Aadarsh Kumar

Subject Name: **Source Code Management**

Subject Code: **CS181**

Cluster: **Beta**

Department: **CSE**


**Submitted by:**

Aadarsh Kumar

2110990001

G01

**Submitted to:**

Dr. Monit Kapoor

Professor and Dean | Beta Cluster

Department of CSE

Chitkara University, Punjab Campus

# INDEX

# 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

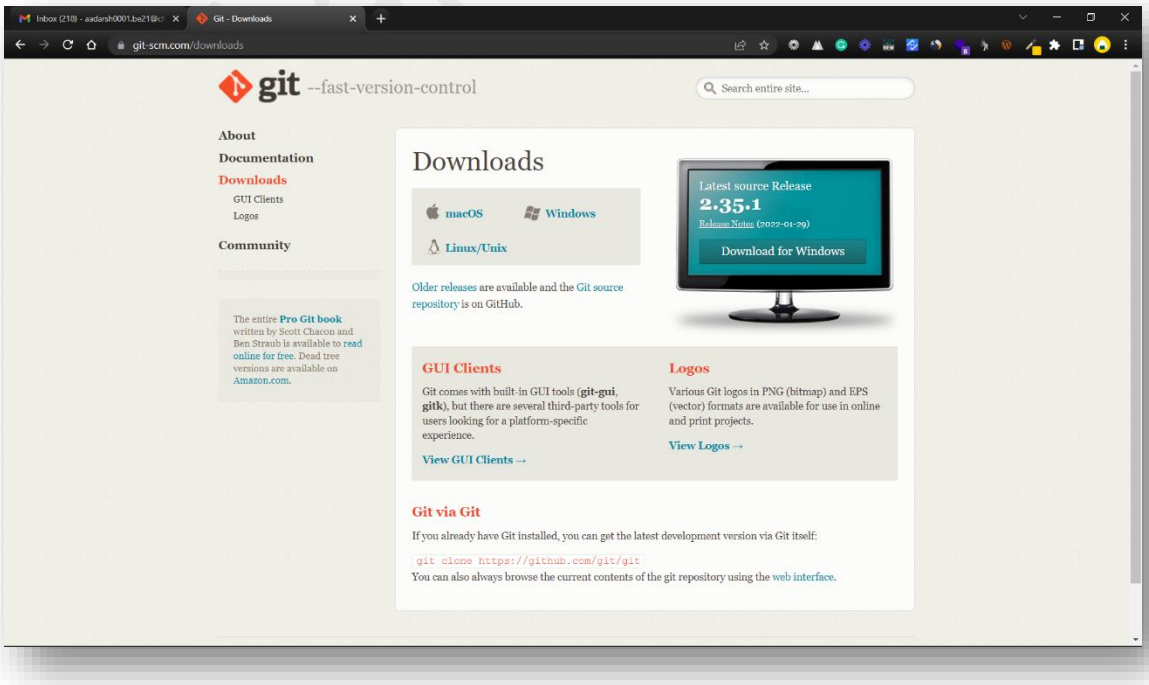
I. **Local Version Control System:** Local Version Control System is located in your local machine. If the local machine crashes, it would not be possible to retrieve the files, and all the information will be lost. If anything happens to a single version, all the versions made after that will be lost.
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.

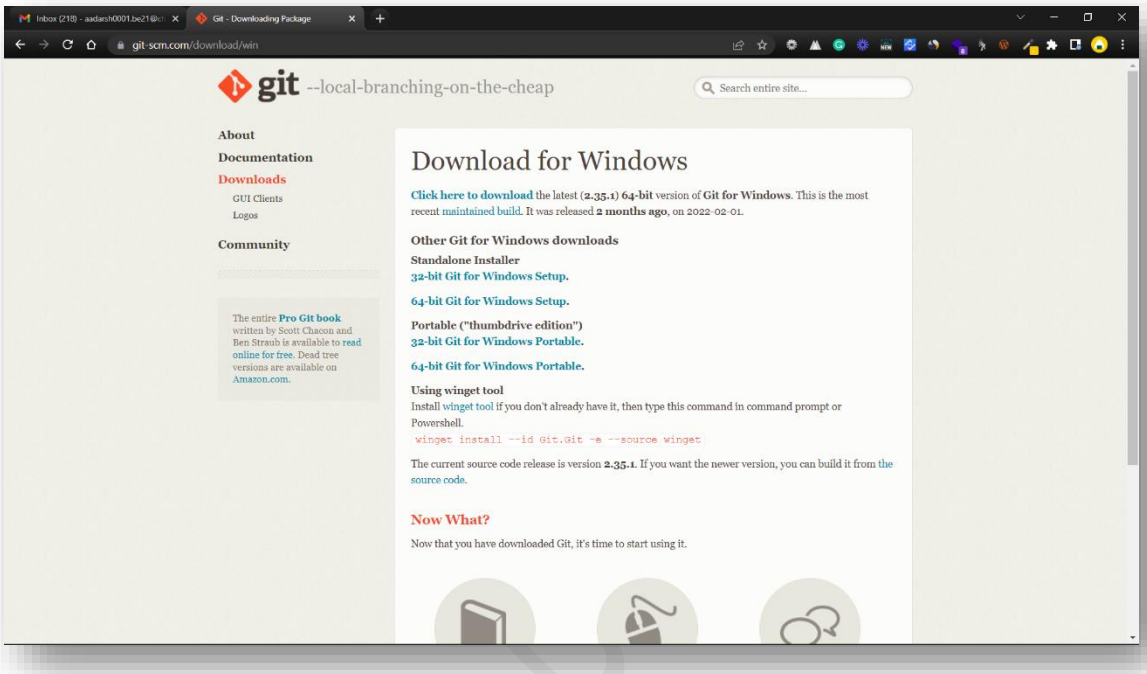
## Experiment No. 01

### *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.

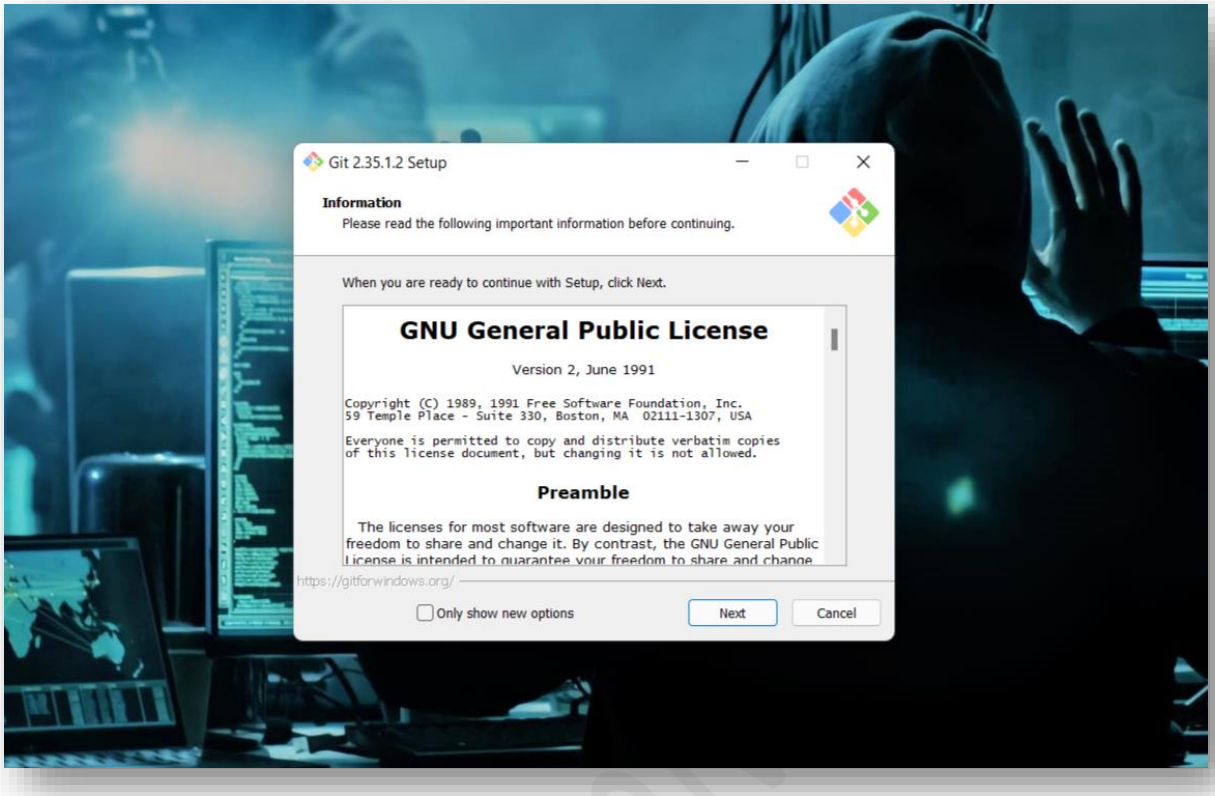



- Select the CPU for your system now. (Most of the system now runs on 64-bit processors.) Your download will begin when you pick a processor.

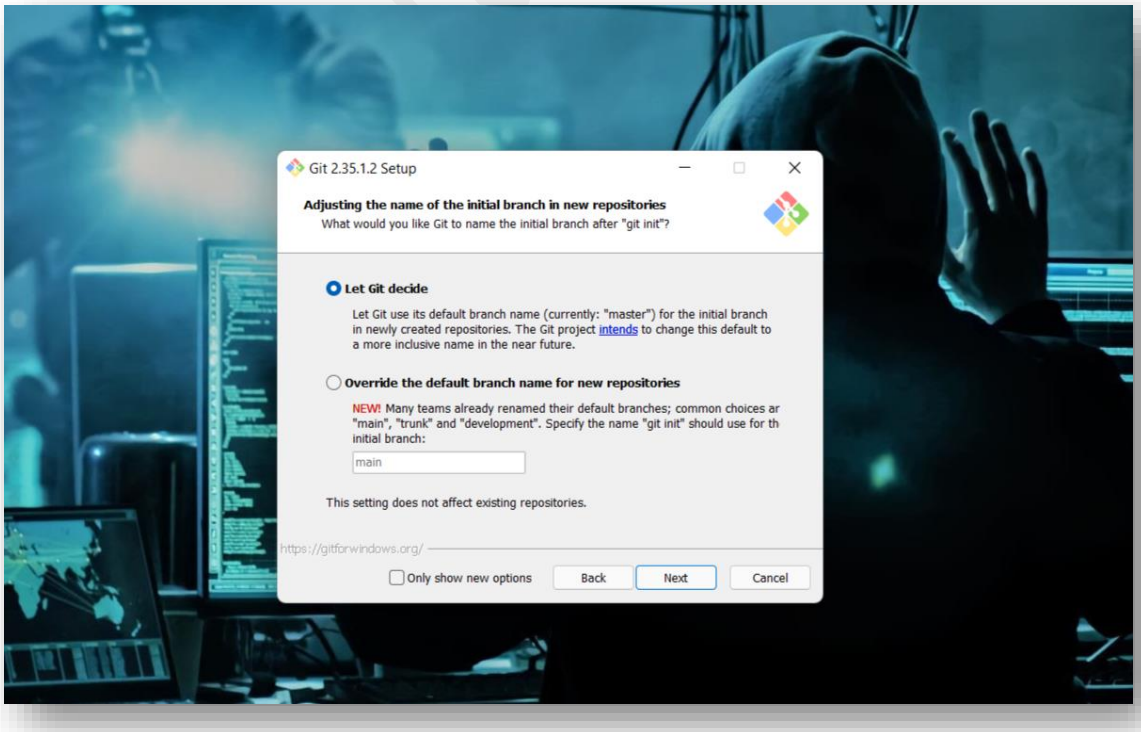

- You must now open the Git folder.

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

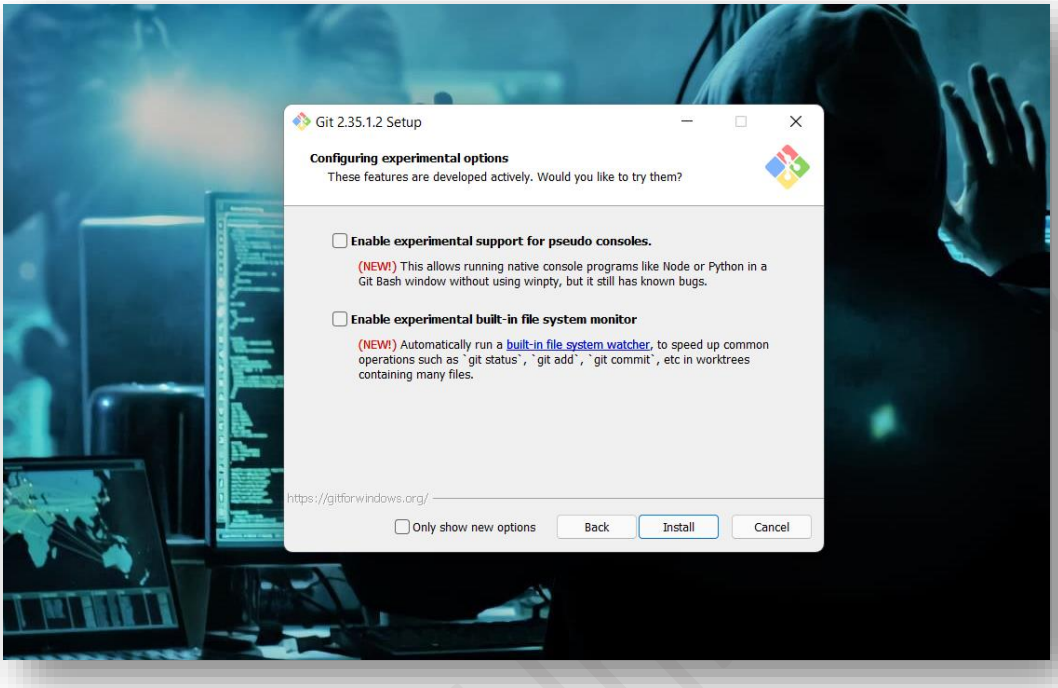
- YES should be selected.



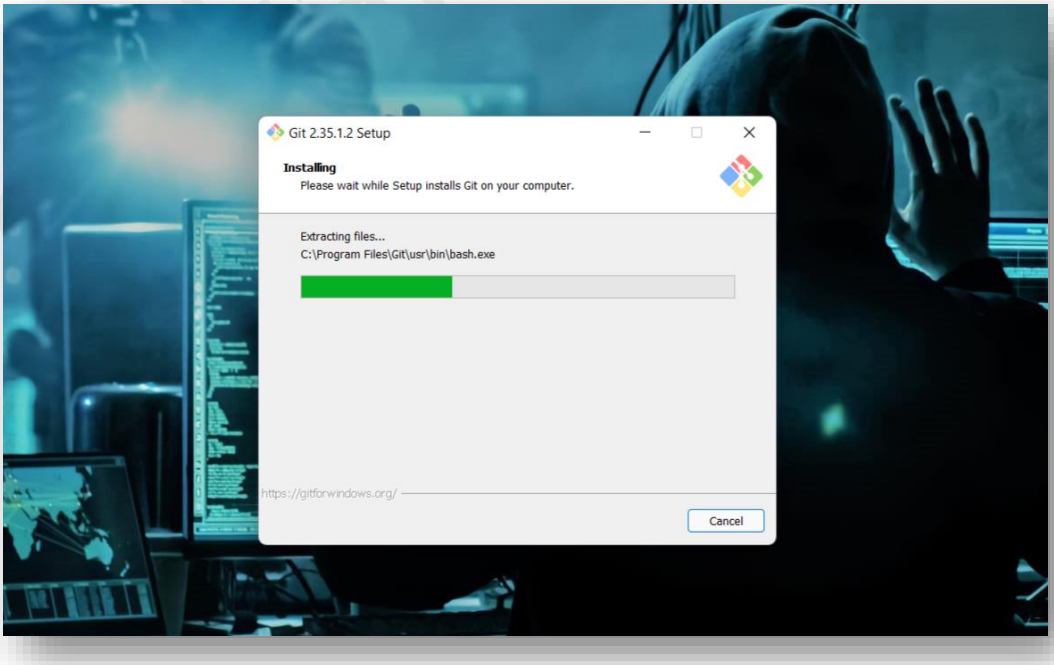

✧ Click on Next



✧ Continue clicking on next few times more




✧ Now select the Install option.



✧ Click on Finish after the installation is finished.

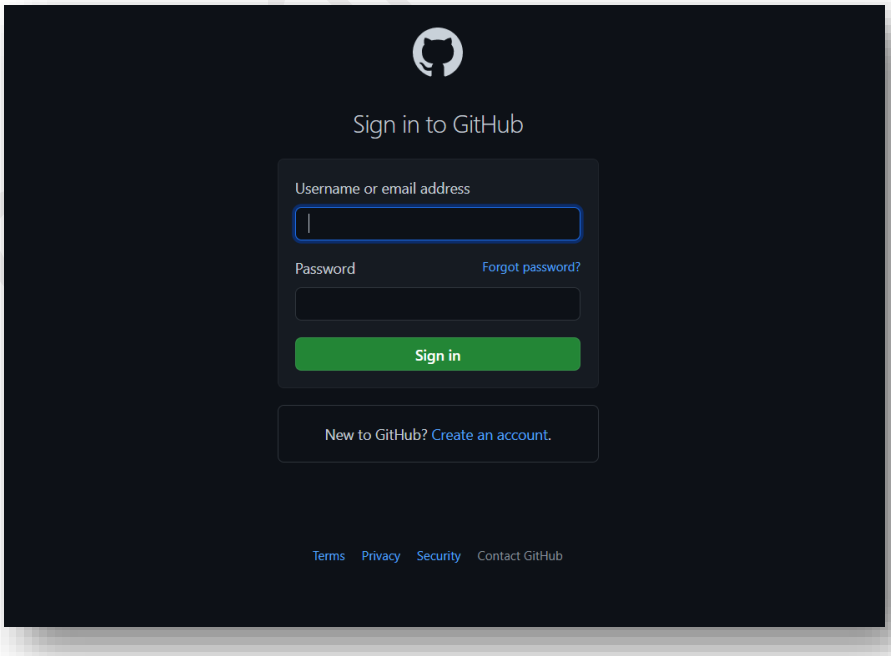
The installation of the git is finished and now we have to setup git client and GitHub account.

## Experiment No. 02

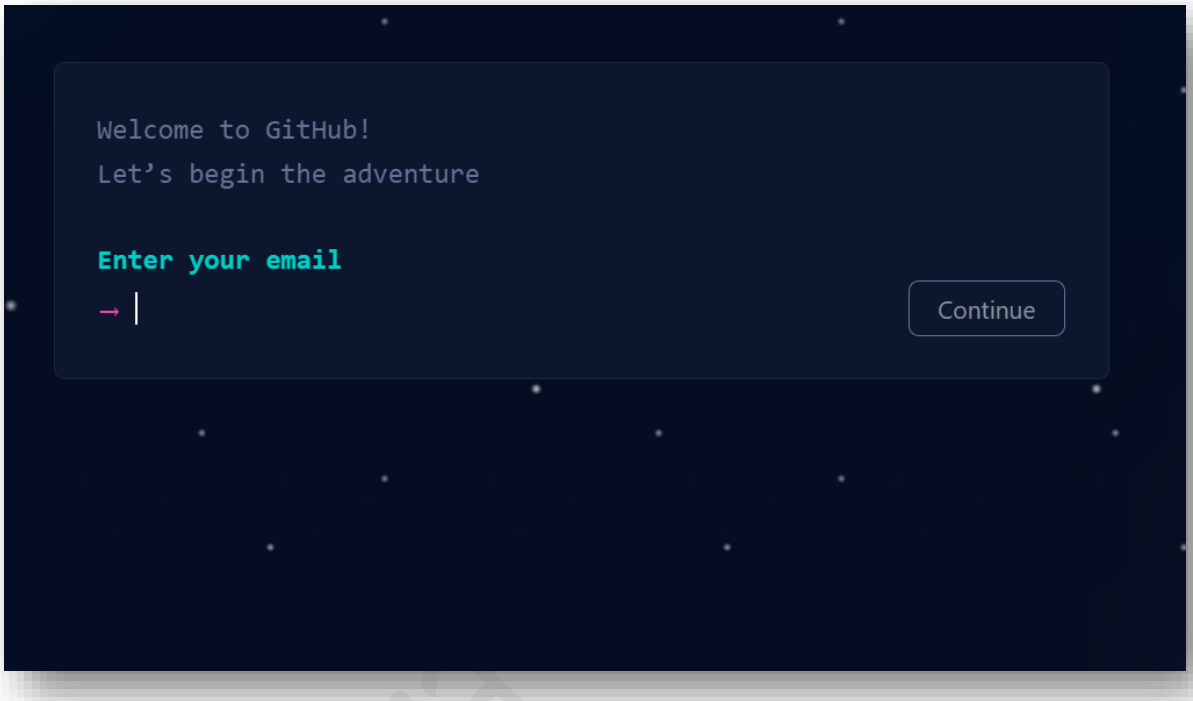
### *Aim: Setting up GitHub Account*

✧ 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.



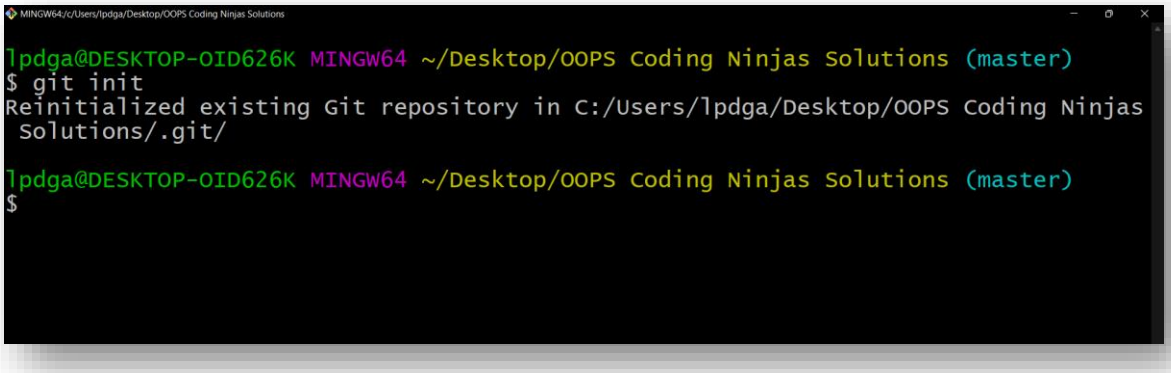

✧ Now Click on Create Account.

✧ 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 “git init” and it creates a folder .git.




✧ 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”**

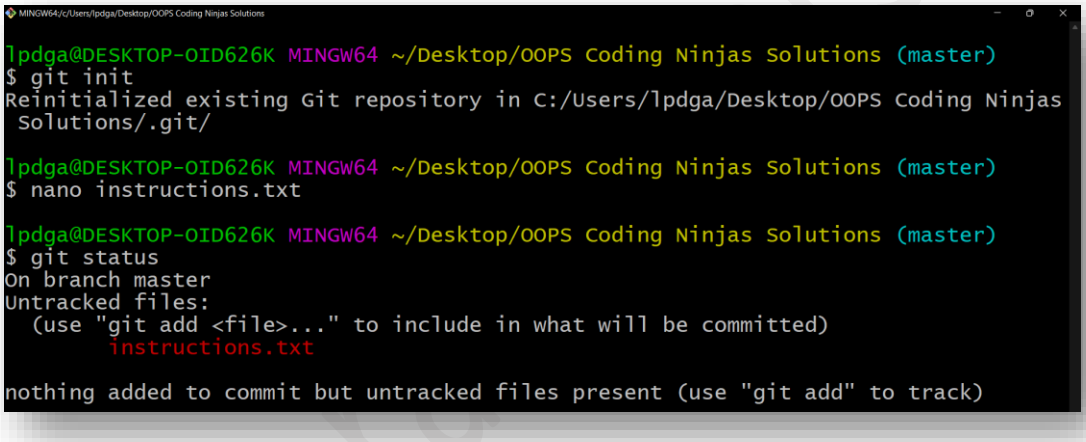
**“git config --global user.email”**

## *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 instructions.txt Now type git status:



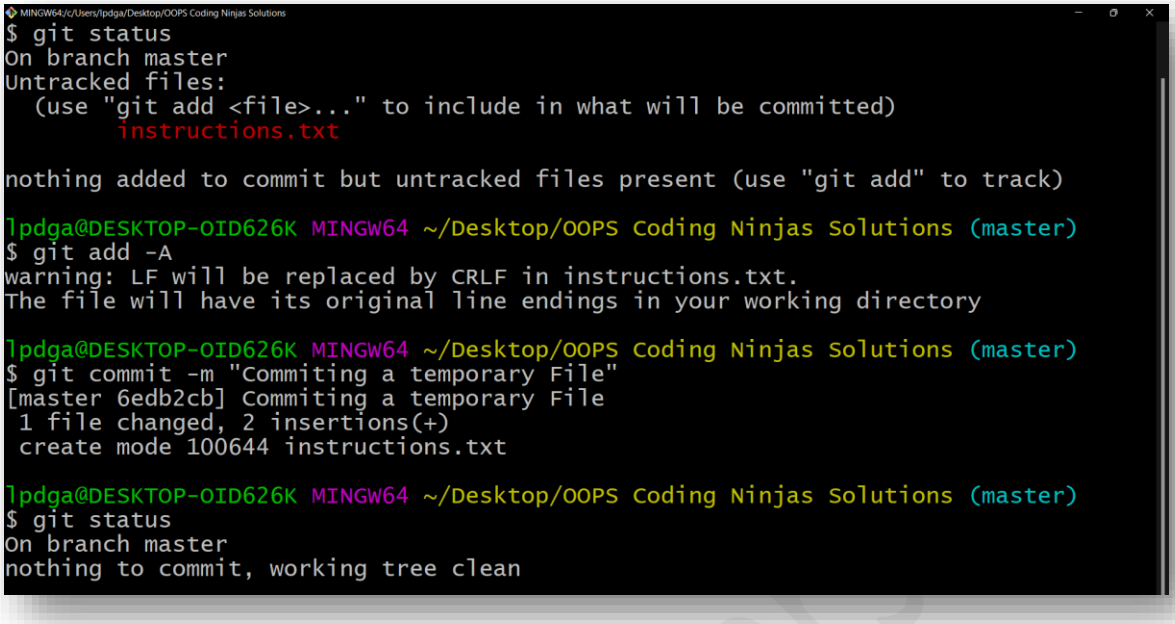

You can see that instructions.txt is in red colour that means it is an untracked file.

Now firstly add the file in staging area and then commit the file.

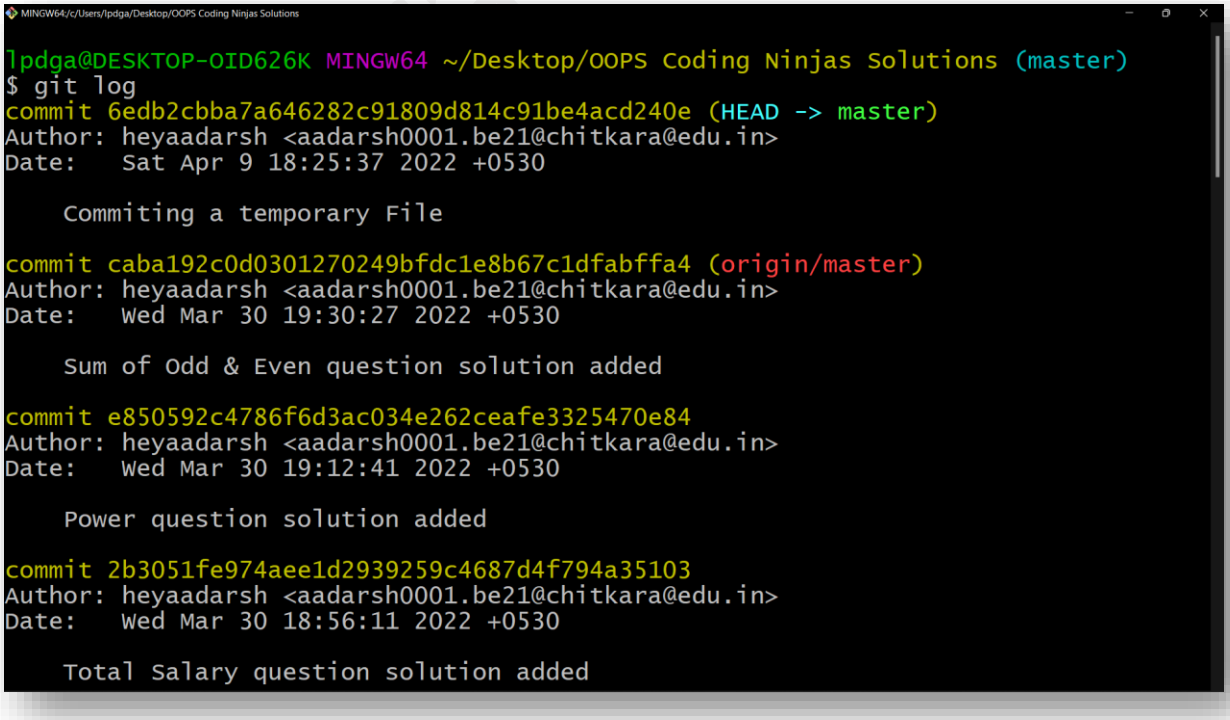
For this, use command →

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

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



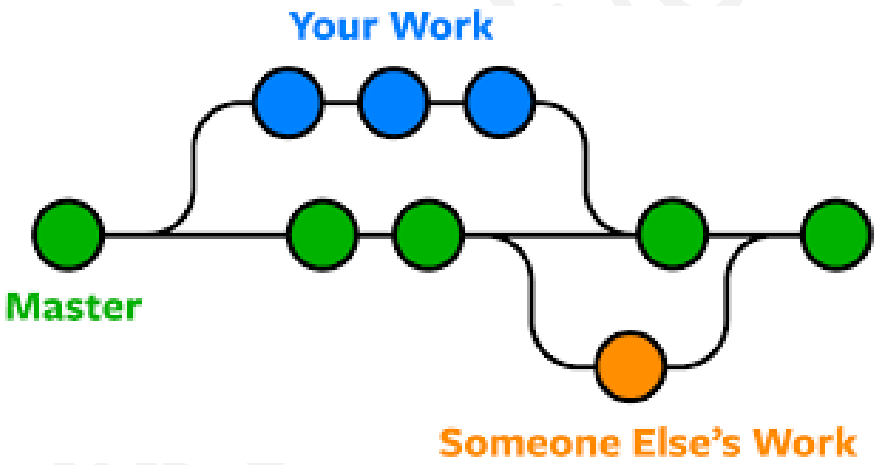

✧ **git log:** *The git log command displays a record of the commits in a Git repository. By default, the git log command displays a commit hash, the commit message, and other commit metadata.*



**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.

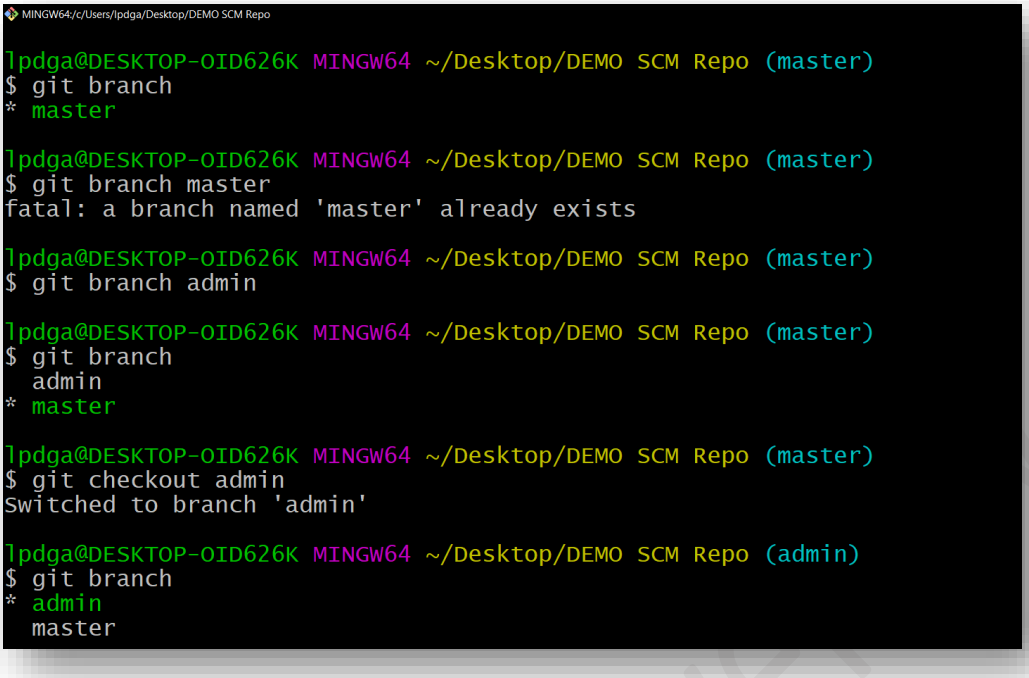



Let us see the command of it:

Firstly, add a new branch, let us suppose the branch name is activity1.

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**]**




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

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

```
lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/DEMO SCM Repo (master)
$ nano nothing.txt

lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/DEMO SCM Repo (master)
$ git checkout admin
Switched to branch 'admin'

lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/DEMO SCM Repo (admin)
$ git add -A
warning: LF will be replaced by CRLF in nothing.txt.
The file will have its original line endings in your working directory

lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/DEMO SCM Repo (admin)
$ git commit -m "Commiting Nothing.txt!"
[admin b739244] Commiting Nothing.txt!
 1 file changed, 1 insertion(+)
 create mode 100644 nothing.txt

lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/DEMO SCM Repo (admin)
$ ls
hello.txt  nothing.txt
```

If we switched to master branch, 'nothing.txt' file is not there. But the file is in admin branch.

```
MINGW64:/c/Users/lpdga/Desktop/DEMO SCM Repo

lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/DEMO SCM Repo (admin)
$ git checkout master
Switched to branch 'master'

lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/DEMO SCM Repo (master)
$ ls
hello.txt

lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/DEMO SCM Repo (master)
$
```

- 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 branch-name [**use to merge branch**]**

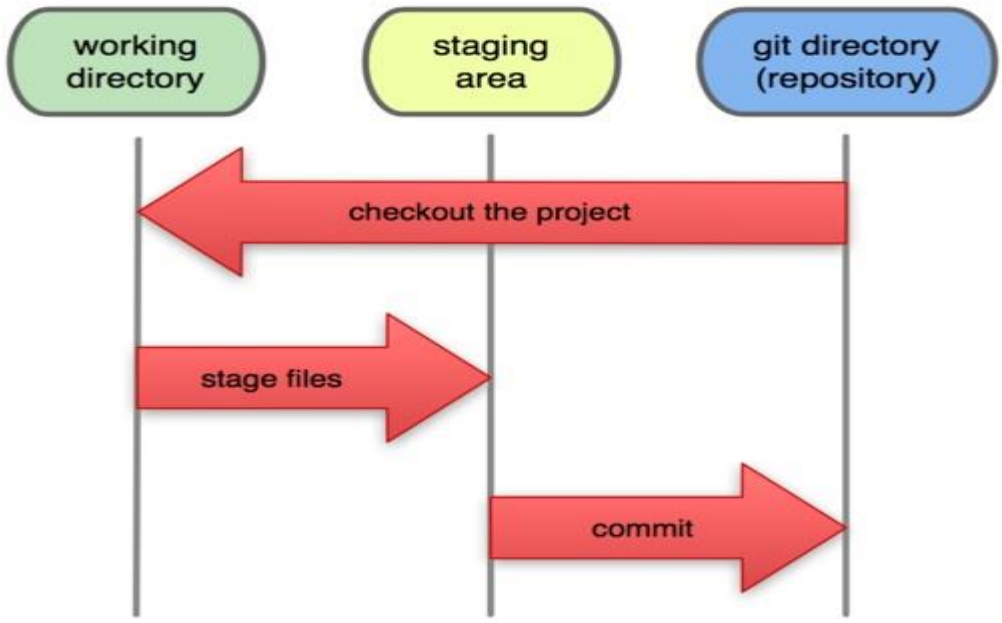
```
MINGW64:/c/Users/lpdga/Desktop/DEMO SCM Repo

lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/DEMO SCM Repo (master)
$ git merge admin
Updating 7cf789d..b739244
Fast-forward
 nothing.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 nothing.txt

lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/DEMO SCM Repo (master)
$ ls
hello.txt  nothing.txt

lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/DEMO SCM Repo (master)
$
```

# 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 File-Name" 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.

**Experiment No. 1.2**

## *Aim: Add collaborators on GitHub Repository*

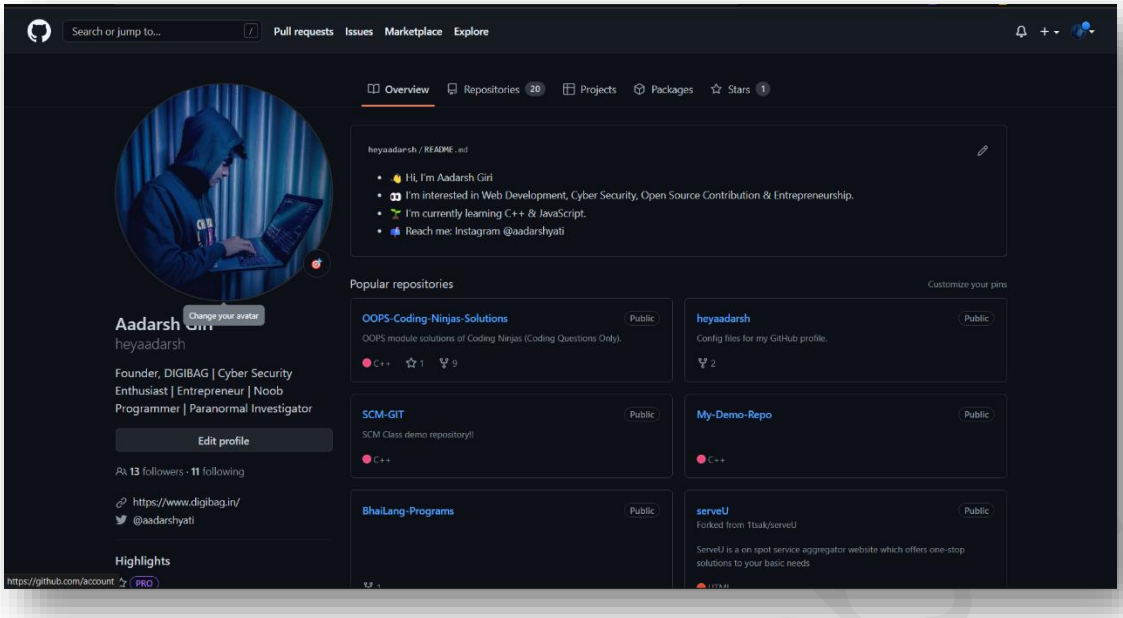
### What is GitHub Collaboration?

GitHub collaboration is a space where you can invite another developer to your repository and work together at an organization level, splitting the task and 2nd person will have the rights to add or merge files to the main repository without further permission.

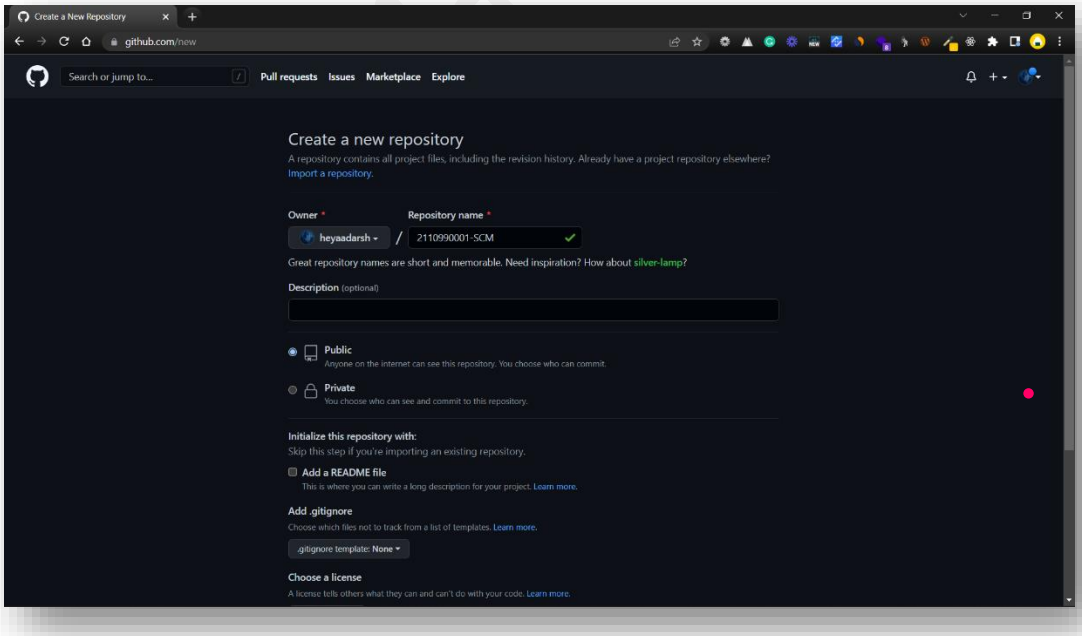
### Let's invite a Collaborator.

In this post, we will create a new repository and invite a collaborator to our repository by sending an invitation. A detailed procedure on how to invite a collaborator to your GitHub repository is mentioned below.

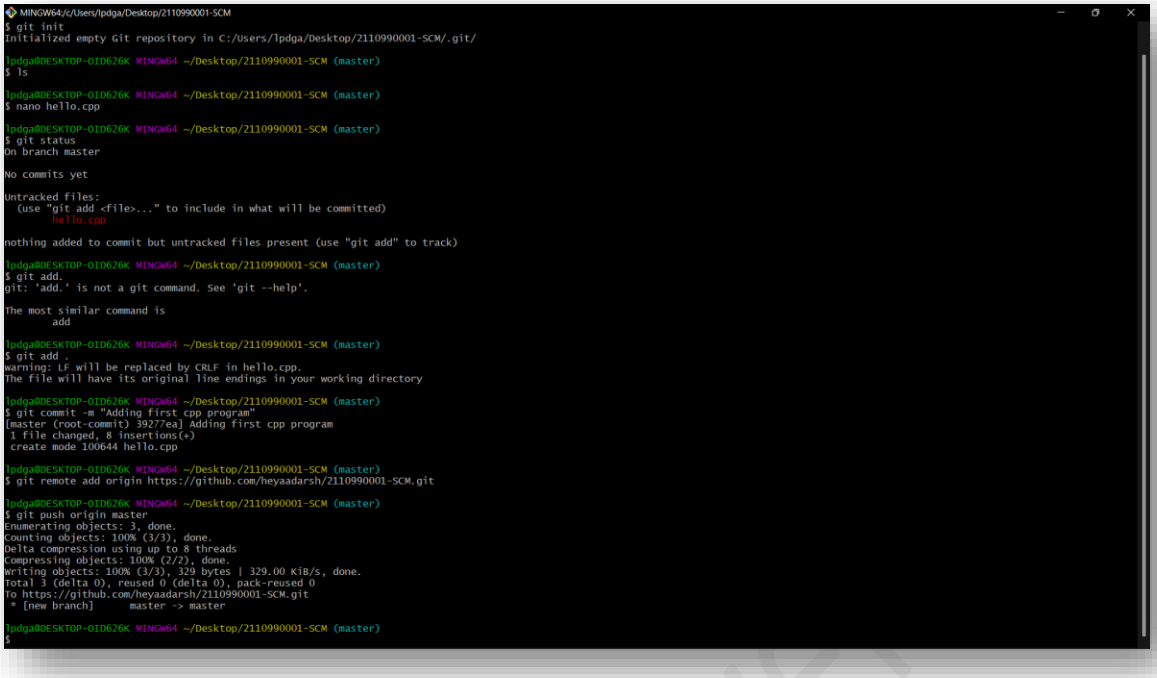
**Step 1:** Went to Git hub and created new Repository, click on the + sign on top right side and drop down will appear click on New Repository.




**Step 2: Specify the Name of the Project,** make It public or primate, check on the readme file. Then click on Create repository. You can also see by default Brach name is main If you want you can change it. **(optional)**

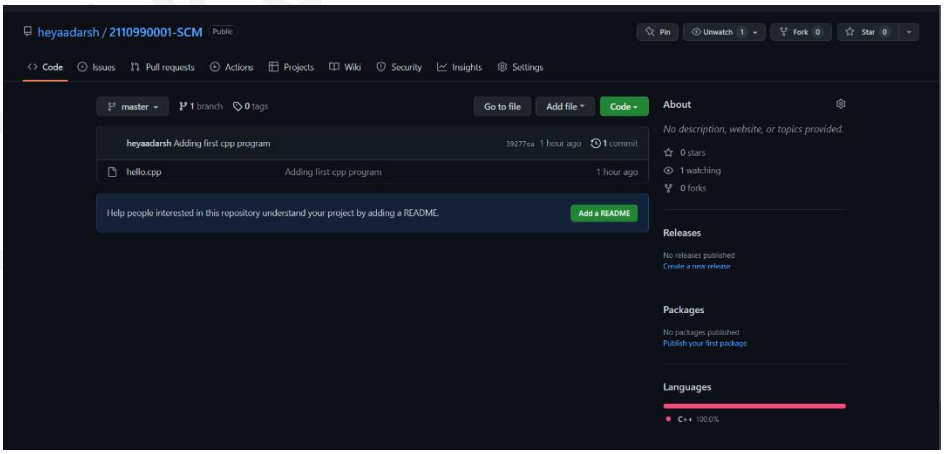



**Step 3:** Now Copy the HTTP link of your repo and paste it on your 'Git CLI', and merge the local repo in remote repo.



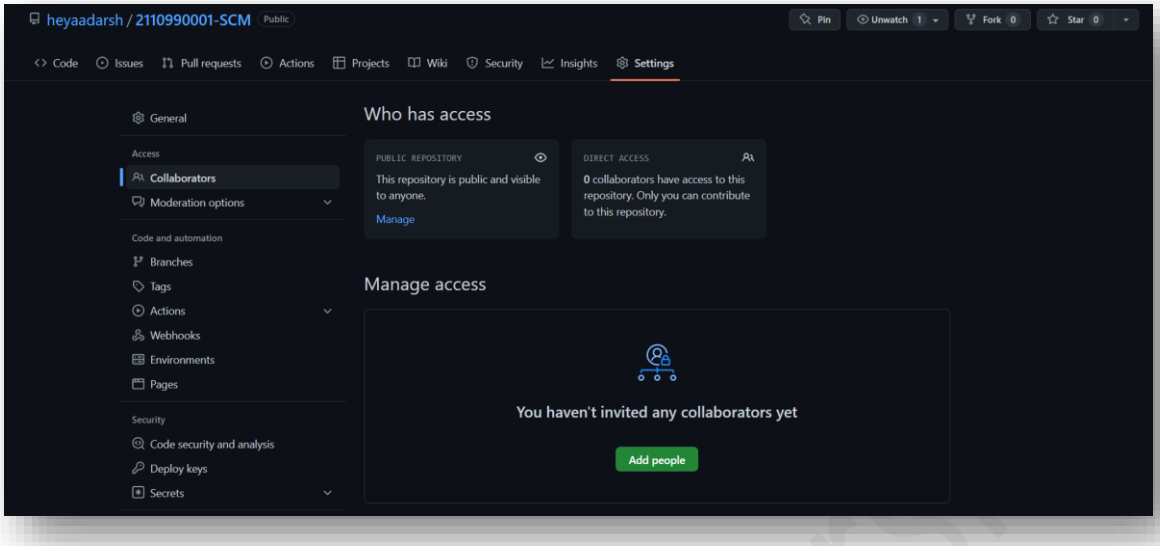

**Step 4:** Now you can see the file created on that name with a default readme file. You can add content to the readme file. Now the next step is to invite the collaborators to the repository. currently, you have your repository ready to collaborate.

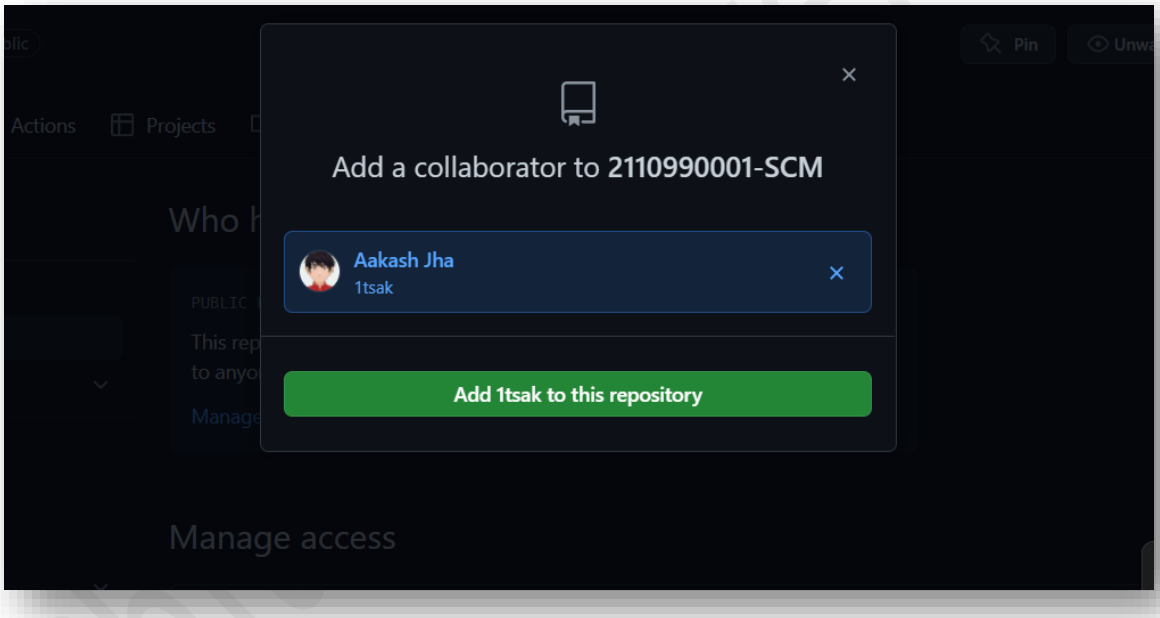
**Step 5:** Click on **settings,** you should be the one who created the repo to do this work, because only the author can send the invite to others.



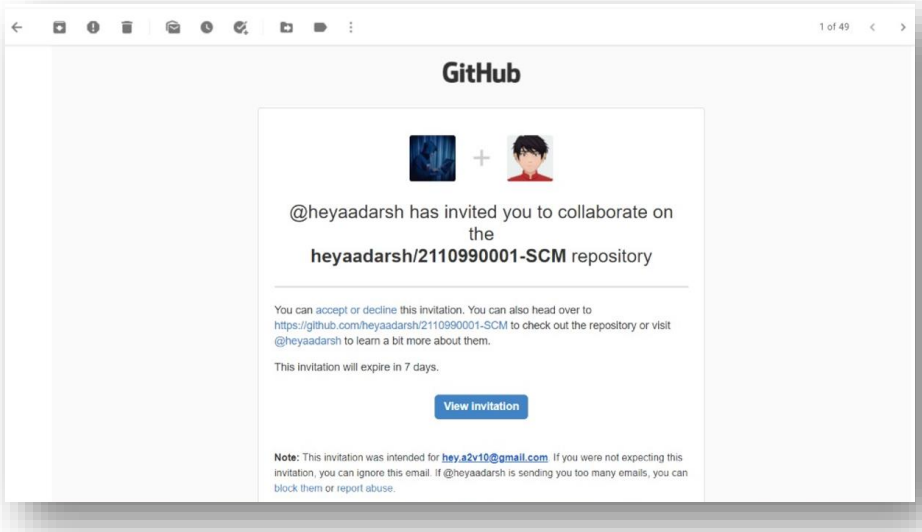

**Step 6:** Click on the invited collaborator and enter the email ID of people you want to add to this repo. Email will go to them to accept the request.

**Once it accepted, they can push the changes to main branch.**

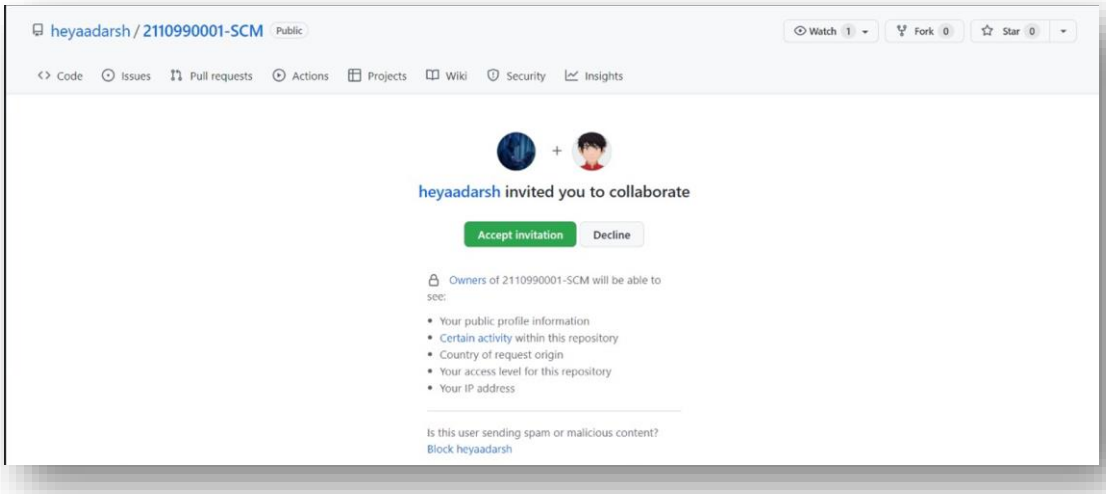



**Step 7:** The person will be getting an email invitation like this for GitHub collaboration, click on the Email View and accept invitation.

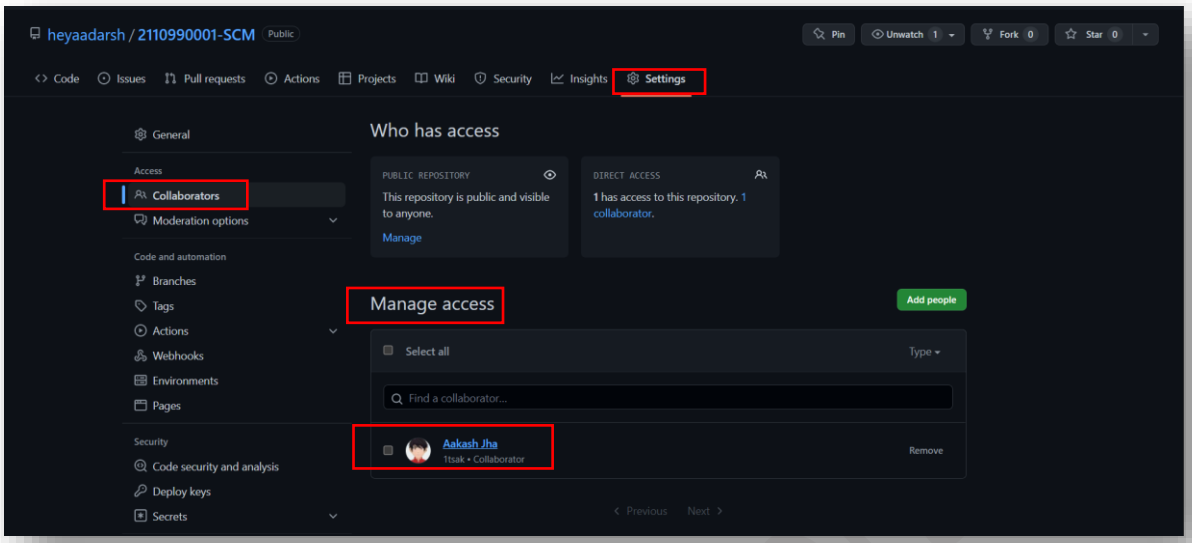



**Step 8:** You will be redirected to GitHub Windows there click on Accept Invitation.




## How to see Existing GitHub Repository collaborator

Go to your respective Repository and click on Manage access under Security tab, sometimes it will ask you to authenticate your account by entering the password. You can see list of people under the Manage access.




## Fatal: unable to access | Error: 403

What will happen if the person not added you as a collaborator and you tried to clone the file and push the repository to the main file? It will lead to the error permission denied. So it's important to give enough permission.

```
fatal: unable to access 'URL': The requested URL returned error: 403
```

## FINAL VERDICT:

In conclusion, I hope you enjoyed reading this article on "**How to add Collaborators into GitHub Repository?**". Again, inviting a collaborator to GitHub is good to approach, but making the changes and pushing to the main repository is not a good approach, instead, we create individual branches and push the work to branch and later merge to the main repository.

### How to delete a collaborator from GitHub Repository?

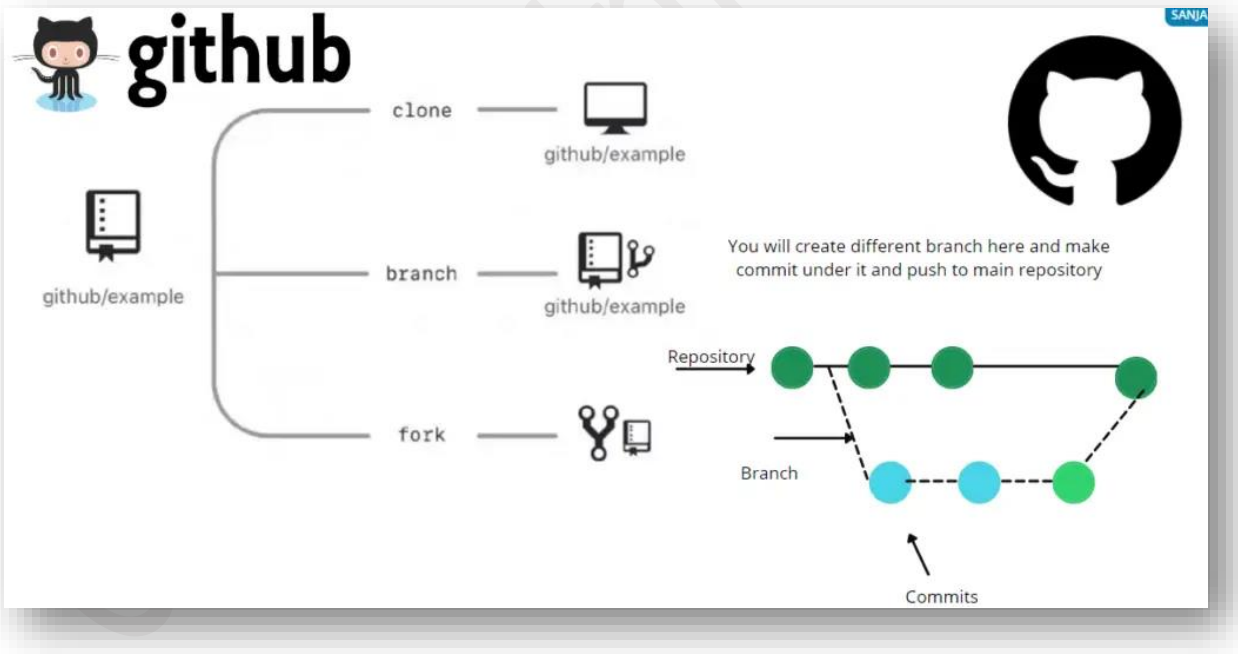
Go to the Repository, Click on Manage access under settings tab, you can see a list of collaborators under Mange access, on right side of each listing there is a delete icon.

## *Aim: Fork and Commit*

### What is Forking in GitHub?

Forking an Existing GitHub repository helps to create a **copy of the main file in production**, hence provide the user to experiment with the copied repository on their GitHub branch. So, any changes made to the forked file won't affect the main file in production.

The above image shows how the fork is working once you added the new feature to the copied branch and this committee will be happening in the branch, once you tested this out, you have an option **to pull request and merge the change with the main branch**. Then the owner of the particular repository will look at the changes and do the code review and accept the changes.

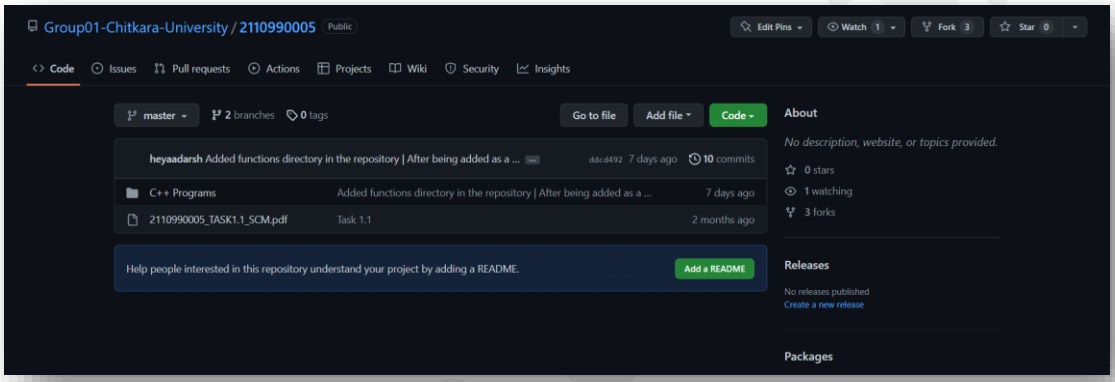


# Forking a repository from GitHub

You might a project to propose changes to the upstream, or original, repository. In this case, it's good practice to regularly sync your fork with the upstream repository. To do this, you'll need to use Git on the command line. You can practice setting the upstream using the same repository you just forked.

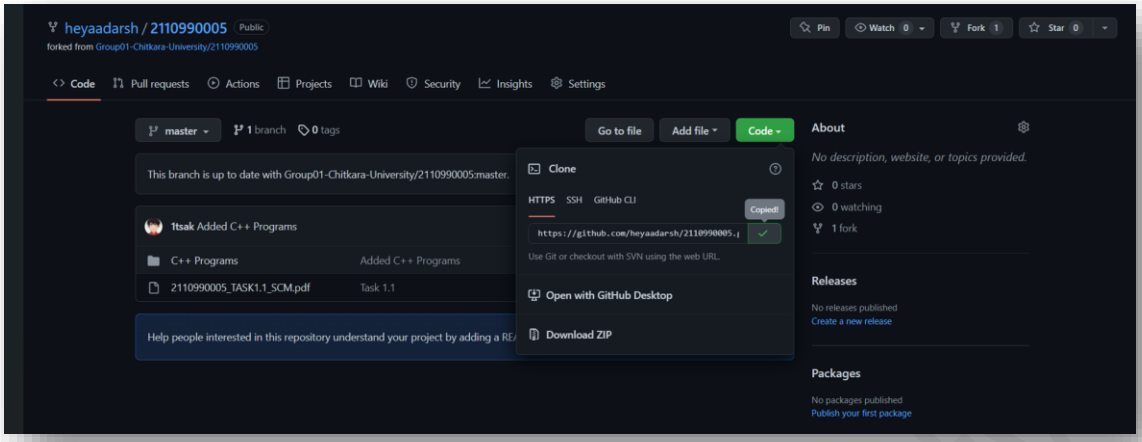
**Step 1:**
On GitHub.com, navigate to the Group01-Chitkara-University/2110990005 repository.

**Step 2:** Go In the top-right corner of the page, click fork.




**Step 3:** Now go to your profile and see the latest forked repository in your repository tab.

**Step 4:** The below screen shows the GitHub Profile that I have forked now. Then the next process is to clone the project for this click on the code button as highlighted below.




To clone the repository using HTTPS, under "Clone with HTTPS", click. To clone the repository using an SSH key, including a certificate issued by your organization's SSH certificate authority, click **Use SSH**, then click. To clone a repository using GitHub CLI, click **Use GitHub CLI**, then click.

1. Open Git Bash.
2. Change the current working directory to the location where you want the cloned directory.
3. Type git clone, and then paste the URL you copied earlier. It will look like this, with your GitHub username instead of YOUR-USERNAME:

$ git clone https://github.com/heyaadarsh/2110990005

4. Press **Enter**. Your local clone will be created.

```
lpdga@DESKTOP-OID626K MINGW64 ~/Desktop (master)
$ git clone https://github.com/heyaadarsh/2110990005.git
Cloning into '2110990005'...
remote: Enumerating objects: 16, done.
remote: Counting objects: 100% (16/16), done.
remote: Compressing objects: 100% (15/15), done.
remote: Total 16 (delta 1), reused 15 (delta 0), pack-reused 0
Receiving objects: 100% (16/16), 1.31 MiB | 2.31 MiB/s, done.
Resolving deltas: 100% (1/1), done.

lpdga@DESKTOP-OID626K MINGW64 ~/Desktop (master)
$ cd 2110990005/
```

**Step 5:** That's it you have made changes to the file and it's time to stage and commit the file and push. **Git add** will add the file to commit.

```
lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/2110990005 (master)
$ ls
 2110990005_TASK1.1_SCM.pdf  'C++ Programs'/

lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/2110990005 (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean

lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/2110990005 (master)
$ git branch
* master
```

```
MINGW64:/c/Users/lpdga/Desktop/2110990005
lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/2110990005 (master)
$ git checkout aadarsh
Switched to branch 'aadarsh'

lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/2110990005 (aadarsh)
$ git branch
* aadarsh
  master

lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/2110990005 (aadarsh)
$ git status
On branch aadarsh
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        C++ Programs/Patterns/

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

lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/2110990005 (aadarsh)
$ git add .
```

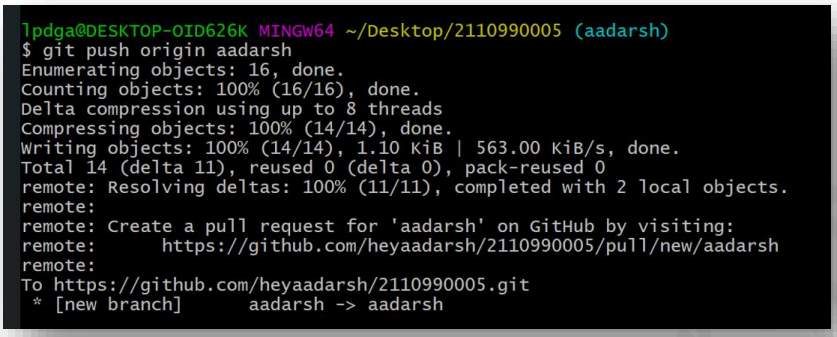
**Step 6:** Commit the changes by below commit code. Make sure you add the comment this will make the other user understand the other user to find what's your change is all about.

```
git commit -m "Commit Message"
```

```
lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/2110990005 (aadarsh)
$ git commit -m "Added Patterns Code in C++ Programs Directory"
[aadarsh 4219af2] Added Patterns Code in C++ Programs Directory
 10 files changed, 170 insertions(+)
 create mode 100644 C++ Programs/Patterns/sqpattern1.cpp
 create mode 100644 C++ Programs/Patterns/sqpattern2.cpp
 create mode 100644 C++ Programs/Patterns/sqpattern3.cpp
 create mode 100644 C++ Programs/Patterns/sqpattern4.cpp
 create mode 100644 C++ Programs/Patterns/tripattern1.cpp
 create mode 100644 C++ Programs/Patterns/tripattern2.cpp
 create mode 100644 C++ Programs/Patterns/tripattern3.cpp
 create mode 100644 C++ Programs/Patterns/tripattern4.cpp
 create mode 100644 C++ Programs/Patterns/tripattern5.cpp
 create mode 100644 C++ Programs/Patterns/tripattern6.cpp
```

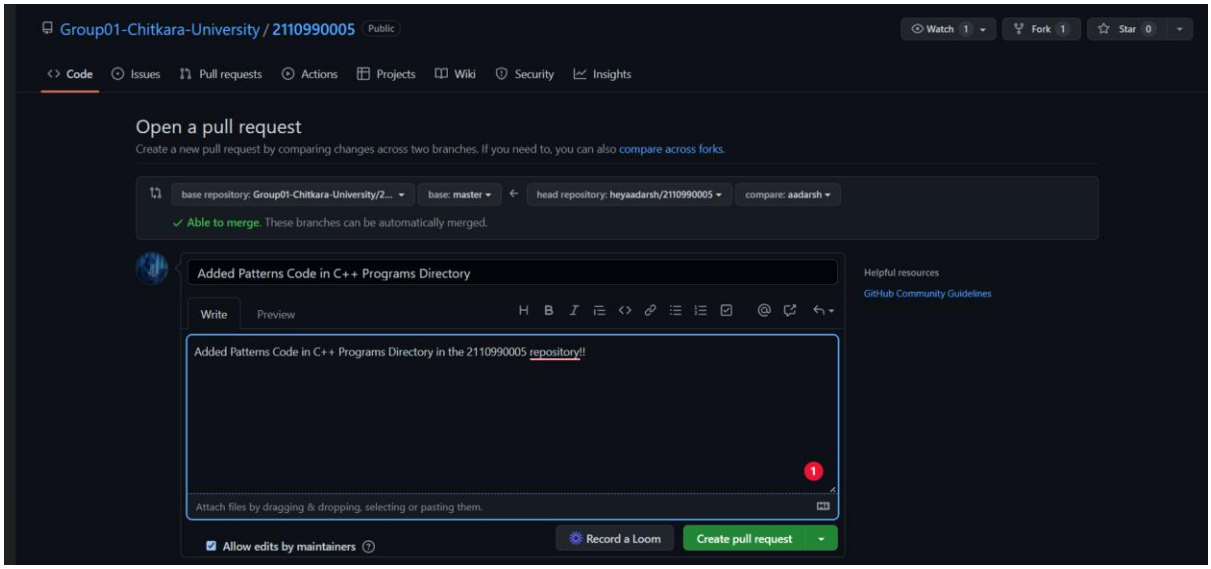
**Step 7:** Pushing your repository to GitHub, here I created this master branch so I'm pushing the file to that branch.

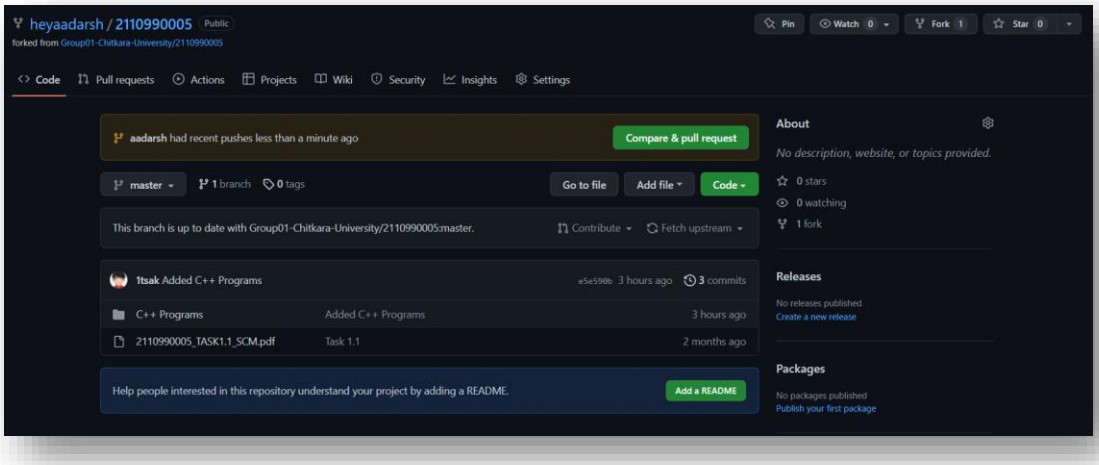
```
git push origin <name of your branch>
```

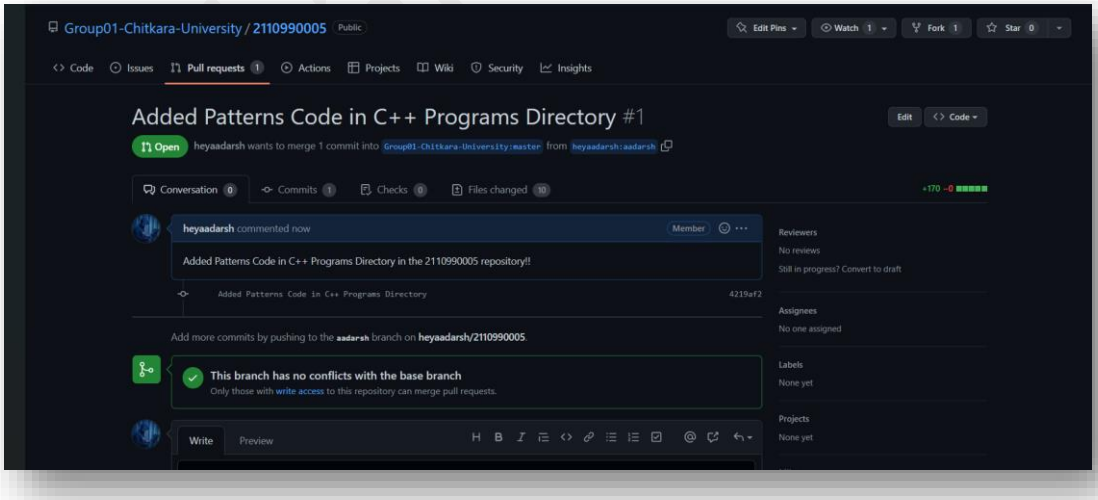



**Step 8:** Now jump to your repository, under your branch that you pushed you can see an option to merge the pull request to the main branch.

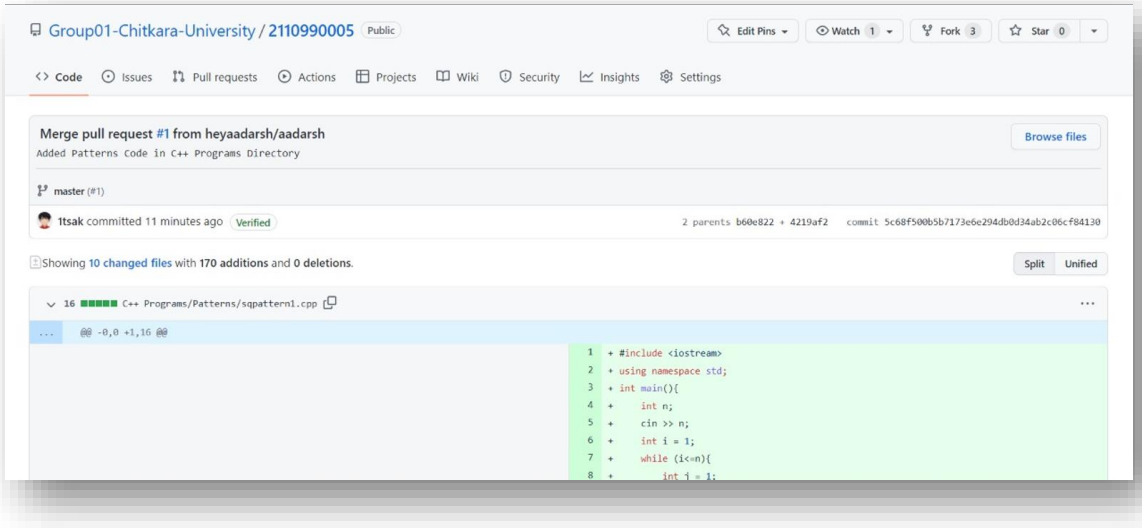
**Note:** A pull request allows your changes to be merged with the original project.




**Step 9:** Now go to GitHub and accept the merge request.




**Step 10:** Make sure to add a good comment in the merge and explain what's your commit is all about.

**Step 11:** In few Organisations, there will be Github actions scheduled for auto-check and any one of the code reviewers will review the file and approve the changes or suggest any issue with your pull request.

---

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

---

### Understanding GitHub Conflicts

Let's imagine a case where two developers working on one repository by creating 2 different branch and same code line and both of them pulled the file to main repository, Now the GitHub is good understanding the code and merge the conflicts automatically but it fails understanding if developer has put any comments. Now Let's look into How to resolve merge conflicts in GitHub.

### Let's resolve GitHub Conflicts

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

## Commit in aadarsh Branch

```
lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/2110990005/C++ Programs (aadarsh)
$ nano hello_world.cpp

lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/2110990005/C++ Programs (aadarsh)
$ git status
On branch aadarsh
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:   hello_world.cpp

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

lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/2110990005/C++ Programs (aadarsh)
$ git add .

lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/2110990005/C++ Programs (aadarsh)
$ git commit -m "Modified hello_world.cpp file | Branch: aadarsh"
[aadarsh 0e3cdce] Modified hello_world.cpp file | Branch: aadarsh
 1 file changed, 3 insertions(+), 3 deletions(-)
```

---

**Step 2:** Now try to merge from master branch it will give Conflicts Error.

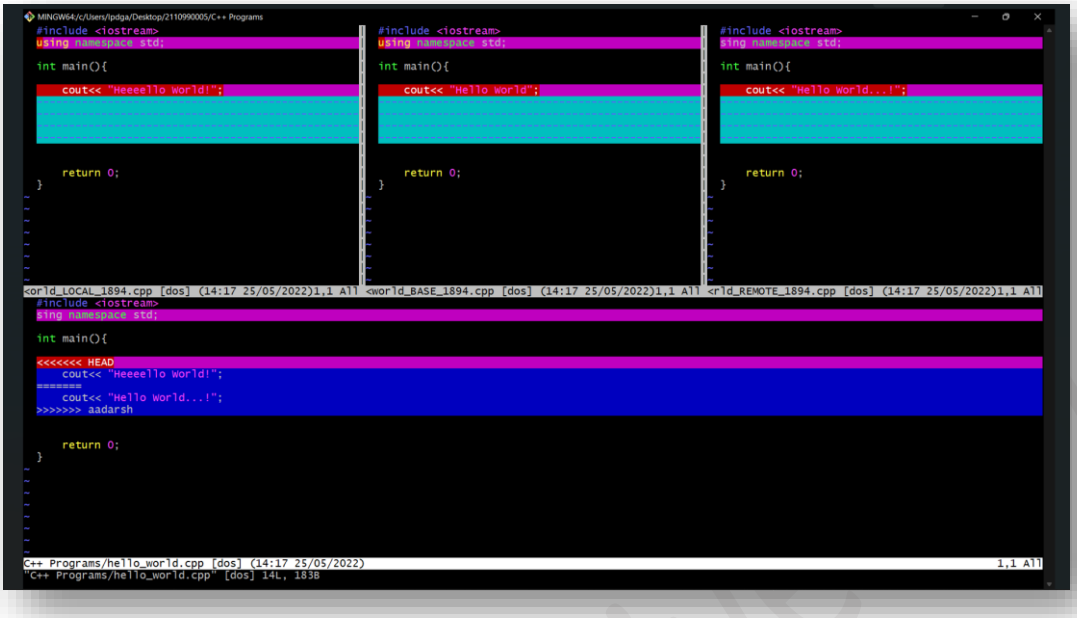
```
lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/2110990005/C++ Programs (master)
$ git merge aadarsh
Auto-merging C++ Programs/hello_world.cpp
CONFLICT (content): Merge conflict in C++ Programs/hello_world.cpp
Automatic merge failed; fix conflicts and then commit the result.
```

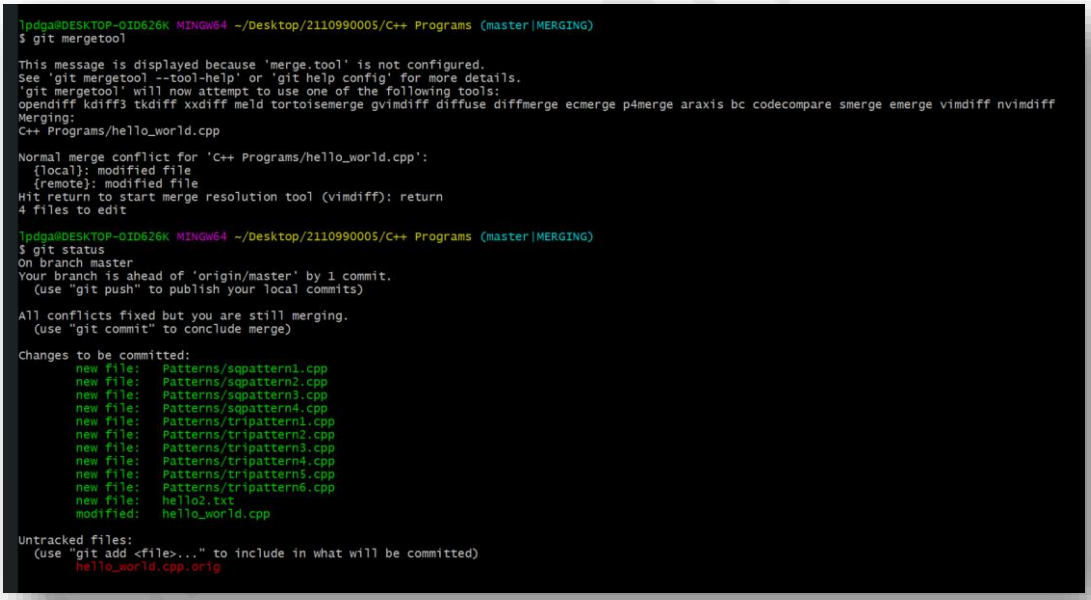
**Step 3:** Use Command “git mergetool” to solve the conflict.

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

```
lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/2110990005/C++ Programs (master)
$ git merge aadarsh
Auto-merging C++ Programs/hello_world.cpp
CONFLICT (content): Merge conflict in C++ Programs/hello_world.cpp
Automatic merge failed; fix conflicts and then commit the result.

lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/2110990005/C++ Programs (master|MERGING)
$ git mergetool
```

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



**Step 5:** Commit Merge to do this merging with the main branch

```
lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/2110990005/C++ Programs (master|MERGING)
$ git add .

lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/2110990005/C++ Programs (master|MERGING)
$ git commit -m "merged aadarsh branch to master"
[master 3ba4dda] merged aadarsh branch to master

lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/2110990005/C++ Programs (master)
$
```

## *Aim: Reset and revert*

### How to reset a Git commit

Let's start with the Git command reset. Practically, you can think of it as a "rollback"—it points your local environment back to a previous commit. By "local environment," we mean your local repository, staging area, and working directory.

Take a look at Figure 1. Here we have a representation of a series of commits in Git. A branch in Git is simply a named, movable pointer to a specific commit. In this case, our branch *master* is a pointer to the latest commit in the chain.

```
$ git log --oneline
b764644 File with three lines
7c709f0 File with two lines
9ef9173 File with one line
```

What happens if we want to roll back to a previous commit. Simple—we can just move the branch pointer. Git supplies the reset command to do this for us. For example, if we want to reset *master* to point to the commit two back from the current commit, we could use either of the following methods:

$ git reset 9ef9173 (using an absolute commit SHA1 value 9ef9173)

or

$ git reset current~2 (using a relative value -2 before the "current" tag)

Figure 2 shows the results of this operation. After this, if we execute a git log command on the current branch (*master*), we'll see just the one commit.

```
$ git log --oneline
9ef9173 File with one line
```

The git reset command also includes options to update the other parts of your local environment with the contents of the commit where you end up. These

options include: hard to reset the commit being pointed to in the repository, populate the working directory with the contents of the commit, and reset the staging area; soft to only reset the pointer in the repository; and mixed (the default) to reset the pointer and the staging area.

Using these options can be useful in targeted circumstances such as git reset --hard <commit sha1 | reference>. This overwrites any local changes you haven't committed. In effect, it resets (clears out) the staging area and overwrites content in the working directory with the content from the commit you reset to. Before you use the hard option, be sure that's what you really want to do, since the command overwrites any uncommitted changes.

```
lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/Reset and Revert (master)
$ git log
commit 3779d3f0e7c1c4e17adc397b55a5f68fa09994f9 (HEAD -> master, origin/master)
Author: heyaadarsh <aadarsh0001.be21@chitkara@edu.in>
Date:   Thu Jun 2 15:18:28 2022 +0530

    Reverting commit-2 | Go to Patna

commit c393f20ed549448f28c3068af35b294415c7dc75
Author: heyaadarsh <aadarsh0001.be21@chitkara@edu.in>
Date:   Thu Jun 2 15:14:27 2022 +0530

    Commit-3 | Go to Mumbai

commit b220dd0ce442f3b560aa88c0a5514e7c3ac213ec
Author: heyaadarsh <aadarsh0001.be21@chitkara@edu.in>
Date:   Thu Jun 2 15:12:56 2022 +0530

    Commit-2 | Go to Patna

commit bd9f7280aefc60169c32f32af8e2fce9442108bb
Author: heyaadarsh <aadarsh0001.be21@chitkara@edu.in>
Date:   Thu Jun 2 15:10:32 2022 +0530

    Commit-1 | Go to Delhi
```

```
lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/Reset and Revert (master)
$ git reset --soft c393f20ed549448f28c3068af35b294415c7dc75

lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/Reset and Revert (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   go.cpp
```

# How to revert a Git commit

The net effect of the git revert command is similar to reset, but its approach is different. Where the reset command moves the branch pointer back in the chain (typically) to "undo" changes, the revert command adds a new commit at the end of the chain to "cancel" changes. The effect is most easily seen by looking at Figure 1 again. If we add a line to a file in each commit in the chain, one way to get back to the version with only two lines is to reset to that commit, i.e., git reset HEAD~1.

Another way to end up with the two-line version is to add a new commit that has the third line removed—effectively cancelling out that change. This can be done with a git revert command, such as:

```
$ git revert HEAD
```

Because this adds a new commit, Git will prompt for the commit message:

```
Revert "File with three lines"

This reverts commit b764644bad524b804577684bf74e7bca3117f554.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Changes to be committed:
#       modified:   file1.txt
#
```

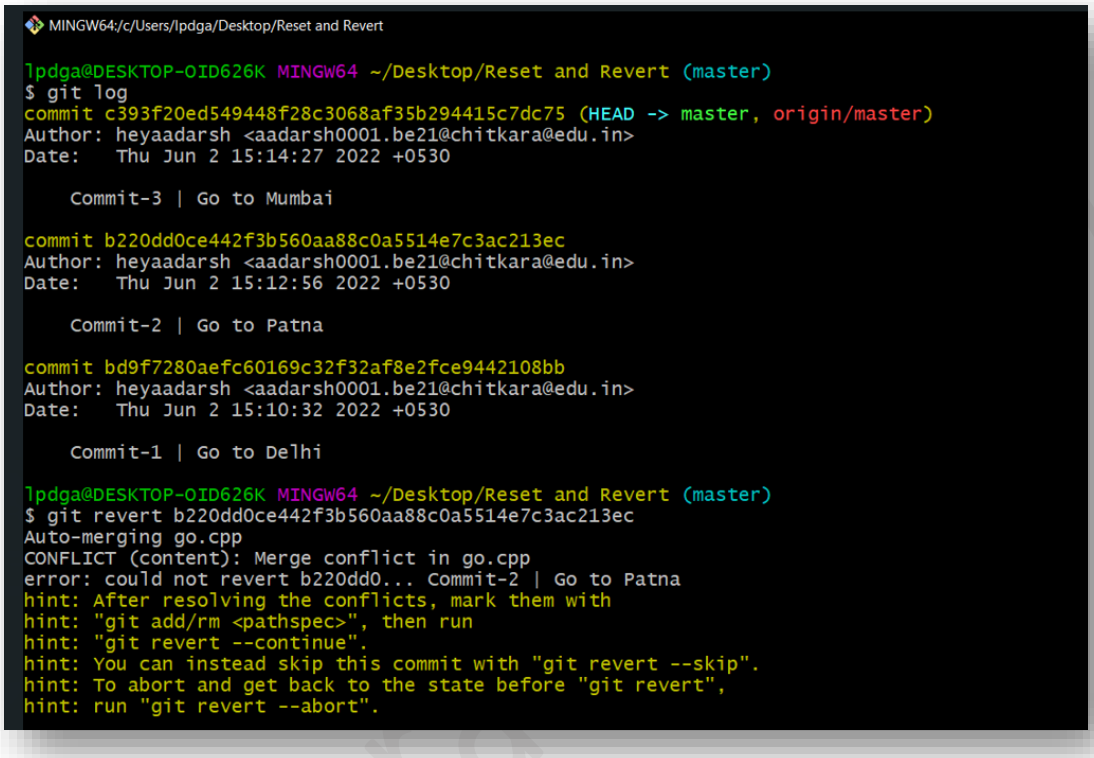
Figure 3 (below) shows the result after the revert operation is completed.

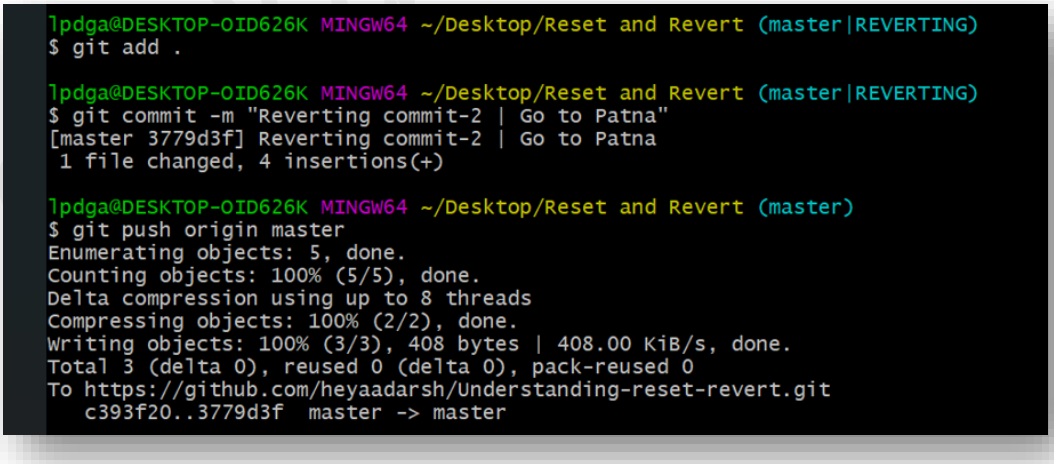
If we do a git log now, we'll see a new commit that reflects the contents before the previous commit.

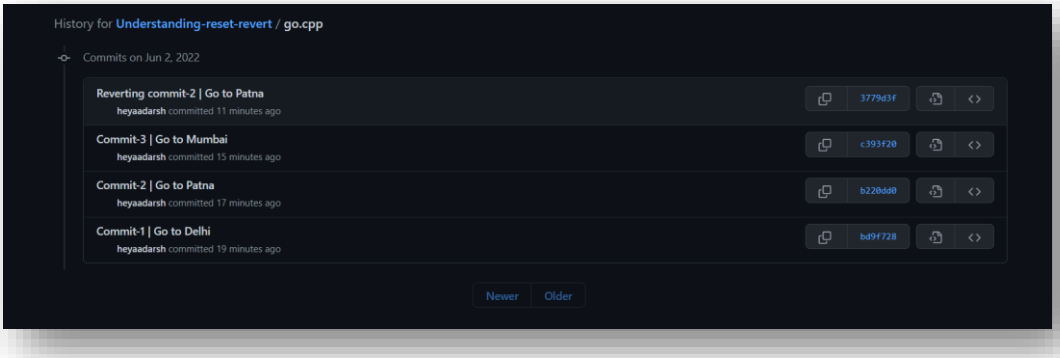
```
$ git log --oneline
11b7712 Revert "File with three lines"
b764644 File with three lines
7c709f0 File with two lines
9ef9173 File with one line
```

Here are the current contents of the file in the working directory:

```
$ cat <filename>
Line 1
Line 2
```





## Experiment No. 2.0 [Project Report]

### 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 operatingsystem kernel.

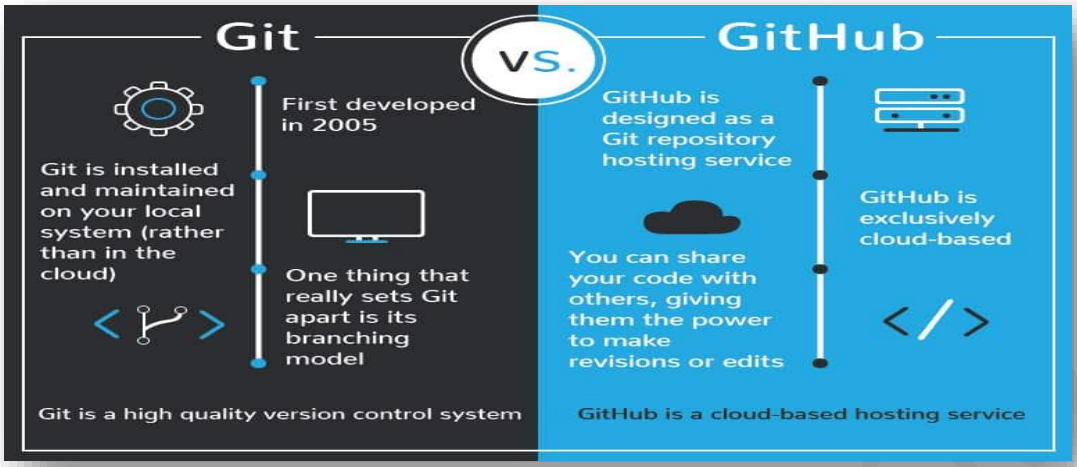
Git is a speedy and efficient distributed VCS tool that can handle projects of anysize, from small to very large ones. Git provides cheap local branching, convenient staging areas, and multiple workflows. It is free, open-source software that lowers the cost because developers can use Git without paying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

Git is an example of a distributed version control system (DVCS) (henceDistributed 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 yourcomputer, or it can be a storage space on GitHub or another online host. You can keep code files, text files, image files, you name it, inside a repository.

## What is Version Control System (VCS)?

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

## Types of VCS

1. Local Version Control System
2. Centralized Version Control System
3. Distributed Version Control System

I. **Local Version Control System:** Local Version Control System is

located inyour 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 tothe 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 everythingrelated 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 thelatest version, but each collaborator will have an exact copy of the main repository on their local machines. Each user has their own repository and aworking 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.

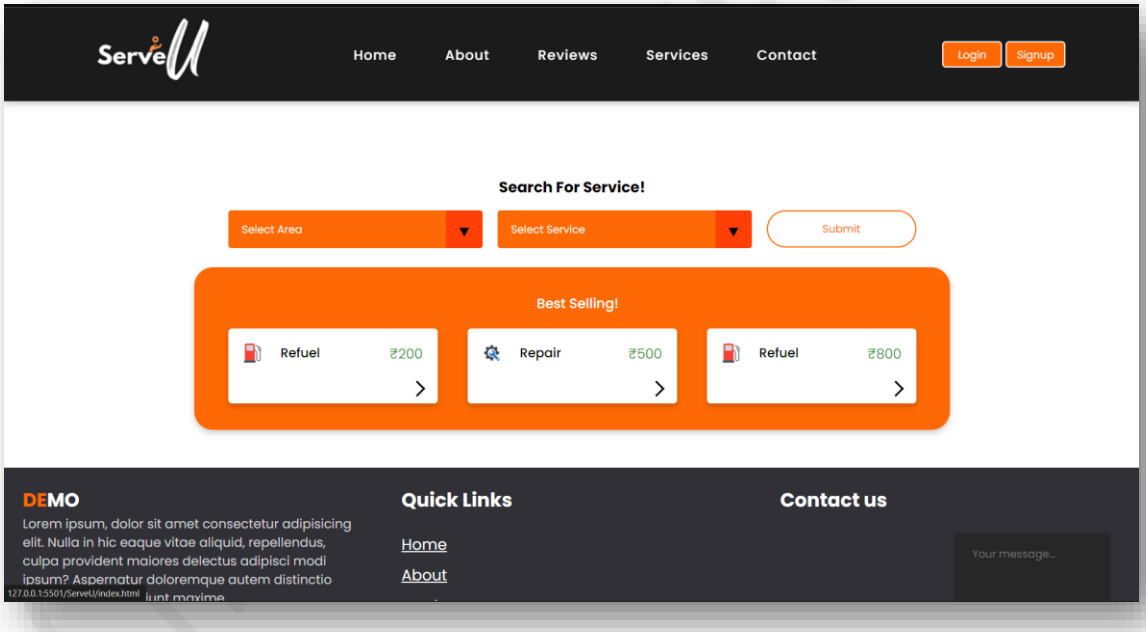
## *Problem Statement*

“Build a website and deploy it on GitHub”

Many a times, while travelling we get into certain unforeseen situations where we run out of fuel or our vehicle gets overheated. On a deserted road, the possibility of finding a petrol pump nearby or a mechanic is negligible. In a situation like this we would need someone to provide assistance to help us get out of that situation. Here comes "ServeU” addressing the real time vehicle breakdown problems of customers in day-to-day life.

## Solution

Vehicle Servicing, Vehicle repairs and Car cleaning - we are your one-stop solution for all things cars. ServeU intends to be the best roadside assistance provider in India by addressing the real time vehicle breakdown problems of customers in day-to-day life. A brainchild of 5 friends - Aadarsh Kumar, Aakash Jha, Aastha Anand, Aayushi Jain and Abhimanyu Nain, ServeU is a network of technology-enabled automobile service centres, offering a seamless car and bike service experience at the convenience of a tap. With our highly skilled technicians, manufacturer recommended procedures and the promise of genuine spare parts we are your best bet. Stay in the comforts of your home or office and make the most of our complimentary pick-up and drop-in service. Count on us to be your personal vehicle care expert, advisor and mechanic.




## Objective

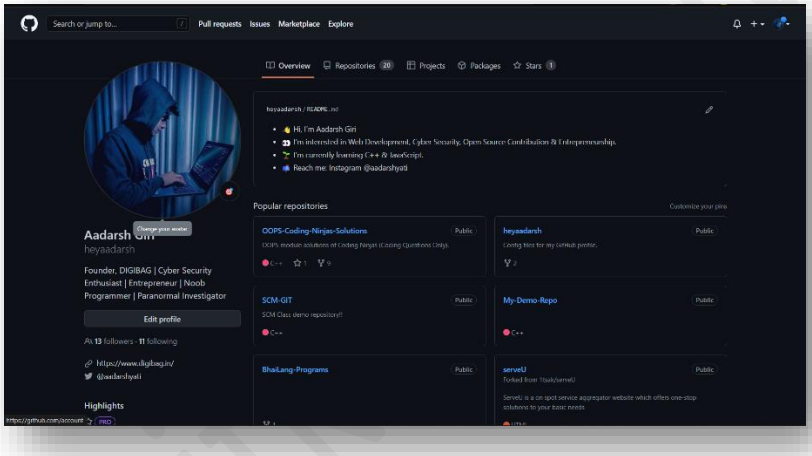
The objective of this project is to associate programming with git because:

1. This is required because the collaboration makes the team work easy.

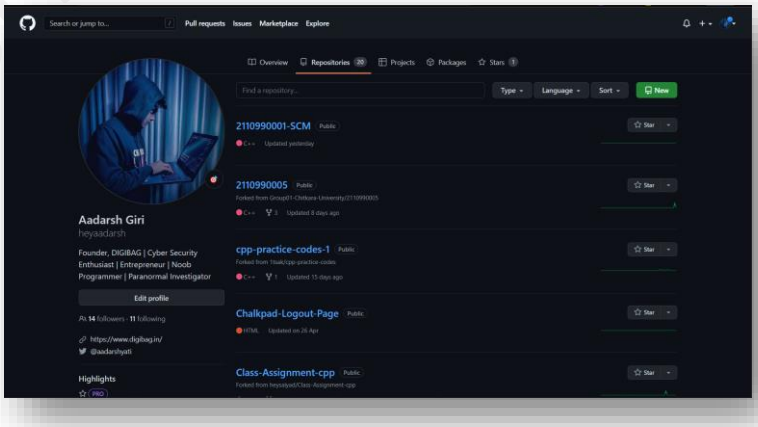
2. The code becomes manageable and we can build a clean repository.
3. Tracking and resolving of the errors is quite feasible in this process.
4. Moreover, we can make our locally available projects, globally available.

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

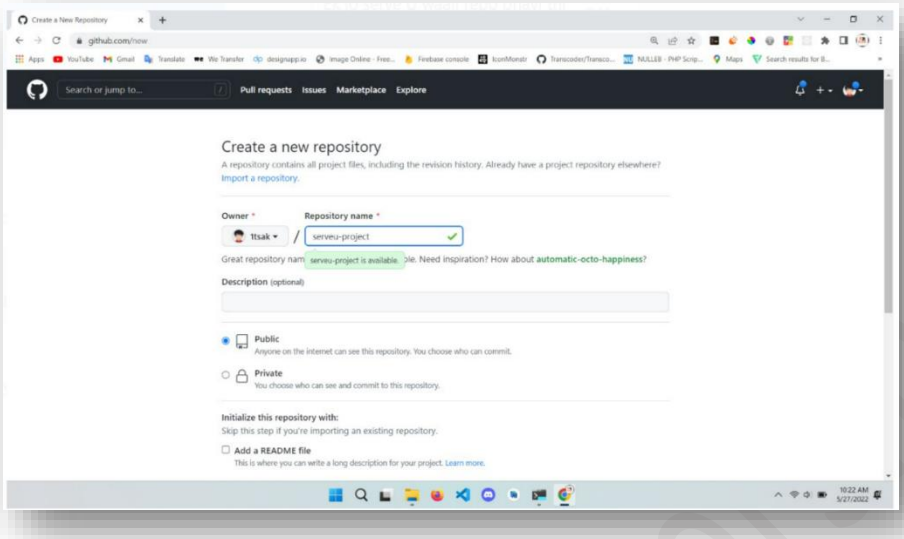
1) Login to your GitHub account and you will land on thehomepage as shown below. Click on Repositories option in the menu bar.




2) Click on the 'New' button in the top right corner.

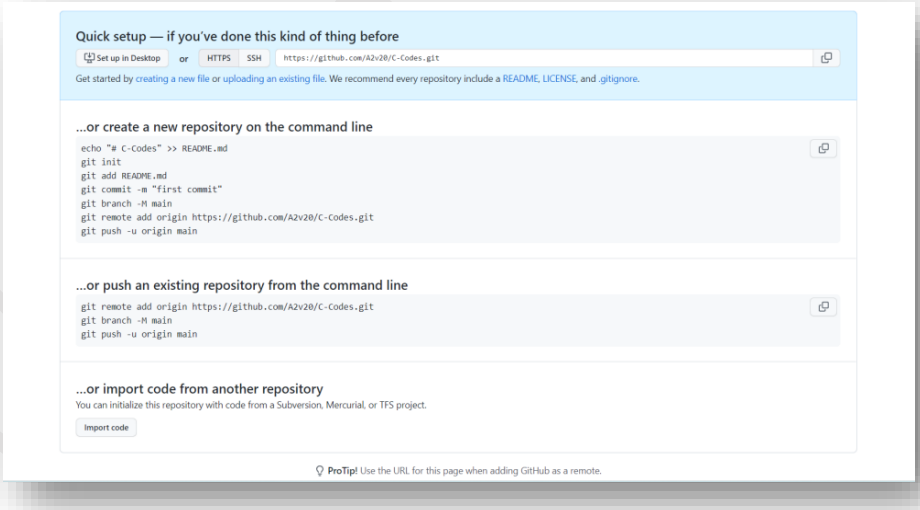


3) Enter the Repository name and add the description of the repository.
4) Select if you want the repository to be public or private.

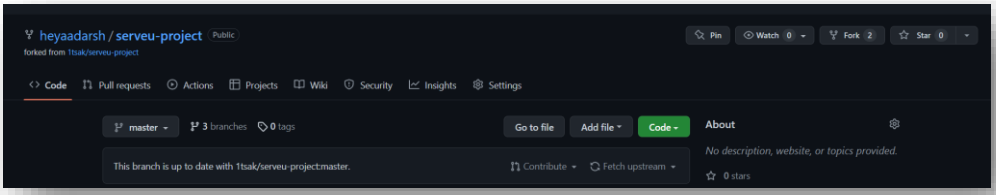



*(Repository created by repository owner: team member- Aakash)*

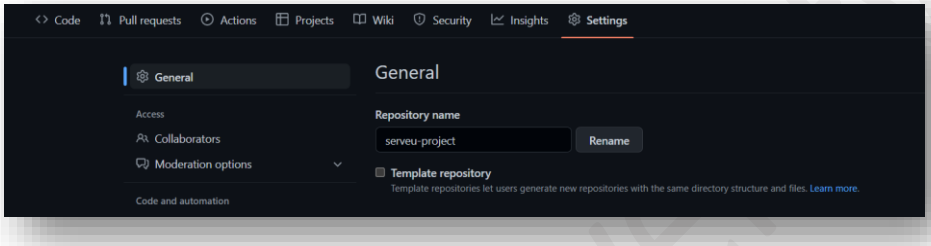
5) If you want to import code from an existing repository select the import code option.



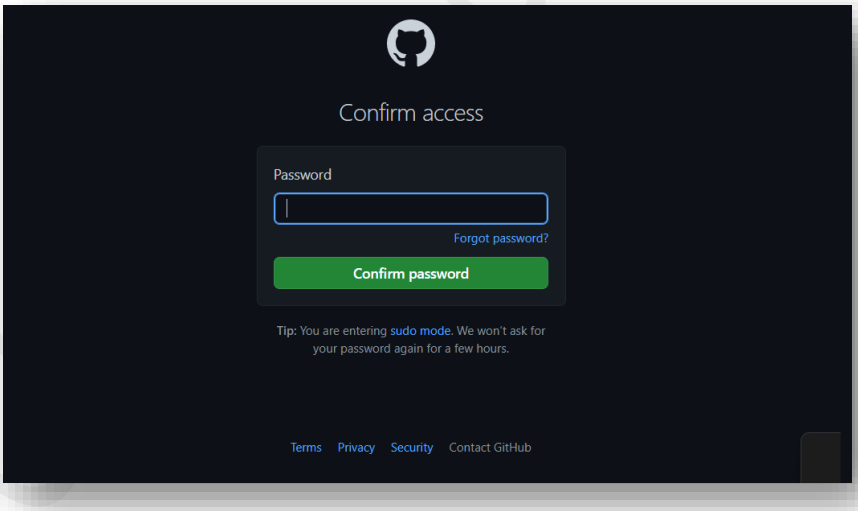

6) Now, you have created your repository successfully.
7) To add members to your repository open your repository and select settings option in the navigation bar.




8) Click on Collaborators option under the access tab.

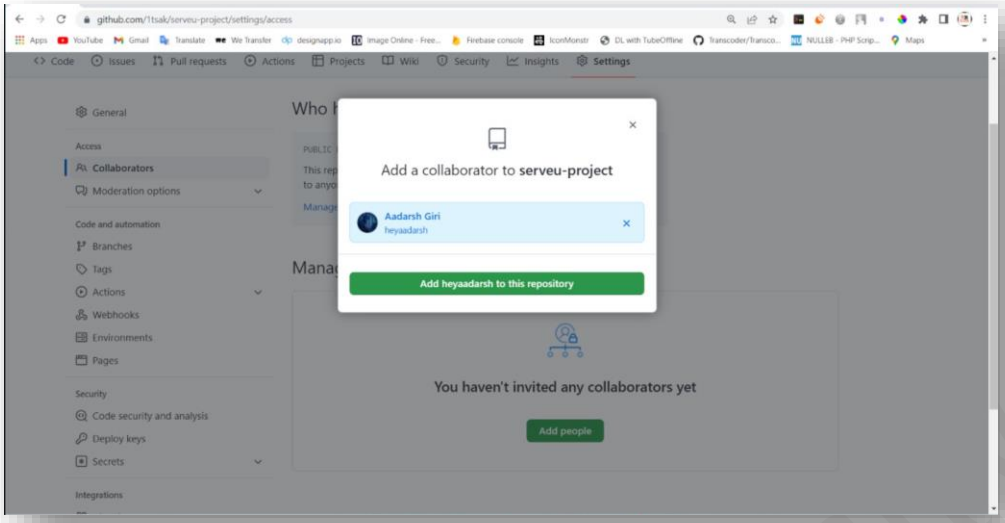



9) After clicking on collaborators GitHub asks you to enter your password to confirm the access to the repository.



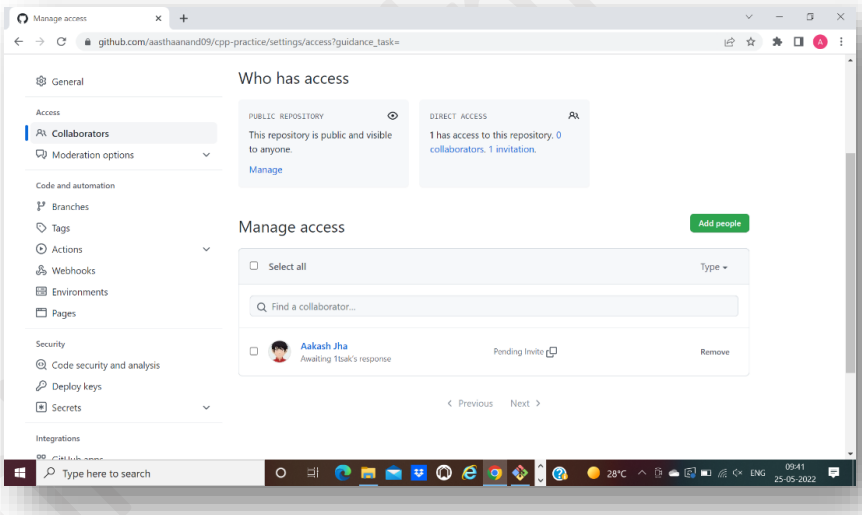

10) After entering the password you can manage access and add/remove team members to your project.

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

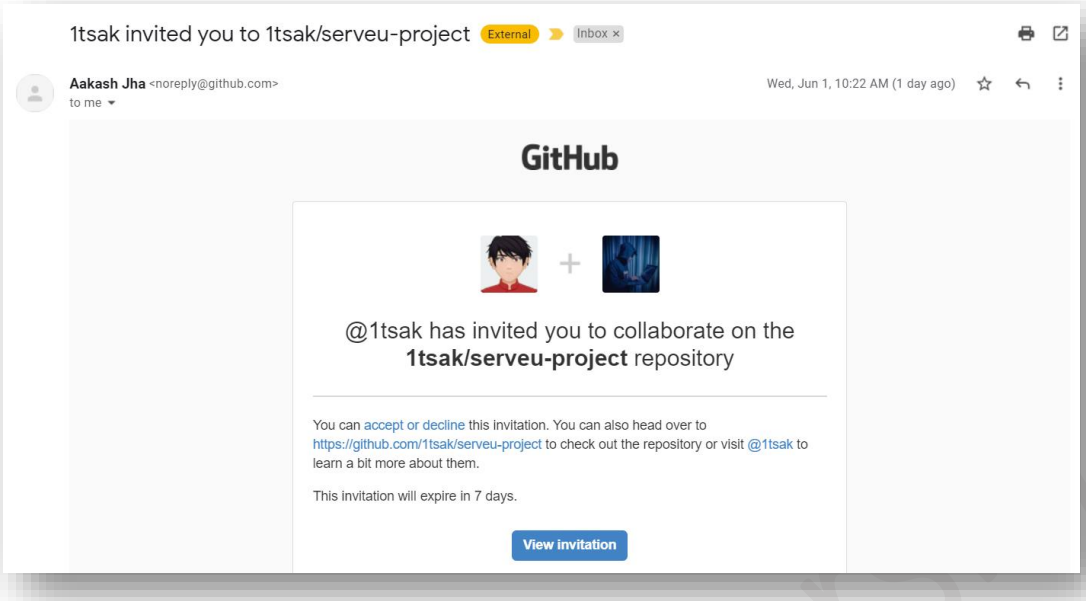



*(Collaborators added by repository owner: team member- Aakash)*

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

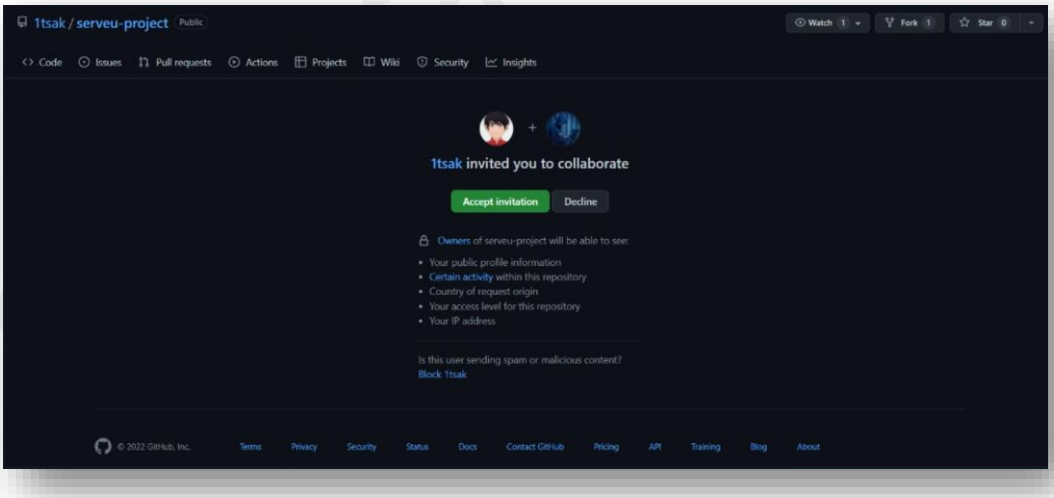



13) To accept the invitation from your team member, open your mail registered with GitHub.



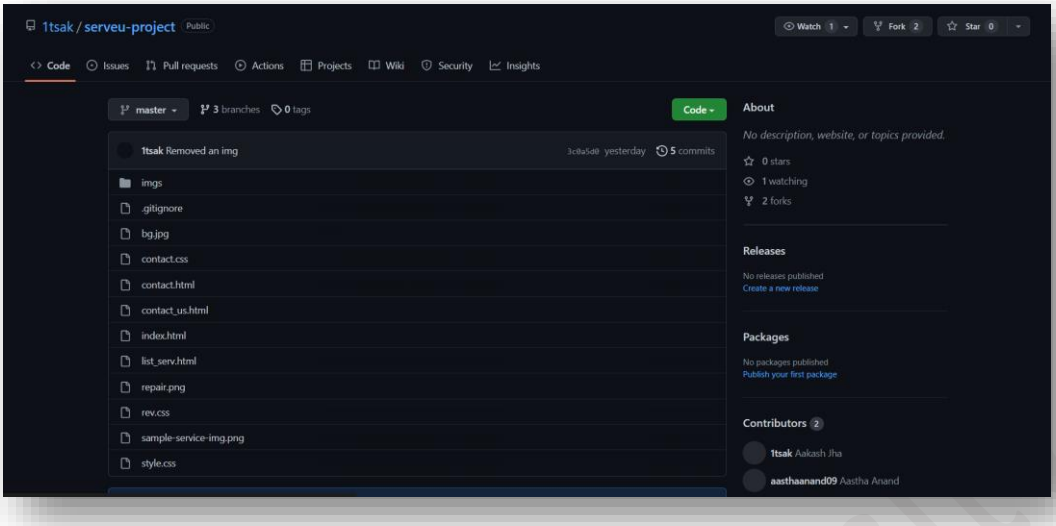

14) You will receive an invitation mail from the repository owner. Open the email and click on accept invitation.

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



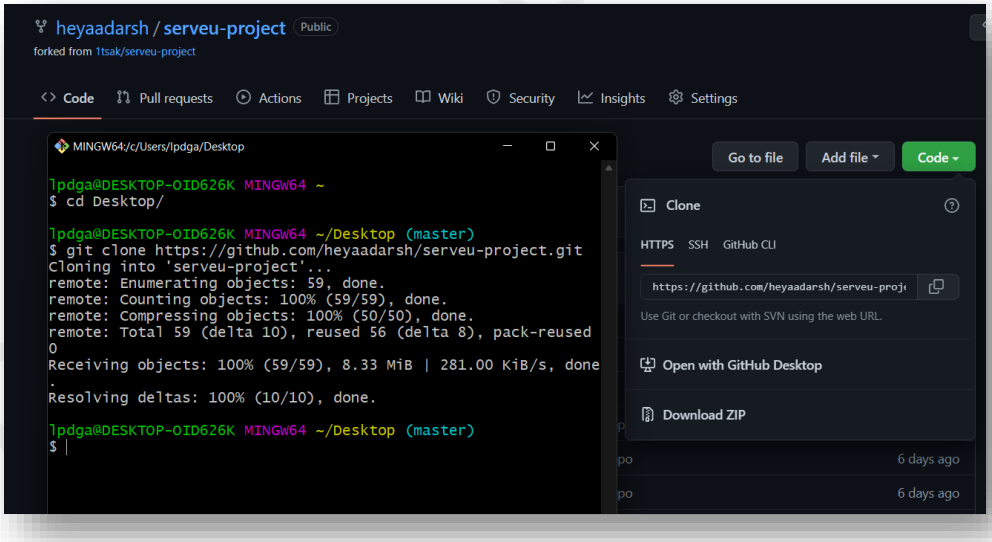

16) You will be shown the option that you are now allowed to push.

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




## *Aim: Open and Close a Pull Request*

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




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

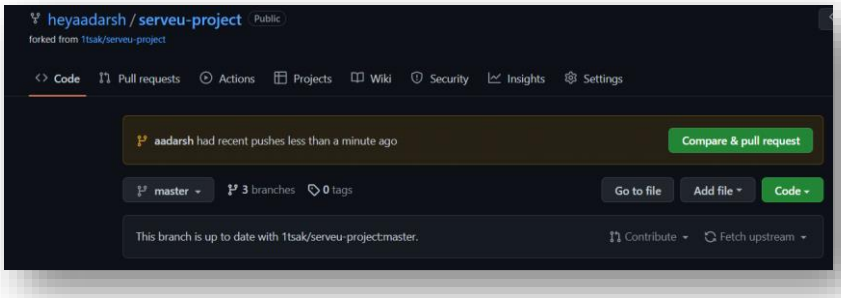
```
lpdga@DESKTOP-OID626K MINGW64 ~/Desktop (master)
$ cd serveu-project/

lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/serveu-project (master)
$ git branch aadarsh

lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/serveu-project (master)
$ git checkout aadarsh
Switched to branch 'aadarsh'

lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/serveu-project (aadarsh)
$ git status
On branch aadarsh
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        review.html
        services.html

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

lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/serveu-project (aadarsh)
$
```

c) Add and commit the changes to the local repository.

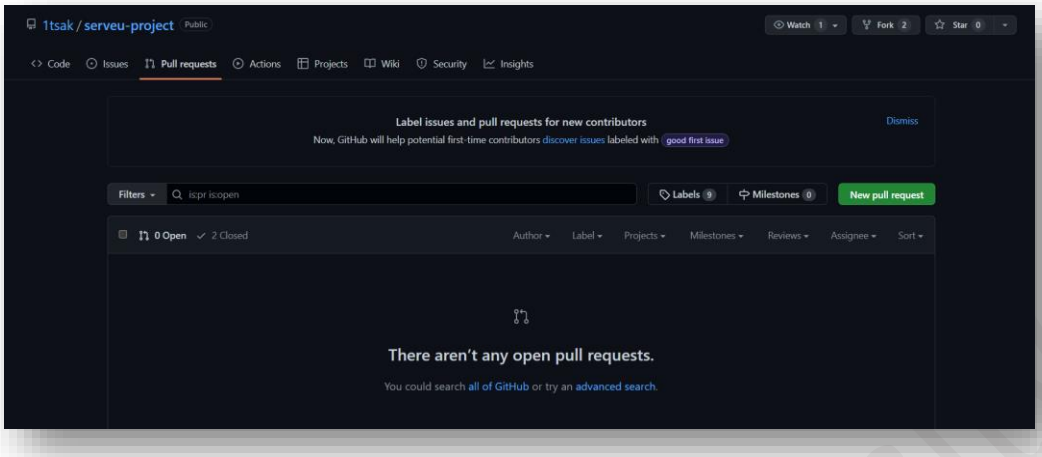
d) Use git push origin *branchname* option to push the new branch to the main repository.

```
lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/serveu-project (aadarsh)
$ git add .

lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/serveu-project (aadarsh)
$ git commit -m "Added Reviews & Services File | From aadarsh Branch"
[aadarsh 2096462] Added Reviews & Services File | From aadarsh Branch
 2 files changed, 414 insertions(+)
 create mode 100644 review.html
 create mode 100644 services.html

lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/serveu-project (aadarsh)
$ git push origin aadarsh
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 5.35 KiB | 5.35 MiB/s, done.
Total 4 (delta 2), reused 2 (delta 1), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
remote:
remote: Create a pull request for 'aadarsh' on GitHub by visiting:
remote:      https://github.com/heyaadarsh/serveu-project/pull/new/aadarsh
remote:
To https://github.com/heyaadarsh/serveu-project.git
 * [new branch]      aadarsh -> aadarsh
```

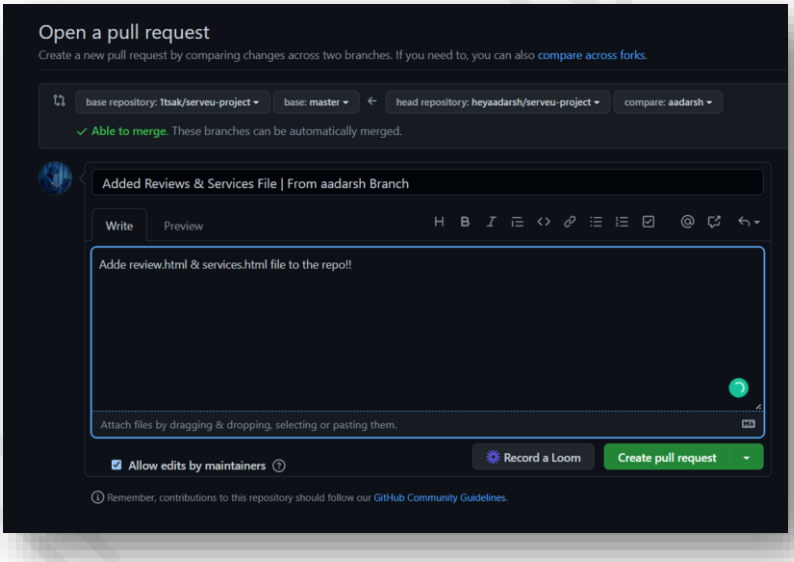
e) After pushing new branch GitHub will either automatically ask you to create a pull request or you can create your own pull request.



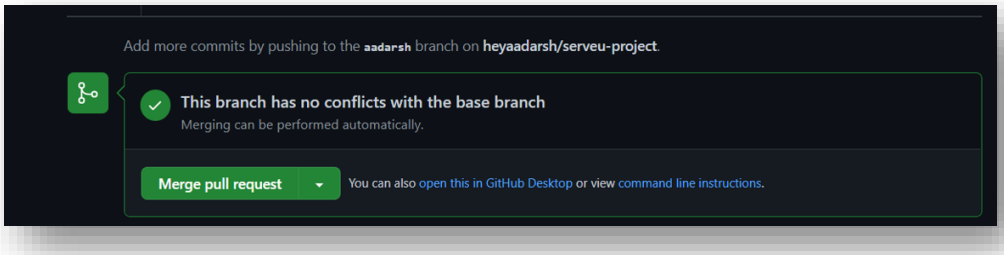
f) To create your own pull request, click on pull request option.




g) GitHub will detect any conflicts and ask you to enter a description of your pull request.




h) After opening a pull request all the team members will be sent the request if they want to merge or close the request.

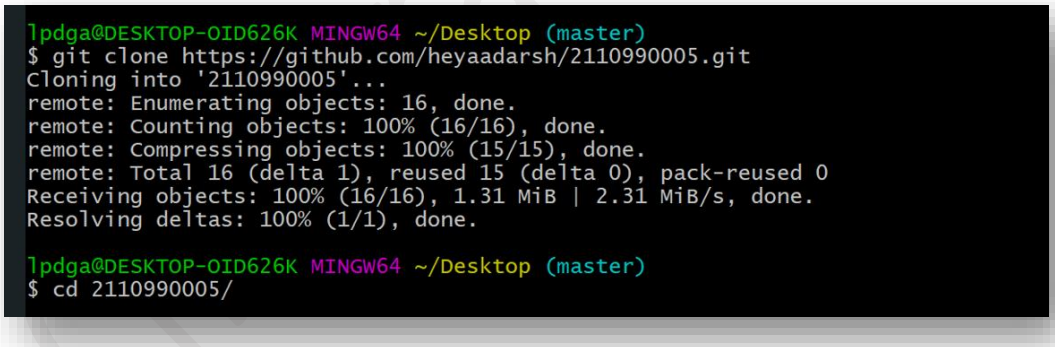


i) If the team member chooses not to merge your pull request they will close your pull request.
j) To close the pull request simply click on close pull request and add comment/ reason why you closed the pull request.
k) You can see all the pull request generated and how they were dealt with by clicking on pull request option.

## *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 byany other team members as a maintainer follow the procedure given below:

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



```
lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/2110990005 (master)
$ ls
 2110990005_TASK1.1_SCM.pdf  'C++ Programs'/

lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/2110990005 (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean

lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/2110990005 (master)
$ git branch
* master
```

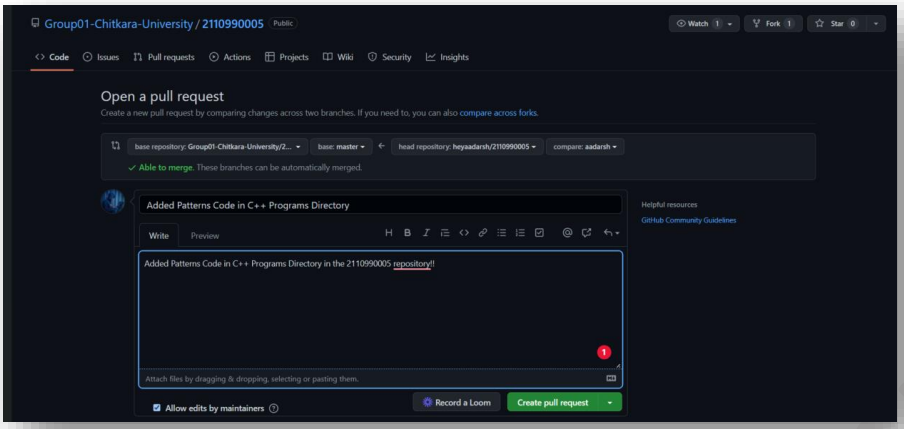
```
MINGW64/c/Users/lpdga/Desktop/2110990005
lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/2110990005 (master)
$ git checkout aadarsh
Switched to branch 'aadarsh'

lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/2110990005 (aadarsh)
$ git branch
* aadarsh
  master

lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/2110990005 (aadarsh)
$ git status
On branch aadarsh
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        C++ Programs/Patterns/

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

lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/2110990005 (aadarsh)
$ git add .
```

2. Push the modified branch using git push origin branchname.

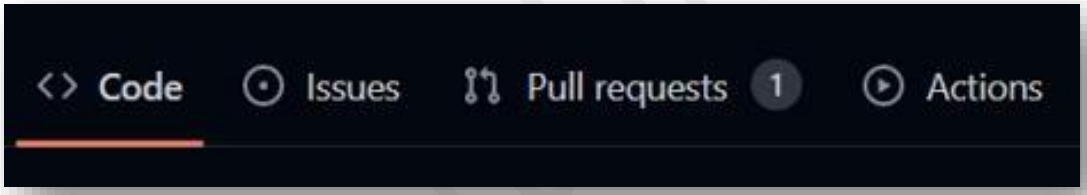
```
lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/2110990005 (aadarsh)
$ git commit -m "Added Patterns Code in C++ Programs Directory"
[aadarsh 4219af2] Added Patterns Code in C++ Programs Directory
 10 files changed, 170 insertions(+)
 create mode 100644 C++ Programs/Patterns/sqpattern1.cpp
 create mode 100644 C++ Programs/Patterns/sqpattern2.cpp
 create mode 100644 C++ Programs/Patterns/sqpattern3.cpp
 create mode 100644 C++ Programs/Patterns/sqpattern4.cpp
 create mode 100644 C++ Programs/Patterns/tripattern1.cpp
 create mode 100644 C++ Programs/Patterns/tripattern2.cpp
 create mode 100644 C++ Programs/Patterns/tripattern3.cpp
 create mode 100644 C++ Programs/Patterns/tripattern4.cpp
 create mode 100644 C++ Programs/Patterns/tripattern5.cpp
 create mode 100644 C++ Programs/Patterns/tripattern6.cpp
```

```
lpdga@DESKTOP-OID626K MINGW64 ~/Desktop/2110990005 (aadarsh)
$ git push origin aadarsh
Enumerating objects: 16, done.
Counting objects: 100% (16/16), done.
Delta compression using up to 8 threads
Compressing objects: 100% (14/14), done.
Writing objects: 100% (14/14), 1.10 KiB | 563.00 KiB/s, done.
Total 14 (delta 11), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (11/11), completed with 2 local objects.
remote:
remote: Create a pull request for 'aadarsh' on GitHub by visiting:
remote:      https://github.com/heyaadarsh/2110990005/pull/new/aadarsh
remote:
To https://github.com/heyaadarsh/2110990005.git
 * [new branch]      aadarsh -> aadarsh
```

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

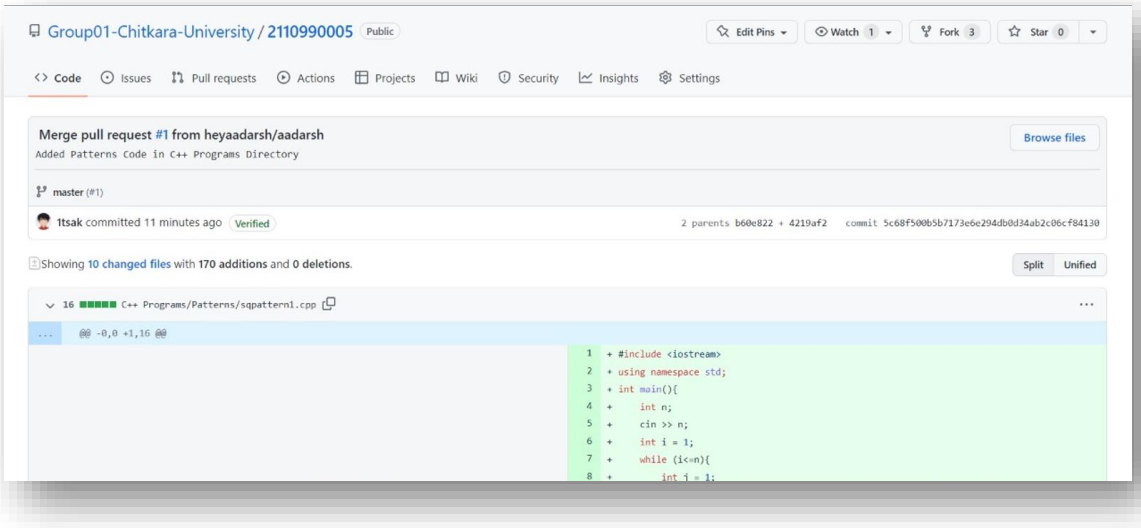



4. The pull request will be created and will be visible to all the team member.
5. Ask your team member to login to his/her Github account.
6. They will notice a new notification in the pull request menu.




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

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




## *Aim: Publish and Print the 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 onthe 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 abetter understanding of who's using your repository and why they're using it.

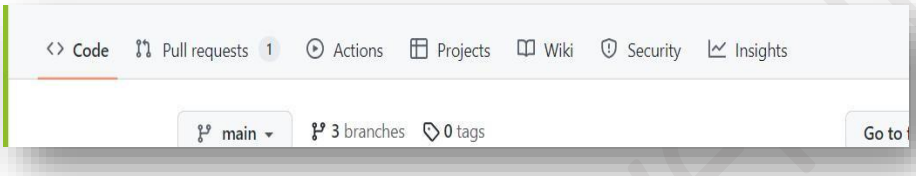
Some repository graphs are available only in public repositories withGitHub Free:

- Pulse
- Contributors

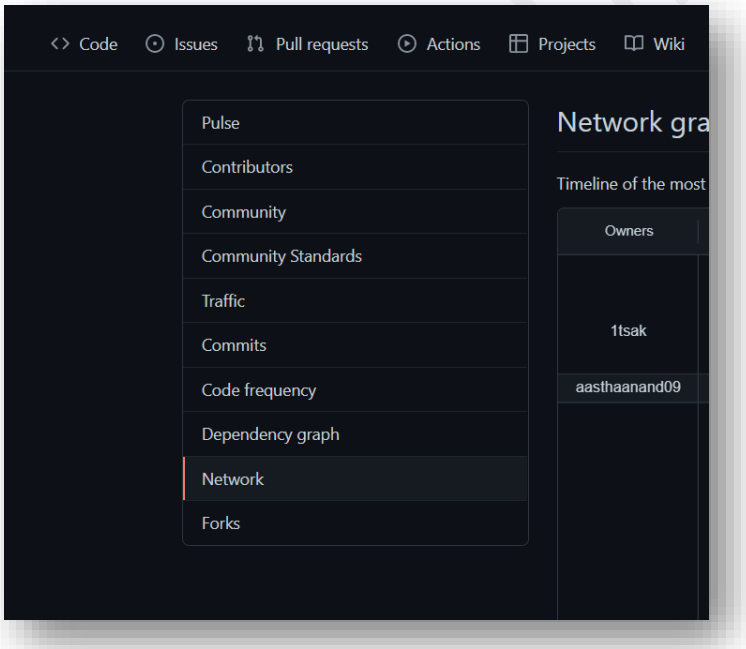
- Traffic
- Commits
- Code frequency
- Network

## Steps to acess network graphs of respective repository

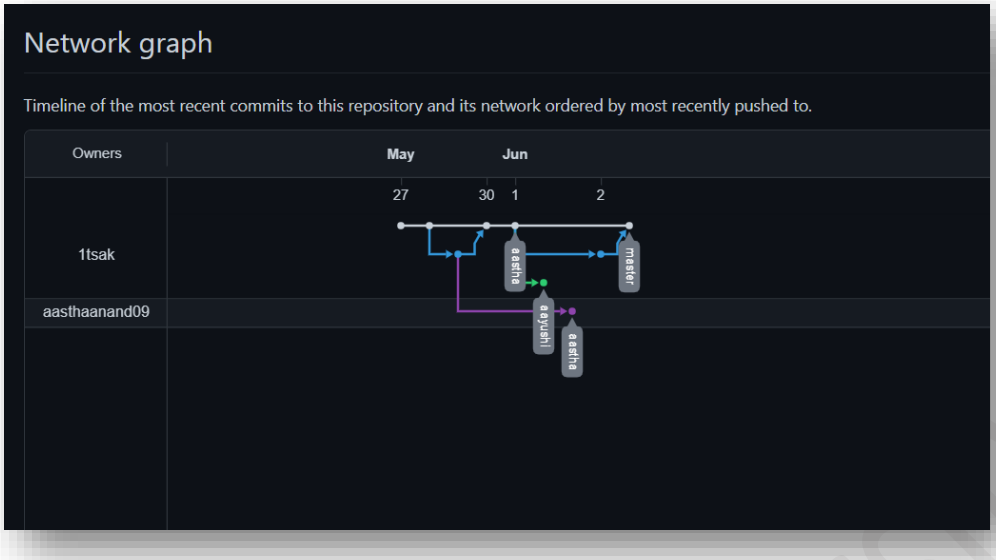
1. On GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click Insights.




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.



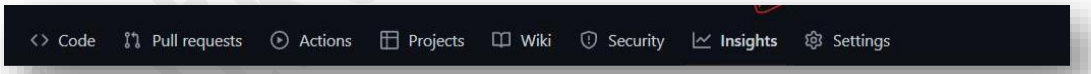

## Listing the forks of a repository

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

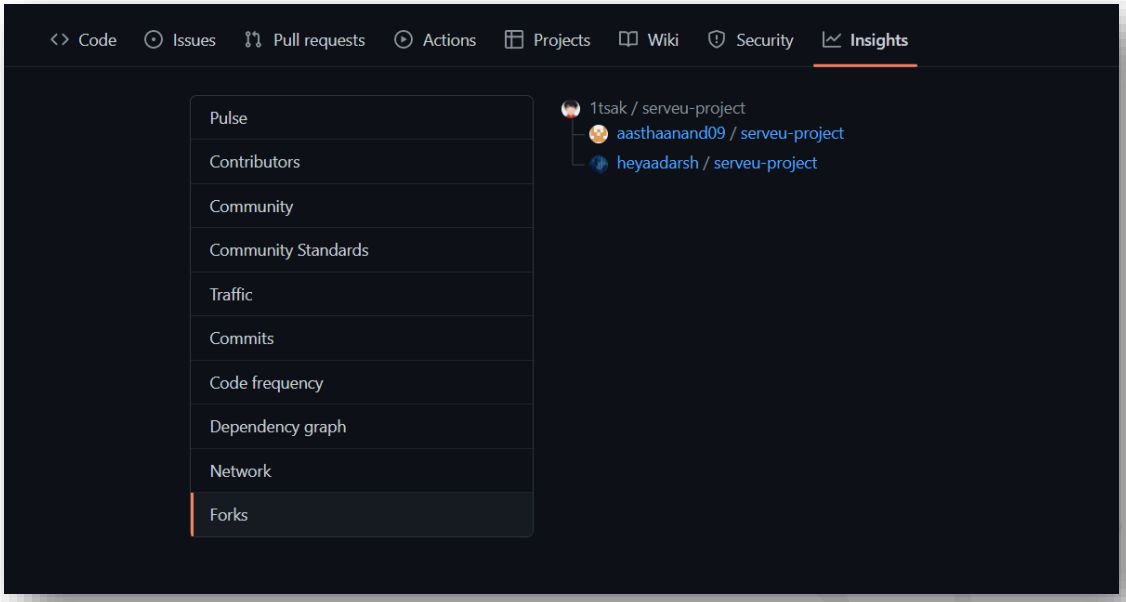
Clicking the number of forks shows you the full network. From there you can click "members" to see who forked the repo.

A. On GitHub.com, navigate to the main page of the repository.

B. Under your repository name, click Insights.




C. 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.

**Thank You!**

**Submitted By-**

Aadarsh Kumar

2110990001

G01-A