Subject Name: **Source Code Management**

Subject Code: **22CS003**

Session: **2022-23**

Department: **DCSE**

****

|  |  |  |
| --- | --- | --- |
| **Submitted By:**  Bhuvesh Mittal  2210991450  G5-A |  | **Submitted To:**  Dr. Garima Chopra |

**List of Programs**

|  |  |  |
| --- | --- | --- |
| **S. No** | **Program Title** | **Page No.** |
| 1 | Setting up of Git Client | 3-4 |
| 2 | Setting up GitHub Account | 5-6 |
| 3 | Generate logs | 7 |
| 4 | Create and visualize branches | 8-10 |
| 5 | Git life cycle description | 11-12 |
| 6 | Add collaborators on GitHub Repo | 13-14 |
| 7 | Fork & commit | 15-17 |
| 8 | Merge and Resolve conflicts created due to own activity and collaborators activity. | 18-20 |
| 9 | Reset and revert | 21-24 |

Experiment 1. Setting up of Git Client

Git

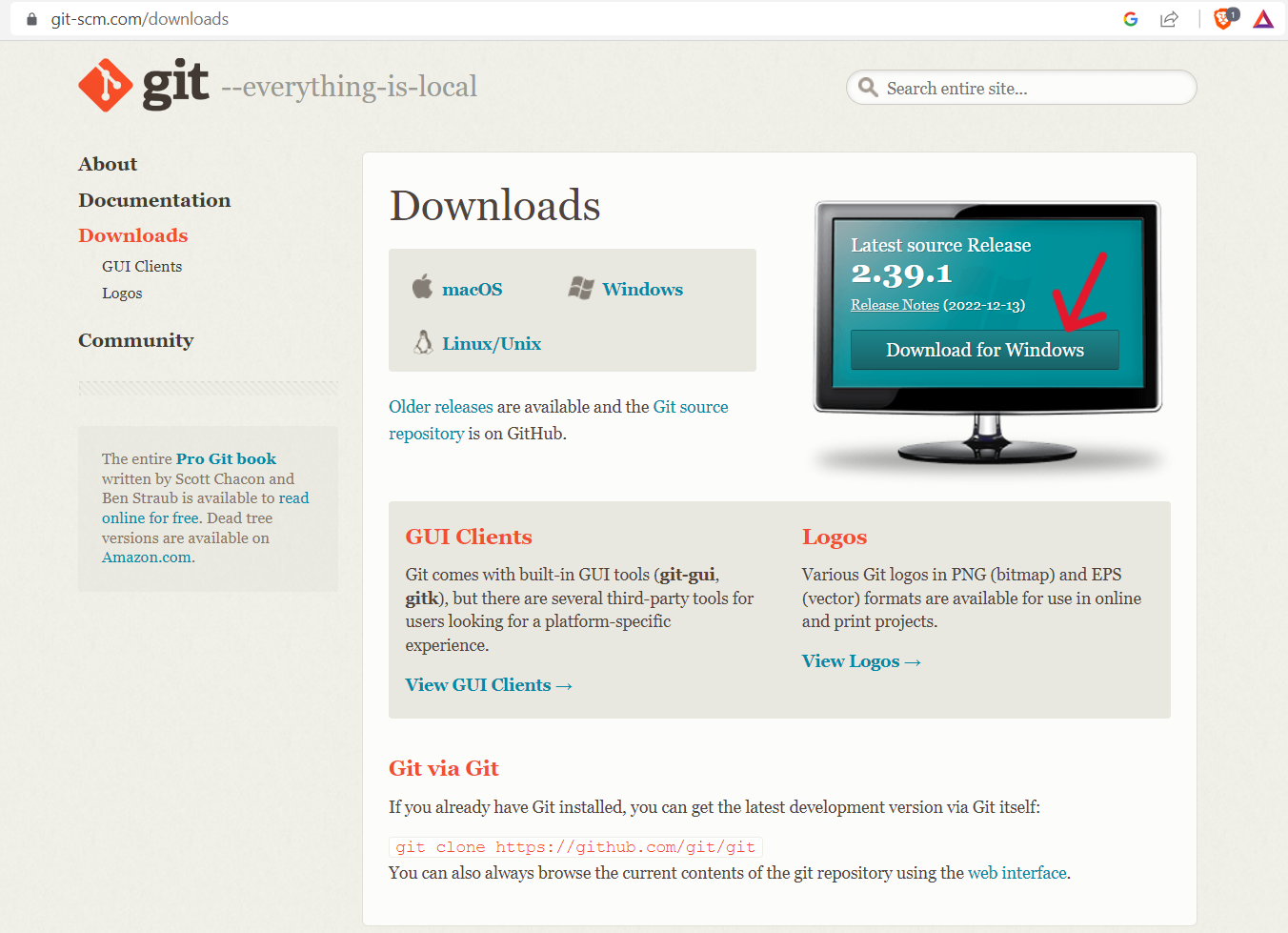
*Git* is a distributed version control system that tracks changes in any set of computer files, usually used for coordinating work among programmers collaboratively developing source code during software development. Its goals include speed, data integrity, and support for distributed, non – linear workflows.

Advantages

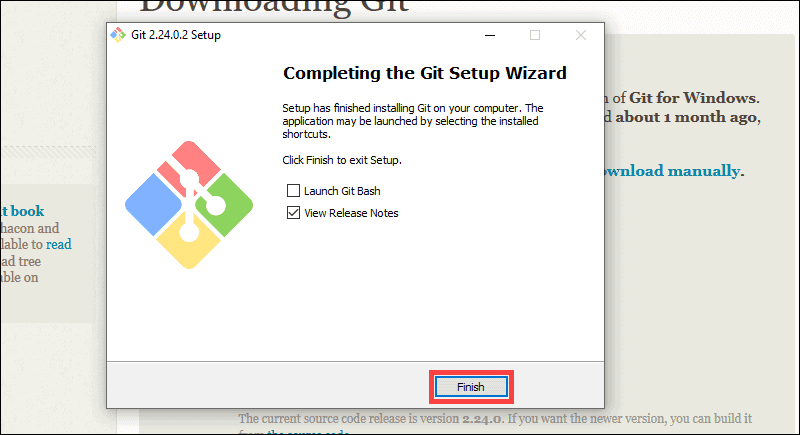
* Performance
* Security
* Flexibility
* Wide Acceptance
* Quality open-source project

**Step 1.** Open the git website: [***https://git-scm.com/downloads***](https://git-scm.com/downloads)

**Step 2.** Click on ***Download for Windows***

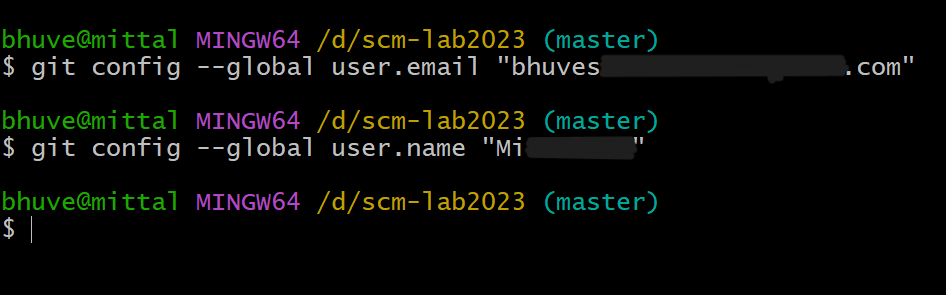


**Step 3.** Click on ***finish*** for complete installation.



**Step 4.** Set up **git** using the following commands:

* *$ git config –global user.email “\*\*\*@gmail.com”*
* *$ git config –global user.name “\*\*\*”*



Finally, the git client is successfully setup.

**To unset the git client, following commands will be used:**

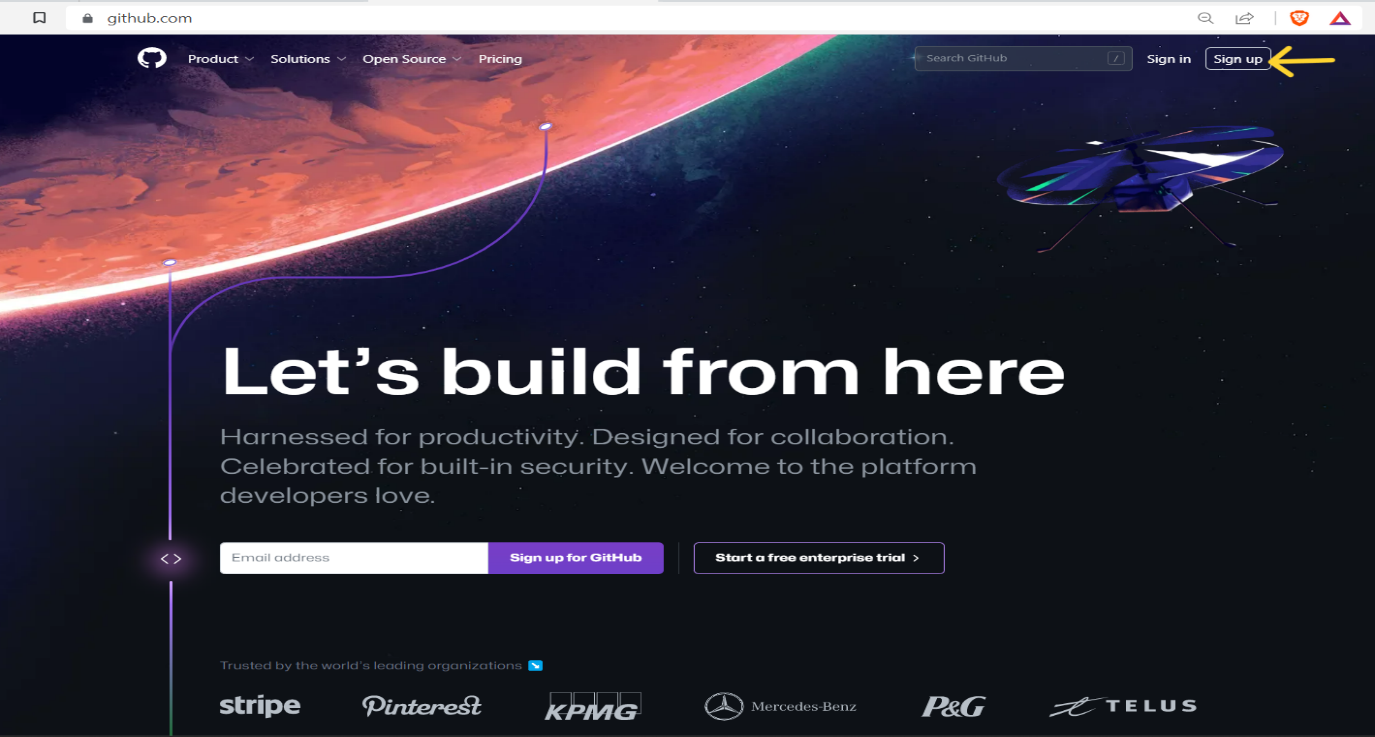
* *$ git config --global --unset user. Email*
* *$ git config --global --unset user.name*

Experiment 2. Setting up of Git Hub account

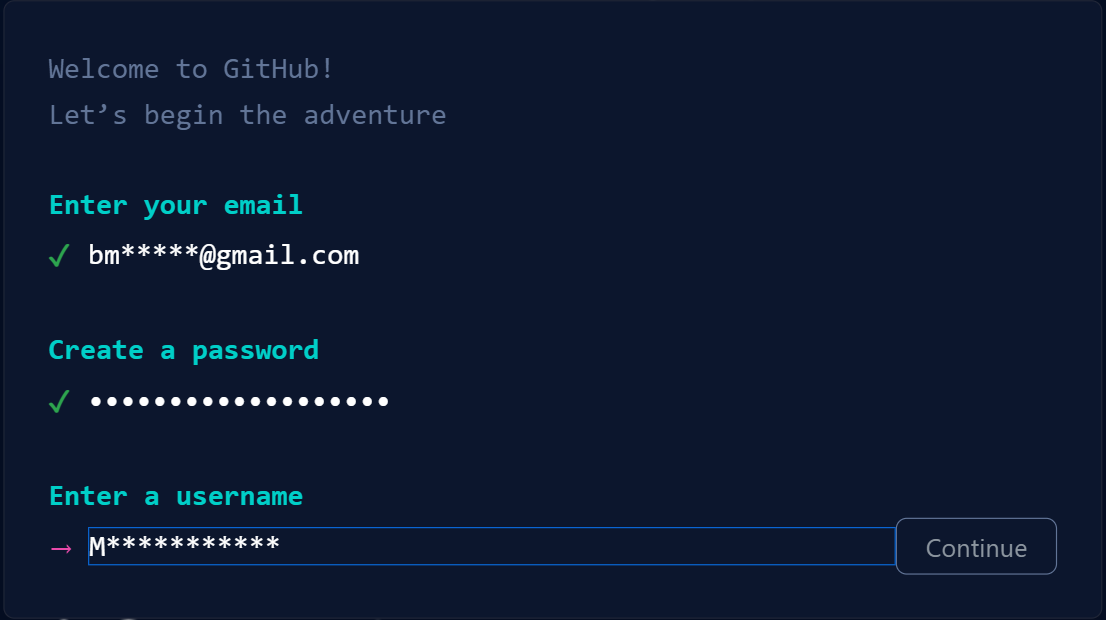
**GitHub** is an internet hosting service for software development and version control using git. It provides the distributed version control of git plus access control, bug tracking, software feature

Requests, task managements, continuous integration, and wikis for every project.

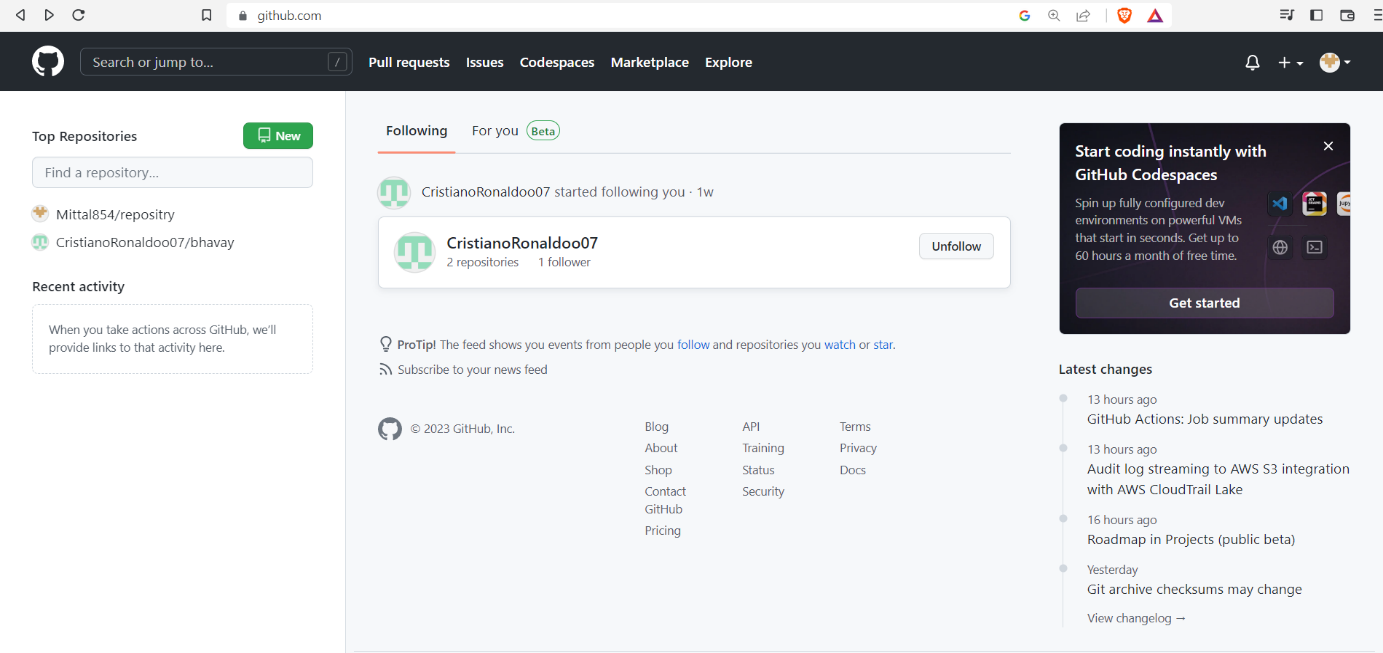
**Step 1.** Browse the GitHub website: [***https://github.com/***](https://github.com/) & click on **sign up**.



**Step 2.** Enter your personal details for the account.



**Step 3.** Successfully, account is created.

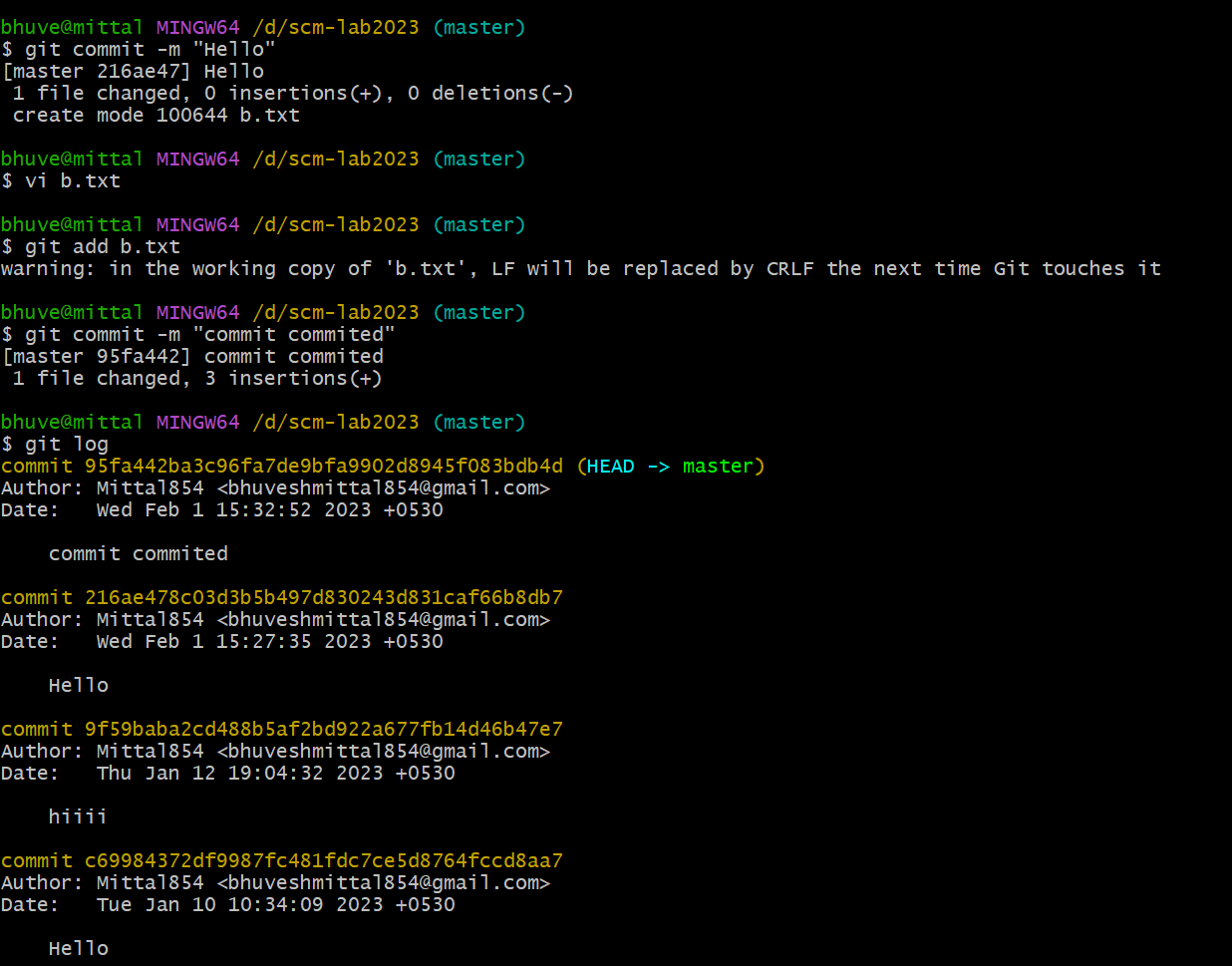


Experiment 3. Generate Logs

It displays all the commits being made in that repository in multiple lines along with the commit id, author name, date and commit message. Logs helps us to check the changes made in code or files and by whom. It also contains time at which change was made.

The command to generate logs is:

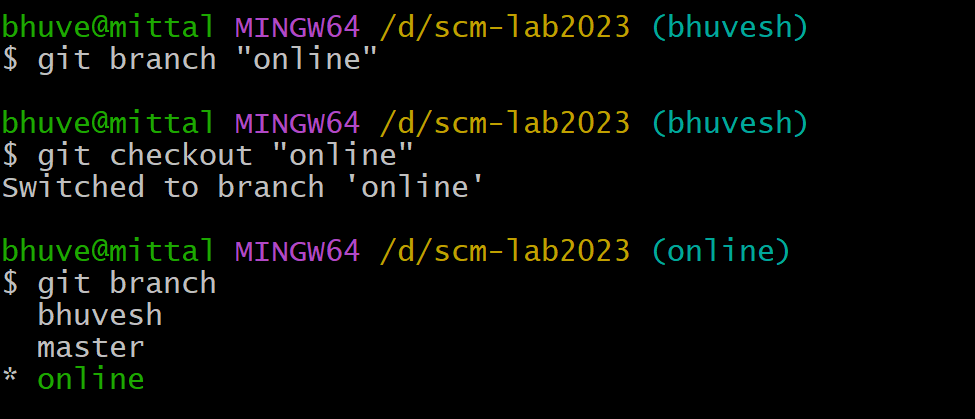
* $ git log



Experiment 4. Create & visualize Branches

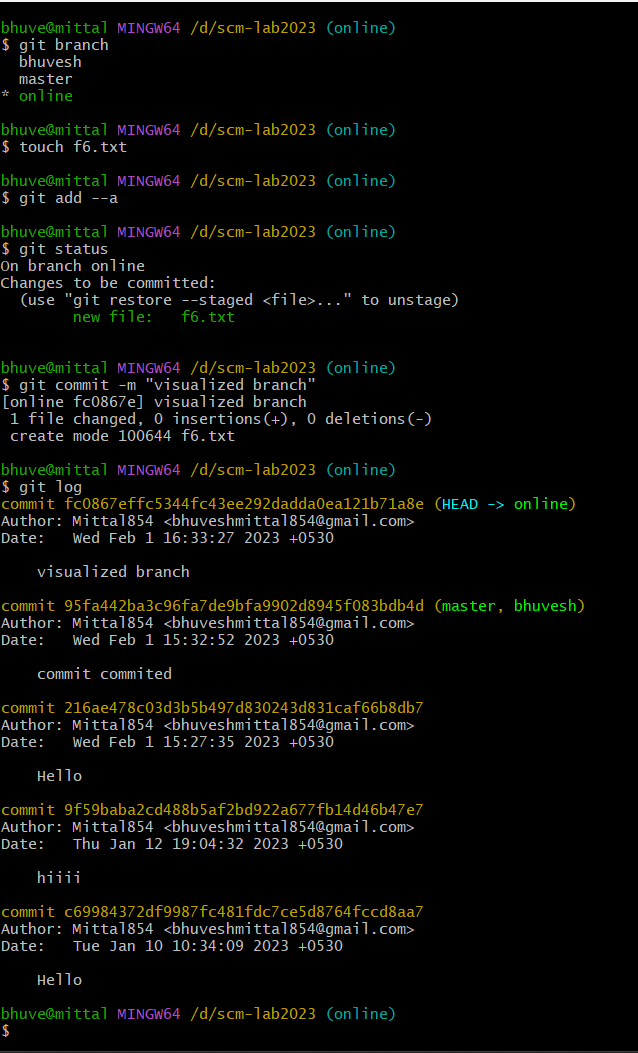
In Git, a branch is a new/separate version of the main repository. Branches allow you to work on different parts of a project without impacting the main branch. When the work is complete, a branch can be merged with the main project. You can even switch between branches and work on different projects without them interfering with each other.

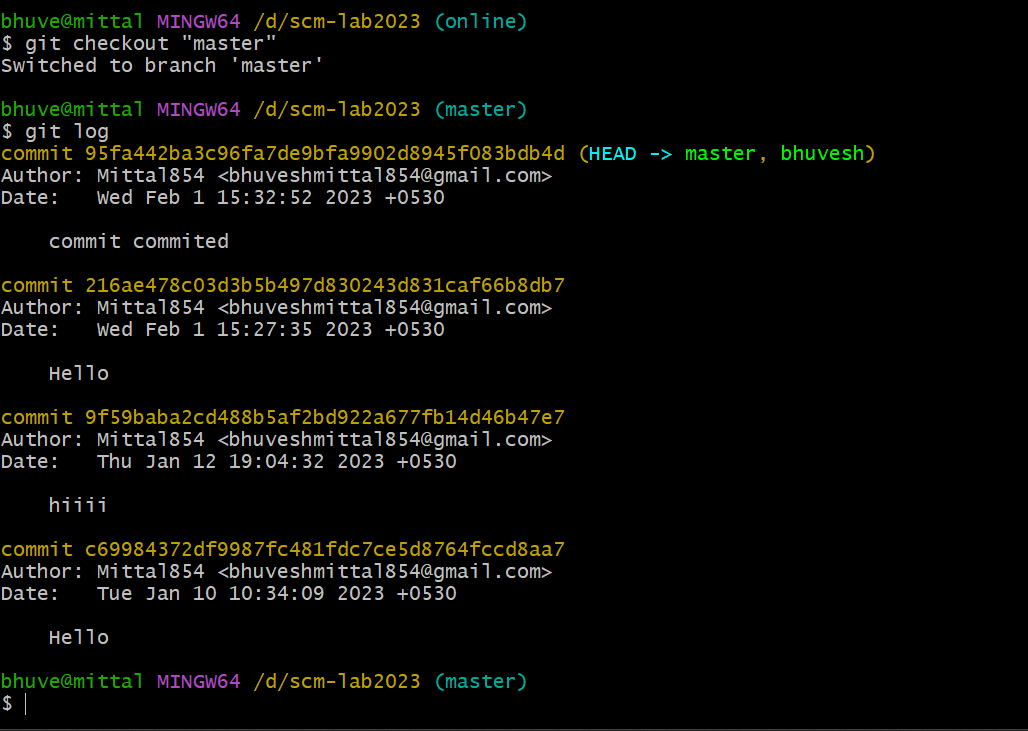
* **For creating a new branch:** git branch “name of the branch”
* **To change the present working branch:** git checkout “name of the branch”
* **To check how many branches:** git branch



**Visualization branches:**

To visualize, we have to create a new file in the new branch “online” instead of the master branch. After this we have to do three step architecture i.e., working directory, staging area and git repository.





Experiment 5. Git life cycle description

When a directory is made a git repository, there are mainly 3 states which make the essence of Git Version Control System. The three states are –

* Working Directory
* Staging Area
* Git Directory

**1. Working Directory**

Whenever we want to initialize our local project directory to make it a git repository, we use the ***git init*** command. After this command, git becomes aware of the files in the project although it doesn’t track the files yet. The files are further tracked in the staging area.

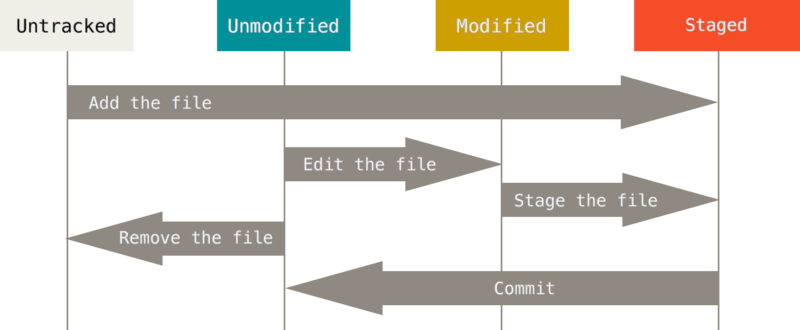
**2. Staging Area**

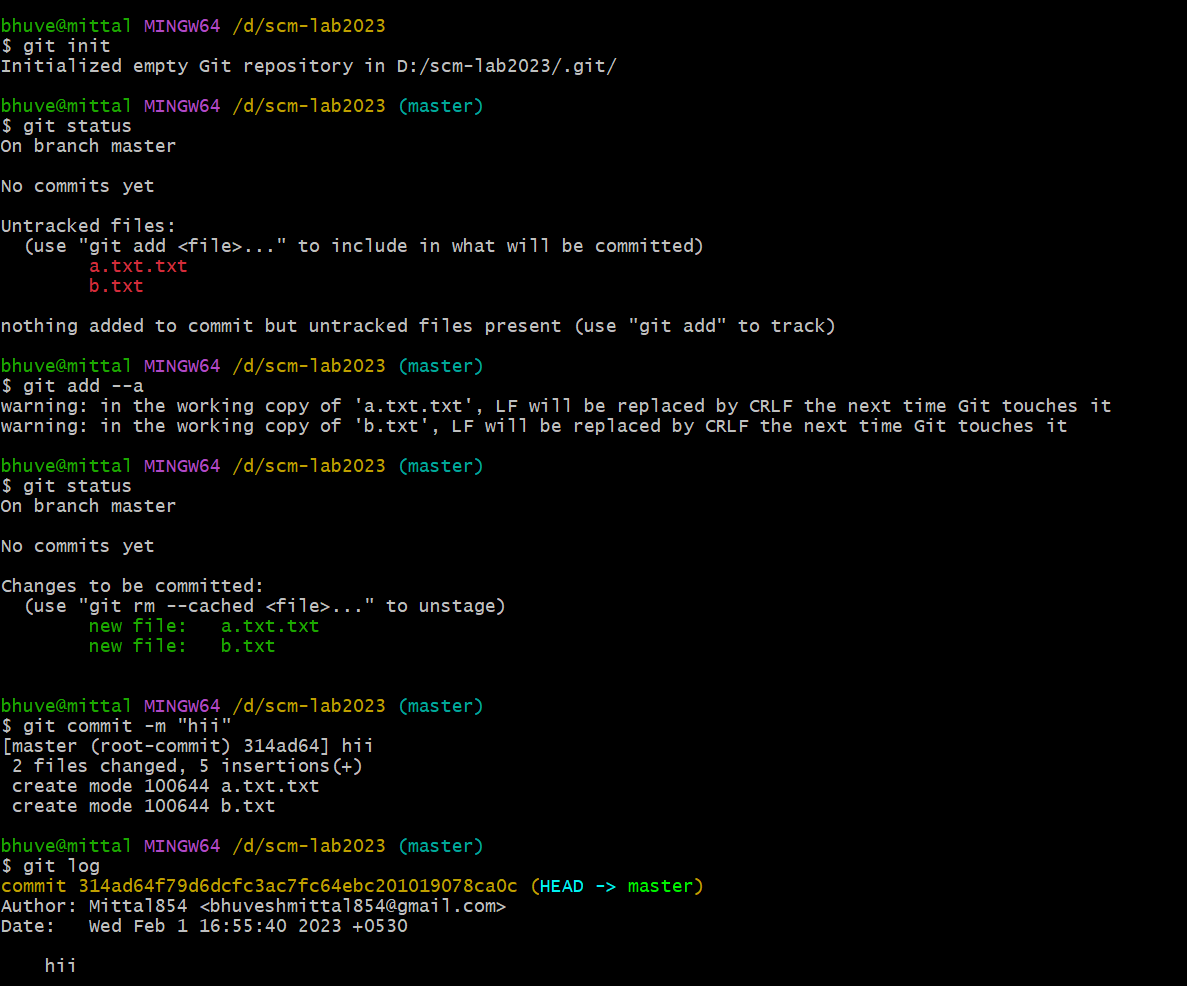
Staging area is the playground where you group, add and organise the files to be committed to Git for tracking their versions.

**3. 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.



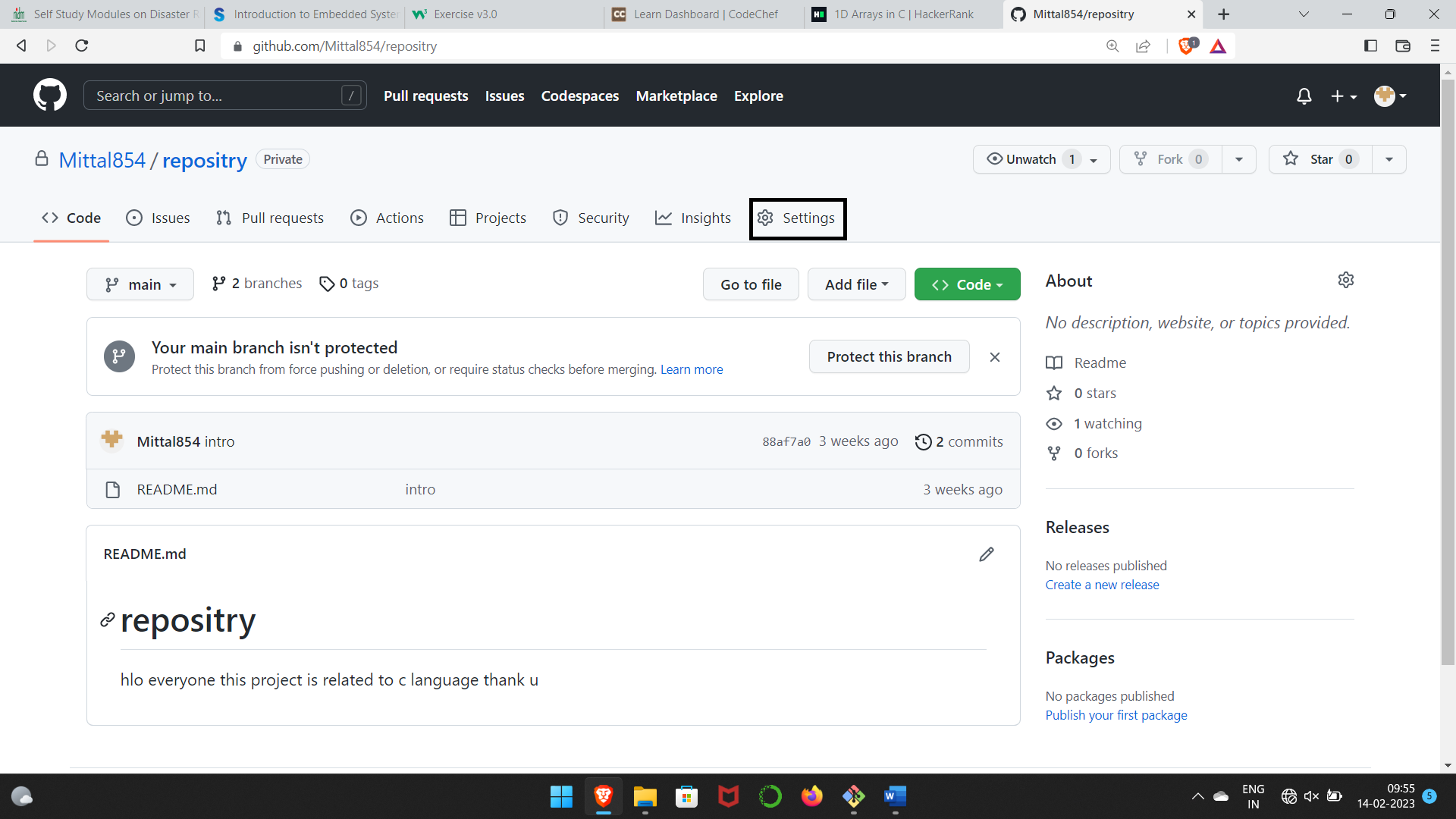


Experiment 6. Add collaborators to GitHub Repo

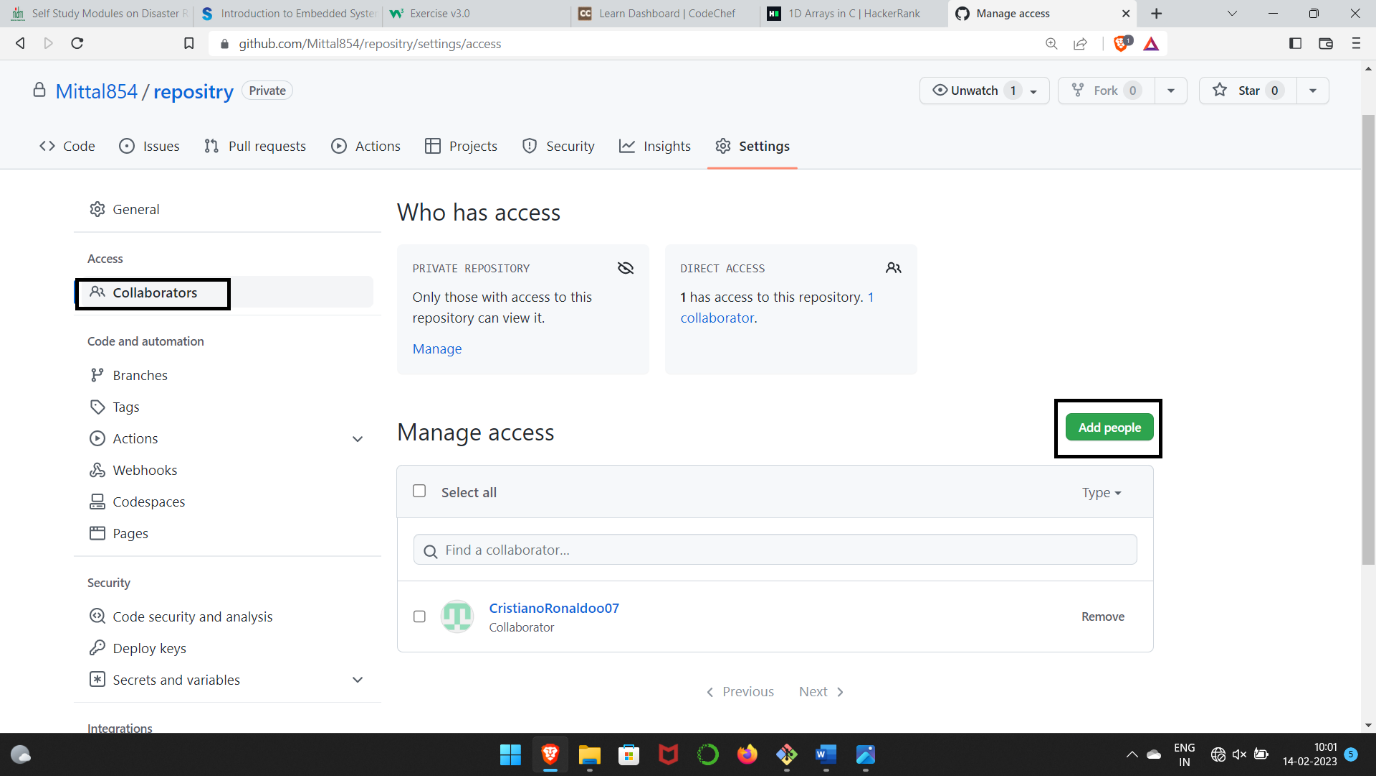
**Step 1.** Open the GitHub website: [***https://github.com/***](https://github.com/) and log in to your account.

**Step 2.** Create a new repository.

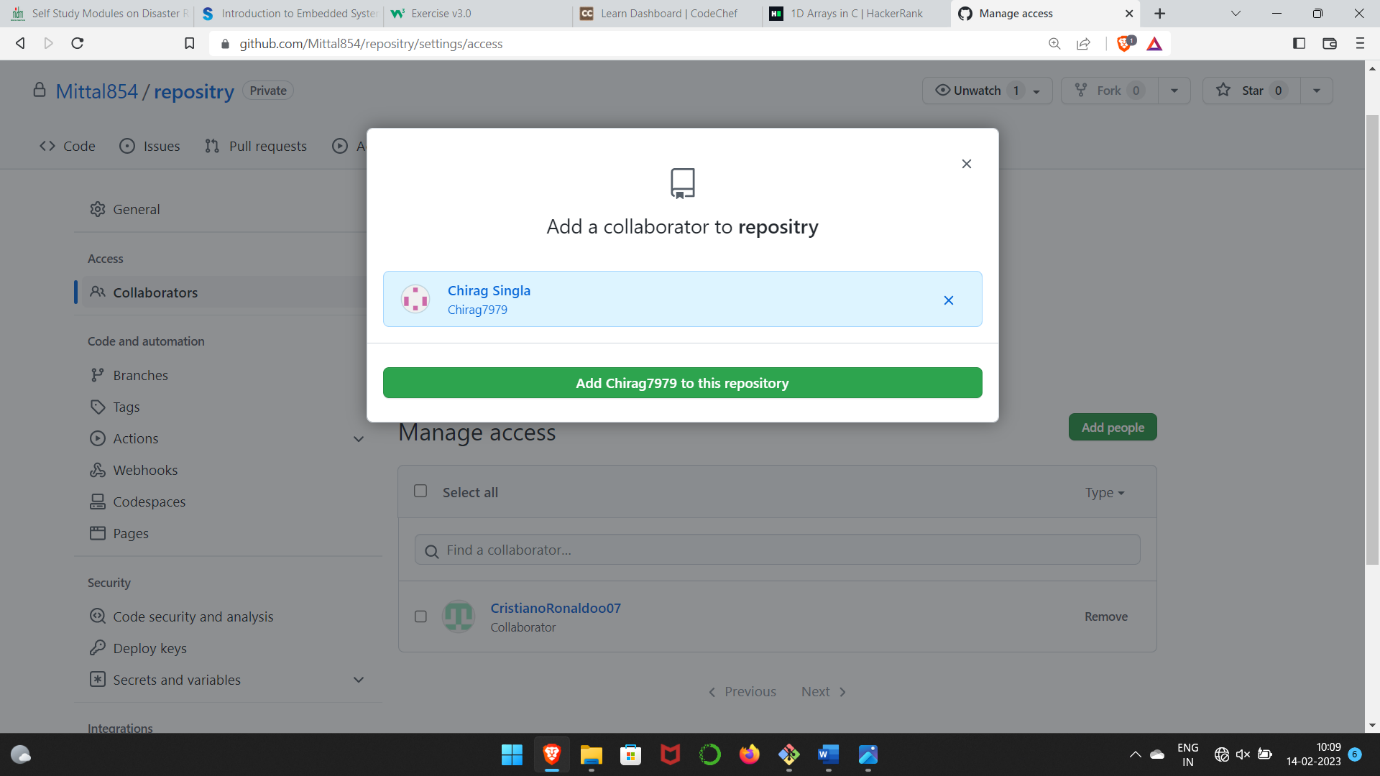
**Step 3.** After creating a repository, go to settings.



**Step 4.** Go to collaborators & click on add people.

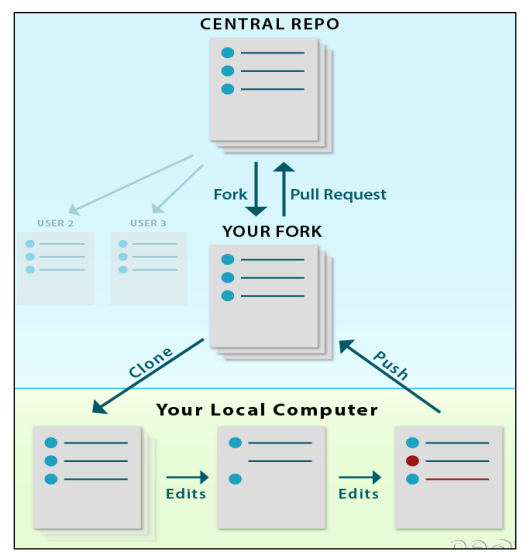


**Step 5.** Search the collaborators name and add him to the repository.



Experiment 7. Fork & Commit

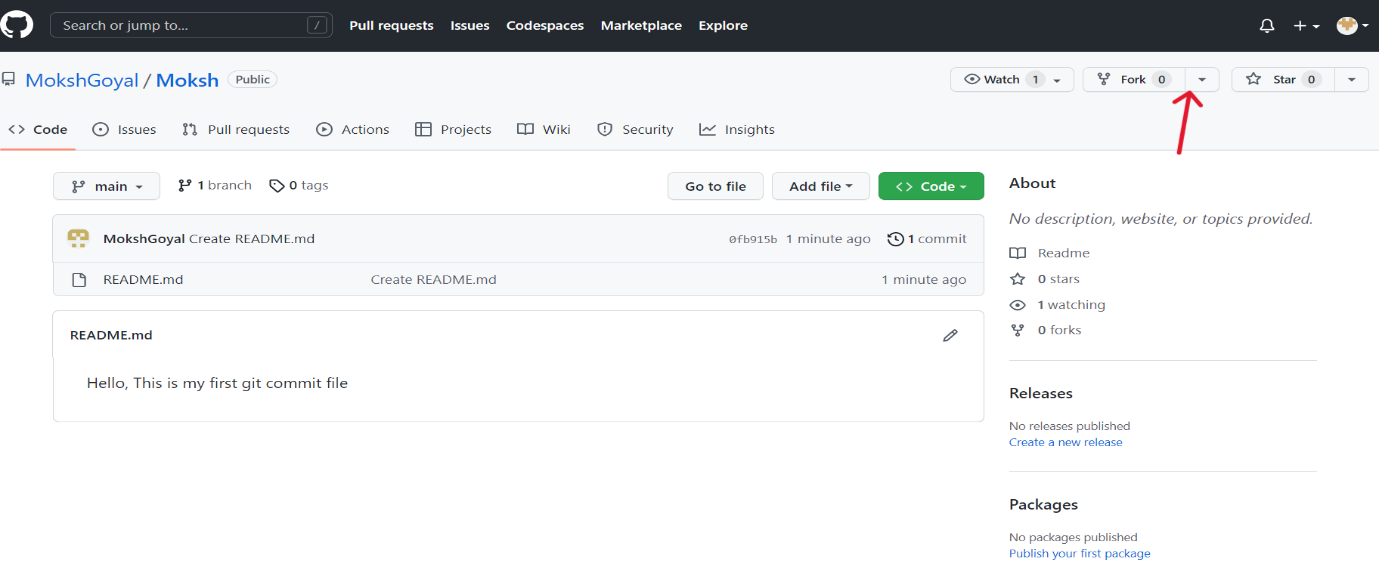
A fork is a copy of a repository that you manage. It allows us to freely experiment with the data. After creating a fork, we can make any desired change like adding collaborators, rename files, generate GitHub pages but all these changes won’t be reflected in the original repository.



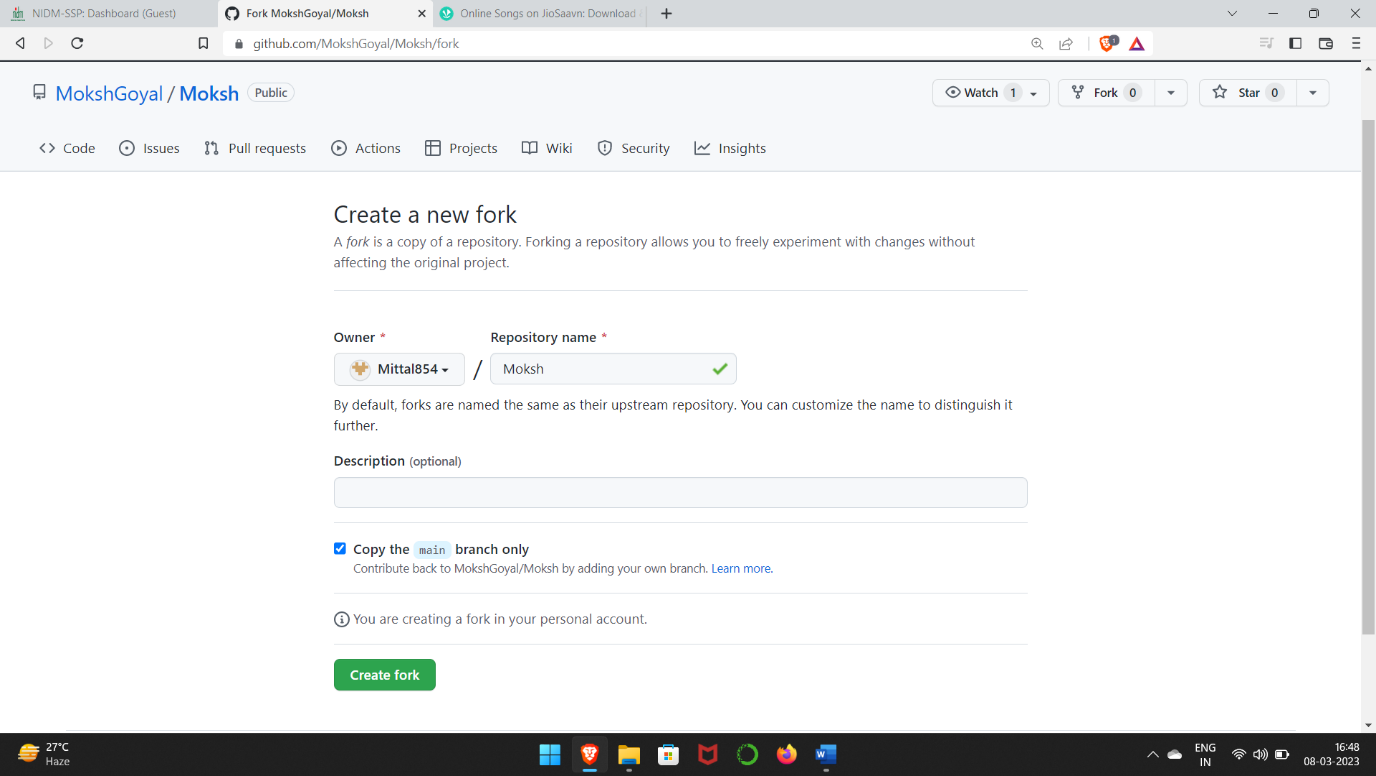
To import the changes into the original repository, the user needs to send a pull request to the maintainer. If the maintainer closes the pull request only then the content can be added to the original repository.

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

**Step 1.** Open the repository which you want to fork & click on fork option.

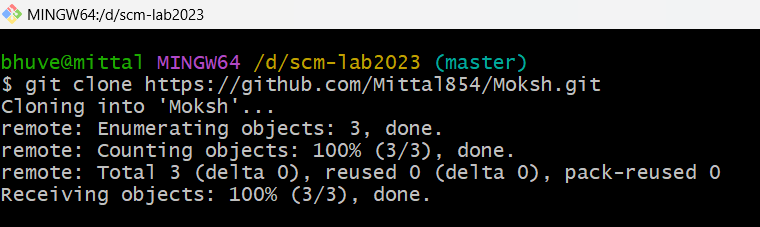


**Step 2.** Click on create fork.

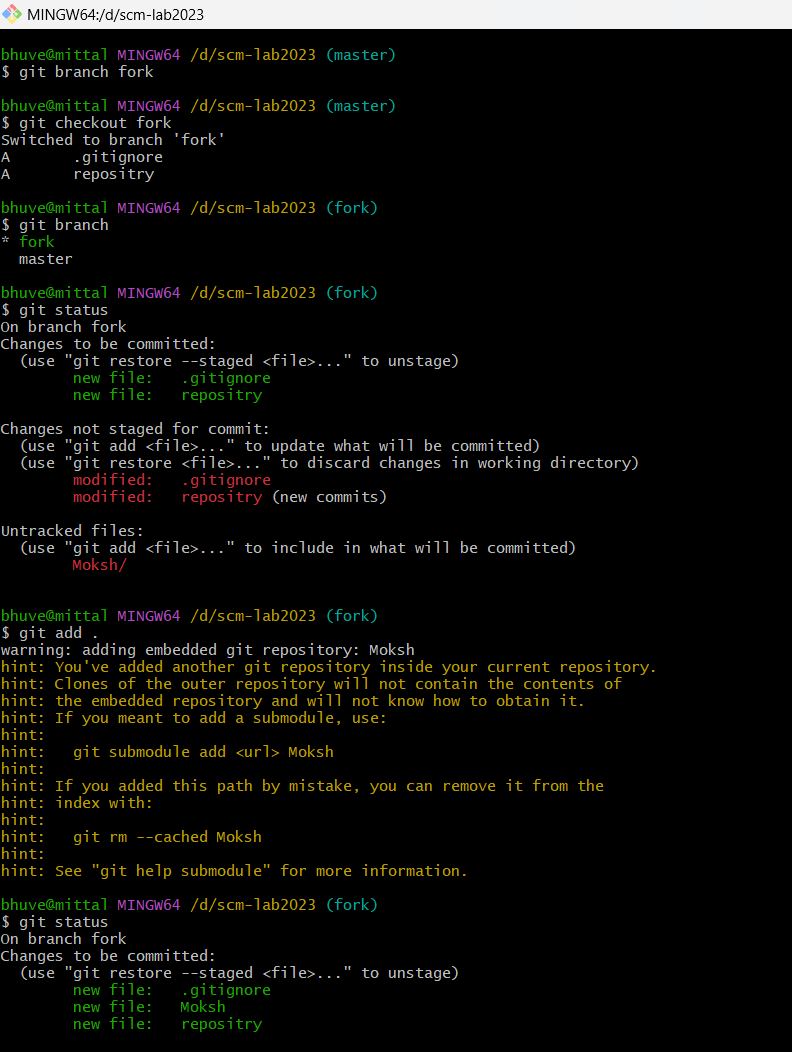


**Step 3.** A copy of the repo which is forked from other user is formed. We can now make modification without changing main source code.

**Step 4.** Now use the command **git clone <URL>** to fetch the remote repo or clone the repo.

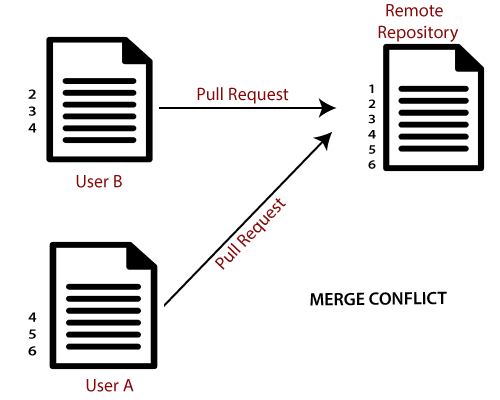


**Step 5.**  Now we can open the file, make changes & commit.



Experiment 8. Merge and Resolve conflicts created due to own activity and collaborators activity.

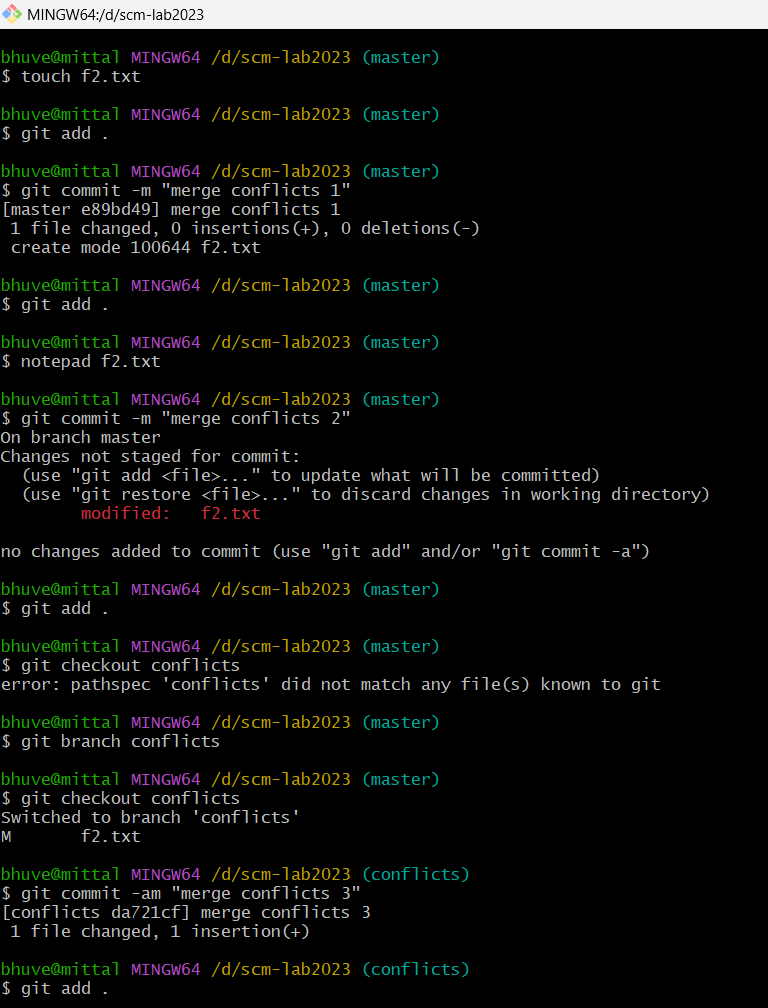
Version control systems are all about managing contributions between multiple distributed authors (usually developers). Sometimes multiple developers may try to edit the same content. If Developer A tries to edit code that Developer B is editing a conflict may occur. reference for picture:



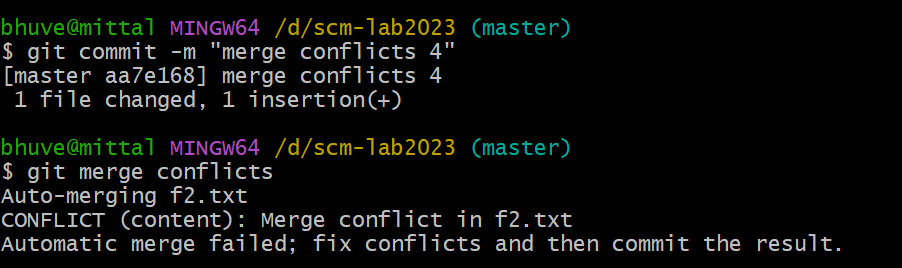
If you have a merge conflict on the command line, you cannot push your local changes to GitHub until you resolve the merge conflict locally on your computer.

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.

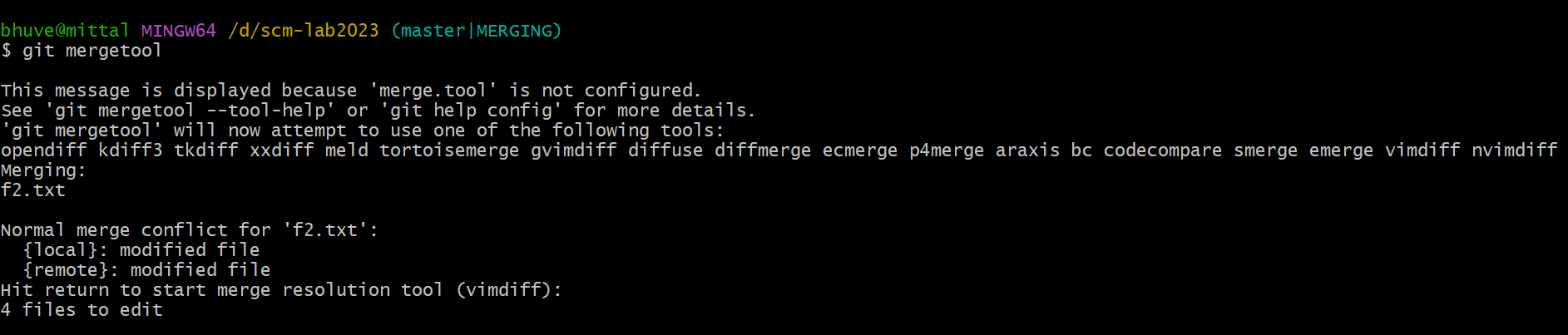
**Step 1.** Make changes in the master branch & commit them. Now checkout to another branch, make changes & commit them also.



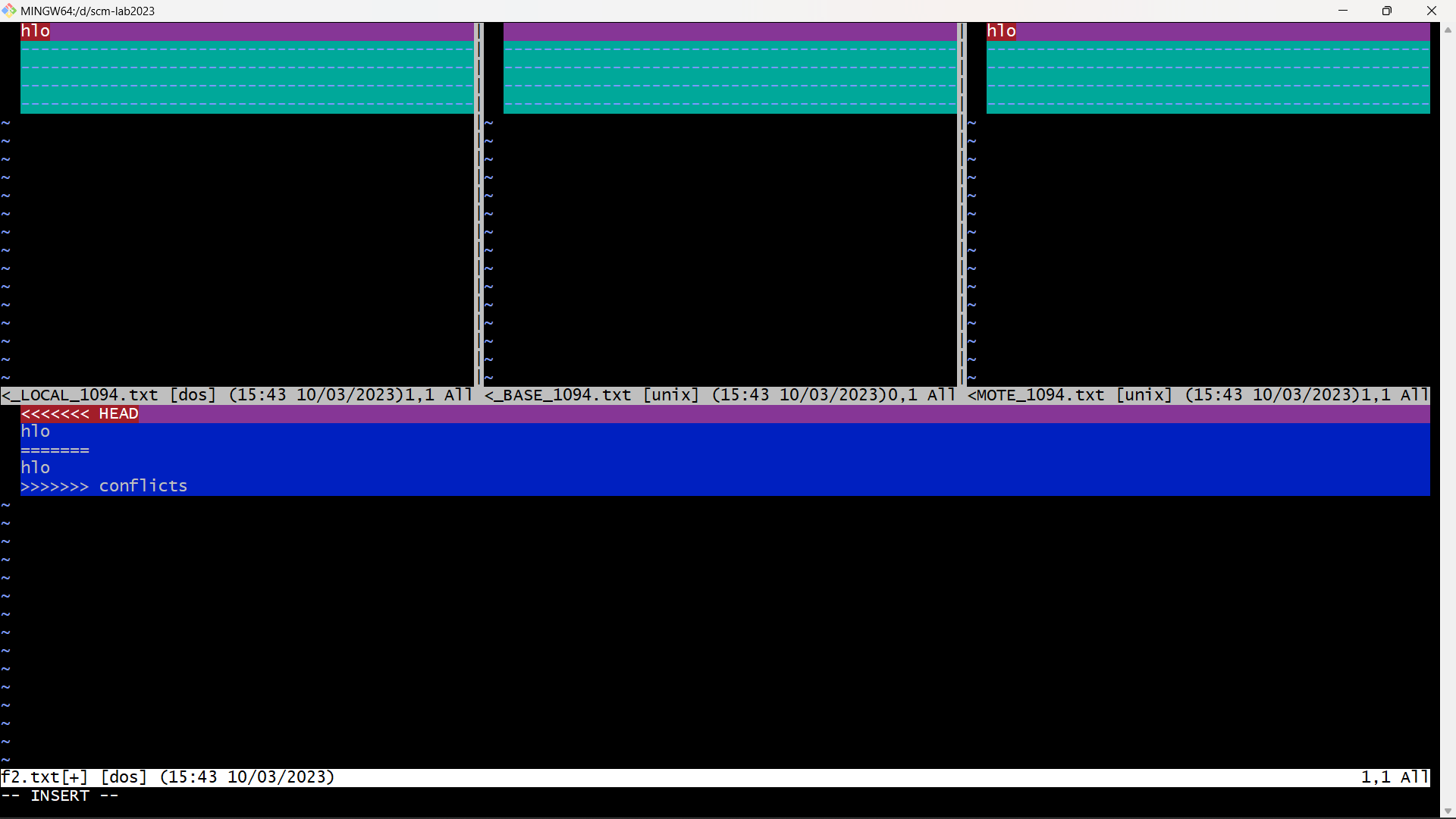
**Step 2.**  Now merge both the branches. It will give Conflicts error.



**Step 3.** Use **git mergetool** command to resolve the conflict. It runs the merge conflict resolution tools to resolve merge conflicts.

****

**Step 4.** To insert, press **I**. Then type **:wq**. The merge conflict is solved and secondary branch is merged into the master branch.



Experiment 9. Reset and Revert

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.

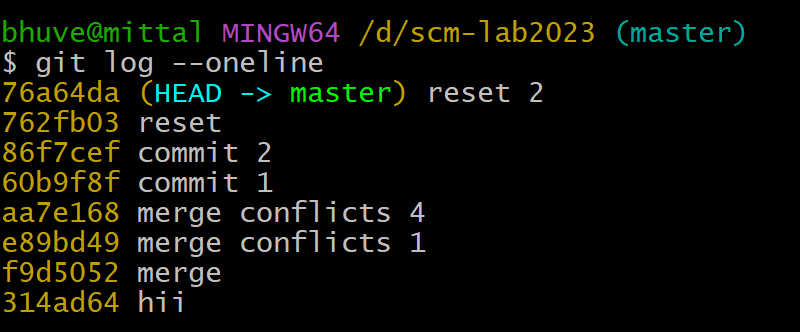
A revert is an operation that takes a specified commit and creates a new commit which inverses the specified commit. git revert can only be run at a commit level scope and has no file level functionality.

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

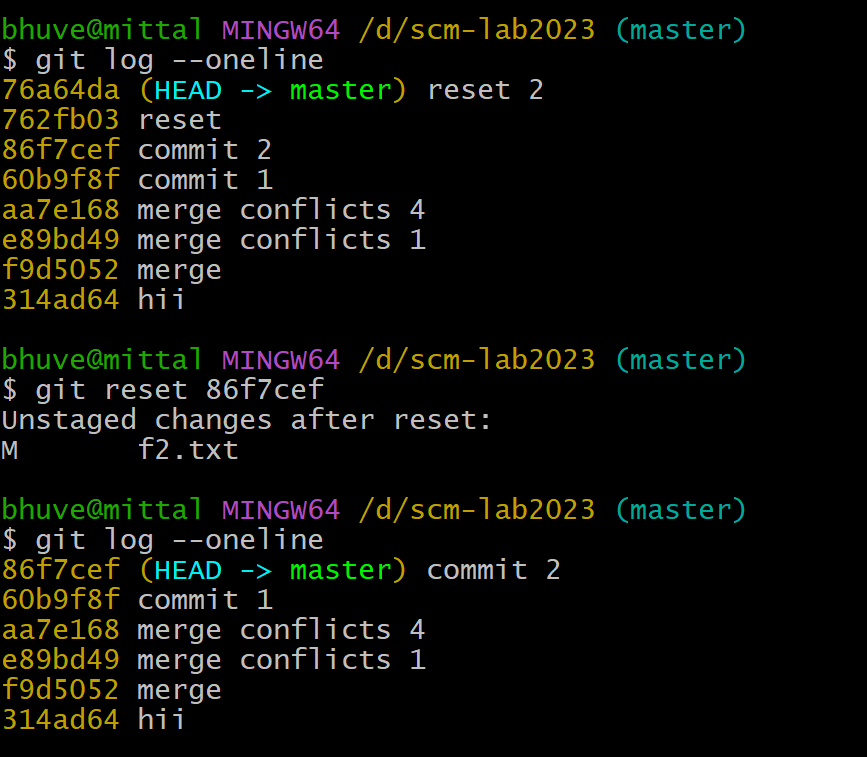
**Reset: -**

**Step 1.** Prepare a log of multiple commits to use reset command.

**Step 2.** Check git log.



**Step 3.** Pick any commit which we want the repository to rollback. Paste the checksum of the commit in the **git reset** command.

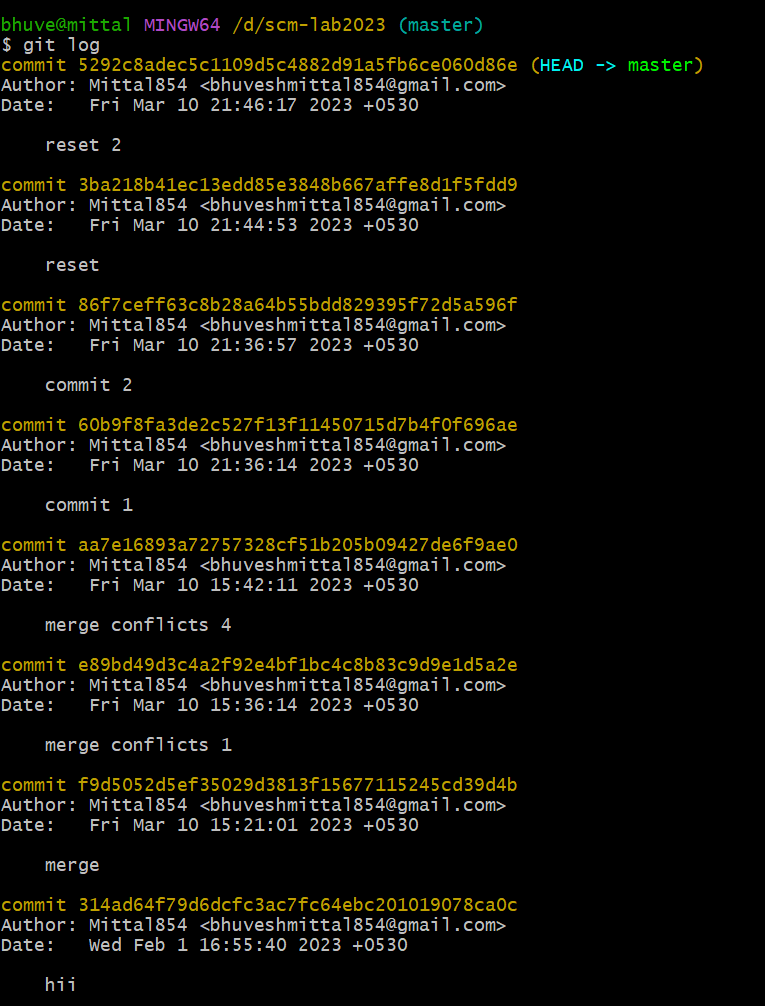


The head is now pointing the commit whose checksum we have provided that means the commit that followed vanished.

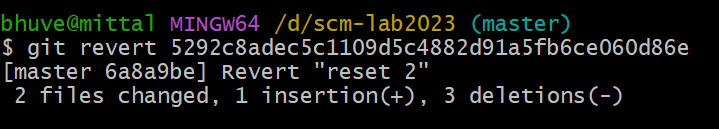
**Revert: -**

**Step 1.** Prepare a log of multiple commits to use reset command.

**Step 2.** Check git log.



**Step 3.** Pick up the commit which we want to revert. Paste the checksum in the **git revert** command.



**Step 4.** A window will appear. Enter the statement which we want to be displayed after reverting the commit.

**Step 5.** Check the git log. We will find that another commit is added without affecting the rest commits.



The change associated to the reverted commit has disappeared.

A Project report

On

**“Task 2”**

With

**Source Code Management**

(22CS003)

**Submitted by:**

Name: Bhuvesh Mittal

Roll No.: 2210991450

# Submitted To:

Dr. Garima Chopra

Department of Computer Science & Engineering, Chitkara University Institute of Engineering and Technology Rajpura, Punjab

Team Member 1 Name: Bhavya Sood Roll No:2210991443

Team Member 2 Name: Bhoomika Roll No:2210991446

Team Member 3 Name: Bhuvesh Mittal Roll No:2210991450



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

Name **Technology**

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

Program Name: -

**Bachelor of Engineering (B.E.), Computer Science& Engineering**

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

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Management** |  |  |
| Course Code: - | **22CS003** | Batch: **1st/2023** |  |
| Vertical Name: - | **First Year** | Group No: 05/A |  |

**Faculty Name: Dr. Garima Chopra**

**Signature:**

**Date:**

**Table of Content**

|  |  |  |
| --- | --- | --- |
| **S. NO** | **Title** | **Page No.** |
| 1. | Version control with Git | 28 |
| 2. | Problem statement | 29 |
| 3. | Objective | 30 |
| 4. | Resources Requirements – FrontEnd/  Backend | 31 |
| 5. | Concepts and commands | 32 |
| 6. | Workflow and Discussion | 33-40 |
| 7. | Reference | 41 |

Version Control with Git

▪ 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.

▪ Git is a distributed type of VCS which means it allows the user to store data both on a local system and on a server.

▪ It allows the user to make changes in a file without disturbing the original file.

▪ Git shows all the information about the changes made which makes it really easy to detect when an error was made and by whom.

▪ Git allows the user to make different versions of a project and work on them separately and all this is done without changing the original project.

▪ It helps the users to collaborate on a same project even without being in a same place.

▪ It allows the user the user to merge or delete different versions or branches of a project.

▪ Git also helps the user to easily recover a file in case it gets deleted or due to any malfunction.

PROBLEM STATEMENT

1. How to use git as a Version Control.

2. To take care of the Collaboration problem.

3. Using Git to solve basic arithmetic problems.

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 as we can exactly pinpoint when and where an error was made and by whom.

4. Moreover, we can make our projects locally as well as globally available.

RESOURCE REQUIREMENTS

Back End –

1. C Programming Language

2. Git Bash

3. GitHub

CONCEPTS AND COMMANDS

There are various concepts and commands used in this project.

▪ Firstly, we have used the used the concept of adding collaborators on GitHub.

▪ The concept of push and pull has been used to upload the data to GitHub from the local system and to merge the branches to the main branch respectively.

▪ The concept of cloning has been used to copy the repository made on GitHub to the local computer.

▪ Basic git commands like git add, git config, git commit, git status, git branch, git checkout, notepad, touch etc. have been used to perform their required functions.

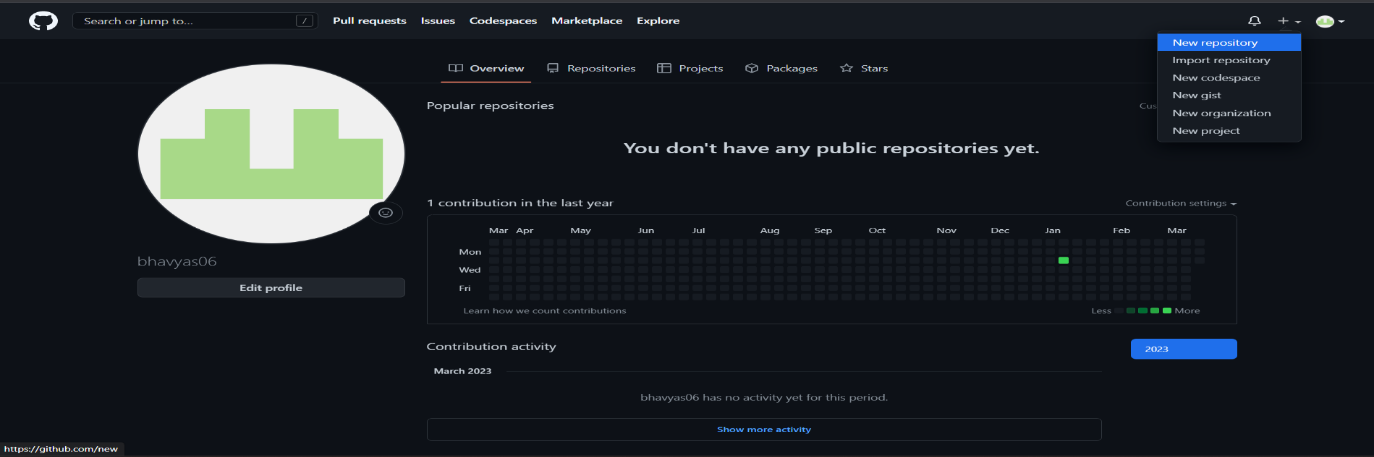
▪ And, we have used to concept of merging to merge the branches and the main branch.

WORKFLOW AND DISCUSSION

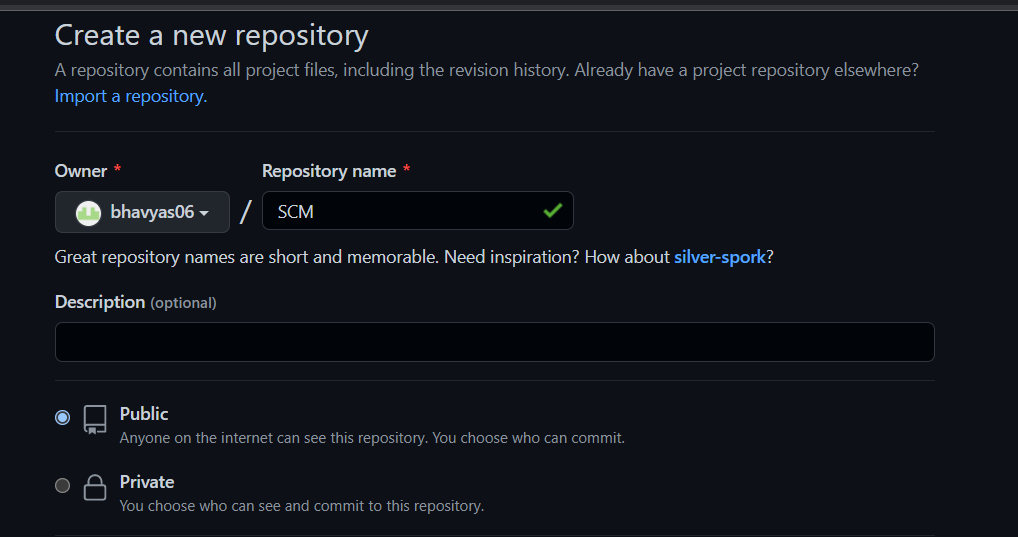
**Create a distributed repository and add members of your project team**

Step 1: Log in to your GitHub account.

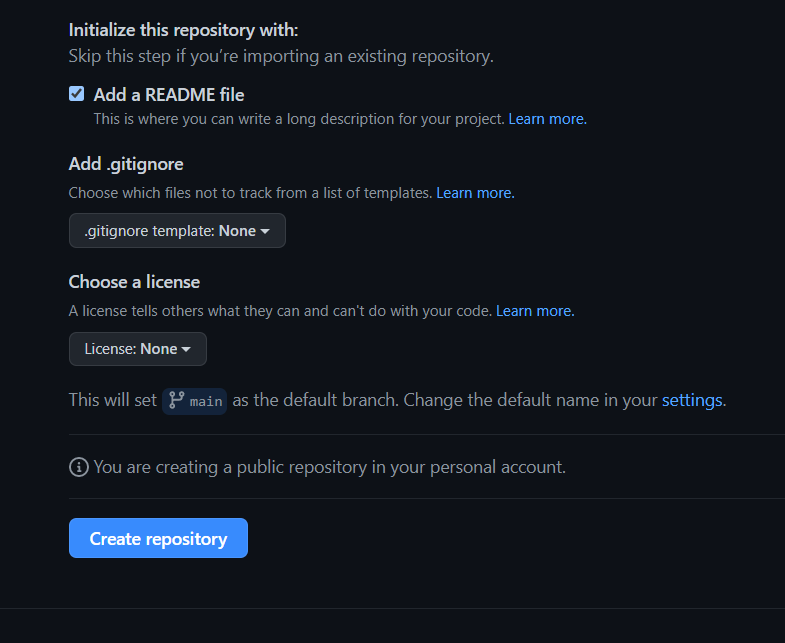
Step 2: Create a new repository by clicking on the ‘New’ icon on the top right corner.



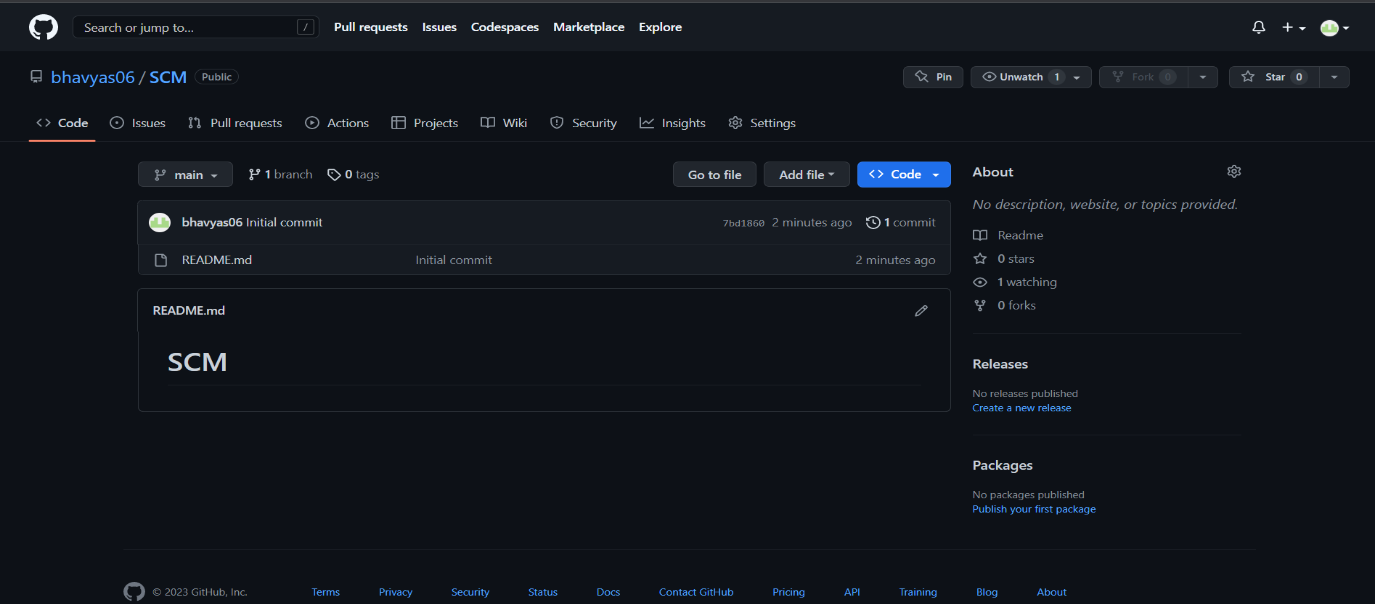
Step 2: Enter the repository name and add the description of the project.



Step 3: Select if you want the repository to be public or private and create a new repository.

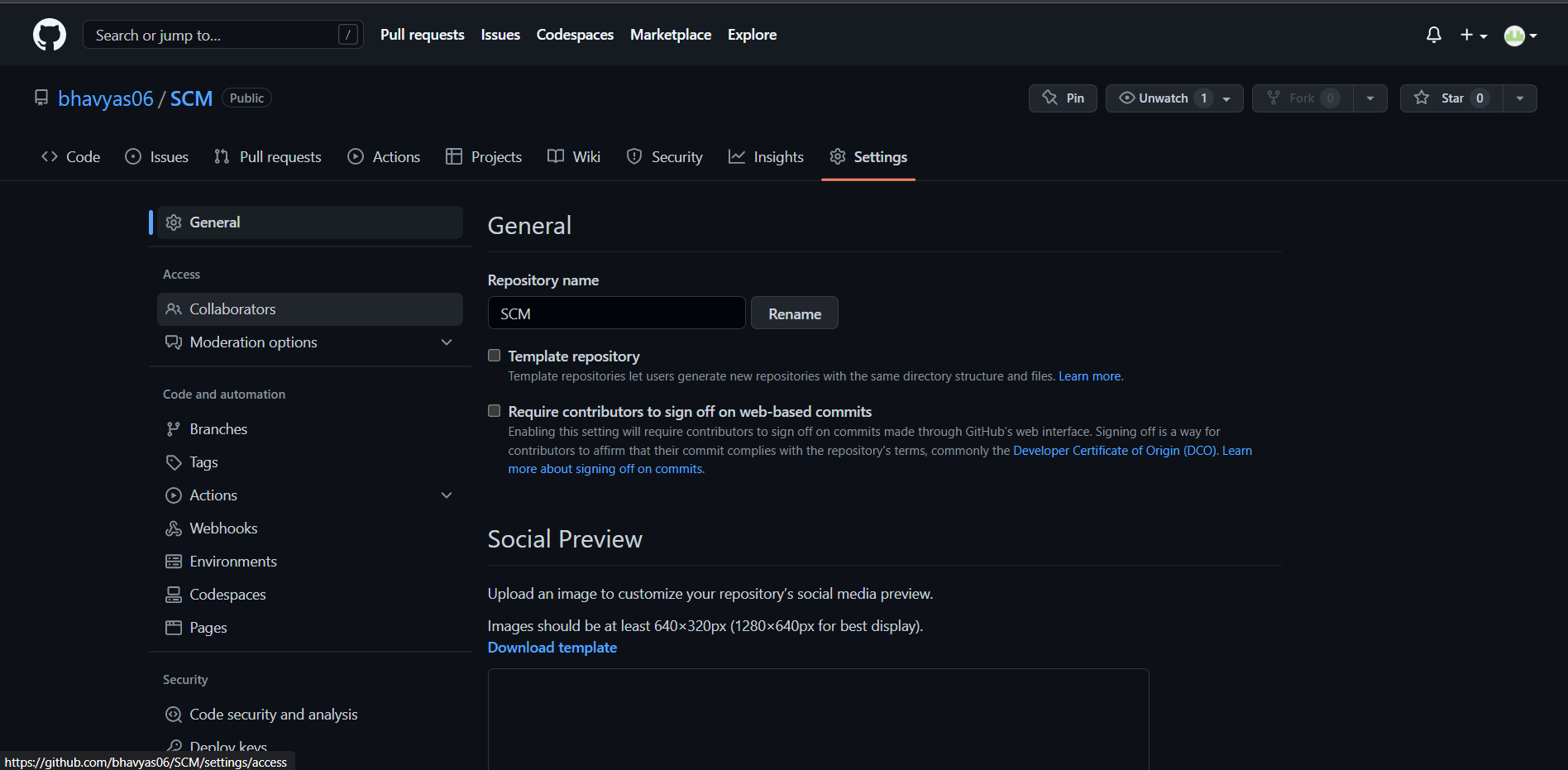


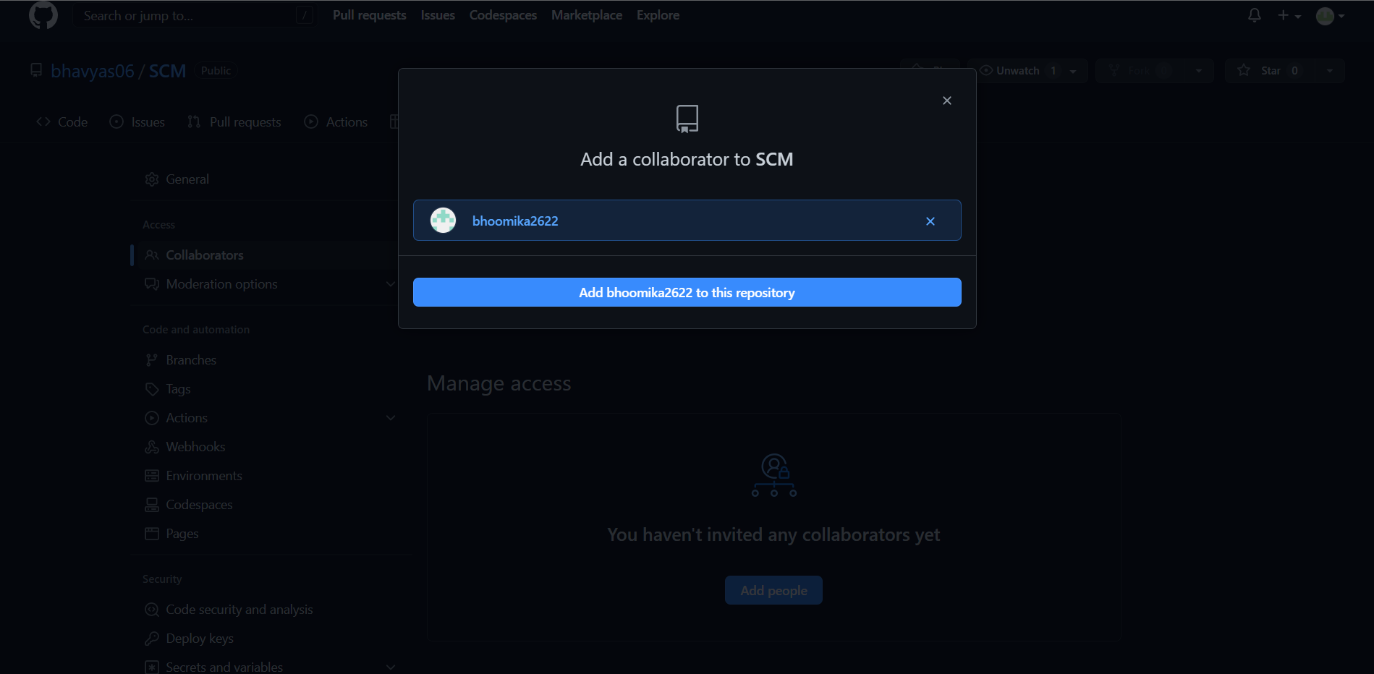
Step 4: Your new repository is available.



Step 5: Click on the ‘Settings’ tool and you will see a ‘Collaborators’ option on the left side.

Step 6: Click on that and now you can add your team members by writing their respective usernames.

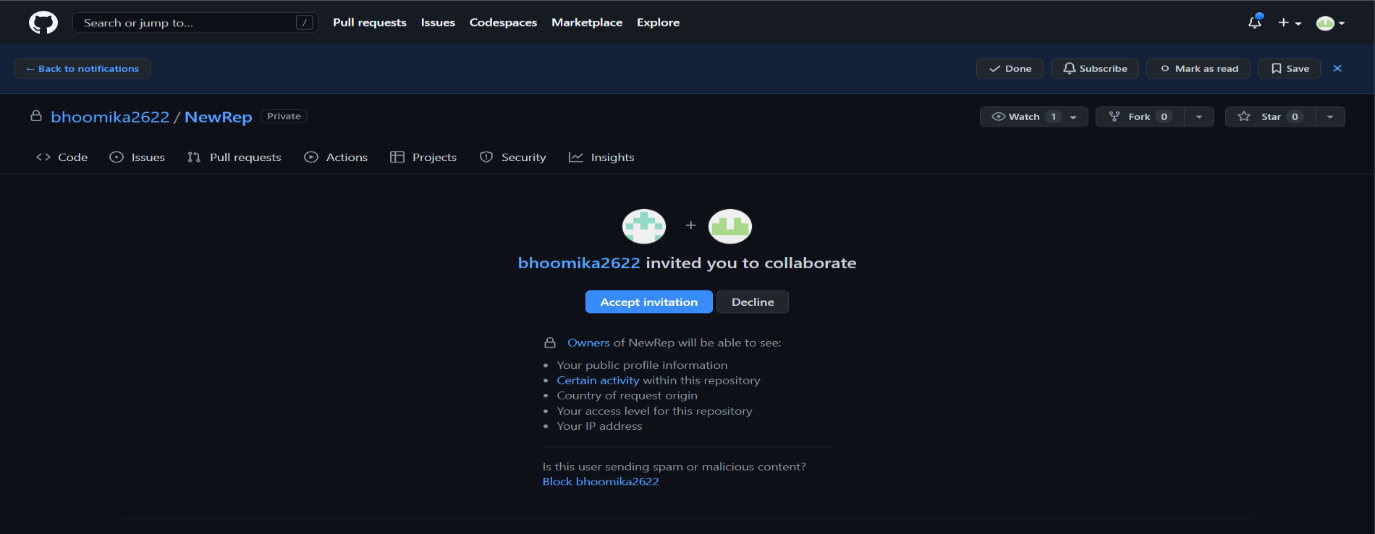






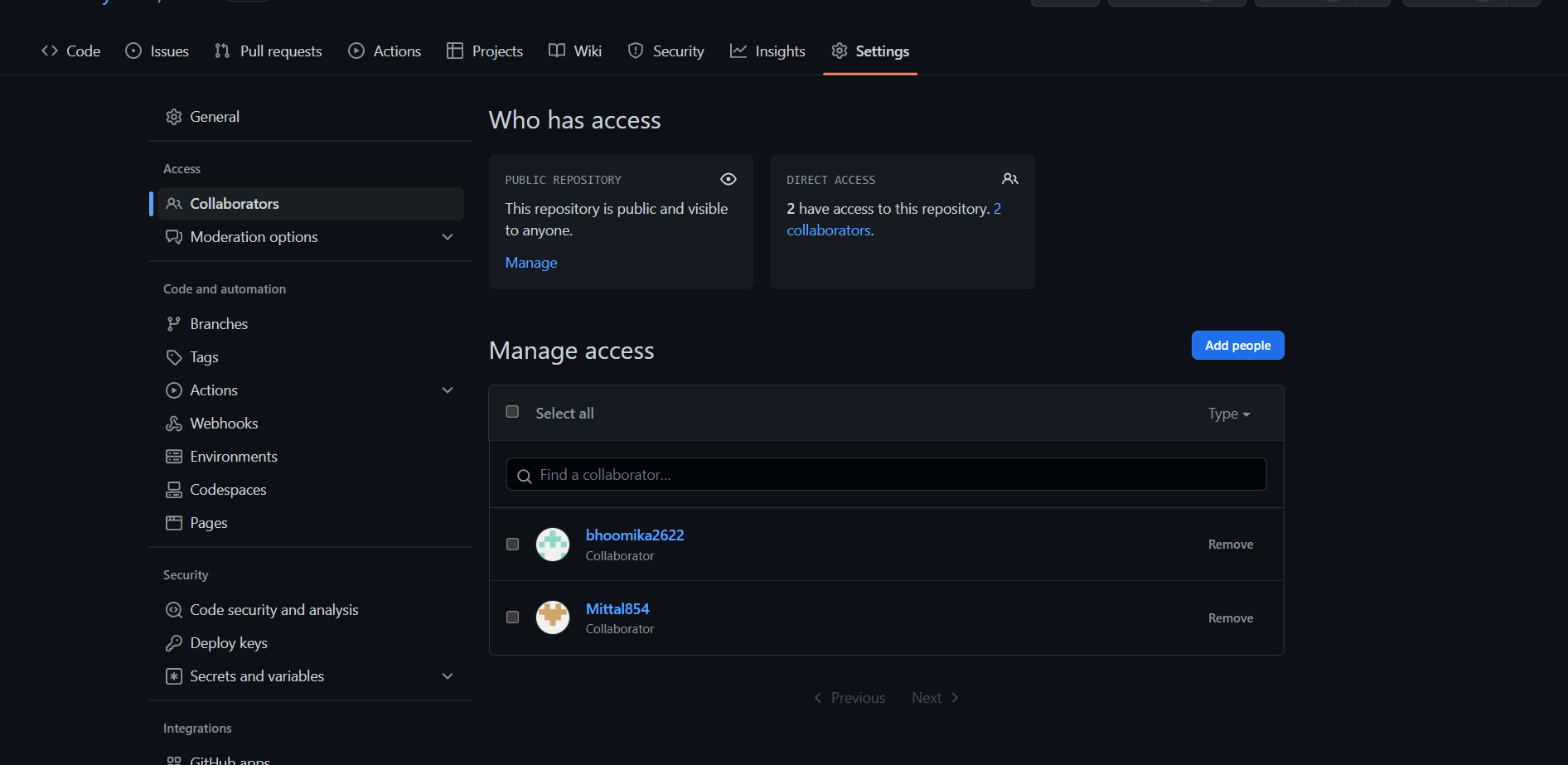
Step 7: Once you sent out the requests, they will receive the request in their ‘notifications’ tool at the top right corner and can either accept or decline the request.

Here’s how you will receive an invitation.



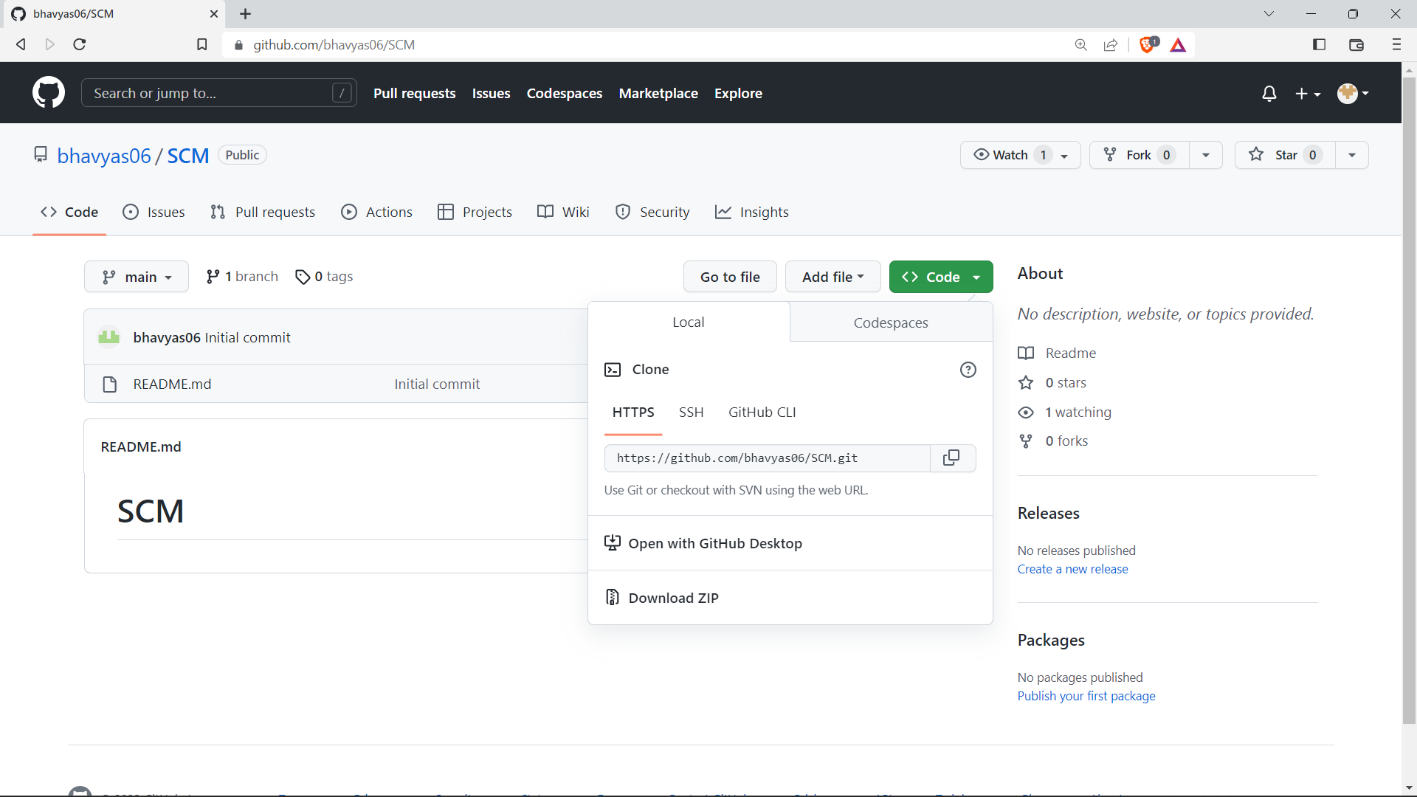
Step 8: You can also remove a collaborator.

Step 9: Once everyone is added, you can now start your project.



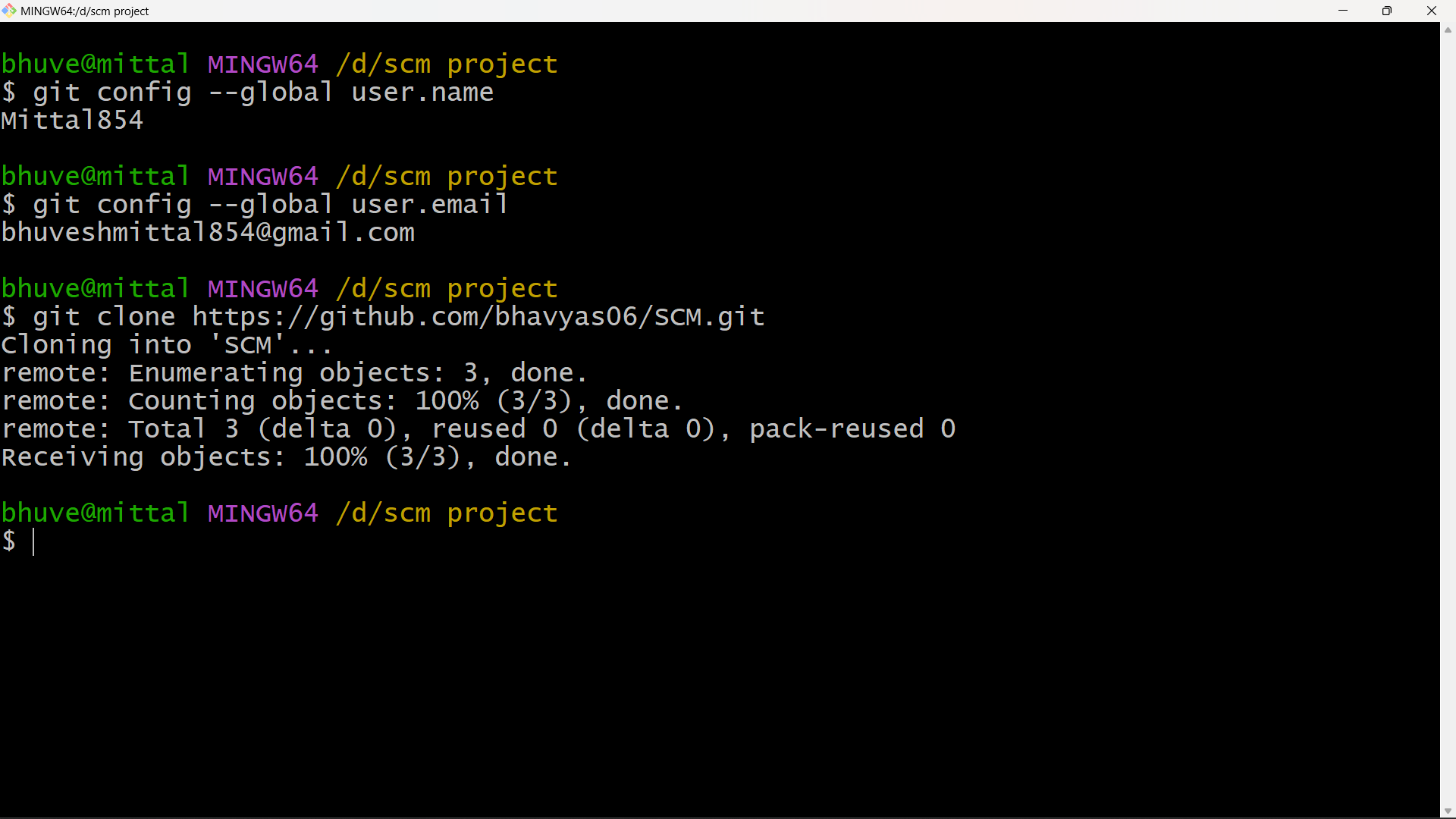
**Open and Close a Pull Request**

Step 1: Copy the ‘url’ of your created repository from the Code tool.

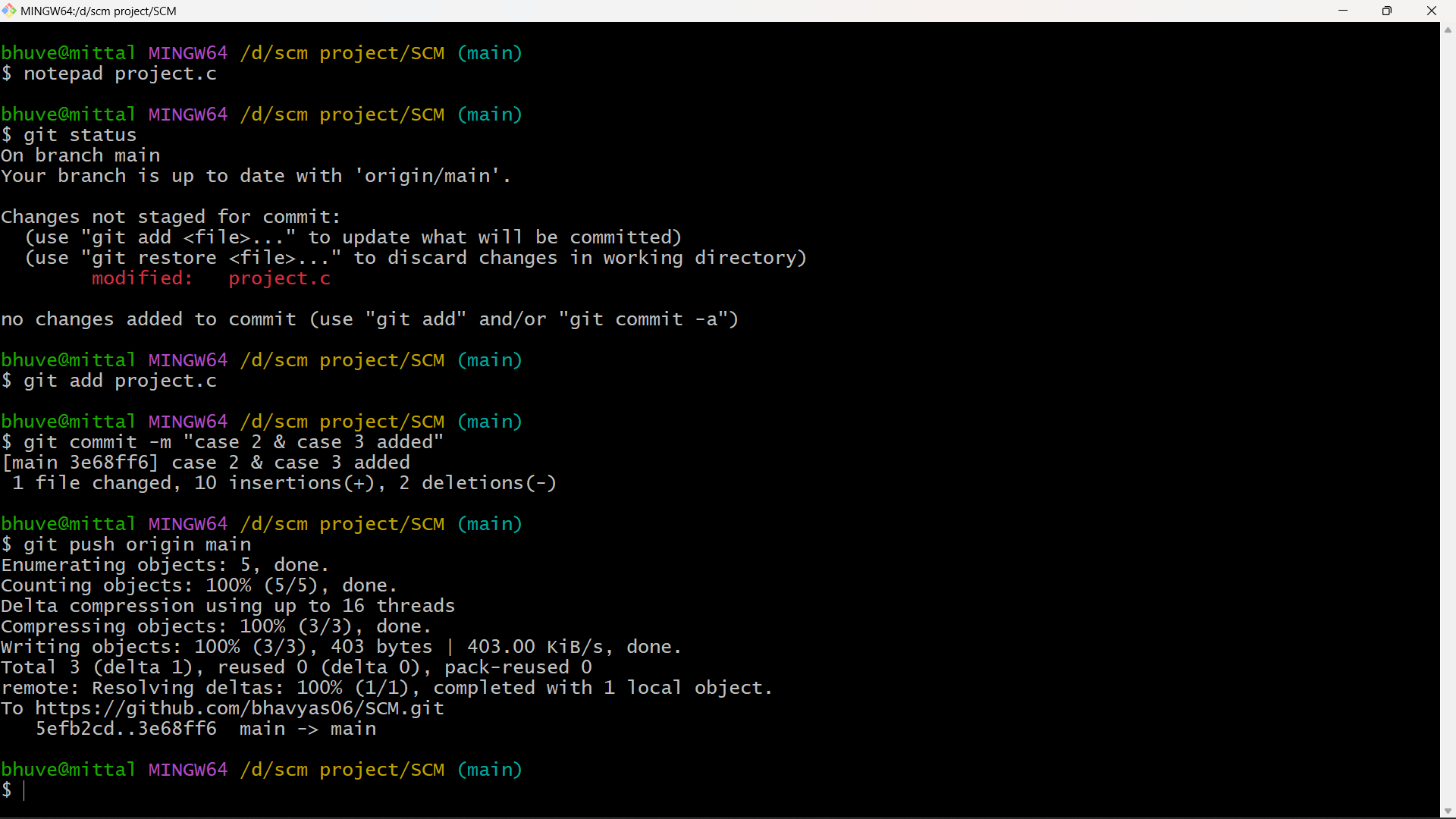


Step 2: Open Git Bash Here in your working directory.

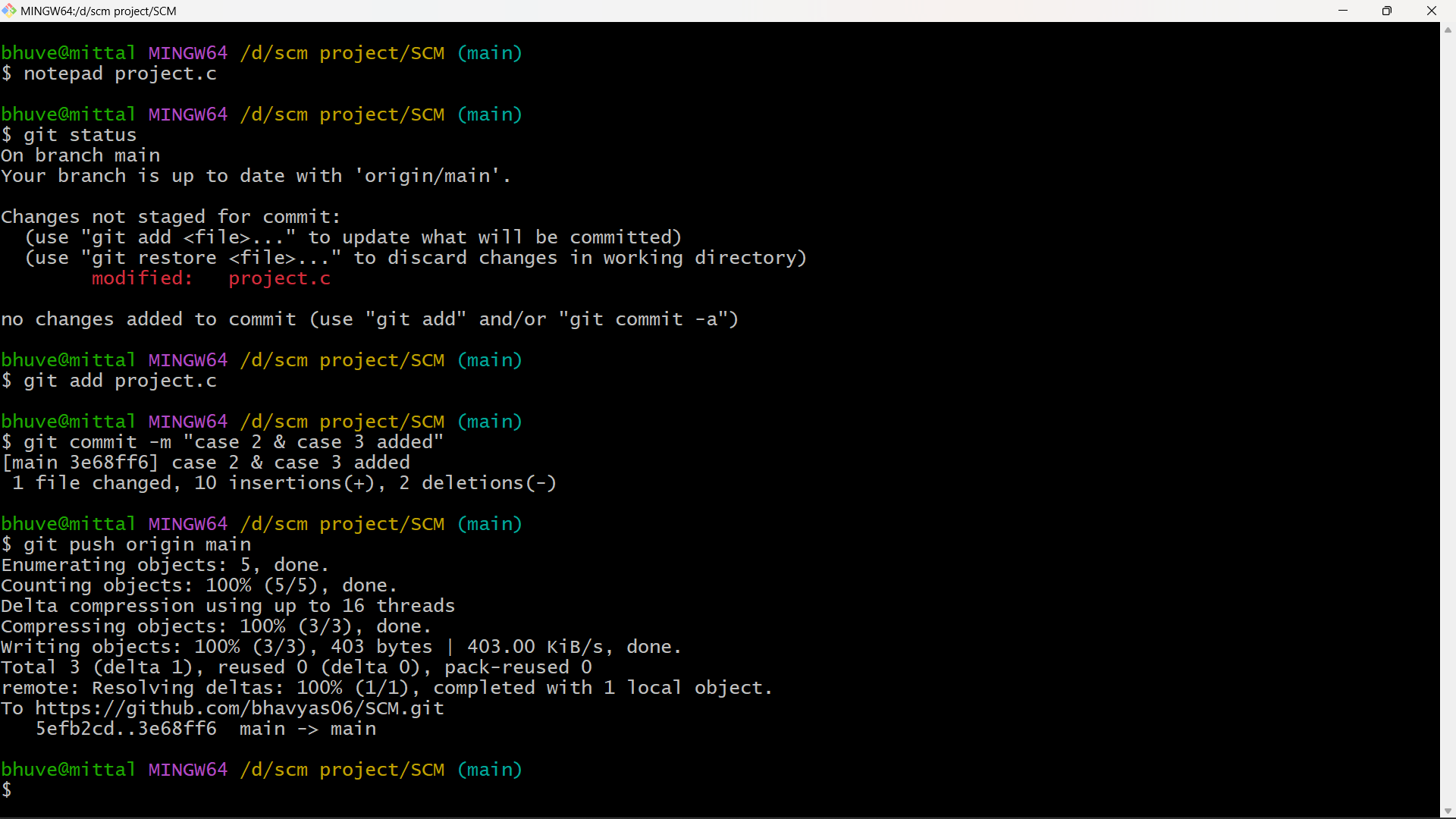
Step 3: Use the *$ git clone ‘url’* command to clone the repository into your local system.



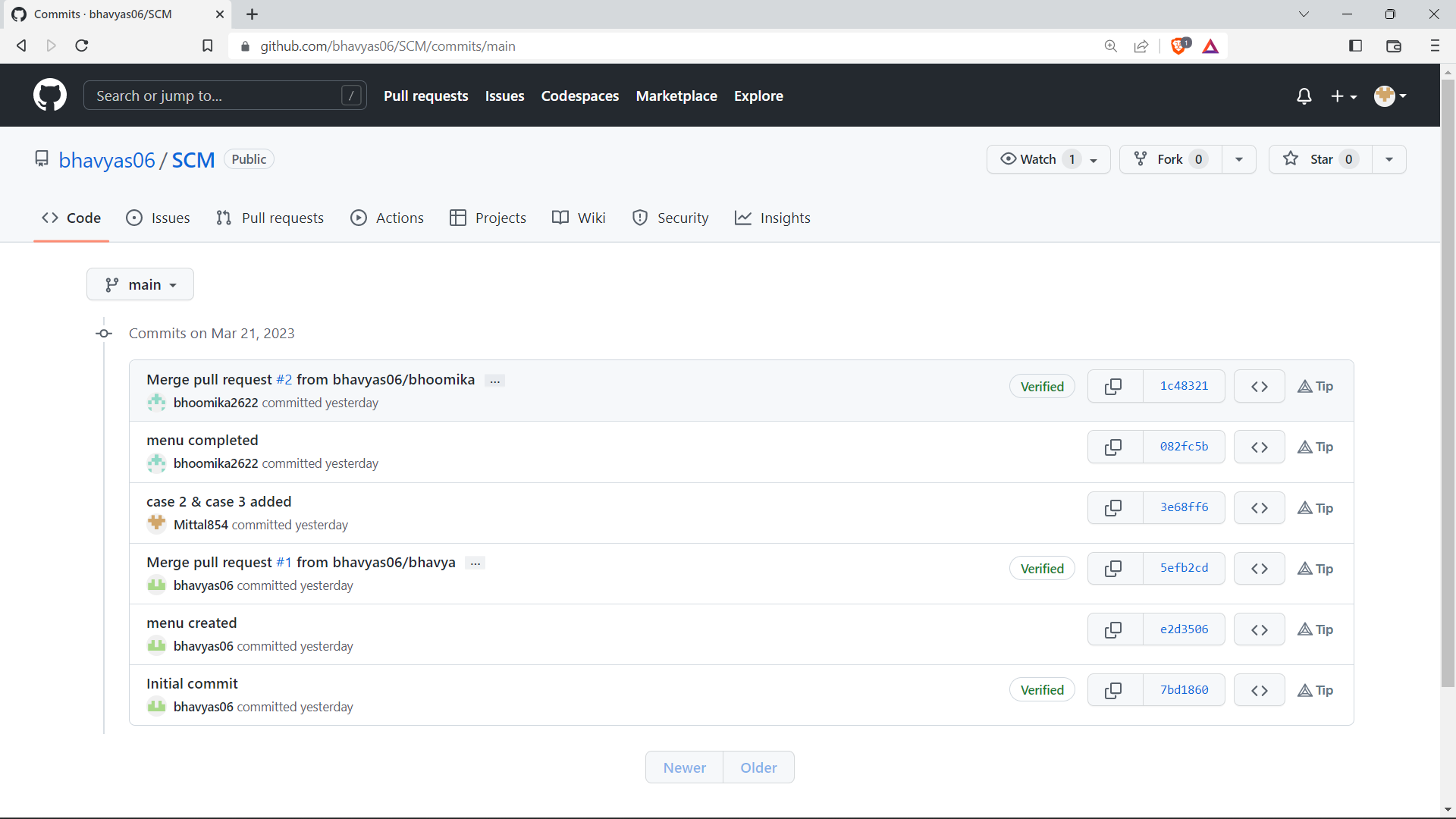
Step 4: Once the cloning is completed, Open the editor and make the changes, add them and commit them.



Step 5: After all your changes are done, use the *$ git push origin ‘branch name’* to push these changes into the GitHub repository.



Step 6: After this, our changes will be added in the main branch.



**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 graph gives 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 is using your repository and why they are using it.

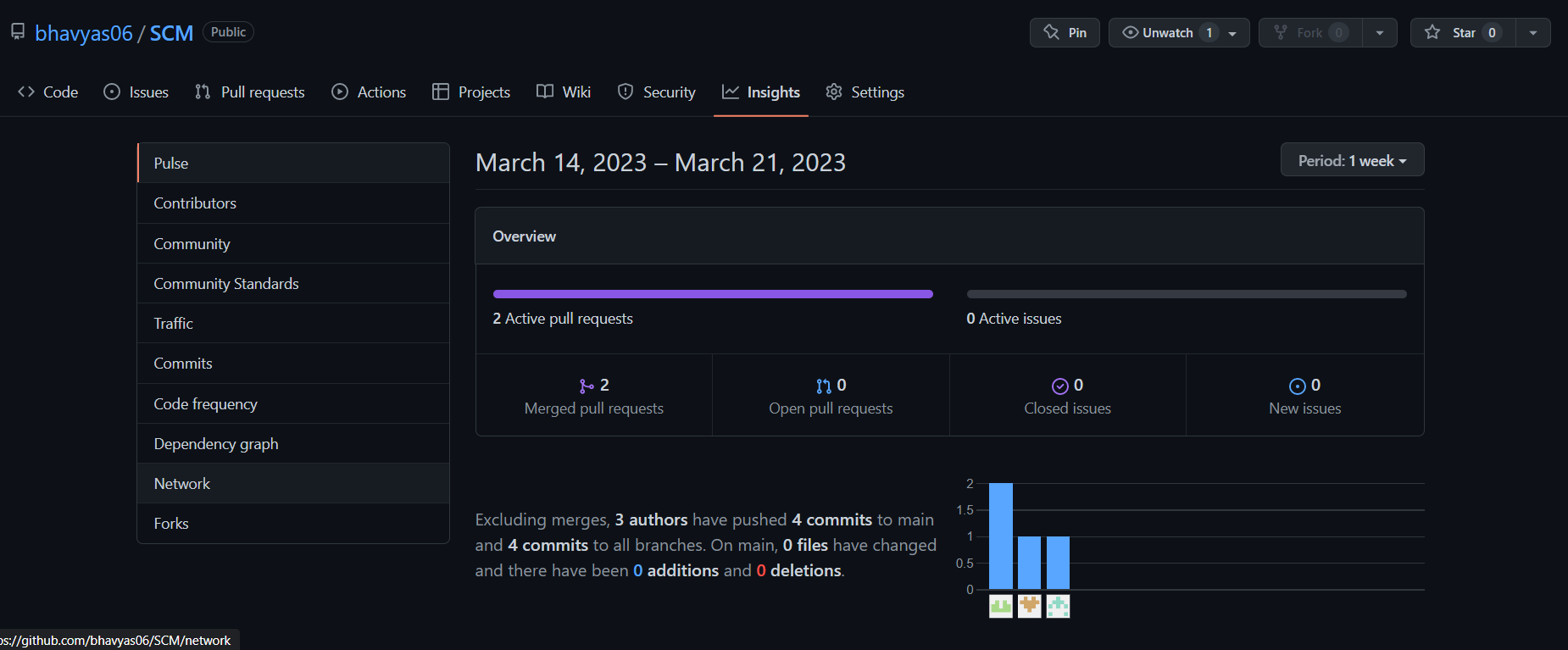
Here’s how you can access the network graphs of your respective repository--

Step 1: On GitHub.com, navigate to the main page of the repository.

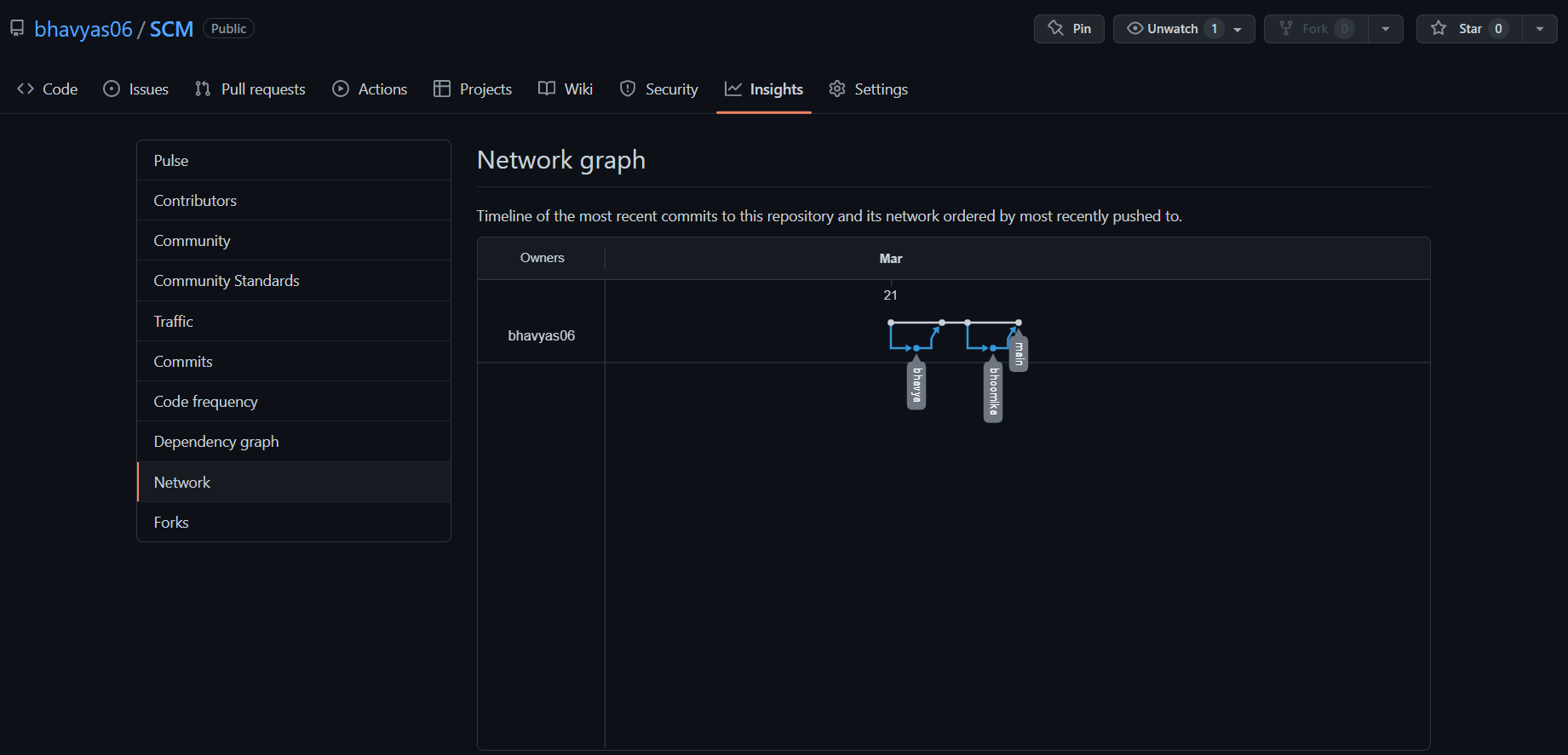


Step 2: Under your repository name, click on insights.

Step 3: At the left sidebar, click on Network.



Step 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 the 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.

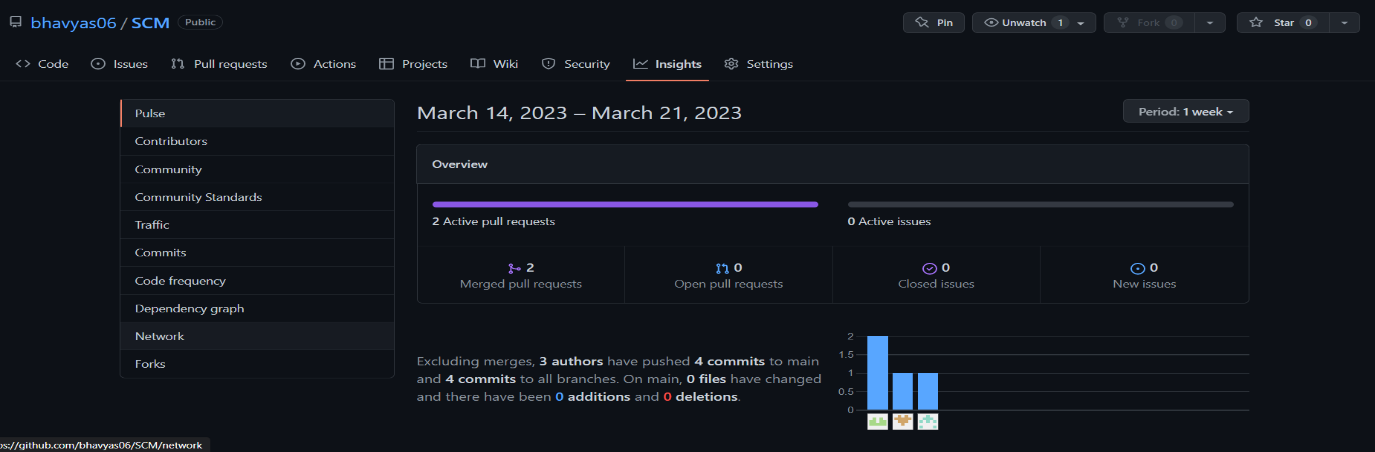
Clicking the number of forks shows you the full network. From there you can click ‘members’ to see who forked the repo.

Step 1: On GitHub.com, navigate the main page of your repo.

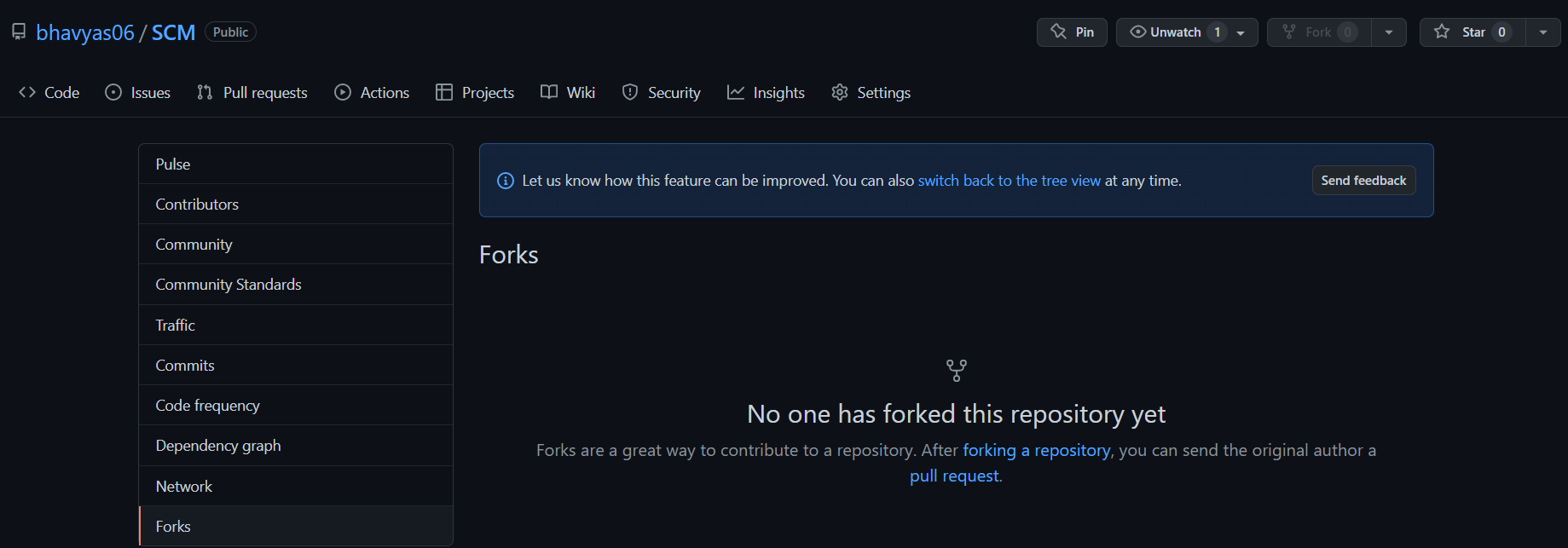


Step 2: Under your repo name, click on Insights.

Step 3: At the left sidebar, click on Forks.



Step 4: Here you can see all the forks.



Viewing the Dependencies of a Repository—

You can use the dependency graph to explore the code of 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.

REFERENCE

To complete this project, we took the help of Git Documentation.

Also, the book, Pro Git written by Scott Chacon was used.