Subject Name: **Source Code Management**

Subject Code: **CS181**

Cluster: **Beta**

Department: **DCSE**

Submitted By:
Anish Kumar Garg
2110990192
G8-A

Submitted To:
Dr. Monit Kapoor

Introduction

What is GIT and why is it used?

Git is a source code management technology used by DevOps. Git is a piece of software that allows you to track changes in any group of files. It is a free and open-source version control system that may be used to efficiently manage small to big projects.

Git is a version control system that allows numerous developers to collaborate on non-linear development projects.

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

What is GITHUB?

GitHub is a version management and collaboration tool for programming. It allows you and others to collaborate on projects from any location.

What is the difference between GIT and GITHUB?

Git	GitHub
1. It is a software	1. It is a service
2. It is installed locally on the system	2. It is hosted on Web
3. It is a command line tool	3. It provides a graphical interface
4. It is a tool to manage different versions of edits, made to files in a git repository	4. It is a space to upload a copy of the Git repository
5. It provides functionalities like Version Control System Source Code Management	5. It provides functionalities of Git like VCS, Source Code Management as well as adding few of its own features

What is Repository?

A repository stores all of your project's files, as well as the revision history for each one. Within the repository, you may discuss and monitor your project's progress. The.git/ subdirectory within a project is a Git repository. This repository keeps track of any changes made to files in your project over time, creating a history. That is, if you delete the.git/ subdirectory, you are also deleting the history of your project.

What is Version Control System (VCS)?

Version Control Systems are the software tools for tracking/managing all the changes made to the source code during the project development. It keeps a record of every single change made to the code. It also allows us to turn back to the previous version of the code if any mistake is made in the current version. Without a VCS in place, it would not be possible to monitor the development of the project.

Types of VCS

- Local Version Control System
- Centralized Version Control System
- 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.

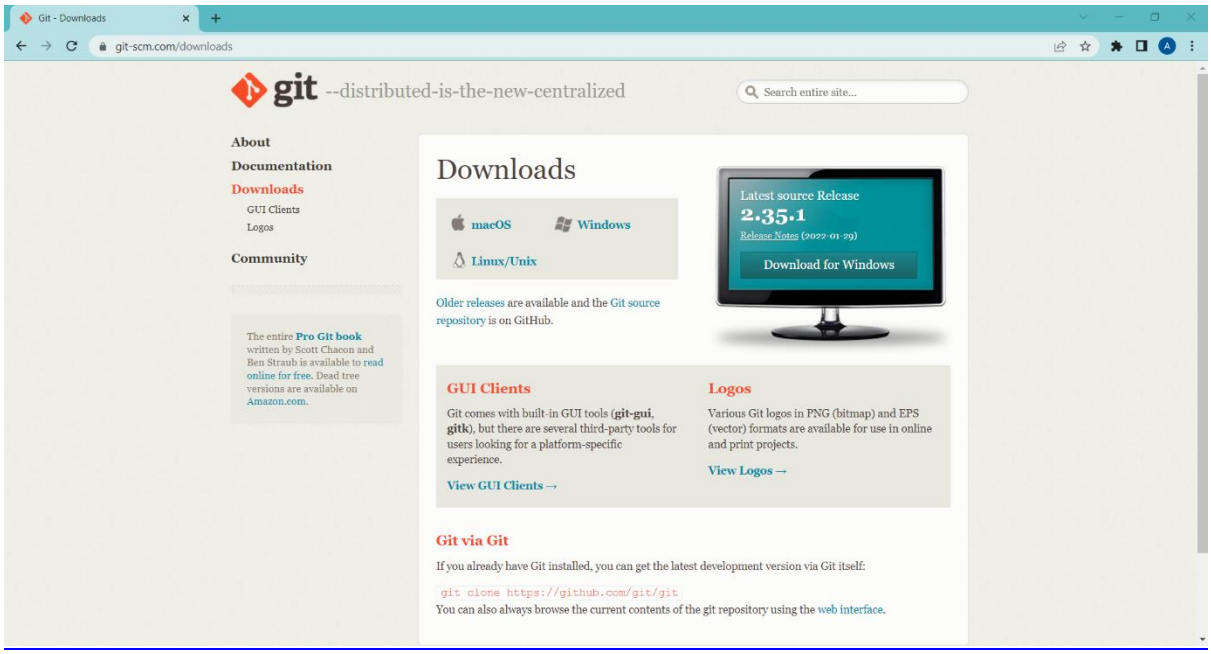
II. **Centralized Version Control System:** In the Centralized Version Control Systems, there will be a single central server that contains all the files related to the project, and many collaborators checkout files from this single server (you will only have a working copy). The problem with the Centralized Version Control Systems is if the central server crashes, almost everything related to the project will be lost.

III. **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.

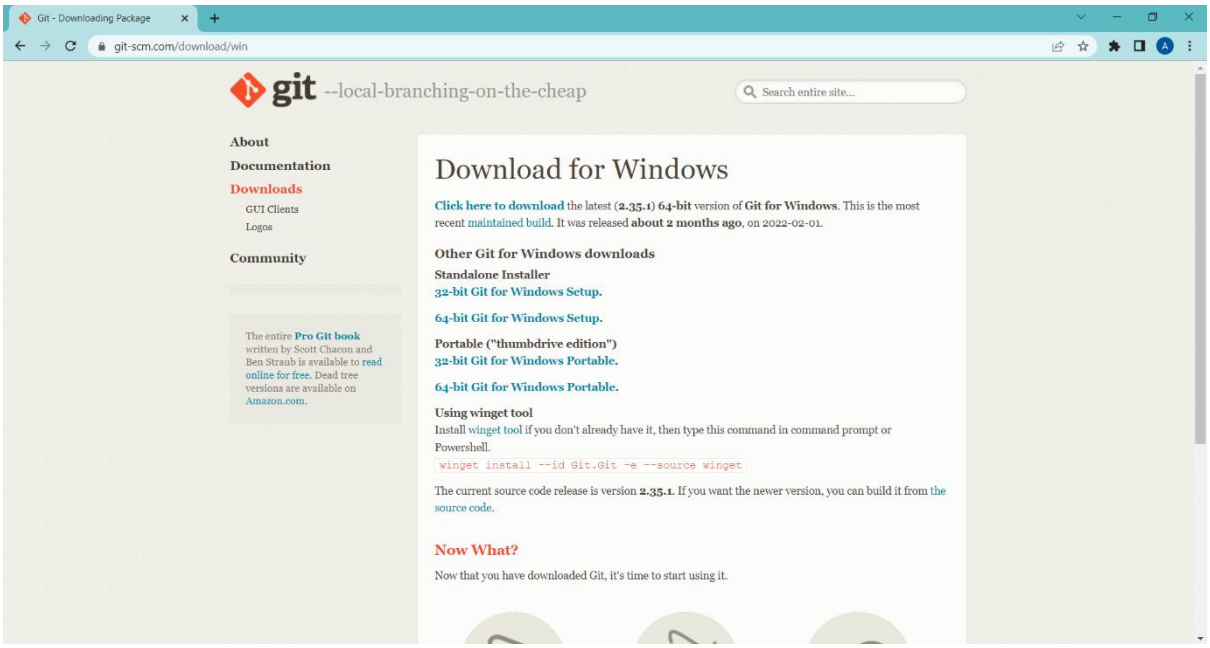
Experiment No. 01

Aim: Setting up of Git Client

✧ For git installation on your system, go to the linked URL.
https://git-scm.com/downloads



✧ You must first access this webpage and then choose your operating system by clicking on it. I'll walk you through the processes for the Windows operating system in this article.

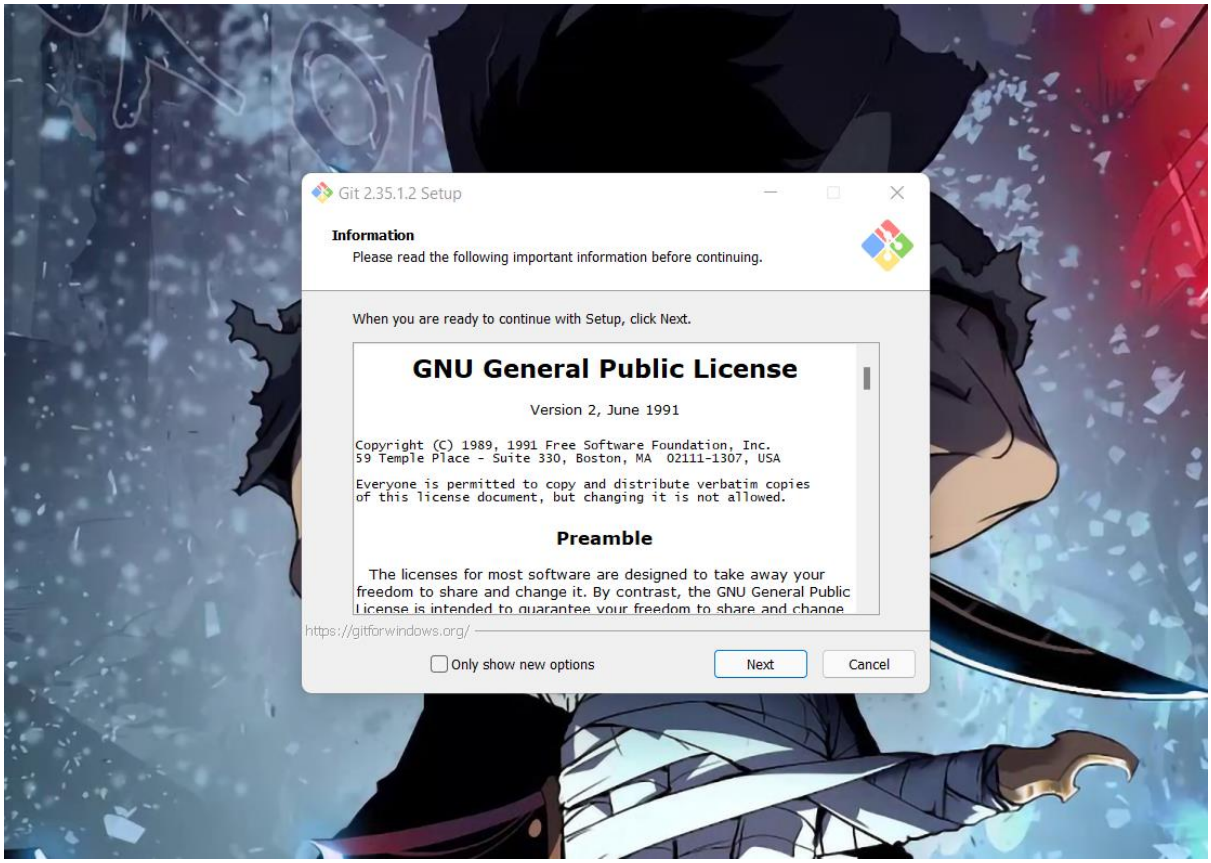


✧ Select the CPU for your system now. (Most of the system now runs on 64-bit processors.) Your download will begin when you pick a processor.

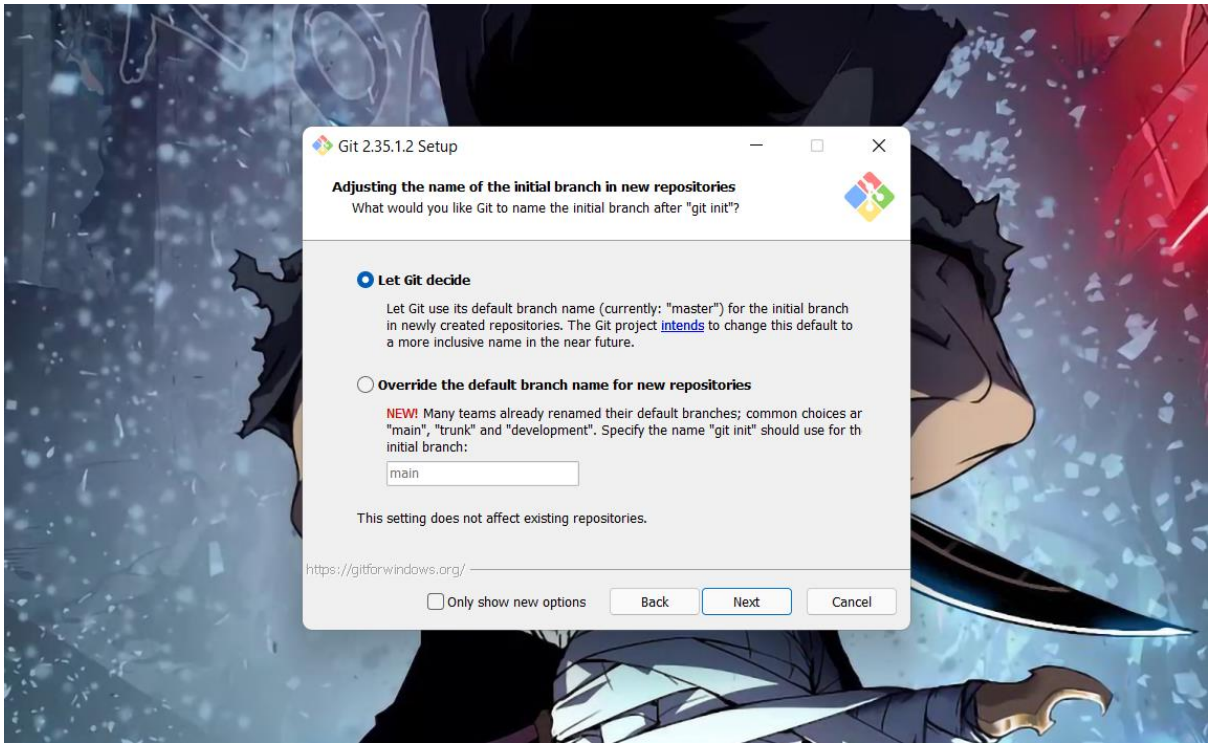
✧ You must now open the Git folder.

✧ You will be asked if you want to enable this programe to make modifications to your PC once you launch it.

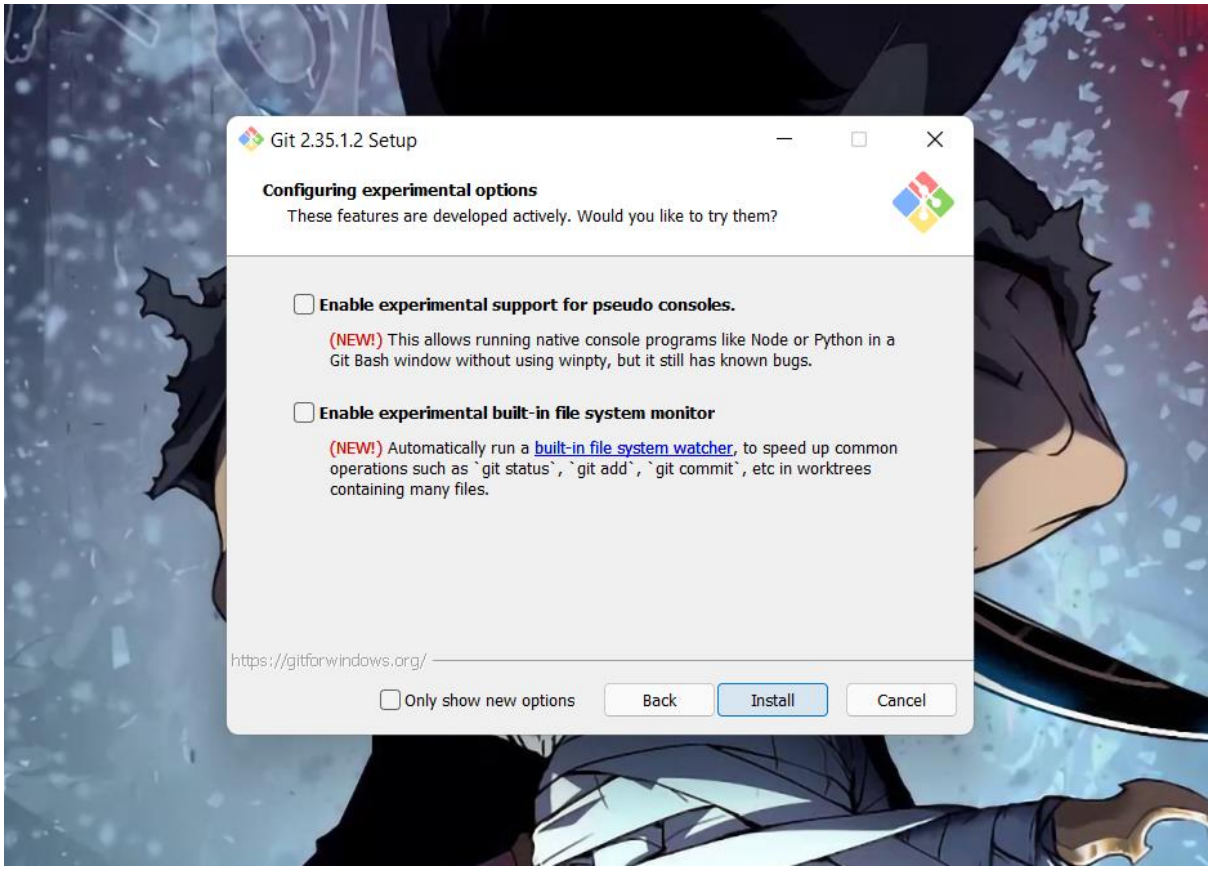
✧ YES should be selected.



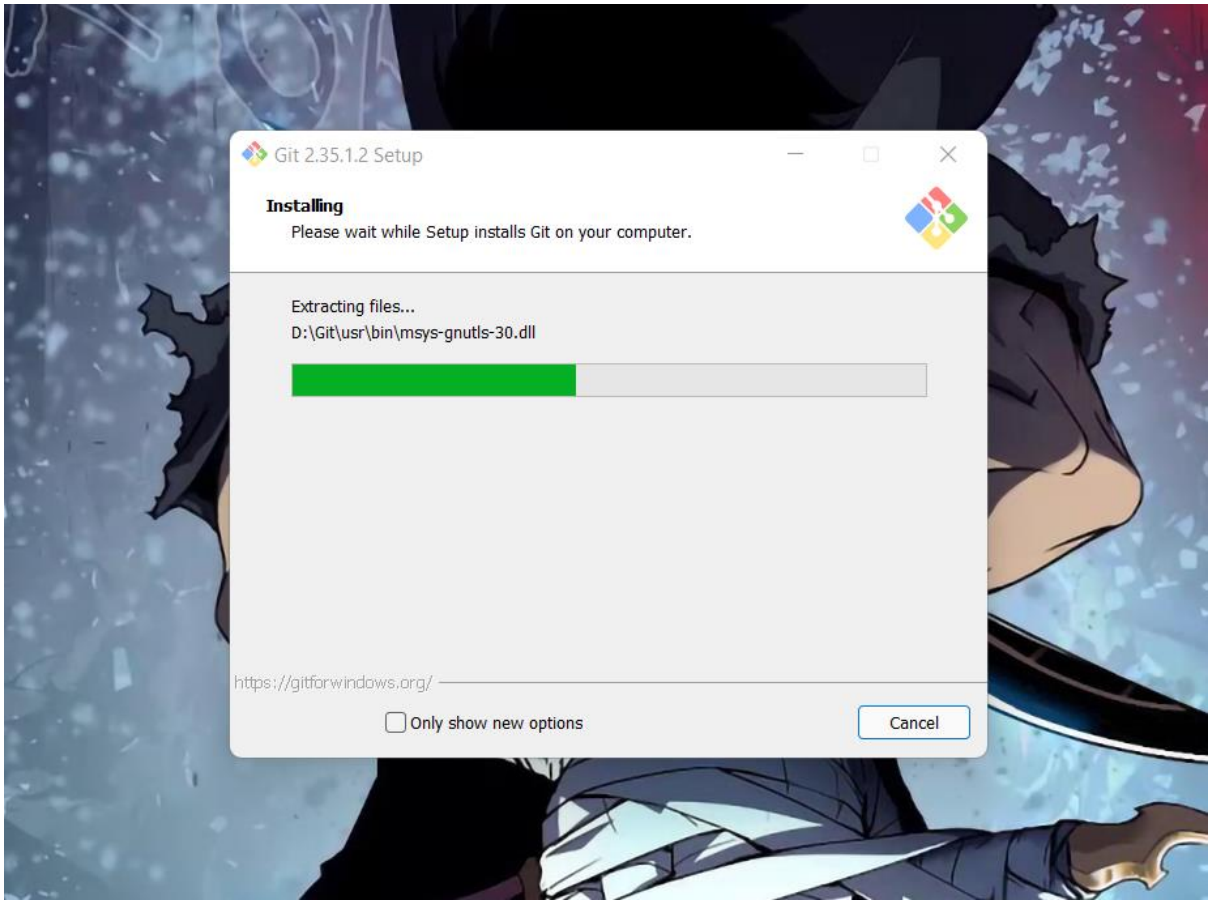
✧ Click on Next



✧ Continue clicking on next few times more



✧ Now select the Install option.



✧ Click on Finish after the installation is finished.

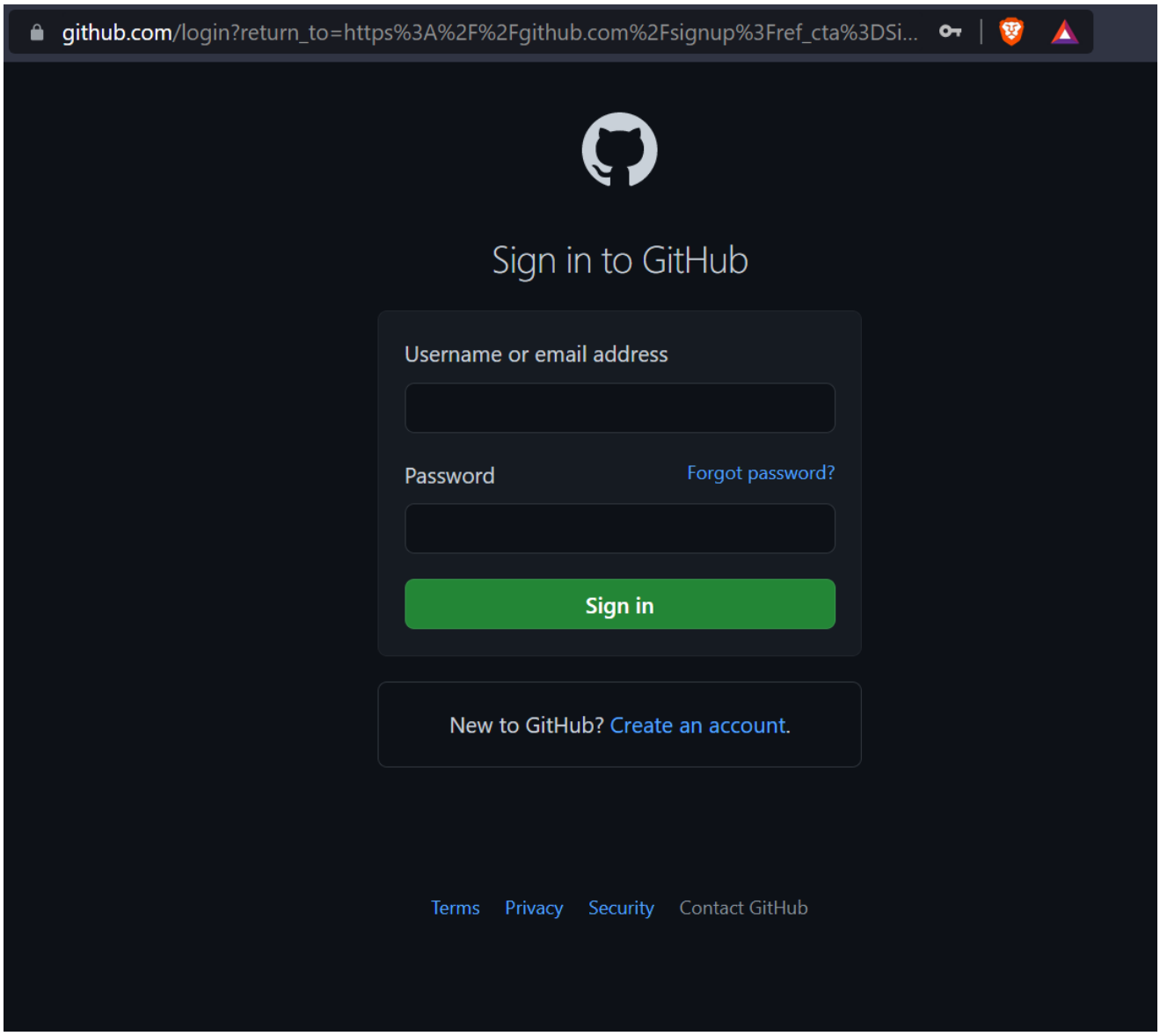
The installation of the git is finished and now we have to setup git client and GitHub account.

Experiment No. 02

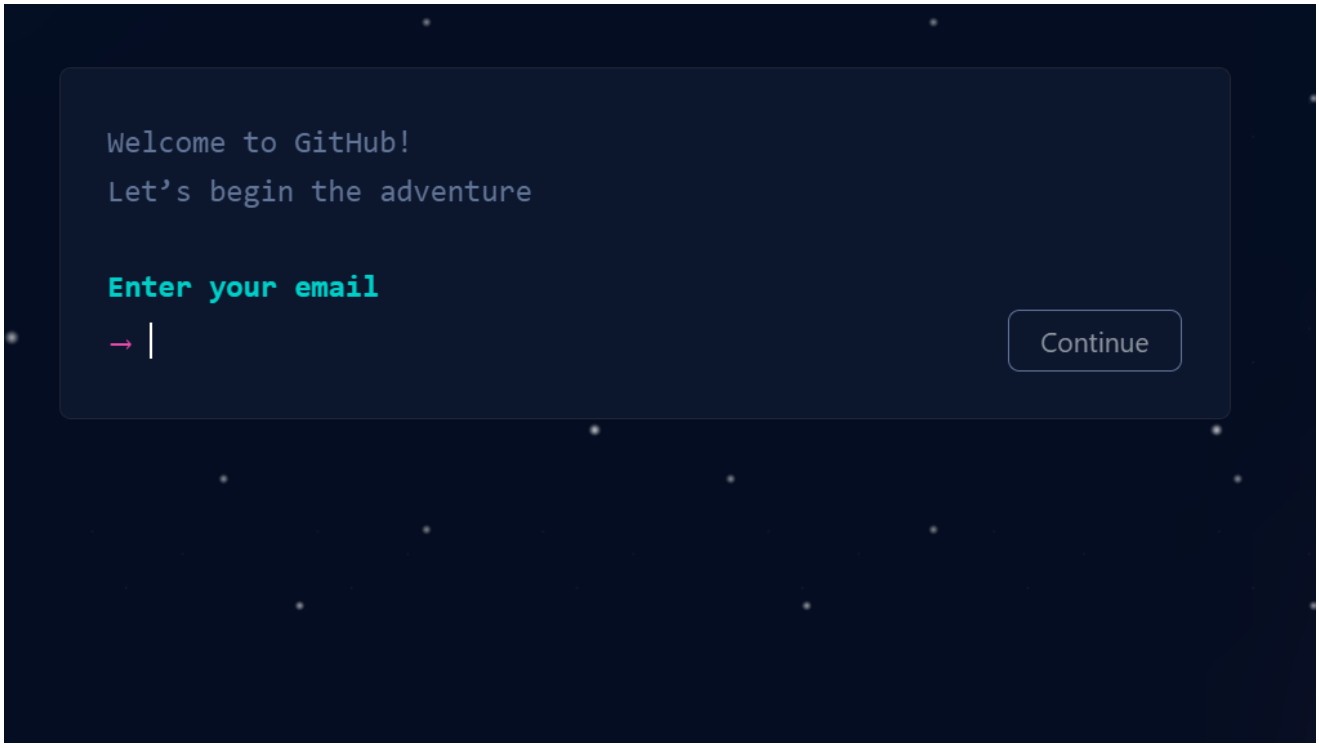
Aim:Setting up GitHub Account

✧ Open your web browser search GitHub login.

✧ Click on Create an account if you are a new user or if you have already an account, please login.



✧ After clicking on "Create a New Account," you will be sent to a new page where you must enter your email address for your account. Now type in the password you'd want to use for your GitHub account. Then you'll be prompted to enter your username.



✧ Now Click on Create Account.

✧ Verify it from your email and you are all set to go.

Experiment No. 03

Aim: Program to Generate logs

✧ 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 "**gitinit**" and it creates a folder.**git**.

```
lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/projects
$ git init
Initialized empty Git repository in D:/SCM/projects/.git/

lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/projects (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.

Now, we have to create some files in the repository. Suppose we created test.txt Now type git status:

```
lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/projects (master)
$ touch test.txt

lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/projects (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        test.txt

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

You can see that *test.txt* is in red colour that means it is an untracked file.
Now firstly add the file in staging area and then commit the file.

For this, use command →

git add -A [For add all the files in staging area.]

git commit -m "write any message" [For commit the file]

```
lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/projects (master)
$ git add test.txt

lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/projects (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   test.txt


lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/projects (master)
$ git commit -m "commiting test file"
[master (root-commit) c5d4153] commiting test file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 test.txt

lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/projects (master)
$ git log
commit c5d4153b913acc28b676906fc62622c904bde2f0 (HEAD -> master)
Author: AnishKumarGarg <anish0192.be21@chitkara.edu.in>
Date:   Sun Apr 10 22:02:10 2022 +0530

    commiting test file
```

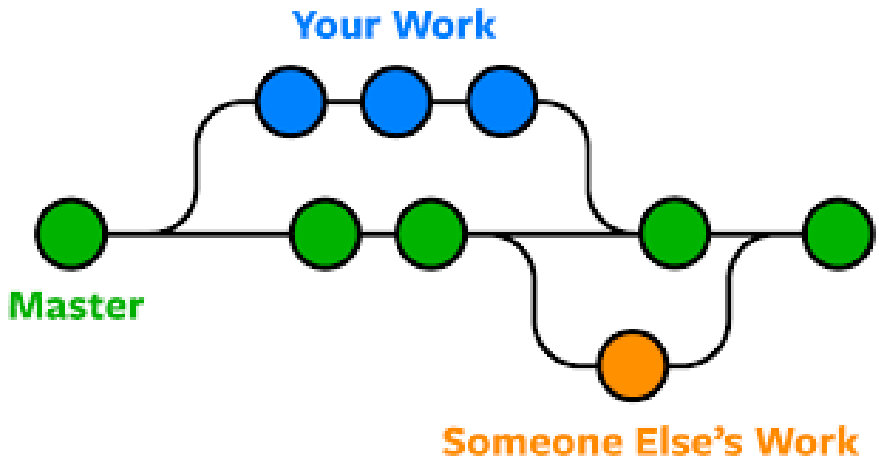
✧ **git log:** *The git log command displays a record of the commits in a Git repository. By default, the git log command displays a commit hash, the commit message, and other commit metadata.*

```
lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/projects (master)
$ git log
commit c5d4153b913acc28b676906fc62622c904bde2f0 (HEAD -> master)
Author: AnishKumarGarg <anish0192.be21@chitkara.edu.in>
Date:   Sun Apr 10 22:02:10 2022 +0530

    commiting test file

lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/projects (master)
$
```

Experiment No. 04

Aim: Create and visualize branches

✧ **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.



Let us see the command of it:
Firstly, add a new branch, let us suppose the branch name is main.
For this use command →

- **git branch name** [adding new branch]
- **git branch** [use to see the branch's names]
- **git checkout *branch name*** [use to switch to the given branch]

```
lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/projects (master)
$ git branch
* master

lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/projects (master)
$ git branch main

lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/projects (master)
$ git checkout main
Switched to branch 'main'

lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/projects (main)
$ git branch
* main
  master
```

In this you can see that firstly ‘git branch’ shows only one branch in green colour but when we add a new branch using ‘git branch main’, it shows 2 branches but the green colour and star is on master. So, we have to switch to main by using ‘git checkout main’. If we use ‘git branch’, now you can see that the green colour and star is on main. It means you are in main branch and all the data of master branch is also on main branch. Use “ls” to see the files.

Now add a new file in main branch, do some changes in file and commit the file.

```
lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/projects (main)
$ touch temp.html

lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/projects (main)
$ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        temp.html

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

lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/projects (main)
$ git add temp.html

lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/projects (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   temp.html


lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/projects (main)
$ git commit -m "committing temp.html "
[main 94f644e] committing temp.html
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 temp.html

lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/projects (main)
$ git log
commit 94f644e07490aaea465acc39c5c811cd94789e80 (HEAD -> main)
Author: AnishKumarGarg <anish0192.be21@chitkara.edu.in>
Date:   Sun Apr 10 22:47:08 2022 +0530

    committing temp.html

commit c5d4153b913acc28b676906fc62622c904bde2f0 (master)
Author: AnishKumarGarg <anish0192.be21@chitkara.edu.in>
Date:   Sun Apr 10 22:02:10 2022 +0530

    commiting test file
```

If we switched to master branch, ‘temp.html’ file is not there. But he file is in main branch.

```
lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/projects (main)
$ git branch
* main
  master

lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/projects (main)
$ git checkout master
Switched to branch 'master'

lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/projects (master)
$ ls
test.txt
```

- To add these files in master branch, we have to do merging. For this firstly switch to master branch and then use command →

  git merge branchname [use to merge branch]

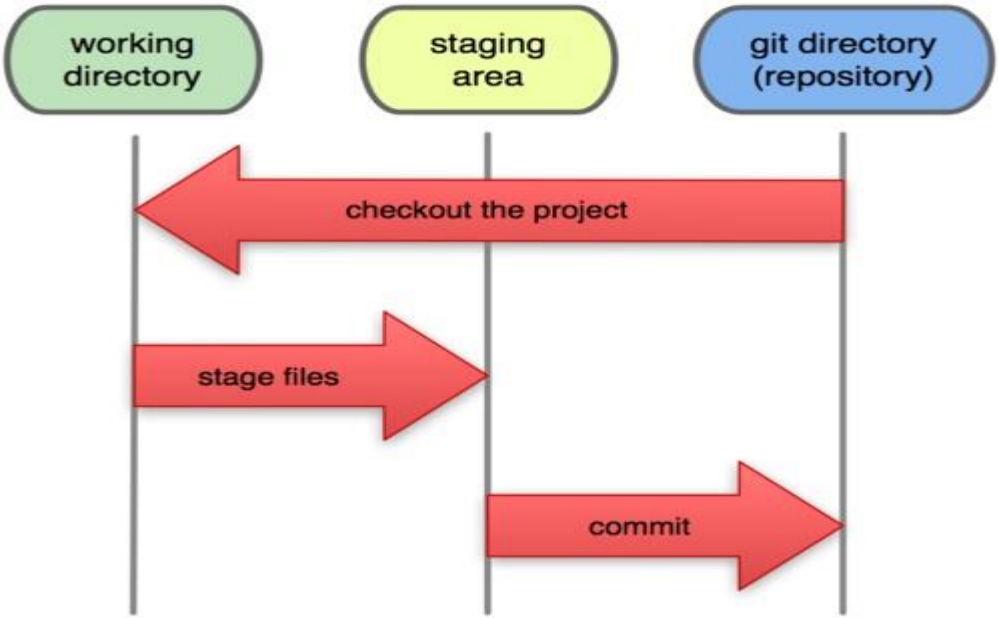
```
lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/projects (master)
$ git merge main
Updating c5d4153..94f644e
Fast-forward
 temp.html | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 temp.html

lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/projects (master)
$ ls
temp.html  test.txt
```

Experiment No. 05

Aim: Git lifecycle description

Stages in GIT Life Cycle:

Now let's understand the three-stage architecture of Git:



- **Working Directory:** This is the directory that we've initialized, and here all the changes are made to commit on GitHub.

- **Staging Area:** This is where we first put out code or files of the working repository. The command that we use to stage code is, "git add --a", "git add FileName" or "git add -A". In simple terms, staging means telling Git what files we want to commit (new untracked files, modified files, or deleted files).

- **Git directory(repository):** This is where all the commits are stored whenever we make a commit. We can revert to an older version of or project using the "git checkout" command from this directory.

INDEX

INDEX

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.

II. 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.

III. 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

There is a problem as we have to code a c++ project but we have to make our part and send it to other participants again and again and then at end we have to combine all the part and attach those snaps which is a tough process. So we want an alternate solution for this to show changes conveniently and accurately. Also we want that if we make the changes it should reflect to all the other participants. Also the other participants can revert the changes and commit the changes.

Objective

The objectives of the project are :

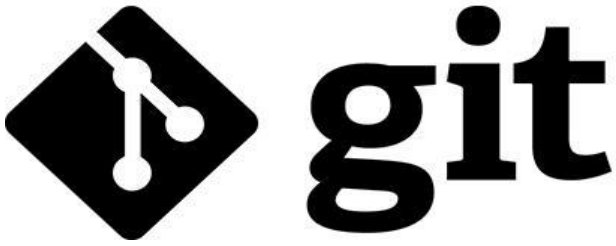
- It is easy to contribute to open source projects via GitHub.
- It helps to create an excellent document..
- You can attract the recruiter by showing off your work.
- If you have a profile on GitHub, you will have a higher chance of being recruited.
- It allows your work to get out there in front of the public.

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.

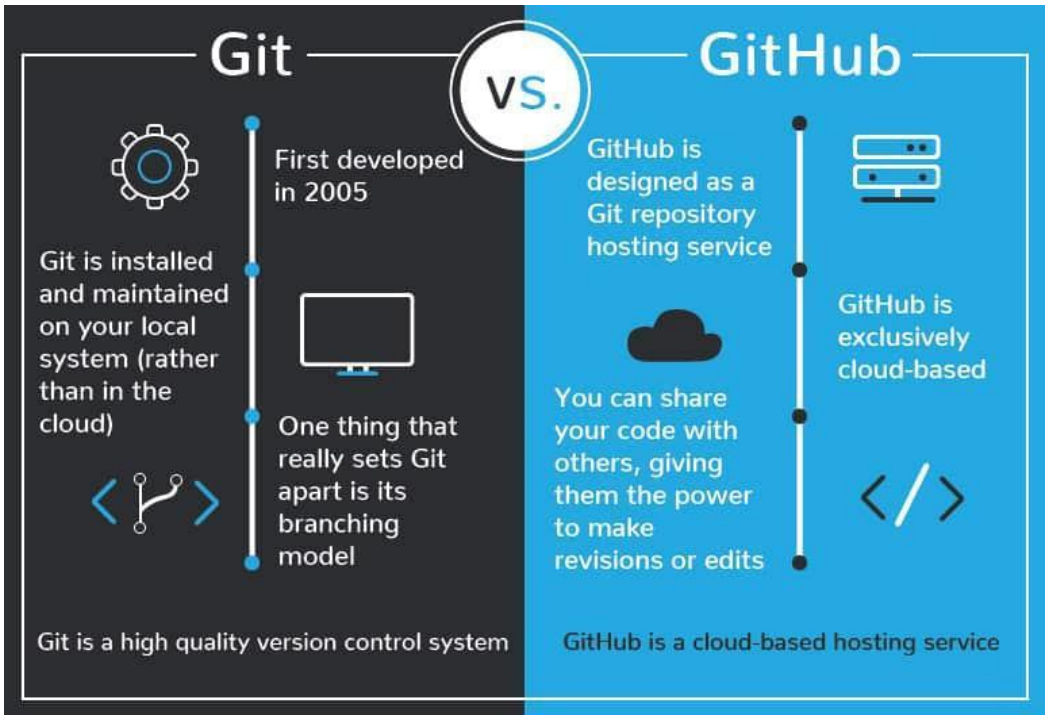
Git is an example of a distributed version control system (DVCS) (hence Distributed Version Control System).



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?



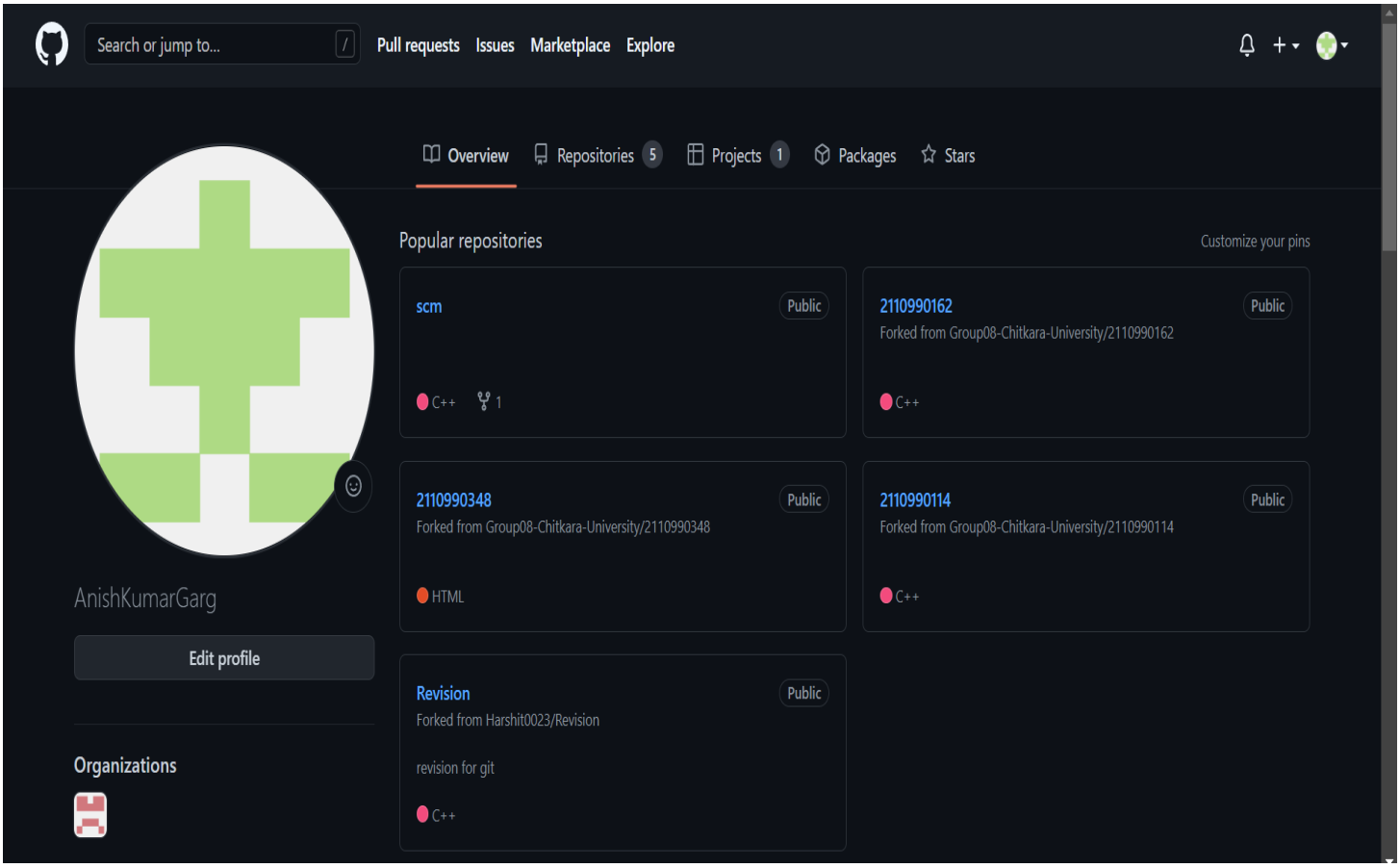
What is Repository?

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.

CONCEPT:

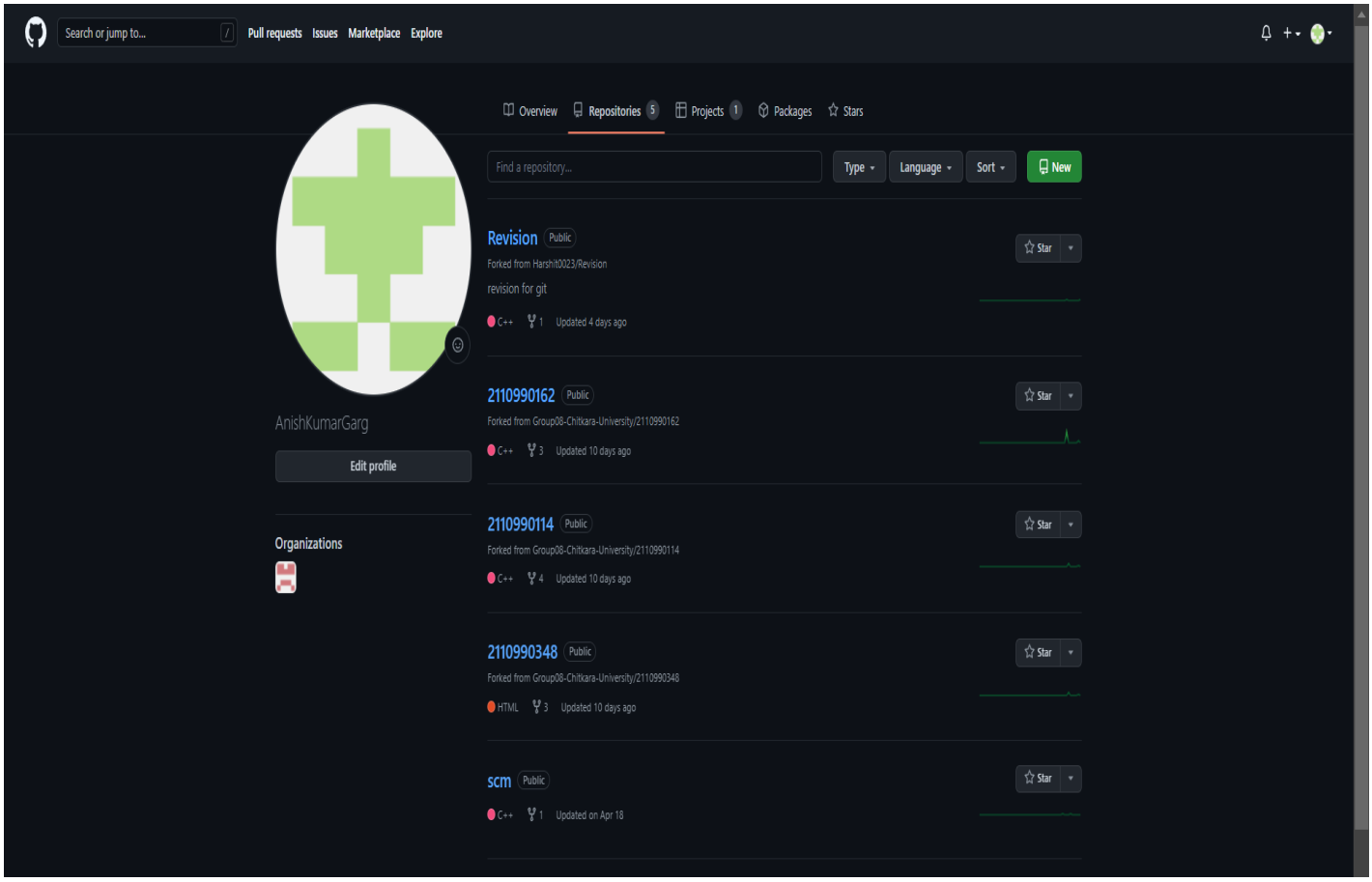
Experiment No. 01

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

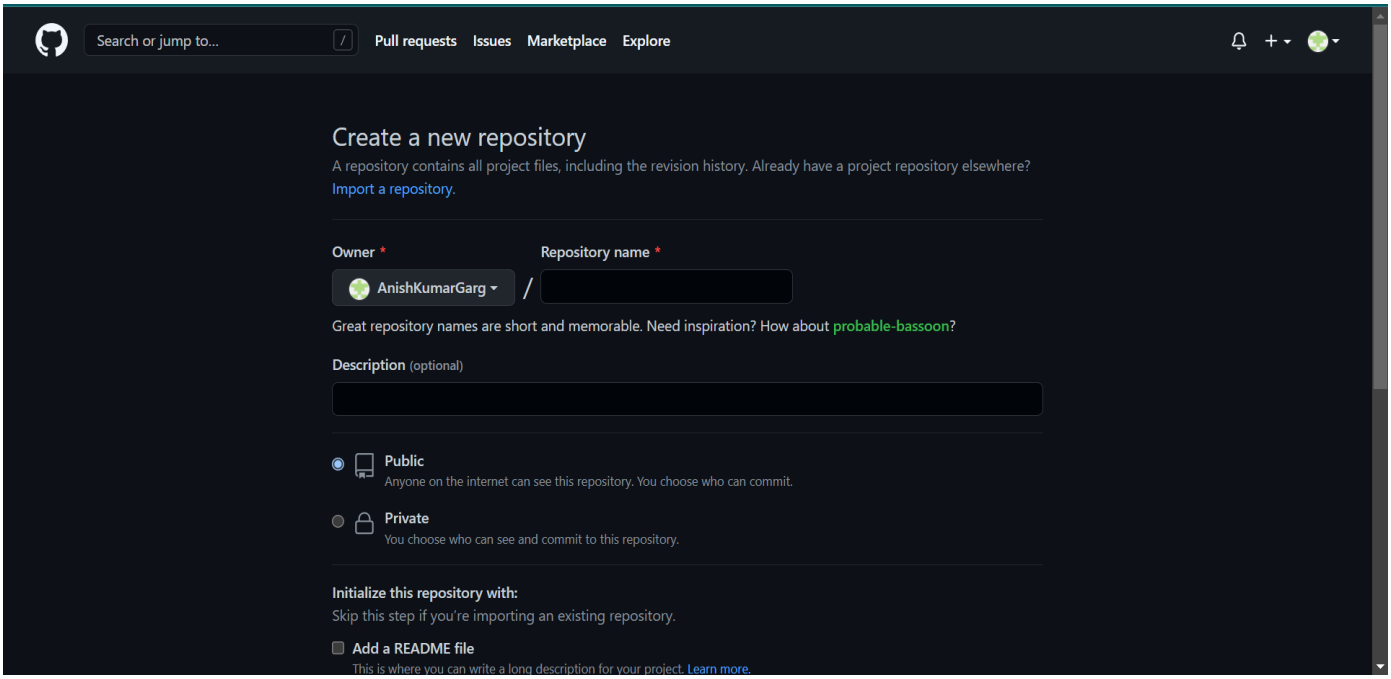


- Login to your Github account and you will land on the homepage as shown below. Click on Repositories option in the menu bar.

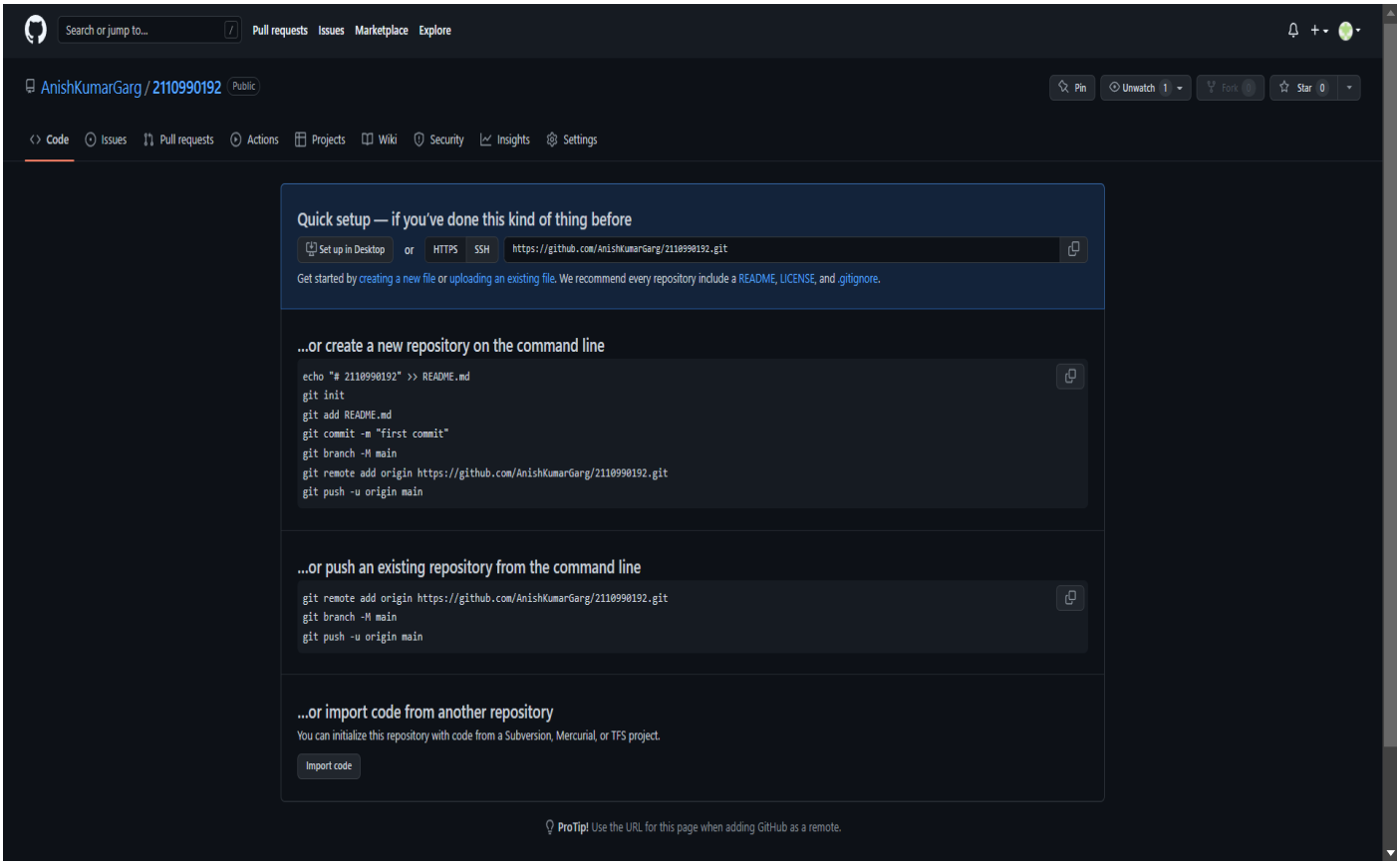
- Click on the 'New' button in the top right corner.



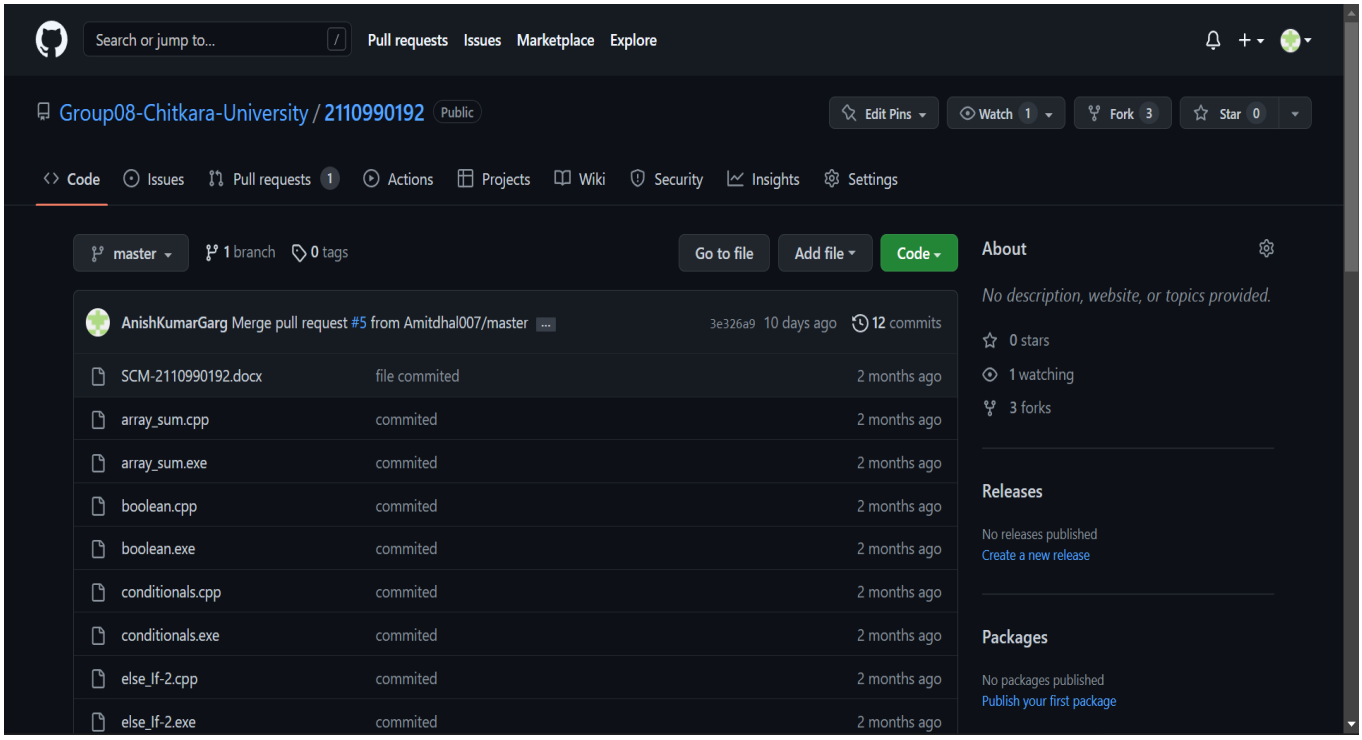
- Enter the Repository name and add the description of the repository.
- Select if you want the repository to be public or private.



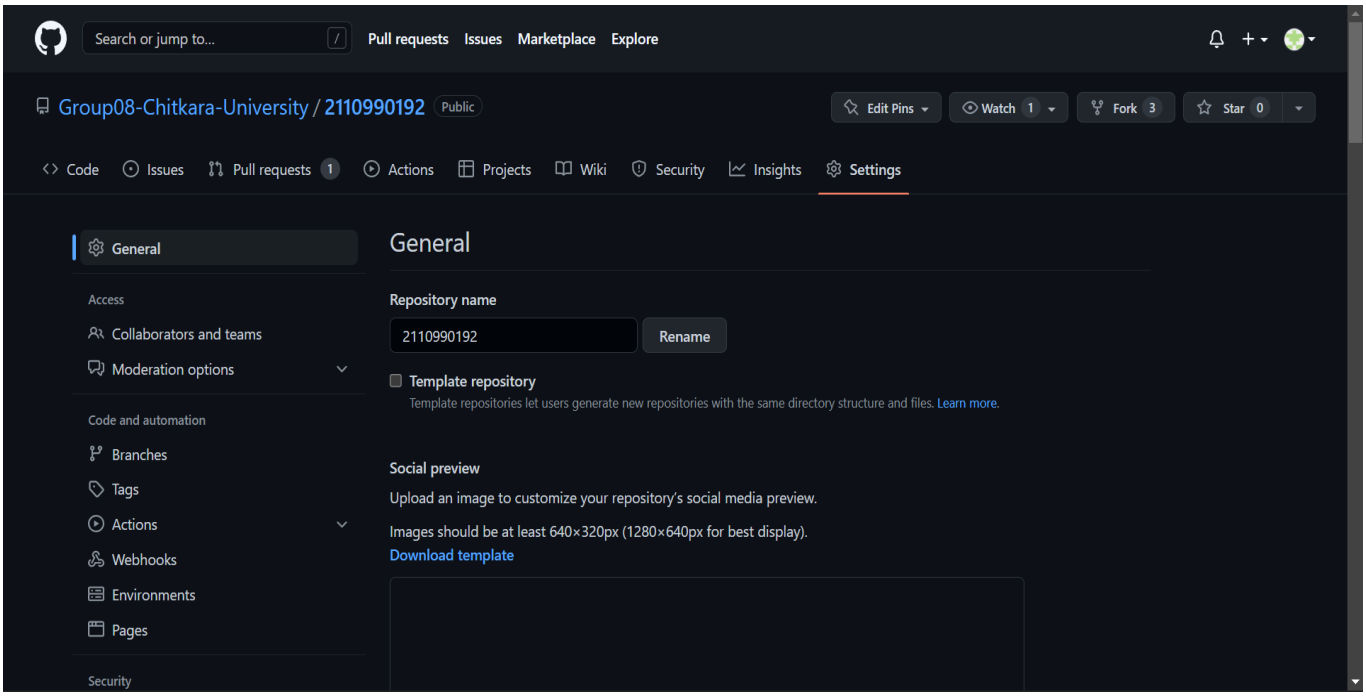
- If you want to import code from an existing repository select the import code option.



- To create a new file or upload an existing file into your repository select the option in the following box.
- Now, you have created your repository successfully.
- To add members to your repository open your repository and select settings option in the navigation bar.



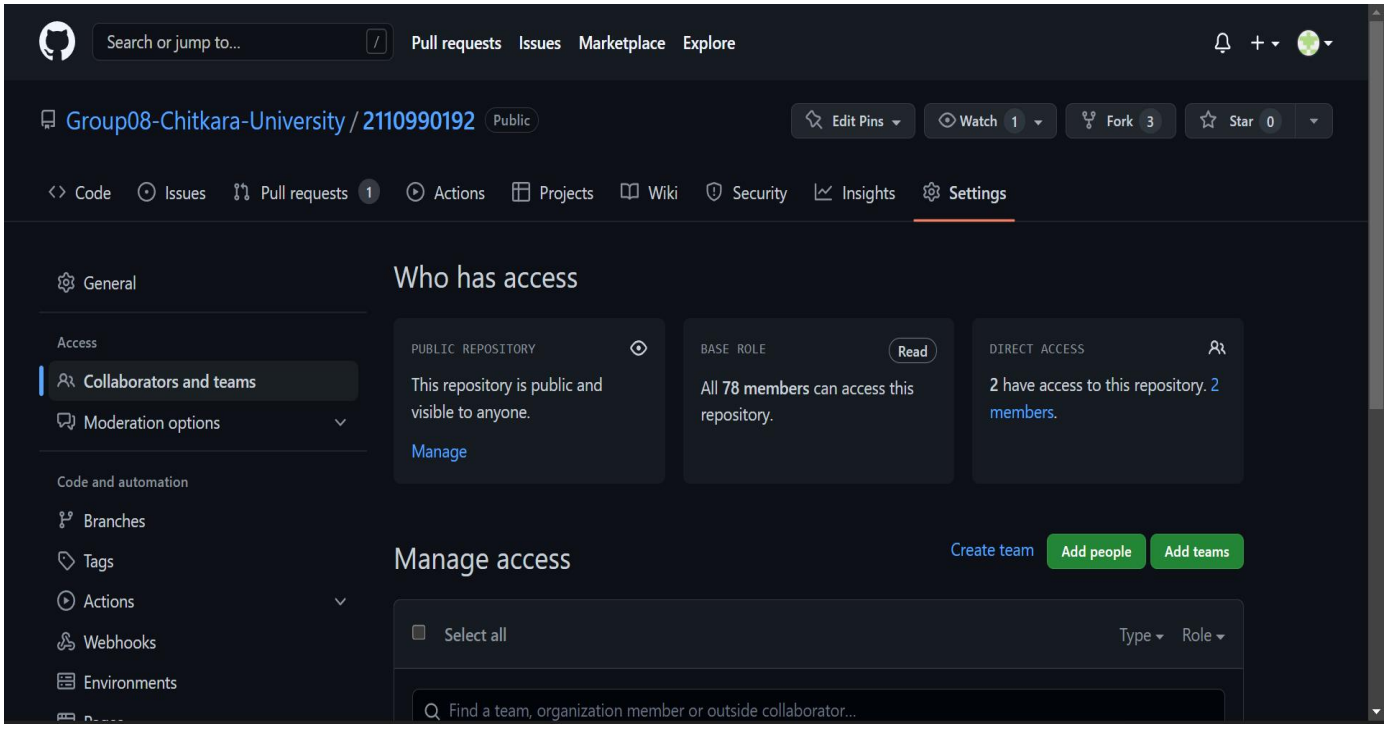
- Click on Collaborators option under the access tab.



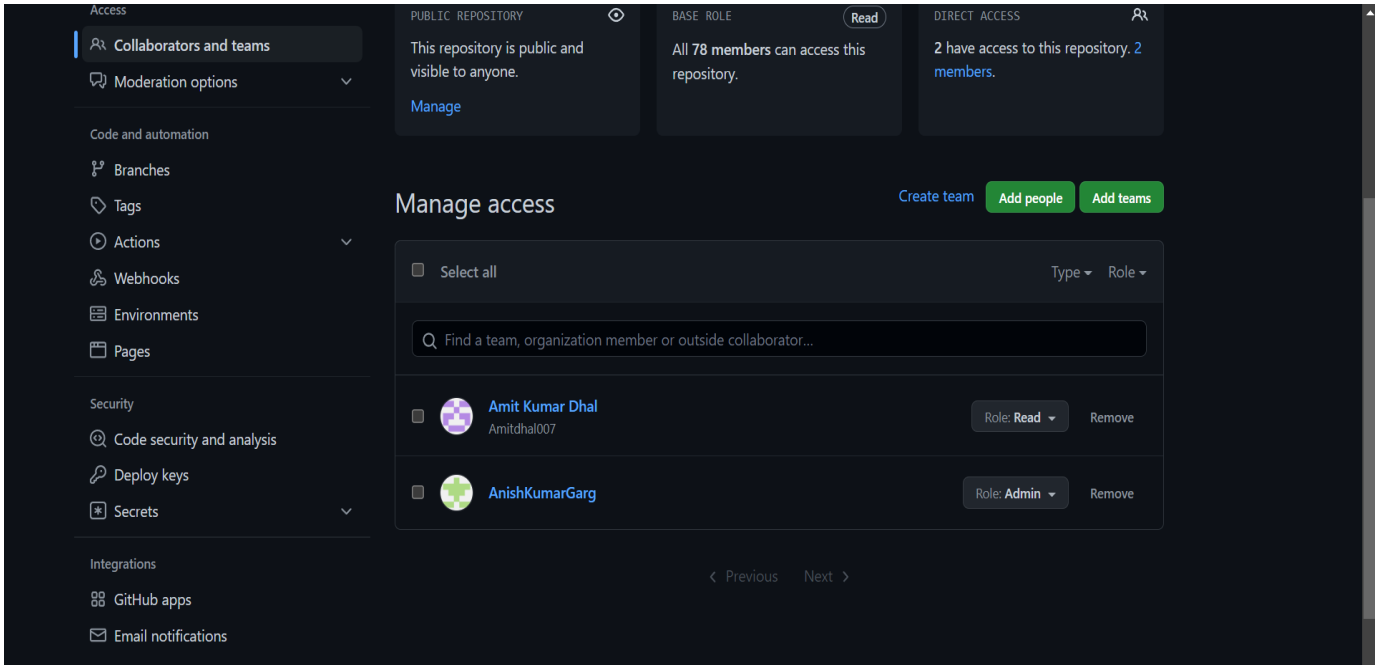
- After clicking on collaborators Github asks you to enter your password to confirm the access to the repository.

- After entering the password you can manage access and add/remove team members to your project.

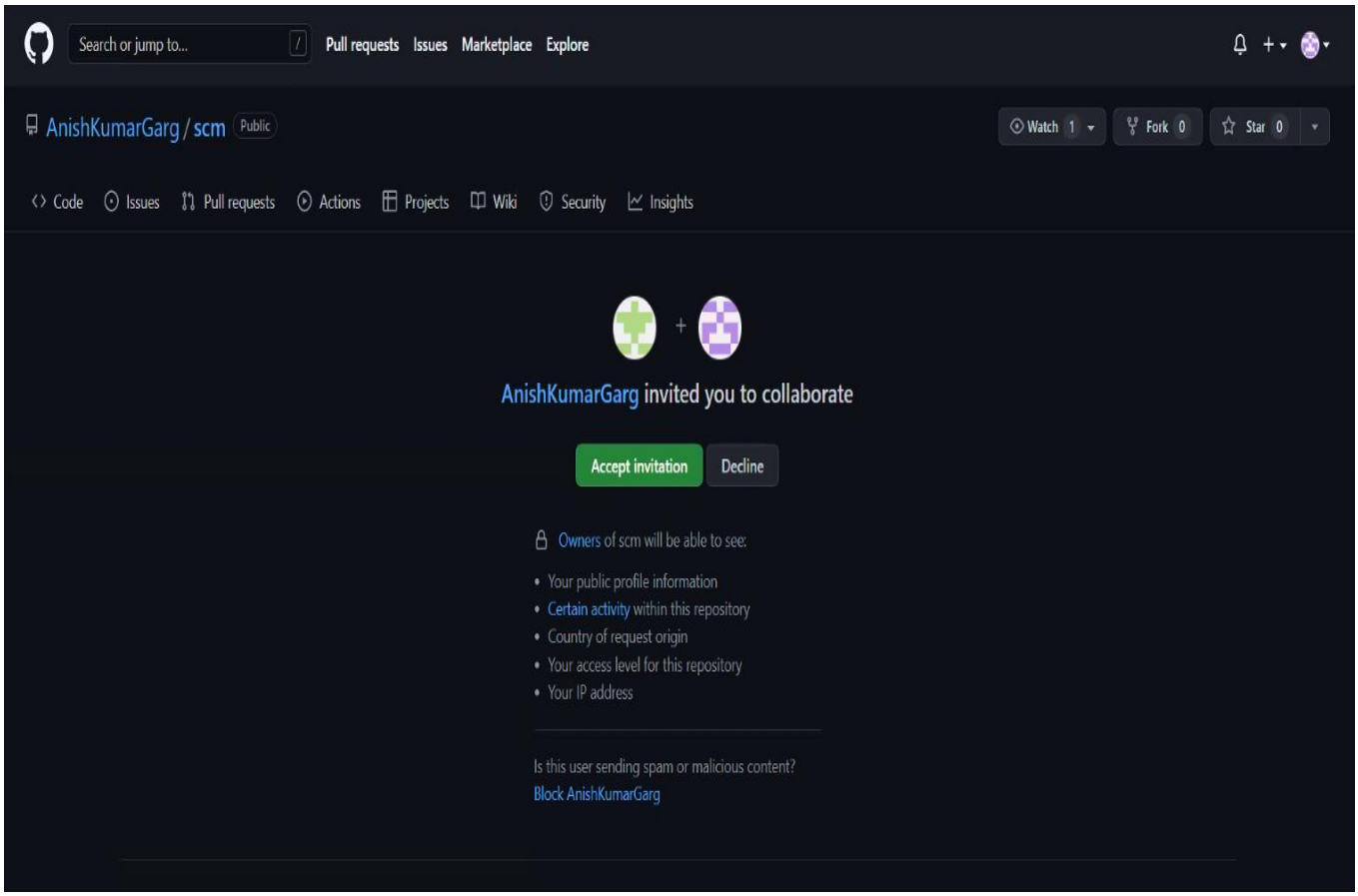
- To add members click on the add people option and search the id of your respective team member.



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



- To accept the invitation from your team member, open your email registered with Github.
- You will receive an invitation mail from the repository owner. Open the email and click on accept invitation.
- You will be redirected to Github where you can either select to accept or decline the invitation.



- You will be shown the option that you are now allowed to push.
- Now all members are ready to contribute to the project.

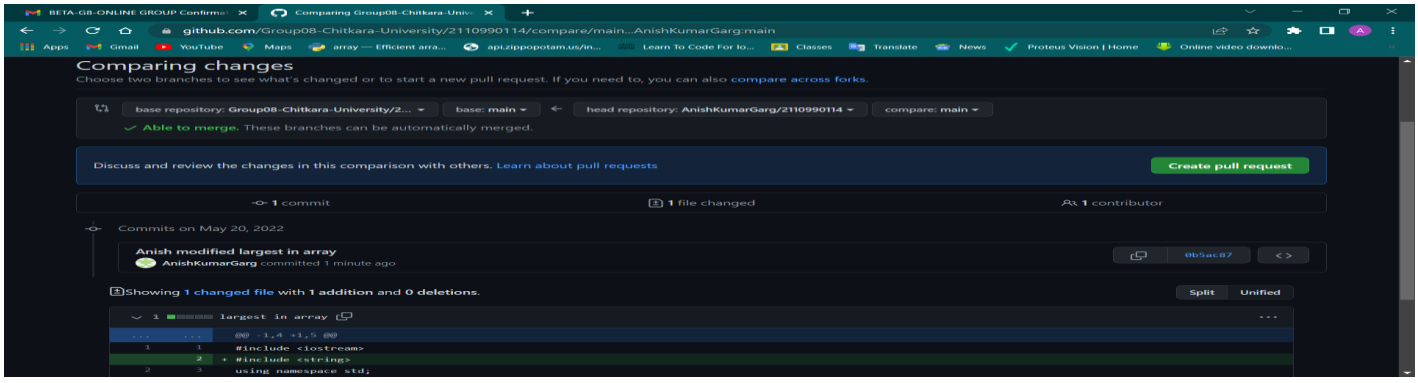
Experiment No. 02

Aim: Open and Close a Pull Request

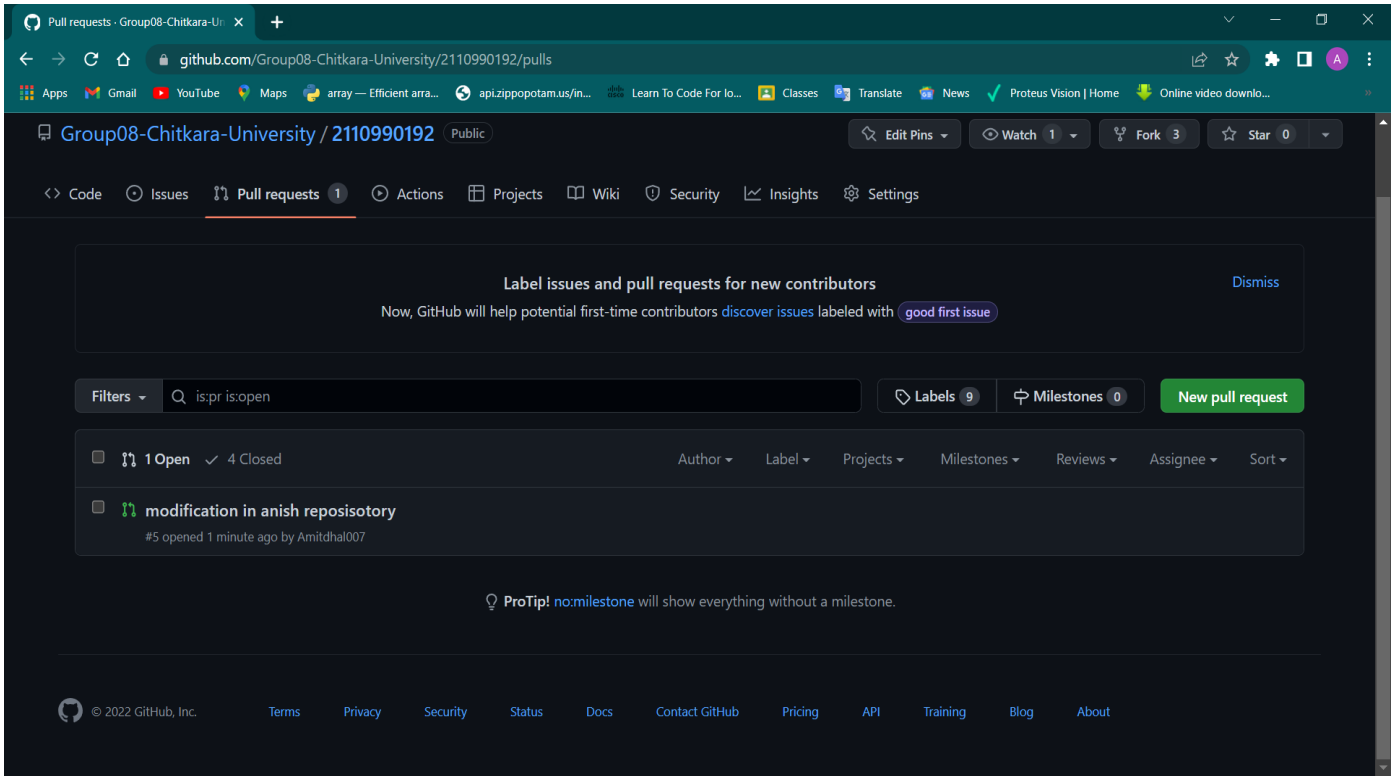
- To open a pull request we first have to make a new branch, by using git branch *branchname* option.
- After making new branch we add a file to the branch or make changes in the existing file.
- Add and commit the changes to the local repository.
- Use git push origin *branchname* option to push the new branch to the main repository.

```
MINGW64:/d/scm/2110990114
 FCN              SCM-2110990114.docx  'array sum'       'largest in array'       'pre and post incre'   transpose
'FINDING SIZE'    addition.cpp         'display grade'   'negative and positive'  'swapping numbers'

lenovo@LAPTOP-GCE1V12S MINGW64 /d/scm/2110990114 (main)
$ vi largest\ in\ array

lenovo@LAPTOP-GCE1V12S MINGW64 /d/scm/2110990114 (main)
$ git add largest\ in\ array

lenovo@LAPTOP-GCE1V12S MINGW64 /d/scm/2110990114 (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   largest in array


lenovo@LAPTOP-GCE1V12S MINGW64 /d/scm/2110990114 (main)
$ git commit -m "Anish modified largest in array"
[main 0b5ac87] Anish modified largest in array
 1 file changed, 1 insertion(+)

lenovo@LAPTOP-GCE1V12S MINGW64 /d/scm/2110990114 (main)
$ git push -u origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 12 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 331 bytes | 331.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/AnishKumarGarg/2110990114.git
   4cba53e..0b5ac87  main -> main
branch 'main' set up to track 'origin/main'.

lenovo@LAPTOP-GCE1V12S MINGW64 /d/scm/2110990114 (main)
```

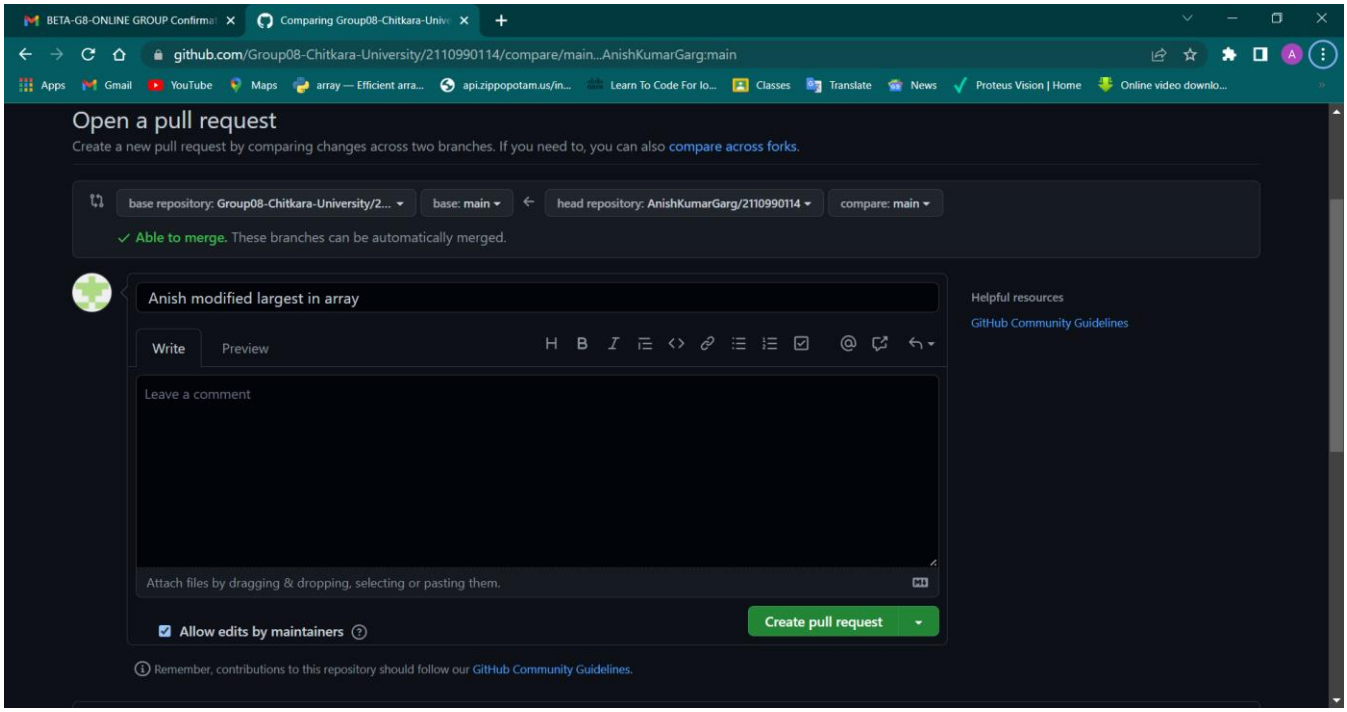
- After pushing new branch Github will either automatically ask you to create a pull request or you can create your own pull request.



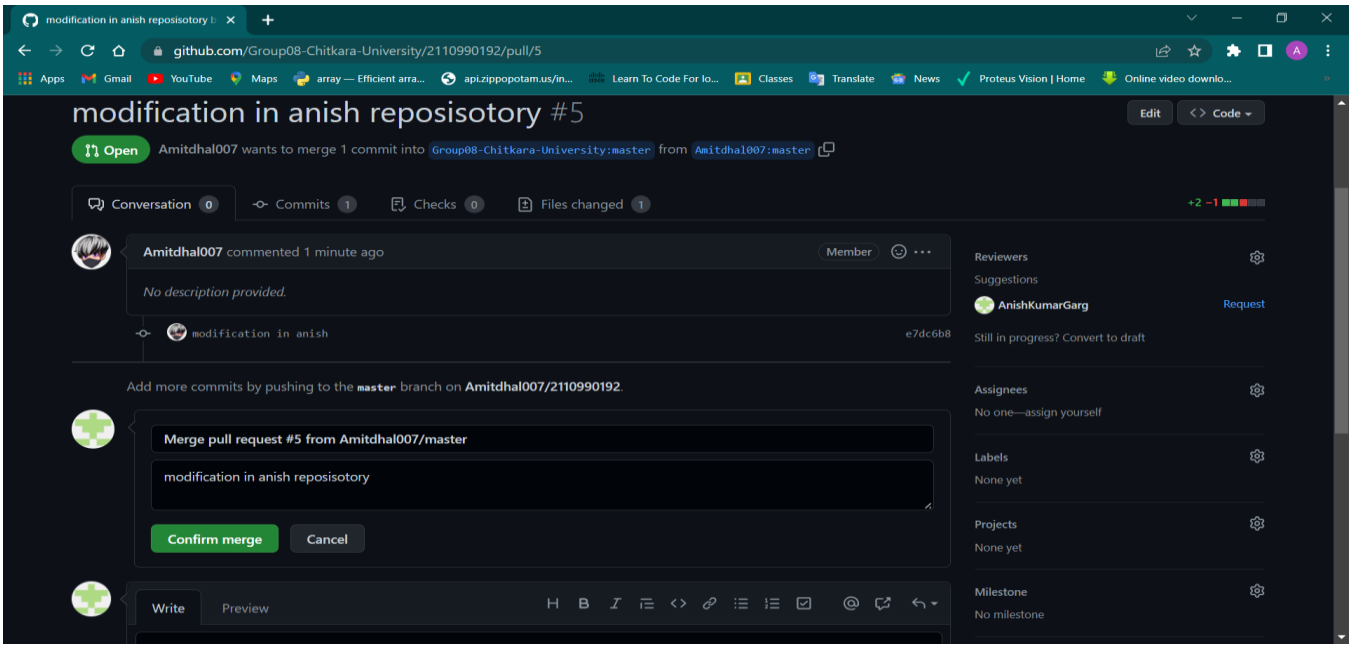
- To create your own pull request click on pull request option.



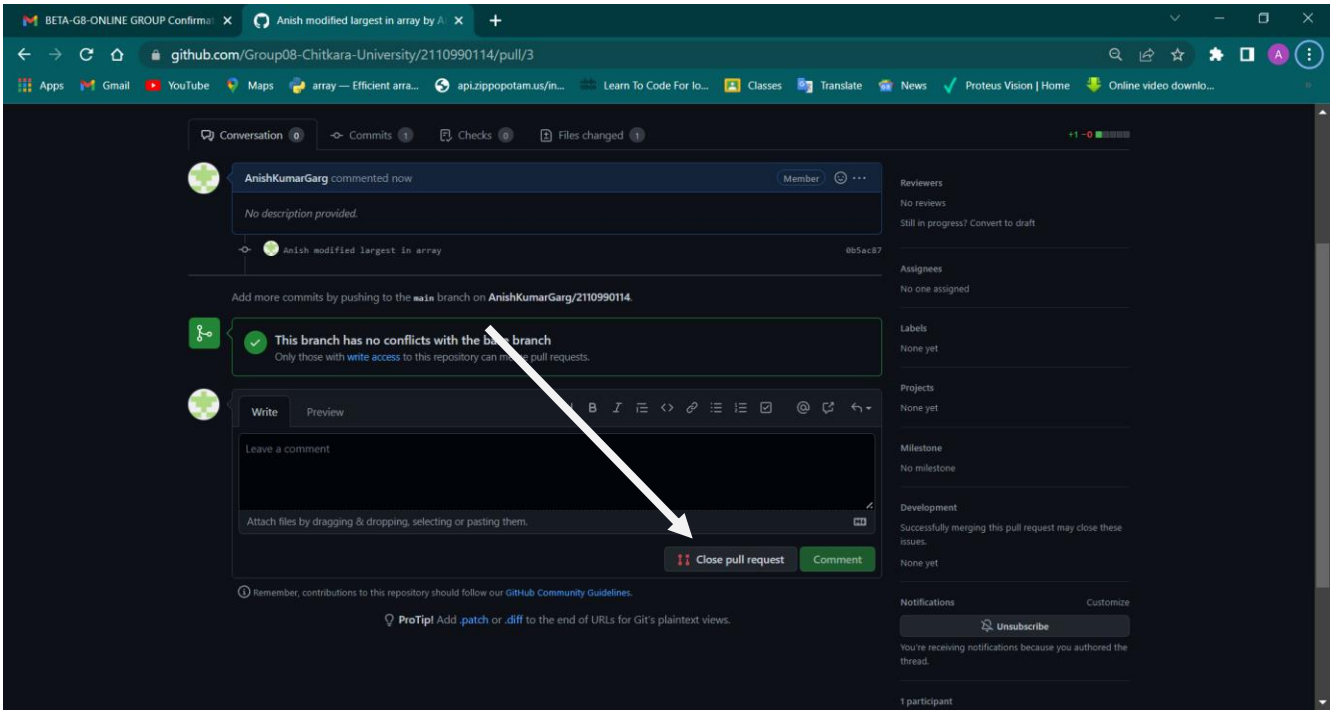
- Github will detect any conflicts and ask you to enter a description of your pull request.



- After opening a pull request all the team members will be sent the request if they want to Merge or close the request.



- If the team member chooses not to merge your pull request they will close you're the pull request.
- To close the pull request simply click on close pull request and add comment/ reason why you closed the pull request.



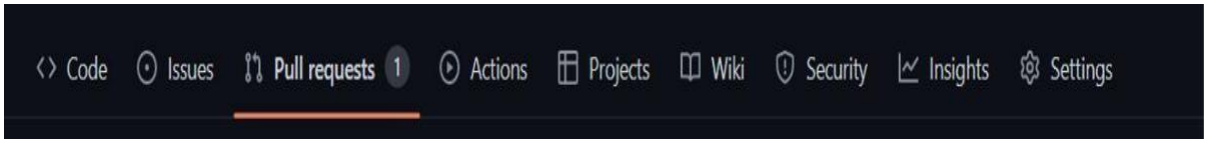
- You can see the entire pull request generated and how they were dealt with by clicking on pull request option.

Experiment No. 03

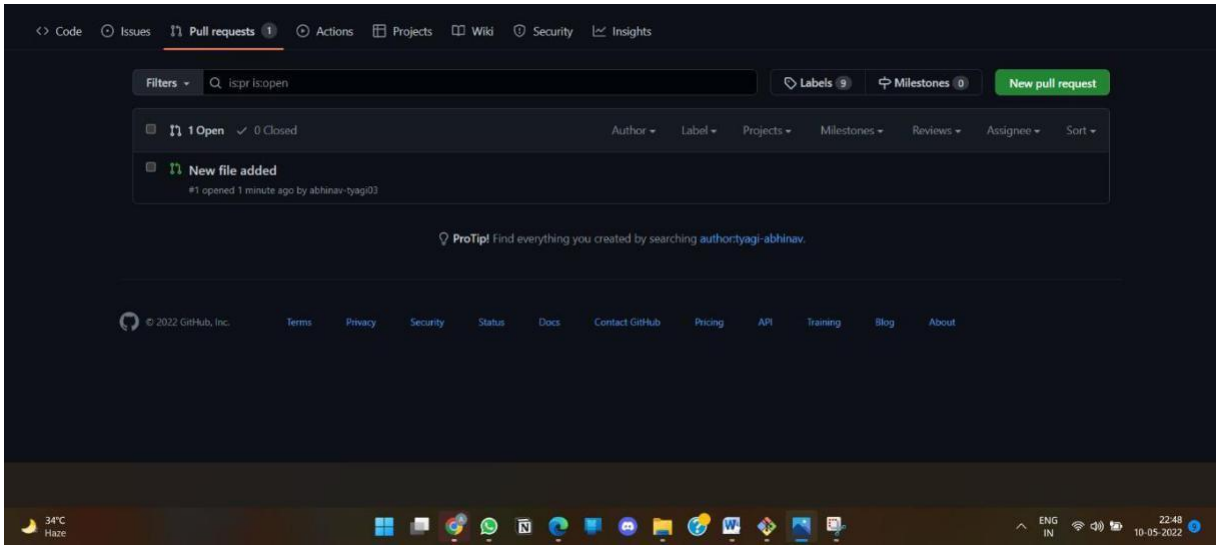
Aim: Create a pull request on a team member's repo and close pull requests generated by team members on own Repo as a maintainer.

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:-

- Do the required changes in the repository, add and commit these changes in the local repository in a new branch.
- Push the modified branch using git push origin *branchname*.
- 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.

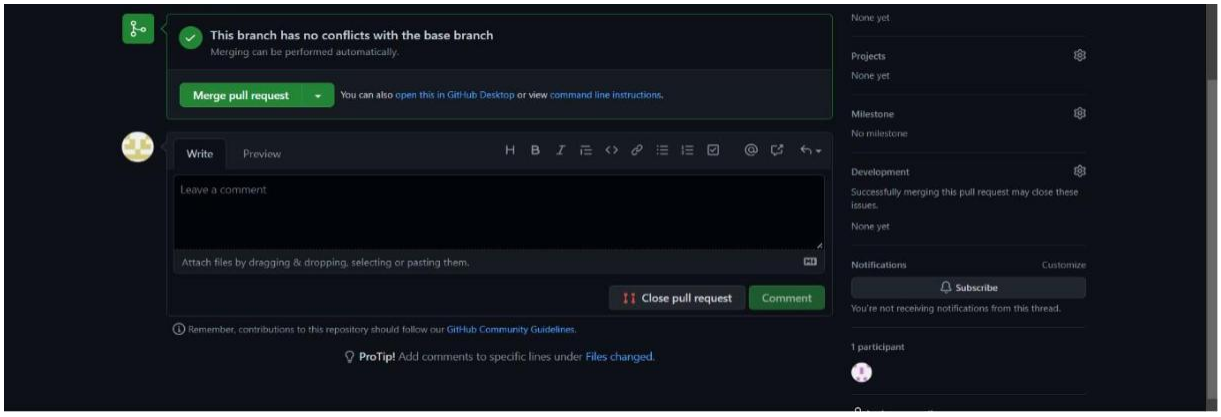


- Click on it. The pull request generated by you will be visible to them.

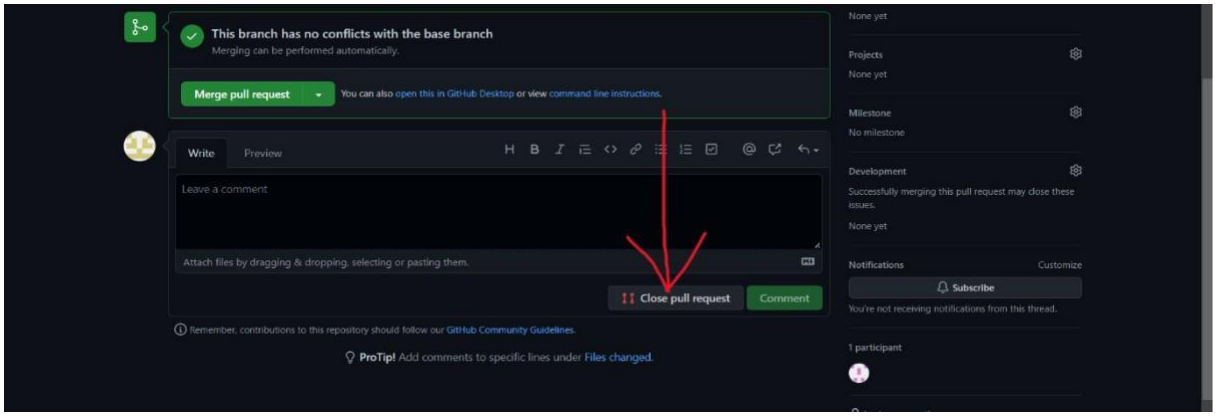


- Click on the pull request. Two options will be available, either to close the pull request or Merge the request with the main branch.

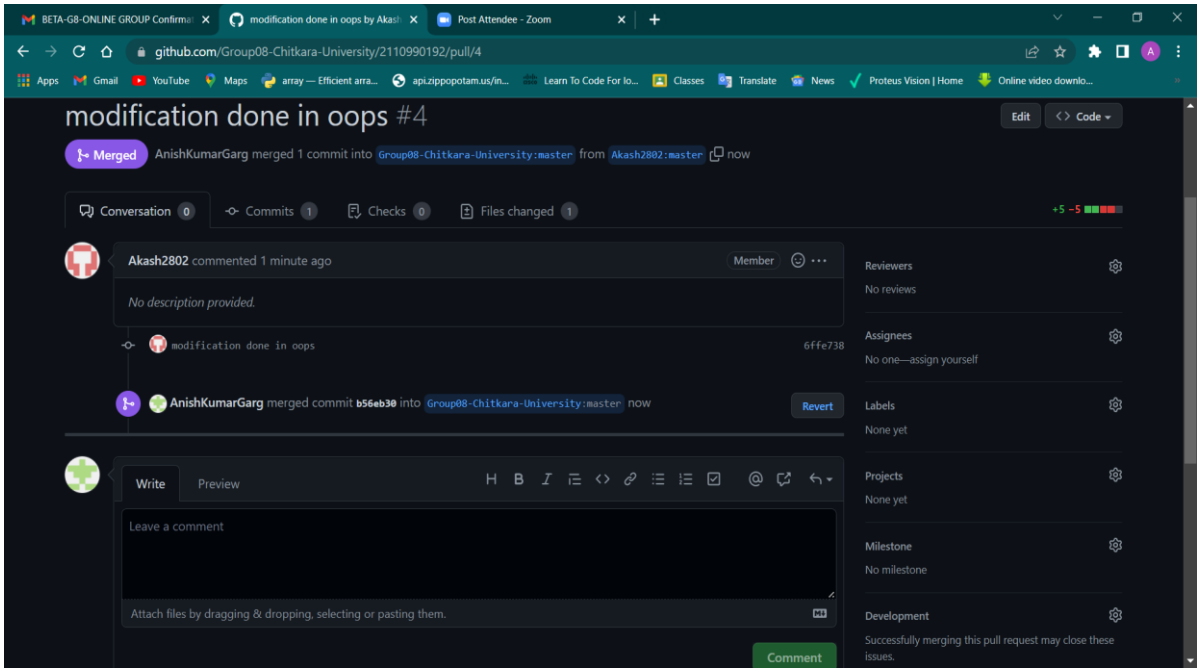
- By selecting the merge branch option the main branch will get updated for all the team members.



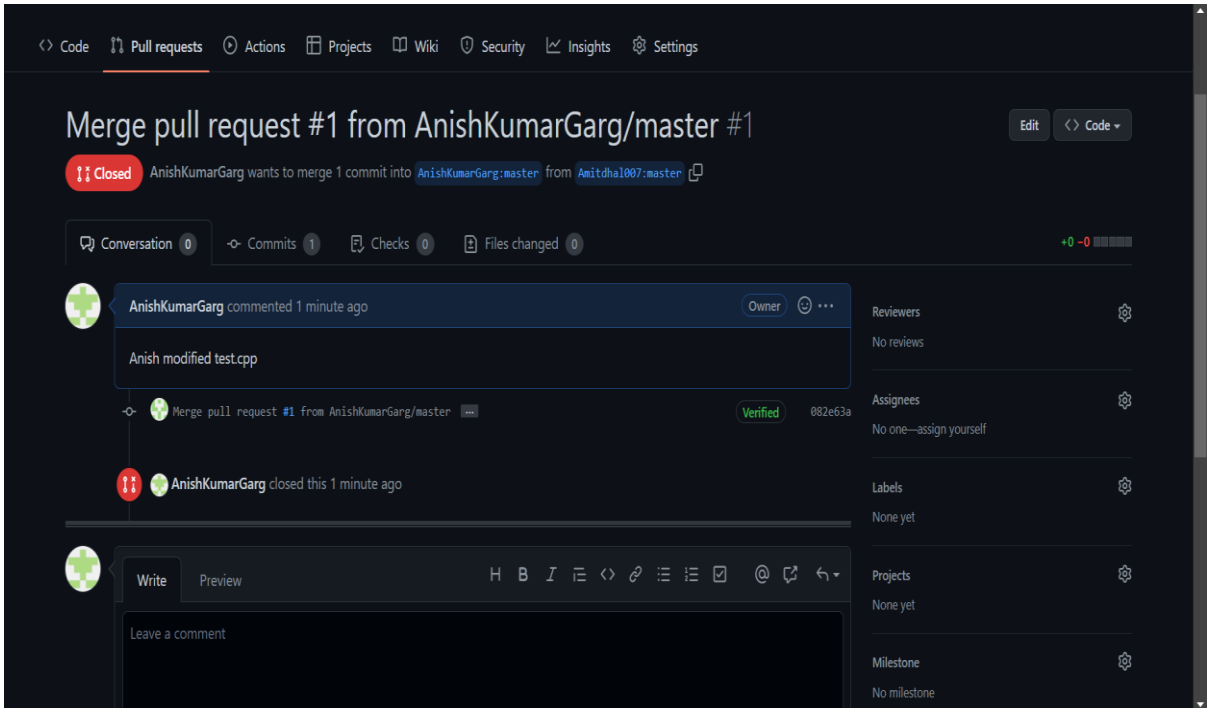
- By selecting close the pull request the pull request is not accepted and not merged with main branch.



- The process is similar to closing and merging the pull request by you. It simply includes an external party to execute.
- The result of merging the pull request is shown below.



- The result of closing the request is shown below.



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

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.

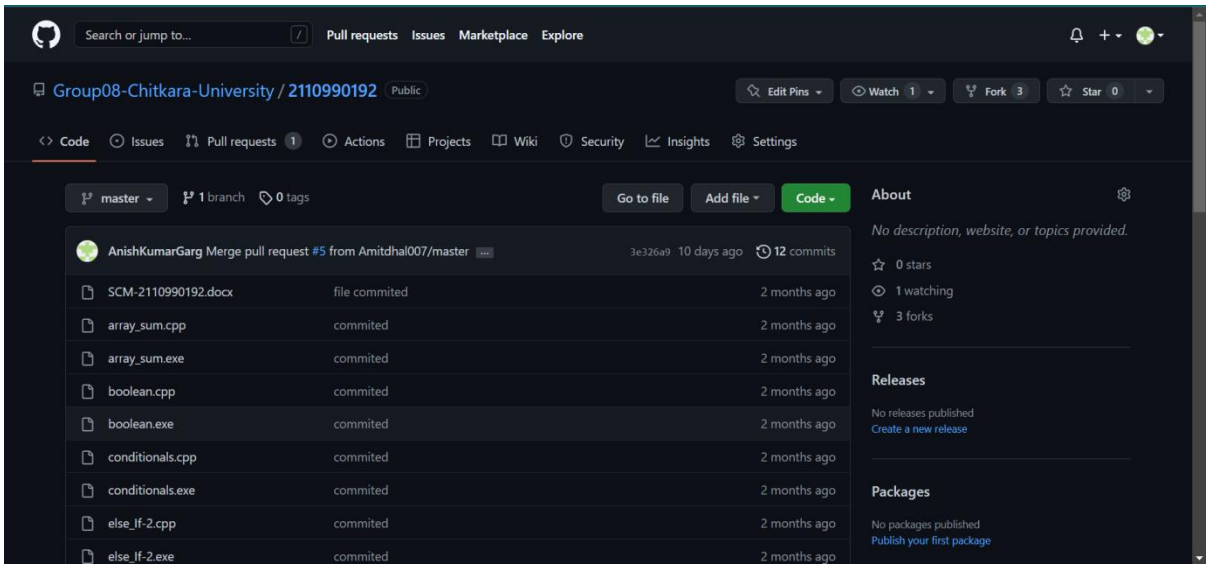
A repository's graphs give you information on traffic, projects that depend on the repository, contributors and commits to the repository, and a repository's forks and network. If you maintain a repository, you can use this data to get a better understanding of who's using your repository and why they're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

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

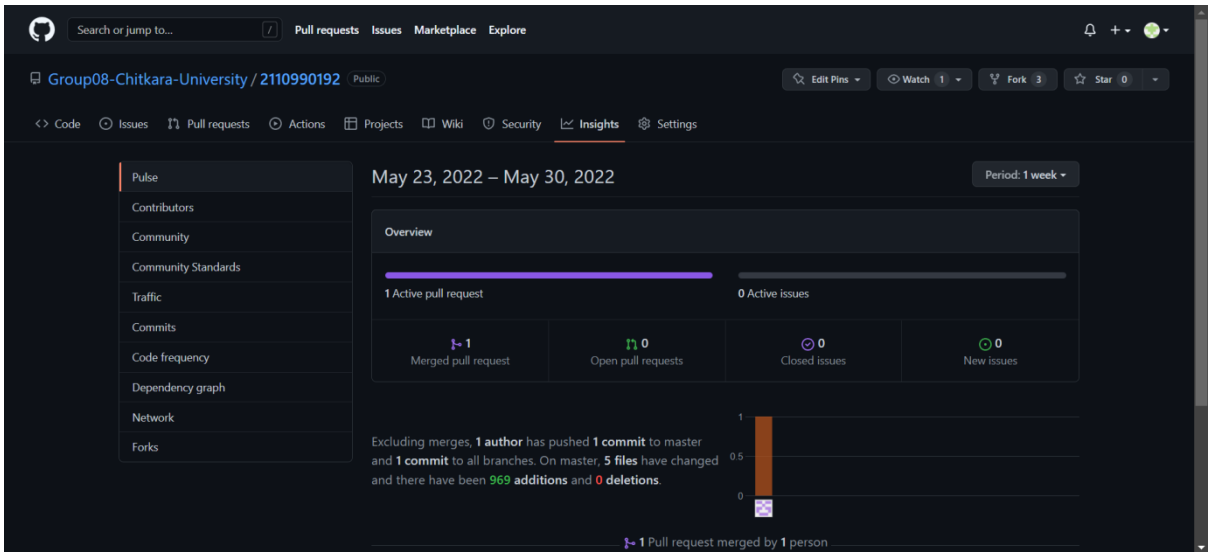
- Pulse
- Contributors
- Traffic
- Commits
- Code frequency
- Network

Steps to access network graphs of respective repository:

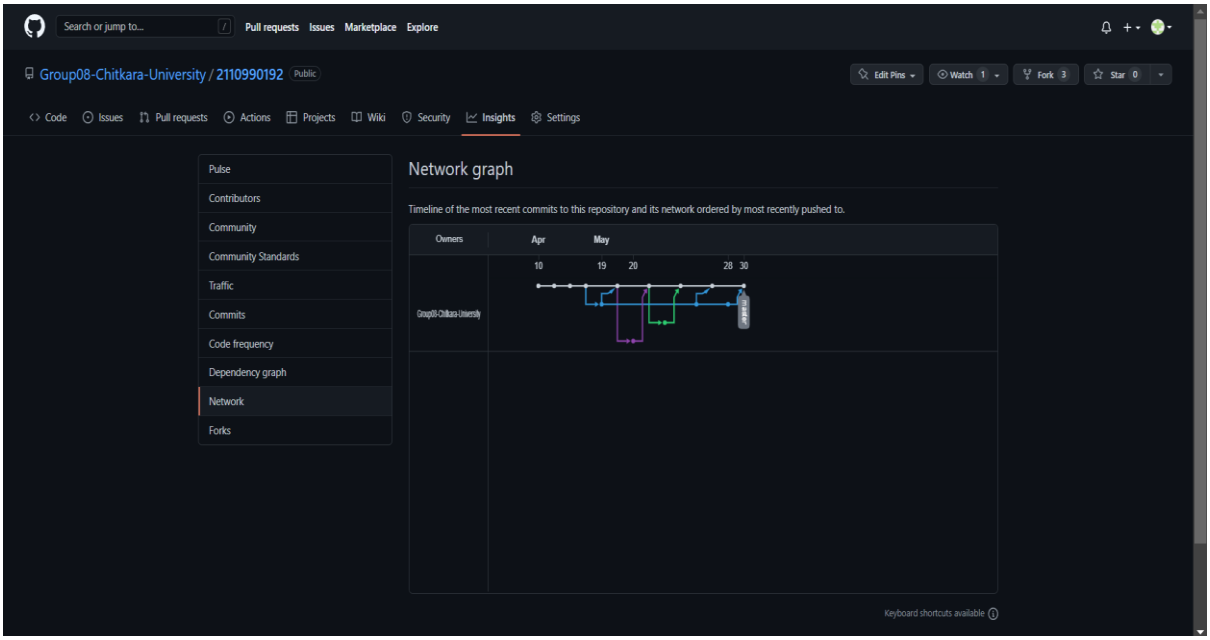
1. On GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click **Insights**.



3. at the left sidebar, click on **Network**.



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.

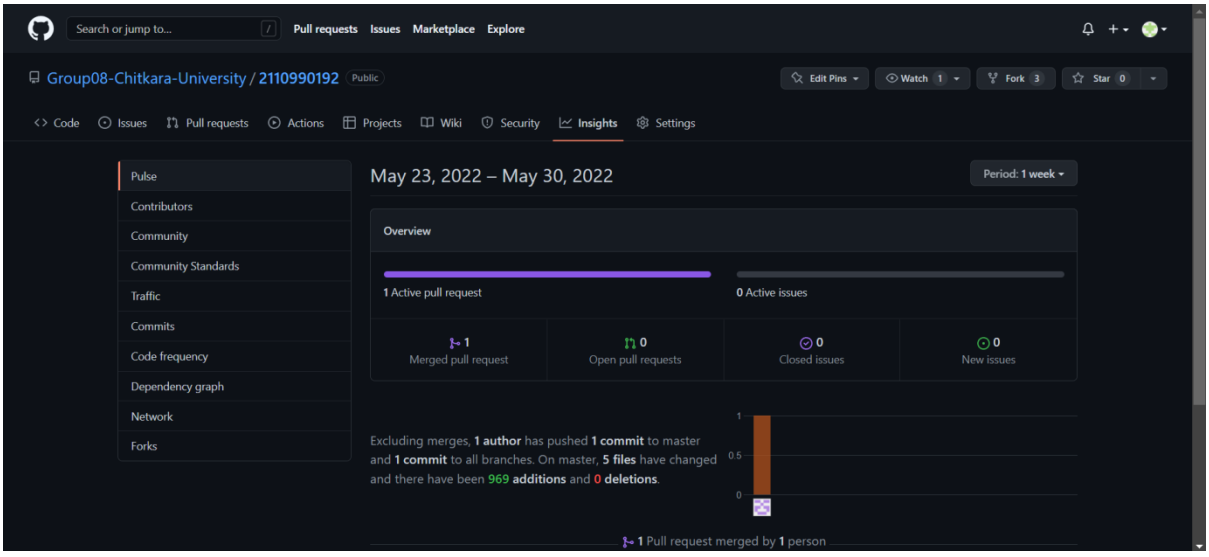


Listing the forks of a repository:

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

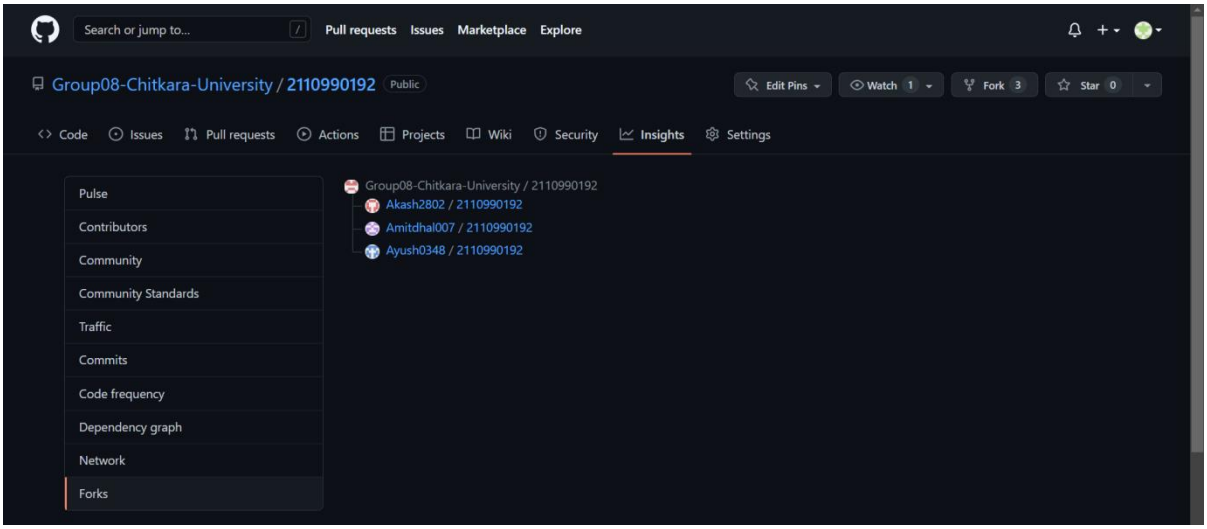
Clicking the number of forks shows you the full network. From there you can click "members" to see who forked the repo

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**.

Here you can see all the forks



Viewing the dependencies of a repository:

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

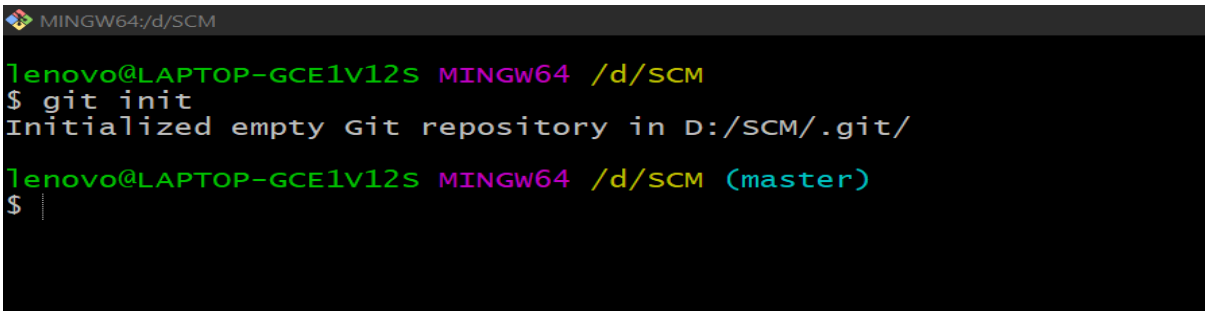
Almost all software relies on code developed and maintained by other developers, often known as a supply chain. For example, utilities, libraries, and frameworks. These dependencies are an integral part of your code and any bugs or vulnerabilities in them may affect your code. It's important to review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

Commands

Some Basic commands of Git :-

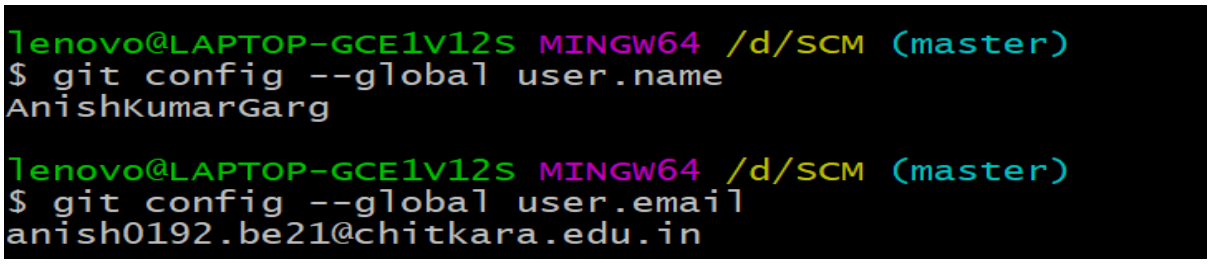
Git init command

This creates a new subdirectory named .git that contains all of your necessary repository files — a Git repository skeleton. At this point, nothing in your project is tracked yet. See Git Internals for 24 more information about exactly what files are contained in the .git directory you just created. If you want to start version-controlling existing files (as opposed to an empty directory), you should probably begin tracking those files and do an initial commit.



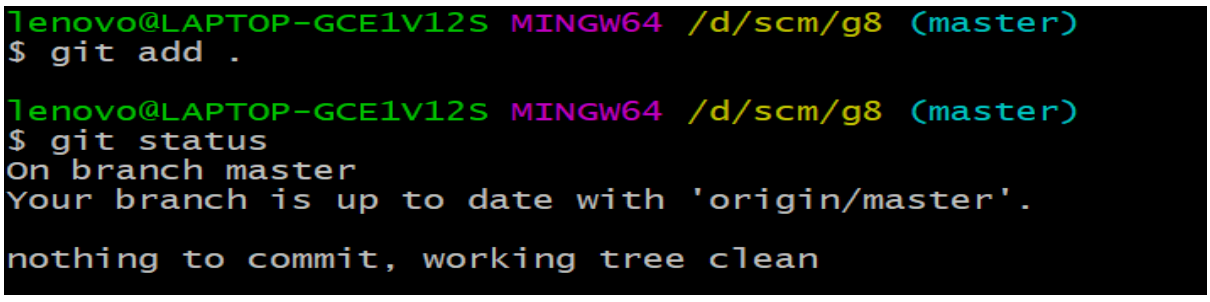
Git config command

This command configures the user. The Git config command is the first and necessary command used on the Git command line. This command sets the author name and email address to be used with your commits. Git config is also used in other scenarios.



Git add command

This command is used to add one or more files to staging (Index) area.



Git commit command

Commit command is used in two scenarios. They are as follows.

```
lenovo@LAPTOP-GCE1V12S MINGW64 /d/scm/g8 (master)
$ git commit -m "commited"
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
```

This command changes the head. It records or snapshots the file permanently in the version history with a message.

Git status command

The status command is used to display the state of the working directory and the staging area. It allows you to see which changes have been staged, which haven't, and which files aren?t being tracked by Git. It does not show you any information about the committed project history. For this, you need to use the git log. It also lists the files that you've changed and those you still need to add or commit.

```
lenovo@LAPTOP-GCE1V12S MINGW64 /d/scm/g8 (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
```

Git push Command

It is used to upload local repository content to a remote repository. Pushing is an act of transfer commits from your local repository to a remote repo. It's the complement to git fetch, but whereas fetching imports commits to local branches on comparatively pushing exports commits to remote branches. Remote branches are configured by using the git remote command. Pushing is capable of overwriting changes, and caution should be taken when pushing.

Git push command can be used as follows.

Git push origin master

This command sends the changes made on the master branch, to your remote repository.

```
lenovo@LAPTOP-GCE1V12S MINGW64 /d/scm/g8 (master)
$ git remote add origin https://github.com/AnishKumarGarg/Anish-G8.git

lenovo@LAPTOP-GCE1V12S MINGW64 /d/scm/g8 (master)
$ git push -u origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 302 bytes | 302.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/AnishKumarGarg/Anish-G8.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
```

Git push -all

This command pushes all the branches to the server repository.

```
$ git push --all
Everything up-to-date
```

Git pull command

Pull command is used to receive data from GitHub. It fetches and merges changes on the remote server to your working directory.

```
lenovo@LAPTOP-GCE1V12S MINGW64 /d/scm/g8 (master)
$ git pull
Already up to date.
```

Git Branch Command

```
lenovo@LAPTOP-GCE1V12S MINGW64 /d/scm/g8 (master)
$ git branch main
```

This is used to create a new branch.

Git Merge Command

```
lenovo@LAPTOP-GCE1V12S MINGW64 /d/scm/g8 (master)
$ git merge main
Already up to date.
```

This command is used to merge the specified branch's history into the current branch.

Git clone command

This command is used to clone or copy a repository from a URL. This URL generally is a bitbucket server, a stash or any other version control and source code management repository holding service.

```
lenovo@LAPTOP-GCE1V12S MINGW64 /d/scm
$ git clone https://github.com/AnishKumarGarg/2110990114.git
Cloning into '2110990114'...
remote: Enumerating objects: 17, done.
remote: Counting objects: 100% (17/17), done.
remote: Compressing objects: 100% (16/16), done.
remote: Total 17 (delta 0), reused 3 (delta 0), pack-reused 0
Receiving objects: 100% (17/17), 3.83 MiB | 3.68 MiB/s, done.
```

Git branch –d [branch name]

It is used to delete the current branch name specified.

```
lenovo@LAPTOP-GCE1V12S MINGW64 /d/scm/g8 (master)
$ git branch -d main
Deleted branch main (was 53b01cd).
```

Git log Command

This command is used to check the commit history.

```
lenovo@LAPTOP-GCE1V12S MINGW64 /d/scm/g8 (master)
$ git log
commit 53b01cdc9d1167b985c314001201714eccfaaa78 (HEAD -> master, origin/master)
Author: AnishKumarGarg <anish0192.be21@chitkara.edu.in>
Date:   Mon May 30 22:25:53 2022 +0530

    commited
```

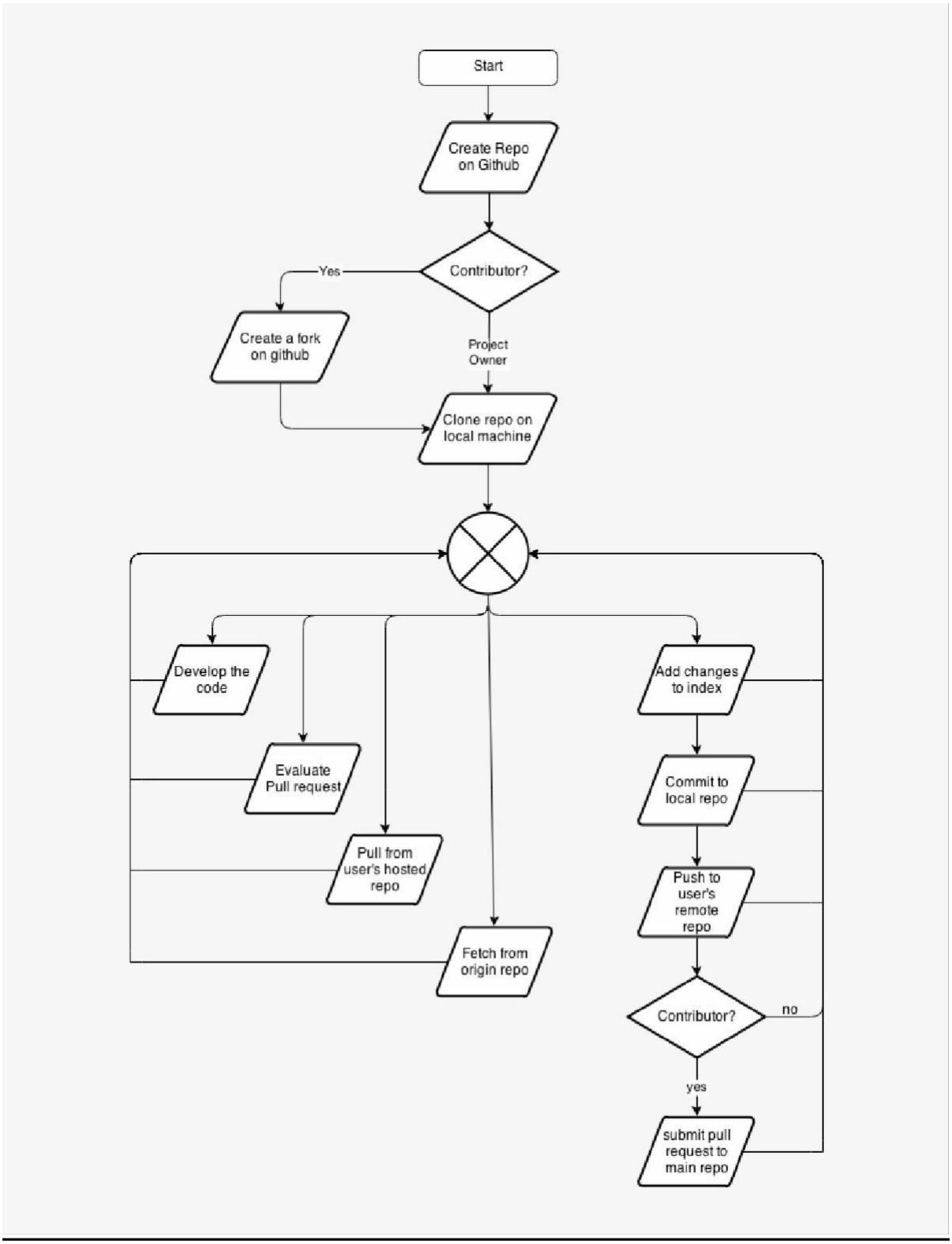
By default, if no argument passed, Git log shows the most recent commits first. We can limit the number of log entries displayed by passing a number as an option, such as -3 to show only the last three entries.

Git remote Command

Git Remote command is used to connect your local repository to the remote server. This command allows you to create, view, and delete connections to other repositories. These connections are more like bookmarks rather than direct links into other repositories. This command doesn't provide real-time access to repositories.

```
lenovo@LAPTOP-GCE1V12S MINGW64 /d/scm/g8 (master)
$ git remote add origin https://github.com/AnishKumarGarg/Anish-G8.git
error: remote origin already exists.
```

Workflow Chart



Discussion

In this project one of the participant added a c++,HTML project through the commands using CLI software Git Bash and the other participants edited the project to make it more convenient and debugged the codes of the project by extraxting the project from Git Hub using clone command , commited the changes and pushed the project to Git Hub account. After that it was reflected to all the Collaborators of the repository.

Reference

https://github.com/

https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

https://git-scm.com/book/en/v2/GitHub-Account-Setup-and-Configuration

https://git-scm.com/book/en/v2/Customizing-Git-Git-Attributes

Subject Name: **Source Code Management**

Subject Code: **CS181**

Cluster: **Beta**

Department: **DCSE**

Submitted By:
Akshay Jha
2110990127
G02

Submitted To:
Dr. Deepak
Thakur

 # List of Programs #

Experiment No. 06

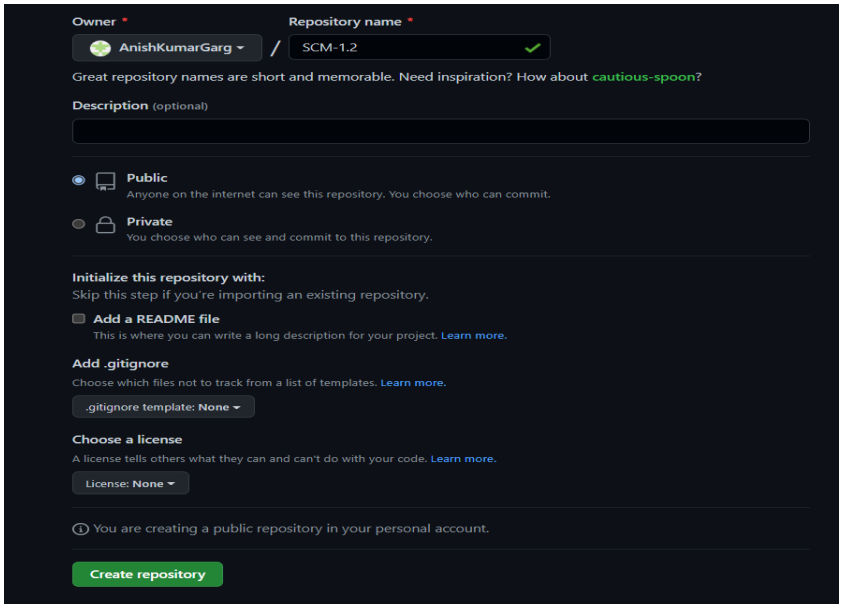
Aim: Add collaborators on Github Repo

Theory:

1. Create a New Repository.

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

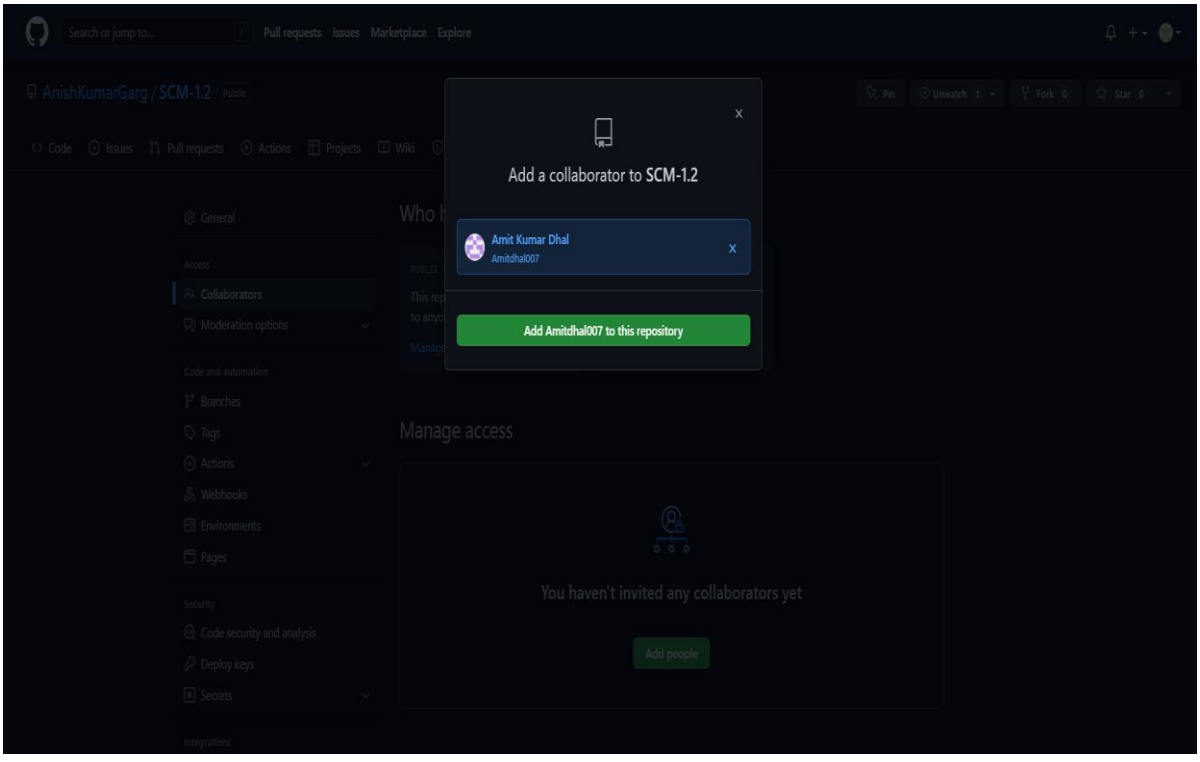
   When you create a new repository, you should initialize the repository with a README file to let people know about your project. For more information, see "Creating a new repository."



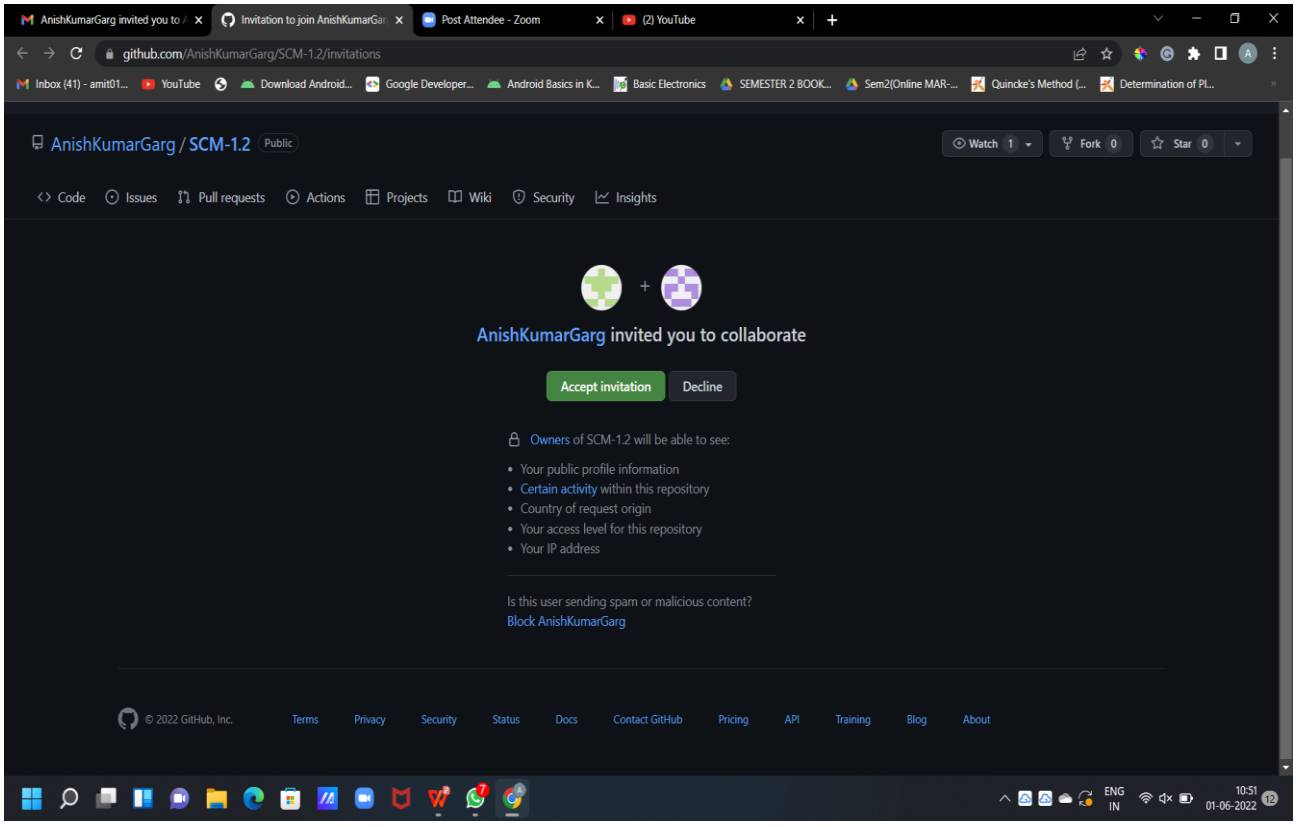
2. Now Copy the HTTP link of your repo and paste it on your 'Git CLI', and merge the local repo in remote repo .

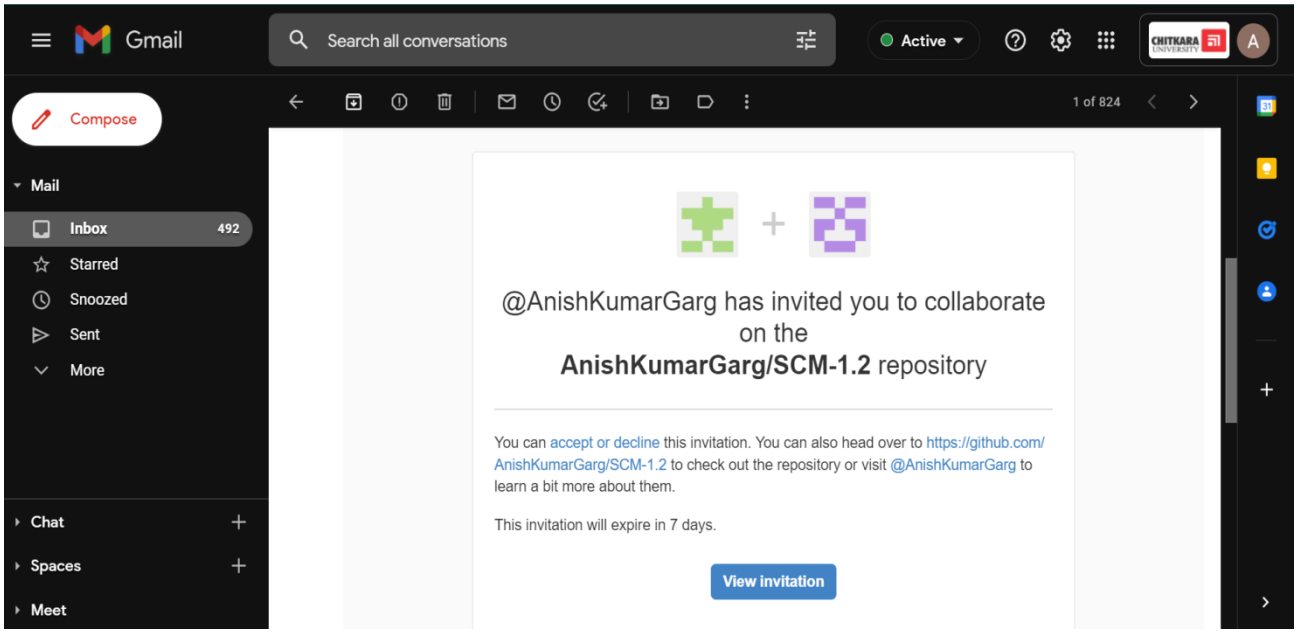
```
MINGW64:/d/SCM/g8

lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/g8
$ git init
Initialized empty Git repository in D:/SCM/g8/.git/

lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/g8 (master)
$ git add test.cpp
warning: LF will be replaced by CRLF in test.cpp.
The file will have its original line endings in your working directory

lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/g8 (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   test.cpp


lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/g8 (master)
$ git commit -m "added test.cpp"
[master (root-commit) 957c9e7] added test.cpp
 1 file changed, 7 insertions(+)
 create mode 100644 test.cpp

lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/g8 (master)
$ git remote add origin https://github.com/AnishKumarGarg/SCM-1.2.git
git push -u origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 309 bytes | 309.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/AnishKumarGarg/SCM-1.2.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
```

3. Go to Collaborators in Repo Setting, Add the username or email of Collaborator you want to add in your Repo.

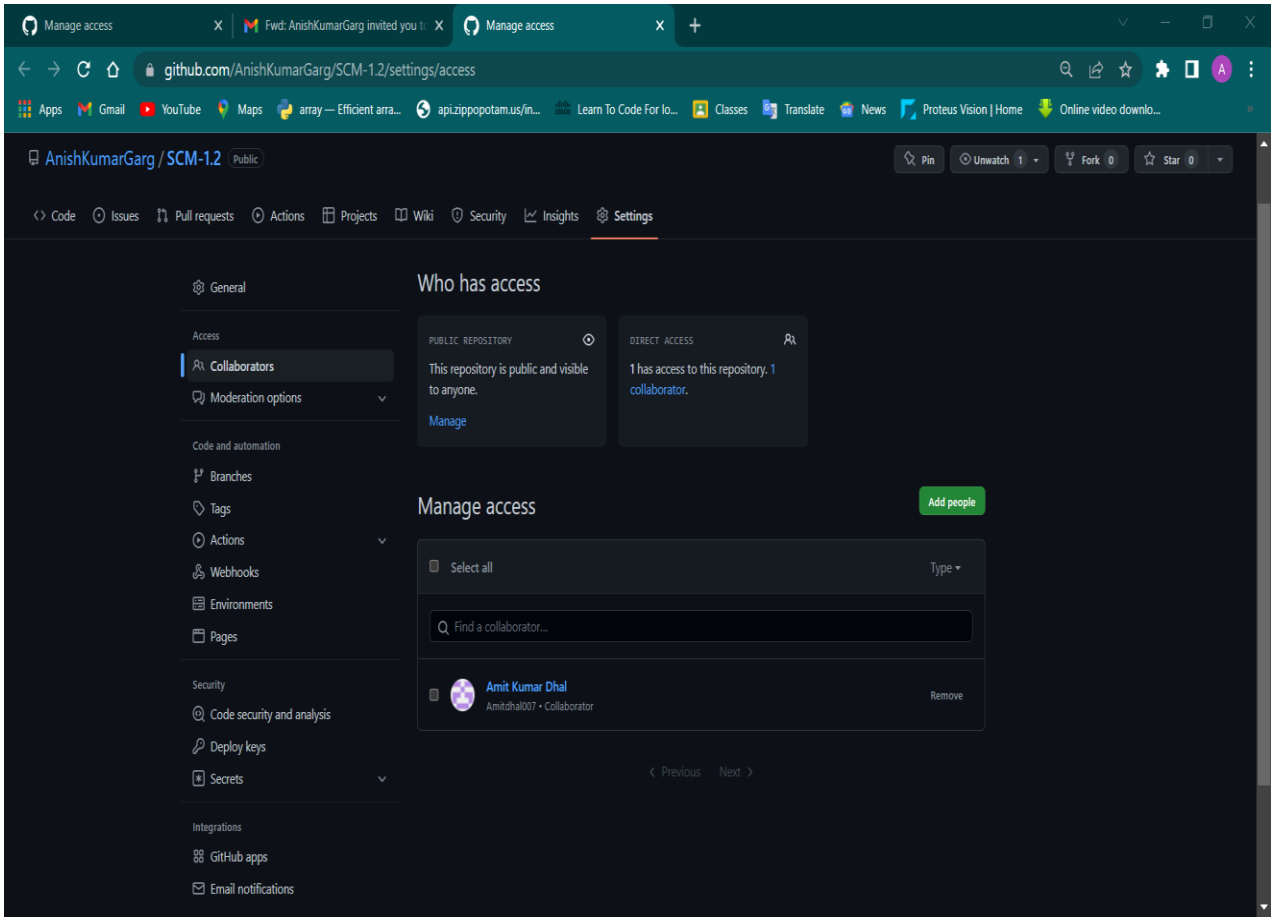


4. Invitation Mail is sent to the Collaborator, the collaborator has to accept this Invitation.





4. New Collaborator has now access to the SCM Repo.



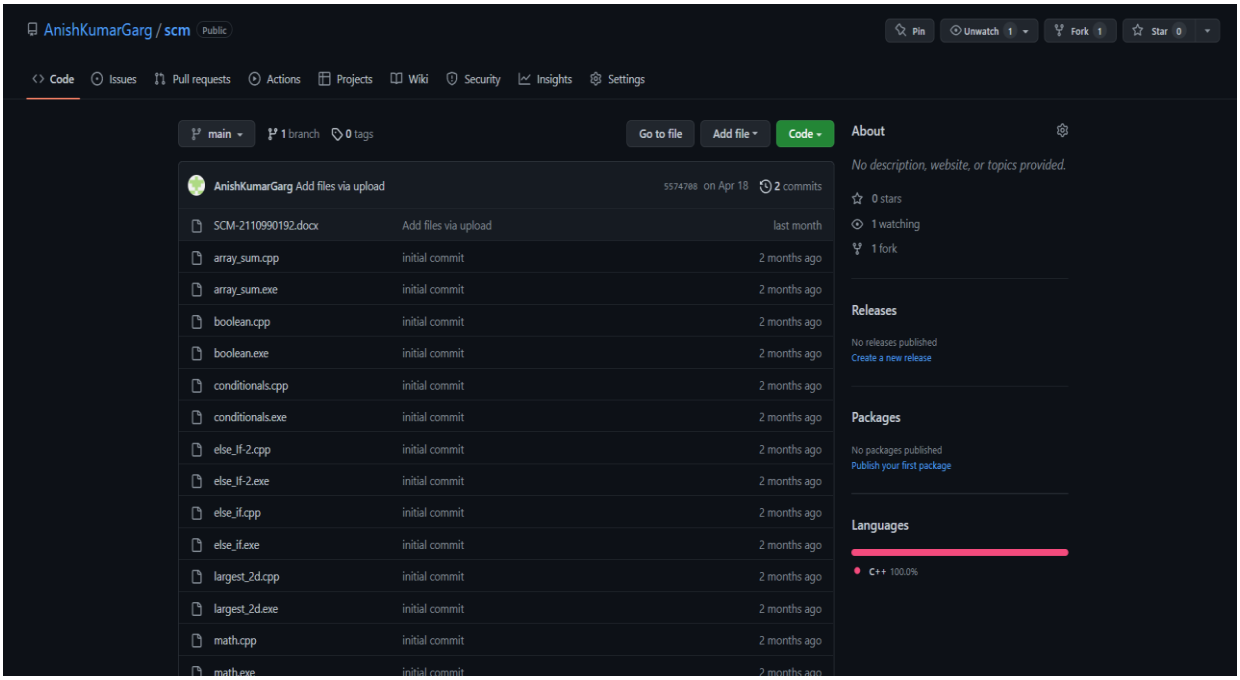
Experiment No. 07

Aim: Fork and Commit

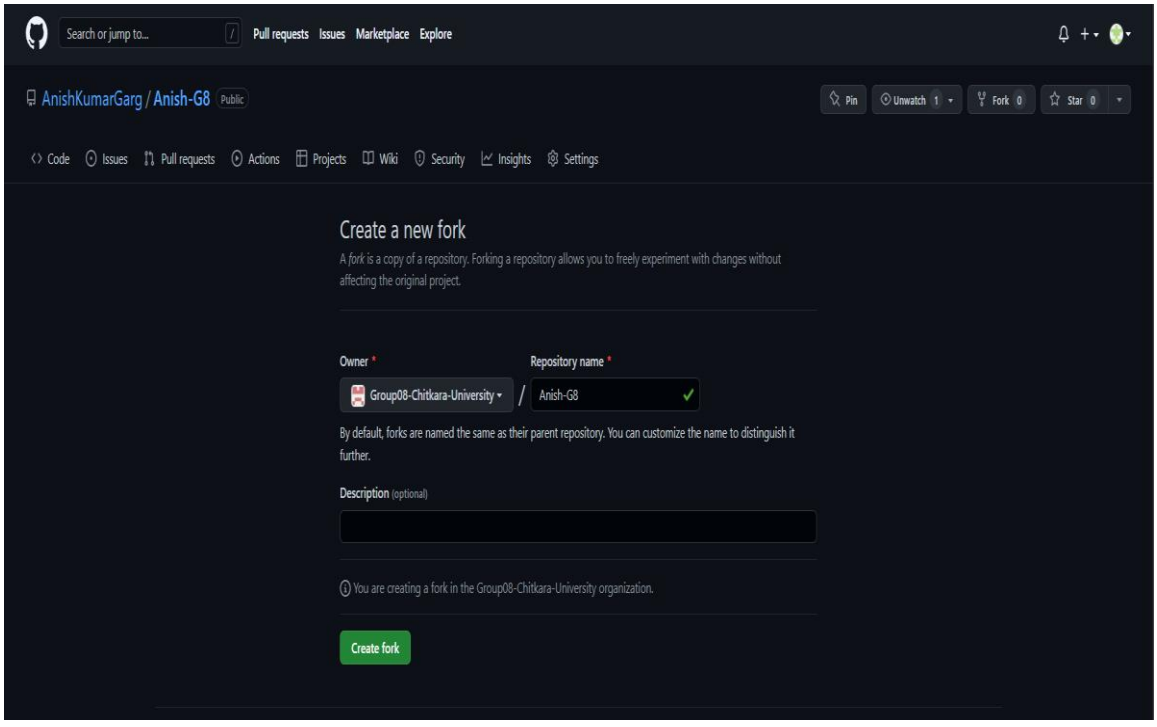
Theory:

Fork :- A Fork is copy of a repository . Forking a repository allows you to freely experiment with changes without affecting the original Project.

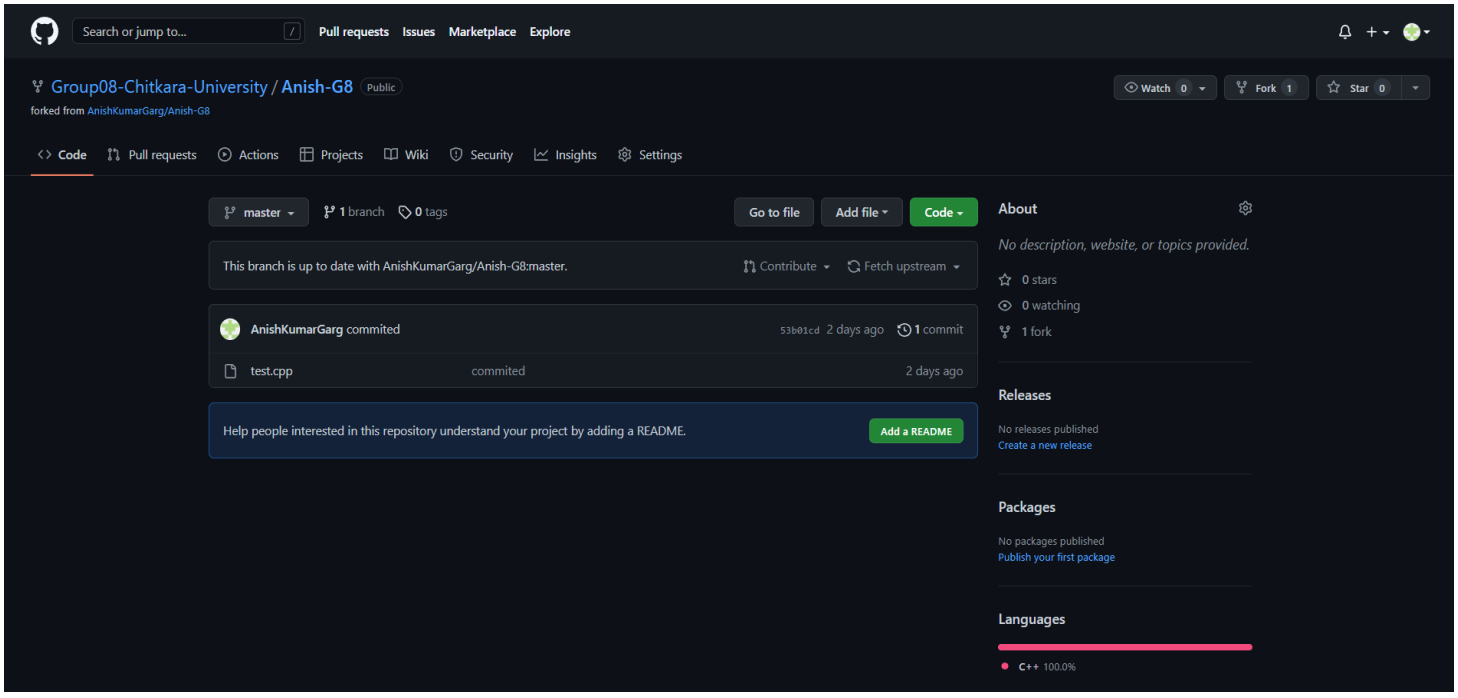
1. To fork a repository first thing you need to do is, sign in to your GitHub account and then you came to the repository you want to fork, so here for demo purpose am using **A2v20/SCM1** repository.



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 https://github.com/AnishKumarGarg/Anish-G8 on CLI.

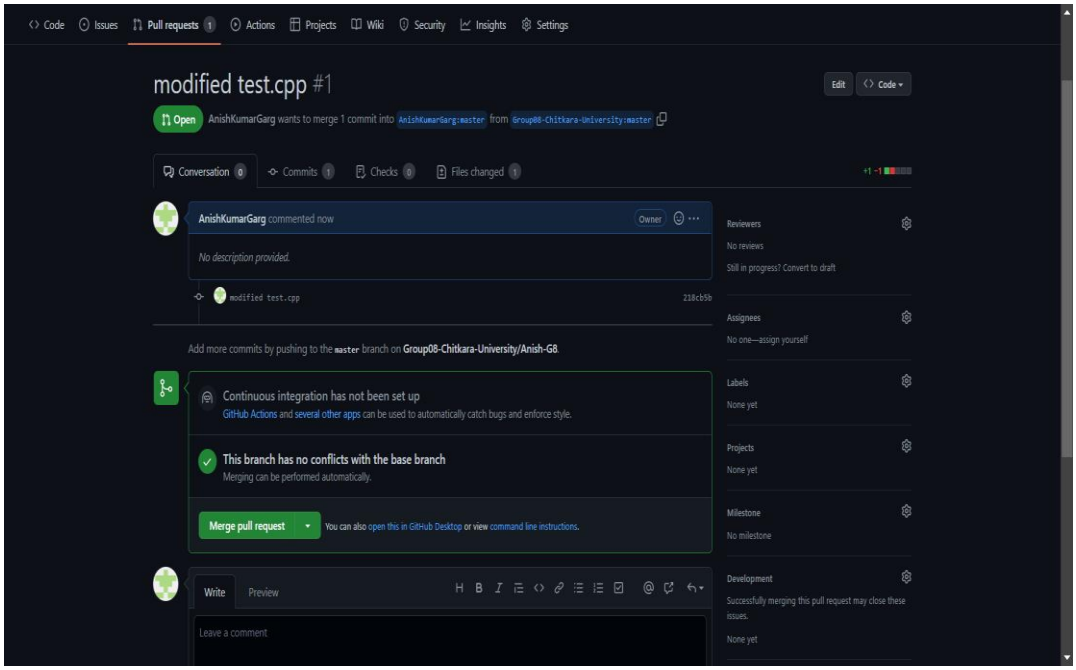
Git clone <url> --> This command is used to fetch the remote repo or to clone the repo.

```
lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/g8 (master)
$ git clone https://github.com/Group08-Chitkara-University/Anish-G8.git
Cloning into 'Anish-G8'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.

lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/g8 (master)
$ cd Anish-G8/
```

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

```
lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/g8 (master)
$ cd Anish-G8/

lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/g8/Anish-G8 (master)
$ vi test.cpp

lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/g8/Anish-G8 (master)
$ git commit -m "modified test.cpp"
On branch master
Your branch is up to date with 'origin/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:   test.cpp

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

lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/g8/Anish-G8 (master)
$ git add test.cpp

lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/g8/Anish-G8 (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   test.cpp


lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/g8/Anish-G8 (master)
$ git commit -m "modified test.cpp"
[master 218cb5b] modified test.cpp
 1 file changed, 1 insertion(+), 1 deletion(-)

lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/g8/Anish-G8 (master)
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 340 bytes | 340.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/Group08-Chitkara-University/Anish-G8.git
   53b01cd..218cb5b  master -> master
```

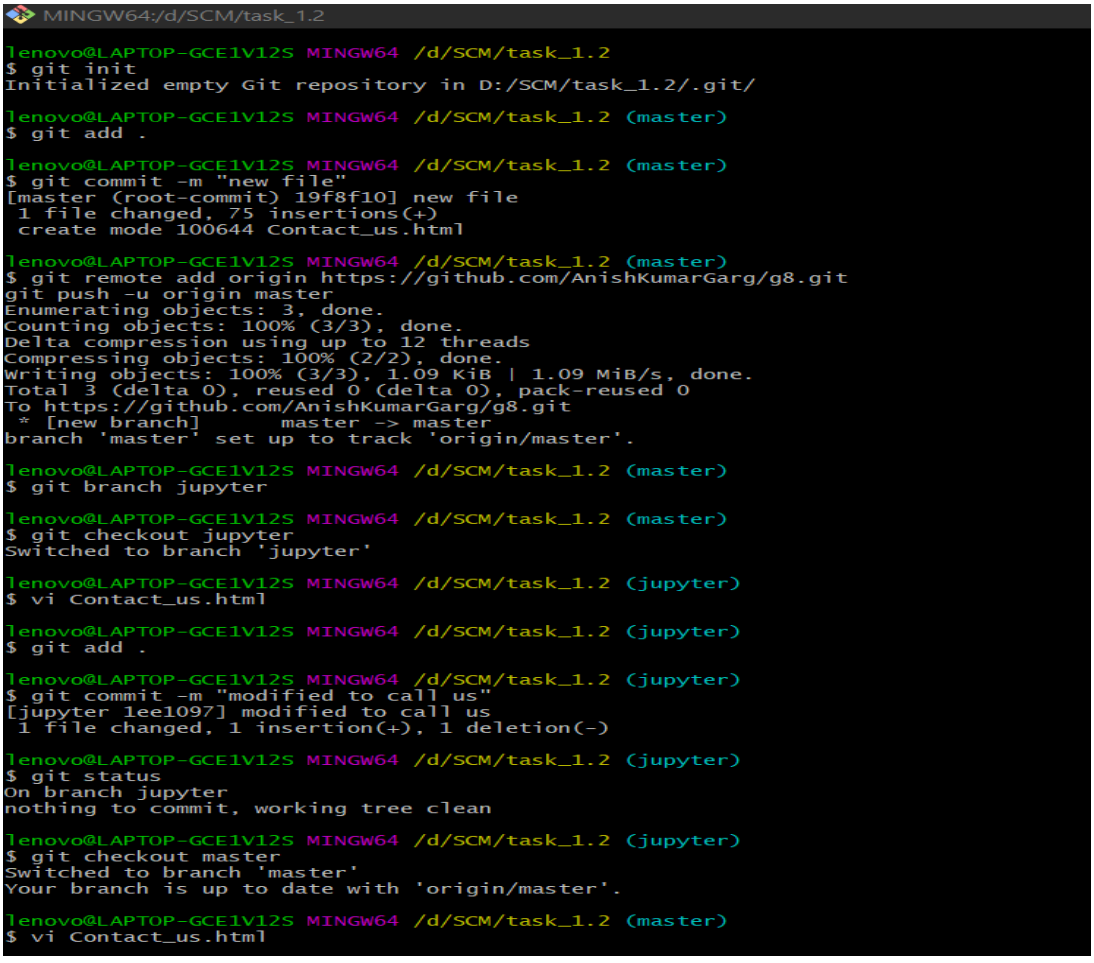
6. Now go to GitHub and accept the merge request.



Experiment No. 08

Aim: Merge and Resolve Conflicts Theory:

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



```
lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/task_1.2 (jupyter)
$ git add .

lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/task_1.2 (jupyter)
$ git commit -m "modified to call us"
[jupyter 1ee1097] modified to call us
 1 file changed, 1 insertion(+), 1 deletion(-)

lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/task_1.2 (jupyter)
$ git status
On branch jupyter
nothing to commit, working tree clean

lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/task_1.2 (jupyter)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/task_1.2 (master)
$ vi Contact_us.html

lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/task_1.2 (master)
$ git add .

lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/task_1.2 (master)
$ git commit -m "changed to remember us"
[master 9275396] changed to remember us
 1 file changed, 1 insertion(+), 1 deletion(-)

lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/task_1.2 (master)
$ git merge jupyter
Auto-merging Contact_us.html
CONFLICT (content): Merge conflict in Contact_us.html
Automatic merge failed; fix conflicts and then commit the result.

lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/task_1.2 (master|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 gvimdiff diffuse diffmerge ecmerge p4merge araxis bc codecompare smerge emerge vimdiff nvimdiff
Merging:
Contact_us.html

Normal merge conflict for 'Contact_us.html':
  {local}: modified file
  {remote}: modified file
Hit return to start merge resolution tool (vimdiff):
4 files to edit

lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/task_1.2 (master|MERGING)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
```

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

```
lenovo@LAPTOP-GCE1V12S MINGW64 /d/SCM/task_1.2 (master)
$ git merge jupyter
Auto-merging Contact_us.html
CONFLICT (content): Merge conflict in Contact_us.html
Automatic merge failed; fix conflicts and then commit the result.
```

3. Use Command “git mergetool” to solve the conflict.
   git -mergetool – Run merge conflict resolution tools to resolve merge conflicts.

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

Experiment No. 09

Aim: Reset and Revert

Theory:

Git-revert – Revert some existing commits.

While Working with Git in certain situations we want to undo changes in the working area or index area, sometimes remove commits locally or remotely and we need to reverse those changes. There are 3 different ways in which we can undo the changes in our repository, these are ***git reset***, ***git checkout***, and ***git revert***. git checkout and git reset in fact can be used to manipulate commits or individual files. These commands can be confusing so it's important to find out the difference between them and to know which command should be used at a particular point of time.

Let's make a sample git repository with a file ***trial.txt*** and "***Hello Geeks***" written inside it.

```
MINGW64:/c/Users/DELL/OneDrive/Desktop/revert

DELL@Akshay MINGW64 ~/OneDrive/Desktop/revert (master)
$ git init
Initialized empty Git repository in C:/Users/DELL/OneDrive/Desktop/revert/.git/

DELL@Akshay MINGW64 ~/OneDrive/Desktop/revert (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        trial.txt

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

DELL@Akshay MINGW64 ~/OneDrive/Desktop/revert (master)
$ git add trial.txt

DELL@Akshay MINGW64 ~/OneDrive/Desktop/revert (master)
$ git commit -m "added new file"
[master (root-commit) 2e35bf7] added new file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 trial.txt

DELL@Akshay MINGW64 ~/OneDrive/Desktop/revert (master)
$ git log
commit 2e35bf73543fa12319f8c0d007fd9f2503ab0516 (HEAD -> master)
Author: Akshaykat2003 <akshaykatoch38@gmail.com>
Date:   Wed May 18 09:04:43 2022 +0530

    added new file

DELL@Akshay MINGW64 ~/OneDrive/Desktop/revert (master)
$ git status
On branch master
nothing to commit, working tree clean

DELL@Akshay MINGW64 ~/OneDrive/Desktop/revert (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:   trial.txt

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

We can see that we have a single commit is done and the text document that has been committed with added new files in it. Now let's add some more text to our text document. Let's add another line ***Hello World.*** Doing this change, our file now needs to be added to

the staging area for getting the commit done. This updates are currently in the working area and to see them we will see those using ***git status***.

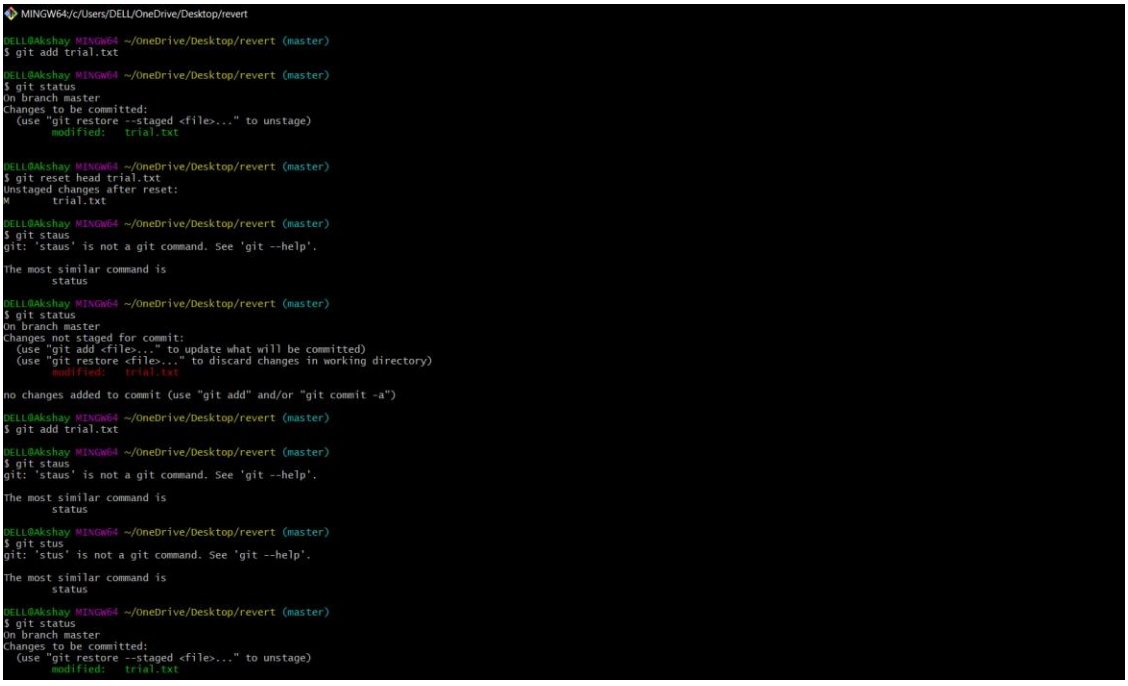
Now we have a change ***Hello World*** which is untracked in our working repository and we need to discard this change. So, the command that we should use here is -

1. git checkout

git checkout is used to discard the changes in the working repository. git checkout

<filename>

When we write git checkout command and see the status of our git repository and also the text document we can see that our changes are being discarded from the working directory and we are again back to the test document that we had before. Now, what if we want to unstage a file. We stage our files before committing them and at a certain point, we might want to unstage a file. Let's add ***Hello World*** again to our text document and stage them using the ***git add*** command.



We want to unstage a file and the command that we would be using to unstage our file is -

2. git reset

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

git reset HEAD <filename>

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

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

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

git reset HEAD~1 Points to

be noted -

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

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

- ***git reset --soft HEAD~1*** - This command will remove the commit but would not unstage a file. Our changes still would be in the staging area.

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

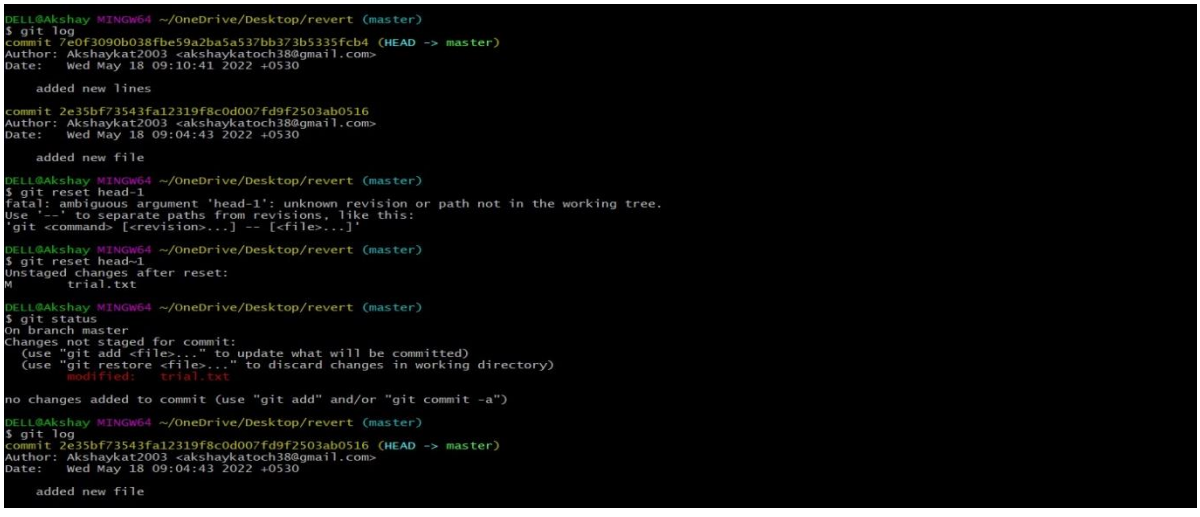
Points to keep in mind while using git reset command -

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

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

3. git revert

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



Now let's push our changes to the remote repository.

Now we want to delete the commit that we just added to the remote repository. We could have used the git reset command but that would have deleted the commit just from the local repository and not the remote repository. If we do this then we would get conflict that the remote commit is not present locally. So, we do not use git reset here. The best we can use here is git revert.

git revert <commit id of the commit that needs to be removed> Points to keep in

mind -

- Using git revert we can undo any commit , not like git reset where we could just remove "n" recent commits.

Now let's first understand what git revert does, git revert removes the commit that we have done but adds one more commit which tells us that the revert has been done. Let's look at the example –

Note: If you see the lines as we got after the git revert command, just visit your default text editor for git and commit the message from there, or it could directly take to you your default editor. We want a message here because when using git revert, it does not delete the commit instead makes a new commit that contains the removed changes from the commit.

We can see that the new commit is being added. However since this commit is in local repository so we need to do ***git push*** so that our remote repository also notices that the change has been done.

And as we can see we have a new commit in our remote repository and ***Hello World*** which we added in our 2nd commit is being removed from the local as well as the remote repository. Let's summarize the points that we saw above -

Difference Table

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