITREPO/invitations>'.

Manage your GitHub email preferences  
[Terms](#) • [Privacy](#) • [Sign in to GitHub](#)

- 14) You will receive an invitation mail from the repository owner. Open the email and click on accept invitation.

- 15) You will be redirected to GitHub where you can either select to accept or decline the invitation.



- 16) You will be shown the option that you are now allowed to push.

- 17) Now all members are ready to contribute to the project.

.gitignore	Adding Code to Github	5 days ago
bg.jpg	Adding Code to Github	5 days ago
contact.css	Addign contact us page to the repo	5 days ago
contact.html	Addign contact us page to the repo	5 days ago
contact_us.html	Addign contact us page to the repo	5 days ago
index.html	Adding Code to Github	5 days ago
list_serv.html	Adding Code to Github	5 days ago
repair.png	Adding Code to Github	5 days ago
rev.css	Adding Code to Github	5 days ago
sample-service-img.png	Adding Code to Github	5 days ago
style.css	Adding Code to Github	5 days ago

Help people interested in this repository understand your project by adding a README. [Add a README](#)

## Experiment No. 02

Aim: Open and Close a Pull Request

- 1) To open a pull request, we first have to make a new branch, by using git branch *branch name* option.

```
himan@DESKTOP-ESG6013 MINGW64 ~/OneDrive/Desktop/scm (master)
$ git clone https://github.com/karan16kkg/Jay-Ho.git
Cloning into 'Jay-Ho'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (6/6), done.

himan@DESKTOP-ESG6013 MINGW64 ~/OneDrive/Desktop/scm (master)
$ |
```

- 2) After making new branch we add a file to the branch or make changes in the existing file.

```
himan@DESKTOP-ESG6013 MINGW64 ~/OneDrive/Desktop/scm (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    Jay-Ho/
nothing added to commit but untracked files present (use "git add" to track)

himan@DESKTOP-ESG6013 MINGW64 ~/OneDrive/Desktop/scm (master)
$ git add --all
warning: adding embedded git repository: Jay-Ho
hint: You've added another git repository inside your current repository.
hint: Clones of the outer repository will not contain the contents of
hint: the embedded repository and will not know how to obtain it.
hint: If you meant to add a submodule, use:
hint:   git submodule add <url> Jay-Ho
hint:
hint: If you added this path by mistake, you can remove it from the
hint: index with:
hint:   git rm --cached Jay-Ho
hint:
hint: See "git help submodule" for more information.

himan@DESKTOP-ESG6013 MINGW64 ~/OneDrive/Desktop/scm (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   Jay-Ho

himan@DESKTOP-ESG6013 MINGW64 ~/OneDrive/Desktop/scm (master)
$ git commit -m "commit1"
[master 1193c92] commit1
 1 file changed, 1 insertion(+)
 create mode 160000 Jay-Ho

himan@DESKTOP-ESG6013 MINGW64 ~/OneDrive/Desktop/scm (master)
$ git status
On branch master
nothing to commit, working tree clean

himan@DESKTOP-ESG6013 MINGW64 ~/OneDrive/Desktop/scm (master)
$
```

- 3) Add and commit the changes to the local repository.
- 4) Use git push origin *Branch name* option to push the new branch to the main repository.

```

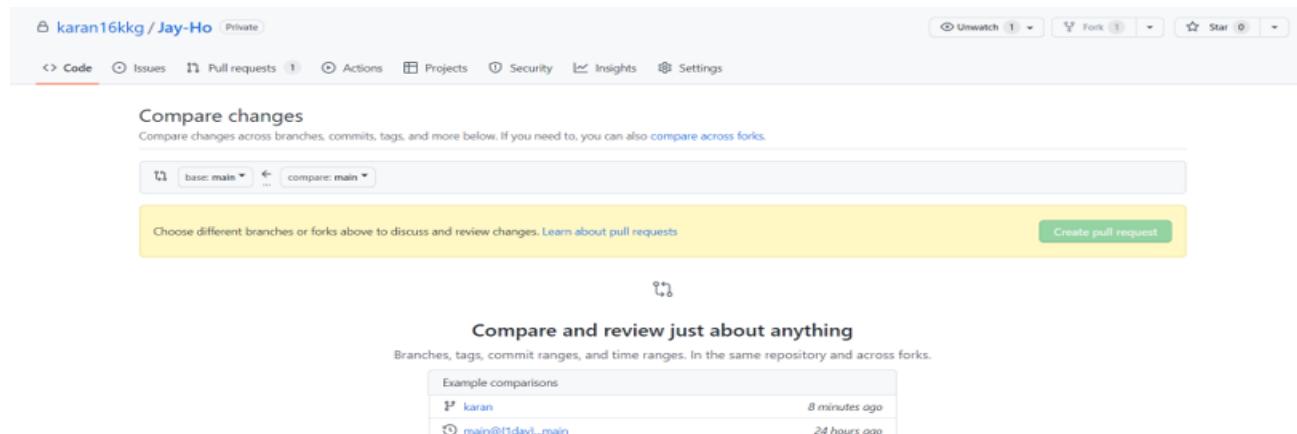
himan@DESKTOP-ESG6013 MINGW64 ~/OneDrive/Desktop/scm/Jay-Ho (karan16kkg)
$ cd karan16kkg / Jay-Ho
bash: cd: too many arguments
himan@DESKTOP-ESG6013 MINGW64 ~/OneDrive/Desktop/scm/Jay-Ho (karan16kkg)
$ ls
Jay-Ho/ Python Python.py
himan@DESKTOP-ESG6013 MINGW64 ~/OneDrive/Desktop/scm/Jay-Ho (karan16kkg)
$ git branch karan
himan@DESKTOP-ESG6013 MINGW64 ~/OneDrive/Desktop/scm/Jay-Ho (karan16kkg)
$ git checkout karan
Switched to branch 'karan'
A   Jay-Ho

himan@DESKTOP-ESG6013 MINGW64 ~/OneDrive/Desktop/scm/Jay-Ho (karan)
$ git status
On branch karan
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   Jay-Ho

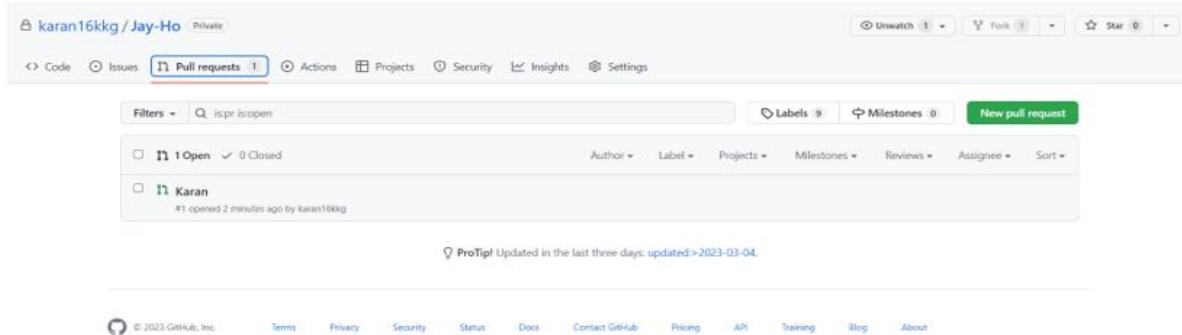
himan@DESKTOP-ESG6013 MINGW64 ~/OneDrive/Desktop/scm/Jay-Ho (karan)
$ git add .
himan@DESKTOP-ESG6013 MINGW64 ~/OneDrive/Desktop/scm/Jay-Ho (karan)
$ git commit -m "Addign contact us page to the repo"
[karan 3dee412] Addign contact us page to the repo
 1 file changed, 1 insertion(+)
 create mode 160000 Jay-Ho

himan@DESKTOP-ESG6013 MINGW64 ~/OneDrive/Desktop/scm/Jay-Ho (karan)
$ git push origin karan
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 4 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 1.53 KiB | 523.00 KiB/s, done.
Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
remote: Create a pull request for 'karan' on GitHub by visiting:
remote:   https://github.com/karan16kkg/Jay-Ho/pull/new/karan
remote:
To https://github.com/karan16kkg/Jay-Ho.git
 * [new branch]      karan -> karan
himan@DESKTOP-ESG6013 MINGW64 ~/OneDrive/Desktop/scm/Jay-Ho (karan)
$
```

- 5) After pushing new branch GitHub will either automatically ask you to create a pull request or you can create your own pull request.



6) To create your own pull request, click on pull request option.



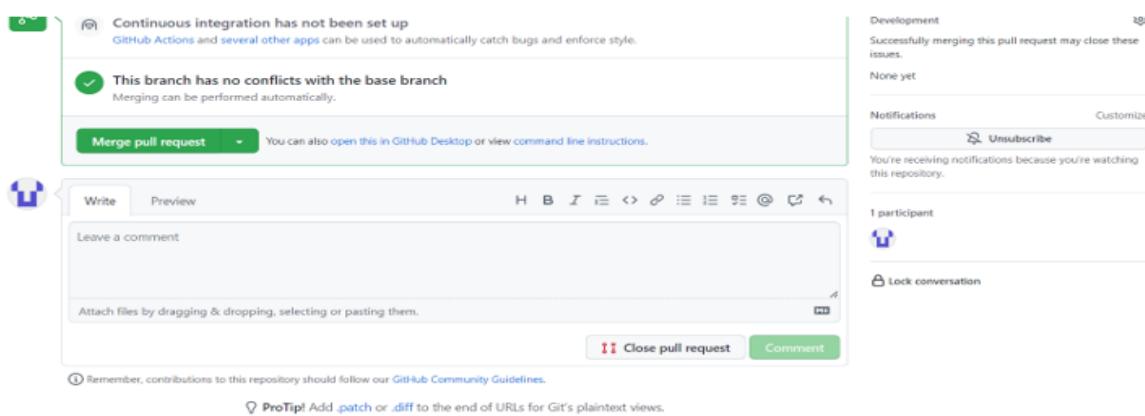
The screenshot shows the GitHub interface for a repository named 'karan16kkg / Jay-Ho'. The 'Pull requests' tab is selected, showing one open pull request. The pull request details are as follows:

- Author: Karan
- Commit message: '#1 opened 2 minutes ago by karan16kkg'
- Status: Open

Below the pull request list, there is a 'ProTip!' message: 'ProTip! Updated in the last three days: updated:>2023-03-04.'

At the bottom of the page, there are various GitHub links: Terms, Privacy, Security, Status, Docs, Contact GitHub, Pricing, API, Training, Help, and About.

- 7) GitHub will detect any conflicts and ask you to enter a description of your pull request.
- 8) After opening a pull request all the team members will be sent the request if they want to merge or close the request.



The screenshot shows a GitHub pull request review interface. The pull request summary includes:

- Continuous integration has not been set up (GitHub Actions and several other apps can be used to automatically catch bugs and enforce style).
- This branch has no conflicts with the base branch (Merging can be performed automatically).

Below the summary, there is a 'Merge pull request' button and a note: 'You can also open this in GitHub Desktop or view command line instructions.'

The main area features a rich text editor with 'Write' and 'Preview' tabs, and a text input field for leaving a comment. A note at the bottom says: 'Remember, contributions to this repository should follow our [GitHub Community Guidelines](#).'

On the right side, there is a sidebar with the following sections:

- Development**: 'Successfully merging this pull request may close these issues.' (None yet)
- Notifications**: 'Customize' and 'Unsubscribe' buttons. A note: 'You're receiving notifications because you're watching this repository.'
- 1 participant**: Shows a user icon.
- Lock conversation**: A lock icon.

At the bottom, there are 'Close pull request' and 'Comment' buttons. A 'ProTip!' note at the bottom says: 'Add .patch or .diff to the end of URLs for Git's plaintext views.'

8) Af

## Experiment No. 03

Aim: Create a pull request on a team member's repo and close pull requests generated by team members on own Repo as a maintainer

To create a pull request on a team member's repository and close requests by any other team members as a maintainer follow the procedure given below:

1. Do the required changes in the repository, add and commit these changes in the local repository in a new branch.

```
Aastha Anand@LAPTOP-G544JMLU MINGW64 ~ (master)
$ mkdir 2110990005-scm
Aastha Anand@LAPTOP-G544JMLU MINGW64 ~ (master)
$ cd 2110990005-scm/
Aastha Anand@LAPTOP-G544JMLU MINGW64 ~/2110990005-scm (master)
$ git clone https://github.com/aasthaanando9/2110990005.git
Cloning into '2110990005'...
remote: Enumerating objects: 16, done.
remote: Counting objects: 100% (16/16), done.
remote: Compressing objects: 100% (16/16), done.
remote: Total 16 (delta 0), reused 0 (delta 0), pack-reused 0.
Receiving objects: 100% (16/16), 1.31 MiB | 259.00 KiB/s, done.
Resolving deltas: 100% (1/1), done.

Aastha Anand@LAPTOP-G544JMLU MINGW64 ~/2110990005-scm (master)
$ ls
2110990005/
Aastha Anand@LAPTOP-G544JMLU MINGW64 ~/2110990005-scm (master)
$ git branch aastha
Fatal: not a valid object name: 'master'
Aastha Anand@LAPTOP-G544JMLU MINGW64 ~/2110990005-scm (master)
$ git branch
Aastha Anand@LAPTOP-G544JMLU MINGW64 ~/2110990005-scm (master)
$ cd 2110990005
Aastha Anand@LAPTOP-G544JMLU MINGW64 ~/2110990005-scm/2110990005 (master)
$ git branch aastha
Aastha Anand@LAPTOP-G544JMLU MINGW64 ~/2110990005-scm/2110990005 (master)
$
```

2. Push the modified branch using `git push origin Branch name`.

```
Aastha Anand@LAPTOP-G544JMLU MINGW64 ~/2110990005-scm (master)
$ cd 2110990005
Aastha Anand@LAPTOP-G544JMLU MINGW64 ~/2110990005-scm/2110990005 (master)
$ git branch aastha
Aastha Anand@LAPTOP-G544JMLU MINGW64 ~/2110990005-scm/2110990005 (master)
$ git branch
  aastha
* master

Aastha Anand@LAPTOP-G544JMLU MINGW64 ~/2110990005-scm/2110990005 (master)
$ git checkout aastha
Switched to branch 'aastha'

Aastha Anand@LAPTOP-G544JMLU MINGW64 ~/2110990005-scm/2110990005 (aastha)
$ git branch
* aastha
  master

Aastha Anand@LAPTOP-G544JMLU MINGW64 ~/2110990005-scm/2110990005 (aastha)
$ pwd
/c/Users/Aastha Anand/2110990005-scm/2110990005

Aastha Anand@LAPTOP-G544JMLU MINGW64 ~/2110990005-scm/2110990005 (aastha)
$ git status
On branch aastha
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    C++ Programs/temperatureproblem.cpp
    C++ Programs/totalsalary.cpp

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

Aastha Anand@LAPTOP-G544JMLU MINGW64 ~/2110990005-scm/2110990005 (aastha)
$ git add .

Aastha Anand@LAPTOP-G544JMLU MINGW64 ~/2110990005-scm/2110990005 (aastha)
$ git status
On branch aastha
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   C++ Programs/temperatureproblem.cpp
    new file:   C++ Programs/totalsalary.cpp

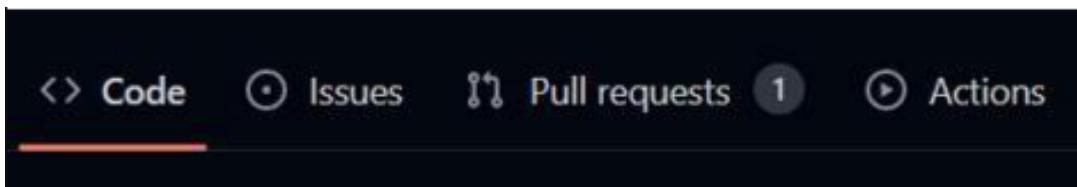
Aastha Anand@LAPTOP-G544JMLU MINGW64 ~/2110990005-scm/2110990005 (aastha)
$
```

3. Open a pull request by following the procedure from the above experiment.

#### Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks.

4. The pull request will be created and will be visible to all the team members.
5. Ask your team member to login to his/her Github account.
6. They will notice a new notification in the pull request menu.



7. Click on it. The pull request generated by you will be visible to them.

#### Adding C++ files to the repo #2

8. Click on the pull request. Two options will be available, either to close the pull request or Merge the request with the main branch.
9. By selecting the merge branch option the main branch will get updated for all the team members.
10. By selecting close the pull request the pull request is not accepted and not merged with main branch.
11. The process is similar to closing and merging the pull request by you. It simply includes an external party to execute.
12. Thus, we conclude opening and closing of pull request. We also conclude merging of the pull request to the main branch.

The screenshot shows a GitHub pull request page for an update titled '#3'. The top navigation bar includes links for Code, Issues, Pull requests (with a count of 1), Actions, Projects, Wiki, Security, Insights, and Settings. The pull request itself has a red 'Closed' button and a note stating 'Itsak wants to merge 1 commit into Group01-Chitkara-University:main from Itsak:main'. The main content area shows a comment from 'Itsak' saying 'No description provided.' and an update from 'aasthaanand09' closing the pull request. On the right side, there are sections for Reviewers (no reviews), Assignees (no one assigned), Labels (none yet), and Projects (none yet). Below the main content is a rich text editor with 'Write' and 'Preview' tabs and various formatting icons.

## Experiment No. 04

### Aim: Publish and print network graphs

The network graph is one of the useful features for developers on GitHub. It is used to display the branch history of the entire repository network, including branches of the root repository and branches of forks that contain commits unique to the network.

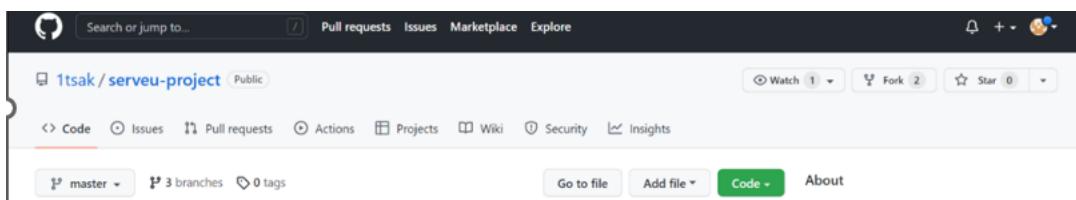
A repository's graphs give you information on traffic, projects that depend on the repository, contributors and commits to the repository, and a repository's forks and network. If you maintain a repository, you can use this data to get a better understanding of who's using your repository and why they're using it.

Some repository graphs are available only in public repositories with GitHub Free:

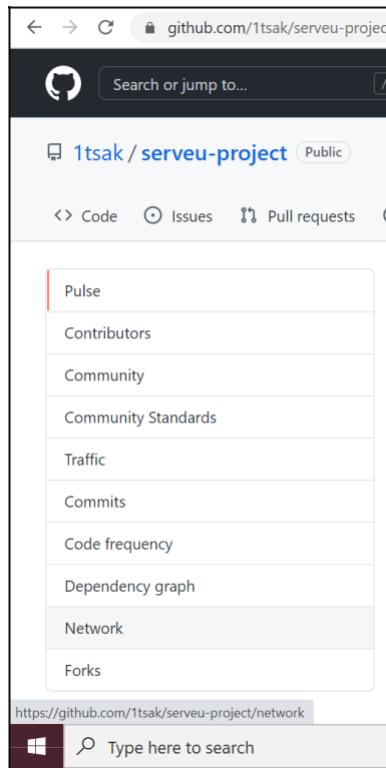
- Pulse
- Contributors
- Traffic
- Commits
- Code frequency
- Network

### Steps to access network graphs of respective repository

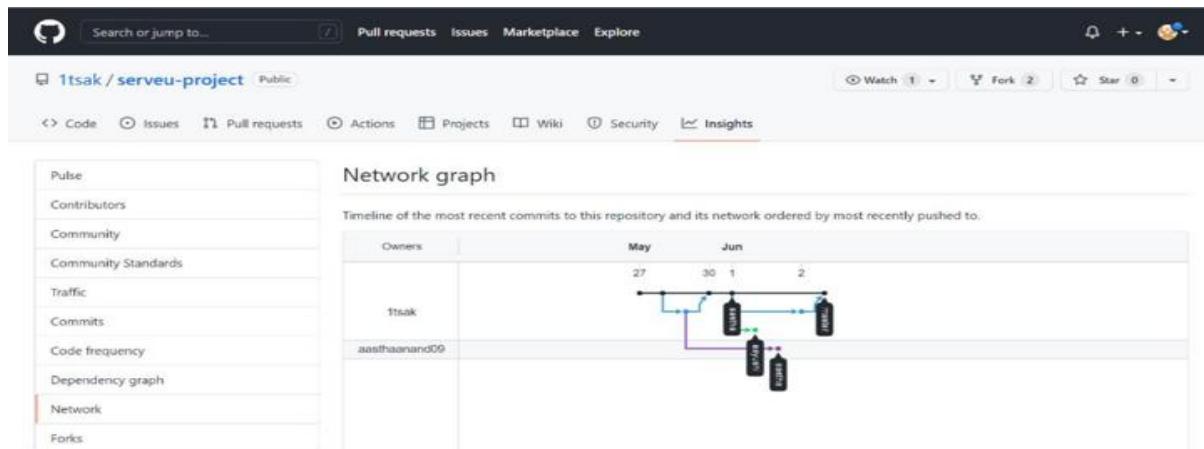
1. On GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click **Insights**.



3. At the left sidebar, click on Network.



4. You will get the network graph of your repository which displays the branch history of the entire repository network, including branches of the root repository and branches of forks that contain commits unique to the network.

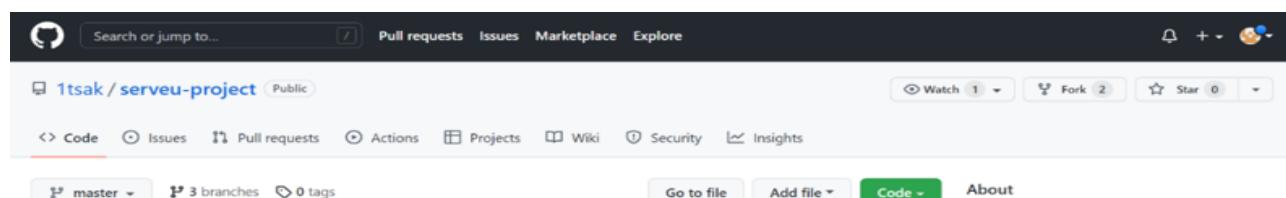


## **Listing the forks of a repository**

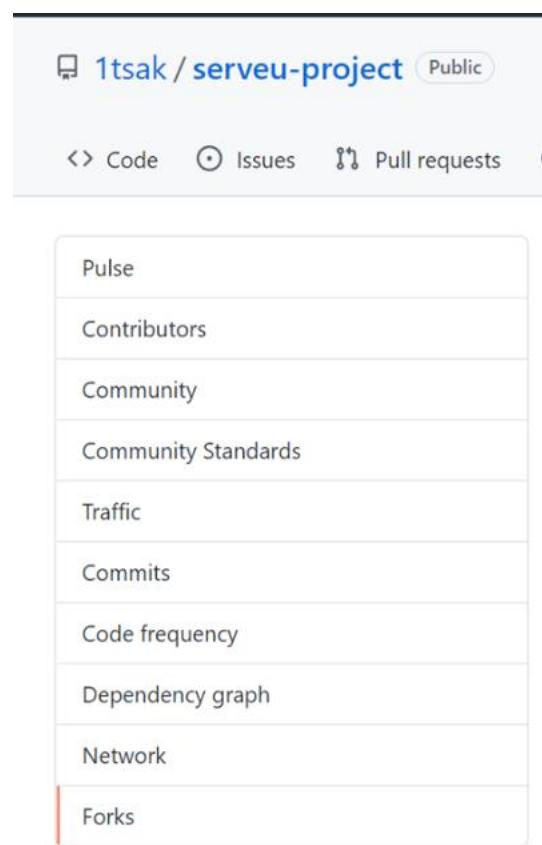
Forks are listed alphabetically by the username of the person who forked the repository

Clicking the number of forks shows you the full network. From there you can click "members" to see who forked the repo.

1. On GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click **Insights**.



3. In the left sidebar, click **Forks**.



#### 4. Here you can see all the forks

The screenshot shows a GitHub repository page for '1tsak / serveu-project'. At the top, there are buttons for Watch (1), Fork (2), and Star (0). Below the header, there are tabs for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, and Insights, with 'Insights' being the active tab. On the left, a sidebar lists various metrics: Pulse, Contributors, Community, Community Standards, Traffic, Commits, Code frequency, Dependency graph, Network, and Forks. The 'Forks' option is currently selected and highlighted with a red border. In the main content area, it shows the original repository '1tsak / serveu-project' and three forks: 'aasthaanand09 / serveu-project' and 'heyaadarsh / serveu-project', each represented by a small profile icon and the repository name.

## Viewing the dependencies of a repository

You can use the dependency graph to explore the code your repository depends on.

Almost all software relies on code developed and maintained by other developers, often known as a supply chain. For example, utilities, libraries, and frameworks. These dependencies are an integral part of your code and any bugs or vulnerabilities in them may affect your code. It's important to review and maintain these dependencies.

---