Subject Name: **Source Code Management**

Subject Code: **CS181**

Cluster: **Beta**

Department: **DCSE**

Submitted By:
Amit Kumar Dhal
2110990162
G08-A

Submitted To:
Dr. Monit Kapoor

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. **<u>Local Version Control System:</u>** 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.

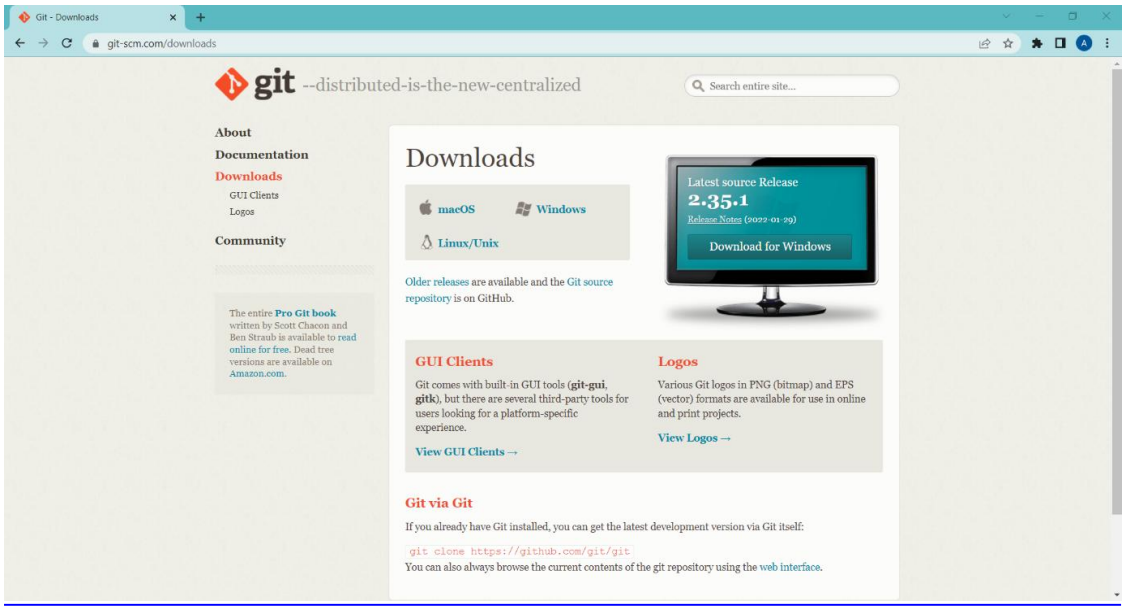
II. **<u>Centralized Version Control System:</u>** In the Centralized Version Control Systems, there will be a single central server that contains all the files related to the project, and many collaborators checkout files from this single server (you will only have a working copy). The problem with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

III. **<u>Distributed Version Control System:</u>** 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.

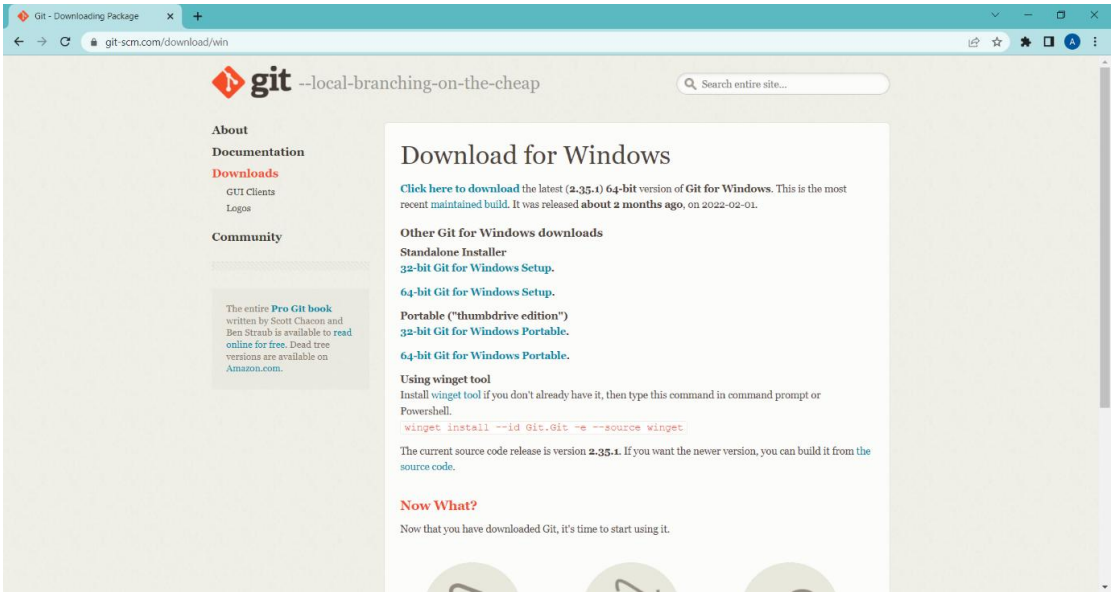
Experiment-1

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.

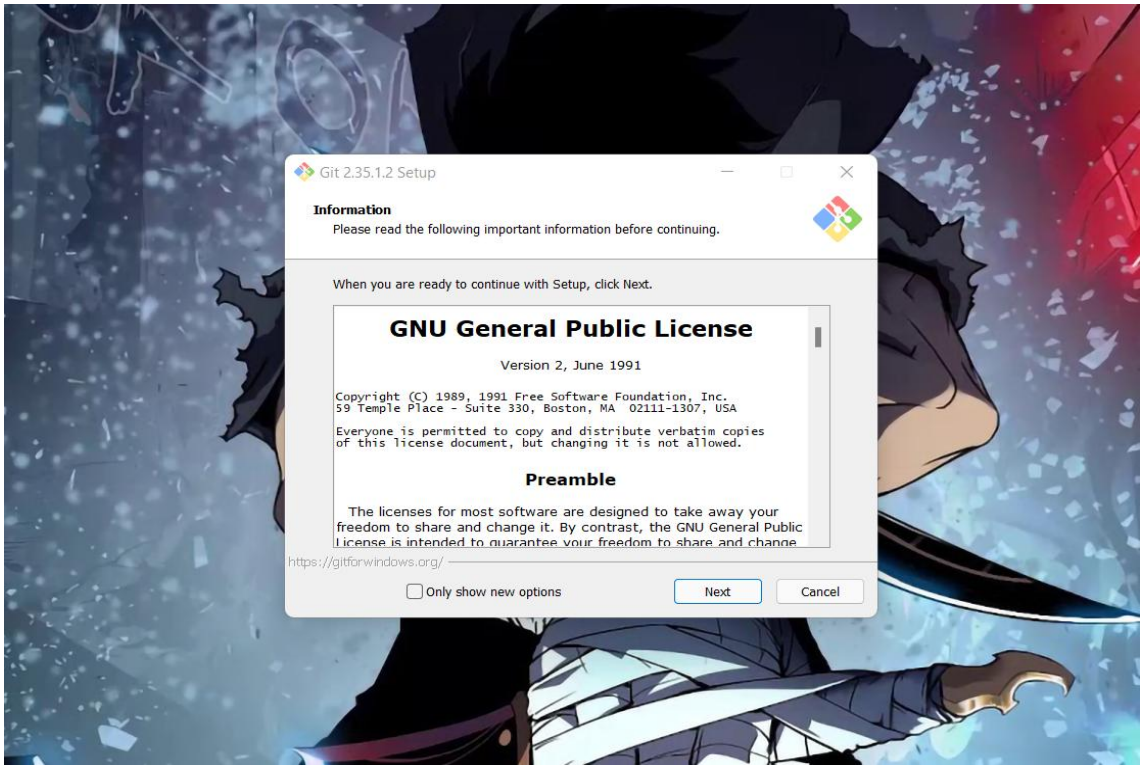


✧ Select the CPU for your system now. (Most of the system now runs on 64-bit processors.) Your download will begin when you pick a processor.

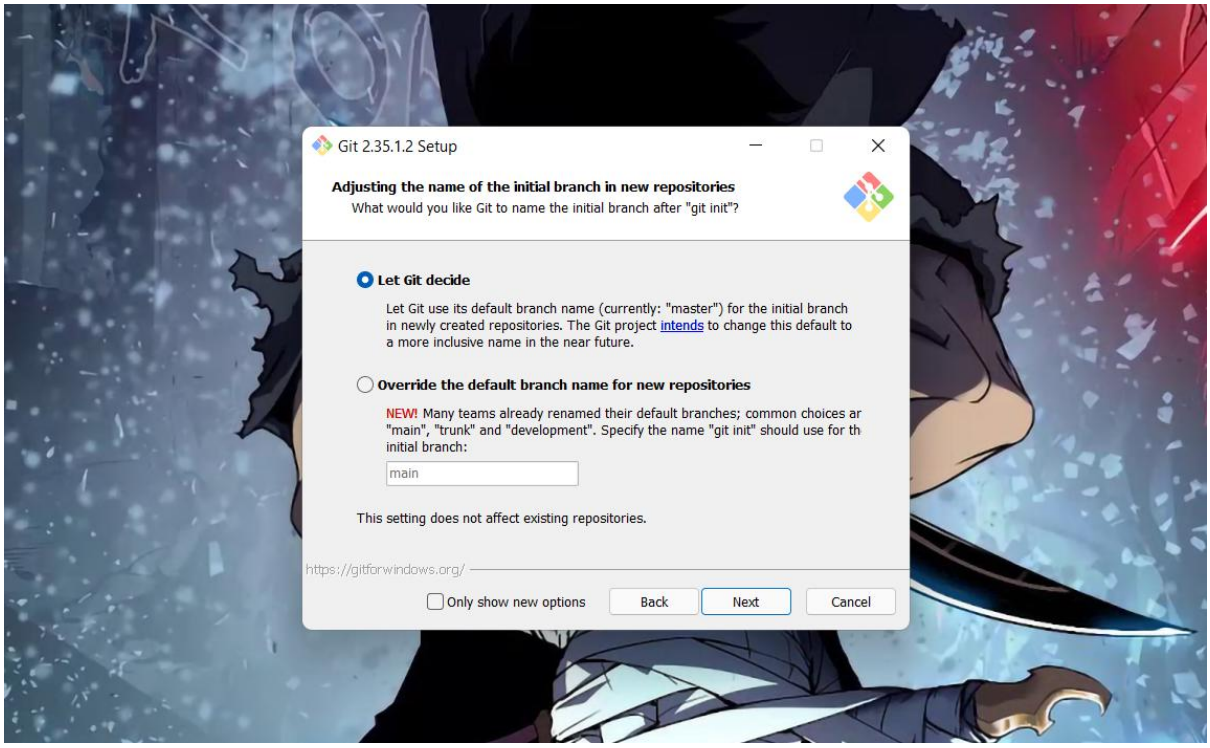
✧ You must now open the Git folder.

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

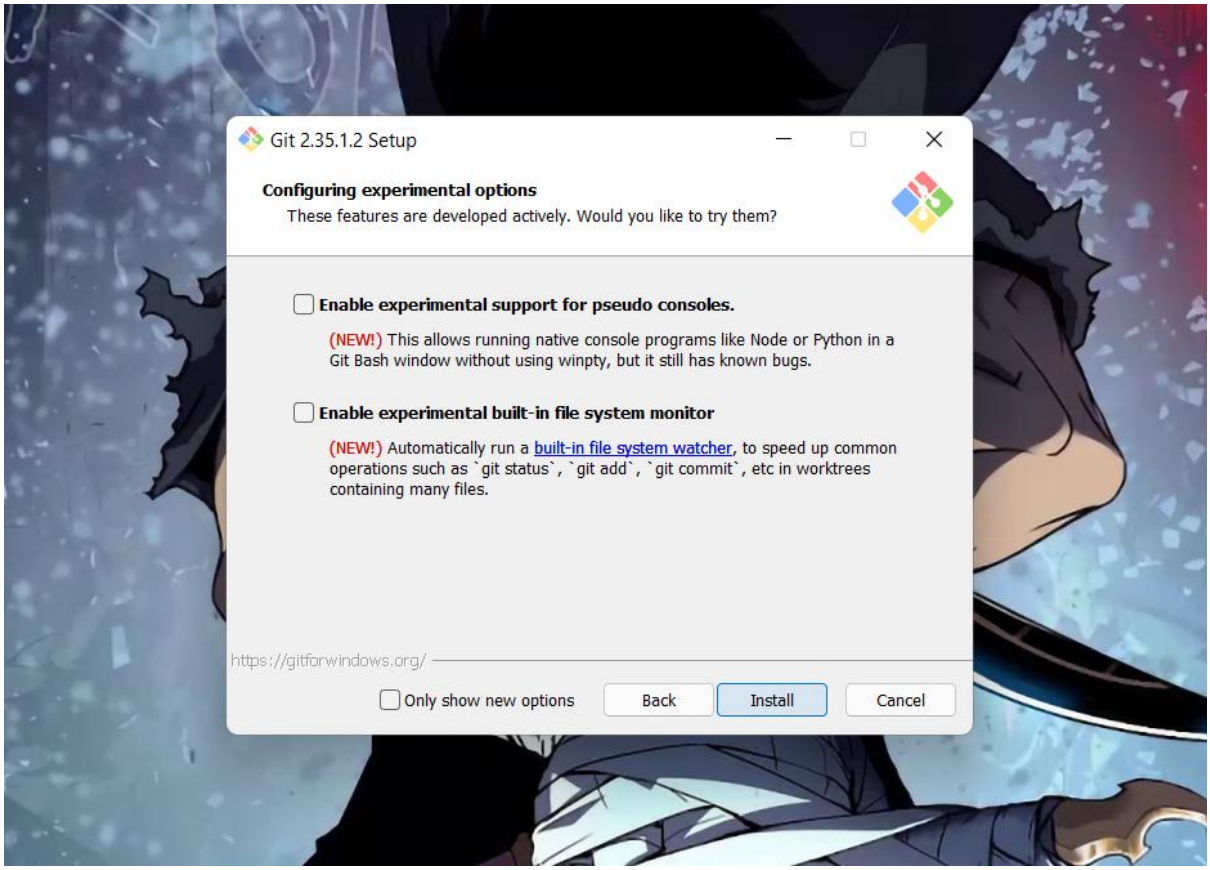
✧ YES should be selected.



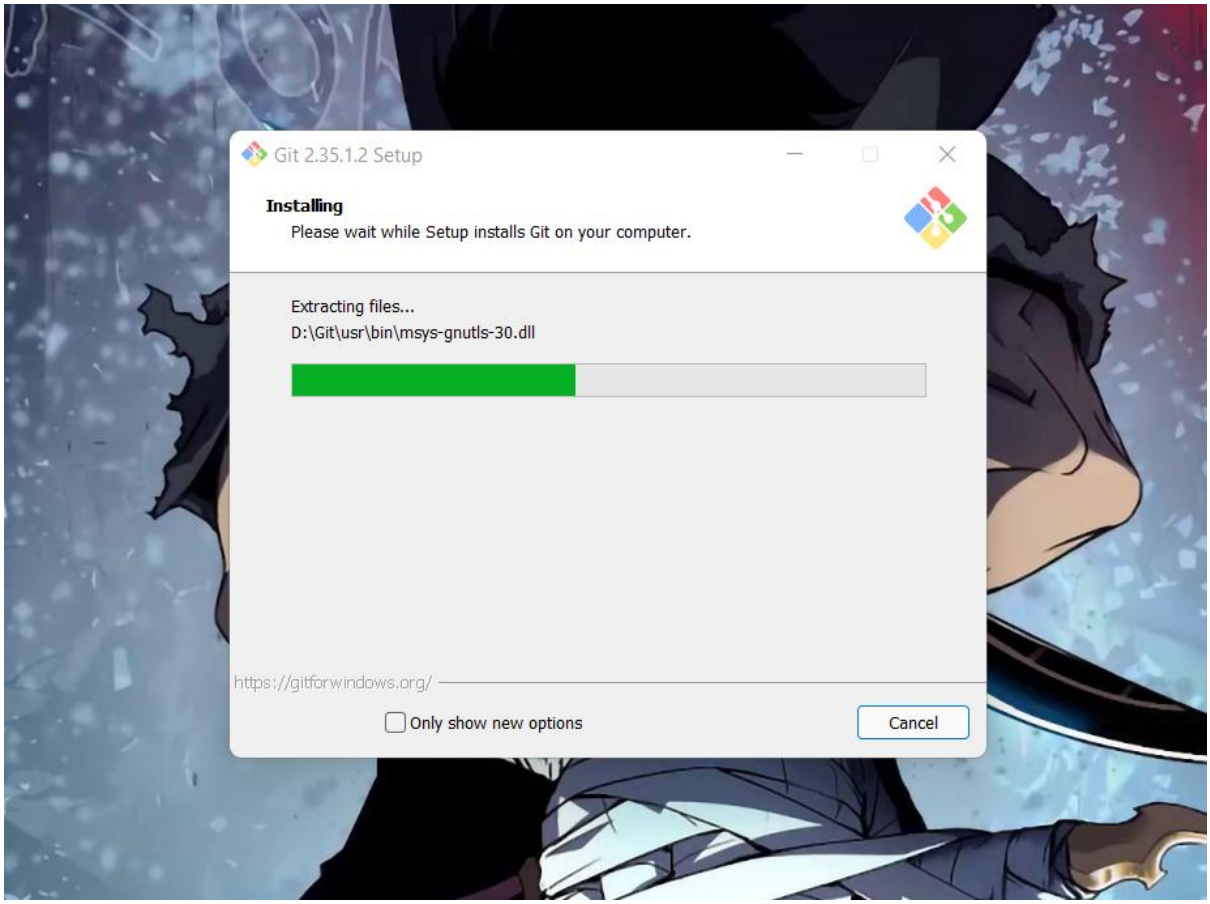
✧ Click on Next



✧ Continue clicking on next few times more



✧ Now select the Install option.



✧ Click on Finish after the installation is finished.

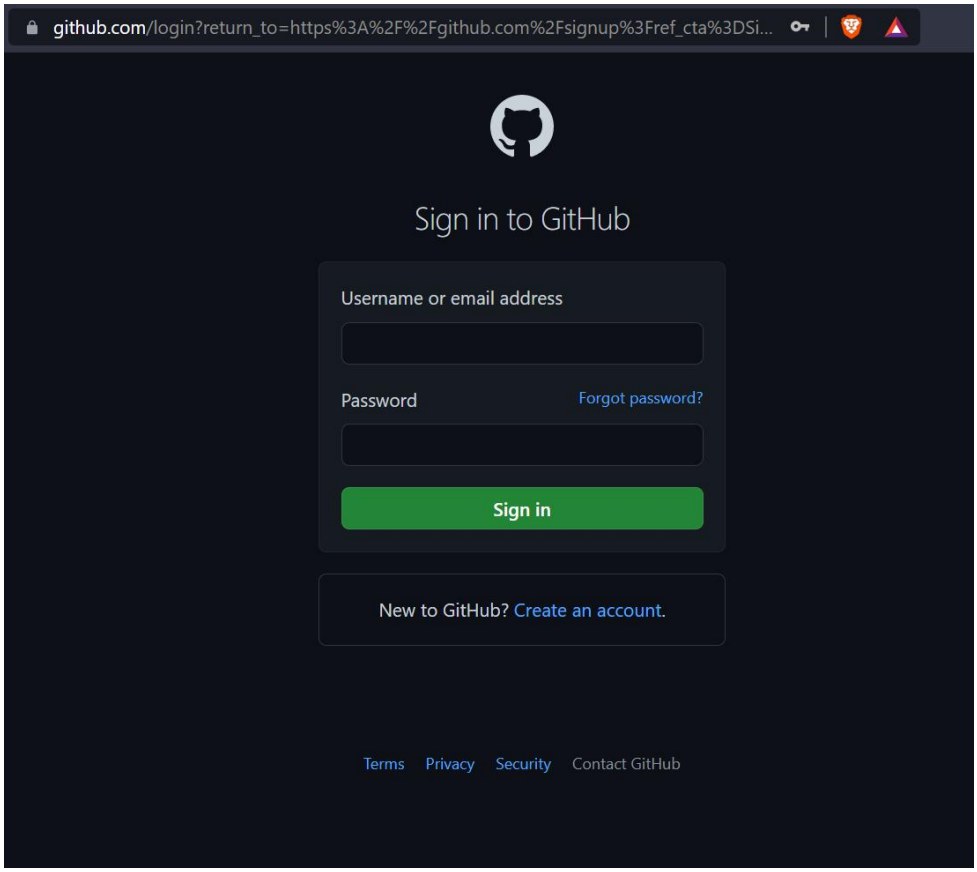
The installation of the git is finished and now we have to setup git client and GitHub account.

Experiment-2

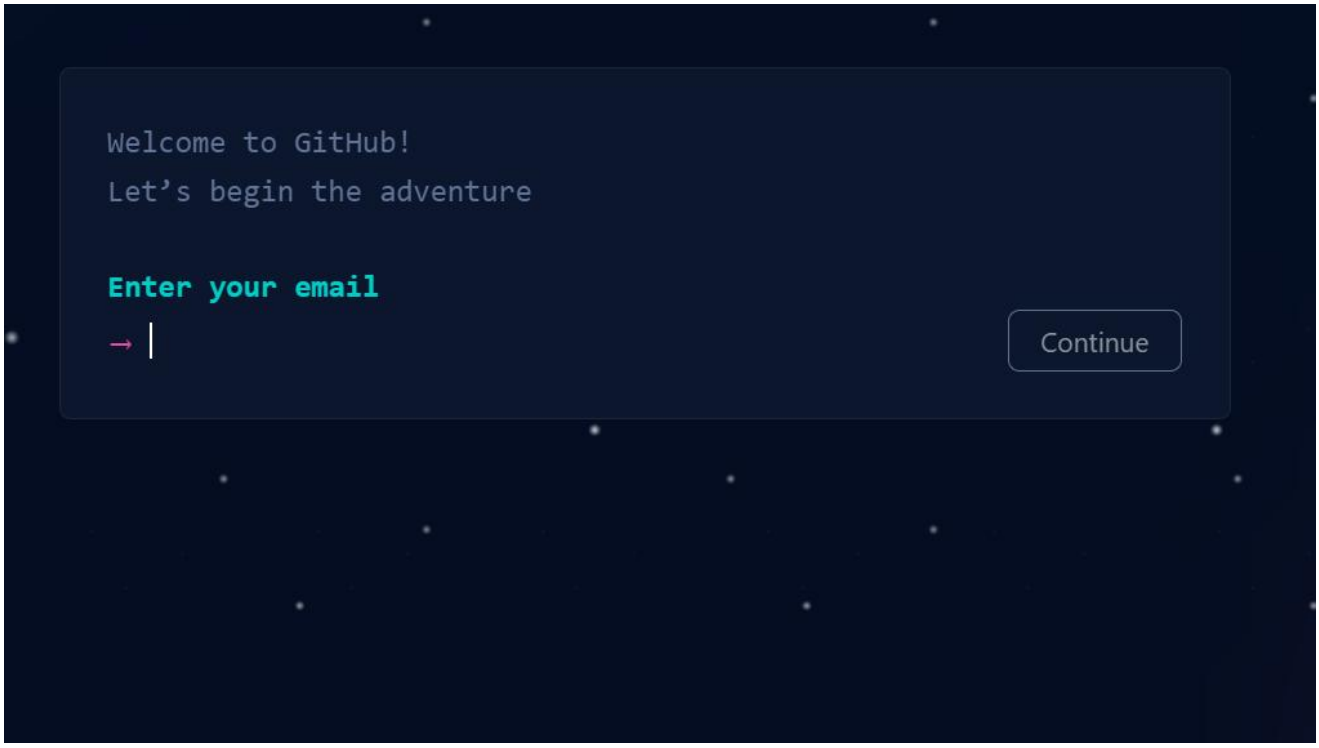
Aim: Setting up GitHub Account

- ✧ 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-3

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**.

```
asus@LAPTOP-84C5CAMB MINGW64 ~/Desktop/SCM projects
$ git init
Initialized empty Git repository in C:/Users/asus/Desktop/SCM projects/.git/

asus@LAPTOP-84C5CAMB MINGW64 ~/Desktop/SCM projects (master)
$ touch hello.txt
```

✧ 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 hello.txt Now type git status:

```
asus@LAPTOP-84C5CAMB MINGW64 ~/Desktop/SCM projects (master)
$ touch hello.txt

asus@LAPTOP-84C5CAMB MINGW64 ~/Desktop/SCM projects (master)
$ git status
On branch master

No commits yet

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

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

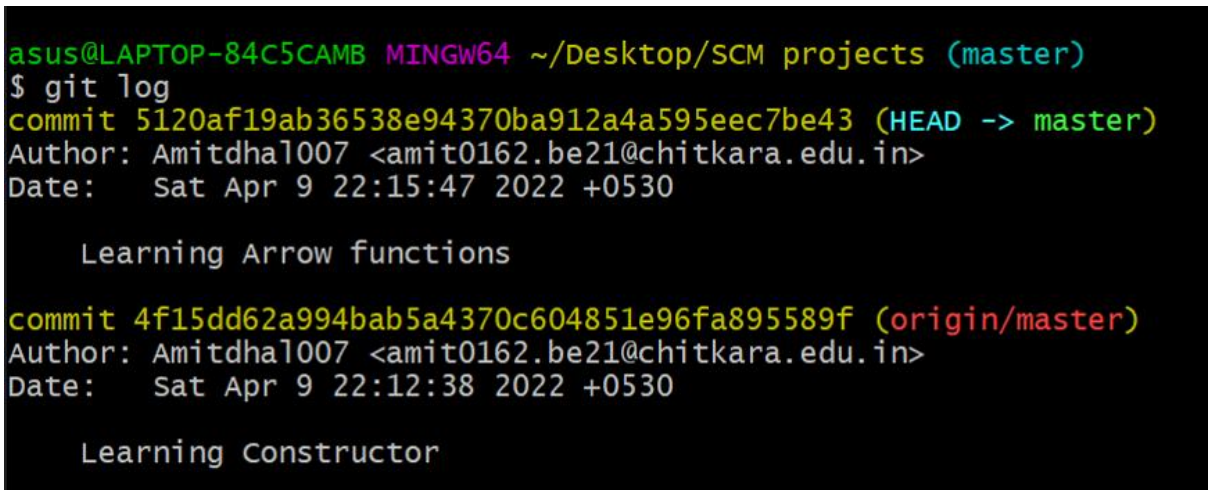
You can see that ***hello.txt*** is in red colour that means it is an untracked file.
Now firstly add the file in staging area and then commit the file.

For this, use command

git add . **[** For adding all the files in staging area. **]**
git commit -m "write any message" **[** For commit the file **]**

```
asus@LAPTOP-84C5CAMB MINGW64 ~/Desktop/SCM projects (master)
$ git add .

asus@LAPTOP-84C5CAMB MINGW64 ~/Desktop/SCM projects (master)
$ git status
On branch master

No commits yet

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


asus@LAPTOP-84C5CAMB MINGW64 ~/Desktop/SCM projects (master)
$ git commit -m "initial commit"
[master (root-commit) ffe6b8e] initial commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 hello.txt
```

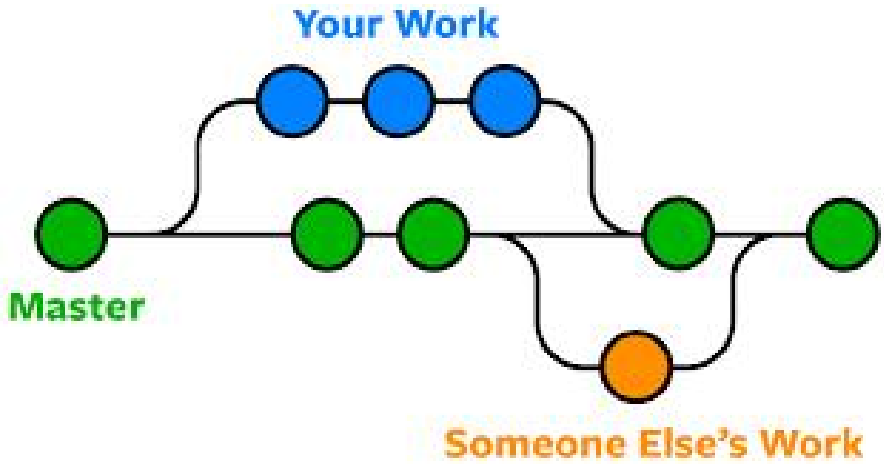
- **git log:** *The git log command displays a record of the commits in a Git repository. By default, the git log command displays a commit hash, the commit message, and other commit metadata.*


asus@LAPTOP-84C5CAMB MINGW64 ~/Desktop/SCM projects (master)
$ git log
commit 5120af19ab36538e94370ba912a4a595eec7be43 (HEAD -> master)
Author: Amitdhal007 <amit0162.be21@chitkara.edu.in>
Date: Sat Apr 9 22:15:47 2022 +0530
Learning Arrow functions
commit 4f15dd62a994bab5a4370c604851e96fa895589f (origin/master)
Author: Amitdhal007 <amit0162.be21@chitkara.edu.in>
Date: Sat Apr 9 22:12:38 2022 +0530
Learning Constructor

Experiment-4

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**]**

```
asus@LAPTOP-84C5CAMB MINGW64 ~/Desktop/SCM projects (master)
$ git branch
* master

asus@LAPTOP-84C5CAMB MINGW64 ~/Desktop/SCM projects (master)
$ git branch activity1

asus@LAPTOP-84C5CAMB MINGW64 ~/Desktop/SCM projects (master)
$ git branch activity2

asus@LAPTOP-84C5CAMB MINGW64 ~/Desktop/SCM projects (master)
$ git branch
  activity1
  activity2
* master

asus@LAPTOP-84C5CAMB MINGW64 ~/Desktop/SCM projects (master)
$ git checkout activity1
Switched to branch 'activity1'

asus@LAPTOP-84C5CAMB MINGW64 ~/Desktop/SCM projects (activity1)
$ git branch
* activity1
  activity2
  master

asus@LAPTOP-84C5CAMB MINGW64 ~/Desktop/SCM projects (activity1)
$
```

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 act1’, it shows 2 branches but the green colour and star is on master. So, we have to switch to act1 by using ‘git checkout act1’. If we use ‘git branch’, now you can see that the green colour and star is on act1. It means you are in activity1 branch and all the data of master branch is also on act1 branch. Use “ls” to see the files.

Now add a new file in activity1 branch, do some changes in file and commit the file.

```
asus@LAPTOP-84C5CAMB MINGW64 ~/Desktop/SCM projects (activity1)
$ touch world.txt

asus@LAPTOP-84C5CAMB MINGW64 ~/Desktop/SCM projects (activity1)
$ git status
On branch activity1
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        world.txt

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

asus@LAPTOP-84C5CAMB MINGW64 ~/Desktop/SCM projects (activity1)
$ git add .

asus@LAPTOP-84C5CAMB MINGW64 ~/Desktop/SCM projects (activity1)
$ git commit -m "Added file in new branch"
[activity1 4bd44ac] Added file in new branch
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 world.txt

asus@LAPTOP-84C5CAMB MINGW64 ~/Desktop/SCM projects (activity1)
$ ls
 Function.html                          NoArgument.CPP       functionOverloading.CPP   jquery.html
'HAVING_ARGUMENT_WITH RETURN VALUE.CPP'  'Practicing C++'/    functionPrototype.cpp     loops.cpp
 HavingArgumennt.CPP                    Regular_Arrow.html   ifElse.cpp                world.txt
 NOARGUMENTNOETURN.CPP                  constructor.html     inlinefunction.CPP
```

If we switched to master branch, ‘contact.html’ file is not there. But he file is in activity1 branch.

```
asus@LAPTOP-84C5CAMB MINGW64 ~/Desktop/SCM projects (activity1)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

asus@LAPTOP-84C5CAMB MINGW64 ~/Desktop/SCM projects (master)
$ ls
 Function.html                          NoArgument.CPP       functionOverloading.CPP   jquery.html
'HAVING_ARGUMENT_WITH RETURN VALUE.CPP'  'Practicing C++'/    functionPrototype.cpp     loops.cpp
 HavingArgumennt.CPP                    Regular_Arrow.html   ifElse.cpp
 NOARGUMENTNOETURN.CPP                  constructor.html     inlinefunction.CPP

asus@LAPTOP-84C5CAMB MINGW64 ~/Desktop/SCM projects (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 branchname [use to merge branch**]**

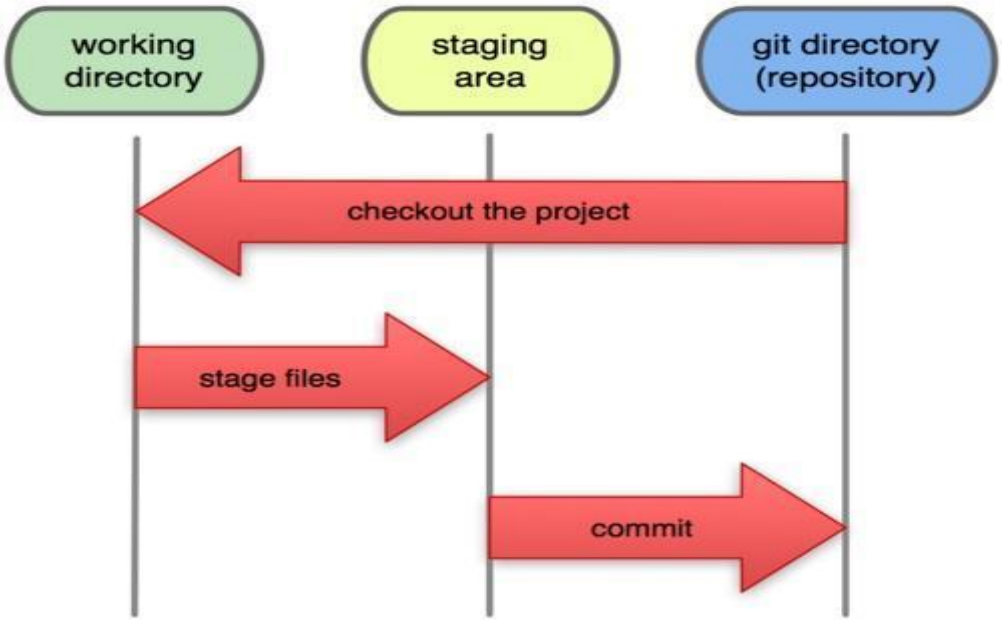
```
asus@LAPTOP-84C5CAMB MINGW64 ~/Desktop/SCM projects (master)
$ git merge activity1
Updating 9e493fa..4bd44ac
Fast-forward
 world.txt | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 world.txt
```

Experiment-5

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.

Experiment No. 06

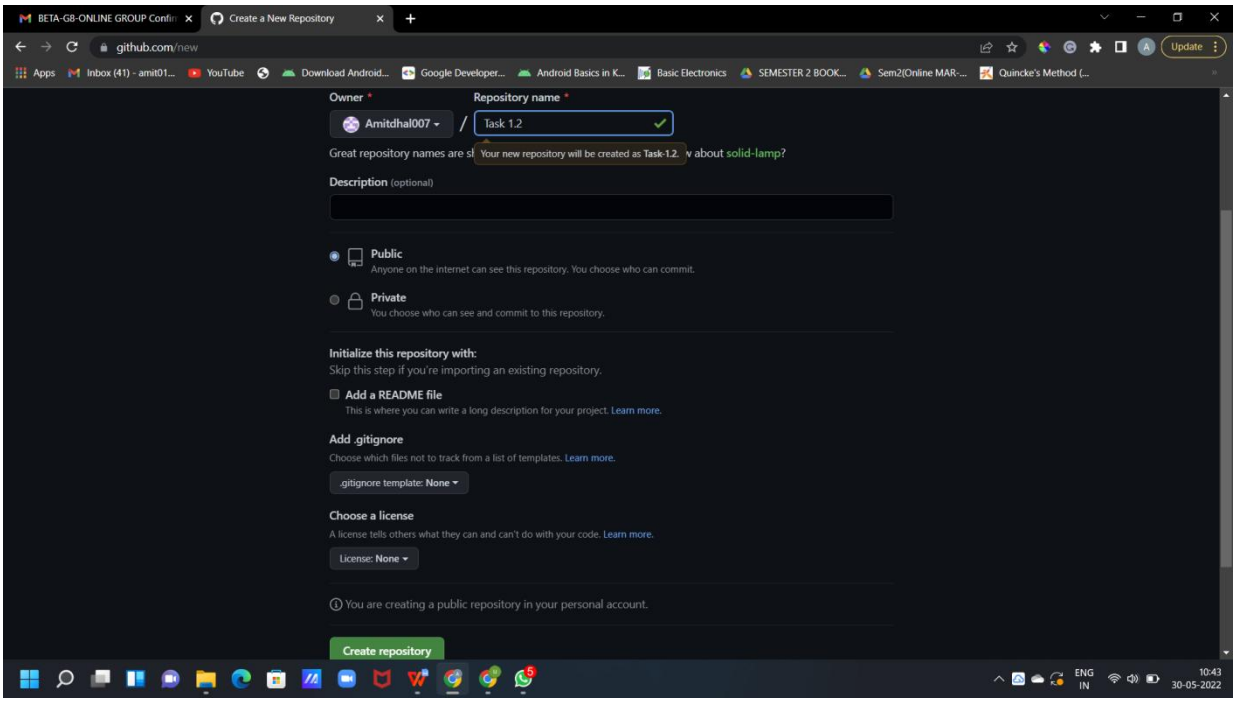
Aim: Add collaborators on Github Repo

Theory:

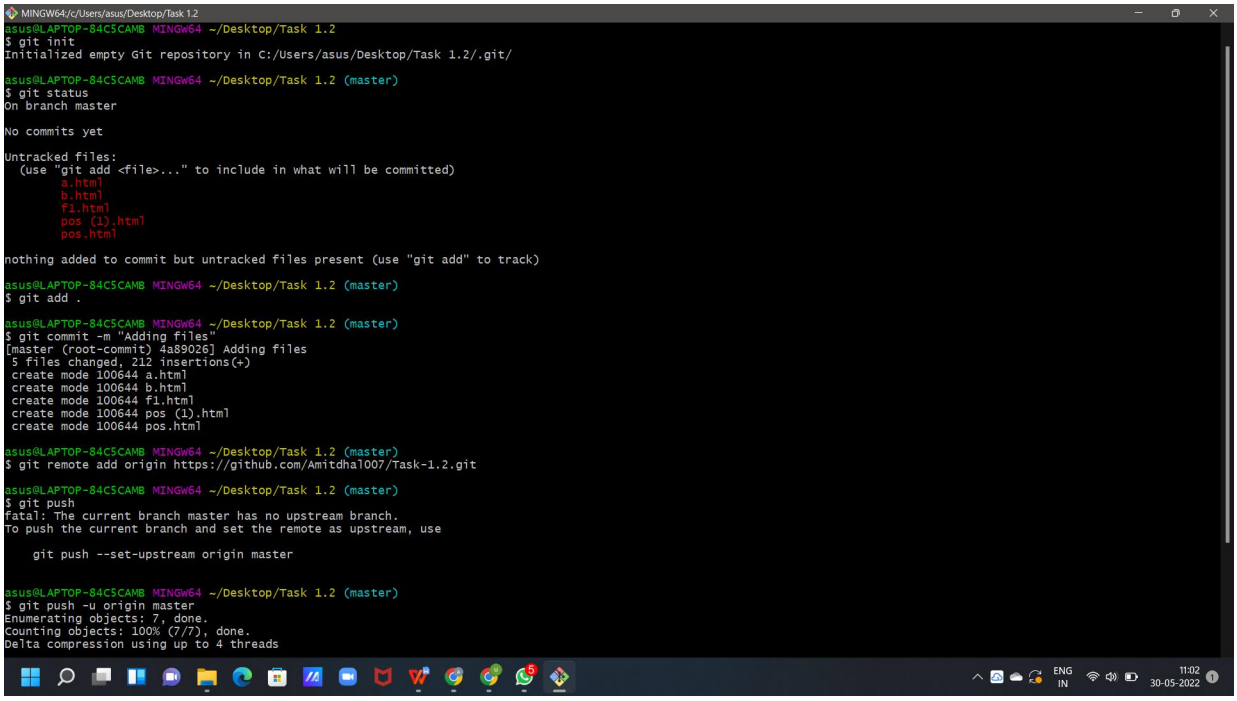
- ## Create a New Repository.

  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 inthe repository. For more information, see "About repositories."

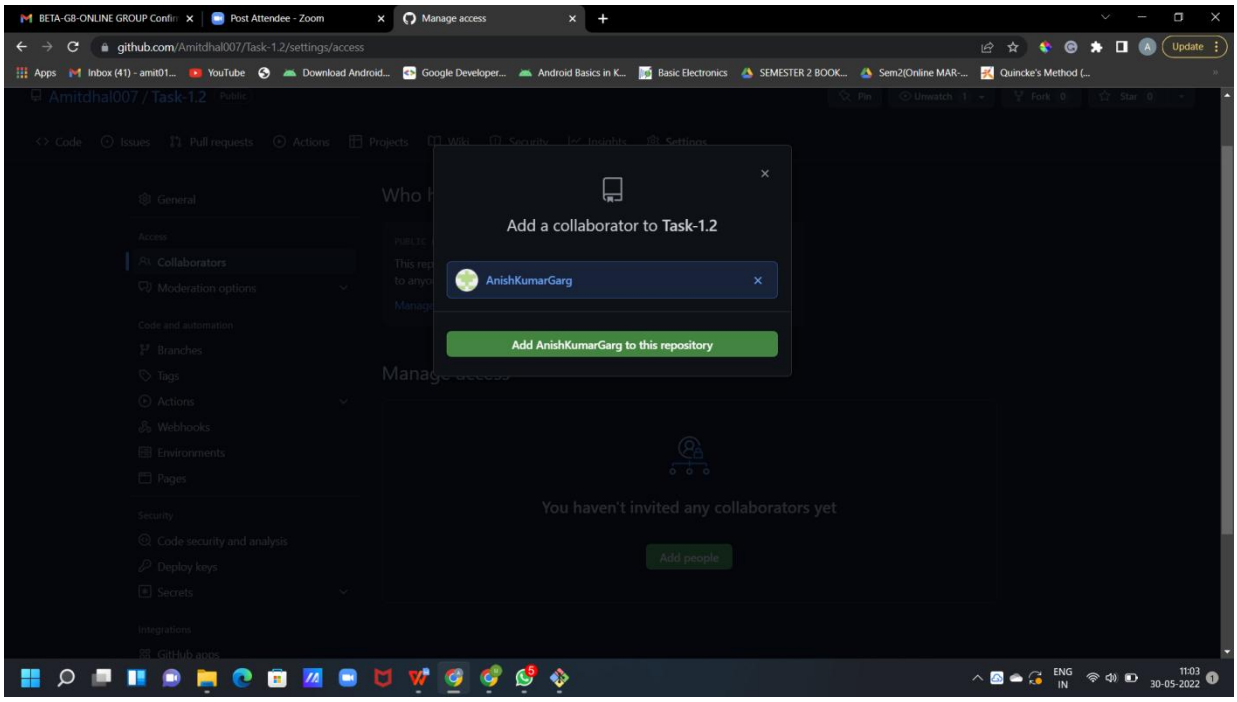
  When you create a new repository, you should initialize the repository with a README file to let people know about your project. For more information, see "Creating a new repository."



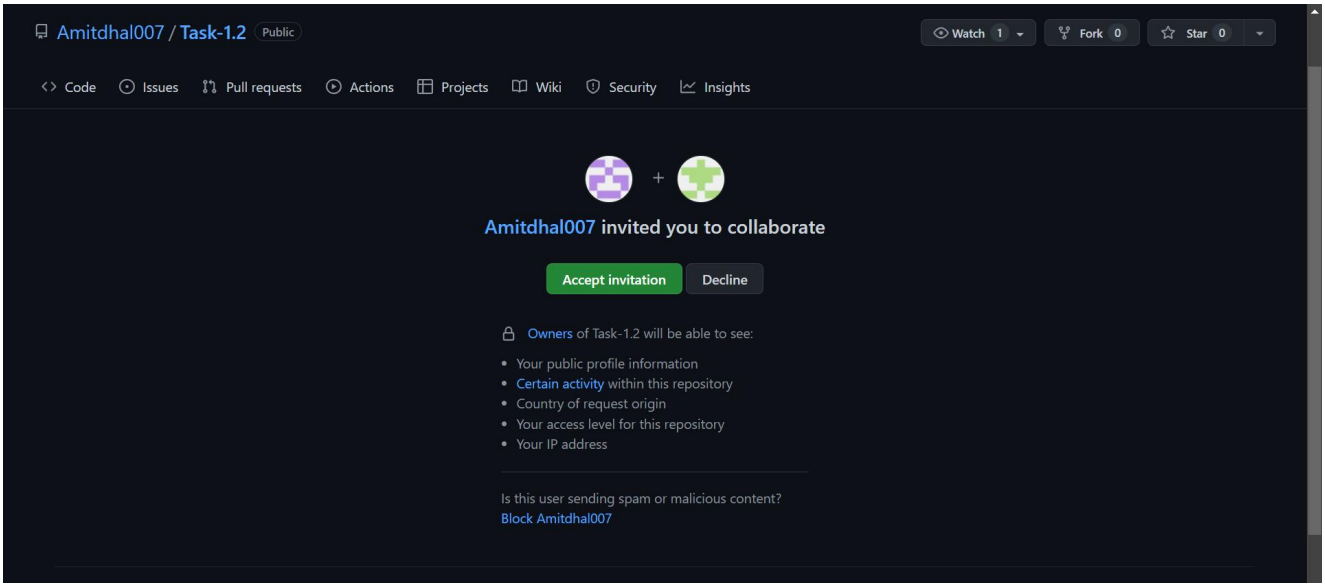
- Now Copy the HTTP link of your repo and paste it it on your ‘Git CLI’, and mergethe local repo in remote repo .

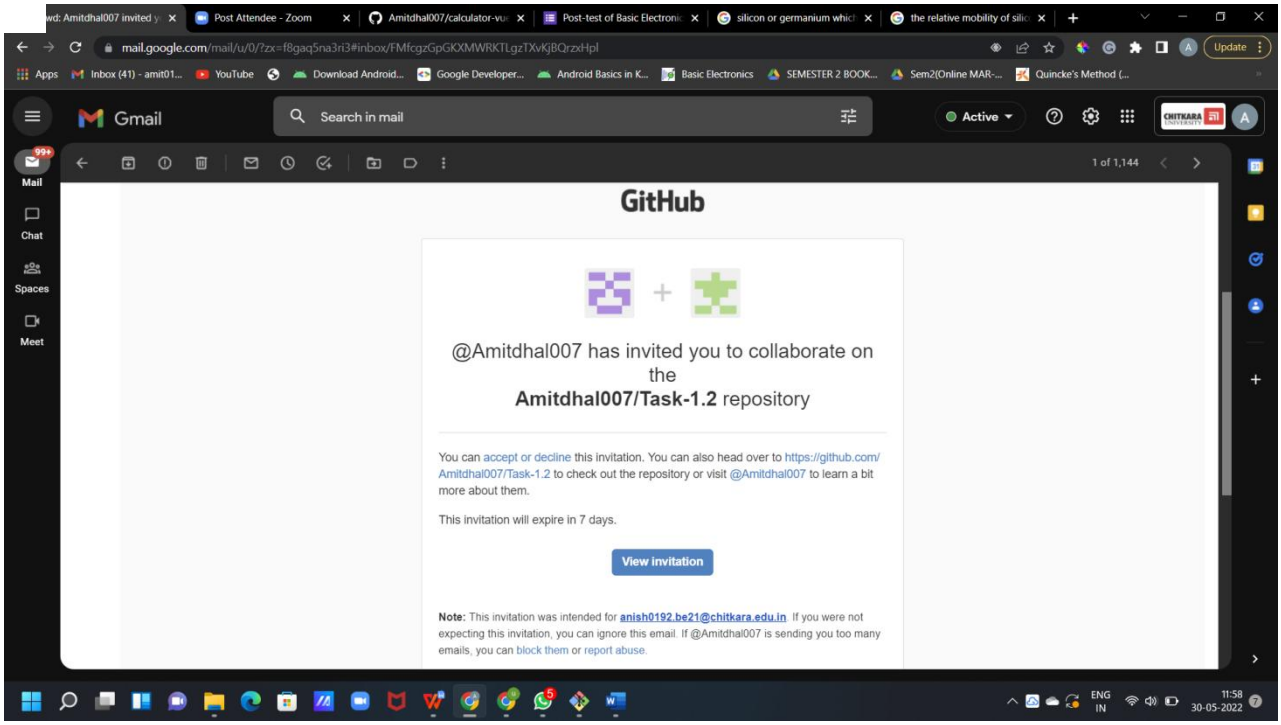


- Go to Collaborators in Repo Setting, Add the username or email of Collaborator you want to add in your Repo.

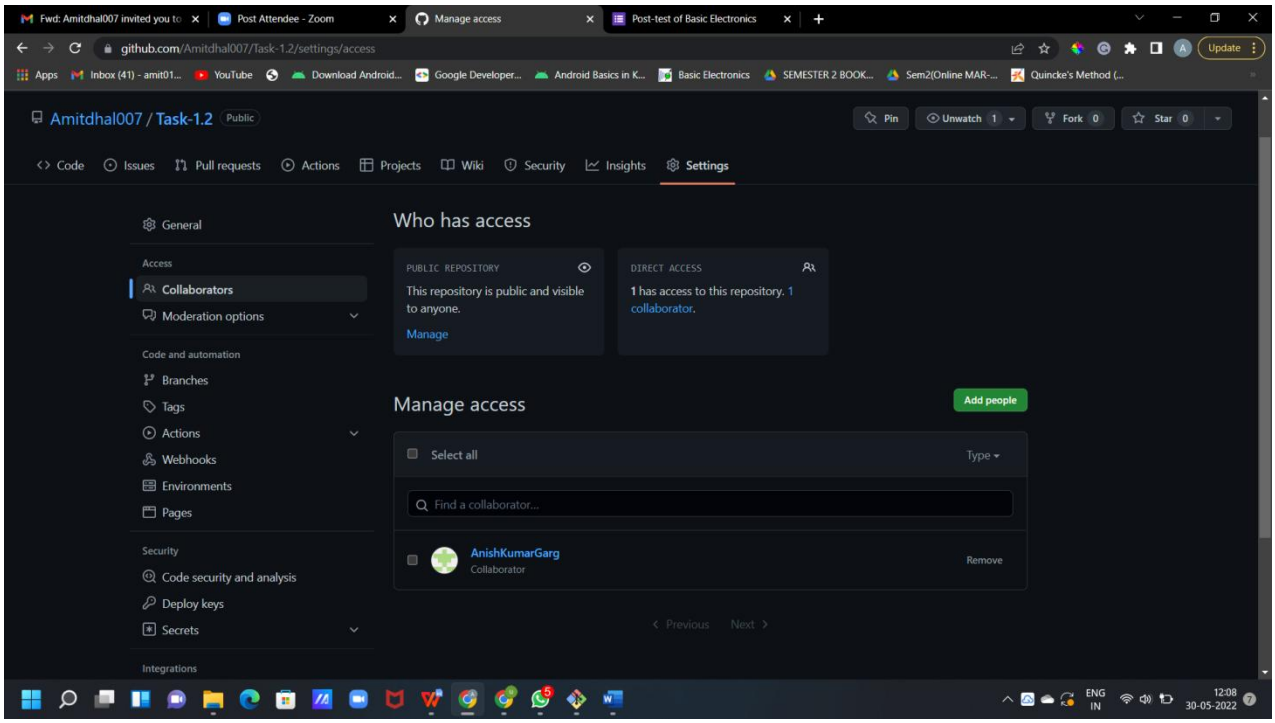


➢ Invitation Mail is sent to the Collaborator, the collaborator has to accept thisInvitation.





New Collaborator has now access to the SCM Repo.



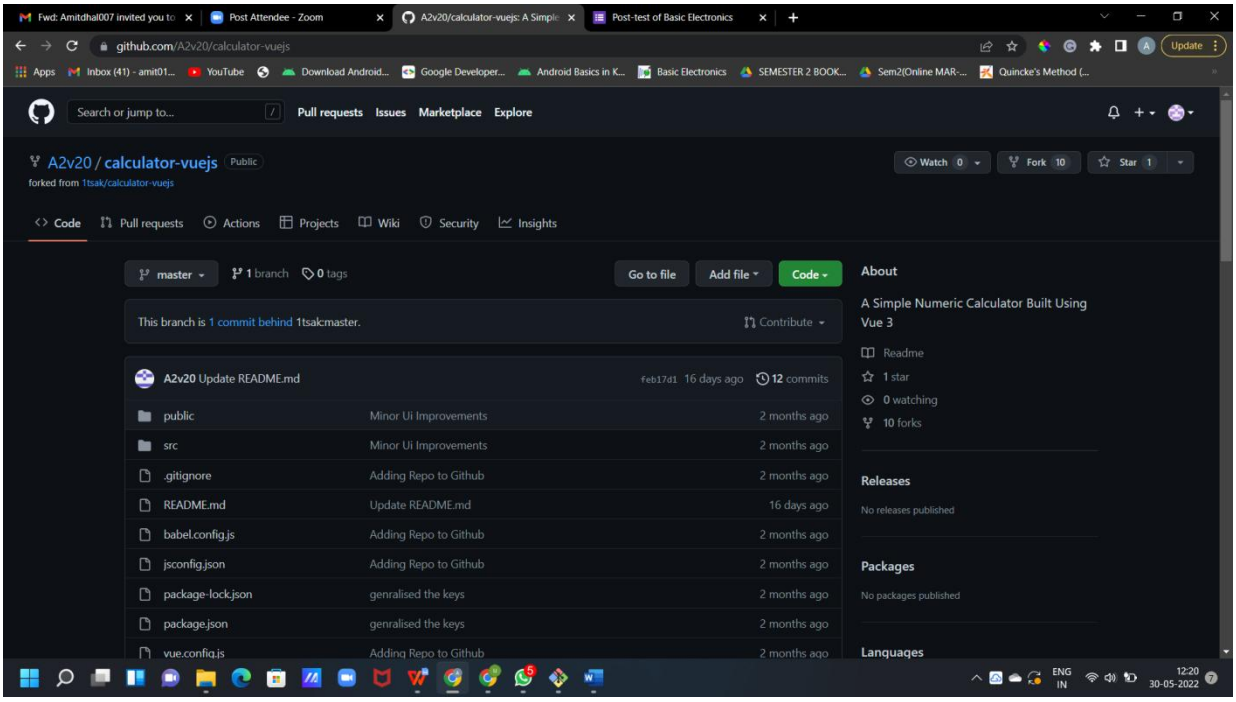
Experiment No. 07

Aim: Fork and Commit

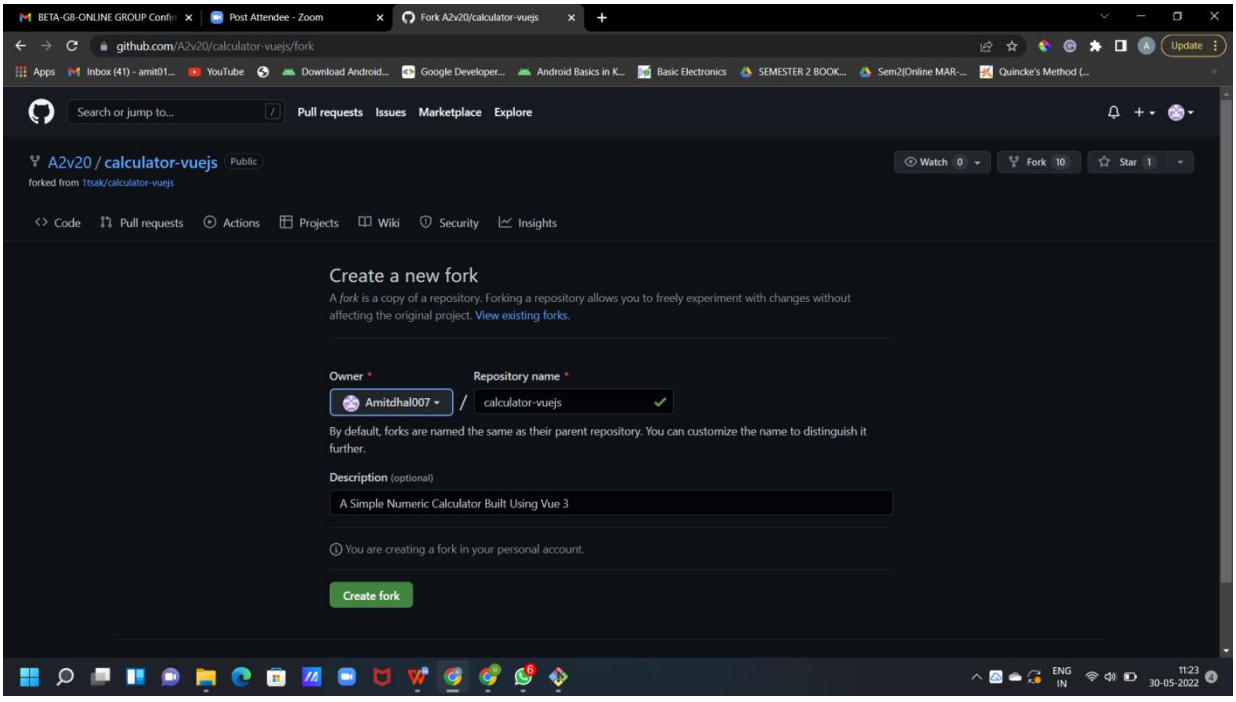
Theory:

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

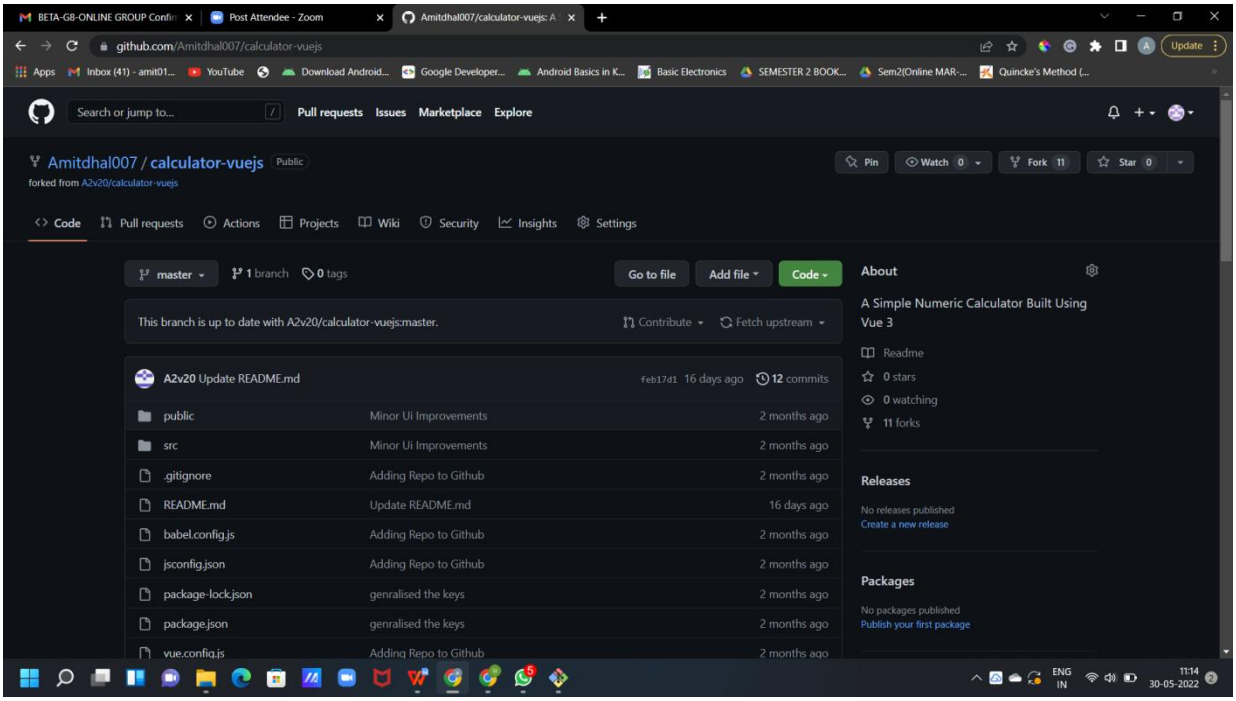
To fork a repository first thing you need to do is, sign in to your GitHub account and then you came to the repository you want to fork.



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.

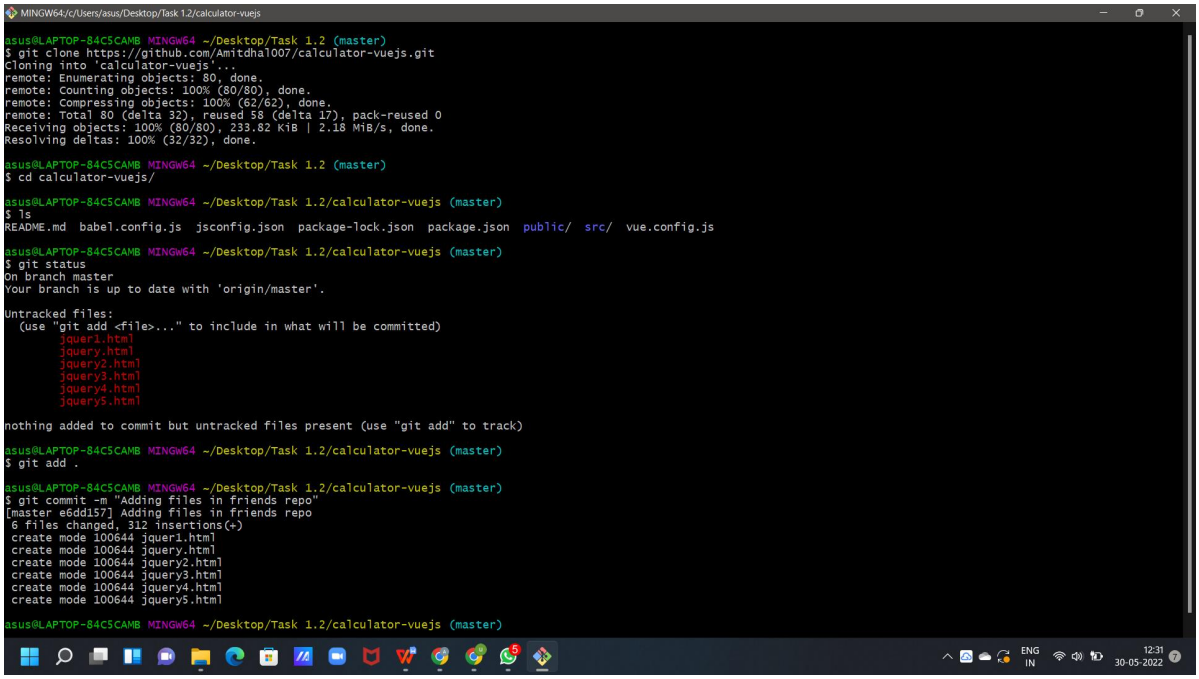


Now you will have a copy of the repo you have forked from other user. Now you can do any modification you want without making changes to main source code.

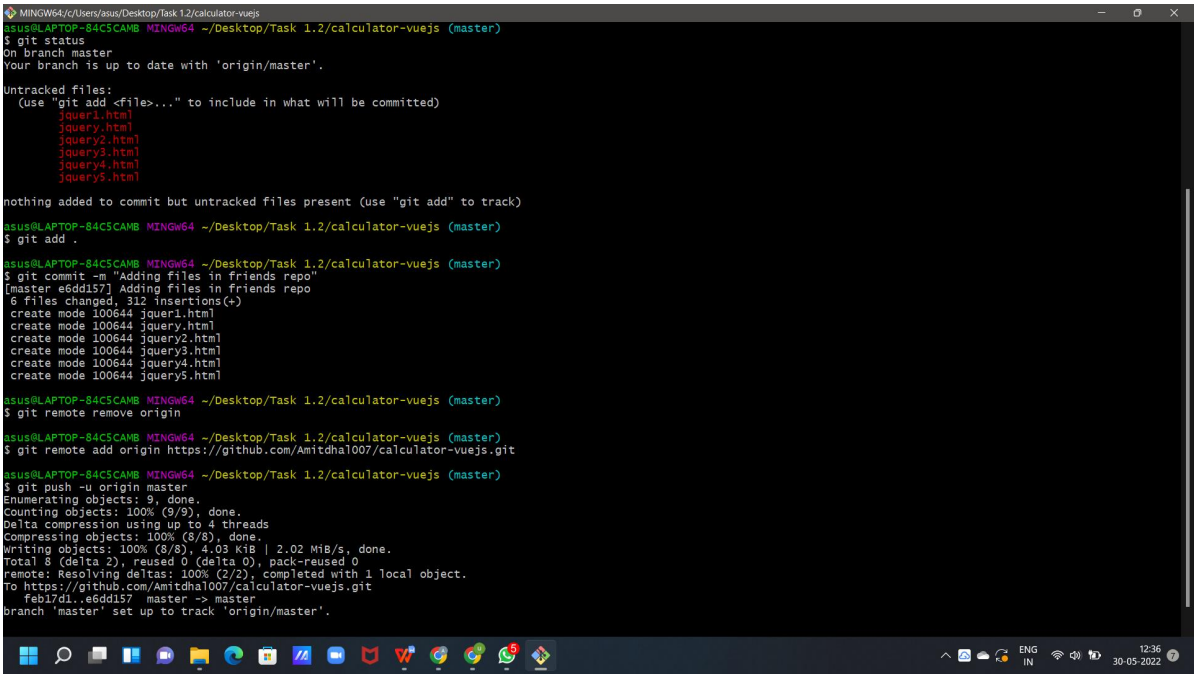


Now type https://github.com/Amitdhal007/calculator-vuejs on CLI.

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



Now Open the file make changes or add some files in it and commit it and push it to remote.

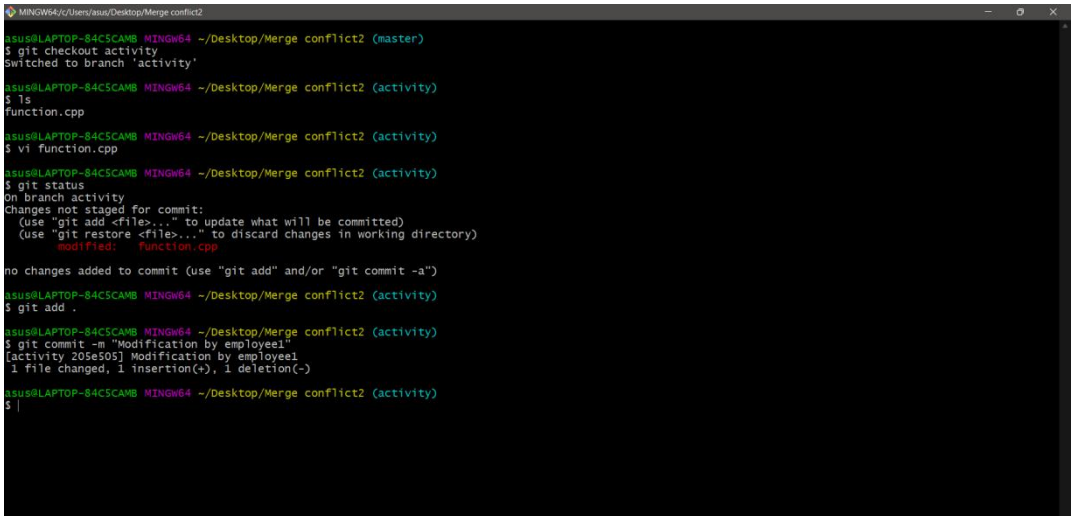


Experiment No. 08

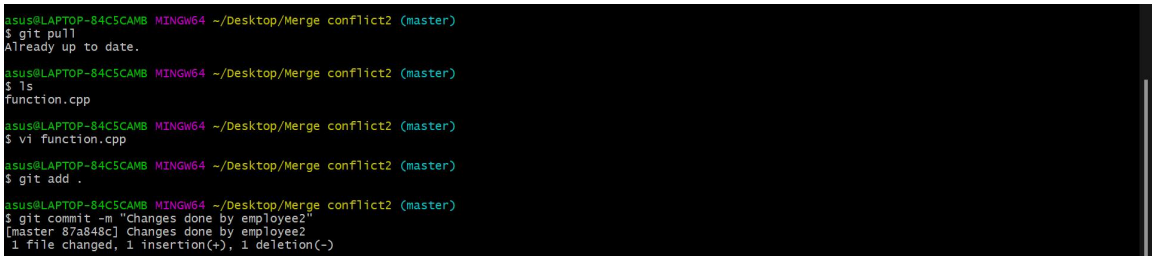
Aim: Merge and Resolve ConflictsTheory:

IV. Do changes in new branch and commit those change. And checkout to master branch and again do changes and commit it. Now in master branch merge the another branch.

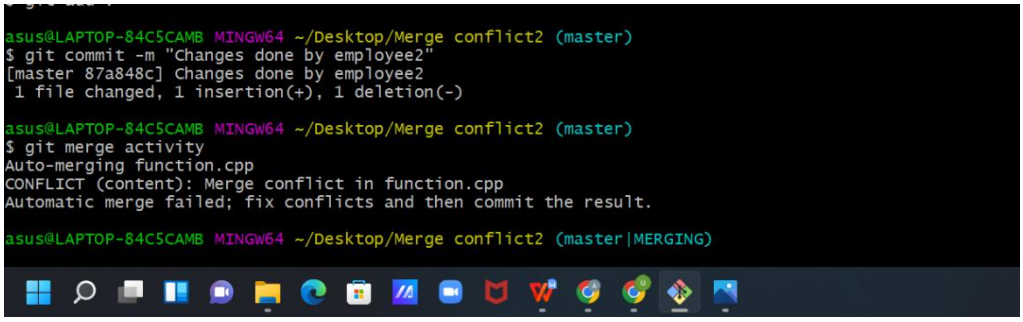
COMMIT IN NEW BRANCH



COMMIT IN MASTER BRANCH

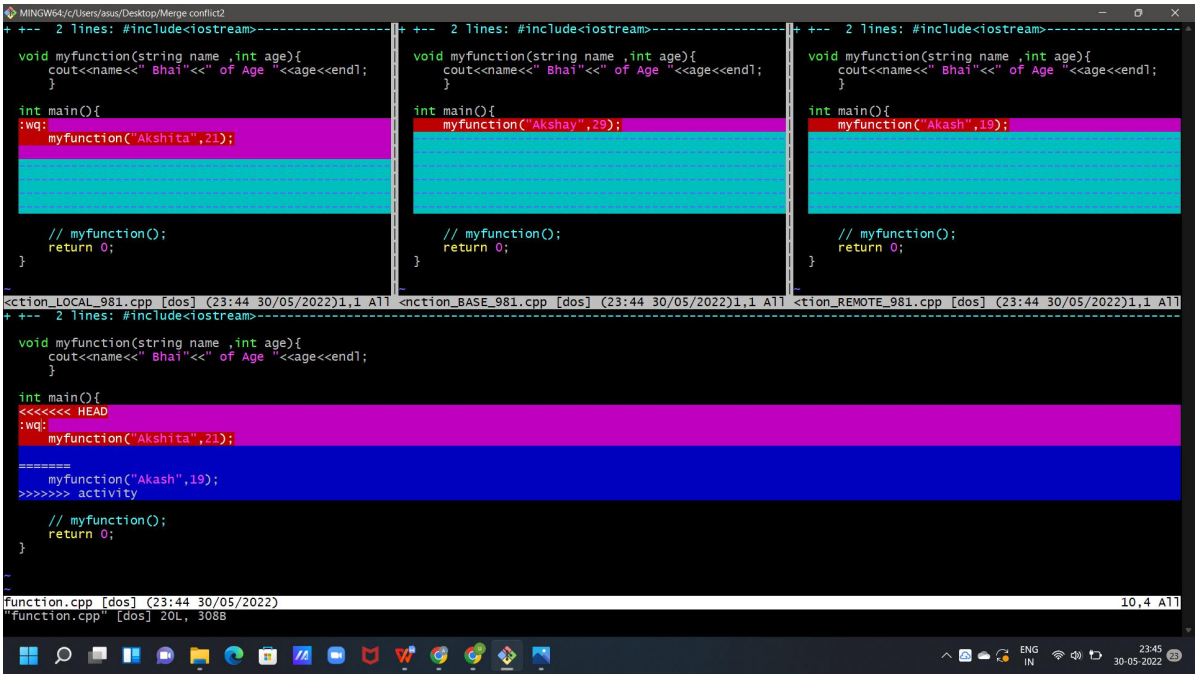


V. Now try to merge it will give Conflicts Error.

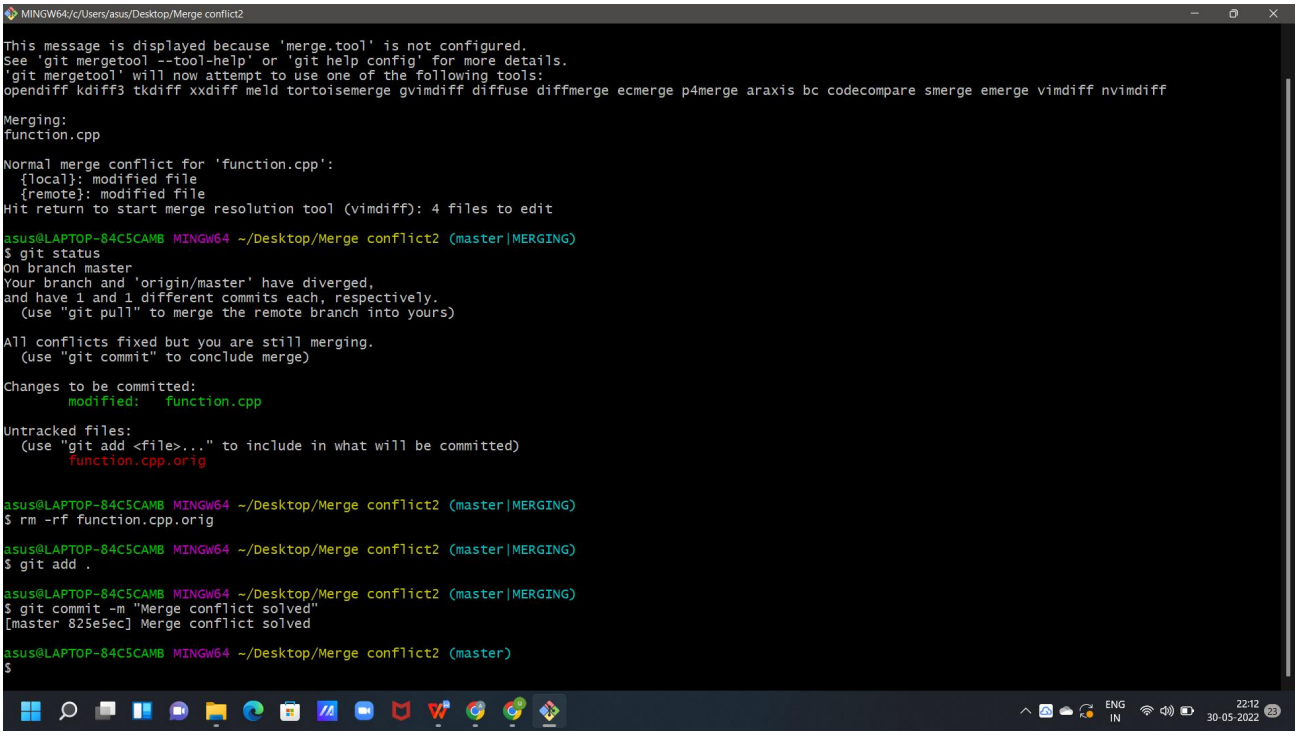


VI. Use Command “git mergetool” to solve the conflict.

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



VII. 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

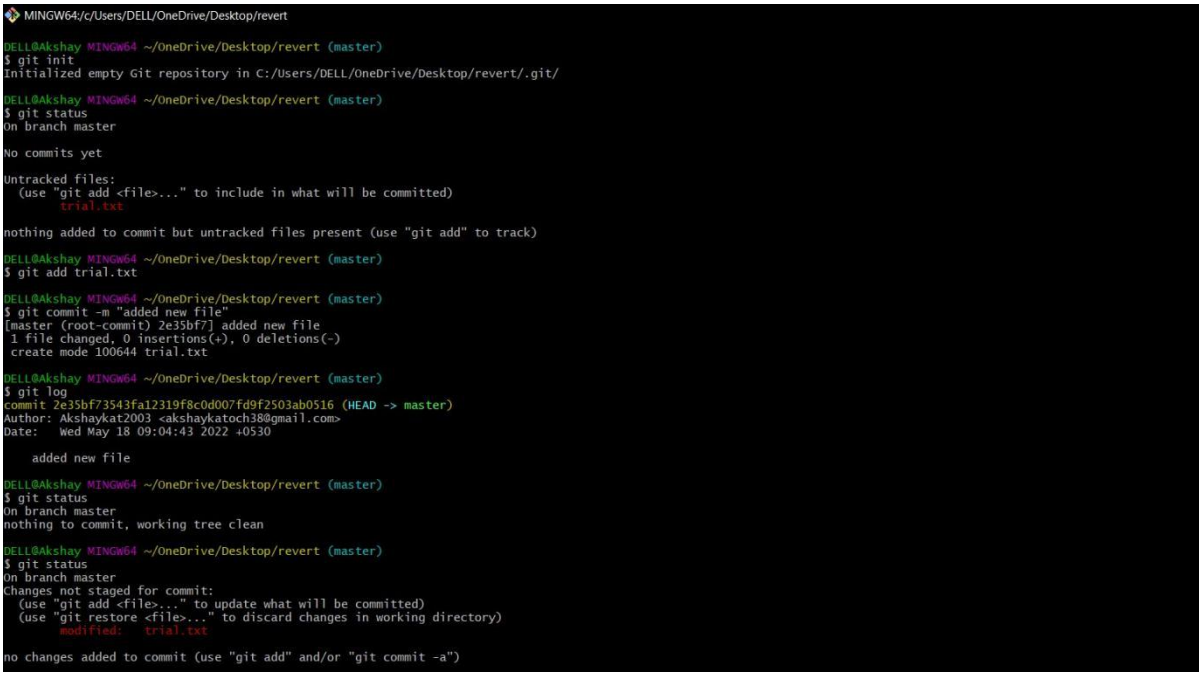
Aim: Reset and Revert

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 usedto manipulate commits or individual files. These commands can be confusing so it'simportant 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.



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 workingarea and to see them we will see those using ***git status***.

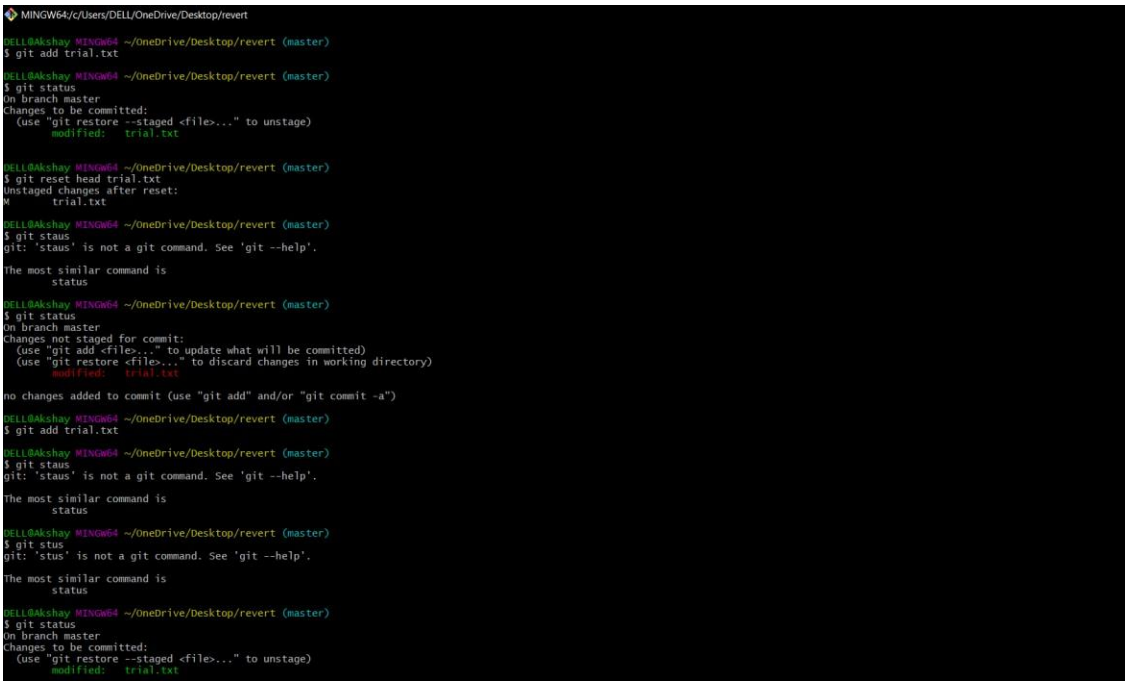
Now we have a change ***Hello World*** which is untracked in our working repository and weneed to discard this change. So, the command that we should use here is -

✧ 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 textdocument 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 -

✧ git reset

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

git reset HEAD <filename>

CS18

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 wewant to remove them from our local repository. To see how to remove the commit fromour local repository let's stage and commit the changes that we just did and then removethat commit.

We have 2 commits now, with the latest being the Added Hello World commit which weare 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 latestcommit that we have done.
- ✧ We cannot remove a specific commit with the help of git reset , for ex : we cannotsay 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 ourfile 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 thatwe 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 destructivecommand as we would not be able to get back the changes so be careful while using this command.

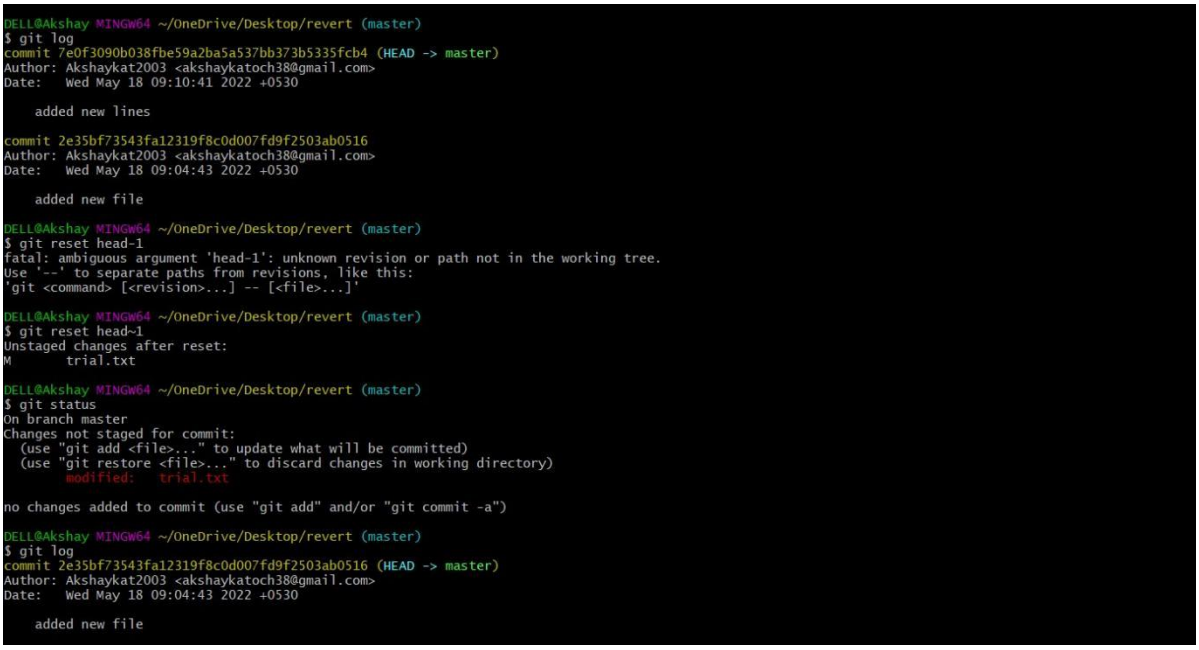
Points to keep in mind while using git reset command -

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 inremote directory.
We cannot remove a specific commit with the help of git reset , for ex : we cannotsay 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 theremote repository, then how do we remove the unwanted commits from the remote repository The command that we use here is -

✧ 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 thatthe 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 justremove "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'slook at the example –

Note: If you see the lines as we got after the git revert command, just visit your default texteditor 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 thechange 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 remoterepository. Let's summarize the points that we saw above -

Difference Table

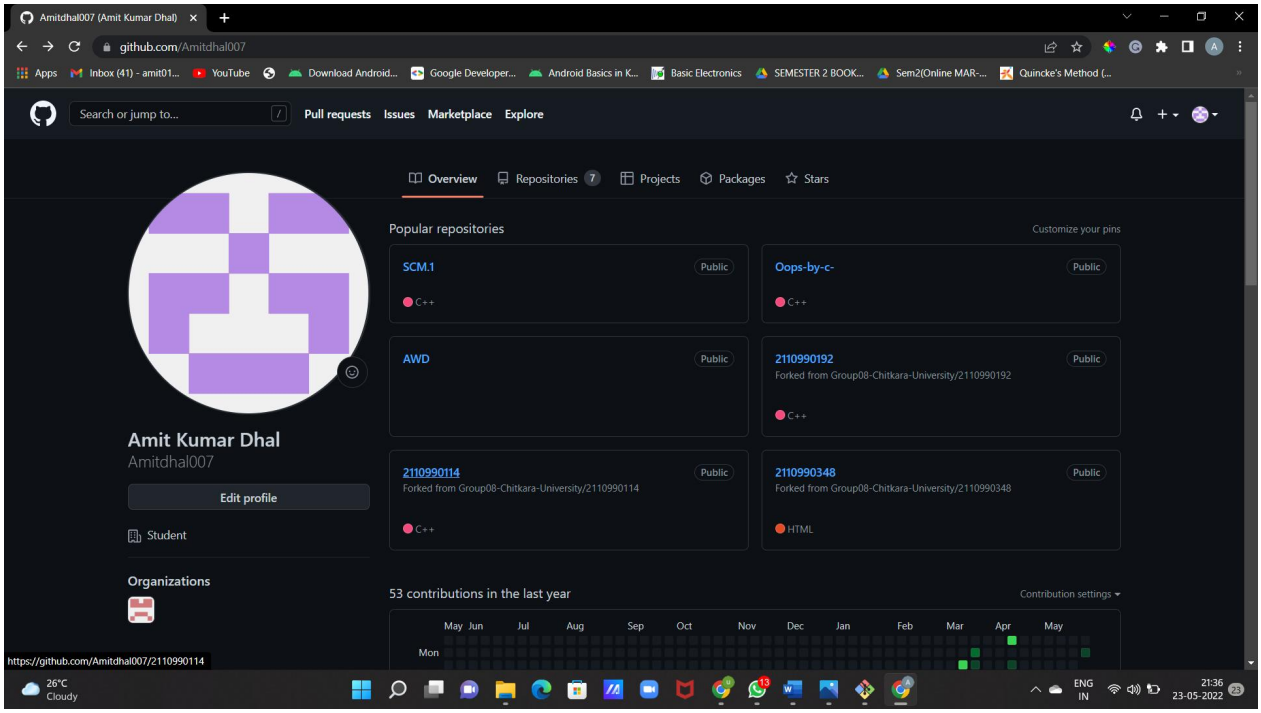
git checkout	git reset	git revert
Discards the changes in the working repository.	Unstages a file and bring our changes back to the working directory	Removes the commits fron the remote repository.
Used in the local repository.	Used in local repository	Used in the remote repository
Does not make any 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.

TASK-2

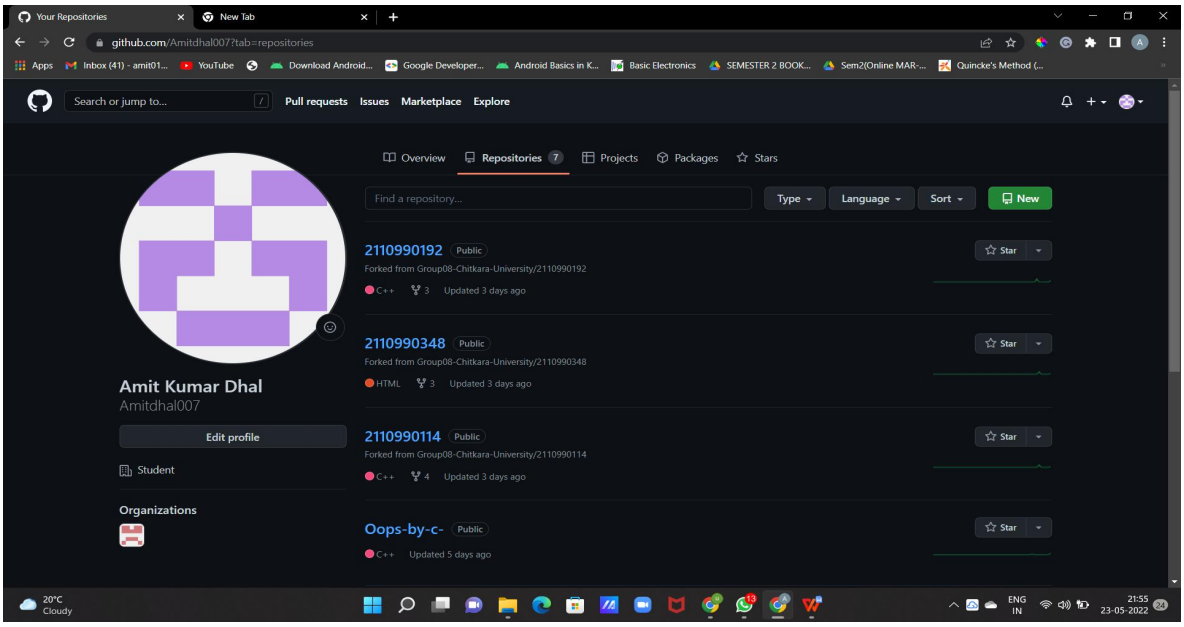
Experiment No. 01

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

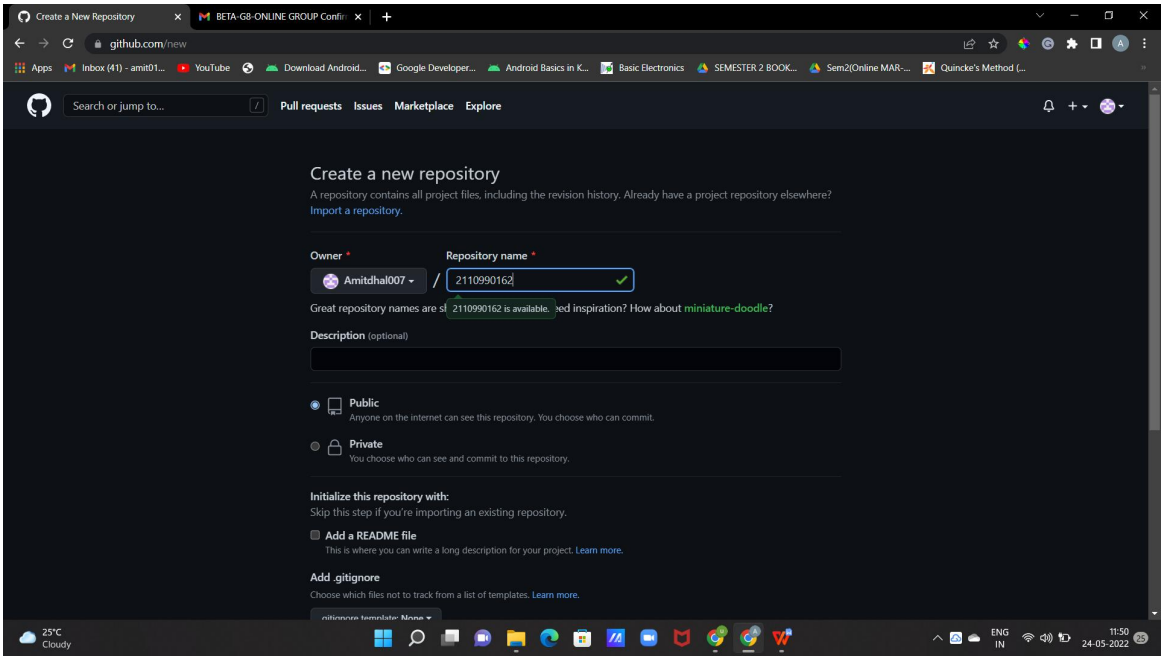
- Login to your Github account and you will land on the homepage as shown below. Click on Repositories option in the menu bar.



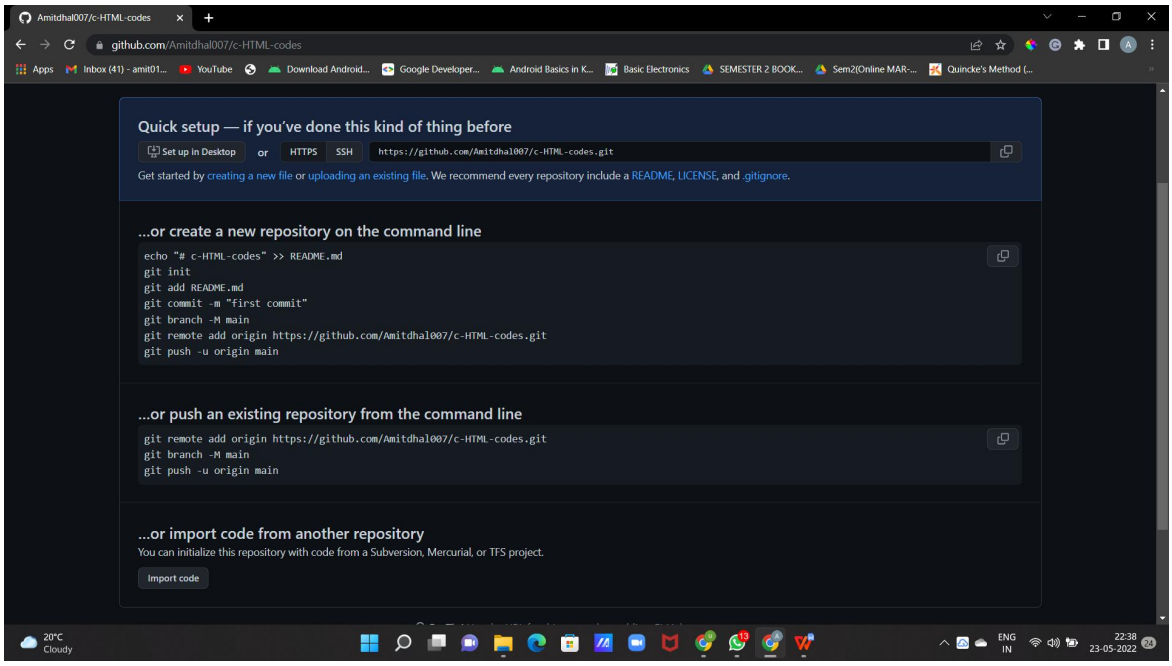
- Click on the 'New' button in the top right corner.



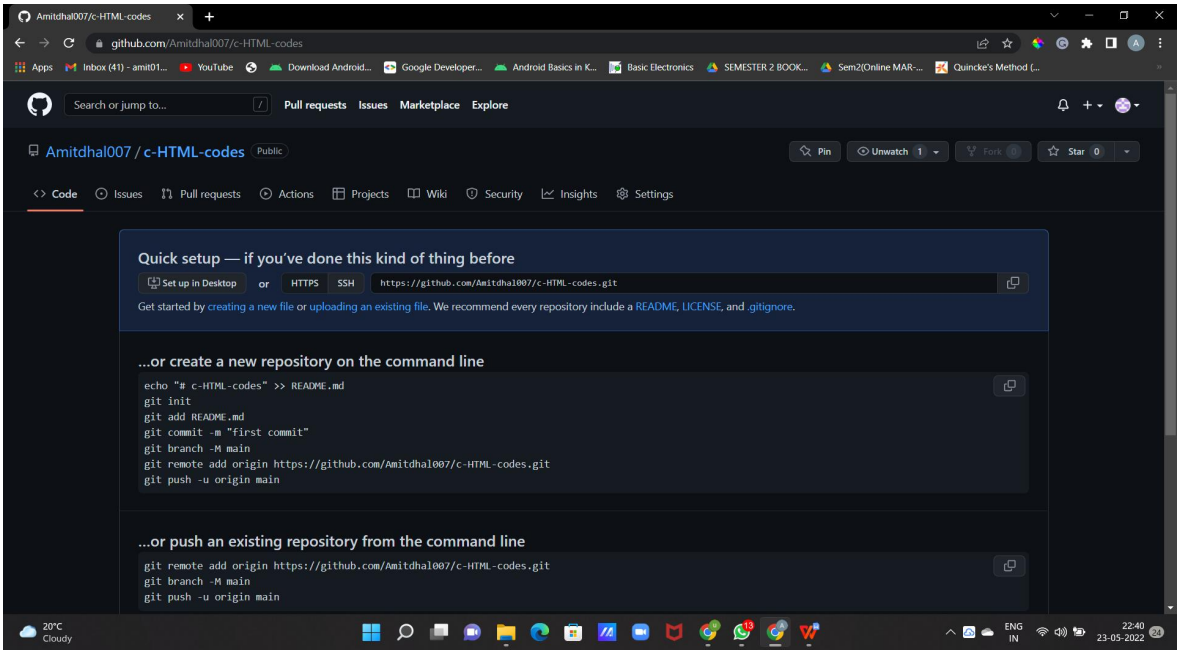
- Enter the Repository name and add the description of the repository.
- Select if you want the repository to be public or private.



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

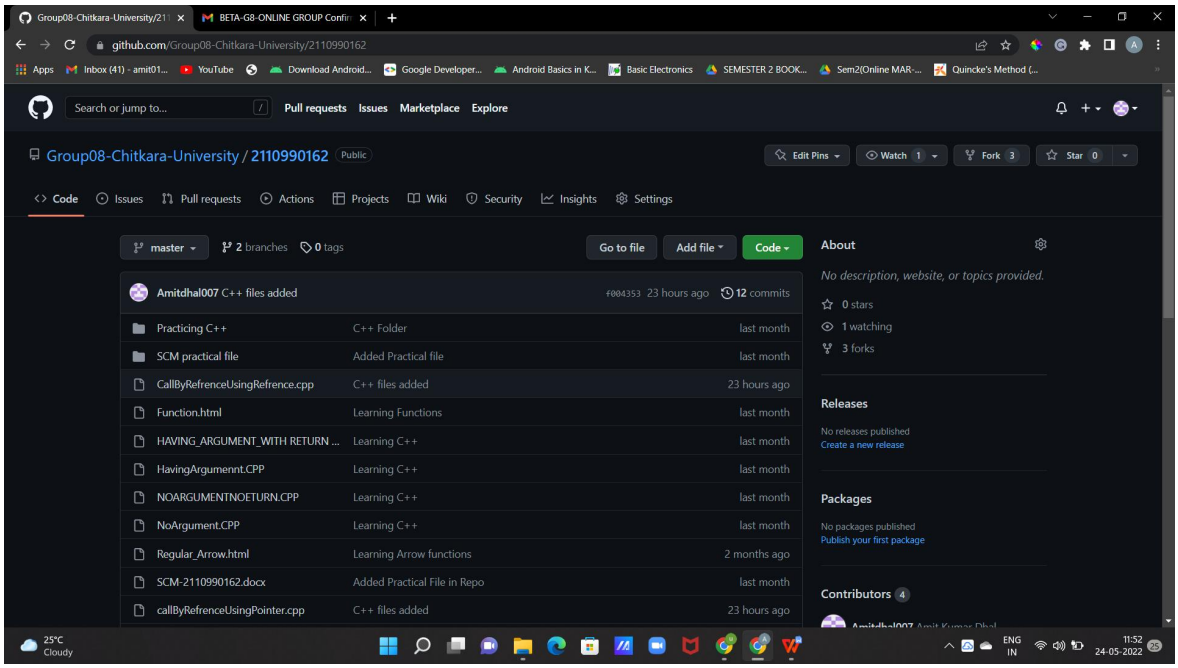


- To create a new file or upload an existing file into your repository select the option in the following box.

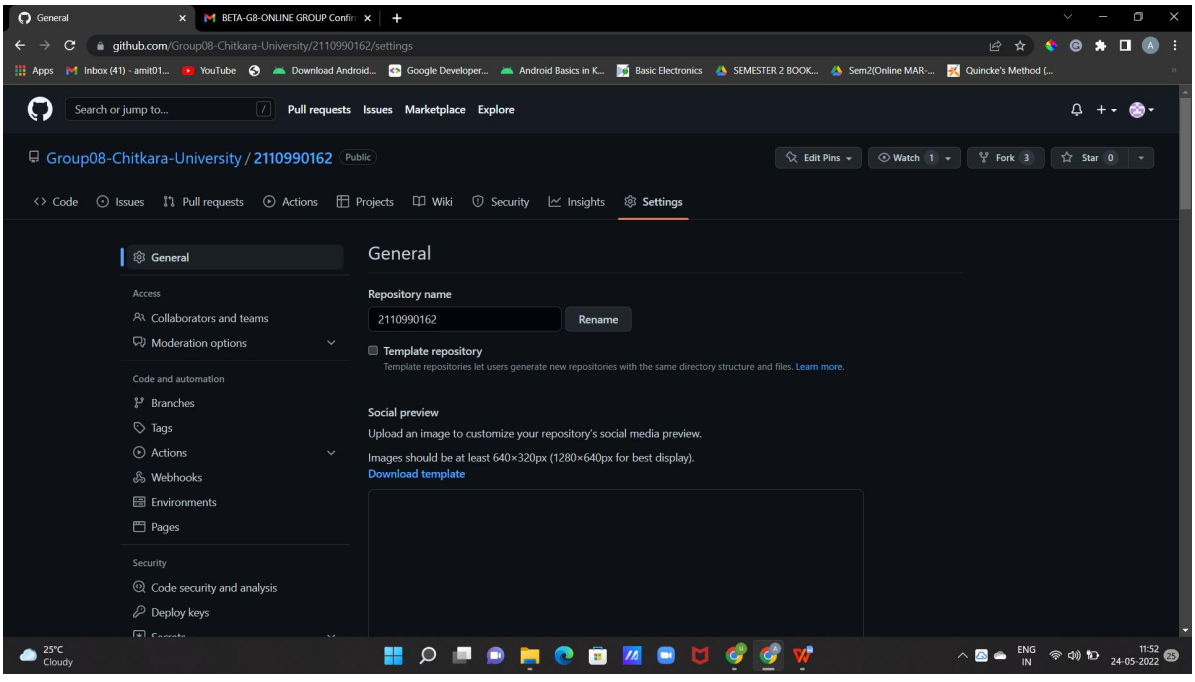


- Now, you have created your repository successfully.

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



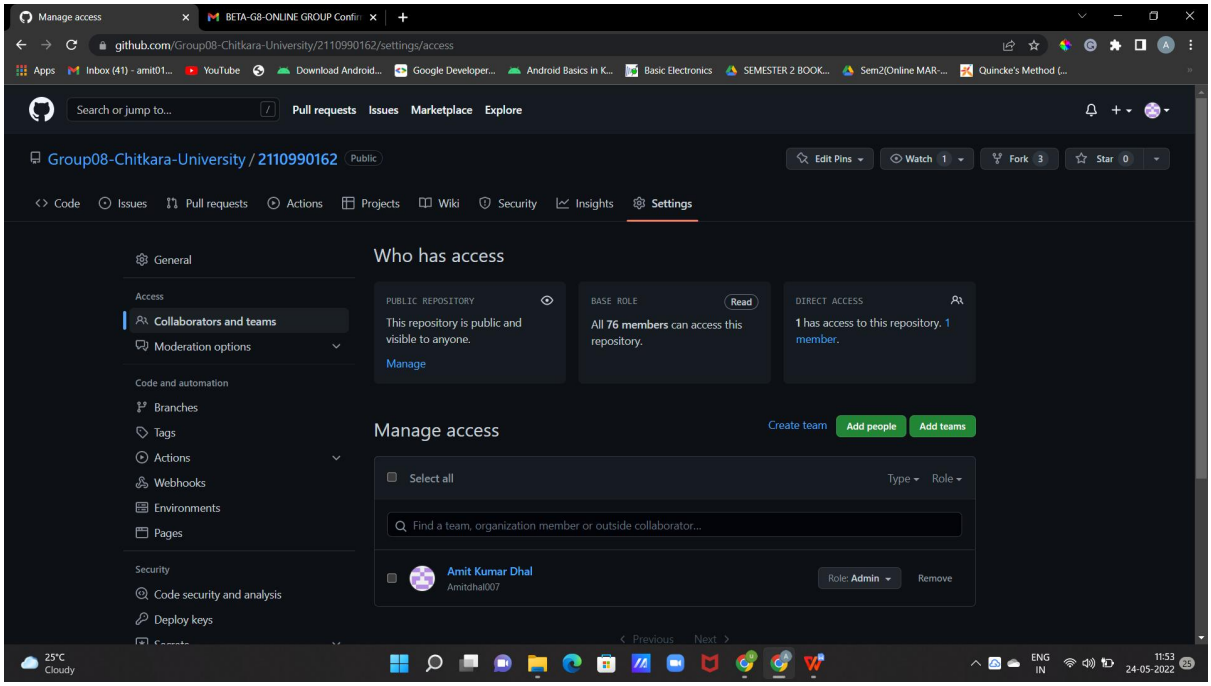
- Click on Collaborators option under the access tab.



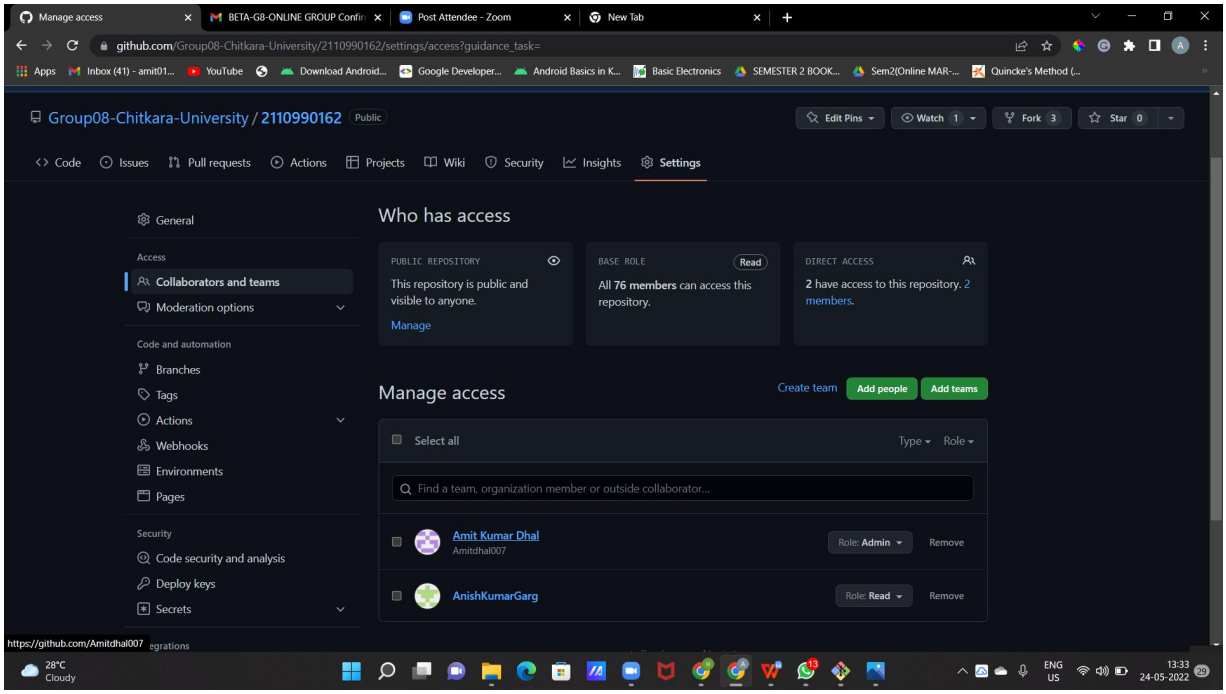
- After clicking on collaborators Github asks you to enter your password to confirm the access to the repository.

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

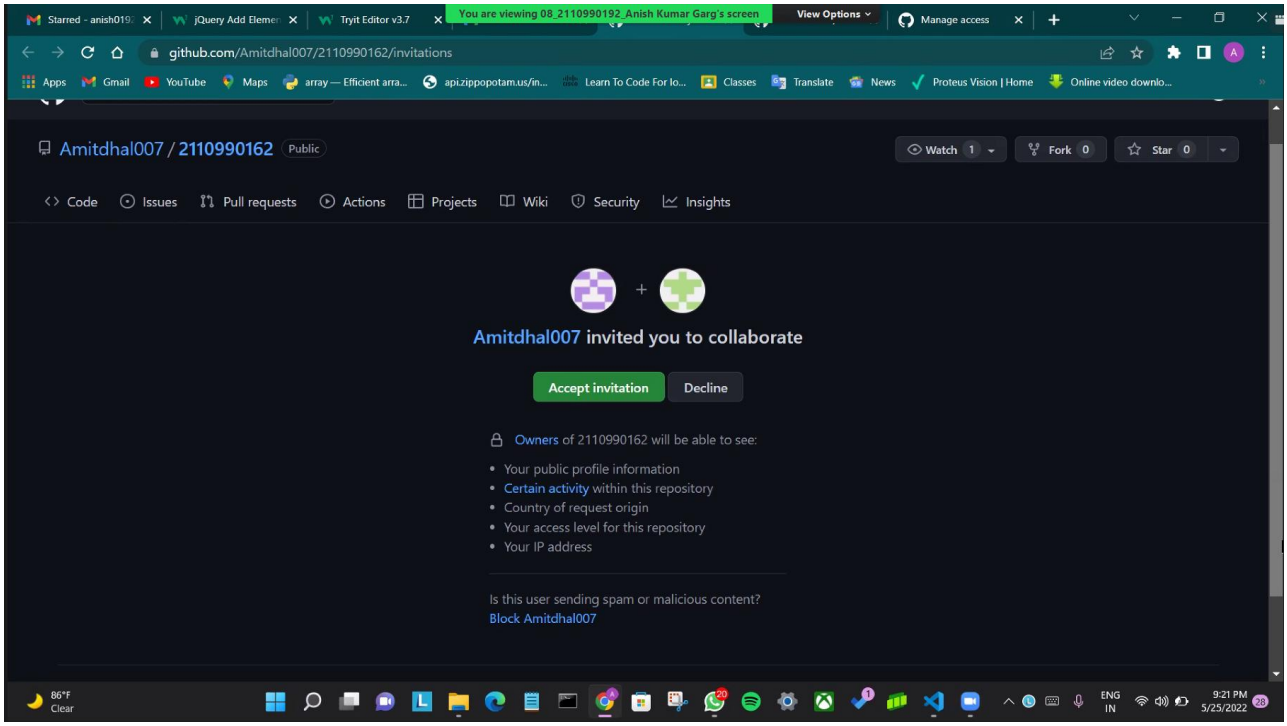
- To add members click on the add people option and search the id of your respective team member.



- 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.

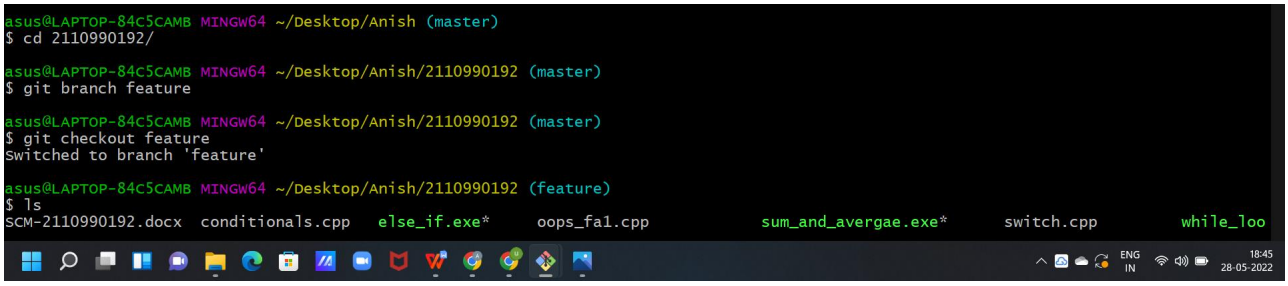


- You will be shown the option that you are now allowed to push.
- Now all members are ready to contribute to the project.

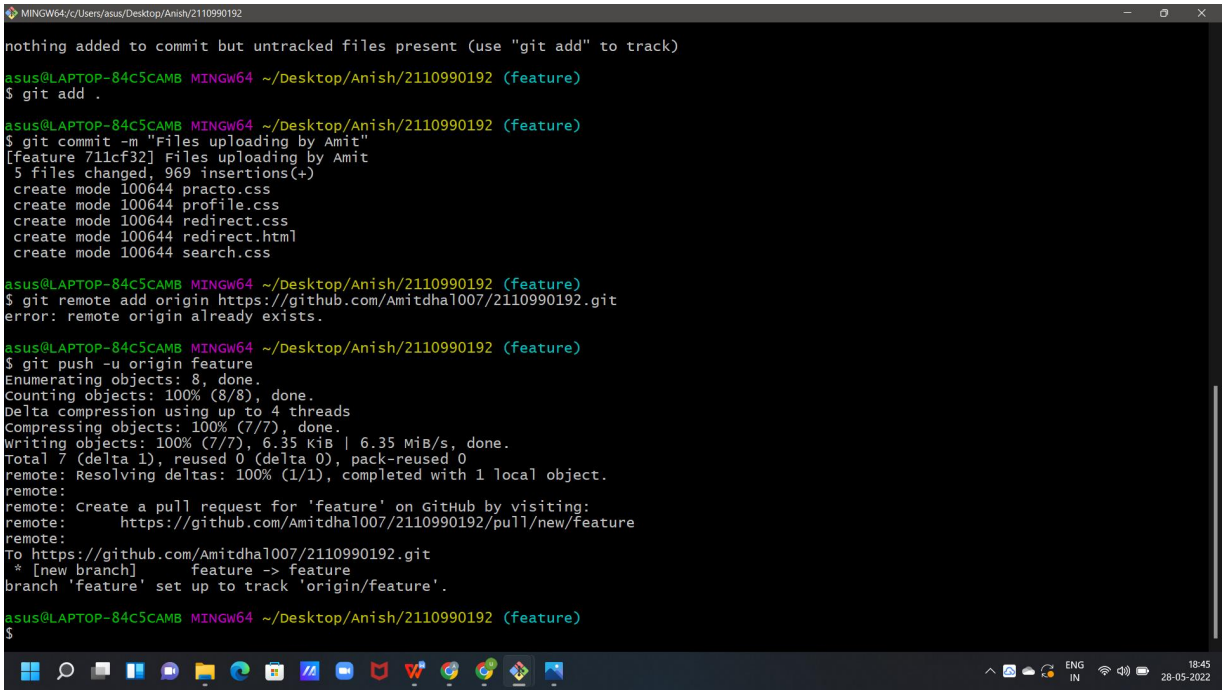
Experiment No. 02

Aim: Open And Close a Pull Request

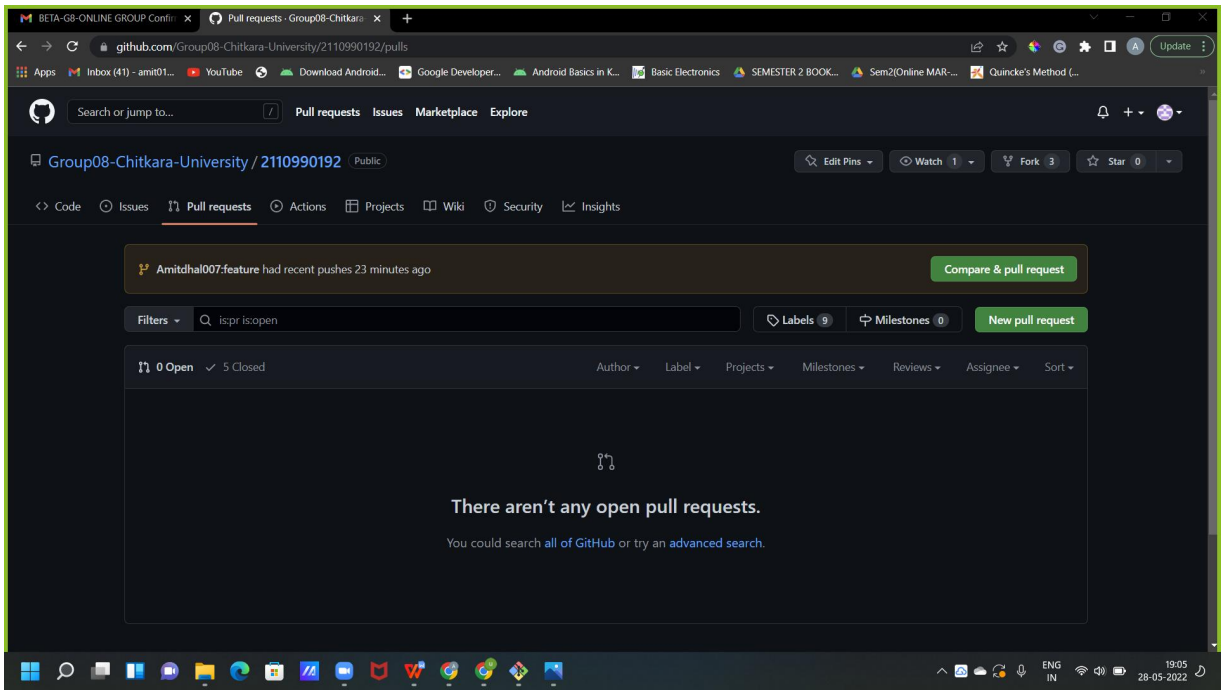
- To open a pull request we first have to make a new branch, by using git branch *branchname* option.
- After making new branch we add a file to the branch or make changes in the existing file.



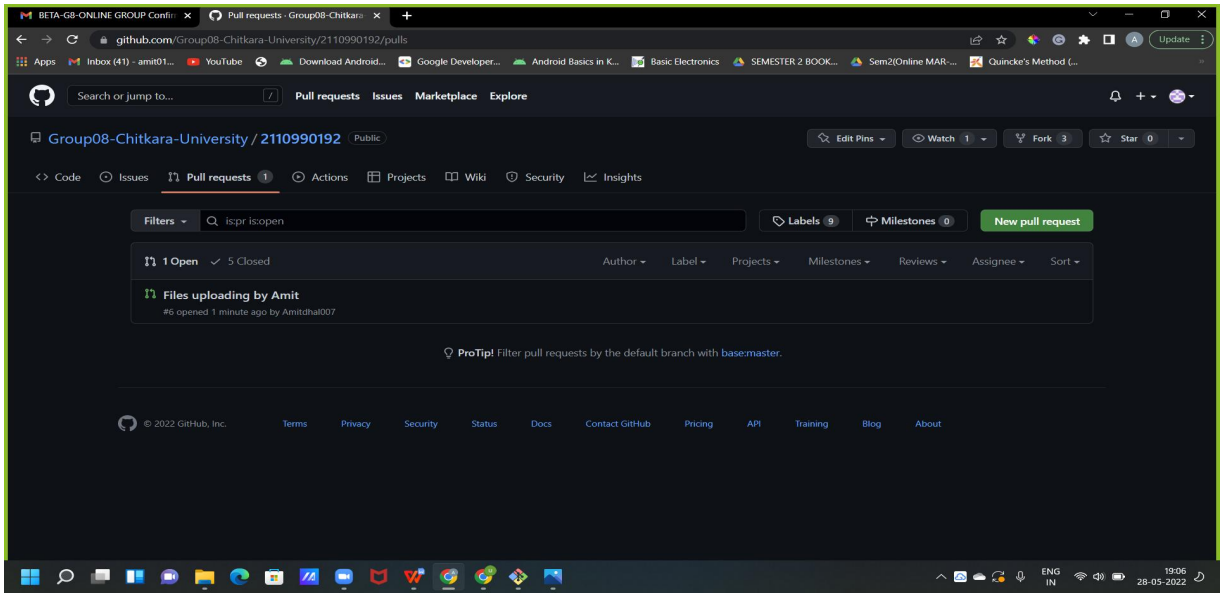
- Add and commit the changes to the local repository.
- Use git push origin *branchname* option to push the new branch to the main repository.



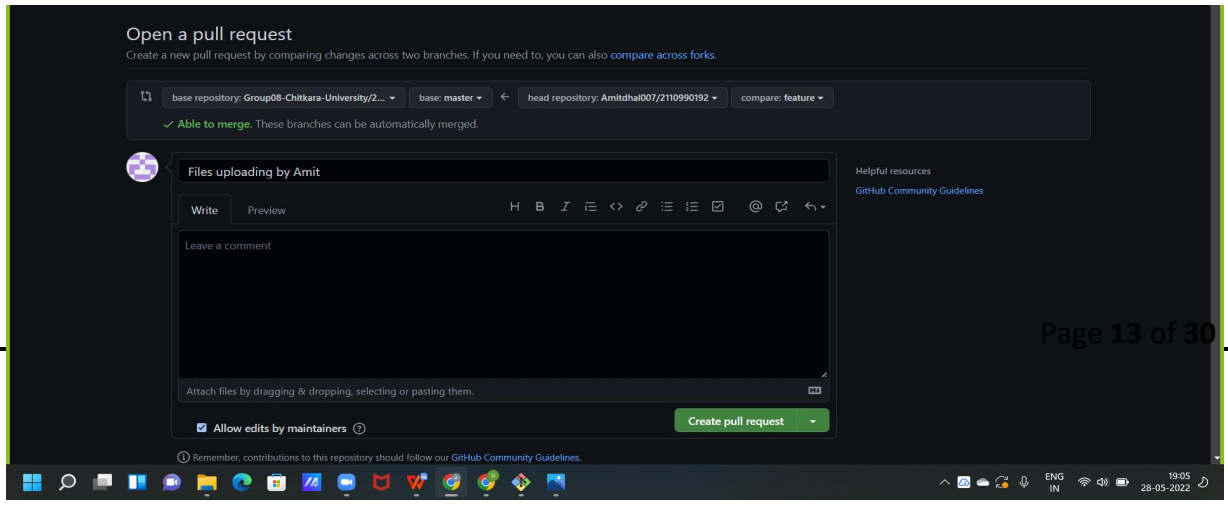
- After pushing new branch Github will either automatically ask you to create a pull request or you can create your own pull request.



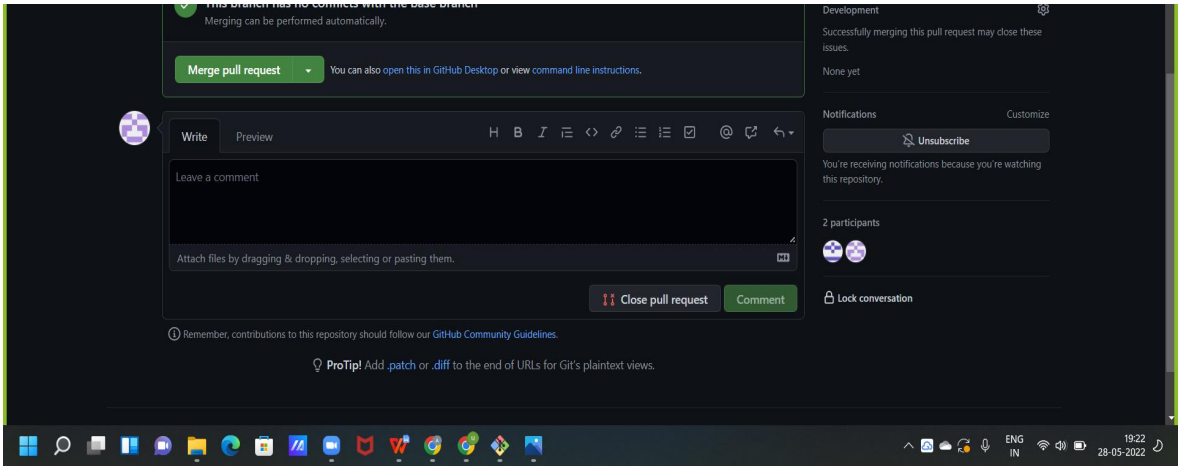
- To create your own pull request click on pull request option.



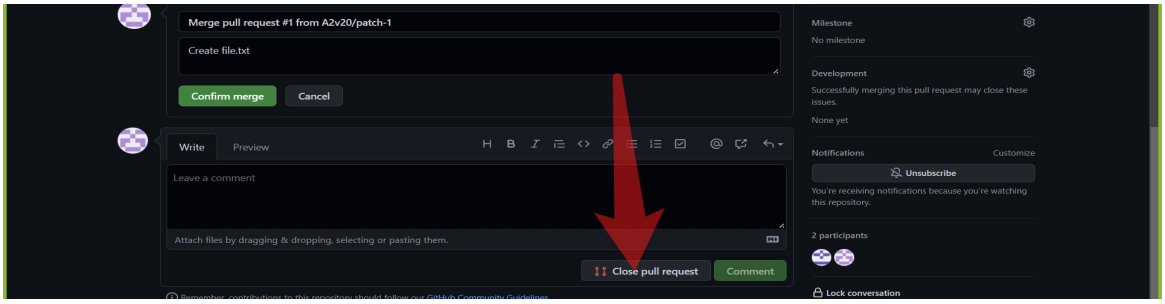
- Github will detect any conflicts and ask you to enter a description of your pull request.



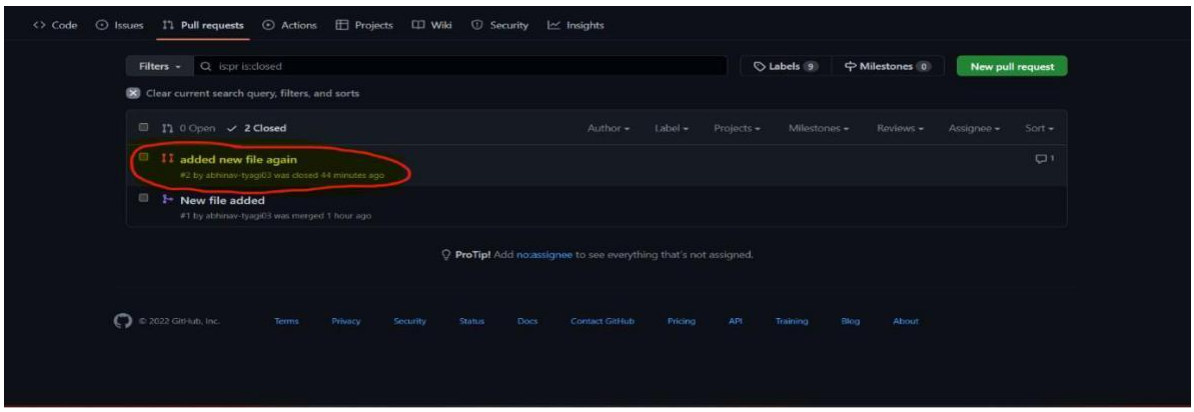
- After opening a pull request all the team members will be sent the request if they want to merge or close the request.



- 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 all the pull request generated and how they were dealt with by clicking on pull request option.

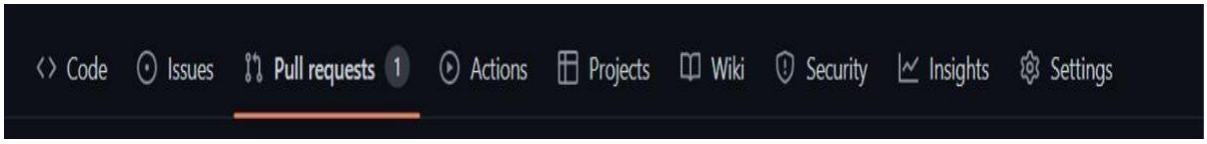


Experiment No. 03

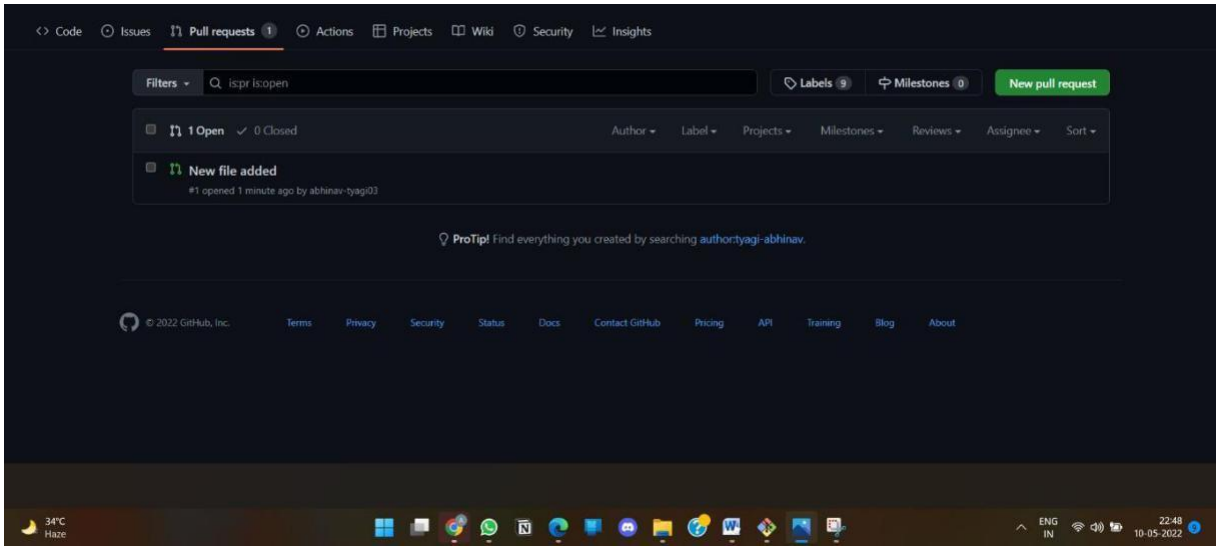
Aim: Create a pull request on a team member's repo and close pull requests generated by team members on own Repo as a maintainer.

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.

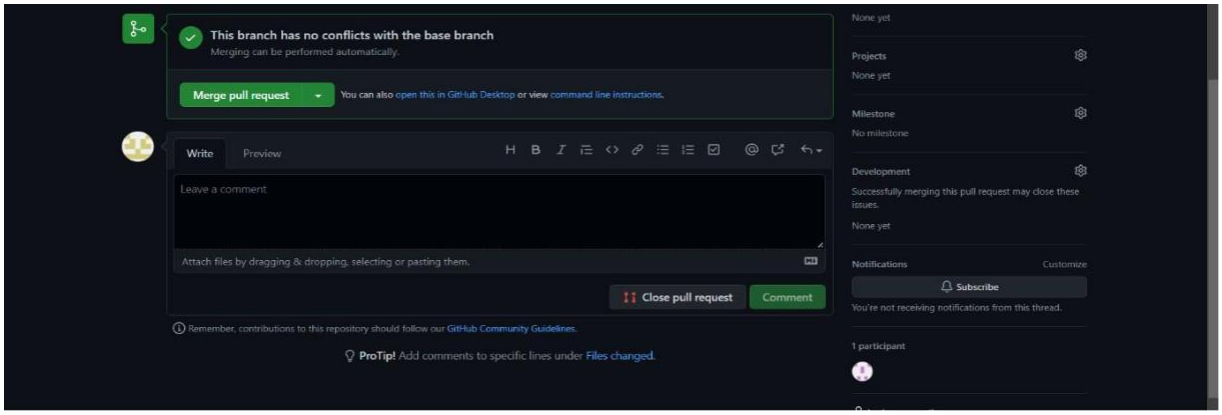


- Click on it. The pull request generated by you will be visible to them.

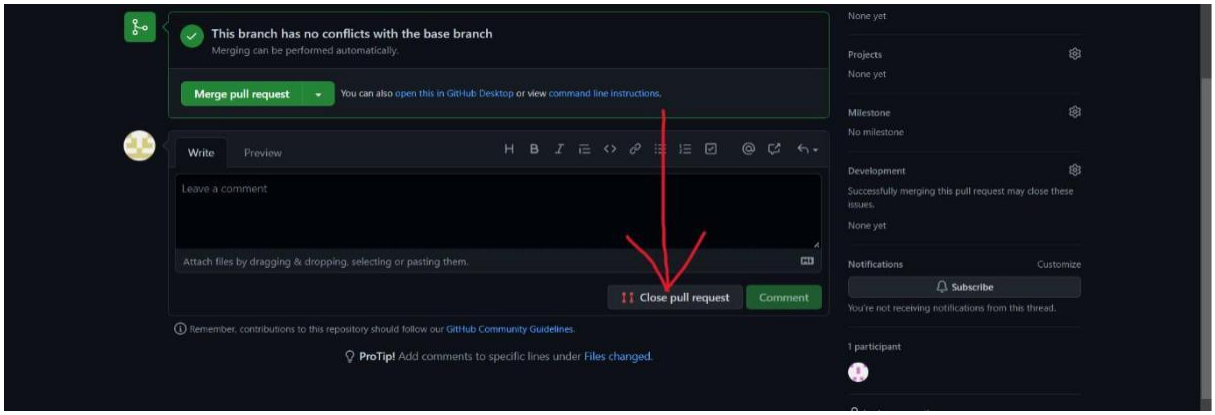


- Click on the pull request. Two option will be available, either to close the pull request or Merge the request with the main branch.

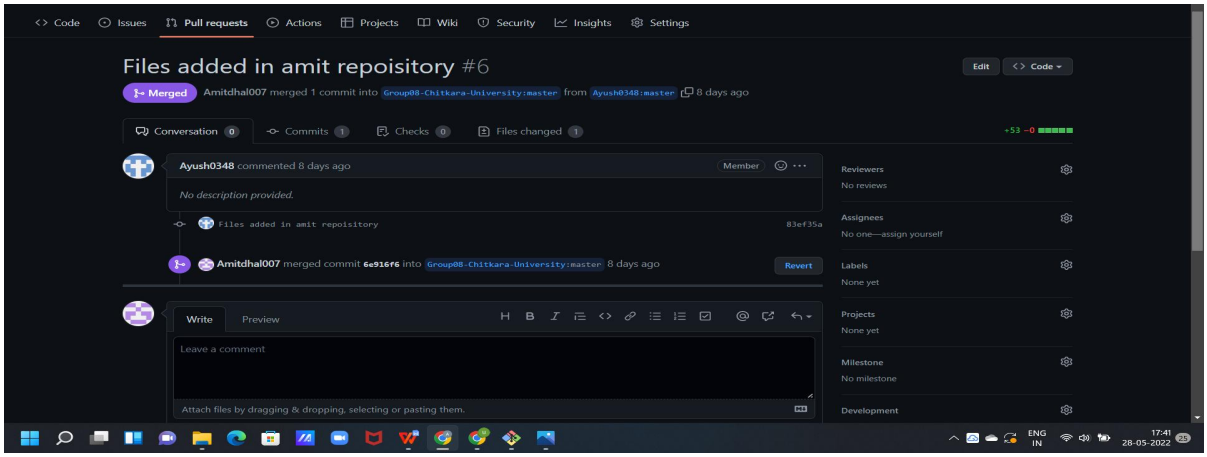
- By selecting the merge branch option the main branch will get updated for all the team members.



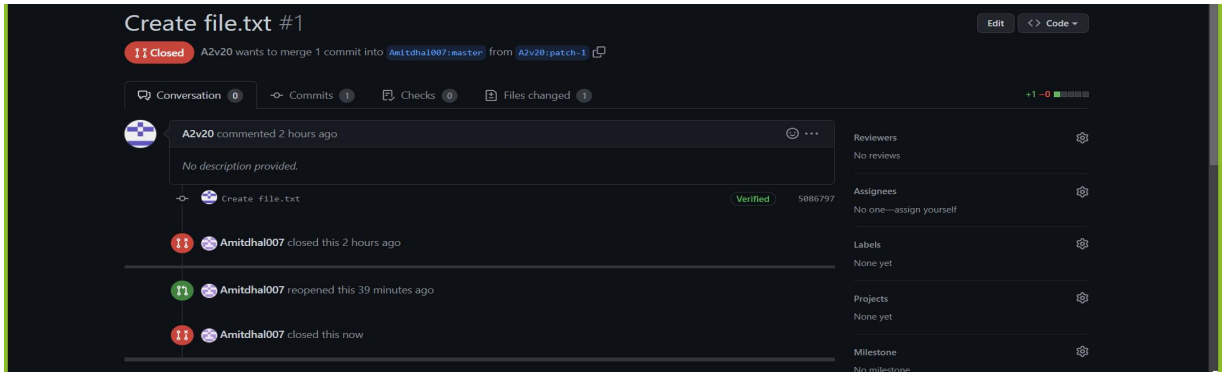
- By selecting close the pull request the pull request is not accepted and not merged with main branch.



- 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.

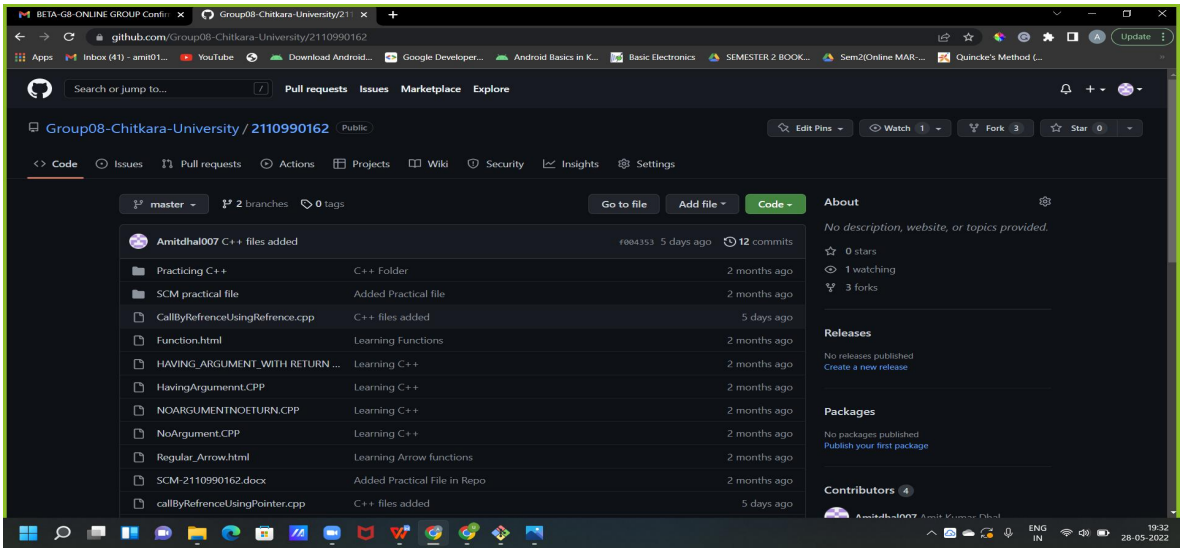
A repository's graphs give you information on traffic, projects that depend on the repository, contributors and commits to the repository, and a repository's forks and network. If you maintain a repository, you can use this data to get a better understanding of who's using your repository and why they're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

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

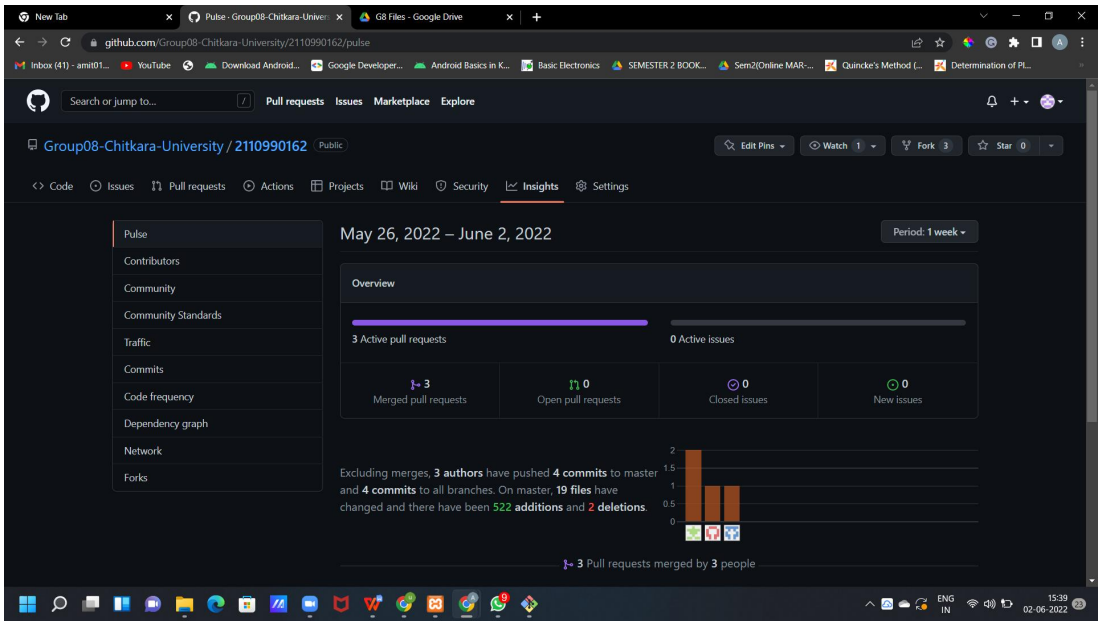
- Pulse
- Contributors
- Traffic
- Commits
- Code frequency
- Network

Steps to access network graphs of respective repository:

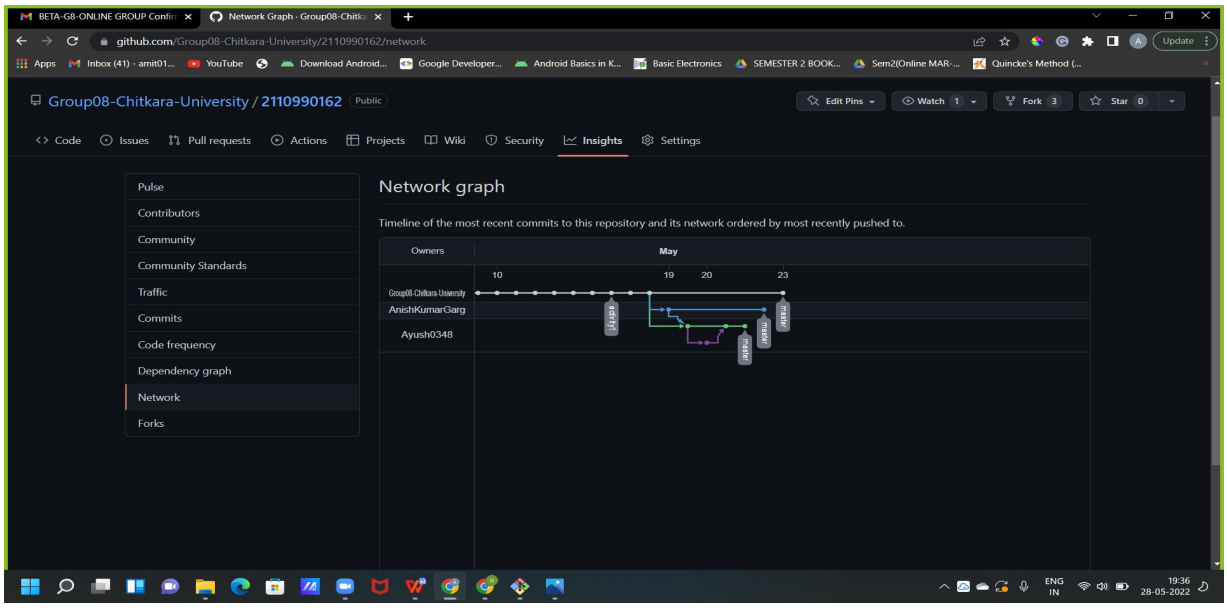
1. On GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click **Insights**.



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.

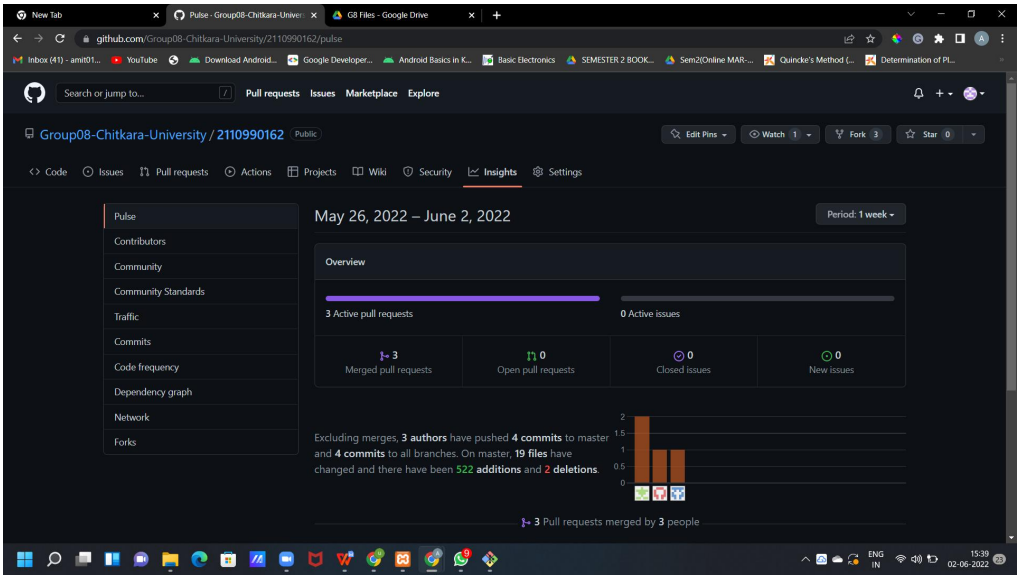


Listing the forks of a repository:

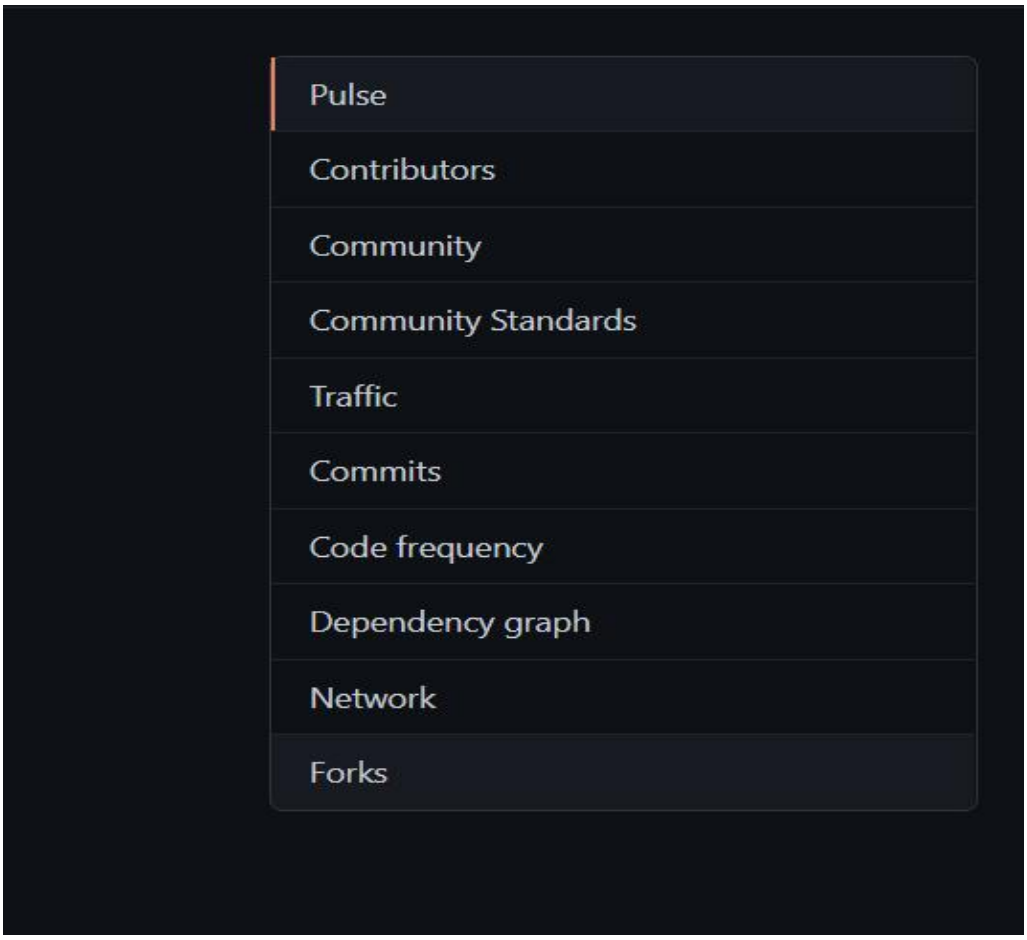
Forks are listed alphabetically by the username of the person who forked the repository

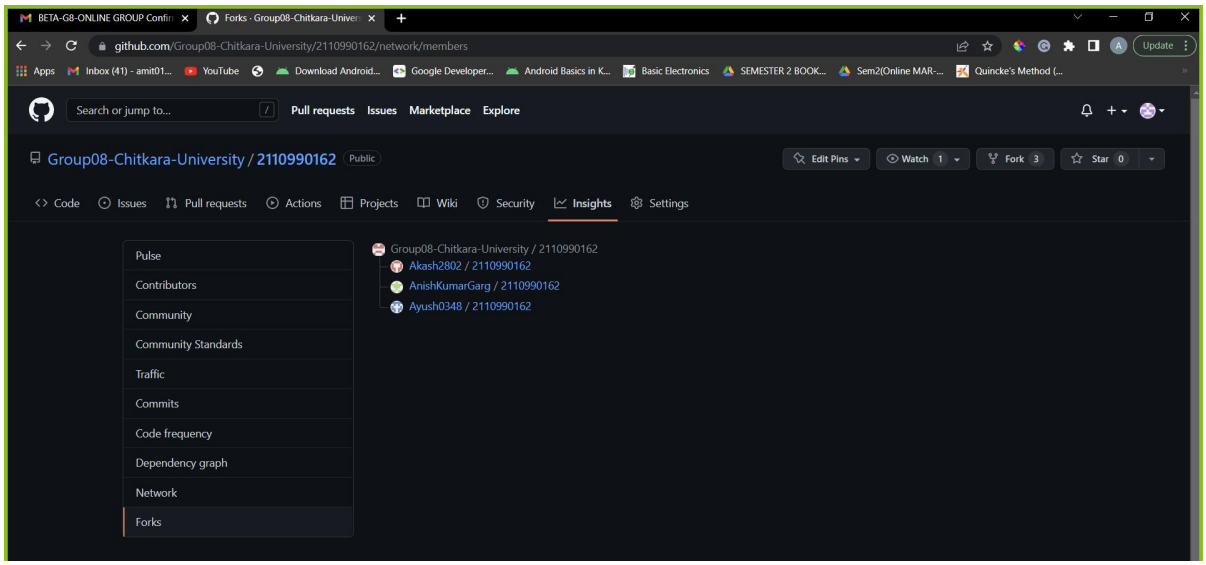
Clicking the number of forks shows you the full network. From there you can click "members" to see who forked the repo

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**.


Pulse
Contributors
Community
Community Standards
Traffic
Commits
Code frequency
Dependency graph
Network
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

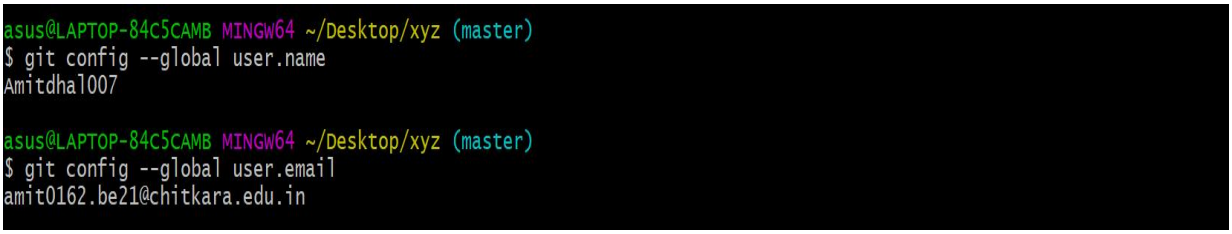
Some Basic commands of Git :-

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.

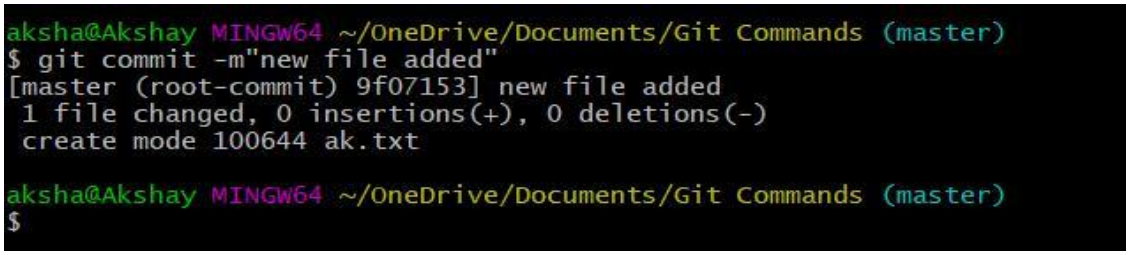


Git add command

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

Git commit command

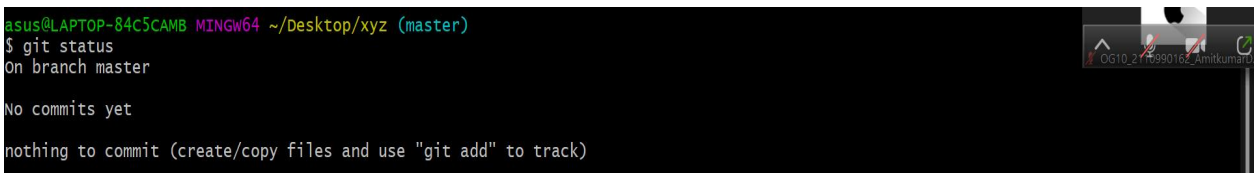
Commit command is used in two scenarios. They are as follows.



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.



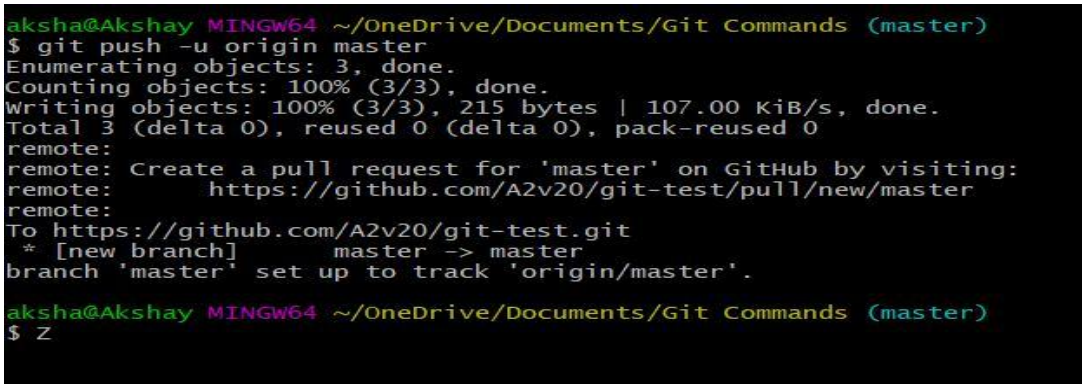
Git push Command

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.

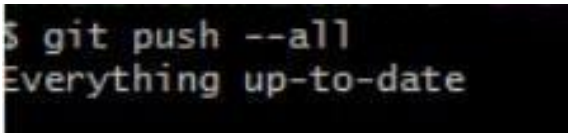
Git push origin master

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



Git push -all

This command pushes all the branches to the server repository.



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.

```
aksha@Akshay MINGW64 ~/OneDrive/Documents/Git Commands (master)
$ git pull
remote: Enumerating objects: 10, done.
remote: Counting objects: 100% (10/10), done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 10 (delta 0), reused 6 (delta 0), pack-reused 0
Unpacking objects: 100% (10/10), 1.72 KiB | 28.00 KiB/s, done.
From https://github.com/A2v20/git-test
 * [new branch]      main       -> origin/main
Already up to date.

aksha@Akshay MINGW64 ~/OneDrive/Documents/Git Commands (master)
$ |
```

Git Branch Command

This is used to create a new branch.

```
asus@LAPTOP-84C5CAMB MINGW64 ~/Desktop/Anish/2110990192 (master)
$ git branch feature

asus@LAPTOP-84C5CAMB MINGW64 ~/Desktop/Anish/2110990192 (master)
$ git checkout feature
Switched to branch 'feature'
```

Git Merge Command

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

```
asus@LAPTOP-84C5CAMB MINGW64 ~/Desktop/Anish/2110990192 (master)
$ git merge feature
Updating e7dc6b8..711cf32
Fast-forward
 practo.css    | 315 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 profile.css   | 129 ++++++++++++++++++++++++
 redirect.css  | 204 +++++++++++++++++++++++++++++++++++++
 redirect.html | 146 ++++++++++++++++++++++++++
 search.css    | 175 ++++++++++++++++++++++++++++++++
 5 files changed, 969 insertions(+)
 create mode 100644 practo.css
 create mode 100644 profile.css
 create mode 100644 redirect.css
 create mode 100644 redirect.html
 create mode 100644 search.css
```

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.

```
asus@LAPTOP-84C5CAMB MINGW64 ~/Desktop/Anish (master)
$ git clone https://github.com/Amitdhal007/2110990192.git
Cloning into '2110990192'...
remote: Enumerating objects: 43, done.
remote: Counting objects: 100% (43/43), done.
remote: Compressing objects: 100% (26/26), done.
remote: Total 43 (delta 19), reused 37 (delta 17), pack-reused 0
Receiving objects: 100% (43/43), 3.85 MiB | 2.84 MiB/s, done.
Resolving deltas: 100% (19/19), done.
```

Git branch –d [branch name]

It is used to delete the current branch name specified.

```
aksha@Akshay MINGW64 ~/OneDrive/Documents/Git Commands (master)
$ git branch -d activity1
Deleted branch activity1 (was 78585d8).

aksha@Akshay MINGW64 ~/OneDrive/Documents/Git Commands (master)
$
```

Git log Command

This command is used to check the commit history.

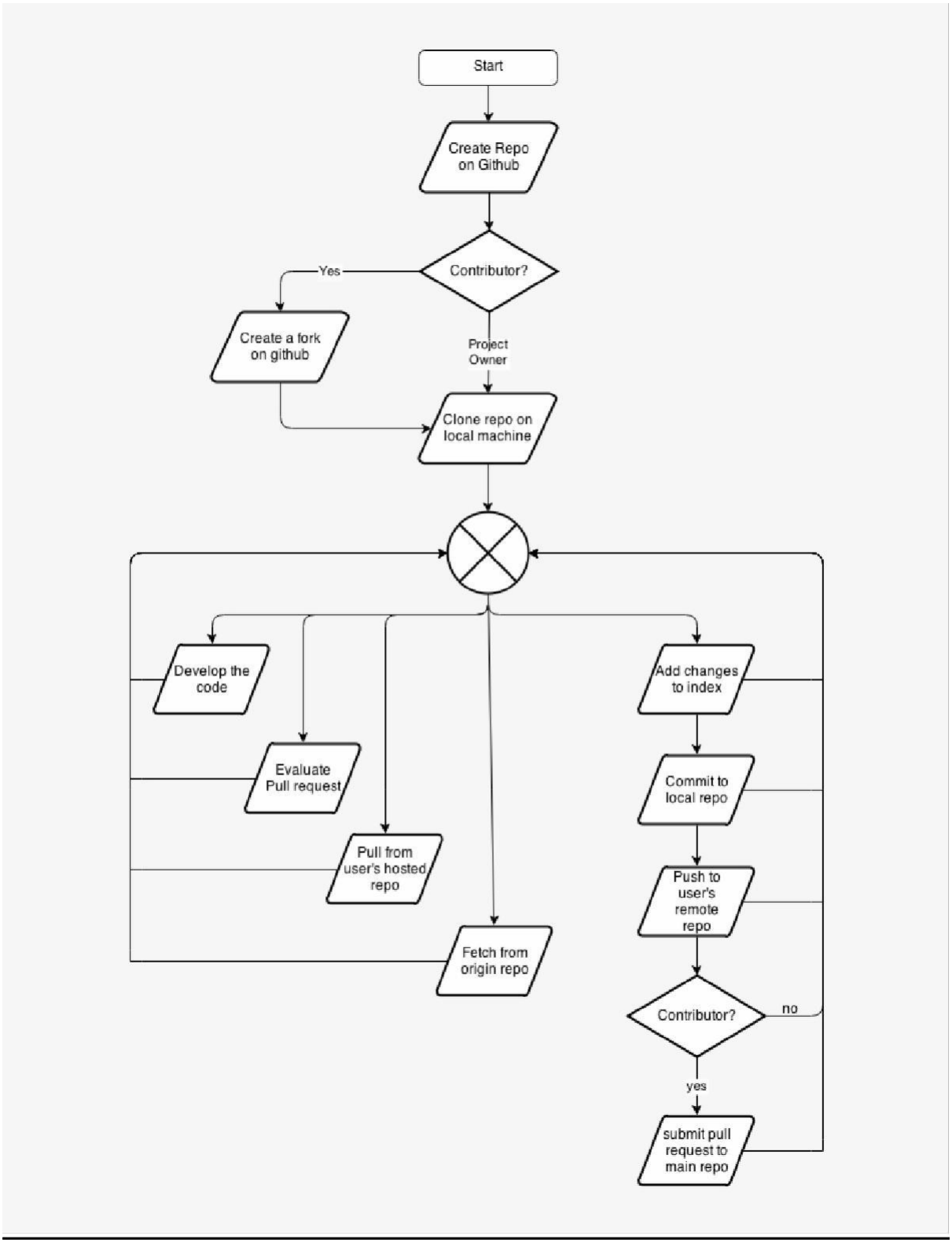
```
aksha@Akshay MINGW64 ~/OneDrive/Documents/Git Commands (master)
$ git log
commit 78585d89f3417b1ff2a2ab3406686f1167742727 (HEAD -> master)
Author: Akshay <akshay0127.be21@chitkara.edu.in>
Date:   Thu May 19 19:15:23 2022 +0530

    files added

commit 9f071538144d350c40c2317ae72f462ca75edbf6 (origin/master)
Author: Akshay <akshay0127.be21@chitkara.edu.in>
Date:   Thu May 19 19:04:37 2022 +0530

    new file added

aksha@Akshay MINGW64 ~/OneDrive/Documents/Git Commands (master)
$
```

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.

Workflow Chart



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