

A Project report  
of  
**Source Code Management**  
(CS181)

Submitted by

**AREEN SIWACH**

**2110990252**

**G8-A**



**Department of Computer Science & Engineering**

Chitkara University Institute of Engineering and Technology, Punjab

(Jan- June 2022)

Institute Name: **Chitkara University Institute of Engineering and Technology**

Department Name :

**Department of Computer Science & Engineering**

Programme Name:

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

Course Name :**Source Code Management**

Session :**2021-22**

Course Code: **CS181**

Semester/Batch: **2<sup>nd</sup>/2021**

Vertical Name: **Beta**

Group No: **G08-A**

Coordinator: **Dr.Neeraj Singla**

Faculty Name: **Dr. Monit Kapoor**

**INDEX**

<b>S. NO</b>	<b>Experiment Name</b>	<b>Slide No.</b>
1.	Task 1.1	4-20
2.	Task 1.2	21-27
3.	Task2	27-33

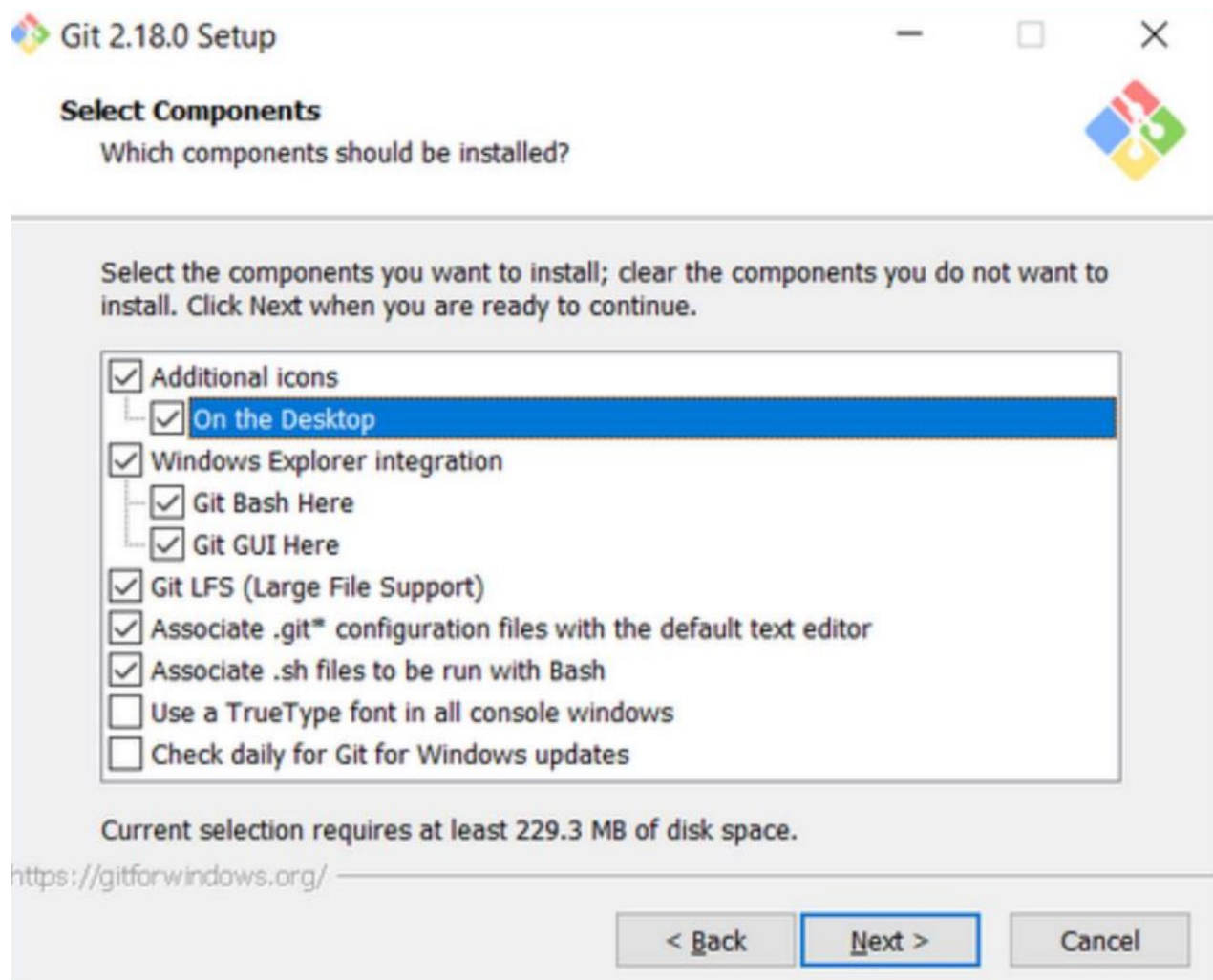
## Task 1.1 [Experiment No. 01]

**Aim:** Setting up the git client.

Git Installation: Download the Git installation program (Windows, Mac, or Linux) from [Git - Downloads \(git-scm.com\)](https://git-scm.com/downloads).

At first, a dialogue box will appear which will ask for the permission to make changes on your computer and will be required to click on yes. After that the installer will start but before that a few more dialogue boxes will appear, you can accept the default selections, except in the screens below where you do not want the default selections:

In the Select Components screen, make sure to select the most preferable option as shown:

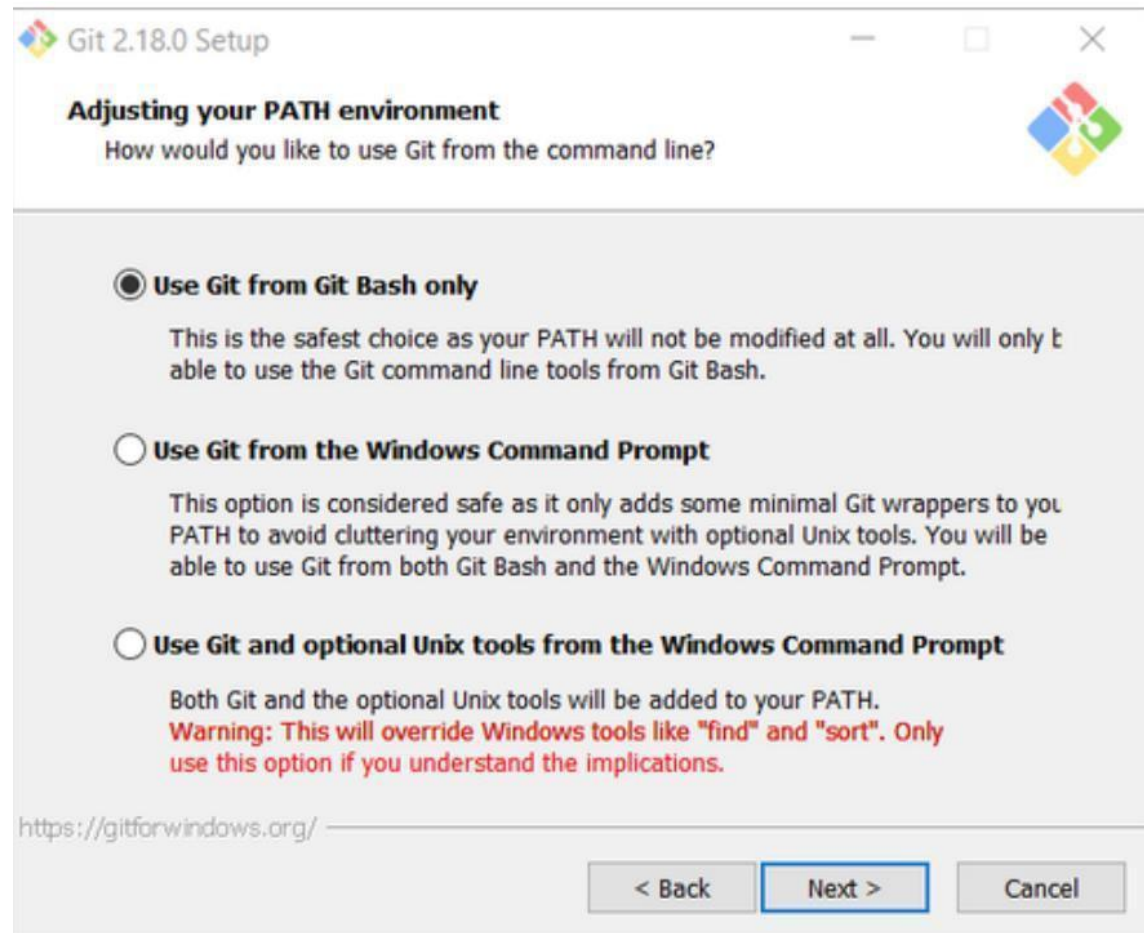


In the choosing the default editor is used by Git dialog, it is strongly recommended that you

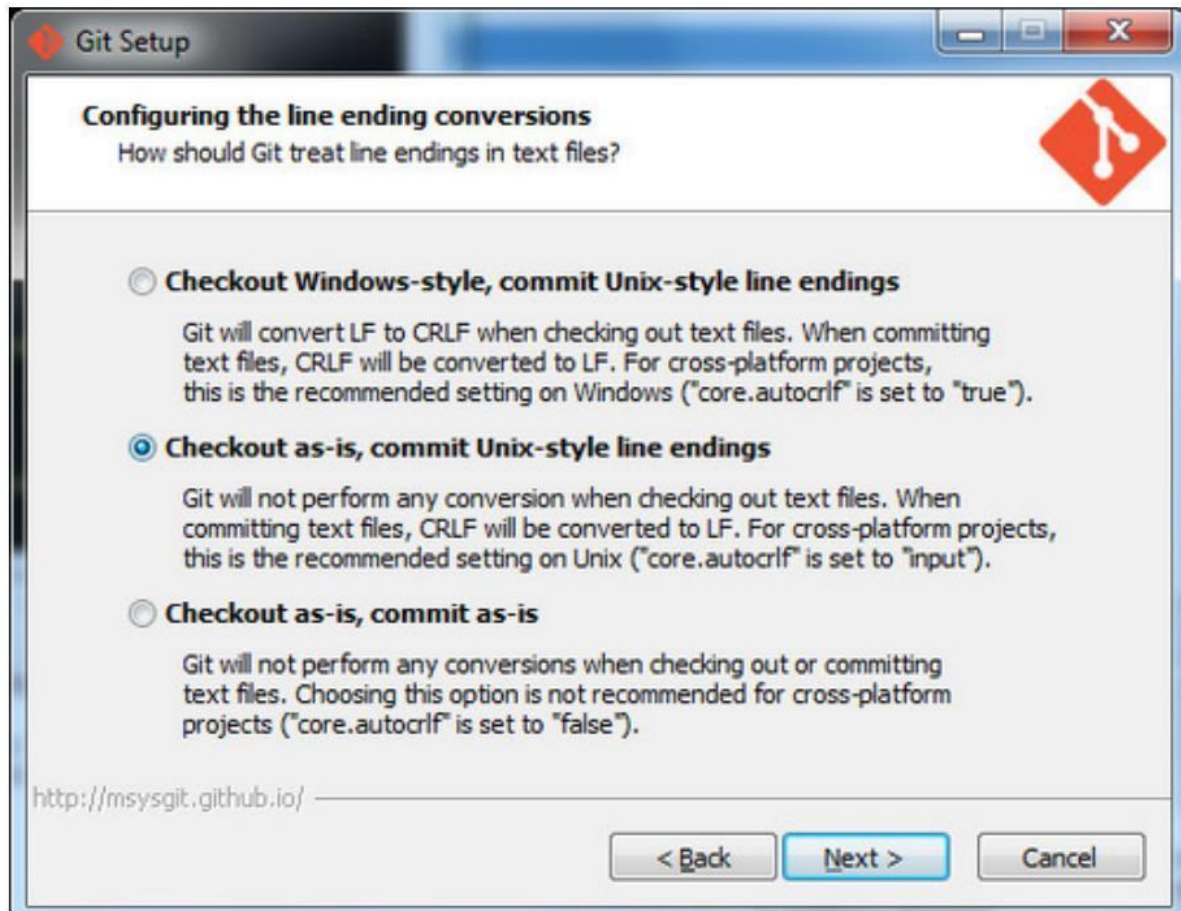
DO NOT select the default VIM editor. It is better to use Notepad++ or Nano.

In the Adjusting your PATH screen, all three options are acceptable:

1. Use Git from Git Bash only: no integration, and no extra command in your command path.
2. Use Git from the windows Command Prompt: add flexibility – you can simply run git from a windows command prompt, and is often the setting for people in industry – but this does add some extra commands.
3. Use Git and optional Unix tools from the Windows Command Prompt: this is also a robust choice and useful if you like to use Unix like commands.



In the Configure the line ending screen, select the middle option (Checkout-as-is, commit Unix-style line endings) as shown. This helps migrate files towards the Unix-style (LF) terminators that most modern IDE's and editor's support. The Windows convention (CR-LF line termination) is only important for Notepad.



Configuring Git to ignore certain files:

This part is extra important and required so that your repository does not get cluttered with garbage files.

By default, Git tracks **all** files in a project. Typically, this is **not** what you want but you want Git to ignore certain files such as .exe files created by an editor or .class files created by the Java compiler. To have Git automatically ignore particular files, create a file named .gitignore ( note that the filename begins with a dot).

Edit this file and add the lines below (just copy/paste them from this screen); these are patterns for files to be ignored (taken from examples provided at <https://github.com/github/gitignore>.)

```
1  # Prerequisites
2  *.d
3
4  # Compiled Object files
5  *.slo
6  *.lo
7  *.o
8  *.obj
9
10 # Precompiled Headers
11 *.gch
12 *.pch
13
14 # Compiled Dynamic libraries
15 *.so
16 *.dylib
17 *.dll
18
19 # Fortran module files
20 *.mod
21 *.smod
22
23 # Compiled Static libraries
24 *.lai
25 *.la
26 *.a
27 *.lib
28
29 # Executables
30 *.exe
31 *.out
32 *.app
```



Once Git is installed, there is some remaining custom configuration you must do. Follow the steps below:

a. From within File Explorer, right-click on any folder. A context menu appears containing the commands "Git Bash here" and "Git GUI here". These commands permit you to launch either Git client. For now, select Git Bash here. (Where name is your MOSE as login name)

b. Enter the command (replacing name as appropriate) `git config -  
- global core.excludesfile c:/users/name/.  
gitignore`

This tells Git to use the .gitignore file you created in step 2

c. Enter the command `git config --global user.  
Email "name@mose.edu"`

This links your Git activity to your email address. Without this, your commits will often show up as "unknown login". Replace name with your own email name.

d Enter the command `git config --global user.name "Your Name"`

Git uses this to log your activity. Replace "Your Name" by your actual first and last name.

e. Enter the command `git config --global push.default simple`

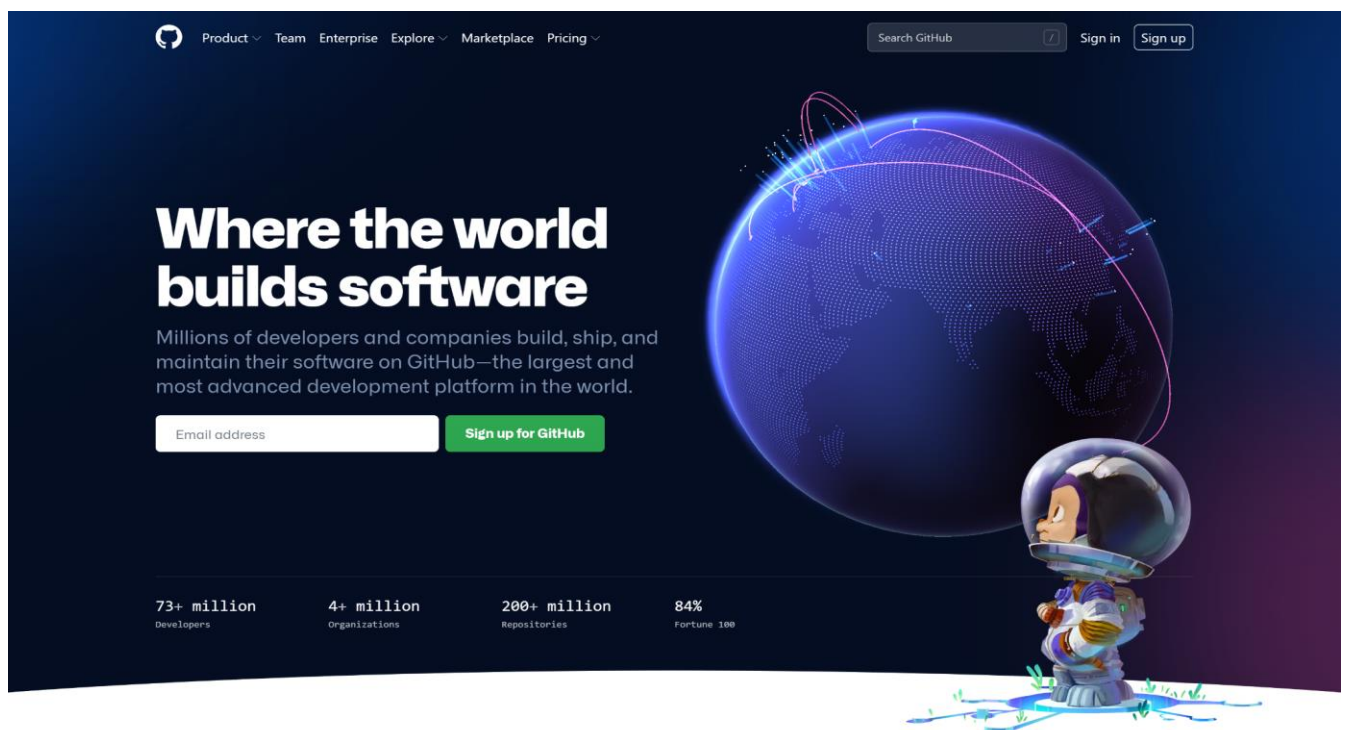
This ensures that all pushes go back to the branch from which they were pulled. Otherwise pushes will go to the master branch, forcing a merge.

## Experiment No. 02

### **Aim:** Setting up GitHub Account

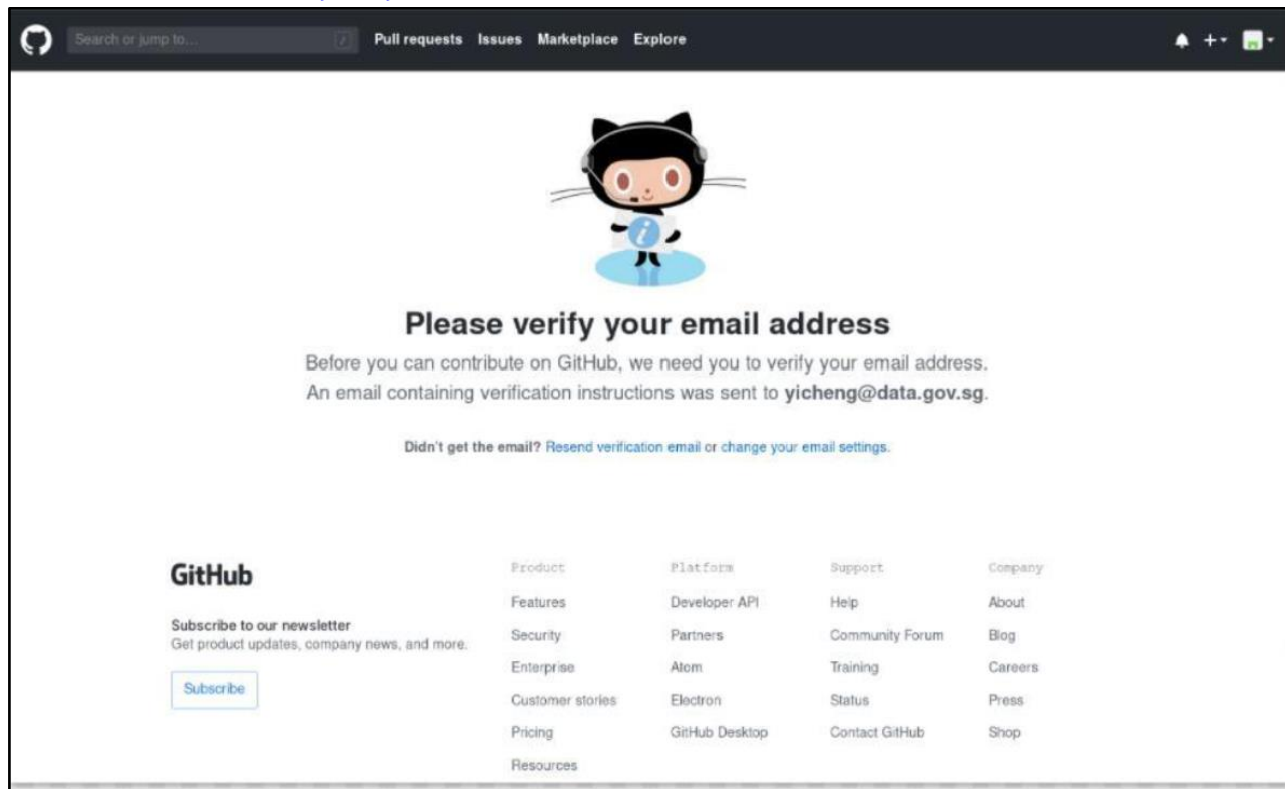
The first step in starting with GitHub is to create an account, choose a product that fits your needs best, verify your email, set up two-factor authentication, and view your profile. There are several types of accounts on GitHub. Every person has their own user name, which can be part of multiple organizations and teams. Your user account is your identity on GitHub.com and represents you as an individual.

- 1. Creating an account:** To sign up for an account on GitHub.com, navigate to <https://github.com/> and follow the prompts.



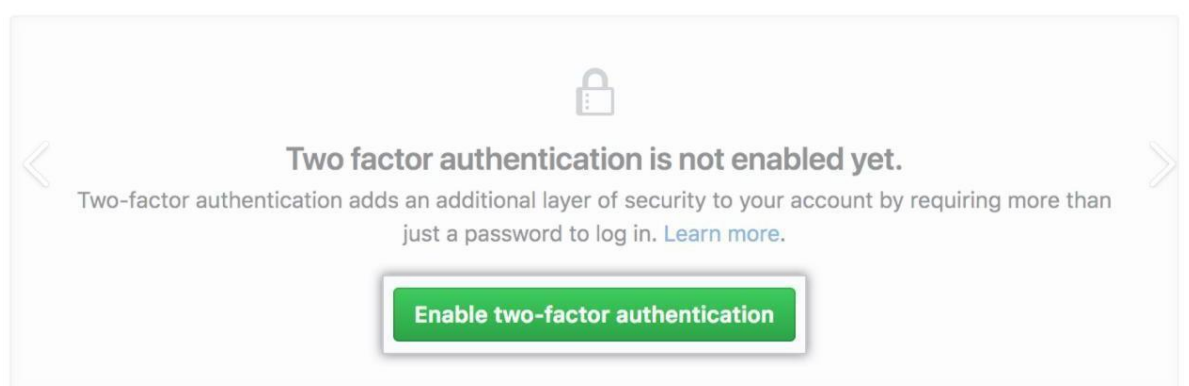
- 2. Choosing your GitHub product:** You can choose GitHub Free or GitHub Pro to get access to different features for your personal account. You can upgrade at any time if you are unsure at first which product you want. For more information on all GitHub's plans, see "[GitHub's products](#)".

- 3. Verifying your email address:** To ensure you can use all the features in your GitHub plan, verify your email address after signing up for a new account. For more information, see “[Verifying your email address](#)”.



- 4. Configuring two-factor authentication:** Two-factor authentication, or 2FA, is an extra layer of security used when logging into websites or apps. We strongly urge you to configure 2FA for the safety of your account. For more information, see “[About two-factor authentication](#).”

## Two-factor authentication



**5.Viewing your GitHub profile and contribution graph:** Your GitHub profile tells what you have done or have made changes to which repositories and gits you’ve pinned, the organization memberships you’ve chosen to publicize, the contributions and the projects you’ve made. For more information, see [“Viewing contributions on your profile.”](#)



**Aim:** Program to generate logs

Basic Git init:

Git init command creates a new Git repository. It can be used to convert an existing, undersigned project to a Git repository or initialize a new, empty repository. Most other Git commands are not available outside of an initialized repository, so this is usually the first command you'll run in a new project.

Git status:

The git status command displays the state of the working directory and the staging area. It lets you see which changes have been staged, which haven't, and which files aren't being tracked by Git. Status output does not show you any information regarding the committed project history.

Basic Git commit:

The git commit command captures a snapshot of the project's currently staged changes. Committed snapshots can be thought of as "safe" versions of a project -Git will never change them unless you explicitly ask it to.

Git Add Command:

The git add command adds a change in the working directory to the staging area. It tells Git that you want to include updates to a particular file in the next commit. However, git add doesn't really affect the repository in any significant way—changes are not actually recorded until you run git commit.

## Basic Git log

Git log command is one of the most common commands of git. It is the most useful command for Git. Every time you need to check the history of commits, you have to use the git log command. The basic git log command will display the most recent commits and the status of the head. It will use as:

MINGW64:/c/GitHub

```
areen@DESKTOP-C90IB9M MINGW64 /c/GitHub (master)
$ git log
commit d6db4e3789e1caad6e530bfacb9bbd075ec01ea9 (HEAD -> master, origin/master)
Author: AreenSwiach <areen0252.be21@chitkara.edu.in>
Date: Sun Apr 10 17:46:12 2022 +0530

    Reverse a Number

commit 165d4afff679753b58bddc118d6131a8a5244356
Author: AreenSwiach <areen0252.be21@chitkara.edu.in>
Date: Sun Apr 10 17:37:56 2022 +0530

    Program

commit 5086967d1b662fe79dd38c9a049144c658a4bc80
Author: AreenSwiach <areen0252.be21@chitkara.edu.in>
Date: Sun Apr 10 16:37:06 2022 +0530

    Finding a Factorial

areen@DESKTOP-C90IB9M MINGW64 /c/GitHub (master)
$
```

**Aim:** Create and visualize branches in Git

### How to create branches?

The main branch in git is called 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 files which are created in branch are not shown in master branch. We also can merge both the parent(master) and child (other branches).

1. For creating a new branch: git branch "name of branch"
2. To check how many branches, we have: git branch
3. To change the present working branch: git checkout "name of the branch"

### Visualizing Branches:

To visualize, we have to create a new file in the new branch "feature1" instead of the master branch. After this we have to do three step architecture i.e., working directory, staging area and git repository.

After this I have done the 3 Step architecture which is tracking the file, sending it to the staging area and finally we can rollback to any previously saved version of this file.

After this we will change the branch from feature1 to master, but when we switch back to master branch the file we created i.e., "hello" will not be there. Hence the new file will not be shown in the master branch. In this way we can create and change different branches. We can also merge the branches by using the git merge command.

In this way we can create and change different branches. We can also merge the different branches by using the git merge command.

MINGW64/c/GitHub

```
areen@DESKTOP-C90IB9M MINGW64 /c/GitHub (master)
$ git branch feature1

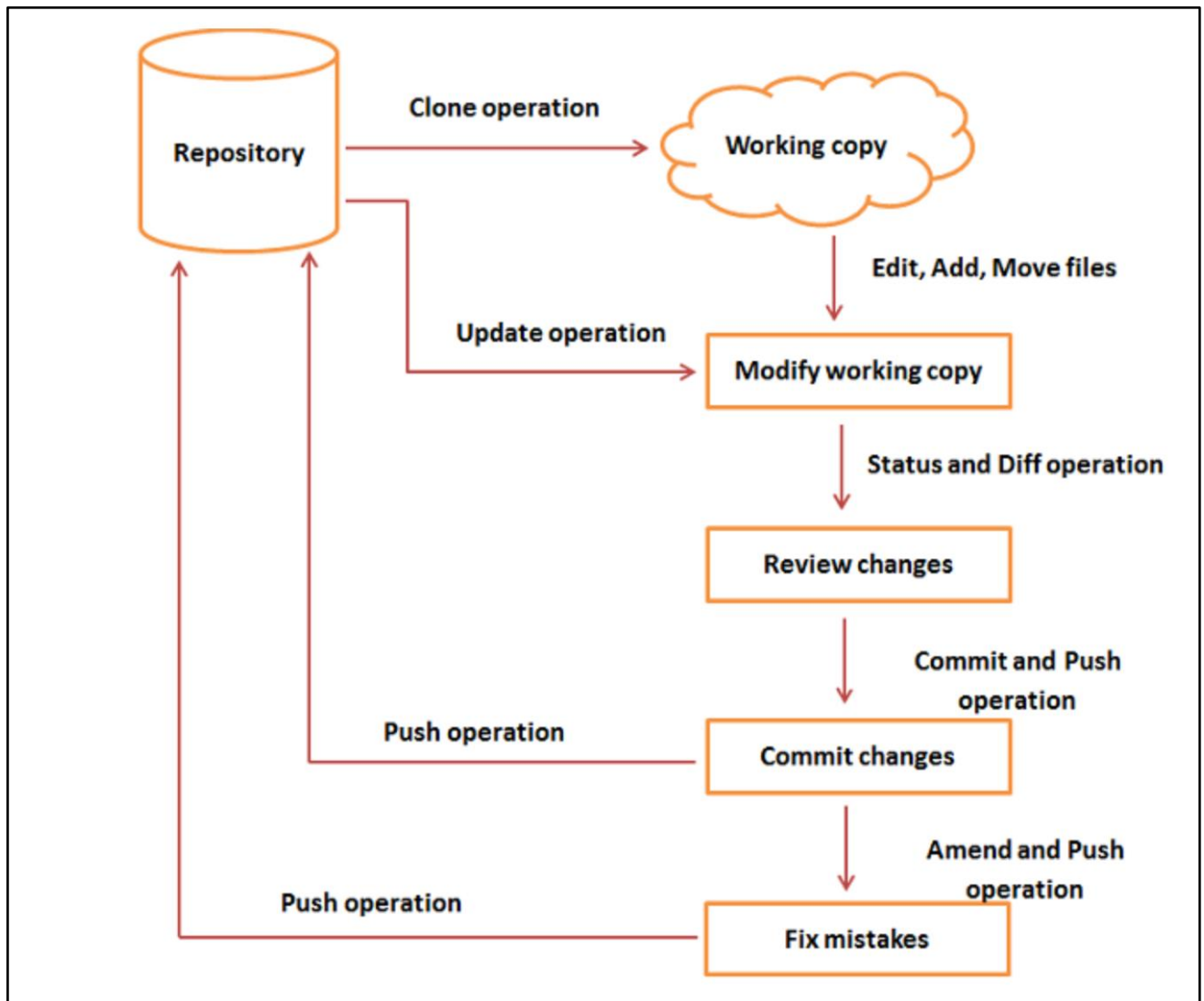
areen@DESKTOP-C90IB9M MINGW64 /c/GitHub (master)
$ git checkout feature1
Switched to branch 'feature1'
D      Reverse.exe

areen@DESKTOP-C90IB9M MINGW64 /c/GitHub (feature1)
$ s
```



**Aim:** Git lifecycle description

Git is used in our day-to-day work, we use Git for keeping a track of our files, working in a collaboration with our team, to go back to our previous code versions if we face some error. Git helps us in many ways. Let us look at the Lifecycle description that git has and understand more about its life cycle. Let us see some of the basic steps that we have to follow while working with Git-

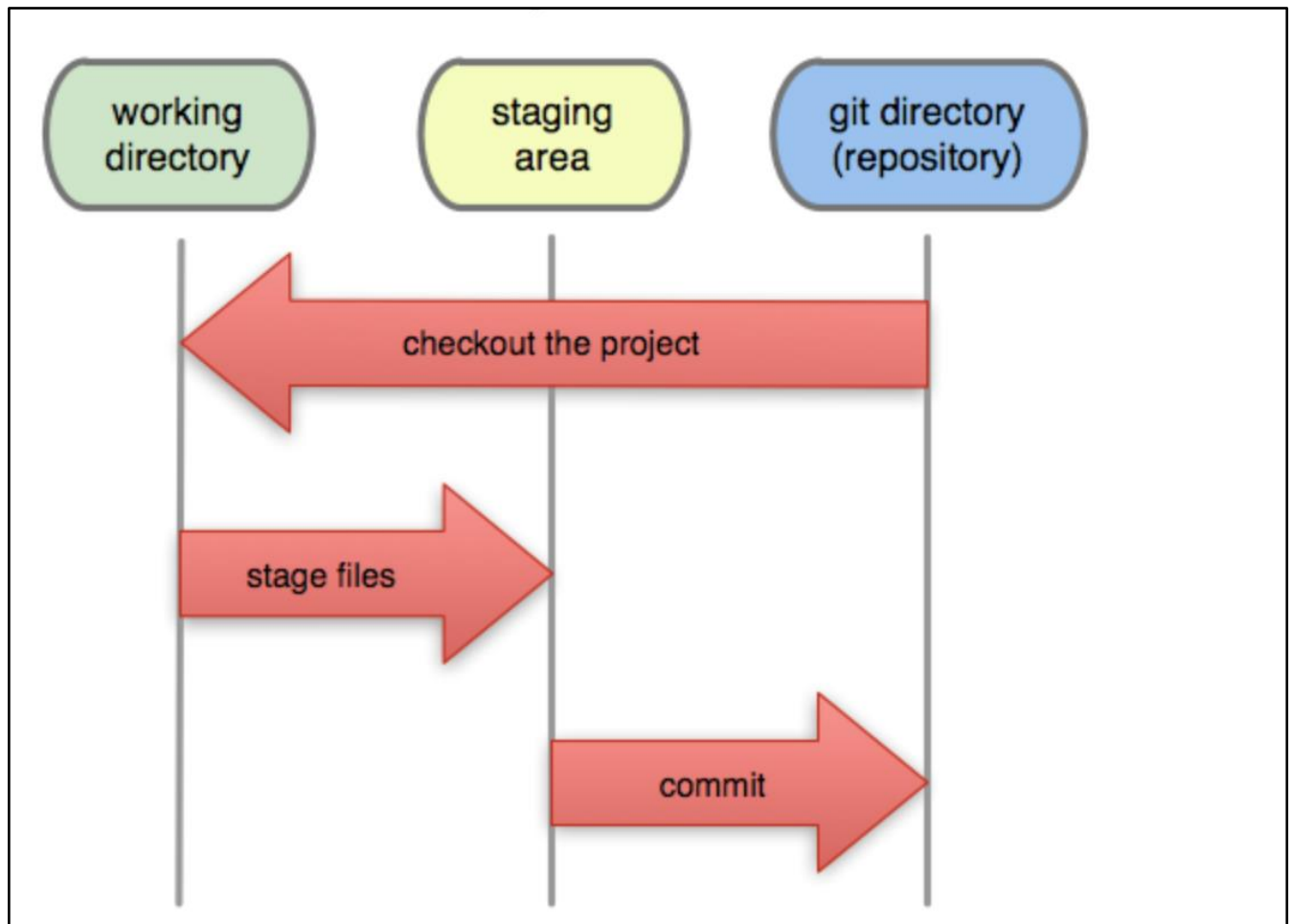


- **Step 1-** We first clone any of the code residing in the remote repository to make our own local repository.
- **Step 2-** We edit the files that we have cloned in our local repository and make the necessary changes in it.
- **Step 3-** We commit our changes by first adding them to our staging area and

committing them with a commit message.

- **Step 4 and Step 5-** We first check whether there are any of the changes done in the remote repository by some other users and we first pull those changes.
- **Step 6-** If there are no changes, we push our changes to the remote repository and we are done with our work.

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



### 1. Working Directory:

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

### 2. Staging Area:

Now, to track files the different versions of our files we use the command `git add`. We can term a staging area as a place where different versions of our files are stored. `git add` command copies the version of your file from your working directory to the staging area. We can, however, choose which files we need to add to the staging area because in our working directory there are some files that we don't want to get tracked, examples include node modules, temporary files, etc. Indexing in Git is the one that helps Git in understanding which files need to be added or sent. You can find your staging area in the `.git` folder inside the index file.

```
git add<filename>
```

```
git add.
```

### **3. Git Directory:**

Now since we have all the files that are to be tracked and are ready in the staging area, we are ready to commit the files using the `git commit` command. Commit helps us in keeping the track of the metadata of the files in our staging area. We specify every commit with a message which tells what the commit is about. Git preserves the information or the metadata of the files that were committed in a Git Directory which helps Git in tracking files; basically it preserves the photocopy of the committed files. Commit also stores the name of the author who did the commit, files that are committed, and the date at which they are committed along with the commit message. `git commit -m <Message>`. Then the `git push` command is used to upload local repository content to a remote repository. Pushing is how you transfer commits from your local repository to a remote repository. Then, it uploads or we can also say it is added to our account on GitHub.

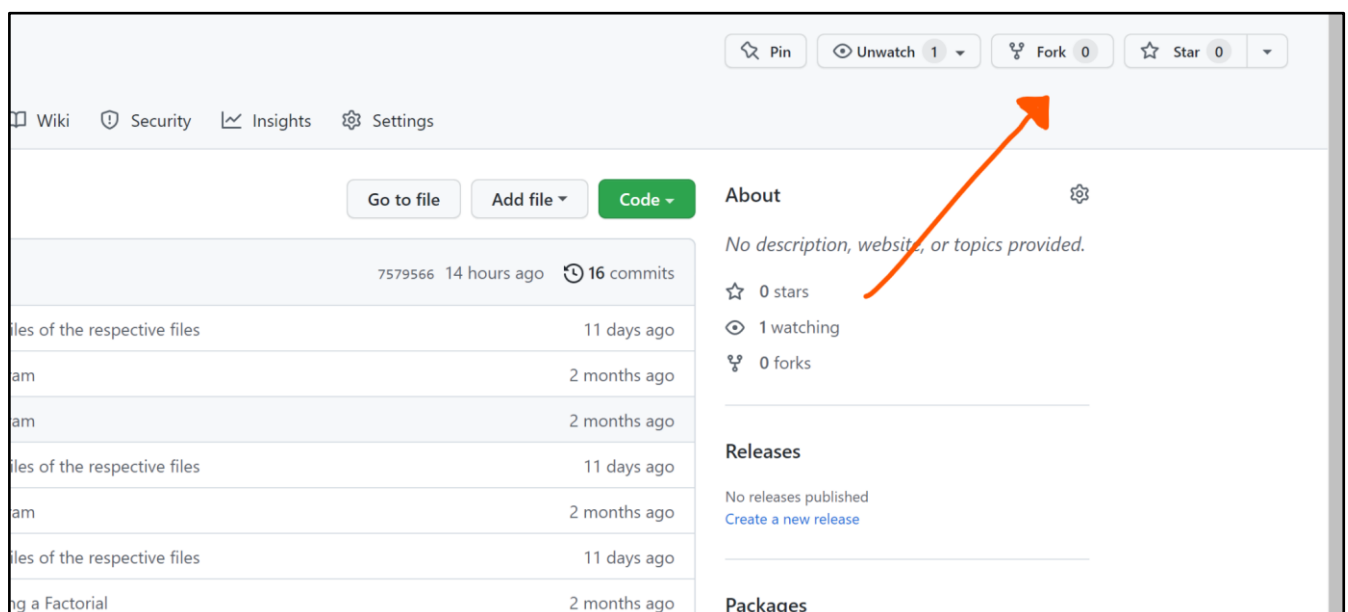
# Task 1.2

## Aim: Fork And Commit

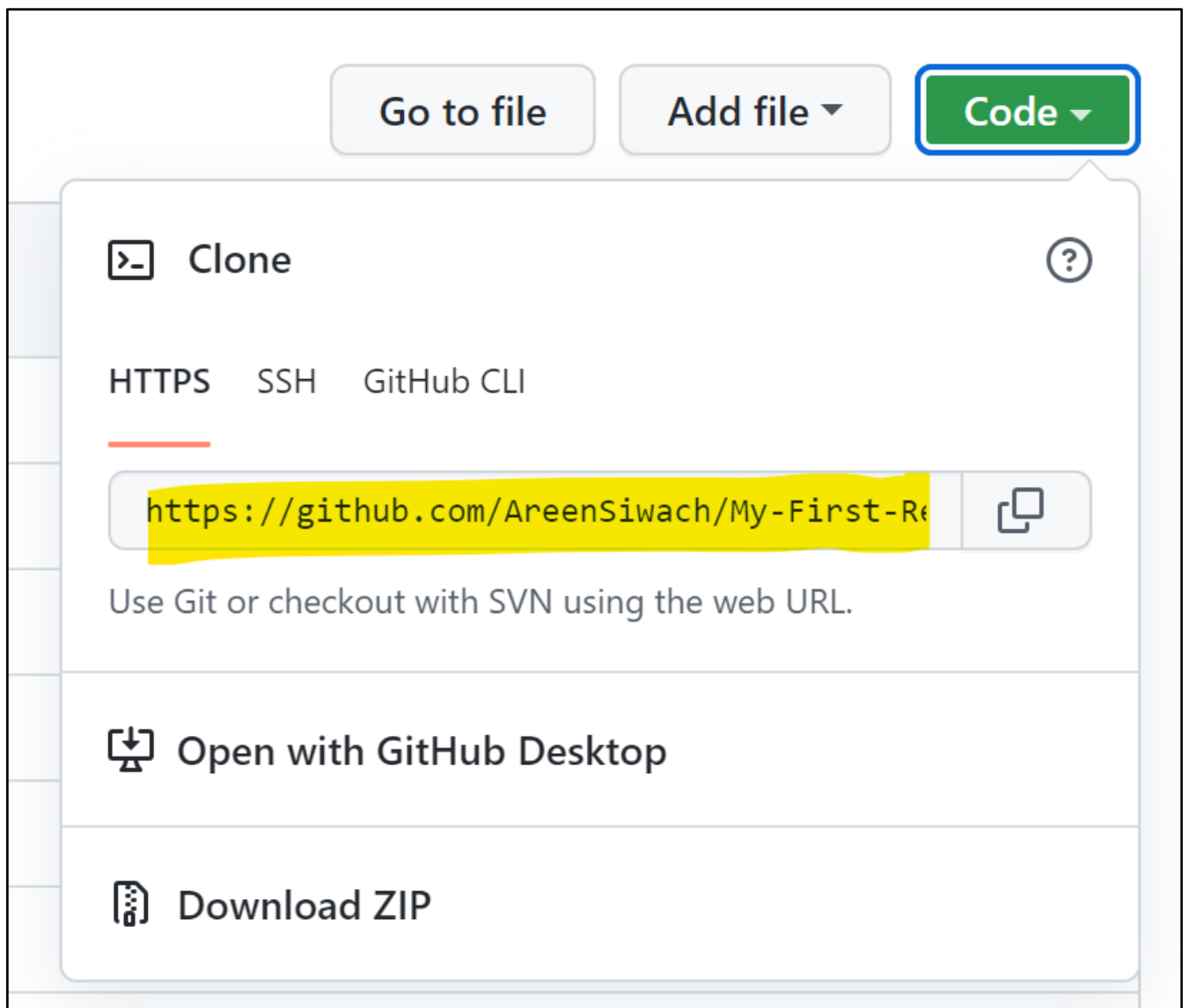
A fork is a copy of a repository that you manage. Forks let you make changes to a project without affecting the original repository. You can fetch updates from or submit changes to the original repository with pull requests. Most commonly, forks are used to either propose changes to someone else's project or to use someone else's project as a starting point for your own idea. So the first step to fork a GitHub repository.

Forking a repository is *really* straightforward:

1. Make sure you're logged into GitHub with your account.
2. Find the GitHub repository with which you'd like to work.
3. Click the Fork button on the upper right-hand side of the repository's page.



Then you first have to clone the repository down to your local system using git clone. Before you can run git clone you need to copy the URL for the forked repository, just navigate to the forked repository and look on the right-hand side of the web page. You should see an area that is labeled "HTTP clone URL" and now copy the URL from there and then paste it in Git Bash with the git clone command. Example as follow:



After that you are ready to start making changes to your local git repository and can make changes according to your needs by using the `git vi` (name of the file) or by directly changing into the file itself. OR Either you can work on a repository by creating another branch which is usually more preferred because you can easily change it without changing the other file which gives you confidence. Once you are confident you can push it to your main branches with the command `git push origin` (Name of the branch).

Once you push a new branch up to your repository, GitHub will prompt you to create a pull request. The maintainers of the original project can use this pull request to pull your changes across to their repository and, if they approve of the changes, merge them into the main repository.

## Aim: Merge And Resolve conflicts created

Merge conflicts happen when people make different changes to the same line of the same file, or when one person edits a file and another person deletes the same file or Merge conflicts happen when you merge branches that have competing commits, and Git needs your help to decide which changes to incorporate in the final merge. Git can often resolve differences between branches and merge them automatically.

First add a file to the main branch and make changes by committing several times. Then create a new branch by using the command which can directly create and switch it to that branch by `git checkout -b (Name of the branch)` [Like `git checkout -b feature2`]. Make changes in this branch also (try to make changes to the similar line to generate the conflicts).

```
commit a946e079edc72744000f13fb4b45e1fd05fd55b6 (feature2)
Author: AreenSiwach <areen0252.be21@chitkara.edu.in>
Date:   Wed Jun 1 22:37:42 2022 +0530

    Changed 3rd time

commit 3780343e442eb52c15c56ca6ef7014d38689fa1f
Author: AreenSiwach <areen0252.be21@chitkara.edu.in>
Date:   Wed Jun 1 22:35:22 2022 +0530

    Added a Quote

commit e454175f4b60078a59c8c86a58dc5457fdc05266
Author: AreenSiwach <areen0252.be21@chitkara.edu.in>
Date:   Wed Jun 1 22:27:44 2022 +0530

    Changed 2nd time

commit f22c34542aa3d9a6b5ecf6f94d781805b9434c23
Author: AreenSiwach <areen0252.be21@chitkara.edu.in>
Date:   Wed Jun 1 22:26:34 2022 +0530

    Changed 1st time

commit 03c029606bd27a84f1dc20f52b82a04ce890a62a
Author: AreenSiwach <areen0252.be21@chitkara.edu.in>
Date:   Wed Jun 1 22:25:14 2022 +0530

    Added HTML
```

Again switch to the main branch and make a final commit, merge the branches by the command `git merge (The branch to be merged)` [Like `git merge feature2`] but before merging make sure the status is clear. A conflict is reflected. Now by using `git mergetool`, make the desired change to resolve the conflict. Now add and commit it and the conflict has been resolved.

\*`git merge` - Use `git mergetool` to run one of several merge utilities to resolve merge conflicts.

```
areen@DESKTOP-C90IB9M MINGW64 /c/GitHub (master|MERGING)
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
    both modified:   HTML.html

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

areen@DESKTOP-C90IB9M MINGW64 /c/GitHub (master|MERGING)
$ git add .

areen@DESKTOP-C90IB9M MINGW64 /c/GitHub (master|MERGING)
$ git commit "Resolved issue"
git: 'commitResolved issue' is not a git command. see 'git --help'.

areen@DESKTOP-C90IB9M MINGW64 /c/GitHub (master|MERGING)
$ git commit -m "Resolved an issue"
[master c92b435] Resolved an issue

areen@DESKTOP-C90IB9M MINGW64 /c/GitHub (master)
$ |
```

```
commit 757956639d2e1411dfa73694dce2f825dbd84ab8 (origin/master)
Merge: a20eae4 a946e07
Author: AreenSiwach <areen0252.be21@chitkara.edu.in>
Date:   Wed Jun 1 22:45:05 2022 +0530

    Merging Branches

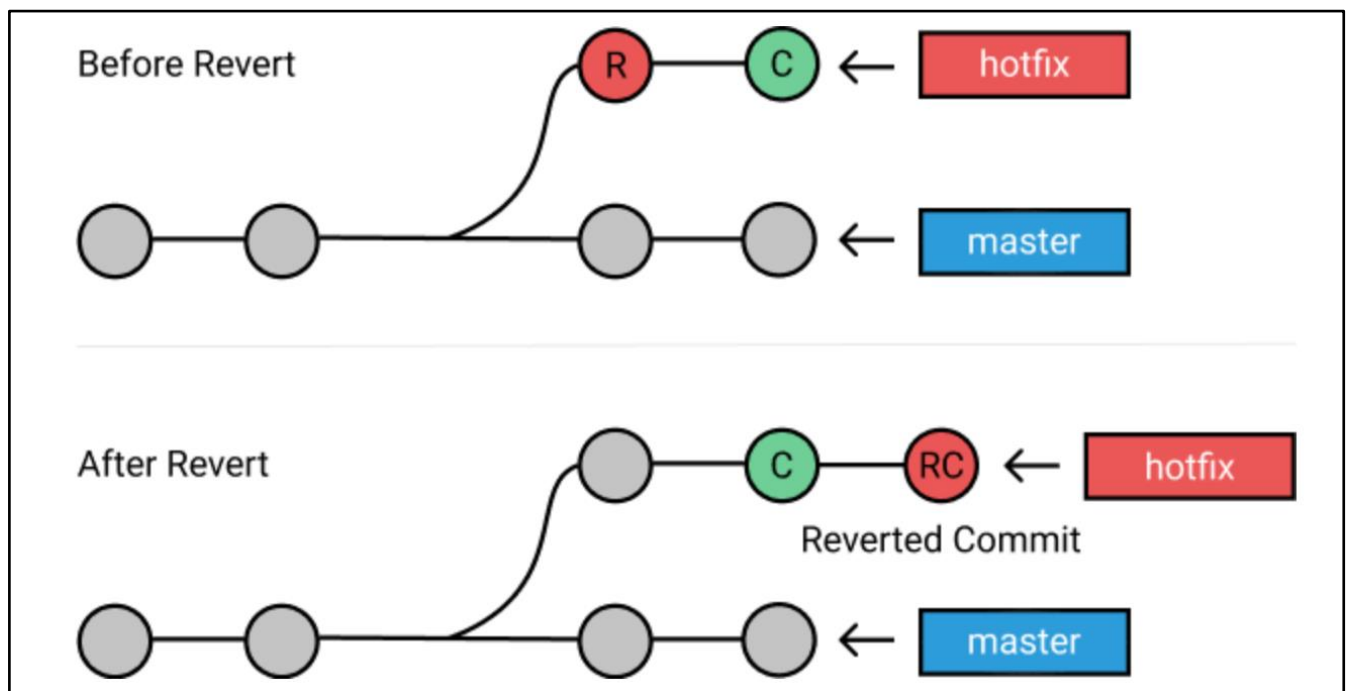
commit a20eae4b16398deb2340af8c65efcd1f4db1fa6a
Author: AreenSiwach <areen0252.be21@chitkara.edu.in>
Date:   Wed Jun 1 22:40:41 2022 +0530

    Final commit
```

## Aim: Reset and Revert

Git revert is a tool for undoing committed changes, while git reset HEAD is for undoing uncommitted changes. Like git checkout, git revert has the potential to overwrite files in the working directory, so it will ask you to commit or stash changes that would be lost during the revert operation.

Diagram to display how revert command work:



MINGW64:/c/GitHub

```
areen@DESKTOP-C90IB9M MINGW64 /c/GitHub (master)
$ git init
Reinitialized existing Git repository in C:/GitHub/.git/

areen@DESKTOP-C90IB9M MINGW64 /c/GitHub (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   HTML.html

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

areen@DESKTOP-C90IB9M MINGW64 /c/GitHub (master)
$ git add .

areen@DESKTOP-C90IB9M MINGW64 /c/GitHub (master)
$ git commit -m"Added another Quote"
[master c11a797] Added another Quote
 1 file changed, 1 insertion(+), 5 deletions(-)

areen@DESKTOP-C90IB9M MINGW64 /c/GitHub (master)
$ git log
commit c11a797438dcf0e4f26b837353f7537b802ec7a5 (HEAD -> master)
Author: AreenSiwach <areen0252.be21@chitkara.edu.in>
Date: Thu Jun 2 23:20:44 2022 +0530

    Added another Quote
```



```
[master 48d947b] Revert "Quote 2"
1 file changed, 1 deletion(-)

areen@DESKTOP-C90IB9M MINGW64 /c/GitHub (master)
$ git log
commit 48d947bcb4875094bbf1b2c109461ee925b502ce (HEAD -> master)
Author: AreenSiwach <areen0252.be21@chitkara.edu.in>
Date: Thu Jun 2 23:25:13 2022 +0530

    Revert "Quote 2"

    This reverts commit 048020a89b2a2048354f94cff96f6886dcd23fed.

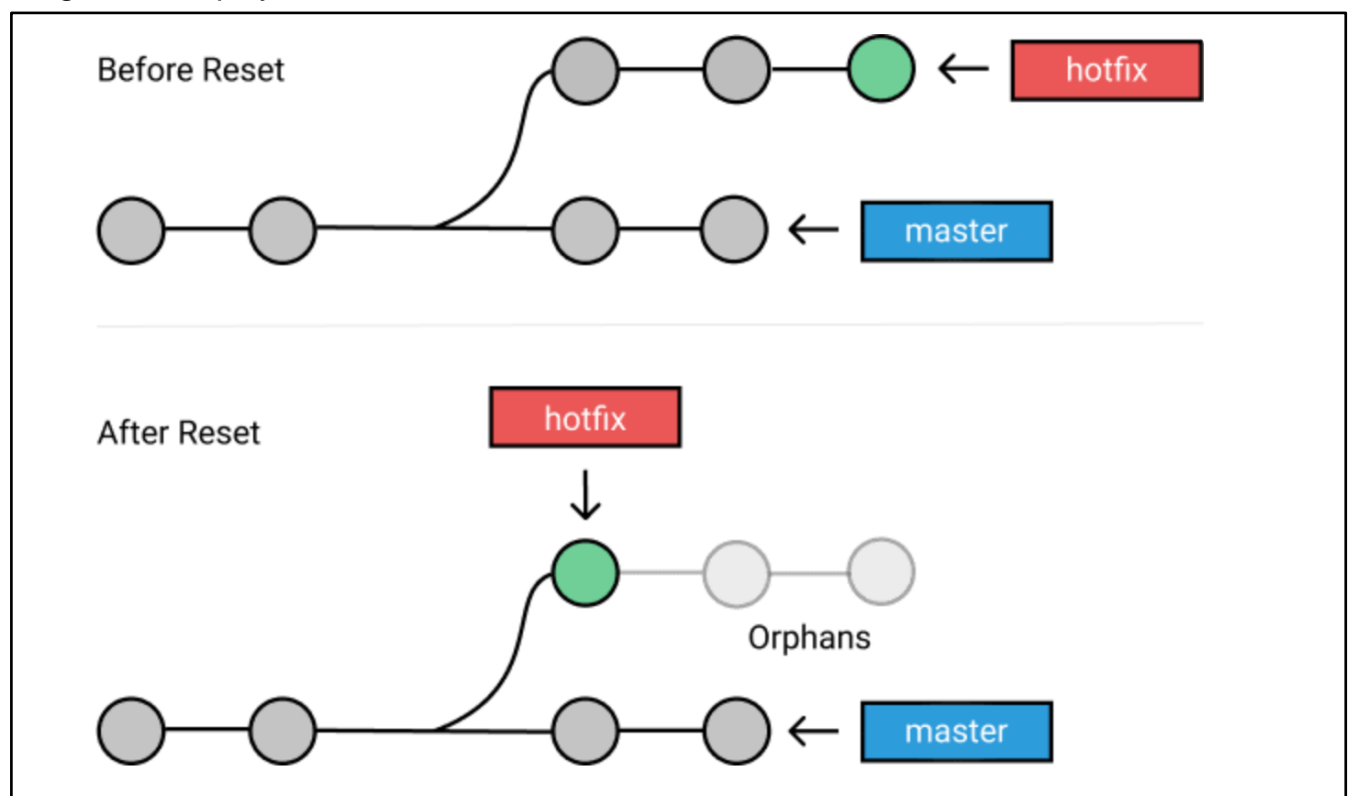
commit 048020a89b2a2048354f94cff96f6886dcd23fed
Author: AreenSiwach <areen0252.be21@chitkara.edu.in>
Date: Thu Jun 2 23:23:33 2022 +0530

    Quote 2

commit c11a797438dcf0e4f26b837353f7537b802ec7a5
Author: AreenSiwach <areen0252.be21@chitkara.edu.in>
Date: Thu Jun 2 23:20:44 2022 +0530

    Added another Quote
```

Diagram to display how reset command work:



```
areen@DESKTOP-C90IB9M MINGW64 /c/GitHub (master)
$ git status
On branch master
nothing to commit, working tree clean

areen@DESKTOP-C90IB9M MINGW64 /c/GitHub (master)
$ git reset --hard 048020a89b2a2048354f94cff96f6886dcd23fed
HEAD is now at 048020a Quote 2

areen@DESKTOP-C90IB9M MINGW64 /c/GitHub (master)
$ git log
commit 048020a89b2a2048354f94cff96f6886dcd23fed (HEAD -> master)
Author: AreenSiwach <areen0252.be21@chitkara.edu.in>
Date: Thu Jun 2 23:23:33 2022 +0530

    Quote 2

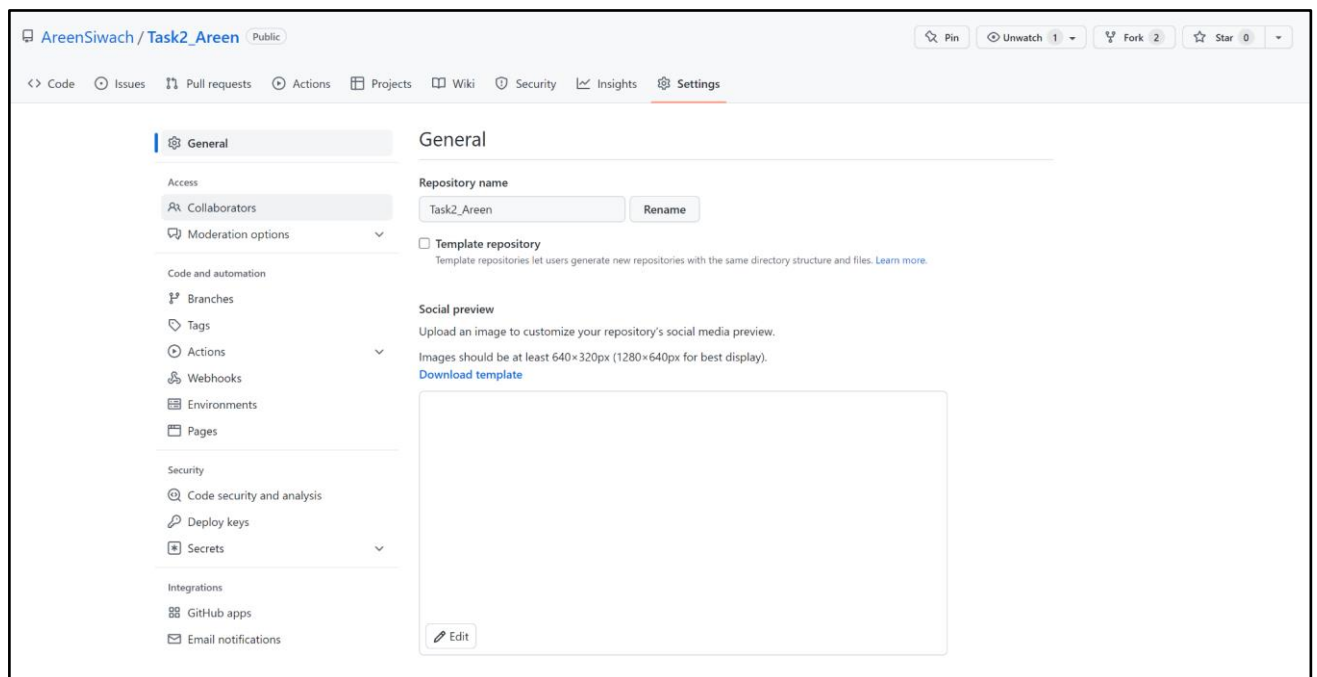
commit c11a797438dcf0e4f26b837353f7537b802ec7a5
Author: AreenSiwach <areen0252.be21@chitkara.edu.in>
Date: Thu Jun 2 23:20:44 2022 +0530

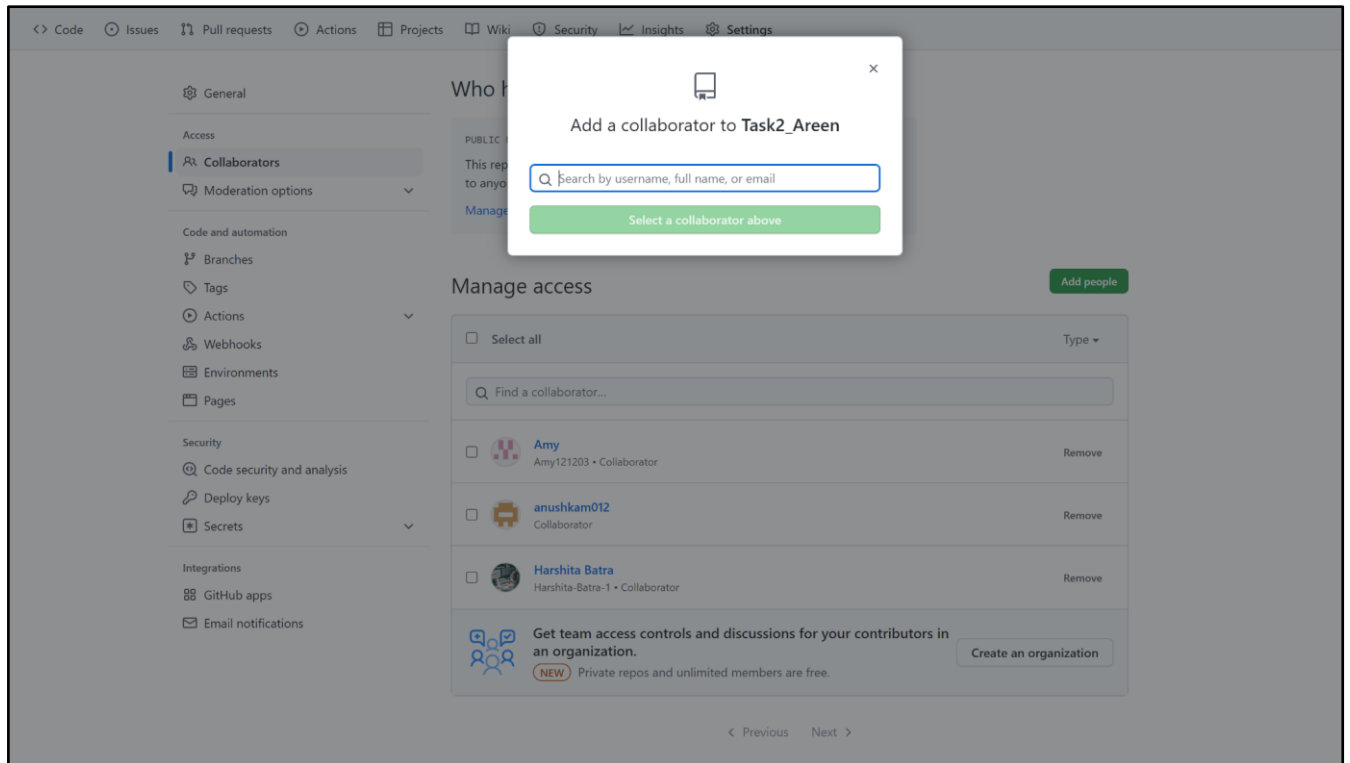
    Added another Quote
```

# Task 2

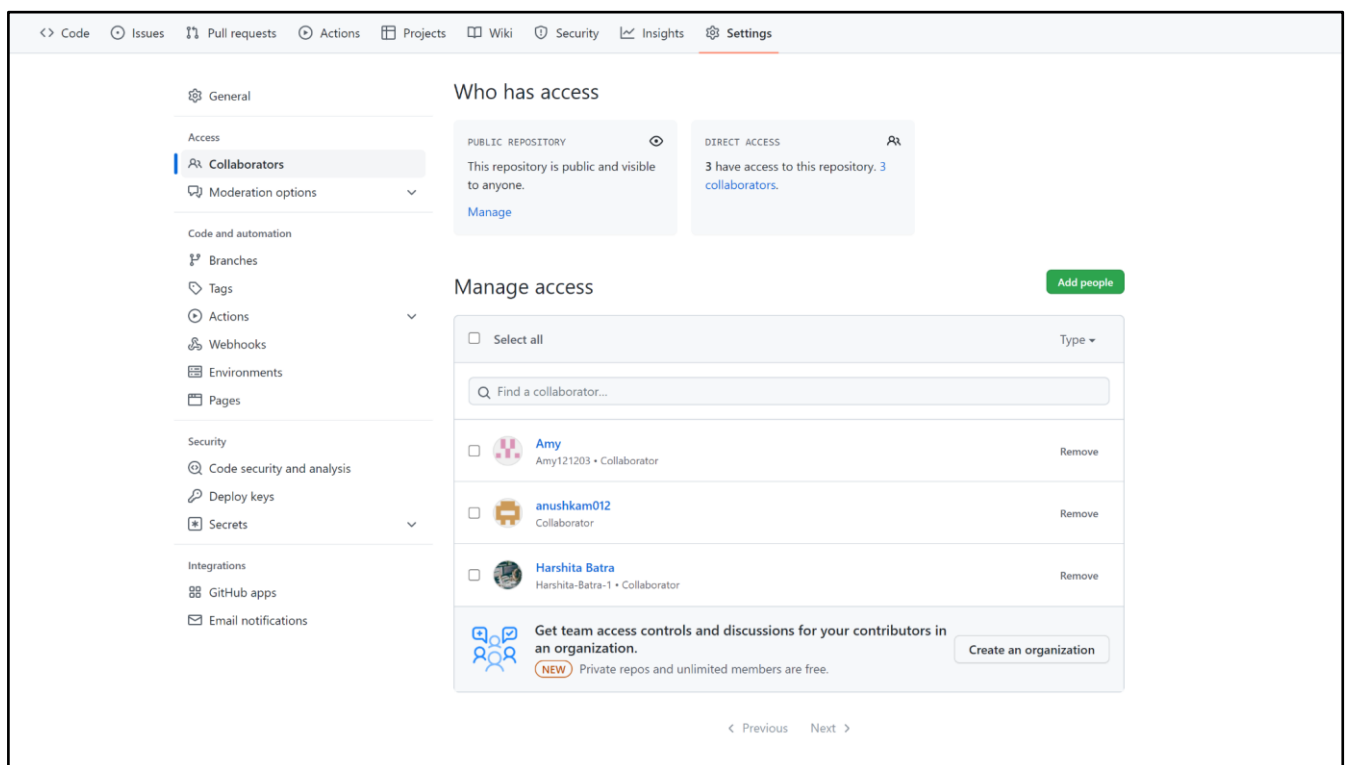
## 1. Create a repository and add your project members to the project team.

To add them, open your repository and select the settings option. Then click on the collaborators mentioned under General on the left hand side, after clicking on collaborators Github asks you to enter your password to confirm the access to the repository and by clicking on add, search for your members id.



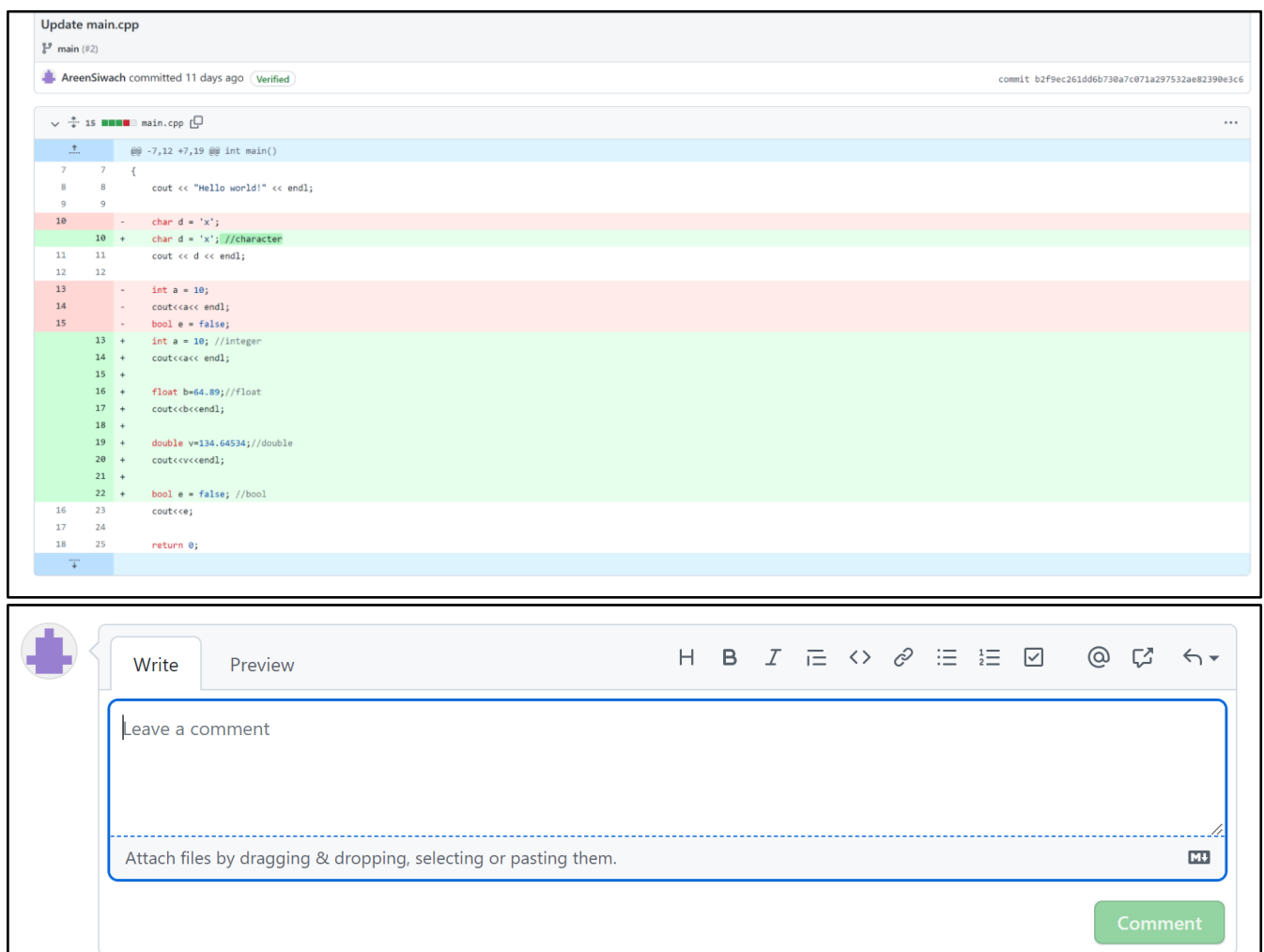


To accept the invitation from others, go to your registered email and open the mail sent by the other members of the team and accept by clicking on the accept the invitation or may decline it. Now you can push options and all the members are to contribute.



## 2. Open and close a Pull Request & Each project member shall create a pull request on a team members repo and close pull requests generated by team members on their own Repo as a maintainer.

Pull requests let you tell others about changes you've pushed to a branch in a repository on GitHub. Once a pull request is opened, you can discuss and review the potential changes with collaborators and add follow-up commits before your changes are merged into the base branch. So for that first fork and clone the repository and then do the required changes in the repository, add and commit these changes in the local repository in a new branch.

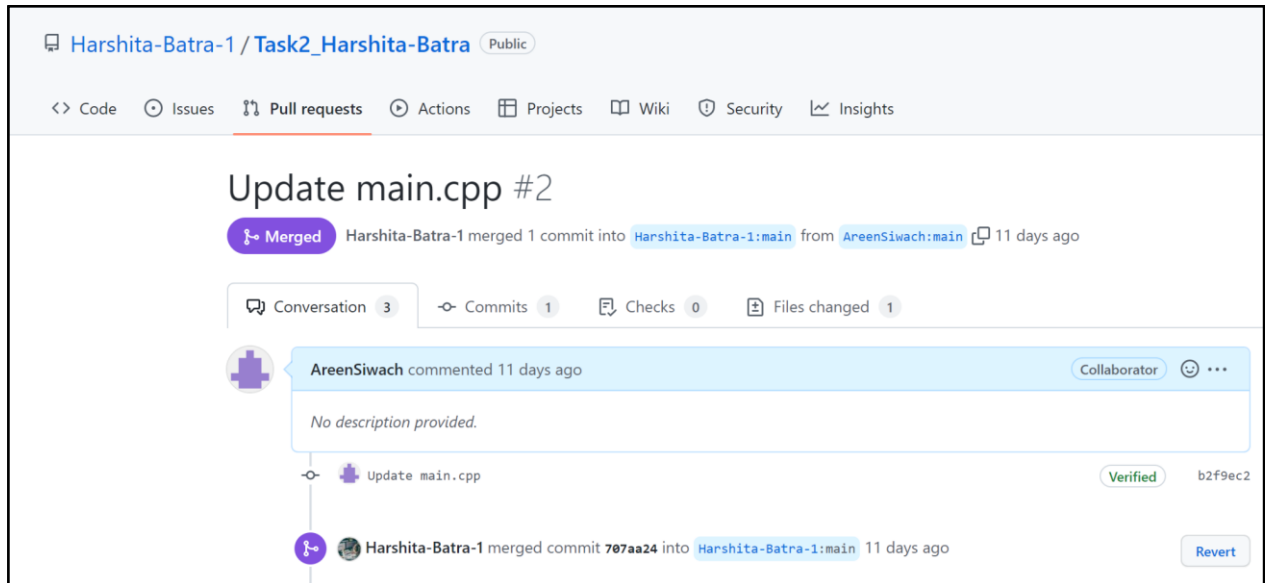


The screenshot displays a GitHub pull request titled "Update main.cpp" for the "main" branch. The pull request was created by "AreenSiwach" 11 days ago and is marked as "Verified". The commit hash is b2f9ec261dd6b730a7c071a297532ae82390e3c6.

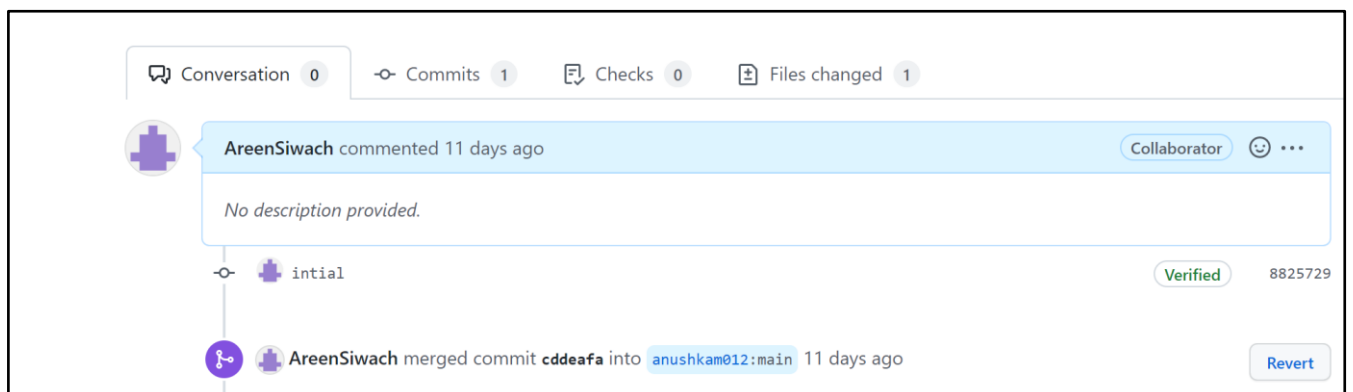
The code diff shows changes to "main.cpp". The original code (lines 7-18) includes a "Hello world!" message and a character 'x'. The updated code (lines 10-25) adds several data types: an integer 'a', a float 'b', a double 'v', and a boolean 'e', each with a comment indicating its type.

Below the code diff, there is a comment box with the placeholder text "Leave a comment". The box includes a rich text editor toolbar with options for bold, italic, link, and other formatting. A "Comment" button is located at the bottom right of the comment box.

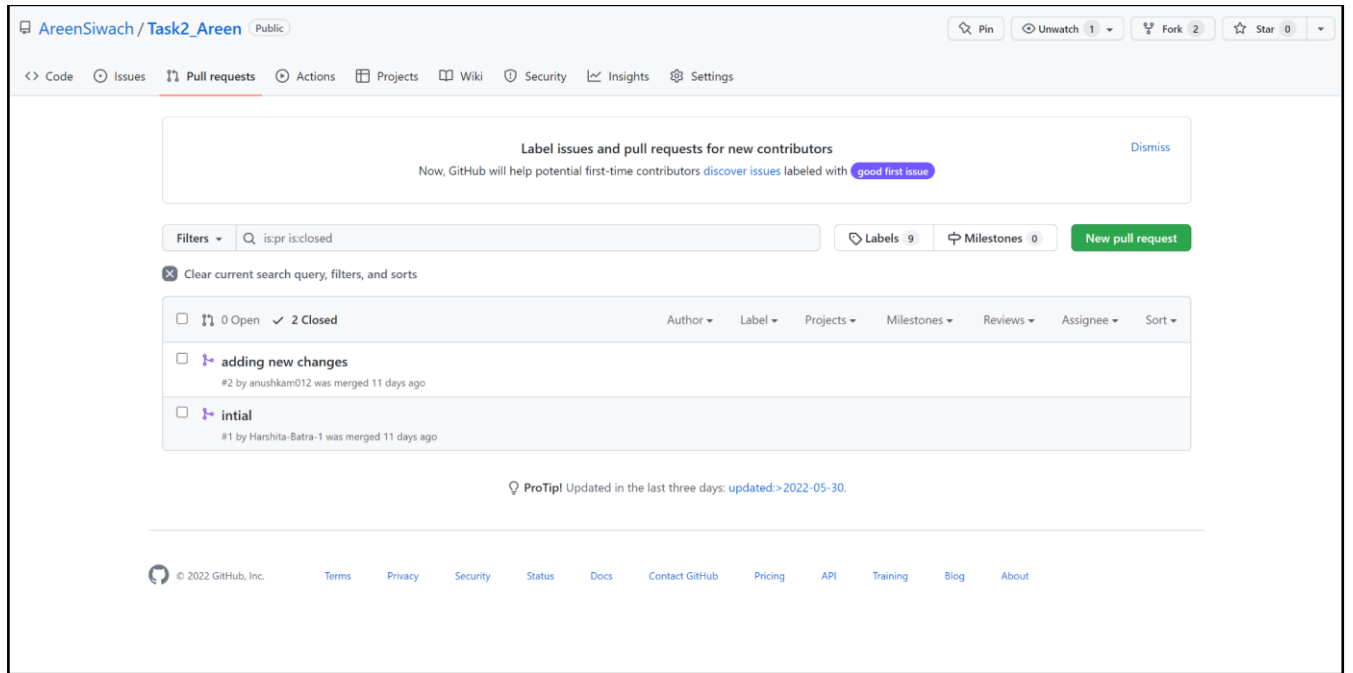
Push the modified branch using git push origin branch name. Open a pull request by following the procedure from the above experiment. The pull request will be created and will be visible to all the team members. Ask your team member to login to his/her Github account. They will notice a new notification in the pull request menu. As soon as they click on it. The pull request generated by you will be visible to them.



By selecting the merge branch option the main branch will get updated for all the team members. By selecting close the pull request the pull request is not accepted and not merged with the main branch. Hence, the merged file will be visible to the visitors.



Thus, we conclude the opening and closing of pull requests. We also conclude merging of the pull request to the main branch.

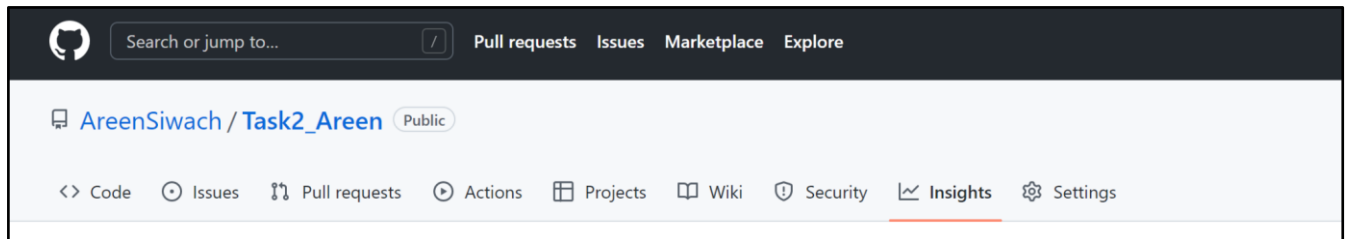


### 3. Publish and print network graphs

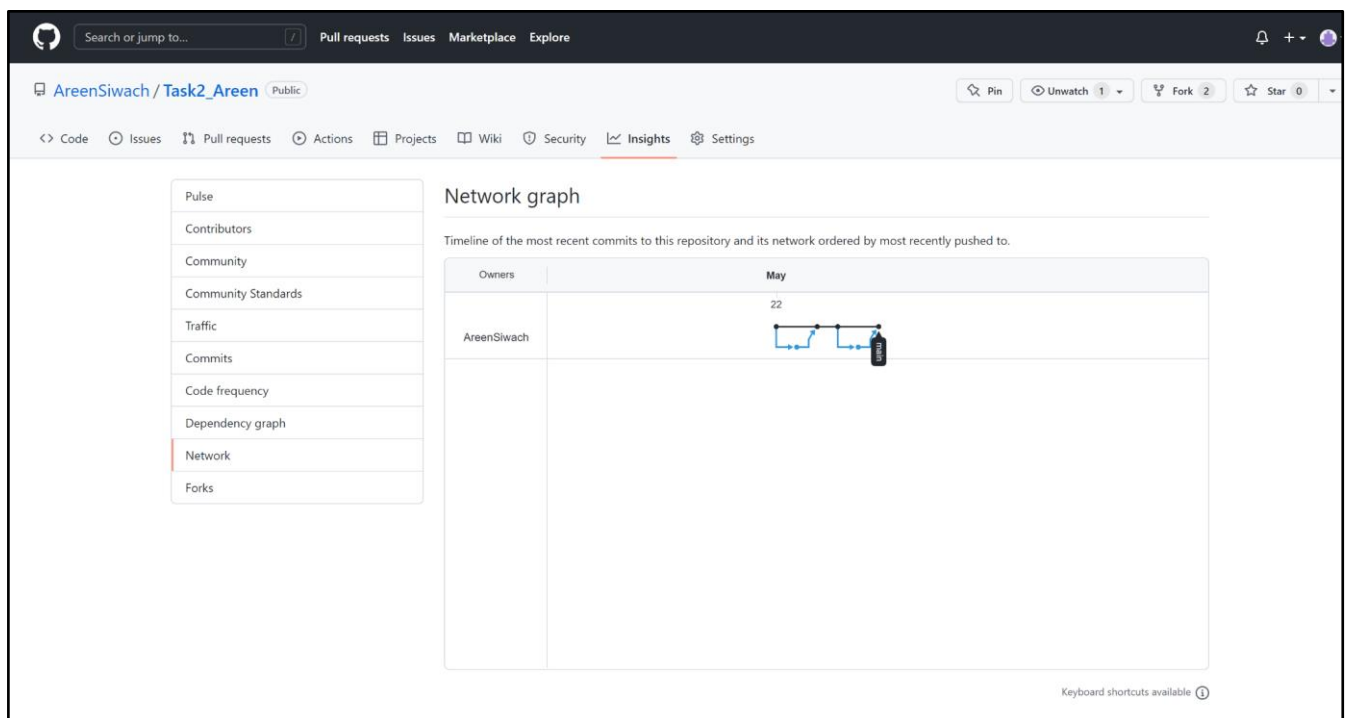
The network graph is one of the useful features for developers on GitHub. It is used to display the branch history of the entire repository network, including branches of the root repository and branches of forks that contain commits unique to the network.

Steps to access the network graph on GitHub:

- First, click on Insights on the main page of the repository.



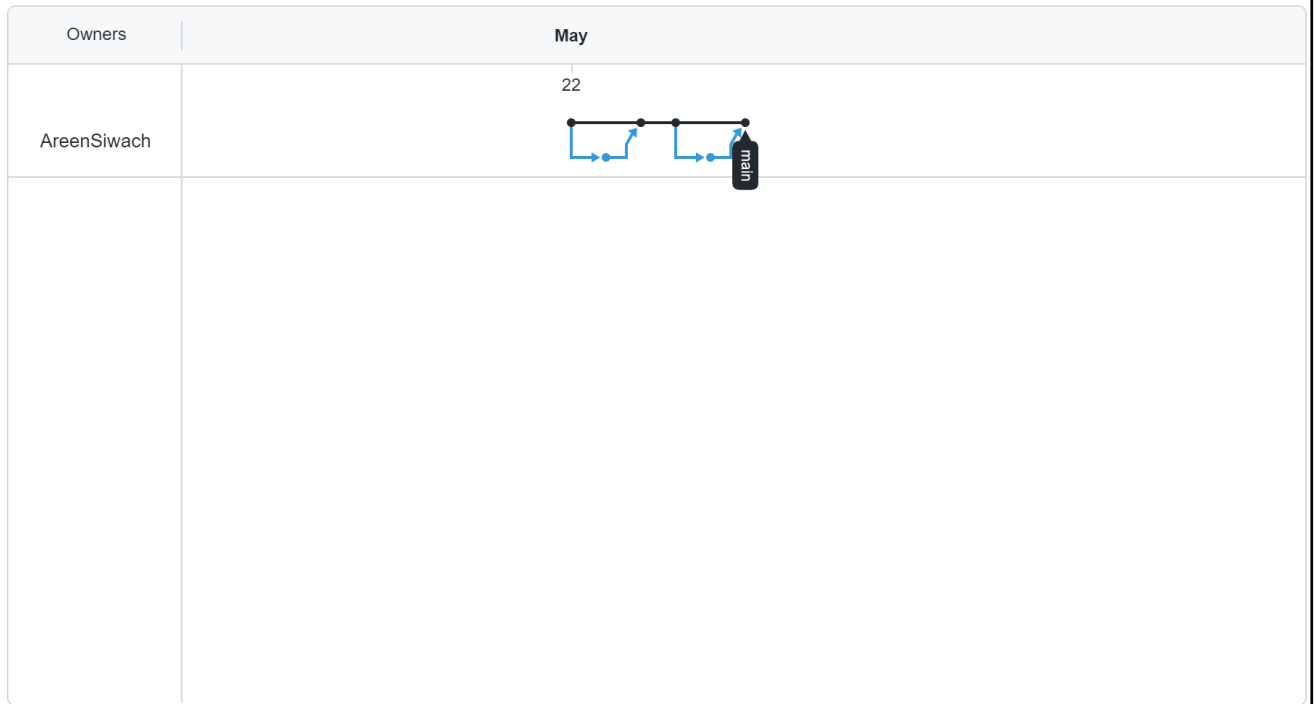
- Then from the left side of the page, select Network.



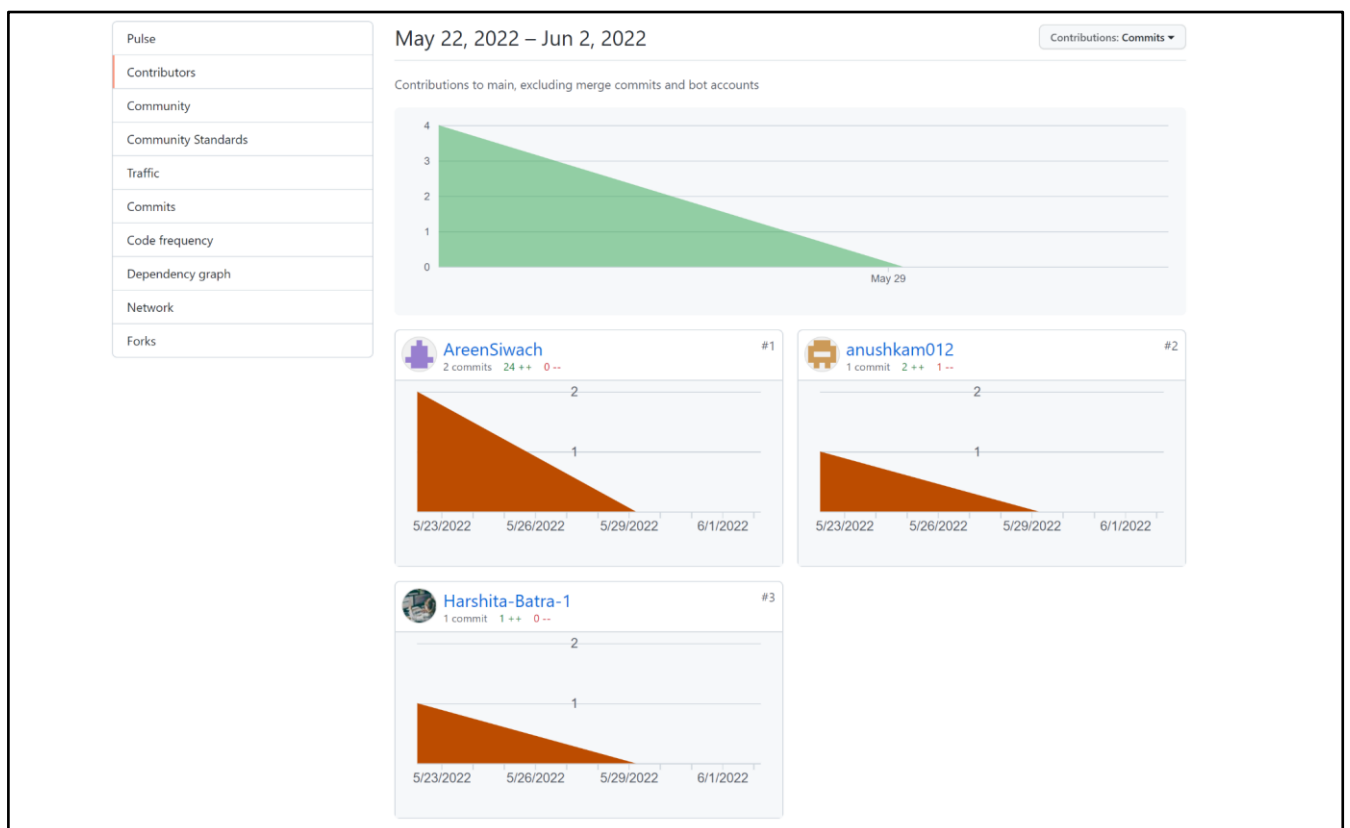


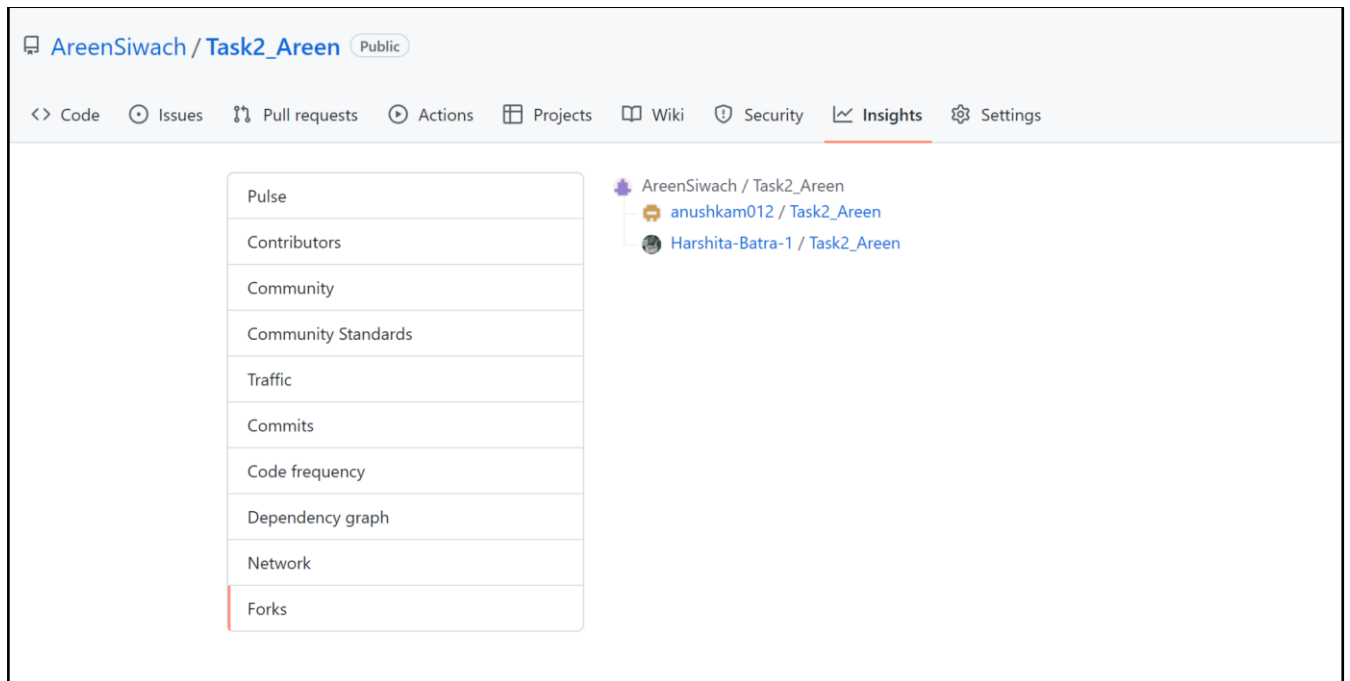
## Network graph

Timeline of the most recent commits to this repository and its network ordered by most recently pushed to.



## Contributors graph:





*Thus, this concludes the Project.*