A Project Report

on

**“Basic C++ Programs”**

with

**Source Code Management**

(CS181)

Submitted by

Team Member 1 Name: Aayush Roll No.2110990026

Team Member 2 Name: Adarsh Roll No.2110990074

Team Member 3 Name: Aakansh Roll No.2110990004



**Department of Computer Science & Engineering**

Chitkara University Institute of Engineering and Technology, Punjab

Jan- June   
(2021-22)

|  |  |  |  |
| --- | --- | --- | --- |
| Institute/School Name | **Chitkara University Institute of Engineering and Technology** | | |
| Department Name | **Department of Computer Science & Engineering** | | |
| Programme Name | **Bachelor of Engineering (B.E.), Computer Science & Engineering** | | |
| Course Name | **Source Code Management** | Session | **2021-22** |
| Course Code | **CS181** | Semester/Batch | **2nd/2021** |
| Vertical Name | **Beta** | Group No | G-02 |
| Course Coordinator | **Dr. Neeraj Singla** | | |
| Faculty Name | **Dr. Monit Kapoor** | | |

Submission

Name: Aayush Bansal

Signature:

Date:

**Table of Content**

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Title** | **Page No.** |
| 1 | Version control with Git | 1-6 |
| 2 | Problem Statement | 7 |
| 3 | Objective | 8 |
| 4 | Concepts and commands | 9-19 |
| 5 | Workflow and Discussion | 20-31 |
| 6 | Reference | 32 |

1. Version control with Git

What is the need of Git and why we use Git ??

Any multinational company may face several problems like collaboration among employees, storing several versions of files being made, and data backing up. All these challenges are essential to be resolved for a company to be successful. This is when version control system comes in picture.

Before we get into Version Control's details, let's first understand its relevance by taking an example under consideration.

Imagine there is a multinational company that has offices and employees all around the globe. No matter how big or small a company is, it indeed has some obstacles in its way. Some of the significant challenges faced by such companies are:

Collaboration

Storing Values

Restoring previous version

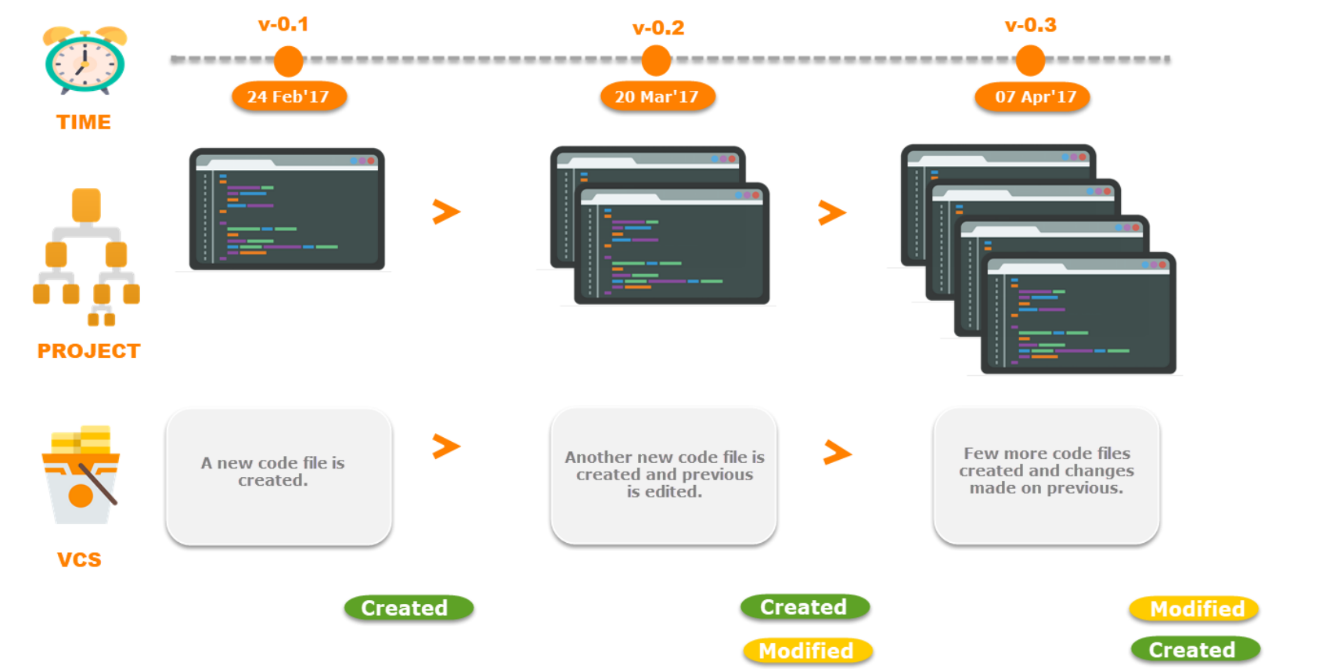
Figure Out what happened

Backup

Solution:

Version control system solve all these challenges…

What is “version control”, and why should you care? Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later. For the examples in this book, you will use software source code as the files being version controlled, though in reality you can do this with nearly any type of file on a computer. If you are a graphic or web designer and want to keep every version of an image or layout (which you would most certainly want to), a Version Control System (VCS) is a very wise thing to use. It allows you to revert selected files back to a previous state, revert the entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more. Using a VCS also generally means that if you screw things up or lose files, you can easily recover. In addition, you get all this for very little overhead.



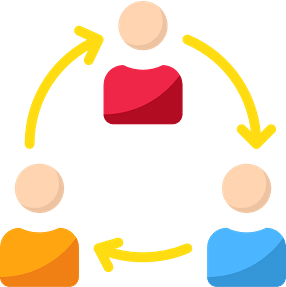
**Benefits of Version Control**

**Managing and Protecting the Source Code**

The Version Control System helps manage the source code for the software team by keeping track of all the code modifications. It also protects the source code from any unintended human error and consequences.

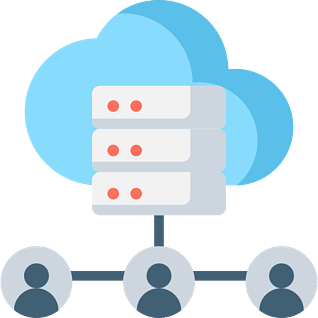
**Keeping Track of All the Modifications Made to the Code**

The team working on the project continuously produces new source codes and keeps making amendments to the existing code. These changes are recorded for future references and can be used if ever needed in the future to discover the root cause of any particular problem.



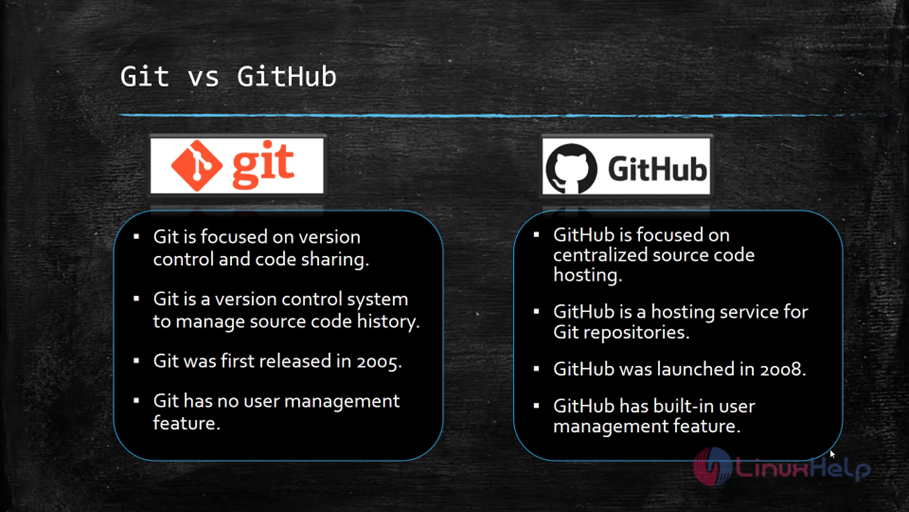
### **Comparing Earlier Versions of the Code**

Since all the versions of the code are saved, this makes it possible for developers to go back at any time and compare the earlier versions of the code to help fix the mistake while reducing disruption to all team members.

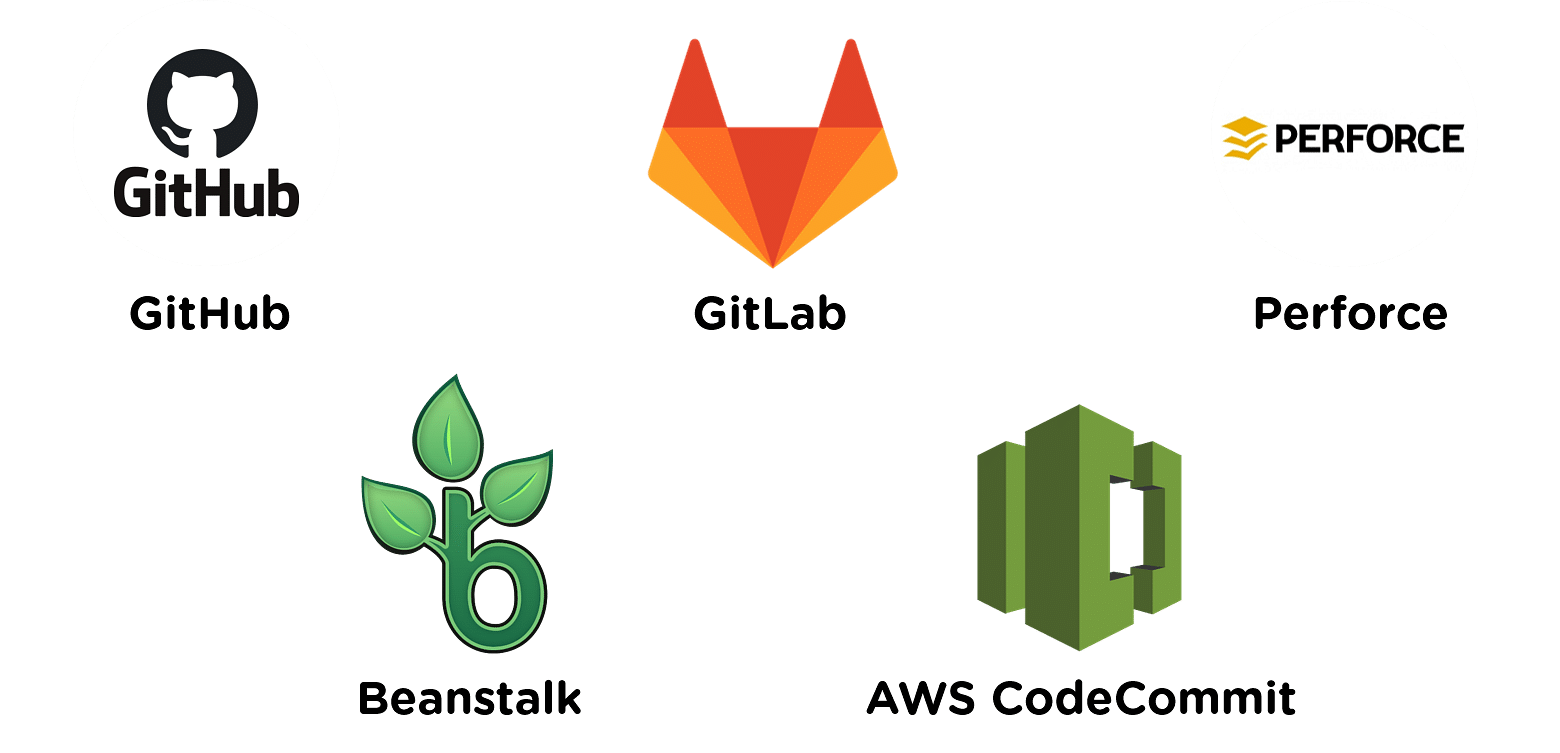


### **Supports the Developers’ Workflow and Not any Rigid Way of Working**

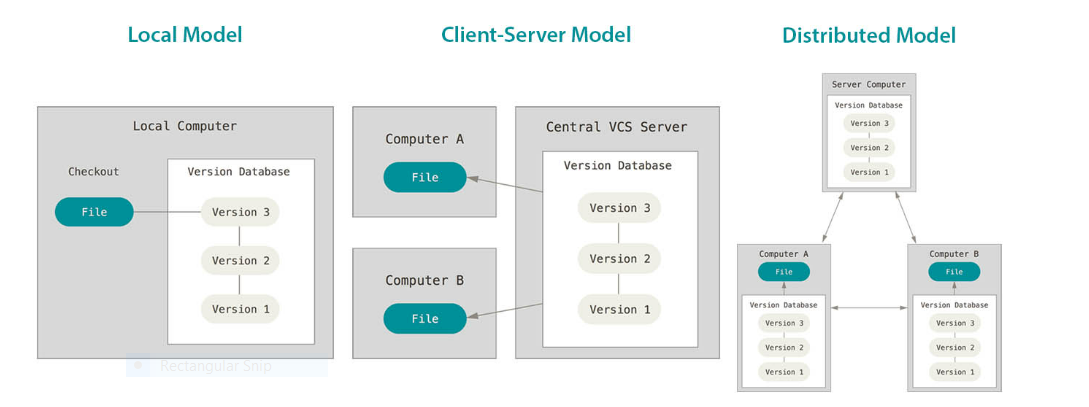
Any suitable Version Control software will not impose any particular way of working. The Version Control Systems are known to provide a smooth and continuous flow of changes made to the code and prevent developers from getting frustrated in this clumsy mechanism.



Finally, let's have a look at some of the best Version Control Systems in the market.

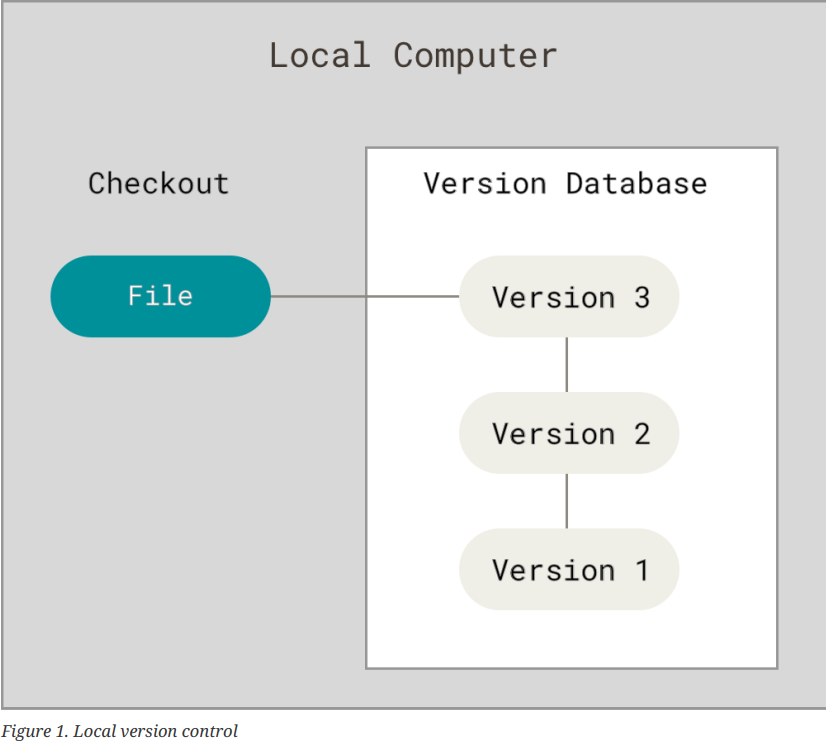


Types of Operating System



**Local Version Control Systems**

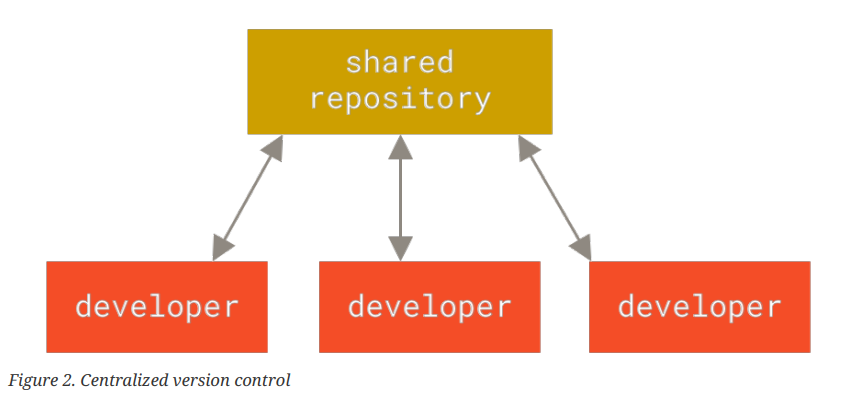
Many people’s version-control method of choice is to copy files into another directory (perhaps a time-stamped directory, if they’re clever). This approach is very common because it is so simple, but it is also incredibly error prone. It is easy to forget which directory you’re in and accidentally write to the wrong file or copy over files you don’t mean to. To deal with this issue, programmers long ago developed local VCSs that had a simple database that kept all the changes to files under revision control.



One of the most popular VCS tools was a system called RCS, which is still distributed with many computers today. RCS works by keeping patch sets (that is, the differences between files) in a special format on disk; it can then re-create what any file looked like at any point in time by adding up all the patches.

**Centralized Version Control Systems**

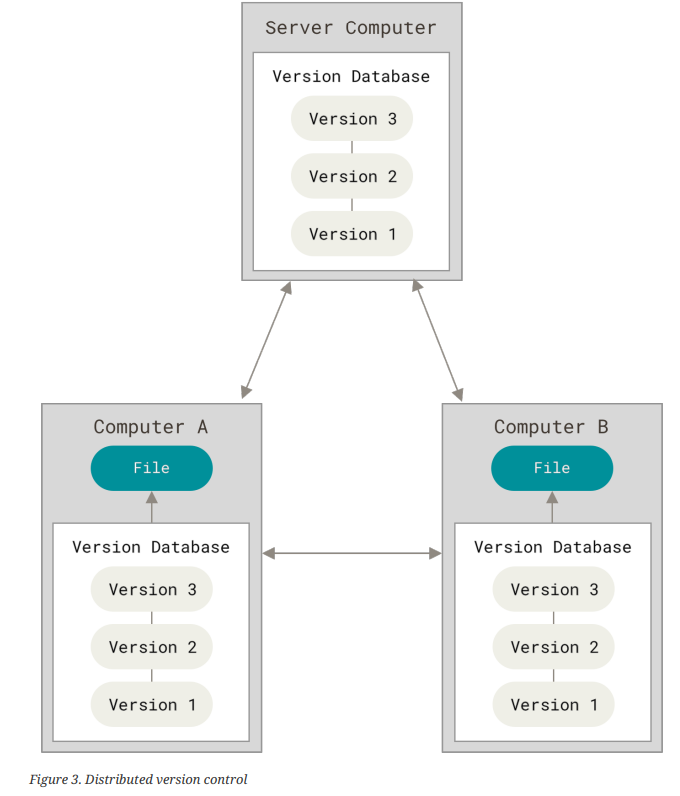
The next major issue that people encounter is that they need to collaborate with developers on other systems. To deal with this problem, Centralized Version Control Systems (CVCSs) were developed. These systems (such as CVS, Subversion, and Perforce) have a single server that contains all the versioned files, and a number of clients that check out files from that central place. For many years, this has been the standard for version control.



This setup offers many advantages, especially over local VCSs. For example, everyone knows to a certain degree what everyone else on the project is doing. Administrators have fine-grained control over who can do what, and it’s far easier to administer a CVCS than it is to deal with local databases on every client. However, this setup also has some serious downsides. The most obvious is the single point of failure that the centralized server represents. If that server goes down for an hour, then during that hour nobody can collaborate at all or save versioned changes to anything they’re working on. If the hard disk the central database is on becomes corrupted, and proper backups haven’t been kept, you lose absolutely everything — the entire history of the project except whatever single snapshots people happen to have on their local machines. Local VCSs suffer from this same problem — whenever you have the entire history of the project in a single place, you risk losing everything.

**Distributed Version Control Systems**

This is where Distributed Version Control Systems (DVCSs) step in. In a DVCS (such as Git, Mercurial, Bazaar or Darcs), clients don’t just check out the latest snapshot of the files; rather, they fully mirror the repository, including its full history. Thus, if any server dies, and these systems were collaborating via that server, any of the client repositories can be copied back up to the server to restore it. Every clone is really a full backup of all the data.



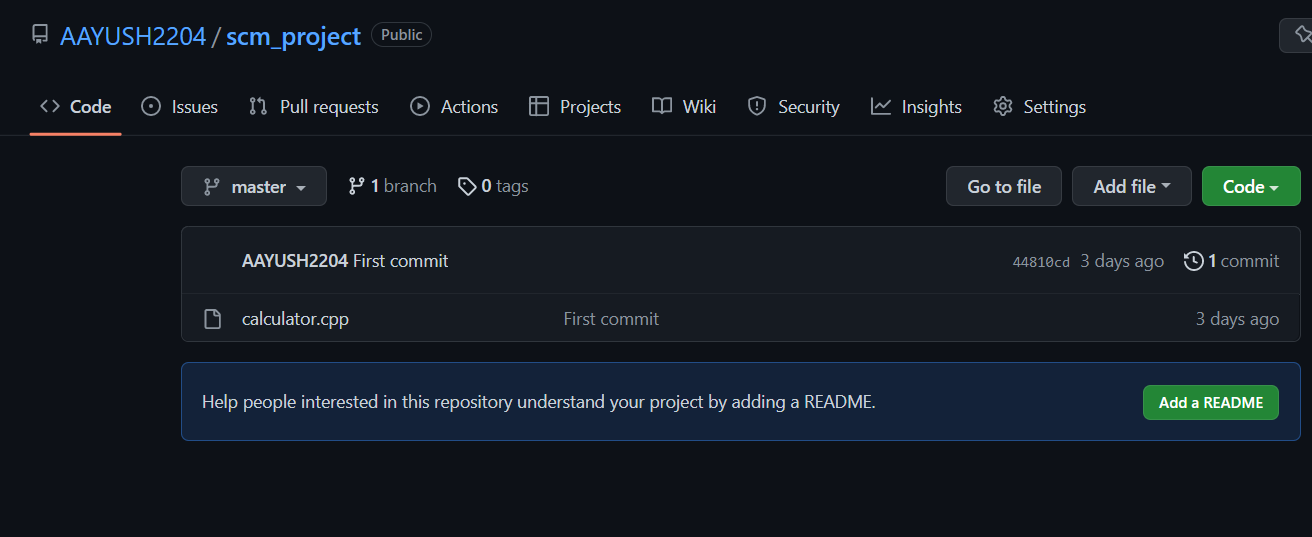
Furthermore, many of these systems deal pretty well with having several remote repositories they can work with, so you can collaborate with different groups of people in different ways simultaneously within the same project. This allows you to set up several types of workflows that aren’t possible in centralized systems, such as hierarchical models.

2. Problem Statement

The Problem Solved with this task is that , many new and aspiring coders face various types of problem with their codes , they stuck in programming/codes and sometimes if their problem isn’t solved they start to leave programming if they aren’t guided properly , the small problems of syntax error or logical error becomes a bigger obstacle if not solved.

Solution:

With this Repository of Ours, we present various basic and common codes which new coders may require and with the help of the collaborating feature of git me and my team has written many common and basic codes with proper comments which will help new coders to easily find their mistakes and build their logic.



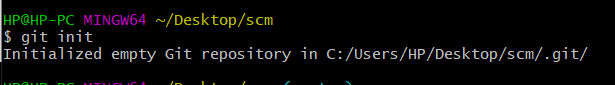
3. Objective

**The objectives of the Aforementioned task are as follows:**

1. Learned how to work in a systematic way in teams.
2. Maximized Space utilization using GitHub.
3. Enhanced collaborations including version control and access control by letting other people contribute to our projects.
4. Easy Project/File management.
5. Code sharing and tracking changes in our code across versions.

4. Concepts and Commands

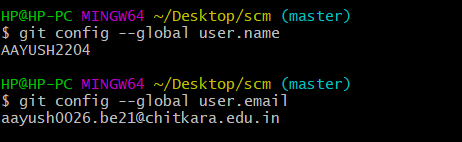
**Git init command**



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.



### **3 stage architecture(touch,status,add,commit):**

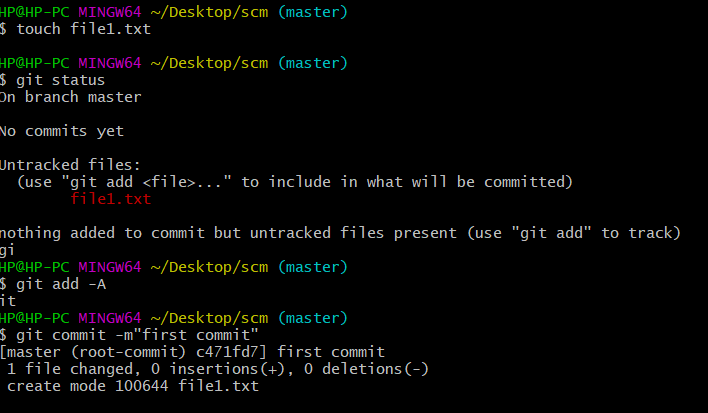
### 

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

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

Touch🡪Used to create files.

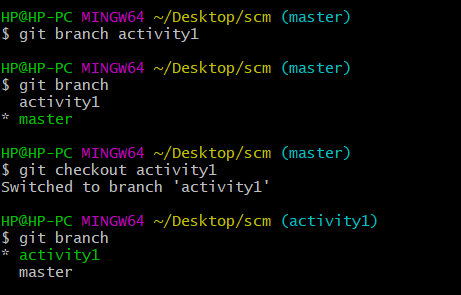
Status-🡪 The status command is used to display the state of the working directory and the staging area.



To create new branch:

Git branch -🡪To check number of branches.

Git checkout branchname-🡪 To switch from one branch to other



### **Change file name:**

### **mv 🡪when we change the name of file with the help of mv command it show that the file is deleted..so after changing we add and commit it.**

### 

### **git mv 🡪when we rename the command it simply renamed it.**

### 

### **Merge 🡪 when we want to merge one branch into another branch we use git merge command.**

### 

### **To delete a branch🡪**

### **Git branch -d 🡪show an error ..first it wanted that the user first completely merge it.**

### 

### **Git branch -D🡪it delete the branch without merging.**

### 

### 

### **To change branch name🡪**

### **Git branch -m “newname”--🡪To change branch name.**

### **Git restore –staged🡪come file from stagging to unstagging area;**

### 

### **Pwd🡪 to check present working directory**

### 

### **Merge Conflict🡪**

### page15image37861808

### **Merge in one branch**

### Some files present in one branch and the context of this file is different from other branch file so when I merge it show an error to solve that problem we use git merge tool.

### 

### 

### 

### **Local 🡨 ------git push-------------🡪Remote**

### **Remote -------------pull/clone-----------🡪Local**

### **Remote 🡨----------Fork-------------------🡪Remote**

### **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 remote

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.

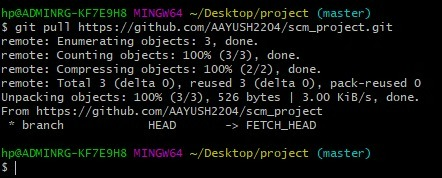
**Git push -all**

This command pushes all the branches to the server repository.



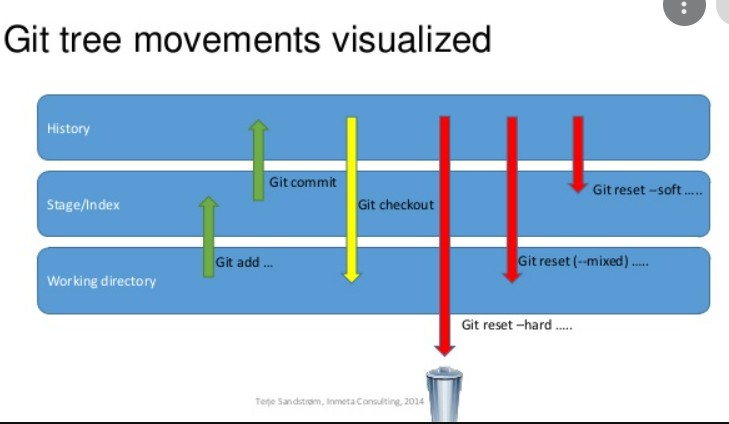
### **Git pull command**

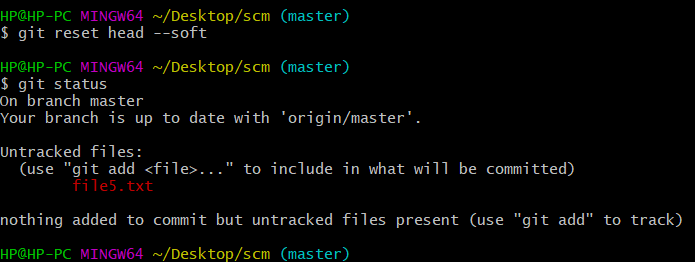
Pull command is used to receive data from GitHub. It fetches and merges changes on the remote server to your working directory.

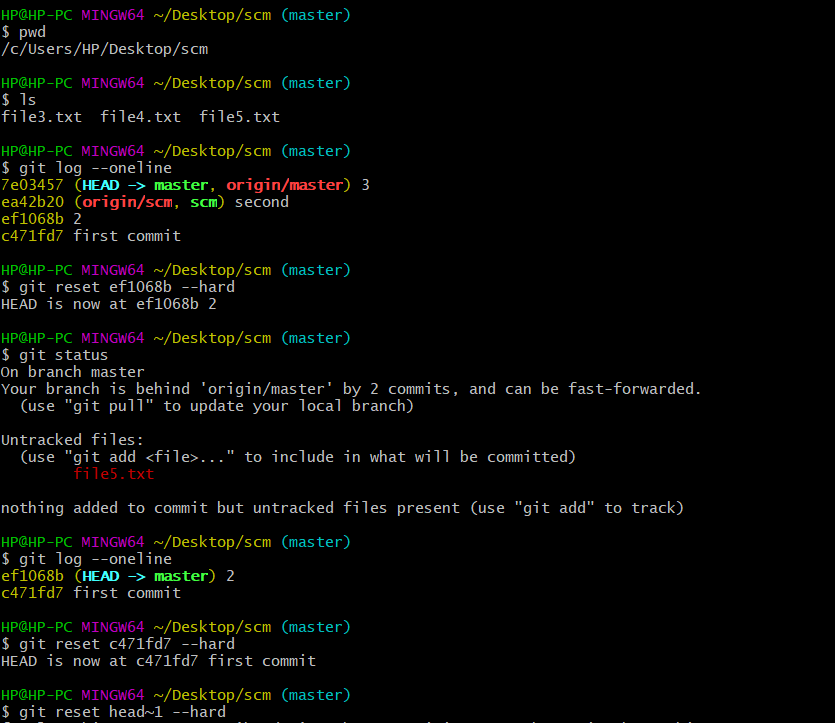


Reset and Revert-🡪

Reset🡪

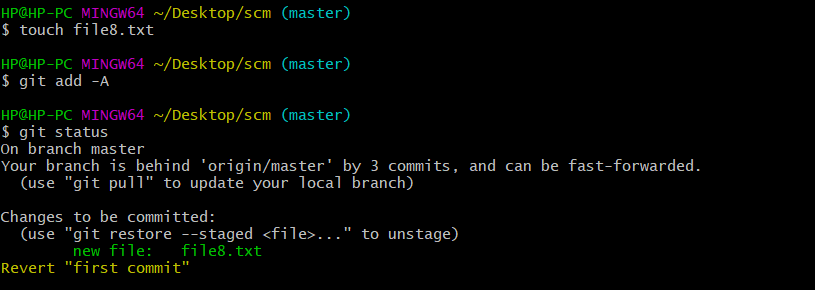








Git revert -🡪

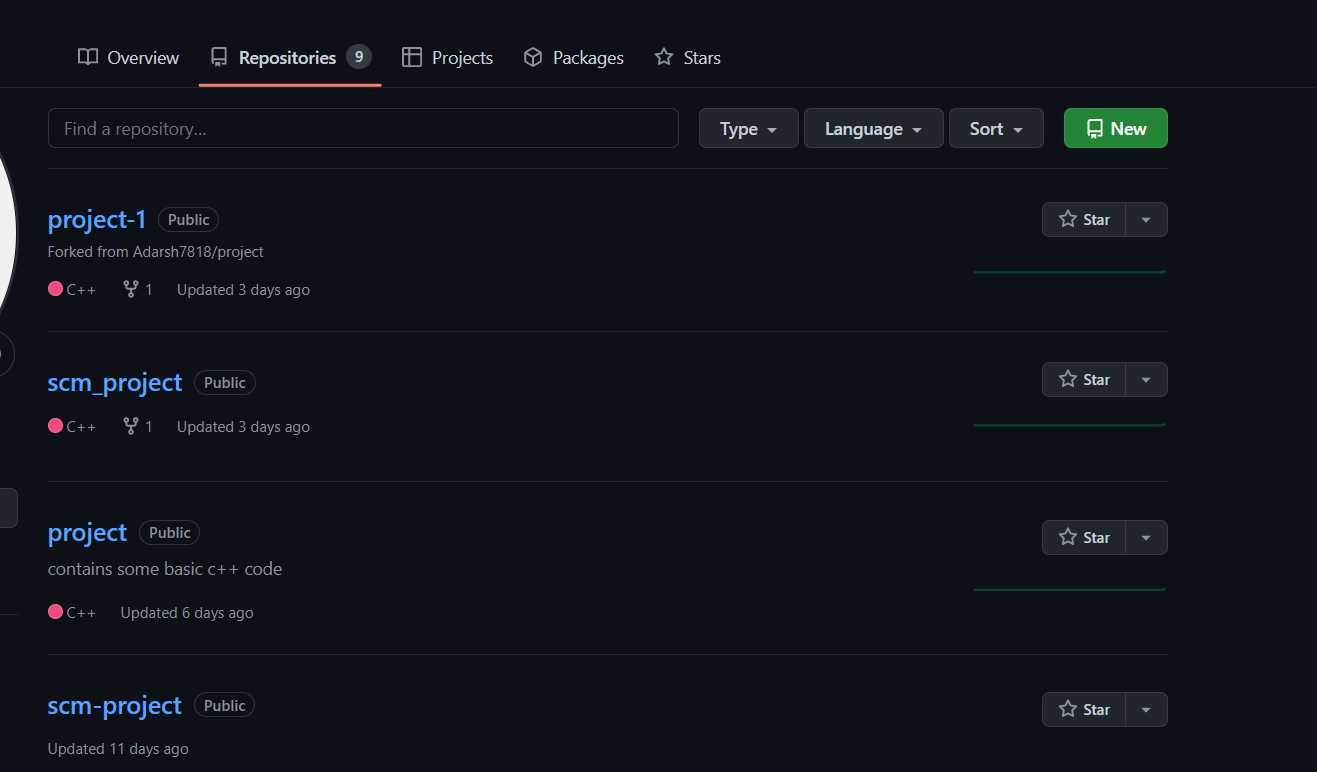


5. Workflow and Discussion

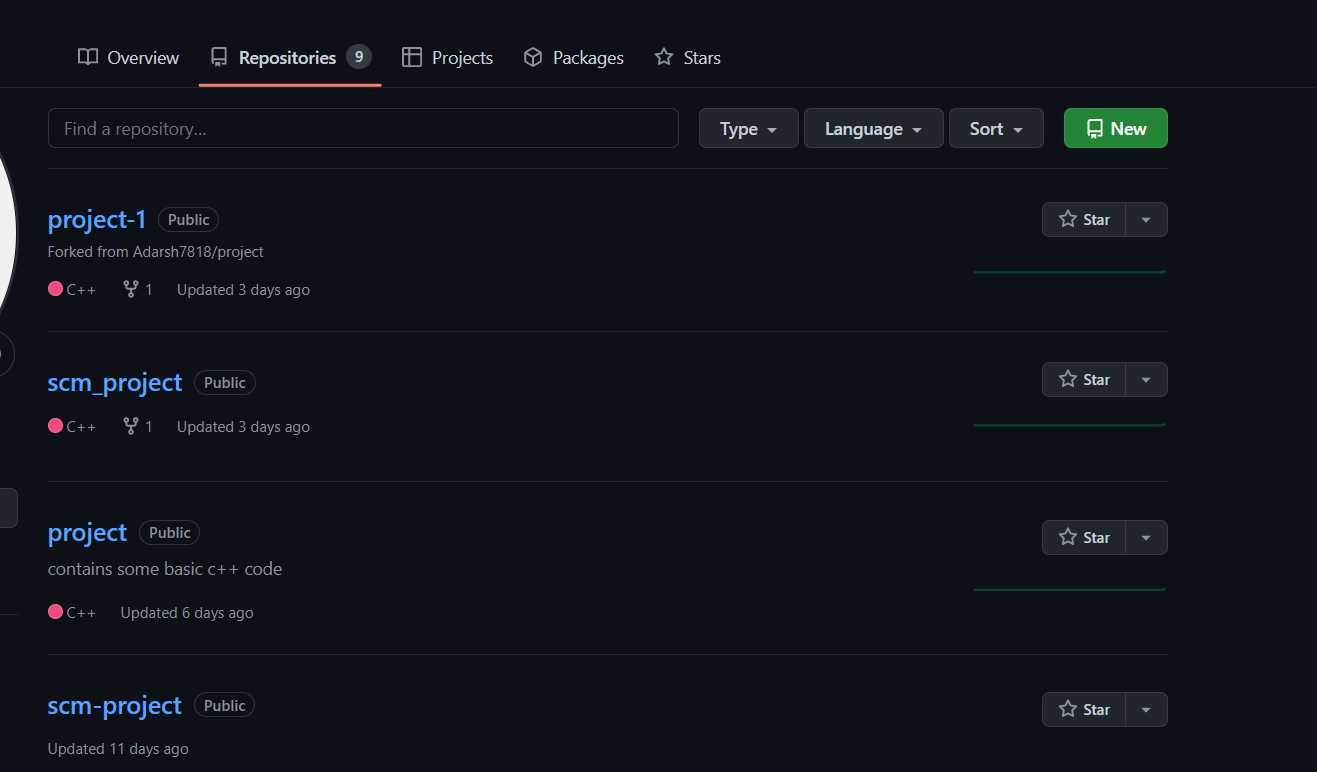
**Create a distributed Repository and add members in project team**

## 

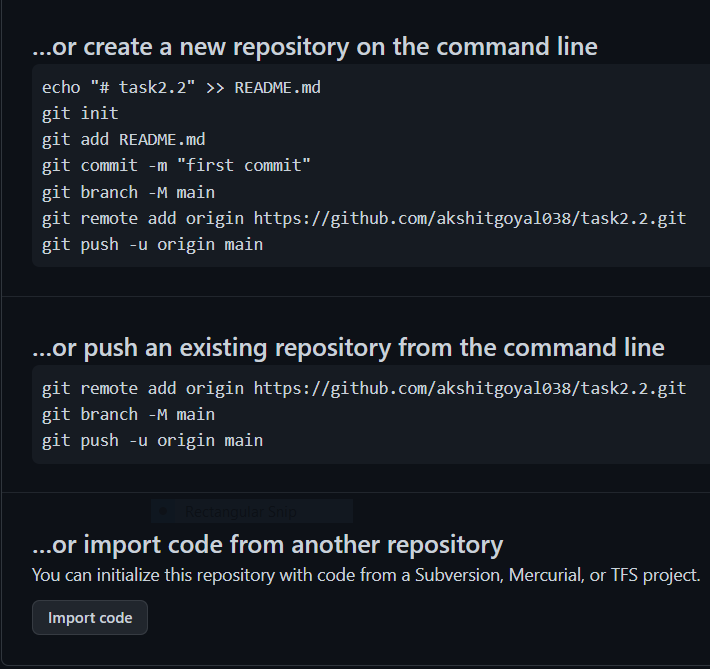
* Login to your Github account and you will land on the homepage as shown below. Click on Repositories option in the menu bar.
* Click on the ‘New’ button in the top right corner.



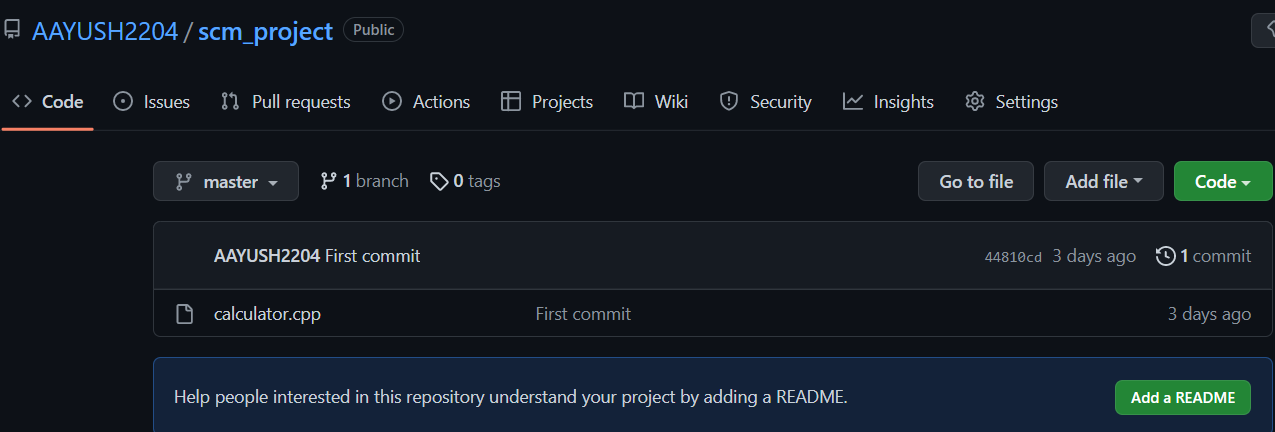
* Enter the Repository name and add the description of the repository.
* Select if you want the repository to be public or private.



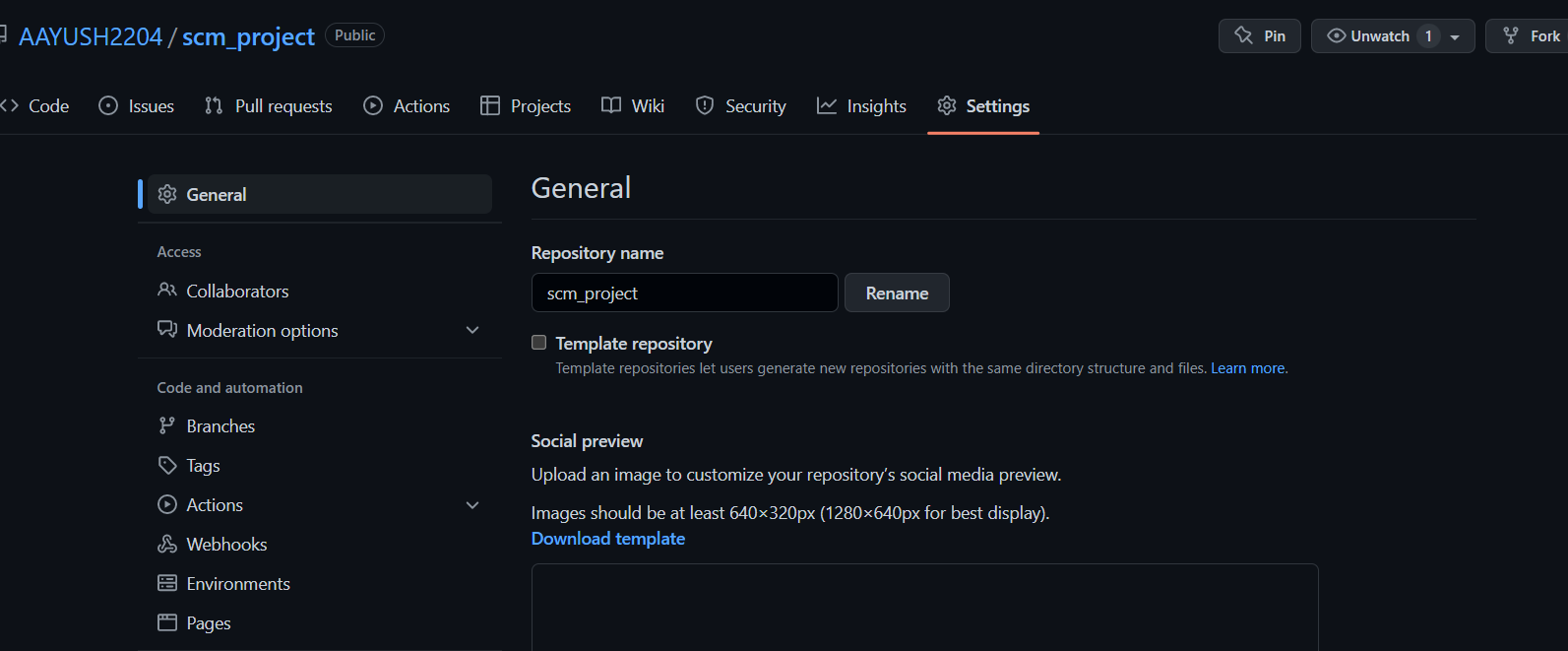
* If you want to import code from an existing repository select the import code option.



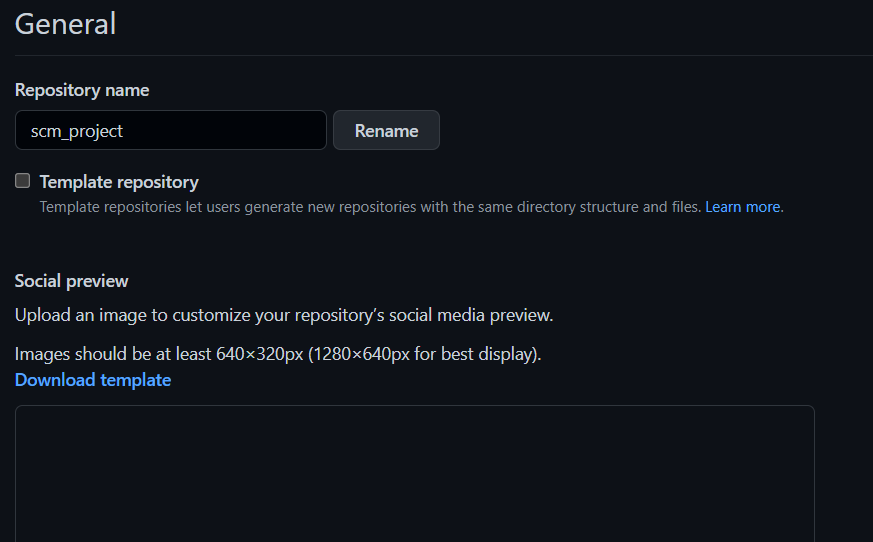
* To create a new file or upload an existing file into your repository select the option in the following box.



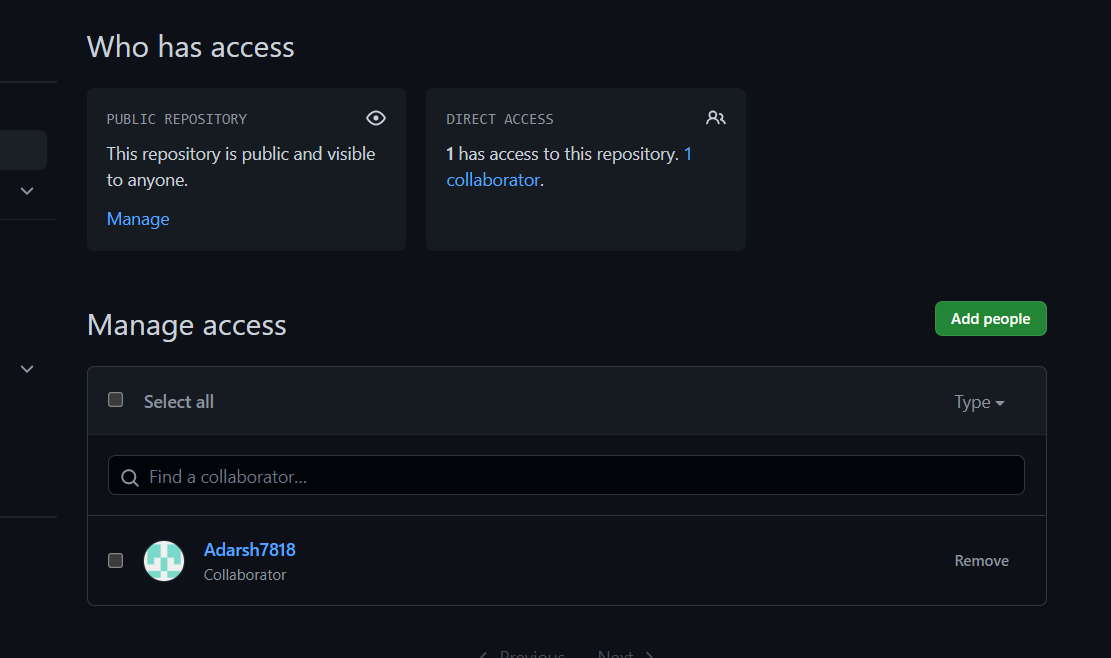
* Now, you have created your repository successfully.
* To add members to your repository open your repository and select settings option in the navigation bar.



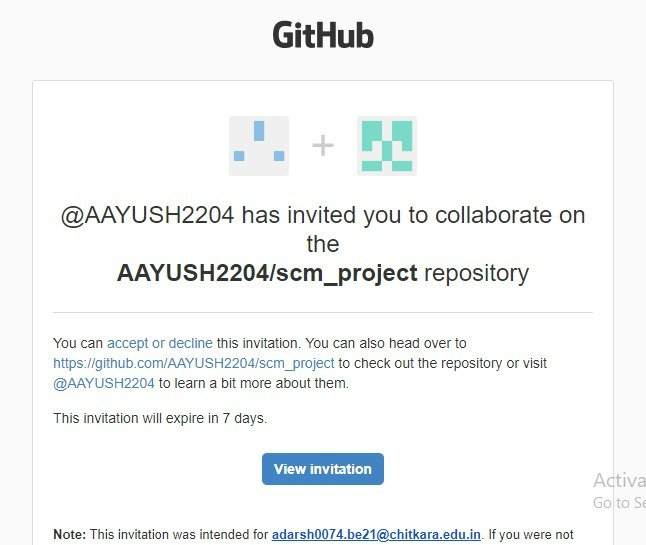
* Click on Collaborators option under the access tab.



* 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.

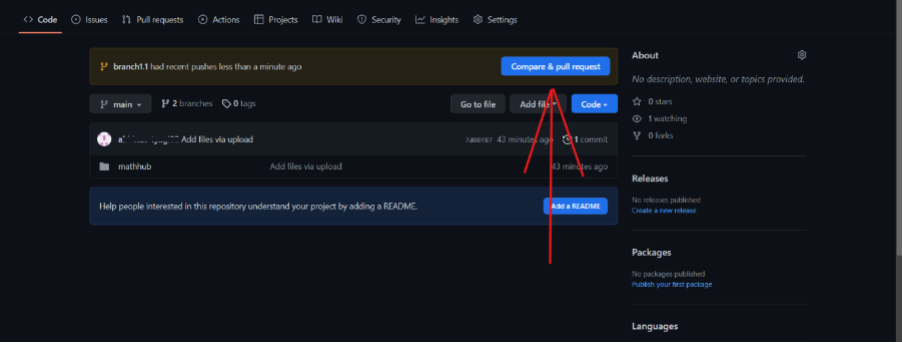


* You will be shown the option that you are now allowed to push.

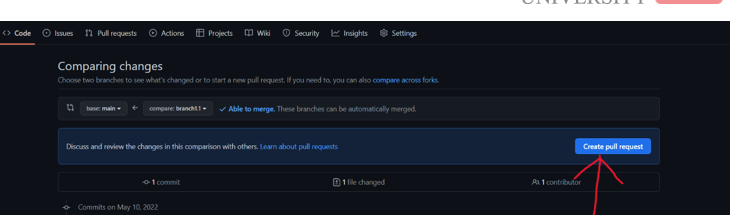
Now all members are ready to contribute to the project.

**Open And Close a Pull Request**

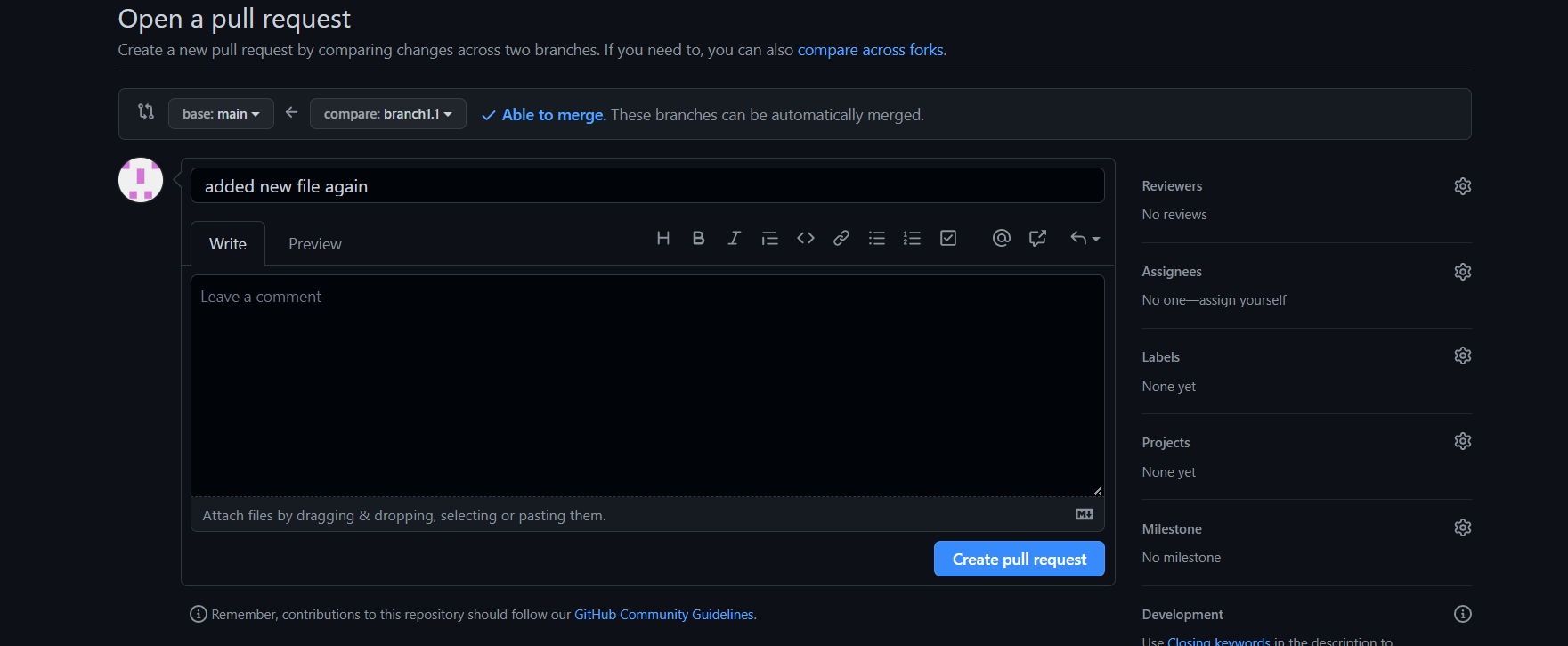
* To open a pull request we first have to make a new branch, by using git branch *branchname* option.
* After making new branch we add a file to the branch or make changes in the existing file.
* Add and commit the changes to the local repository.
* Use git push origin *branchname* option to push the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



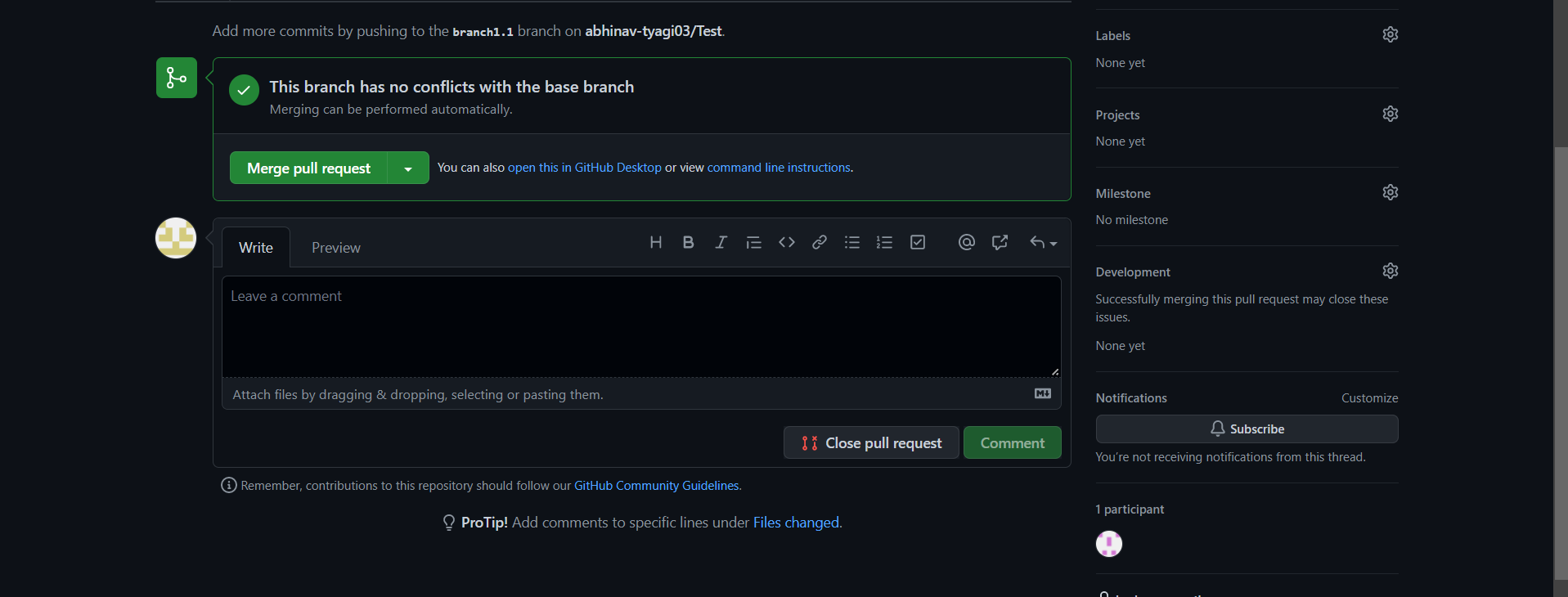
* To create your own pull request click on pull request option.



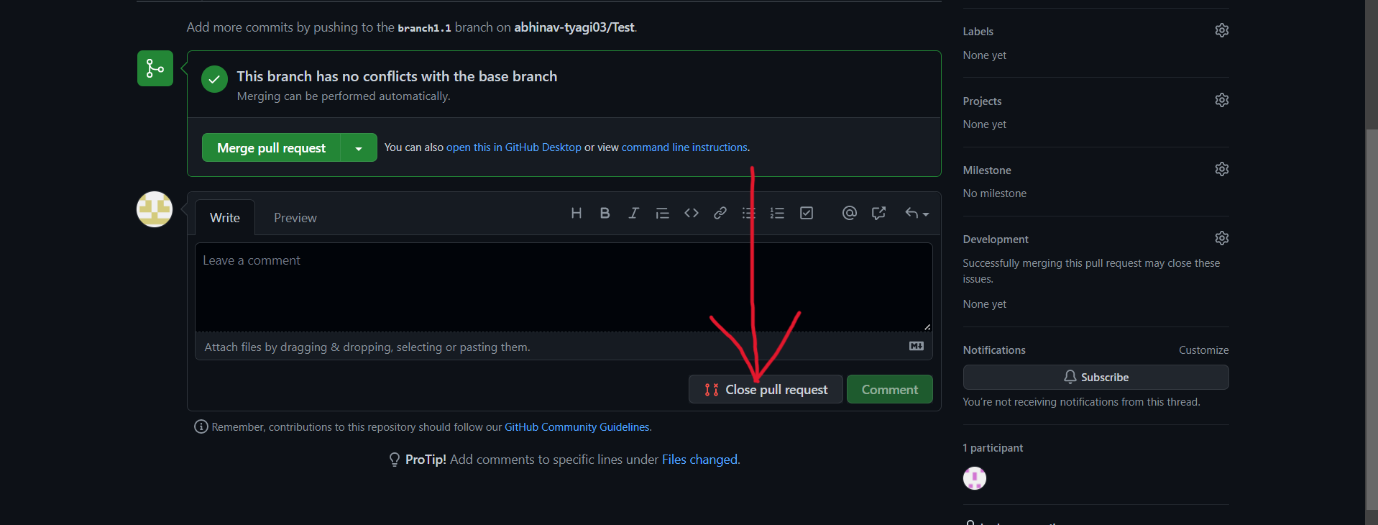
* Github will detect any conflicts and ask you to enter a description of your pull request.



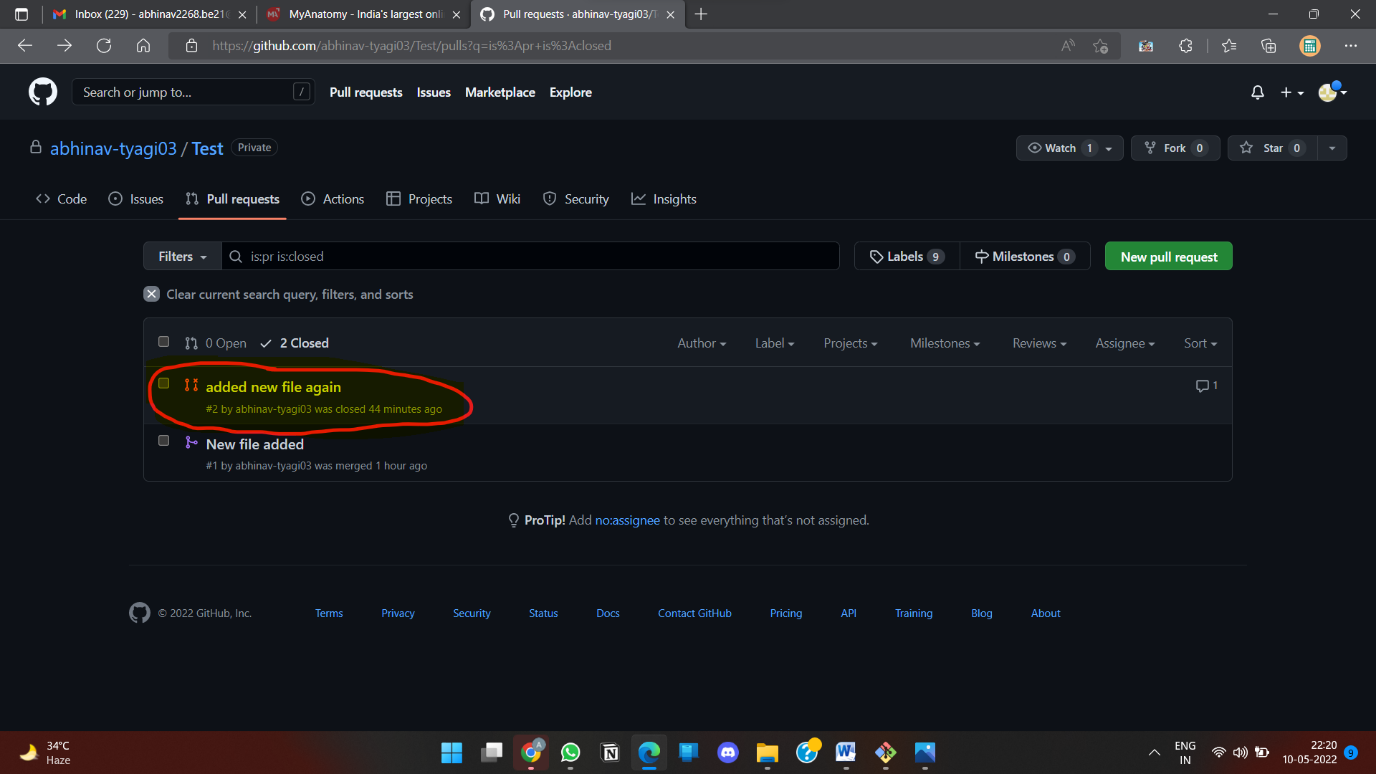
* After opening a pull request all the team members will be sent the request if they want to merge or close the request.



* 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.



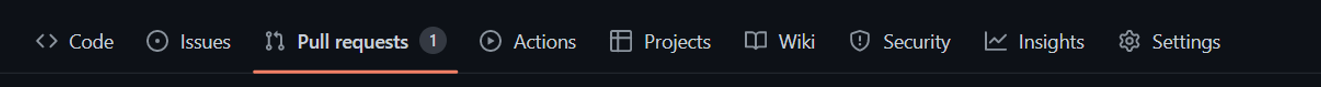
* You can see all the pull request generated and how they were dealt with by clicking on pull request option.



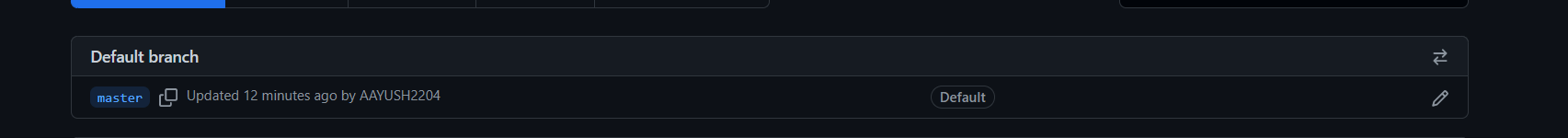
**Create a pull request on a team member’s repo and close pull requests generated by team members on own Repo as a maintainer**

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:-

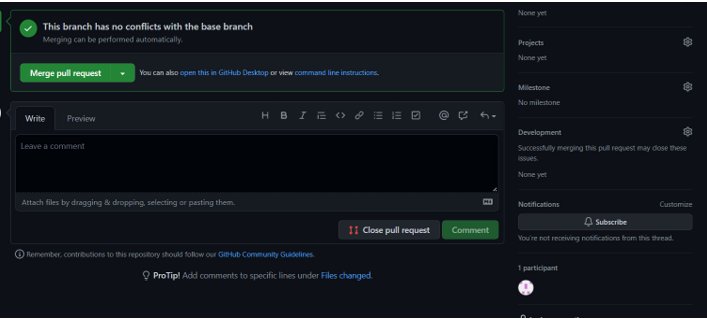
* Do the required changes in the repository, add and commit these changes in the local repository in a new branch.
* Push the modified branch using git push origin *branchname*.
* Open a pull request by following the procedure from the above experiment.
* The pull request will be created and will be 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.

****

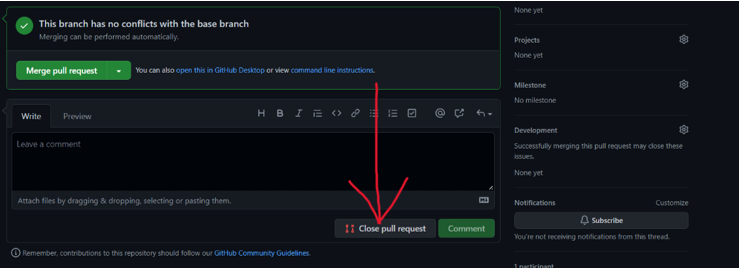
* Click on it. The pull request generated by you will be visible to them.



* 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.

**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:

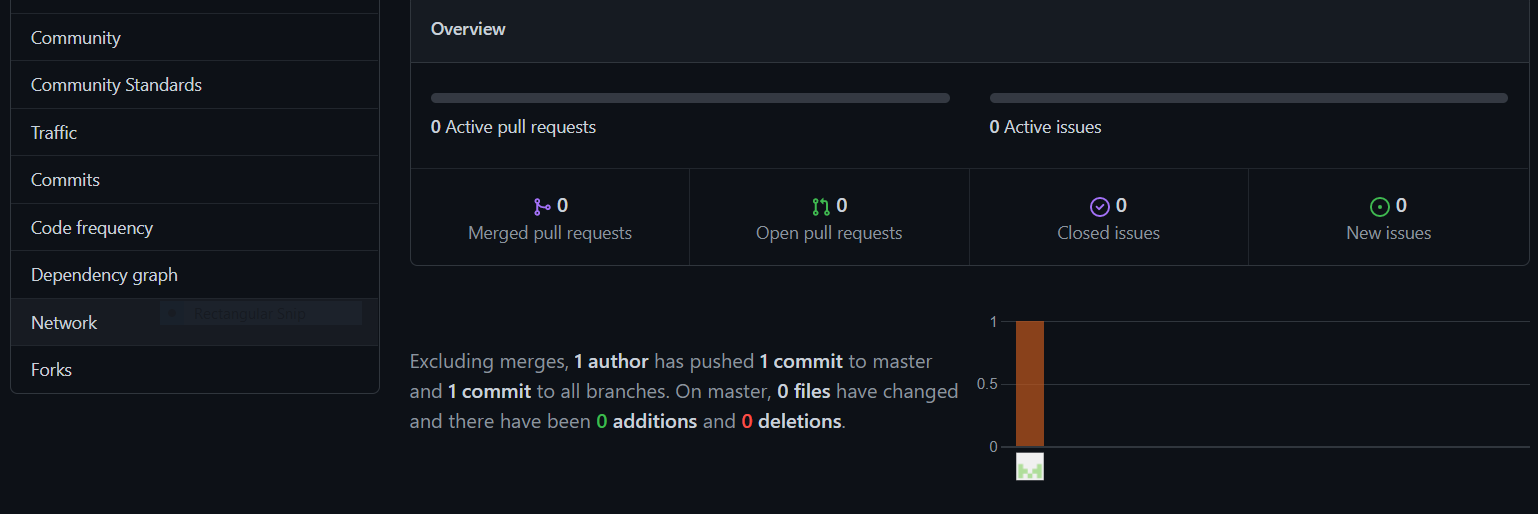
* Pulse
* Contributors
* Traffic
* Commits
* Code frequency
* Network

**Steps to acess network graphs of respective repository**

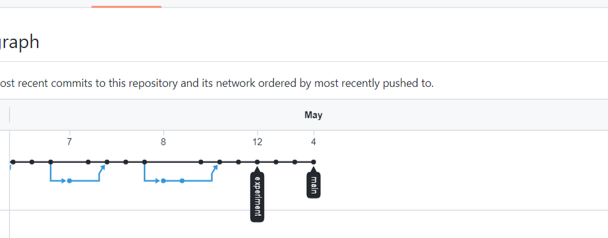
1. On GitHub.com, navigate to the main page of the repository.

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.

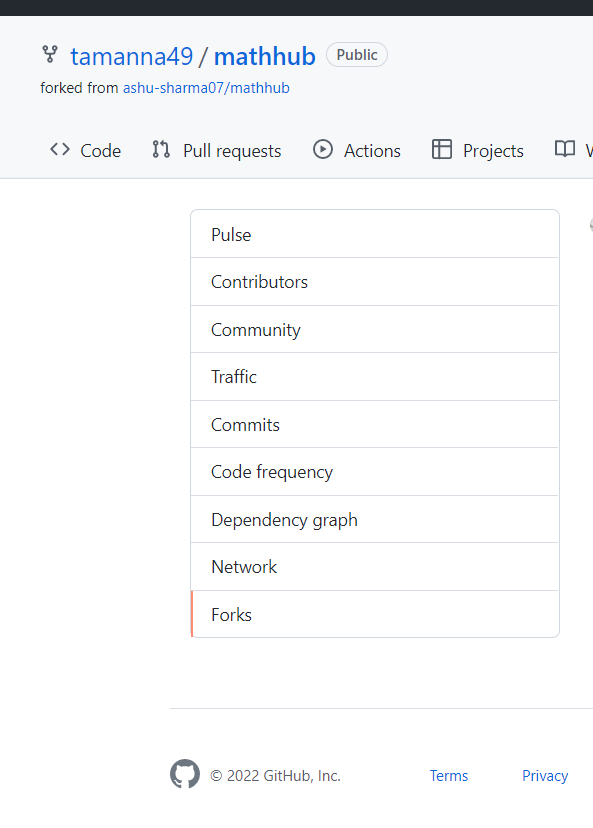


## Listing the forks of a repository

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

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



1. In the left sidebar, click **Forks**.
2. 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.

6. References

1. Progit written by Scott Chacon and Ben Straub
2. GitHub
3. Javapoint
4. Toolsqa.com