

A Project report  
on  
**“RH World”**  
with  
**Source Code Management**  
(CS181)

Submitted by

Hardik	2110992057
Deepanshu	2110992070
Sanjana	2110992028

## **Department of Computer Science & Engineering**

Chitkara University Institute of Engineering and Technology, Punjab

Jan- June  
(2021-22)

Institute/School Name	<b>Chitkara University Institute of Engineering and Technology</b>		
Department Name	<b>Department of Computer Science &amp; Engineering</b>		
Programme Name	<b>Bachelor of Engineering (B.E.), Computer Science &amp; Engineering</b>		
Course Name	<b>Source Code Management</b>	Session	<b>2021-22</b>
Course Code	<b>CS181</b>	Semester/Batch	<b>2<sup>nd</sup>/2021</b>
Vertical Name	<b>Zeta</b>	Group No	<b>G27</b>
Course Coordinator	<b>Dr. Neeraj Singla</b>		
Faculty Name	<b>Dr. Sachendra Singh Chauhan</b>		

Submission

Name:

Signature:

Date:

## Table of Content

<b>S. No.</b>	<b>Title</b>	<b>Page No.</b>
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 on smartphones, 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

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

### Distributed Repository:

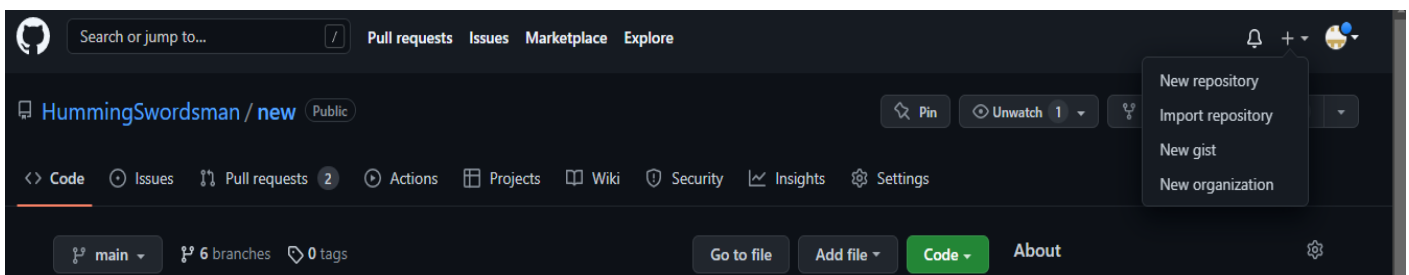
Now let's discuss how to work with the distributed approach. While in centralized approach, everyone on team cloned the single central repository, here in distributed approach everyone clones '*a copy of the source repository*' also known as '*forking*'.

In this approach, all the collaborators will 'fork' the source repo. Upon forking ever member gets their own copy of the source repository in their GitHub account. Collaborators can then go ahead and clone their 'forked repository' on their local machines.

### Steps of creating Distributed Repository:

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

1. Went to Git hub and created new Repository, Click on new sign on top left side .



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

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner \* Repository name \*

HummingSwordsman / origin

Great repository names are short and memorable. Need inspiration? How about **supreme-train**?

Description (optional)

☒ **Public**  
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**  
You choose who can see and commit to this repository.

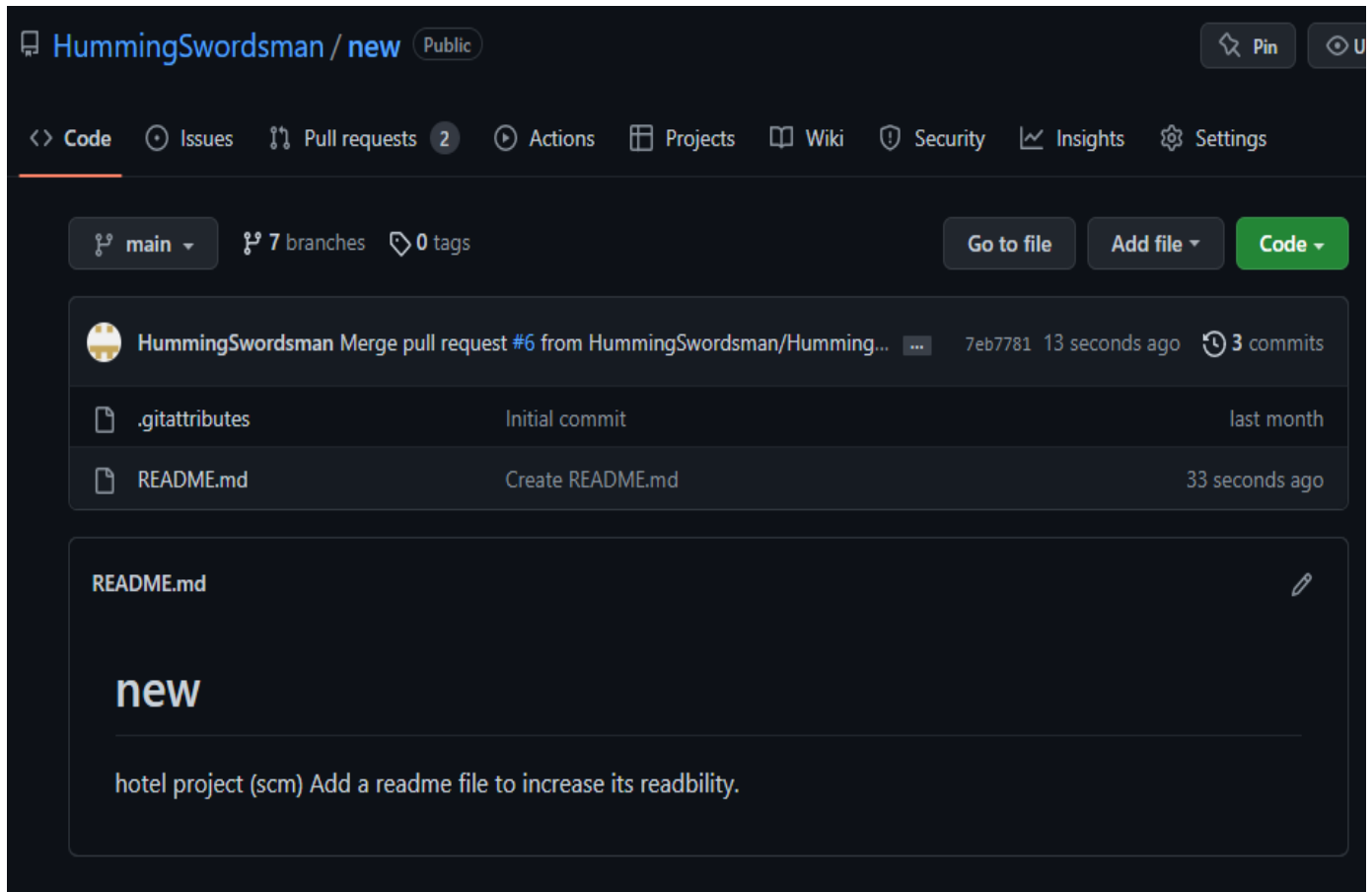
Initialize this repository with:

Skip this step if you're importing an existing repository.

☐ **Add a README file**  
This is where you can write a long description for your project. [Learn more.](#)

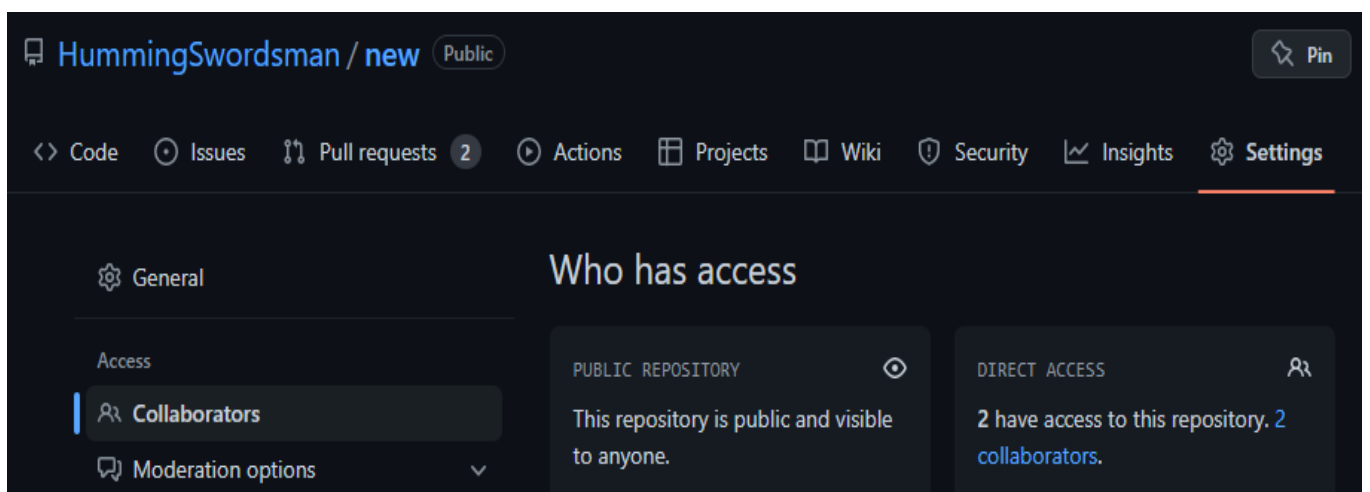
3. Now you can see the file created on that name with a default readme file. You can add content to the readme file. Now the next step is to invite the collaborators to the repository. currently, you have your repository ready to collaborate. Link to Git main Repository:

[HummingSwordsman/new: hotel project \(scm\) \(github.com\)](https://github.com/HummingSwordsman/new:hotel_project_scm)



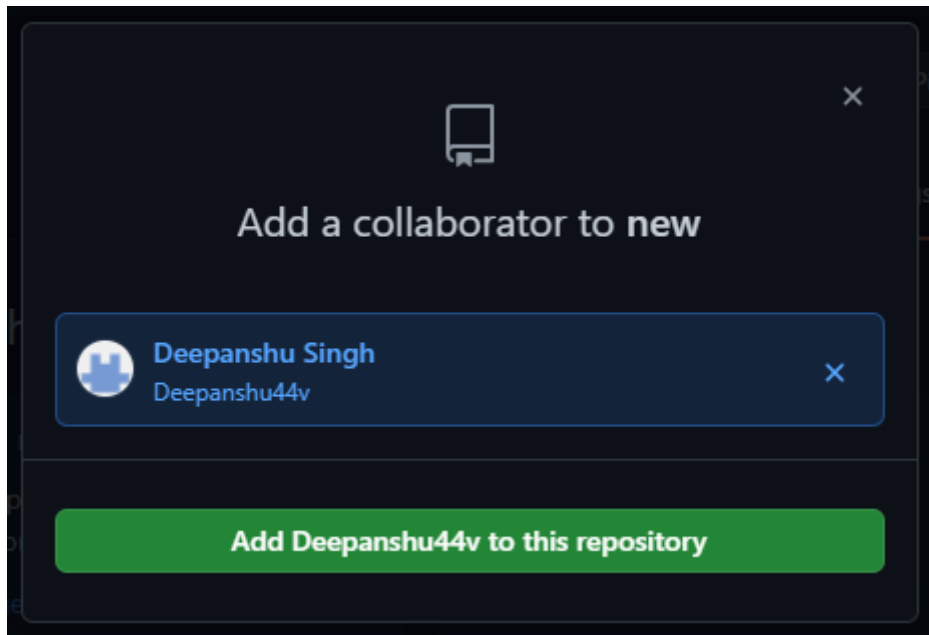
**GitHub collaboration** is a space where you can invite another developer to your repository and work together at an organization level, splitting the task and 2nd person will have the rights to add or merge files to the main repository without further permission. Also, refer to this [Github Beginner guide](#). Let's look into how to add collaborators into Github.

1. Now click on **settings**, you should be the one who created the repository to do this work, because only the author can send the invitation to others.



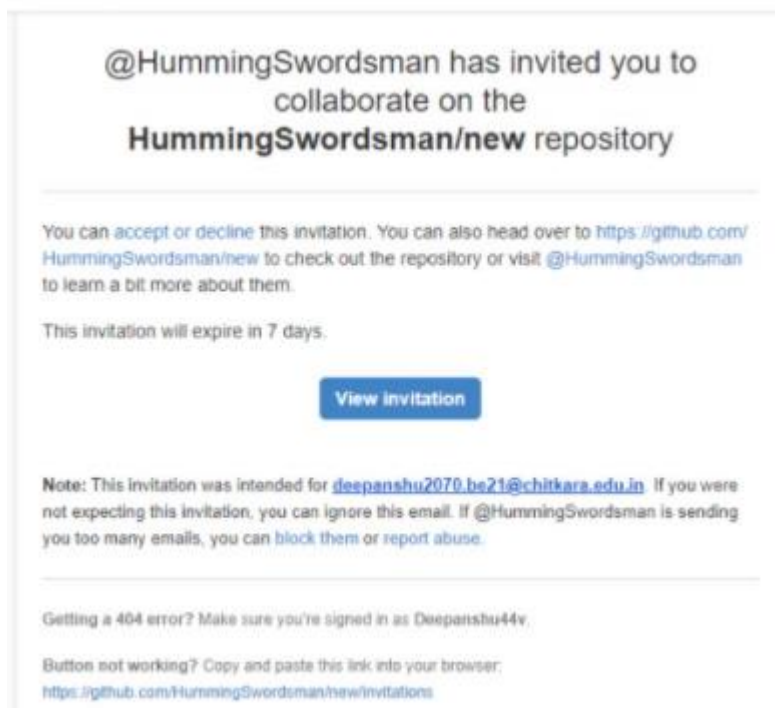


2. Click on the invited collaborator and enter the email ID of people you want to add to this repo. Email will go to them to accept the request.

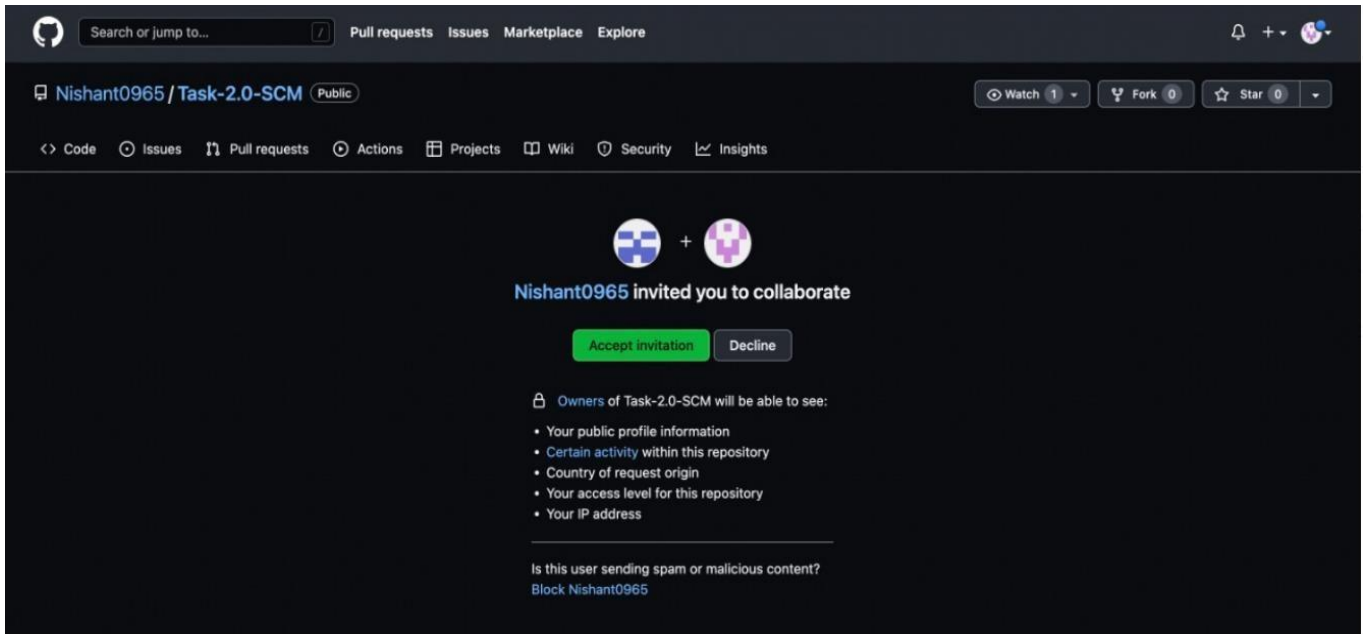


Once it accepted, they can push the changes to main branch.

3. The person will be getting an email invitation like this from GitHub collaboration, click on the Email View and accept invitation.



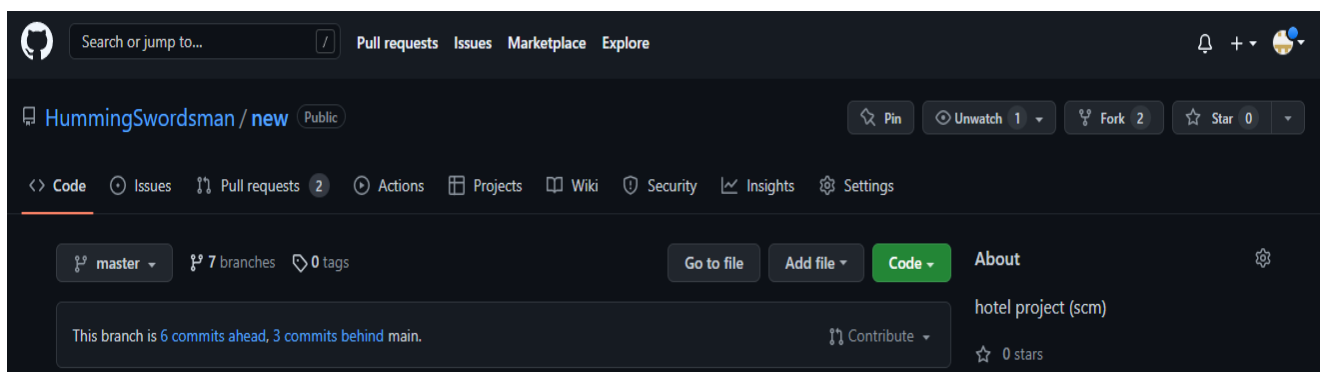
4. You will be redirected to GitHub Windows there click on Accept Invitation.



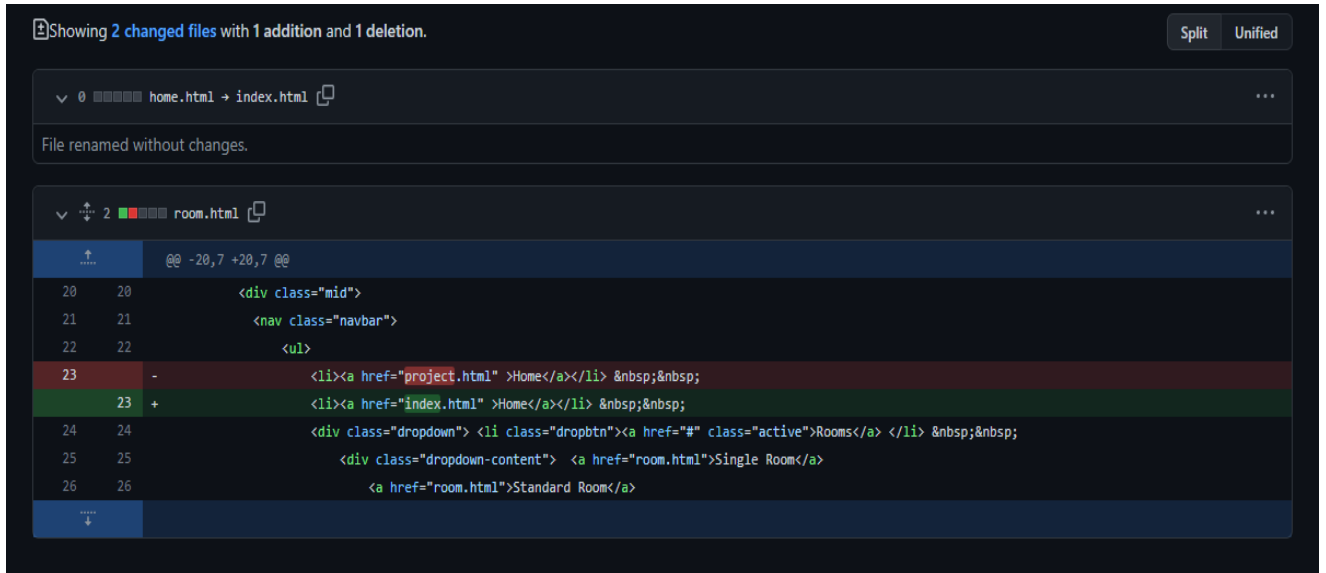
Now all the collaborators will **'fork'** the source repo.

Most commonly, forks are used to either propose changes to someone else's project to which you do not have write access, or to use someone else's project as a starting point for your own idea. You can fork a repository to create a copy of the repository and make changes without affecting the upstream repository

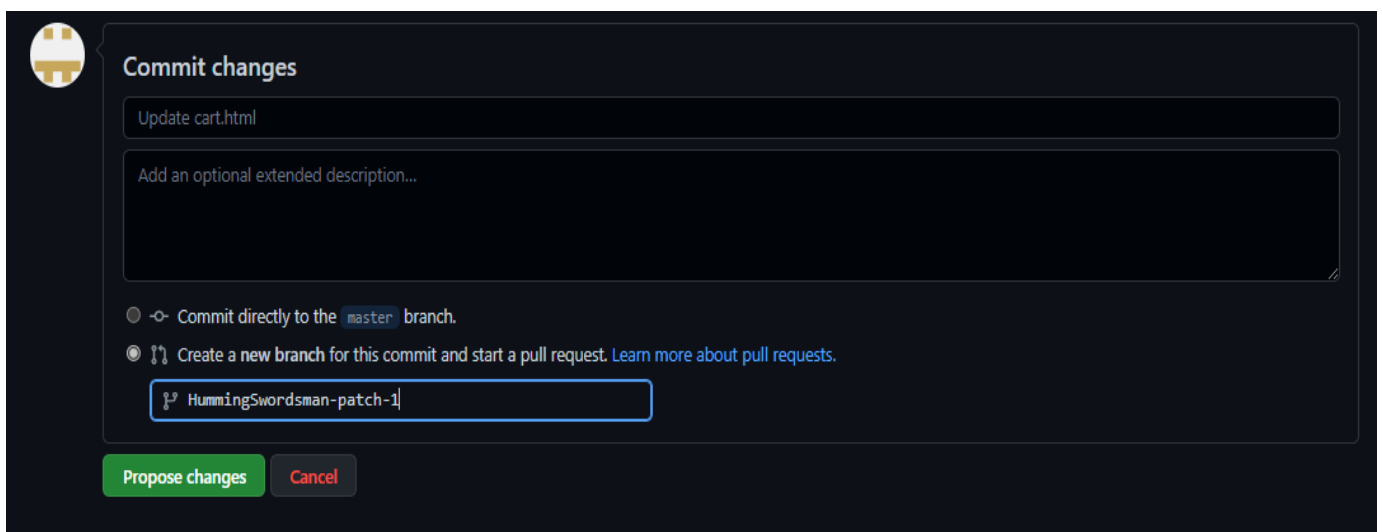
1. On GitHub.com navigate to [HummingSwordsman/new: hotel project \(scm\) \(github.com\)](https://github.com/HummingSwordsman/new:hotel-project-scm) Repository.
2. In the top-right corner of the page, click **Fork**.



Whatever changes are made to "Deepanshu44v/new" and "sanajaaaaaa17/new" will now change the original "HummingSwordsman/new".



We can make my changes here and then make a Pull Request to the maintainers of the project. Now it is in their hand if they will accept or reject your changes to the main project.



After this, we will see how to work in the forked repositories which were forked by the collaborators. If the collaborator tries to change in his forked repository, it will not affect the main repository.

Upon forking every member gets their own copy of the source repository in their GitHub account. Collaborators can then go ahead and clone their ‘forked repository’ on their local machines.

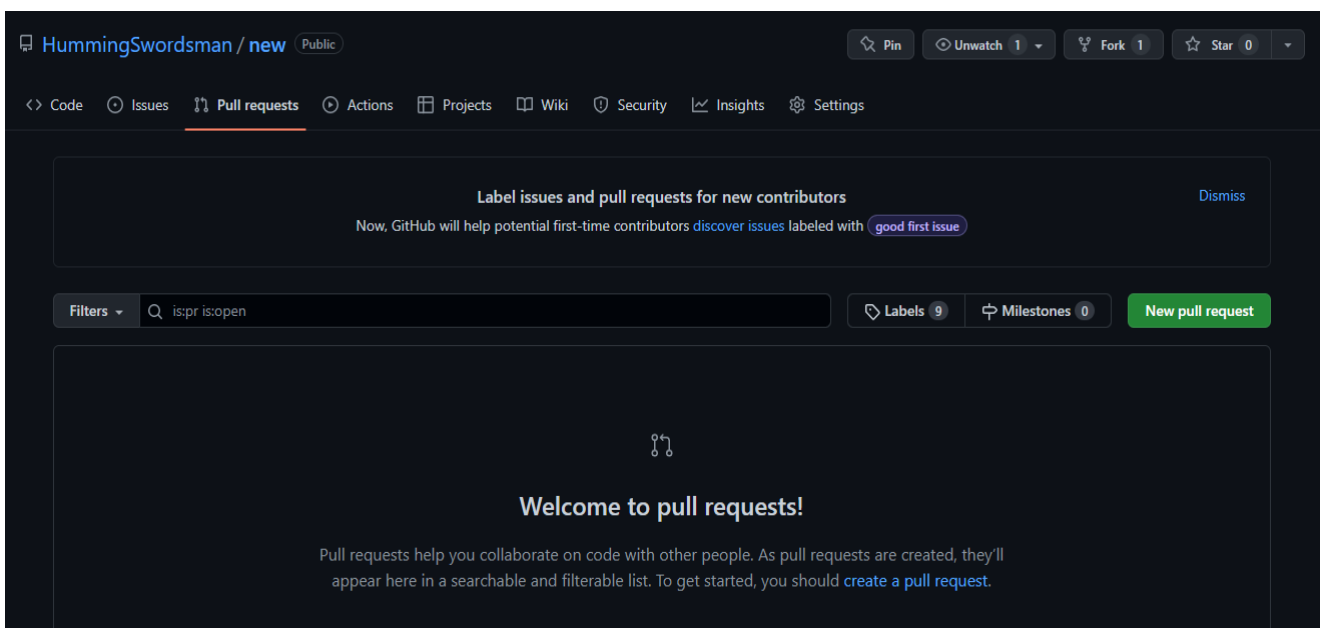


## Task 2 - Open and close a pull request.

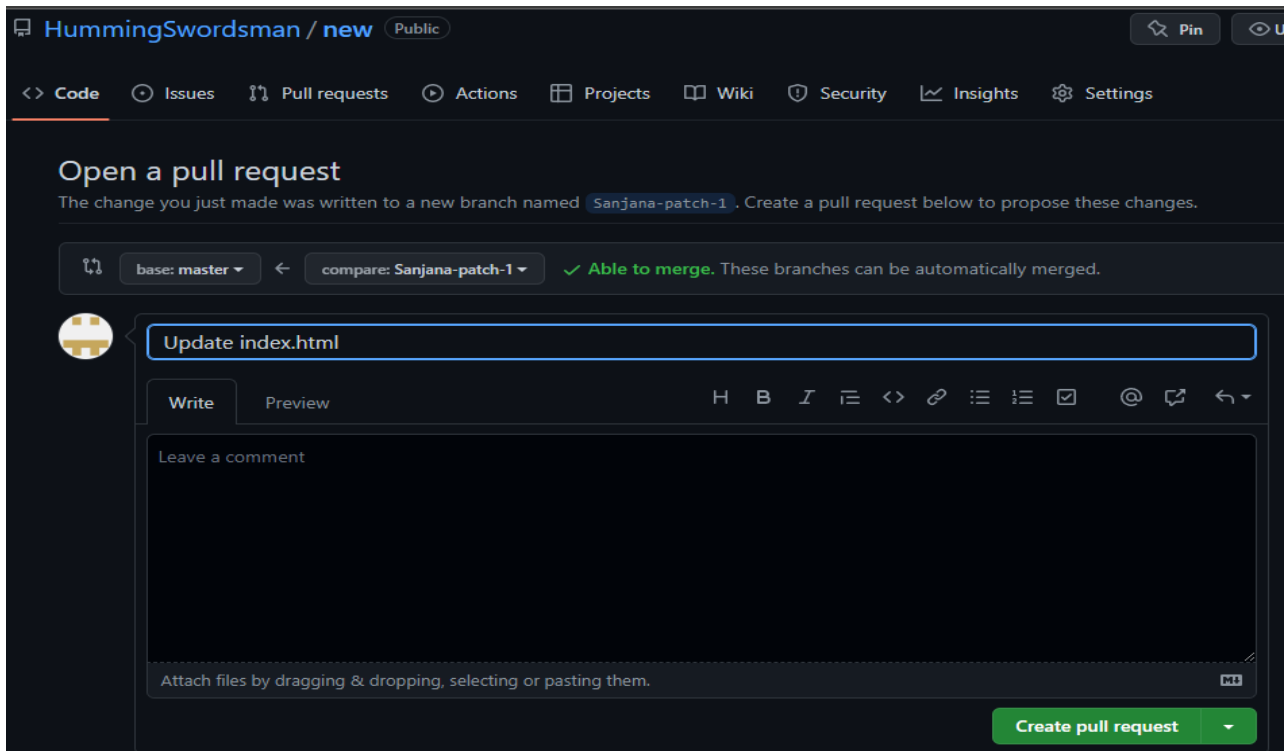
**Aim:** Open and close a pull request

### Open a Pull Request:

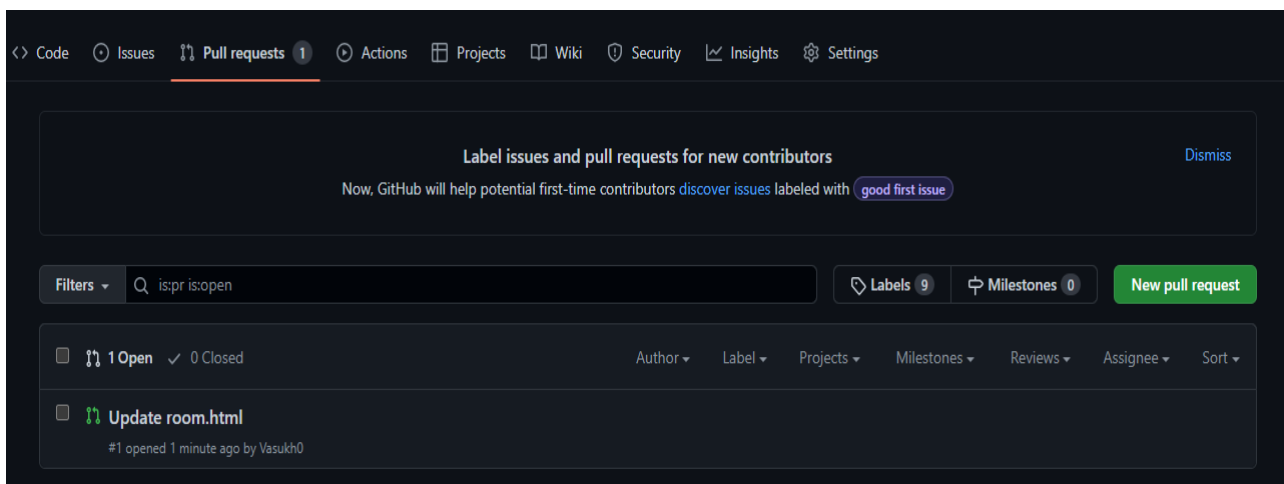
1. click on the **Fork** button in the top-right corner. This creates a new copy of my demo repo under your GitHub user account with a URL like: <https://github.com/<YourUserName>/demo>  
The copy includes all the code, branches, and commits from the original repo. Next,
2. clone the repository and create a new branch.
3. Now you can make changes to the code. The following code creates a new branch, makes an arbitrary change, and pushes it to **new\_branch**:
4. Once you push the changes to your repo, the **Compare & pull request** button will appear in GitHub.



5. Click it and you'll be taken to this screen:



3. Open a pull request by clicking the **Create pull request** button.

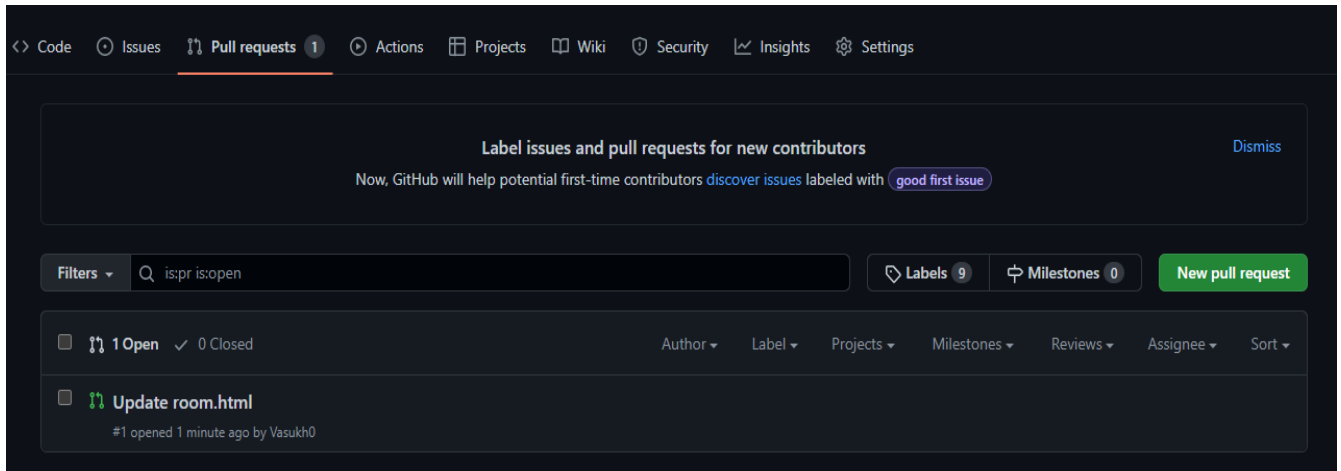


This allows the repo's maintainers to review your contribution. From here, they can merge it if it is good, or they may ask you to make some changes.

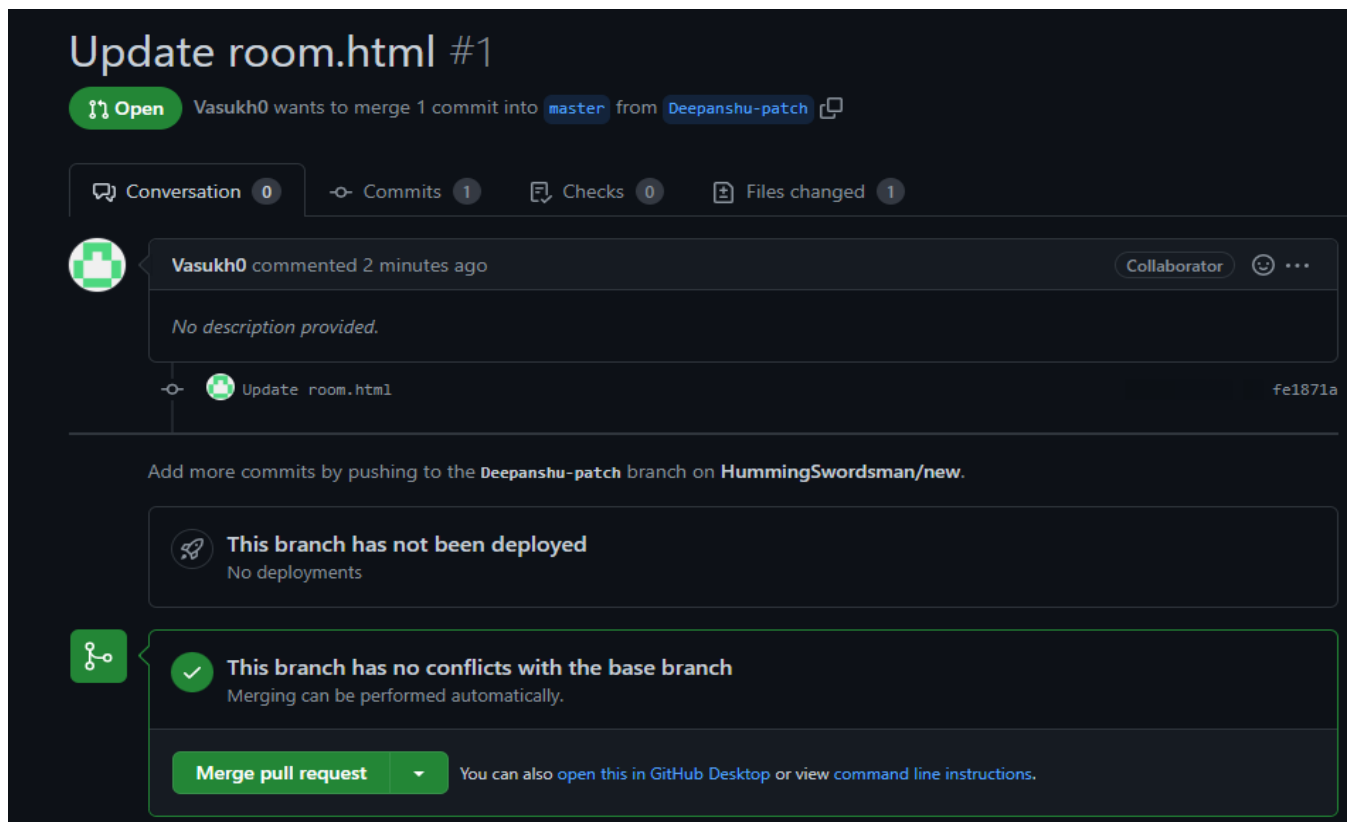
## Closing a pull request

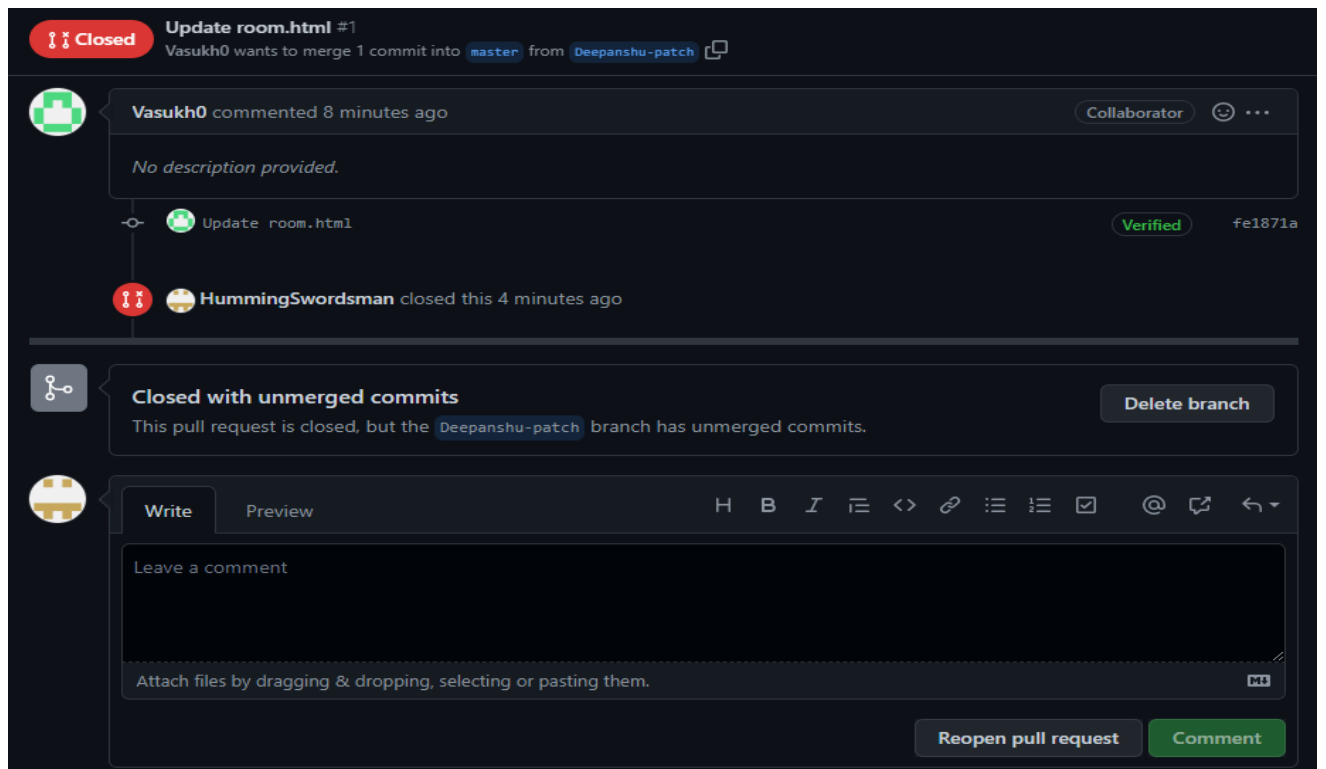
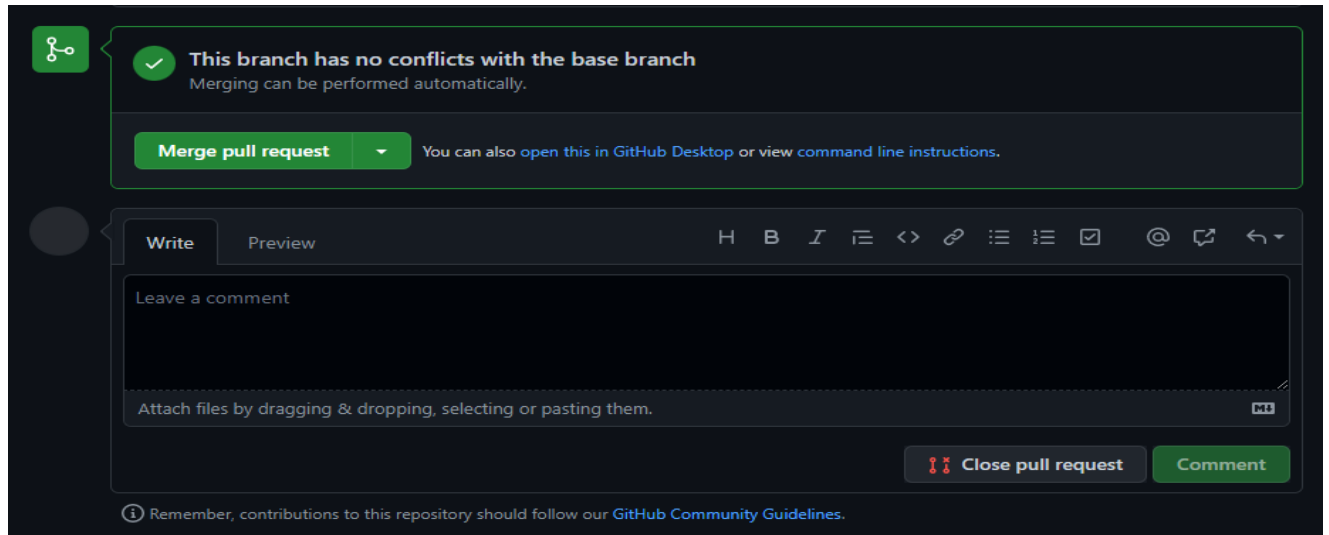
You may choose to *close* a pull request without merging it into the upstream branch. This can be handy if the changes proposed in the branch are no longer needed, or if another solution has been proposed in another branch.

1. Under your repository name, click **Pull requests**



2. At the bottom of the pull request, below the comment box, click **Close/Merge pull request**.





3. Optionally, [delete the branch](#). This keeps the list of branches in your repository tidy.

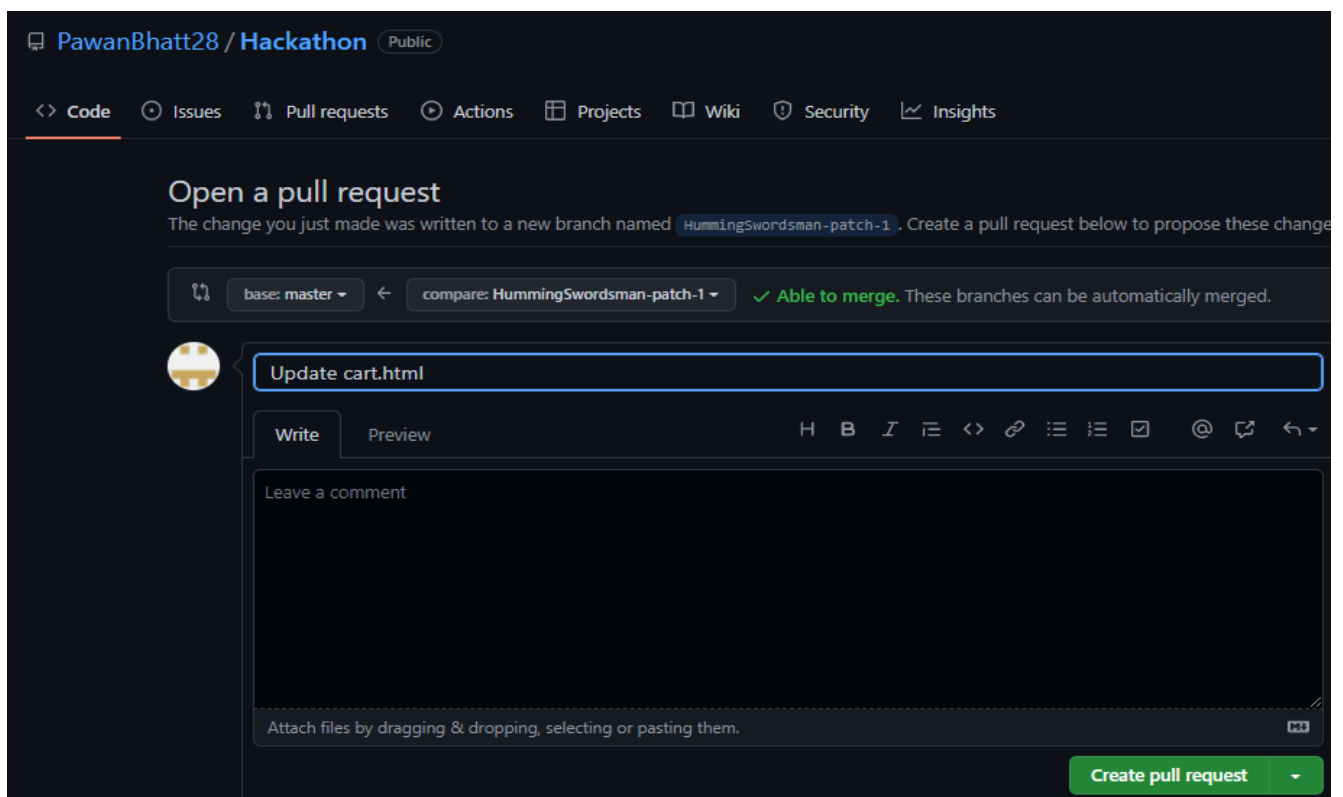


**Task 3** - Create a pull request on a team member's repo and Close pull requests generated by team members on your Repo as a maintainer.

**Aim:** Each project member shall create a pull request on a team members repo and close pull requests generated by team members on own Repo as a maintainer.

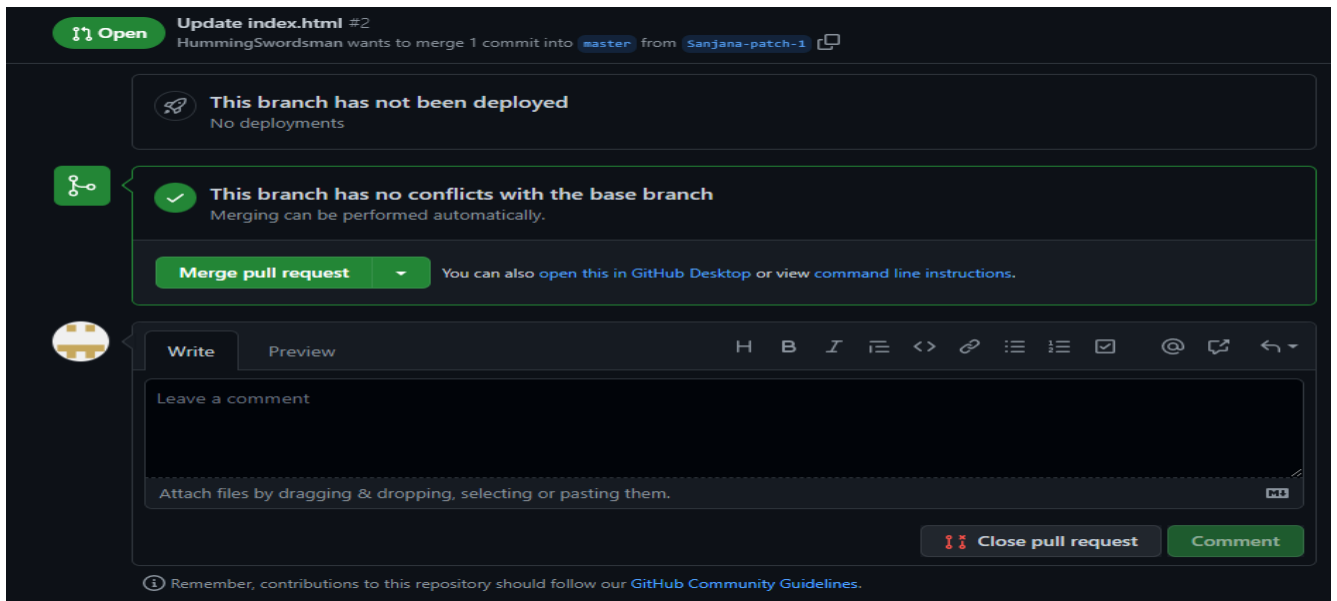
## Pull Request Generated by team members:

### 1. By Deepanshu

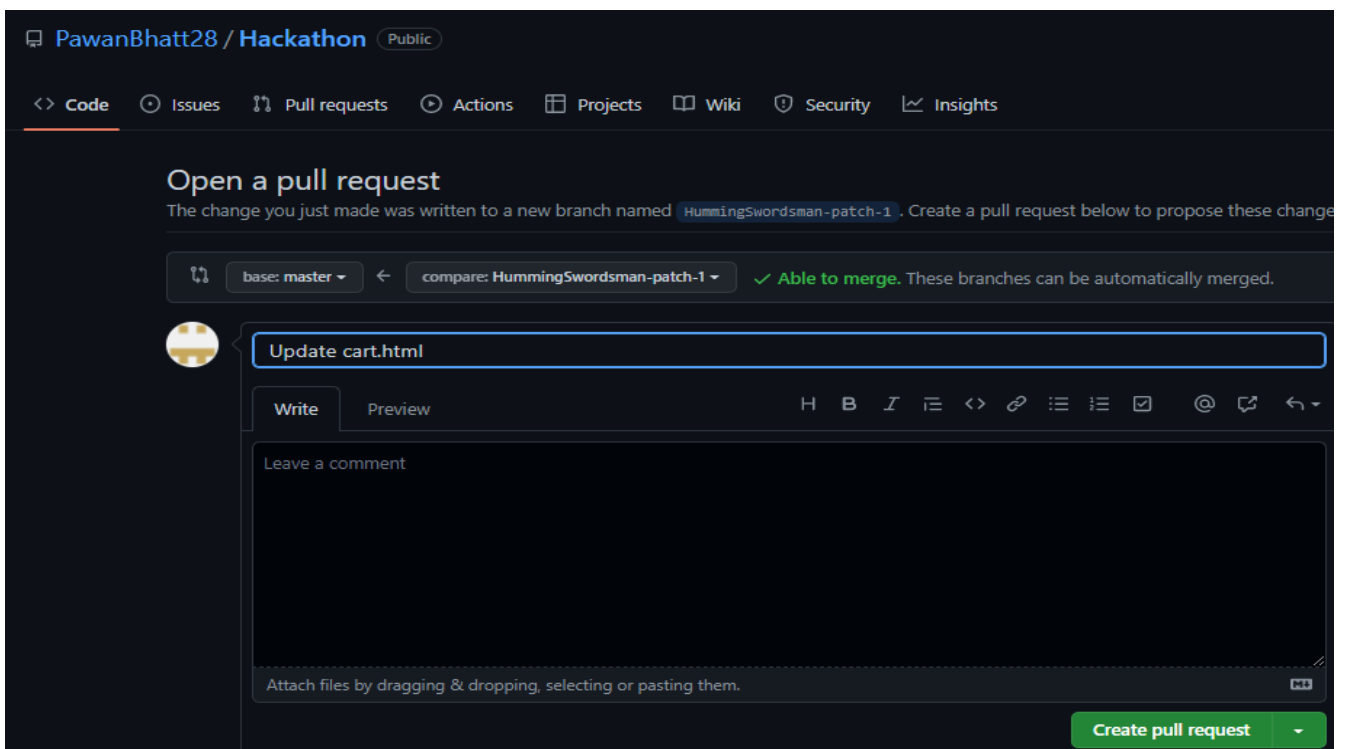


Accepted by Maintainer








## 2. By Sanjana




Accepted By Maintainer.

 **Update index.html #2**  
HummingSwordsman wants to merge 1 commit into `master` from `Sanjana-patch-1`

 **This branch has not been deployed**  
No deployments


 **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](#).


 **HummingSwordsman / new** Public Pin 👁

[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)



## Update index.html #2



 **Merged** HummingSwords... merged 1 commit into `master` from `Sanjana-patch-1` now


 Conversation **0**  Commits **1**  Checks **0**  Files changed **1**

 **HummingSwords...** commented 1 minute ago Owner 😊 ...

*No description provided.*

  Update index.html Verified `5a6ab4e`

  **HummingSwordsman** merged commit `62c26aa` into `master` now Revert

 **Pull request successfully merged and closed** Delete branch

You're all set—the `Sanjana-patch-1` branch can be safely deleted.

## Task 4 - Publish and print network graphs.

### Aim: Publish and print network graphs

#### Network Graph:

The network graph 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.

HummingSwordsman / new Public

Code Issues Pull requests 2 Actions Projects Wiki Security Insights Settings

master 7 branches 0 tags

This branch is 6 commits ahead, 3 commits behind main.

HummingSwordsman Merge pull request #2 from HummingSwordsman/Sanjana-patch-1 62c26aa yesterday 6 commits

File	Commit	Time
about.html	first commit	3 days ago
book2.html	first commit	3 days ago
contact0.html	first commit	3 days ago
foodnew.html	first commit	3 days ago
index.html	Update index.html	yesterday
main.css	first commit	3 days ago
main2.css	first commit	3 days ago

About: hotel project (scm), 0 stars, 1 watching, 2 forks.

Releases: No releases published. Create a new release.

Packages: No packages published. Publish your first package.

HummingSwordsman / new Public

Code Issues Pull requests 2 Actions Projects Wiki Security Insights Settings

Network graph

Timeline of the most recent commits to this repository and its network ordered by most recently pushed to.

Owners	Apr May
HummingSwordsman	13 18 20 21

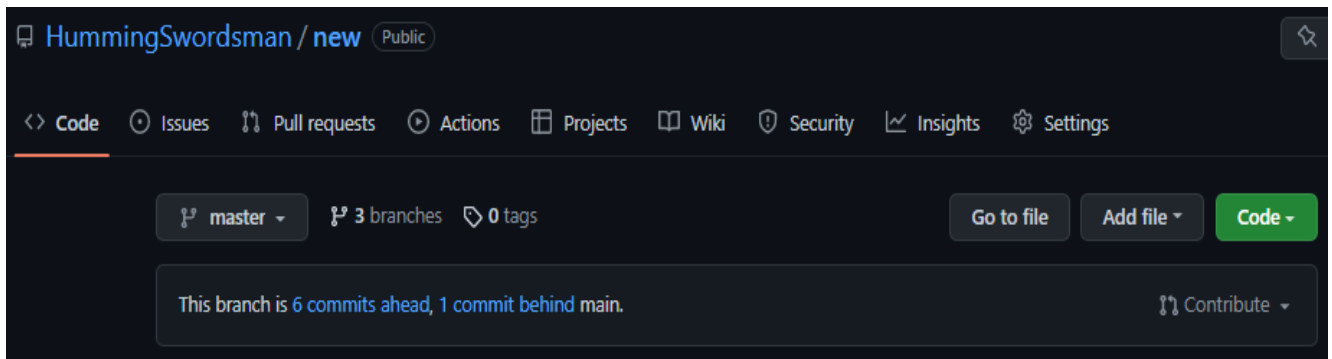
The graph shows a main branch with a commit on April 13, followed by a forked branch 'Sanjana-patch-1' with a commit on April 20, and another forked branch 'HummingSwordsman-patch-1' with a commit on April 21.

Left sidebar: Pulse, Contributors, Community, Community Standards, Traffic, Commits, Code frequency, Dependency graph, Network (selected), Forks.

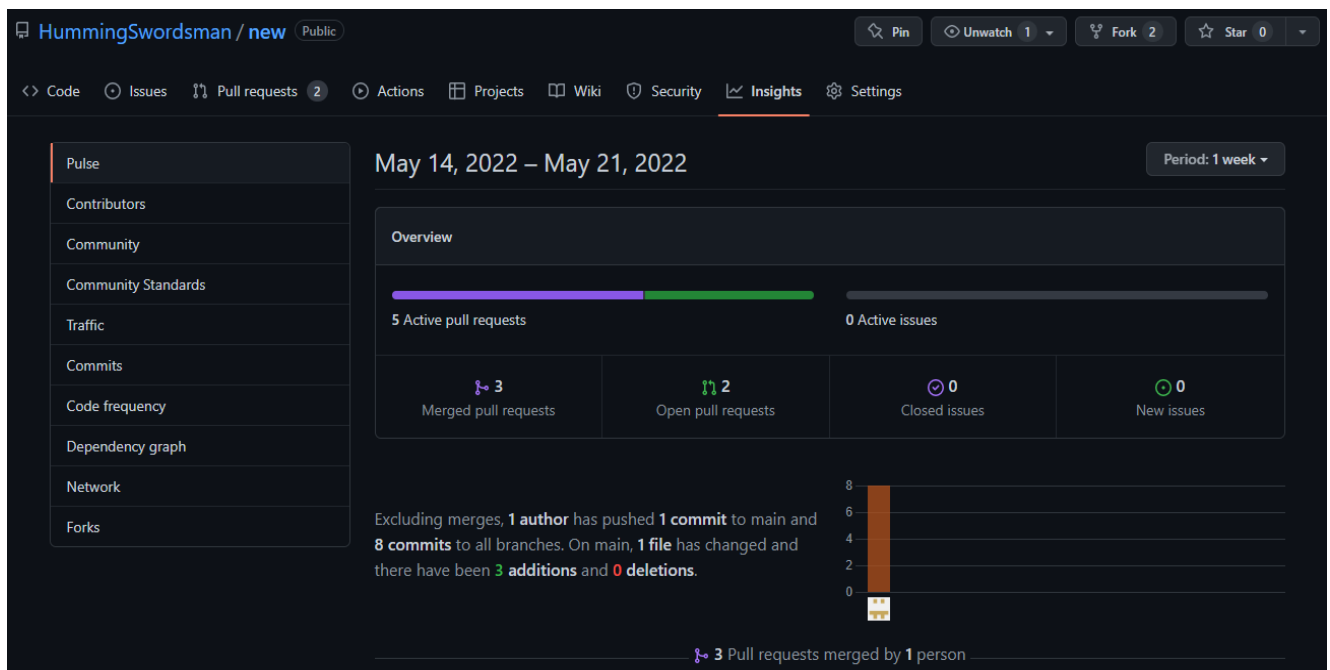
**Tip:** To see older branches, click and drag within the graph.

## Accessing the network graph

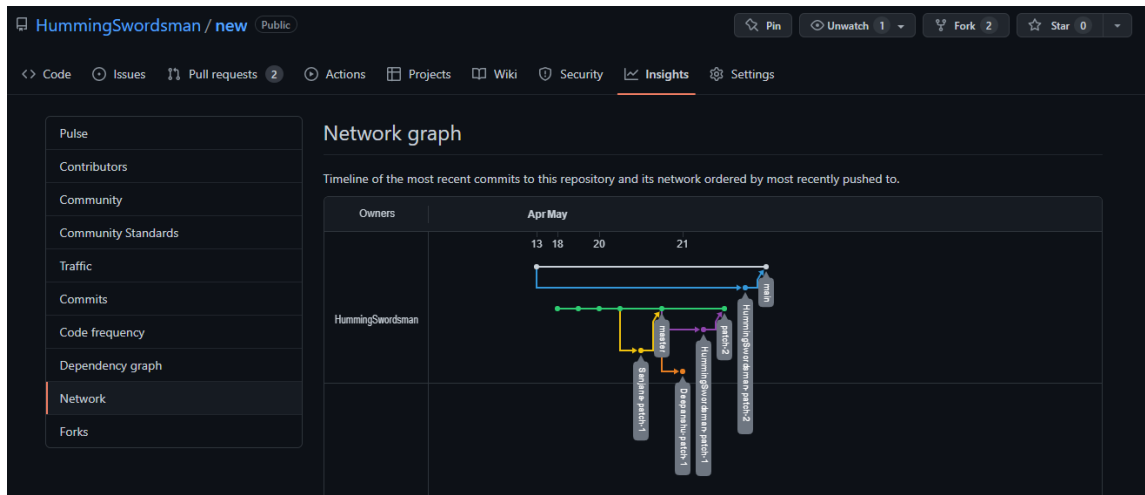
1. On GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click **Insights**.



Now we see,



3. In the left sidebar, click **Network**.



# 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/>