A Project Report

on

"Edunation"

with

Source Code Management

(CS181)

Submitted by

Bhaway Khattar   2110992055

Himanshu Garg   2110992044

Udit   2110992026

Vasu Khattar   2110992054

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	**Zeta**	Group No	G27
Course Coordinator	**Dr. Neeraj Singla**		
Faculty Name	**Dr. Sachendra Singh Chauhan**		

Submission

Name:

Signature:

Date:

Table of Content

S. No.	Title	Page No.
1	**Version control with Git**	**4-9**
2	Problem Statement	10
3	Objective	11
4	Resources Requirements – Frontend	12-13
5	Concepts and commands	14-26
6	Workflow and Discussion	27-30
7	Reference	31

Version Control With Git

❖ What is the Version Control System?

A **Version Control System (VCS)** is a tool that helps software developers keep track of how their software development projects desktop applications, websites, mobile apps, etc - change over time.

Each snapshot or state of the files and folders in a codebase at a given time can be called a "version." Version control systems were created to allow developers a convenient way to create, manage, and share those versions. It allows them to have *control* over managing the versions of their code as it evolves over time.

Version control systems also enable collaboration within a team of software developers, without losing or overwriting anyone's work. After a developer makes a set of code changes to one or more files, they tell the version control system to save a representation of those changes.

A version control system can also be referred to as a source control system, source code management, version control software, version control tools, or other combinations of these terms.

❖ Benefits of the Version Control System

The Version Control System is very helpful and beneficial in software development; developing software without using version control is unsafe. It provides backups for uncertainty. Version control systems offer a speedy interface to developers. It also allows software teams to preserve efficiency and agility according to the team scales to include more developers.

Some key benefits of having a version control system are as follows.

- Complete change history of the file
- Simultaneously working
- Branching and merging

❖ History of VCS

There are 3 types of VCS:

1.   Local Version Control System:

In a local version control system, files are simply copied into a separate directory locally. Versions of the same file are stored so as to allow the easy retrieval of any particular version at any point in time. This system is commonly used for small personal projects or files as it provides the facility of versioning your project in an easy manner locally.

Advantages:

- •   Easy to set up
- •   No internet needed
- •   Cheap to run **Disadvantages:**

- •   Error prone
- •   Unsafe (stored locally)
- •   Not suitable for team projects
- •   As data is stored in local machine. If the local machine crashes, it would not be possible to retrieve the files, and all the information will be lost.

2.   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.

Advantages:

- •   Reasonably easy to set up
- •   Various options (proprietary and open source)
- •   Allows for file sharing amongst team members
- •   Project is stored on a more reliable server (possibly cloud)
- •   Admin can control the use and structure of the repository

Disadvantages:

- Single point of failure (if server fails then changes will not be available)
- File conflicts due to updates from different people

3. 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.

Advantages:

- Reliable (everyone has a copy of all versions)
- Allows for file share amongst team members
- Various Options available **Disadvantages:**

- More complex to use/set up
- Heavy on Local Storage

❖ What is Git?

Git is an **open-source distributed version control system**. It is designed to handle minor to major projects with high speed and efficiency. It is developed to coordinate the work among the developers. The version control allows us to track and work together with our team members at the same workspace.

Git was created by **Linus Torvalds** in **2005** to develop Linux Kernel. It is also used as an important distributed version-control tool for **the DevOps**.

❖ Features of Git

Some remarkable features of Git are as follows:

- **Open Source**
  Git is an **open-source tool**. It is released under the **GPL** (General Public License) license.

- **Scalable**
  Git is **scalable**, which means when the number of users increases, the Git can easily handle such situations.

- **Distributed**
  One of Git's great features is that it is **distributed**. Distributed means that instead of switching the project to another machine, we can create a "clone" of the entire repository. Also, instead of just having one central repository that you send changes to, every user has their own repository that contains the entire commit history of the project. We do not need to connect to the remote repository; the change is just stored on our local repository. If necessary, we can push these changes to a remote repository.

- **Security**
  Git is secure. It uses the **SHA1 (Secure Hash Function)** to name and identify objects within its repository. Files and commits are checked and retrieved by its checksum at the time of checkout. It stores its history in such a way that the ID of particular commits depends upon the complete development history leading up to that commit. Once it is published, one cannot make changes to its old version.

- **Speed**
  Git is very **fast**, so it can complete all the tasks in a while. Most of the git operations are done on the local repository, so it provides a **huge speed**. Also, a centralized version control system continually communicates with a server somewhere.
  Performance tests conducted by Mozilla showed that it was **extremely fast compared to other VCSs**. Fetching version history from a locally

  stored repository is much faster than fetching it from the remote server. The **core part of Git** is **written in C**, which **ignores** runtime overheads associated with other high-level languages.
  Git was developed to work on the Linux kernel; therefore, it is

capable enough to **handle large repositories** effectively. From the beginning, **speed** and **performance** have been Git's primary goals.

- Supports non-linear development

Git supports **seamless branching and merging**, which helps in visualizing and navigating a non-linear development. A branch in Git represents a single commit. We can construct the full branch structure with the help of its parental commit.

- Branching and Merging

Branching and merging are the **great features** of Git, which makes it different from the other SCM tools. Git allows the **creation of multiple branches** without affecting each other. We can perform tasks like **creation, deletion,** and **merging** on branches, and these tasks take a few seconds only. Below are some features that can be achieved by branching:

- We can **create a separate branch** for a new module of the project, commit and delete it whenever we want.

- We can have a **production branch**, which always has what goes into production and can be merged for testing in the test branch.

- We can create a **demo branch** for the experiment and check if it is working. We can also remove it if needed.

- The core benefit of branching is if we want to push something to a remote repository, we do not have to push all of our branches. We can select a few of our branches, or all of them together.

- Data Assurance

The Git data model ensures the **cryptographic integrity** of every unit of our project. It provides a **unique commit ID** to every commit through a **SHA algorithm**. We can **retrieve** and **update** the commit by commit ID. Most of the centralized version control systems do not provide such integrity by default.

- Staging Area

The **Staging area** is also a **unique functionality** of Git. It can be

considered as a **preview of our next commit**, moreover, an **intermediate area** where commits can be formatted and reviewed before completion. When you make a commit, Git takes changes that are in the staging area and make them as a new commit. We are allowed to add and remove changes from the staging area. The staging area can be considered as a place where Git stores the changes.

However, Git doesn't have a dedicated staging directory where it can store some objects representing file changes (blobs). Instead of this, it uses a file called index.

Another feature of Git that makes it apart from other SCM tools is that **it is possible to quickly stage some of our files and commit them without committing other modified files in our working directory.**

- **Maintain the clean history**
  Git facilitates with Git Rebase; It is one of the most helpful features of Git. It fetches the latest commits from the master branch and puts our code on top of that. Thus, it maintains a clean history of the project.

Problem Statement

Let's take a scenario to know the importance of git and to know the real power and the importance of git in everyday life of a coder…

Just Imagine you are working on a website. You completed the first version of the project. You decided to modify it. You made a copy of your project folder and started making some changes without risking the integrity of your first version of the project. You made your second version of the project. Similarly, you made many versions of your project just to make it better and better.

Now the problem with this is that copying or making duplicate folders of your project is not an optimized approach. We have to keep in mind the space and time complexities. But with git it becomes simpler to manage space and time as github is like a cloud where you will upload your data

If we are going to host our Education website Website over. So, we don't need to make copies of the project and imagine we made a mistake and we want to go to the previous versions of the project. How will we roll back to a previous version?
Then, we will realize the power and importance of Git and github.

Objective

Following points are the Objectives of Git:

1. **To perform collaborations:** Git keeps track of changes to files and allows multiple users to coordinate updates to those files.

2. **To Track Histories:** Git is used to track changes in the source code.

3. **To enact distributed development:** Git enables the developers to manage the changes offline and allows you to branch and merge whenever required, giving them full control over the local code base.

4. **To perform branching:** Git allows you to develop features, fix bugs, or safely experiment with new ideas in a contained area of your repository.

5. **To Speed up Tasks:** Git tries to minimize latency, which means good perceived performance. Which means it will be easier and faster to enact changes.

Resources Requirements – Frontend

HTML: The Building Blocks of the Internet

HTML stands for HyperText Markup Language. It is a relatively simple language that allows developers to create the basic structure of a website. Even the most complex websites have HTML at their core. It's also the second-most-used programming language by developers, according to a recent Stack Overflow survey.

You may be asking yourself why HTML is called a "markup language." The reason is that instead of using a programming language to perform the desired functions, HTML (like other markup languages) uses tags to annotate, or "mark up," different types of content on a web page and identify the purposes they each serve to the page's overall design. You likely see snippets of HTML more than you even realize. Have you ever noticed text at the bottom of a printed-out email that reads something like " "? That's HTML. A markup language also helps web developers avoid formatting every instance of an item category separately (e.g., bolding the headlines on a website), which saves time and avoids errors.

HTML uses "elements," or tags, to denote things like the beginning of a paragraph, the bolding of a font, or the addition of a photo caption. In this way, it controls how a webpage looks, how the text is separated and formatted, and what the user sees. For people who have never used programming languages before, HTML is an excellent place to start.

CSS

If HTML represents the building blocks of a website, CSS is a way to shape and enhance those blocks. CSS is a style sheet language used to specify the way different parts of a webpage appear to users. In other words, it's a way to add some style and additional formatting to what you've already built with HTML.

For example, perhaps you've used HTML to add header text, and now you want that header to have a more pleasant font, a background color, or other formatting elements that make it more sleek, professional, and stylish. That's where CSS comes in. CSS also helps websites adapt to different device types and screen sizes so that your pages render equally well onsmartphones, tablets, or desktop computers.

To understand the difference between HTML and CSS, it's important to understand their histories. When HTML was invented in 1990, it was only designed to inform a document's structural content (e.g., separating headlines from body text). However, when stylistic elements like fonts and colors were developed, HTML wasn't able to adapt. To solve this issue, CSS was invented as a set of rules that can assign properties to HTML elements, building off of the existing markup language to create a more complex webpage.

JavaScript

JavaScript is the most complex of the three front end languages discussed in this article, building on top of both HTML and CSS. If you're trying to compare the languages, think of it like this: While HTML creates the basic structure for a website, CSS adds style to that structure, and JavaScript takes all of that work and makes it interactive and more functionally complex.

A classic example of how JavaScript works is the menu button that you're used to seeing on the top corner of most websites. You know the one — the three stacked lines that show a list of website sections you can visit when clicked. These buttons and their functionality are all present thanks to JavaScript. It can also help you develop keyboard shortcuts or change the color of a button when a cursor hovers over it.

JavaScript is crucial to all web development. It's supported by all of the modern web browsers, and it is used on almost every site on the web. According to a recent Stack Overflow survey, JavaScript is the most commonly used programming language by developers around the world, with 67.7 percent of developers putting it to use in their work. So, if you're interested in learning web development — whether professionally or even just as a hobby — you'd be smart to learn JavaScript.

Concepts and commands
Task 1 - Add collaborators on GitHub Repository

You can invite users to become collaborators to your personal repository. If you're using GitHub Free, you can add unlimited collaborators on public and private repositories. Repositories owned by an organization can grant more granular access.

 For more information, see "Access permissions on GitHub."

Pending invitations will expire after 7 days, restoring any unclaimed licenses. If you're a member of an enterprise with managed users, you can only invite other members of your enterprise to collaborate with you.

For more information, see "Types of GitHub accounts." Note: GitHub limits the number of people who can be invited to a repository within a 24-hour period. If you exceed this limit, either wait 24 hours or create an organization to collaborate with more people.

1. Ask for the username of the person you're inviting as a collaborator. If they don't have a username yet, they can sign up for GitHub For more information, see "Signing up for a new GitHub account".
2. On GitHub.com, navigate to the main page of the repository.
3. Under your repository name, click Settings.
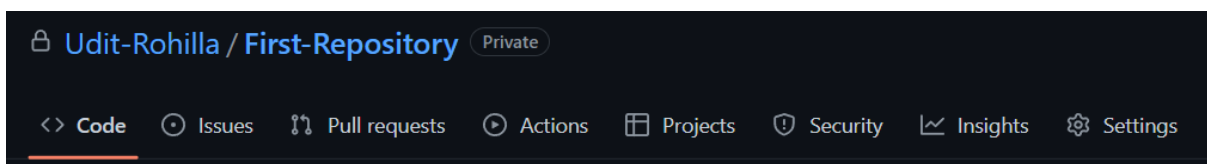
4. In the "Access" section of the sidebar, click Collaborators & teams.
5. Click Invite a collaborator.
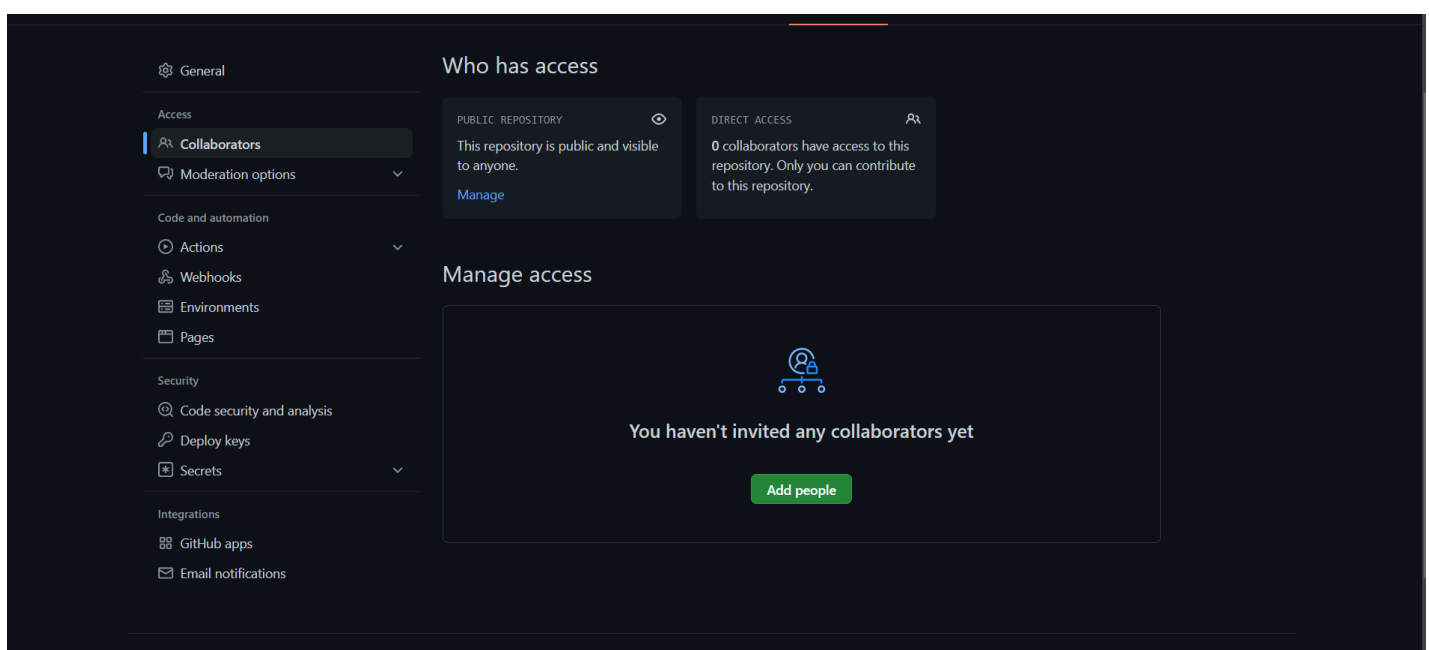
6. In the search field, start typing the name of the person you want to invite, then click a name in the list of matches.
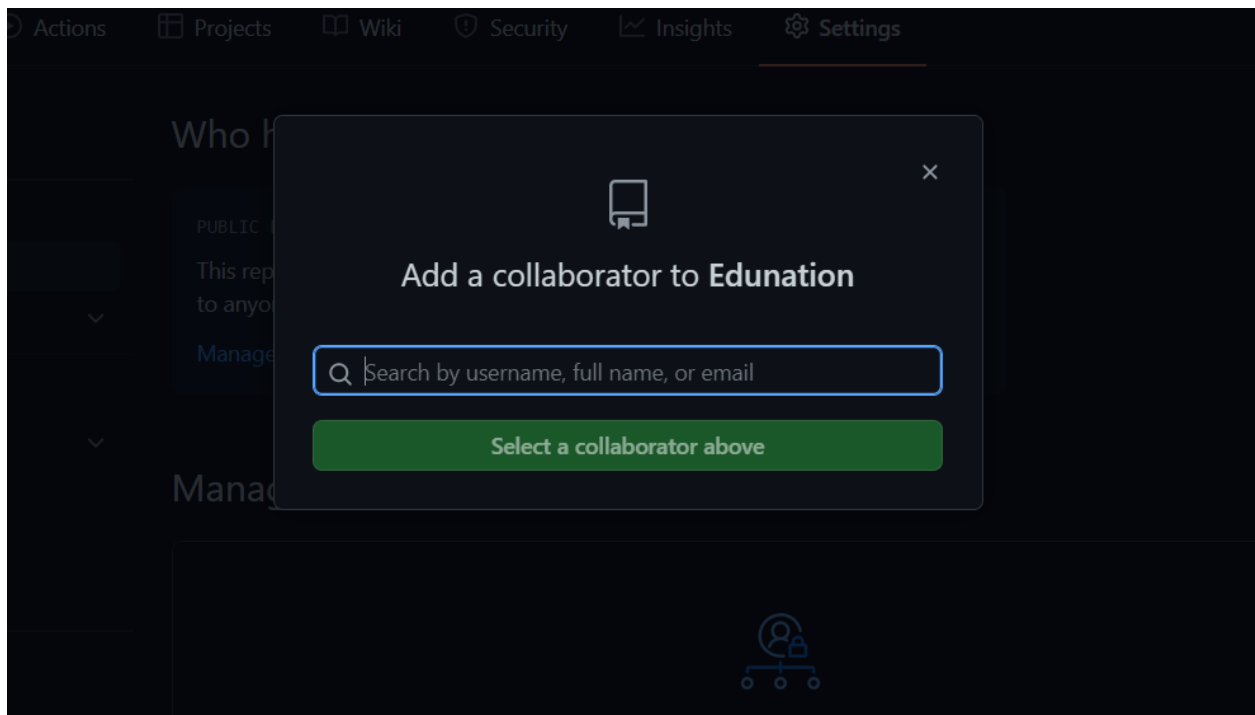
7. Click Add NAME to REPOSITORY.
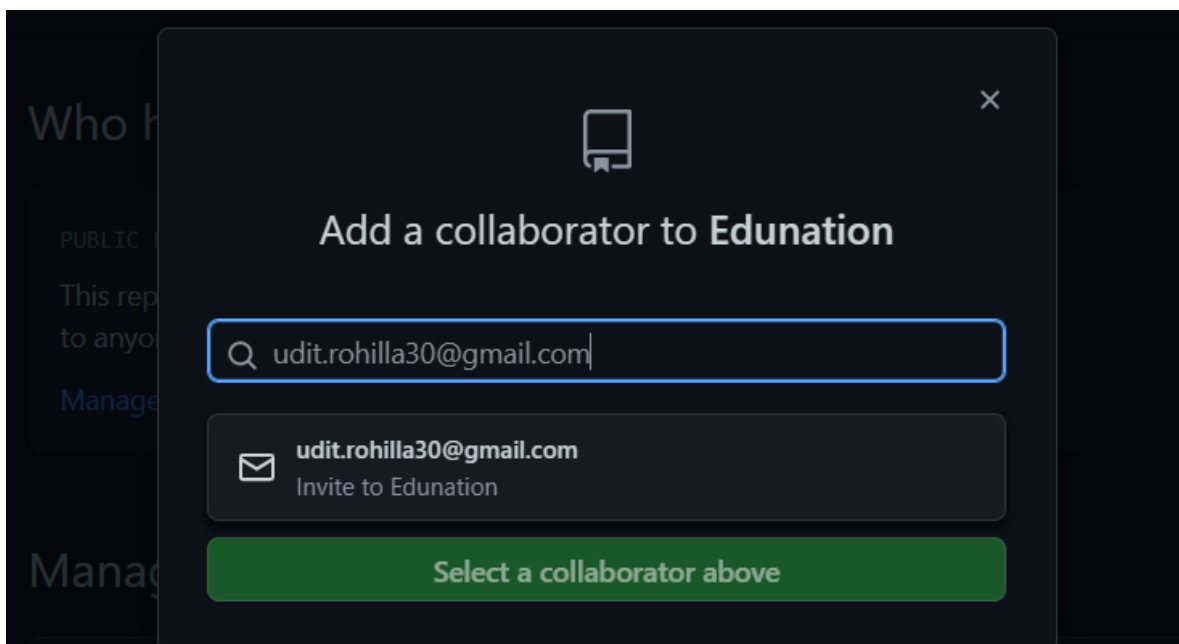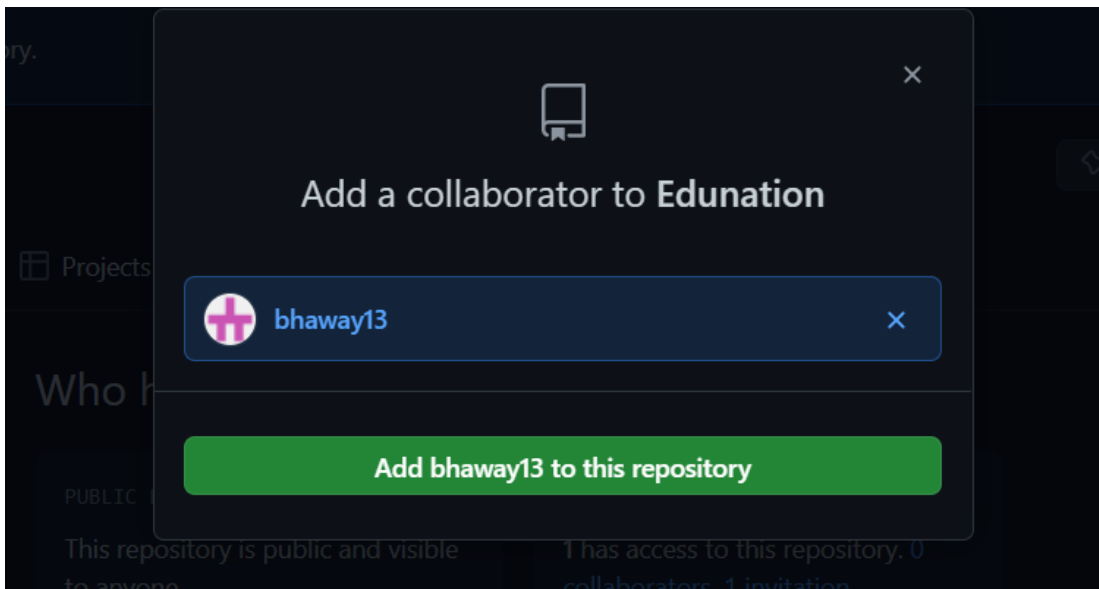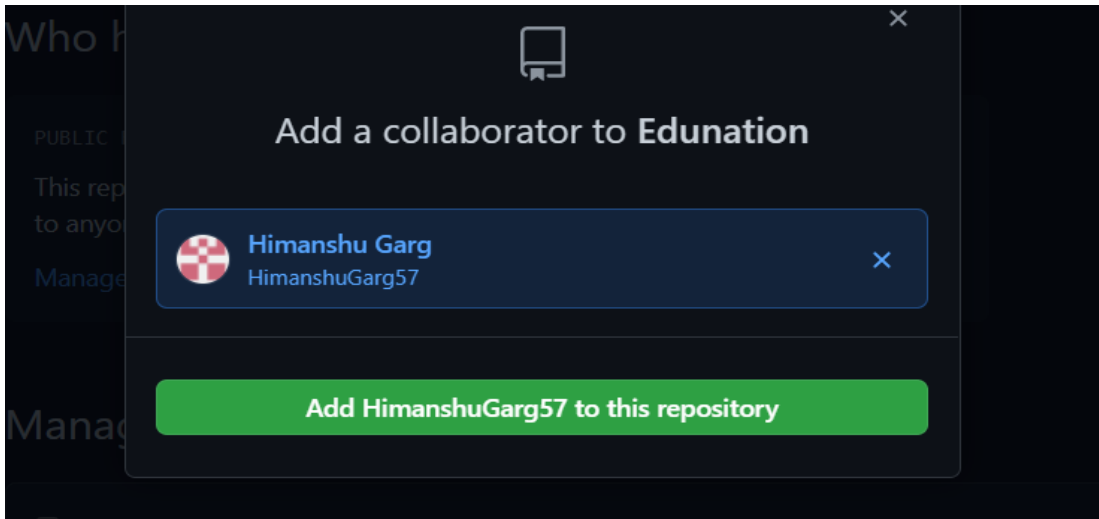
8. The user will receive an email inviting them to the repository. Once they accept your invitation, they will have collaborator access to your repository and after accepting you will see the collaborators name as seen in the image below.
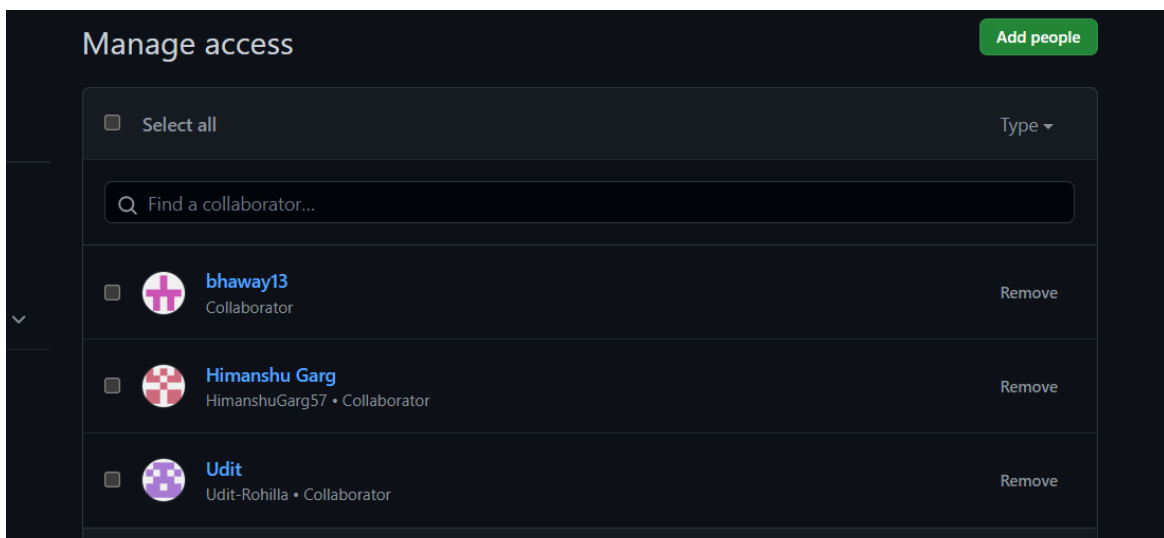
9. Now all members are ready to contribute to the project.

Task 2 - Open and close a pull request.

1. To open a pull request we first have to make a new branch, by using git branch *branchname* option.
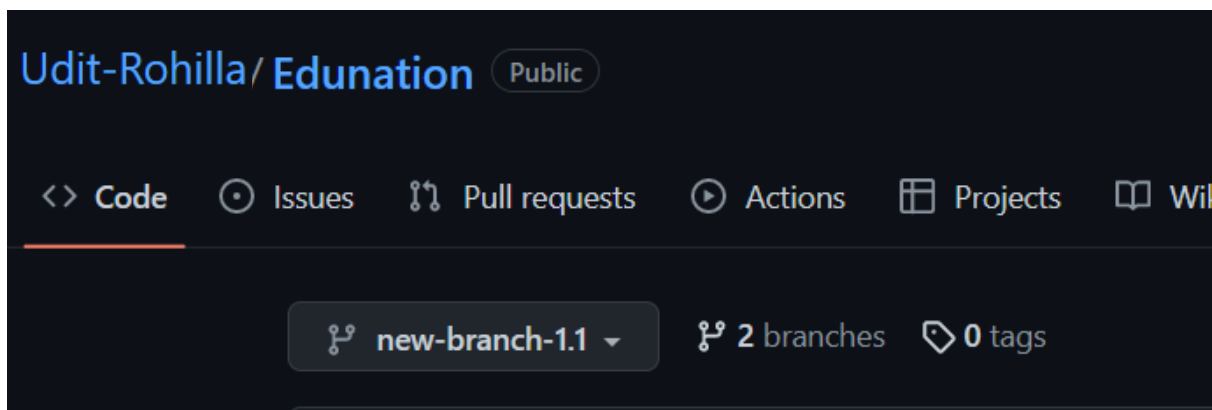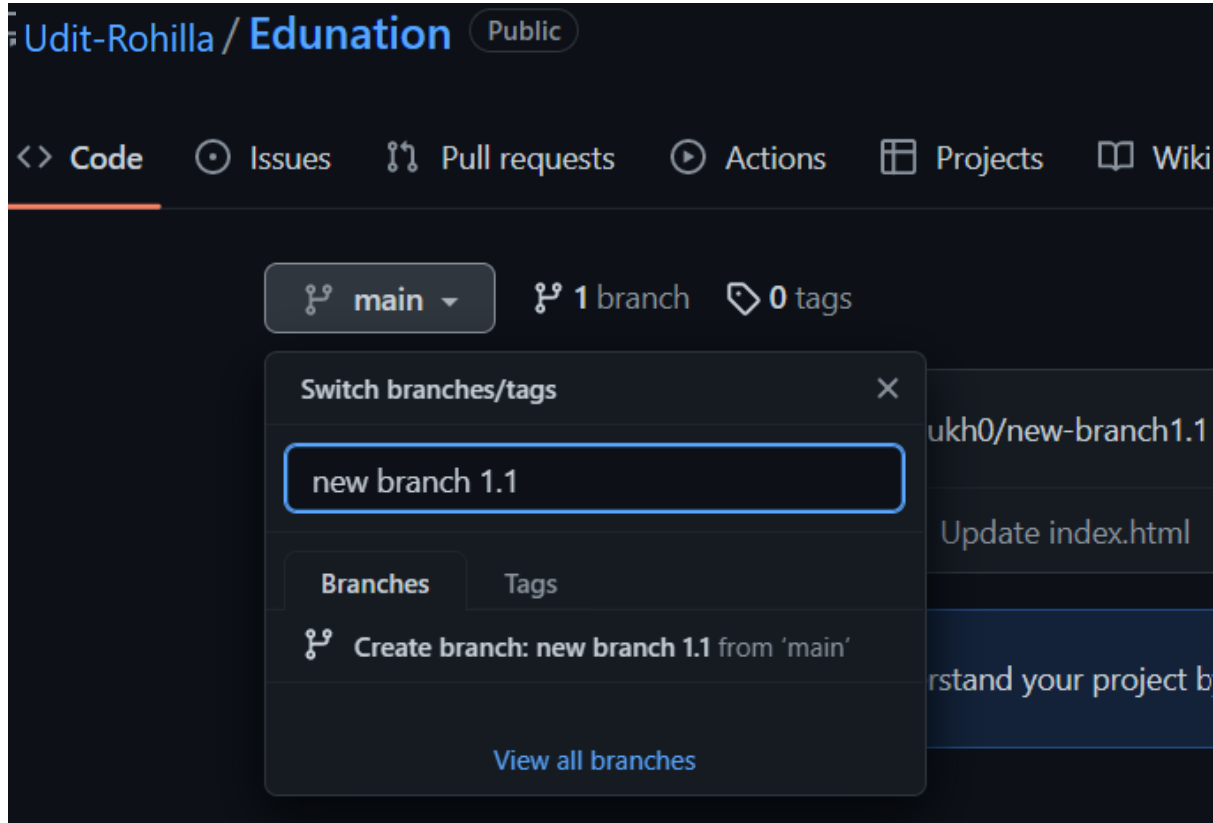
2. After making a new branch we add a file to the branch or make changes in the existing file.
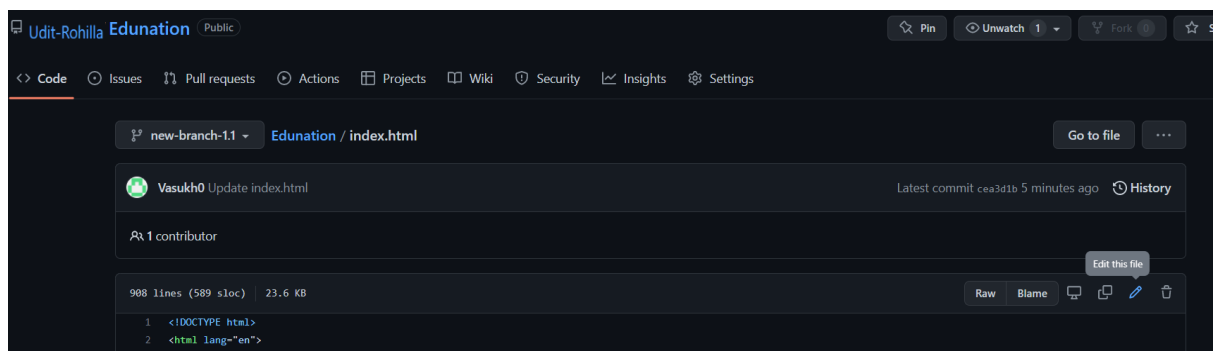
3. Add and commit the changes to the local repository.
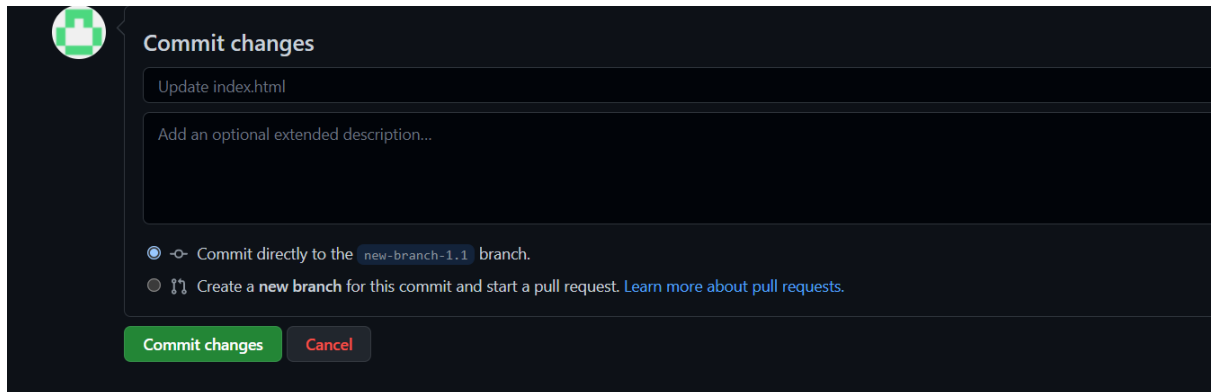
4. After pushing a new branch Github will either automatically ask you to create a pull request or you can create your own pull request.
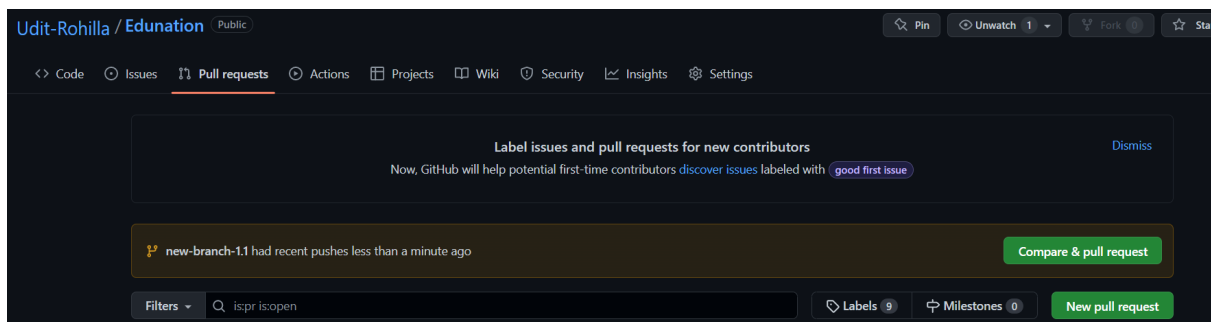
5. To create your own pull request click on pull request option.
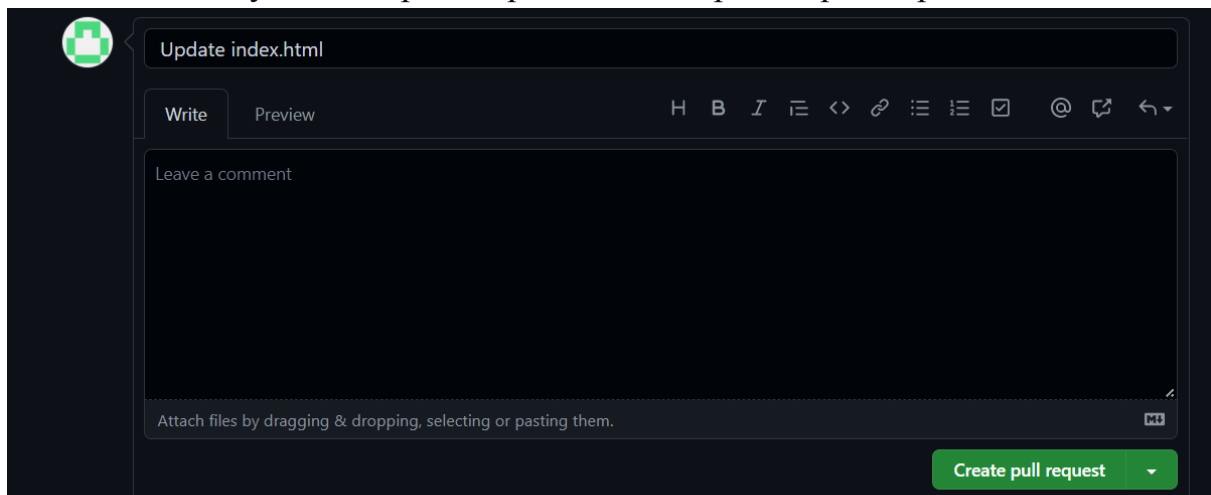
6. Github will detect any conflicts and ask you to enter a description of your pull request.

18
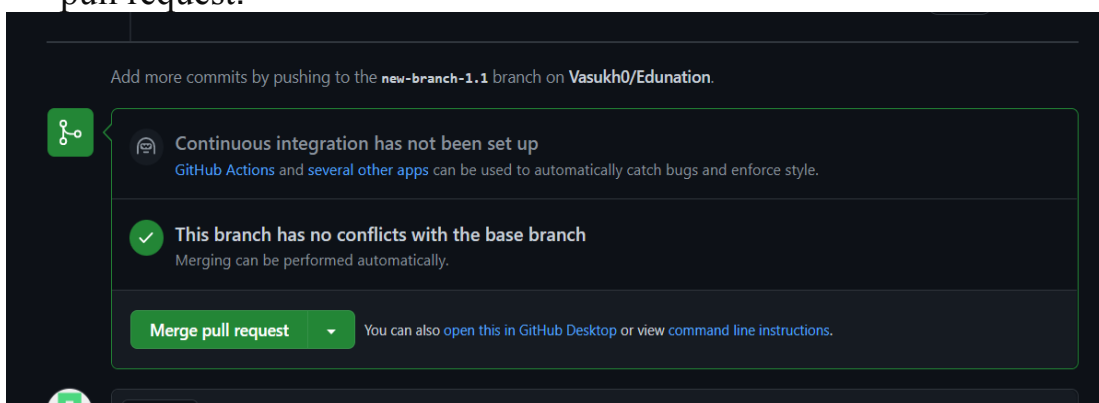
7. After opening a pull request all the team members will be sent the request if they want to merge or close the request.
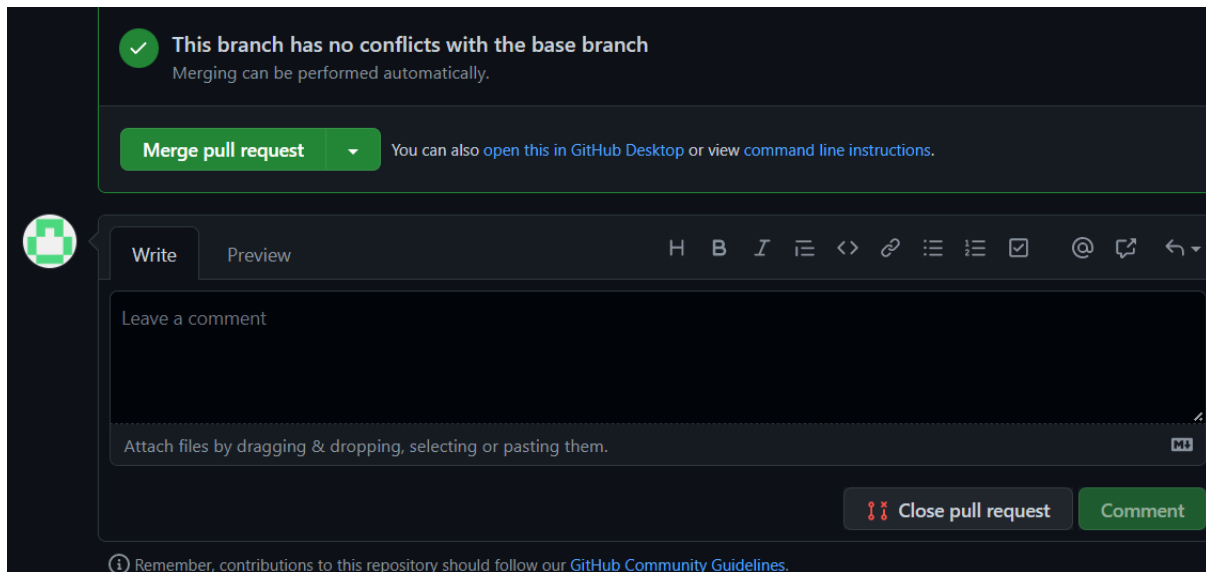
8. If the team member chooses not to merge your pull request they will close your pull request.

9. To close the pull request simply click on close pull request and add comment/ reason why you closed the pull request.

10. You can see all the pull requests generated and how they were dealt with by clicking on pull request option.
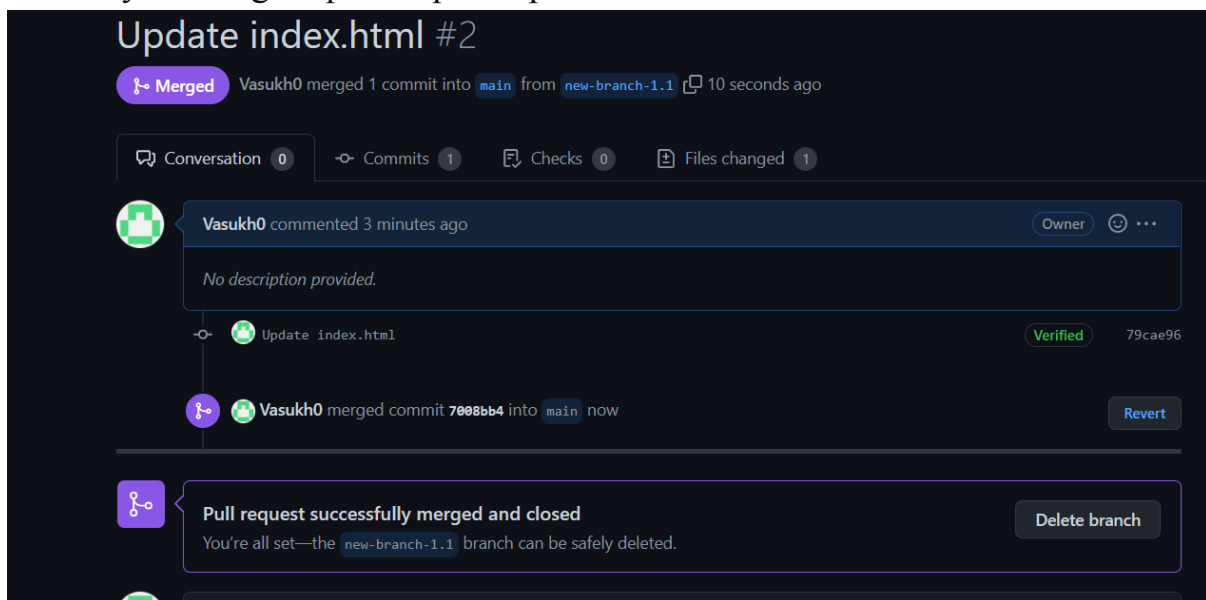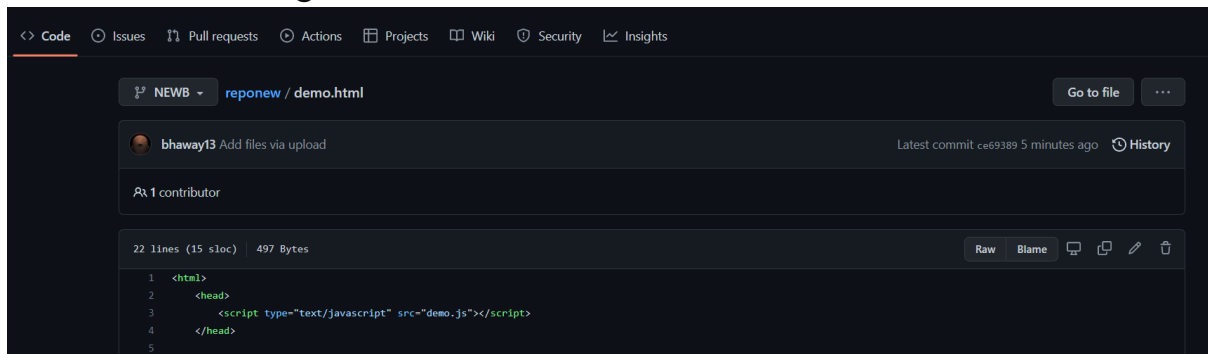
Task 3 - Create a pull request on a team member's repo and Close pull requests generated by team members on your Repo as a maintainer.

<u>Pull request on a team member's Repo</u>

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

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

3. Add and commit the changes to the local repository.

4. Pull request created.

Close requests generated by team members on your Repo

1. Ask your team member to login to his/her Github account.

2. They will notice a new notification in the pull request menu
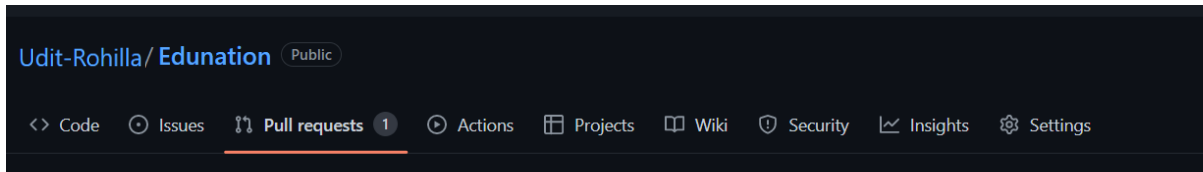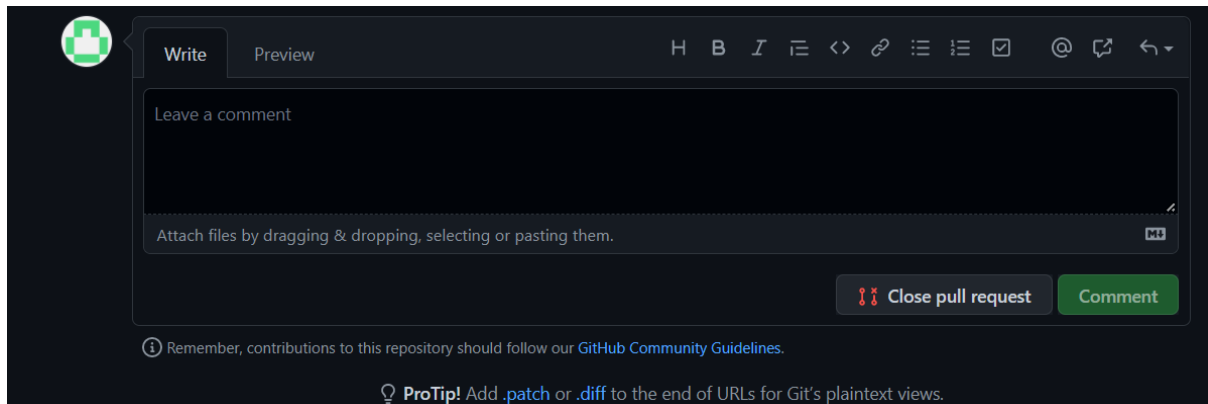

3. Click on it. The pull request generated by you will be visible to them.
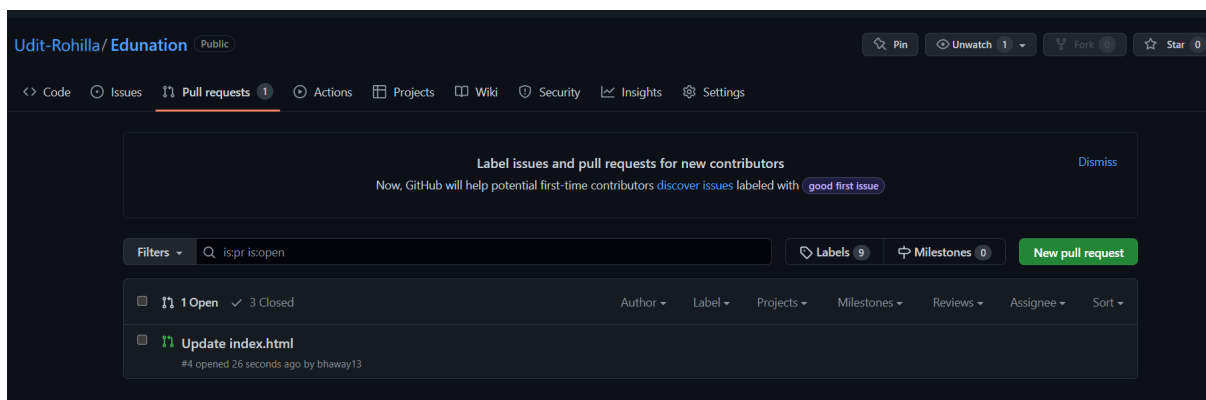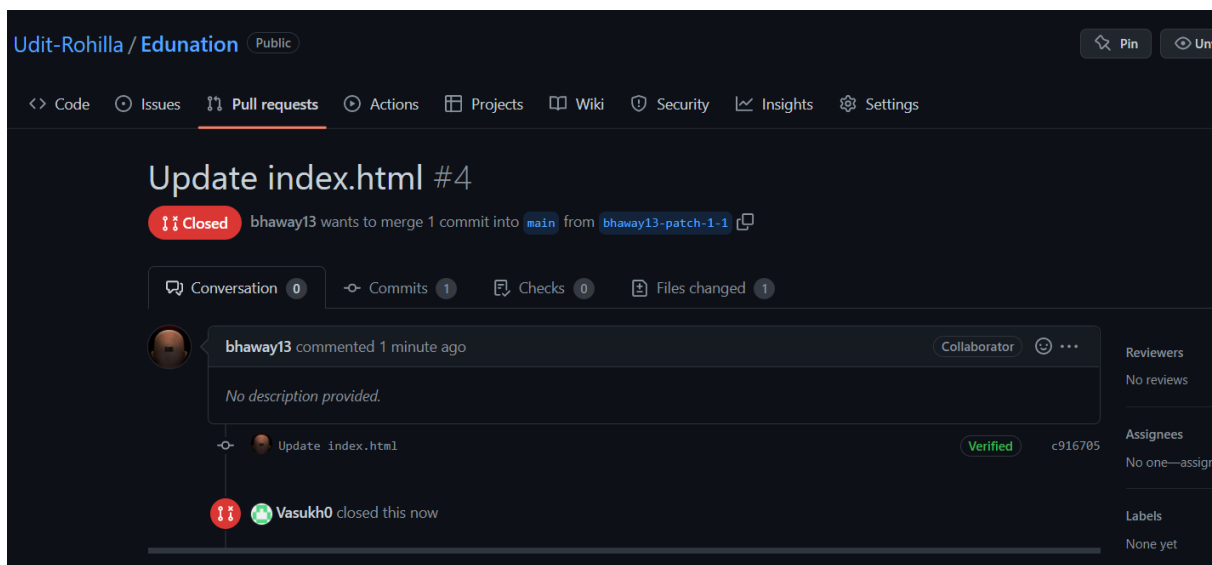

4. Click on it. The pull request generated by you will be visible to them.

5. By selecting the merge branch option the main branch will get updated for all the team members.
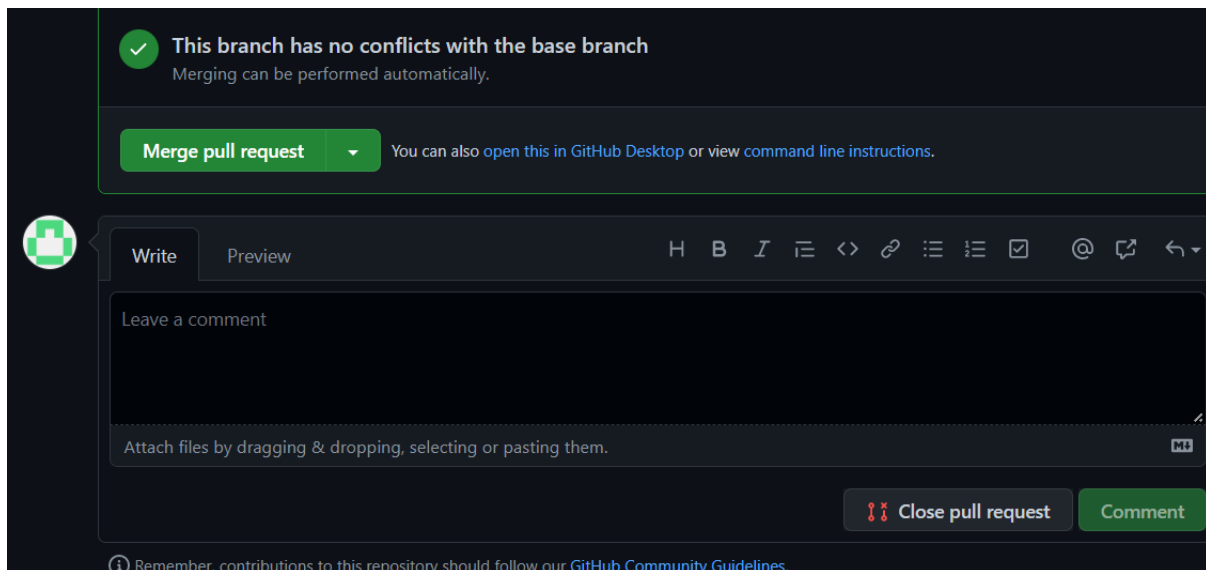
6. By selecting Close the pull request the pull request is not accepted and not merged with the main branch.

7. The result of closing the request is shown below.

Task 4 - 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.

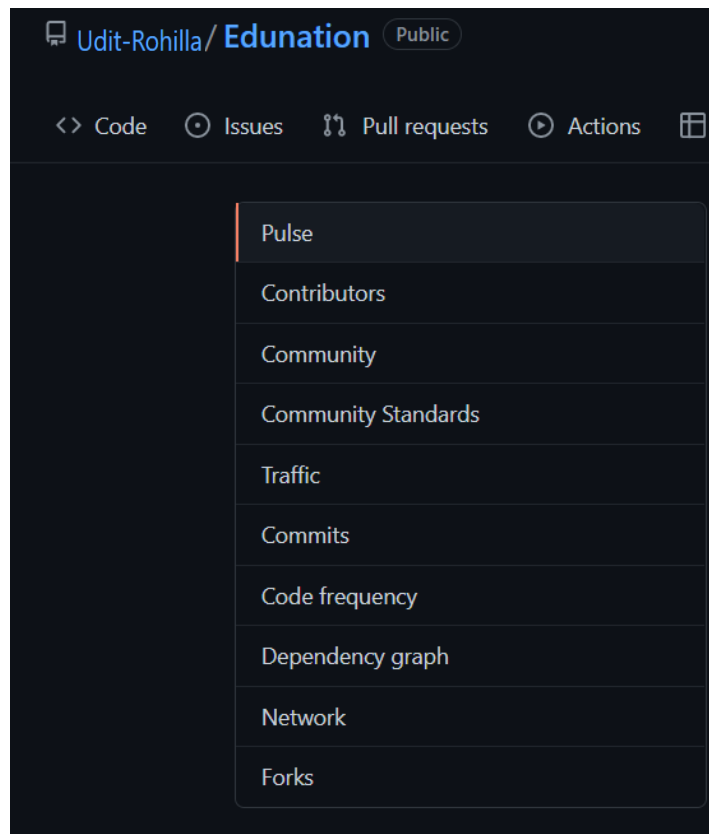Some repository graphs are available only in public repositories with GitHub Free:

- 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**.
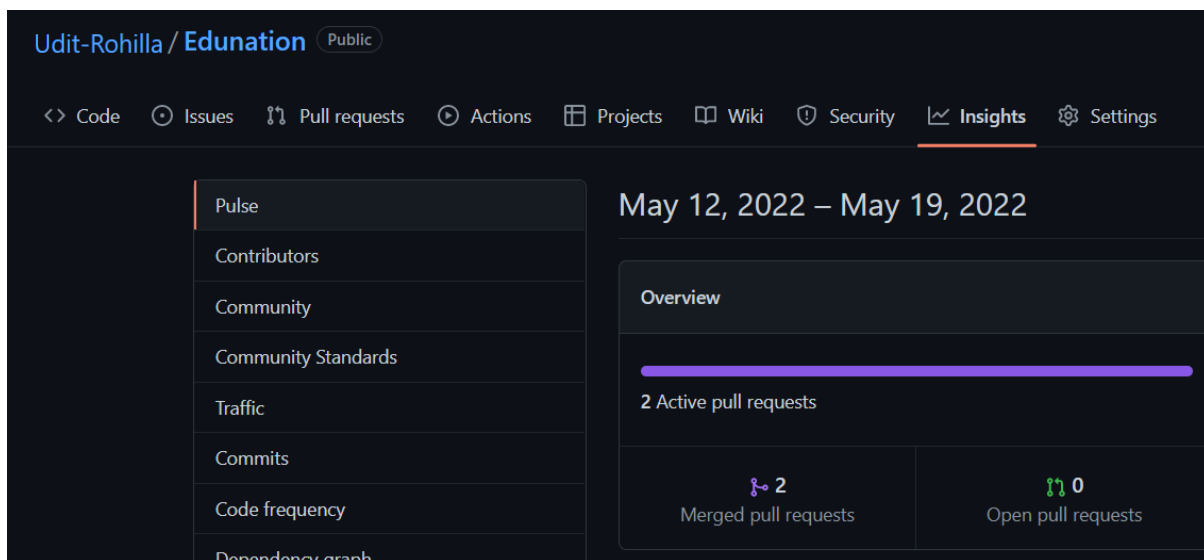
3. At the left sidebar, click on **Network**.

4. 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.
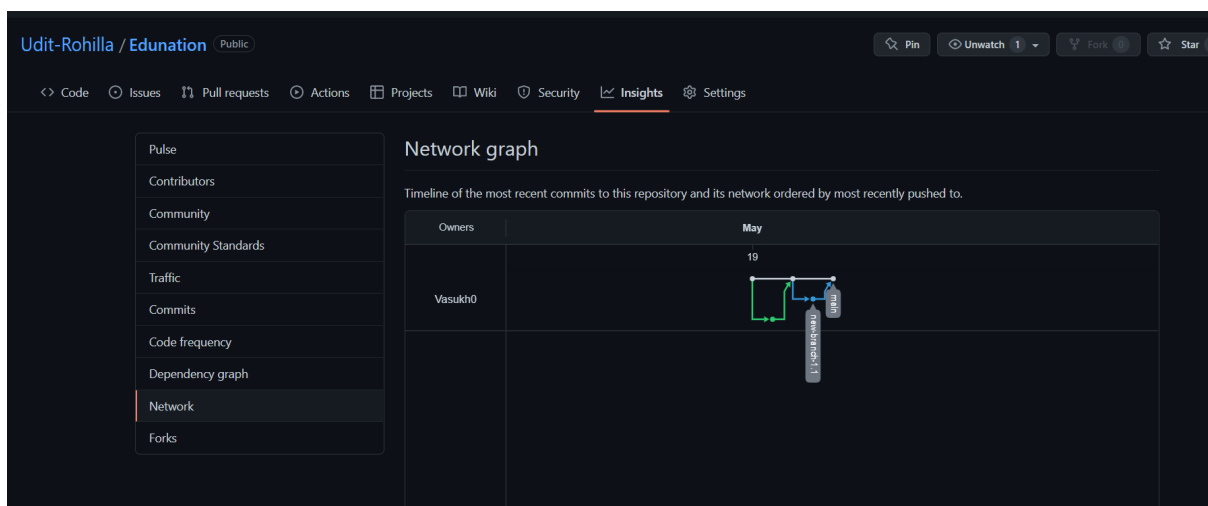
Workflow and Discussion

1. Vasu created a repository by the name Edunation and added all my teammates into the repository (Bhaway , Udit , Himanshu) by clicking on settings < Collaborators < Add a collaborator , then by typing their username into the search box and clicking on add collaborator.
2. Everyone accepted the request on their mail and were added as collaborators in the repository.
3. After that we added the files in the repository and then committed all the files.
4. Then I made a branch and made some changes in the file "index" and committed it and created a pull request.
5. After that a pull request was created and Himanshu got 2 options to merge or to close the pull request. He merged the pull request and code in the index file was changed.
6. Then after discussion Bhaway made changes in the project and added files to the repository. I merged the pull request and all the code was saved.
7. After that Himanshu created a repository and added me , Bhaway and Udit as the collaborators. He added a txt file and added his name , roll no. and group to it.
8. Then I also added my name , roll no. and group to it and created a pull request and merged it and the same was done by Bhaway and Vasu.
9. After that Bhaway made changes to the css file and this created a conflict as I made changes to the file also.
10. I made some changes to the code and merged the pull request.
11. After the merge conflicts got resolved I got the mail regarding it.
12. After that I clicked on Insights < Network to print and publish the network graph of our project repository

Manage access

Add people

☐ Select all

Type ▾

🔍 Find a collaborator...

☐ **bhaway13**
Collaborator

Remove

☐ **Himanshu Garg**
HimanshuGarg57 • Collaborator

Remove

☐ **Udit**
Udit-Rohilla • Collaborator

Remove

Get team access controls and discussions for your contributors in an organization.
NEW Private repos and unlimited members are free.

Create an organization

Udit-Rohilla/ **Edunation** (Public)

<> Code   ⊙ Issues   ⇅ Pull requests   ⊙ Actions   ⊞ Projects   ▭ Wiki

ᛘ main ▾   ᛘ **1 branch**   ⬙ **0 tags**

Switch branches/tags   ✕

new branch 1.1

Branches   Tags

ᛘ **Create branch: new branch 1.1** from 'main'

View all branches

ukh0/new-branch1.1

Update index.html

rstand your project b

Commit changes

Update index.html

Add an optional extended description...

◉ ⟿ Commit directly to the `new-branch-1.1` branch.
◯ ⑂ Create a **new branch** for this commit and start a pull request. Learn more about pull requests.

Commit changes   Cancel

This branch has no conflicts with the base branch
Merging can be performed automatically.

Merge pull request ▾   You can also open this in GitHub Desktop or view command line instructions.

Write   Preview   H B I ☰ ◇ 🔗 ☰ ☰ ☑   @ 🔗 ↩▾

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

⚡ Close pull request   Comment

ⓘ Remember, contributions to this repository should follow our GitHub Community Guidelines.

Update index.html #2

⟫ Merged   Vasukh0 merged 1 commit into `main` from `new-branch-1.1` ⧉ 10 seconds ago

💬 Conversation 0   ⊶ Commits 1   ✔ Checks 0   📄 Files changed 1

Vasukh0 commented 3 minutes ago   Owner ☺ ⋯

No description provided.

⊶ Update index.html   Verified   79cae96

⟫ Vasukh0 merged commit 7008bb4 into `main` now   Revert

⟫ **Pull request successfully merged and closed**   Delete branch
   You're all set—the `new-branch-1.1` branch can be safely deleted.

	NAME	ROLL NO.	GROUP NO.
1	NAME	ROLL NO.	GROUP NO.
2			
3	1.HIMANSHU GARG	2110992044	G27
4	2.VASU KHATTAR	2110992054	G27
5	3.BHAWAY KHATTAR	2110992055	G27-A
6	4.Udit	2110992026	G27-A

Reference

Git Version Control System :

https://www.freecodecamp.org/news/what-is-git-and-how-to-use-it-c341b049ae61/

https://phoenixnap.com/kb/how-git-works

https://bodhizazen.net/git-advantages-and-disadvantages/

https://www.toolsqa.com/git/what-is-git/

Resources Requirements – Frontend:
https://techbootcamps.utexas.edu/blog/html-css-javascript/