

**Subject Name: Source Code Management**

**Subject Code: CS181**

**Cluster: Beta**

**Department: DCSE**



**Submitted By:**

Alisha Gupta

2110990142

G2

**Submitted To:**

Dr. Deepak Thakur

S.No	Program Title	Page no.
1.	Setting up of GIT Client	3-9
2.	Setting up Github Account	10-15
3.	Program to Generate Logs	16
4.	Create and visualize branches	17-19
5.	Git lifecycle Description	20-23
6.	Add collaborators on Github Repo	27-29
7.	Fork and Commit	30-34
8.	Merge and resolve conflicts created due to own activity and collaborators activity	35-48
9.	Reset and Revert	49-56

**Aim:** Setting up of GIT client

### Description:

- GIT is basically a distributed version control system for tracking changes in source code during software development.
- It is designed for coordinating work among programmers, but it can be used to track changes in any set of files.
- Its goal is to increase efficiency, speed and easily manage large projects through version controlling.

### Objective: To Install GIT Bash

Follow the steps given below to install GIT Bash on Windows:

- ❖ Browse to the official GIT website: <https://git-scm.com/downloads>.
- ❖ Click the download link for Windows and allow the download to complete.



The image shows a screenshot of the official Git website's 'Downloads' section. On the left, there are three options: 'Mac OS X', 'Windows' (which is highlighted with a red box and a red arrow pointing to it), and 'Linux/Unix'. Below these options, a note states: 'Older releases are available and the [Git source repository](#) is on GitHub.' On the right, there is a monitor icon displaying the 'Latest source Release' information for version 2.24.0, including the release date (2019-11-04) and a 'Download 2.24.0 for Windows' button. At the bottom of the page, there are two sections: 'GUI Clients' (describing built-in tools like git-gui and gitk, and mentioning third-party clients) and 'Logos' (describing various Git logos available in PNG and EPS formats).

**Downloads**

Mac OS X    Windows    Linux/Unix

Older releases are available and the [Git source repository](#) is on GitHub.

**GUI Clients**

Git comes with built-in GUI tools (`git-gui`, `gitk`), but there are several third-party tools for users looking for a platform-specific experience.

[View GUI Clients →](#)

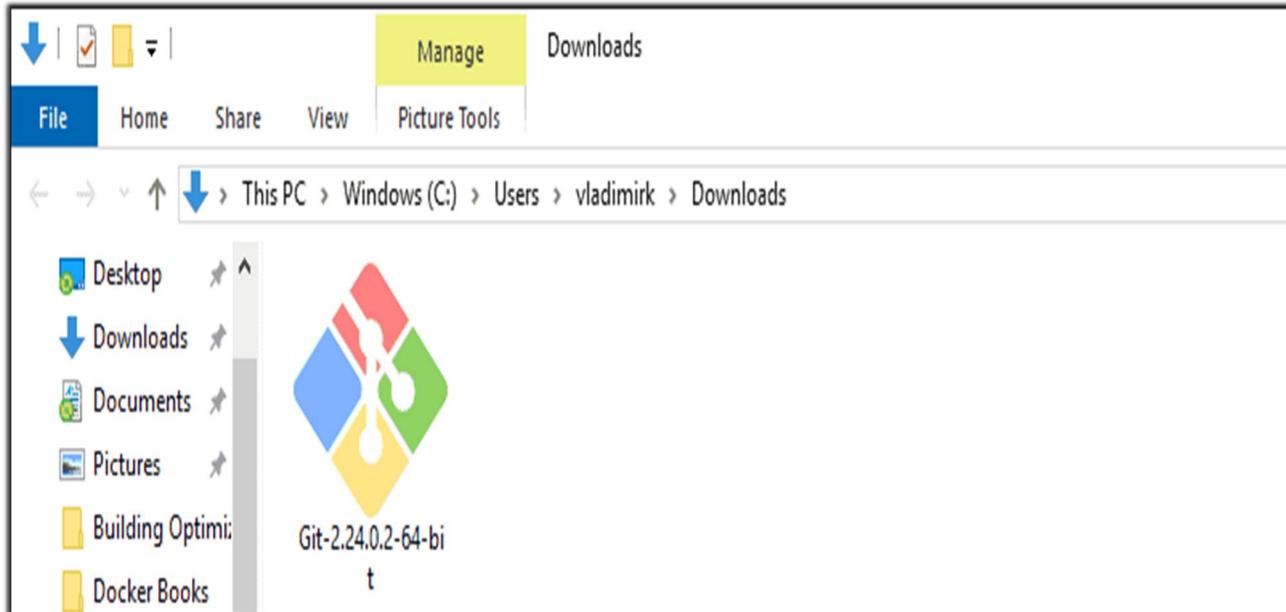
**Logos**

Various Git logos in PNG (bitmap) and EPS (vector) formats are available for use in online and print projects.

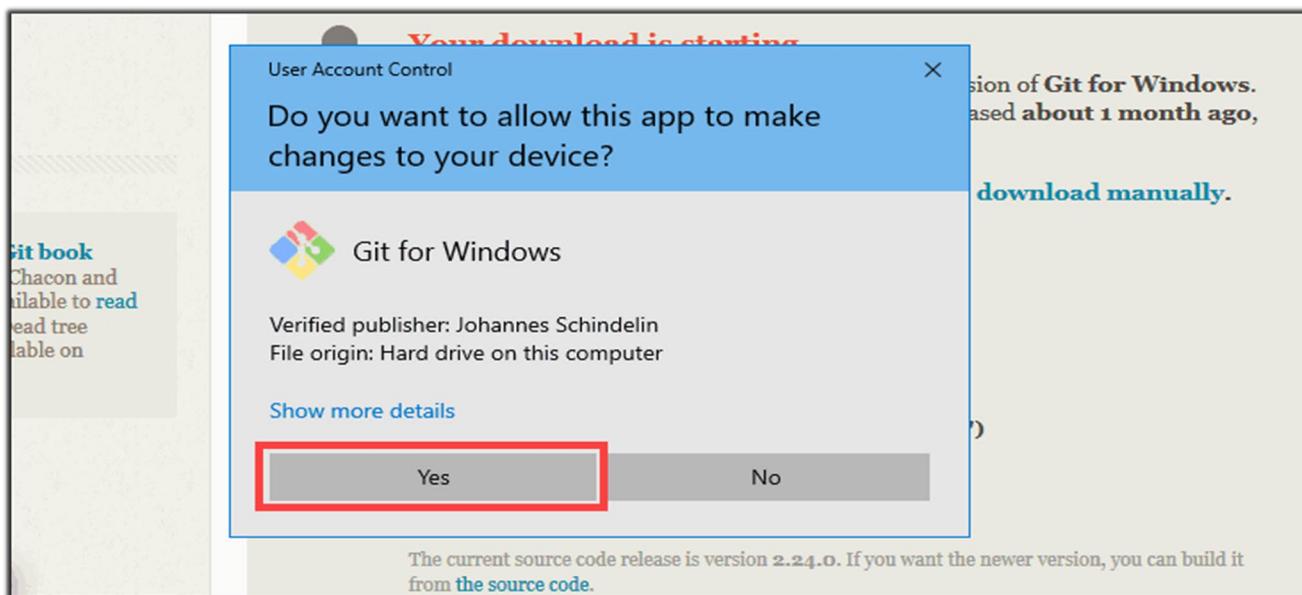
[View Logos →](#)

## Extract and Launch GIT Installer

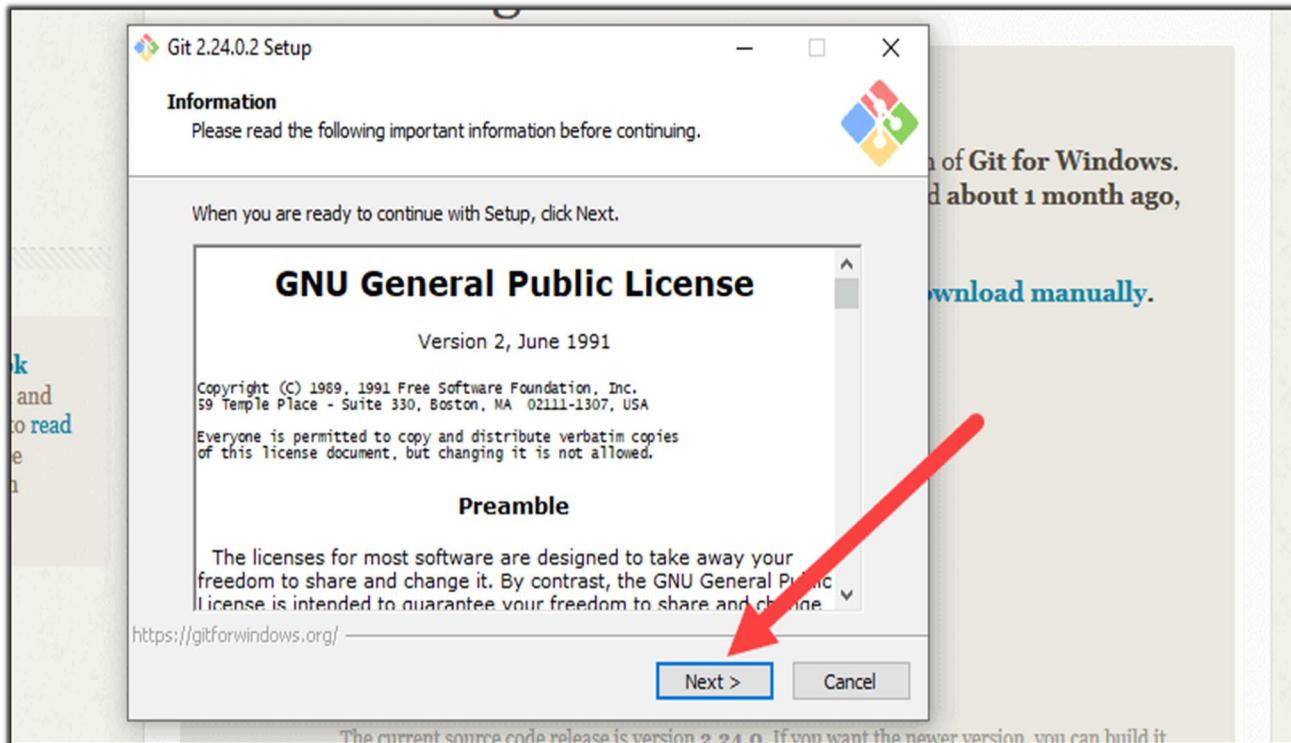
- ❖ Browse to the download location (or use the download shortcut in your browser). Double-click the file to extract and launch the installer.



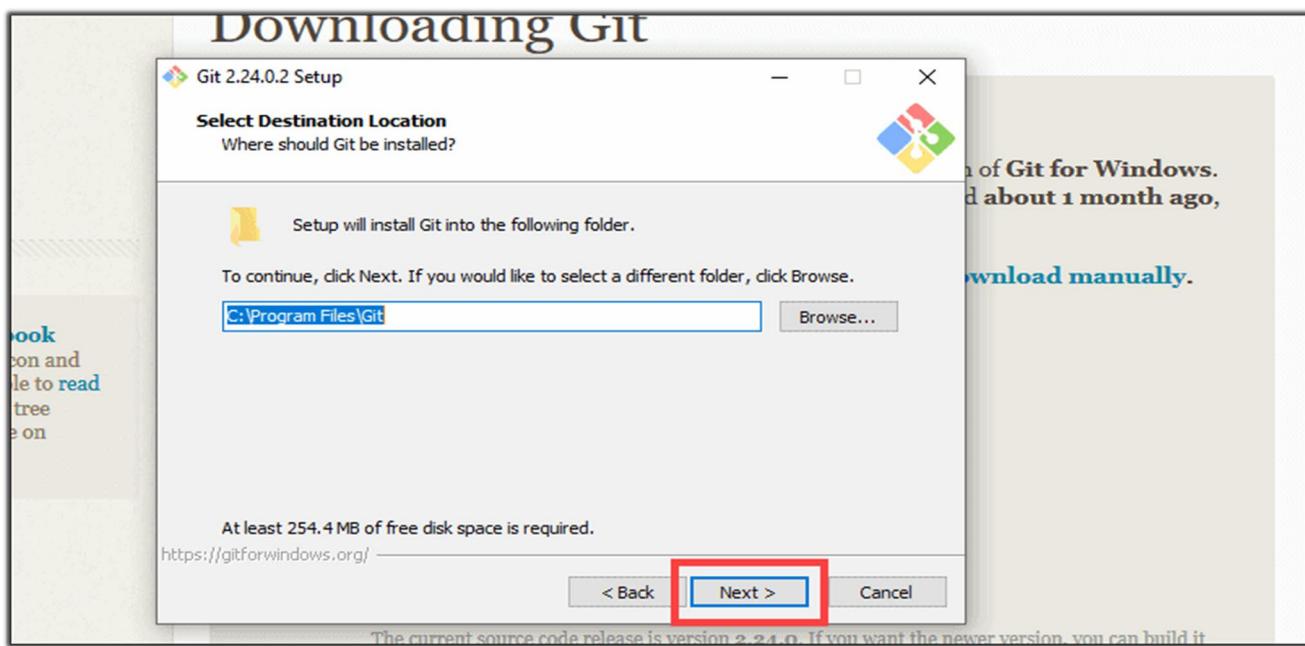
- ❖ Allow the app to make changes to your device by clicking **Yes** on the User Account Control dialog that opens.



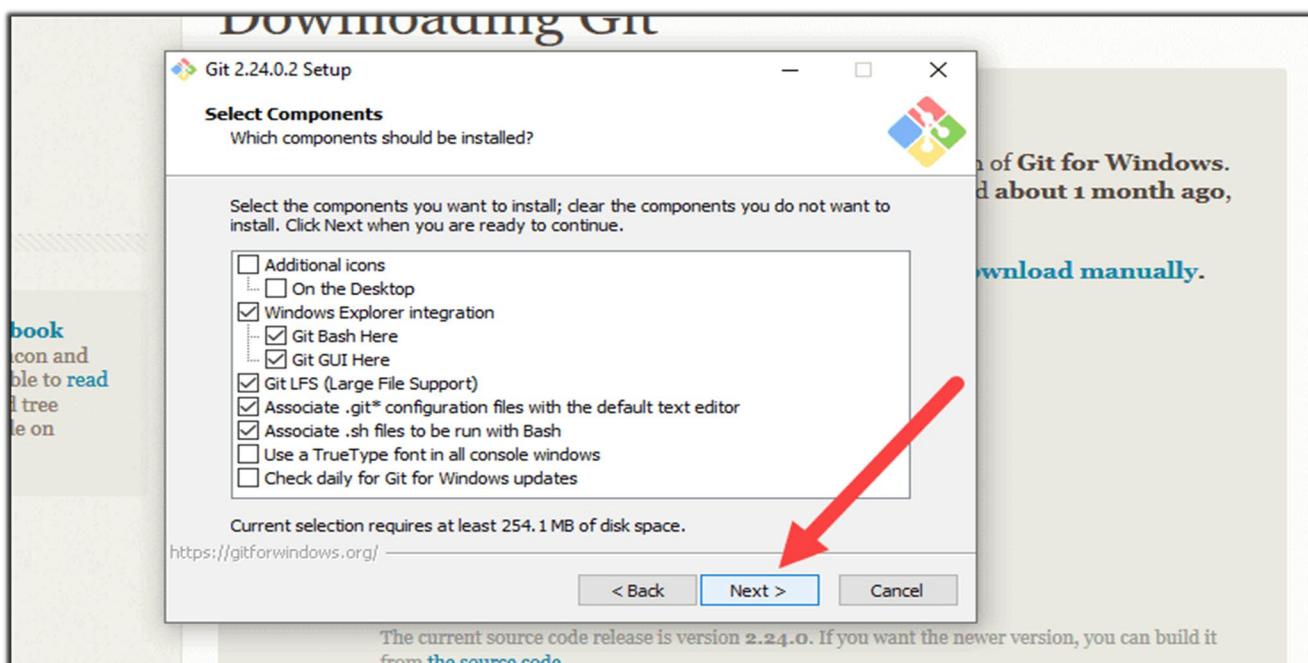
- ❖ Review the GNU General Public License, and when you're ready to install, click **Next**.



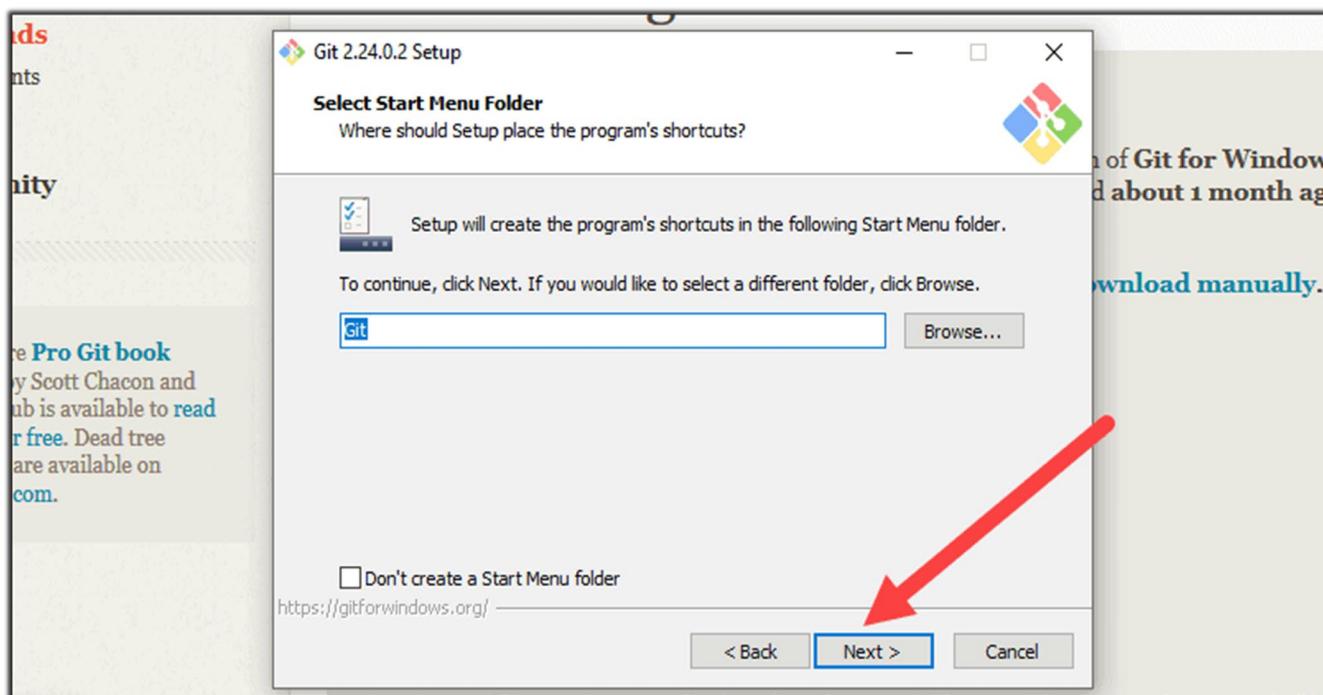
- ❖ The installer will ask you for an installation location. Leave the default, unless you have reason to change it, and click **Next**.



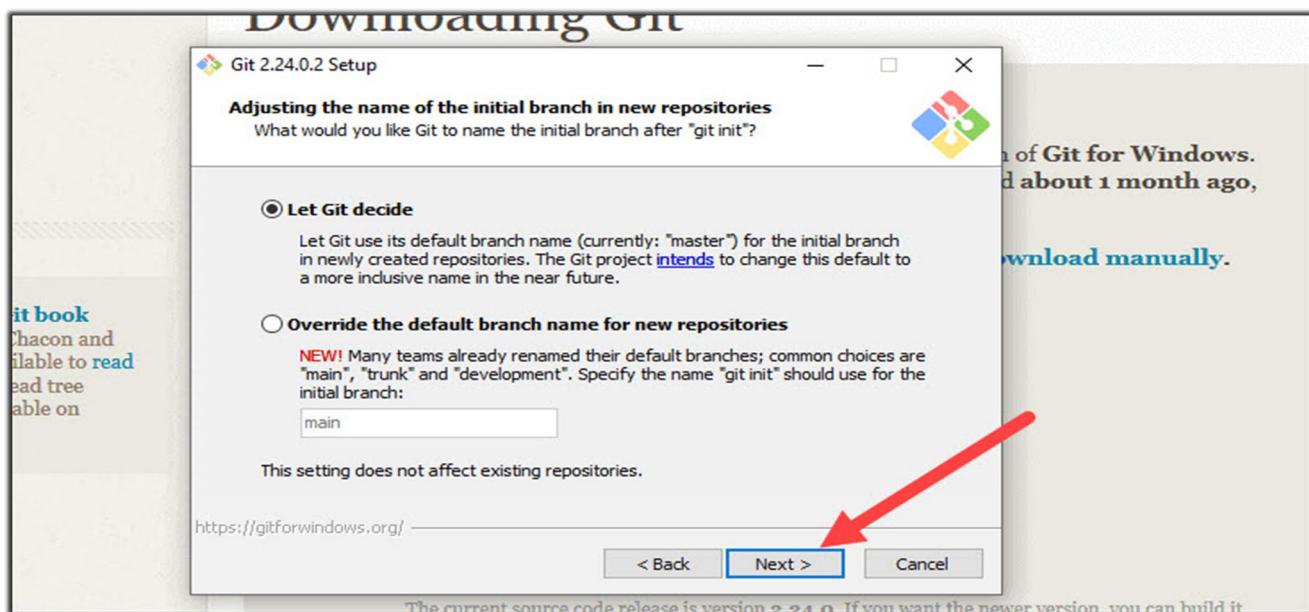
- ❖ A component selection screen will appear. Leave the defaults unless you have a specific need to change them and click **Next** .



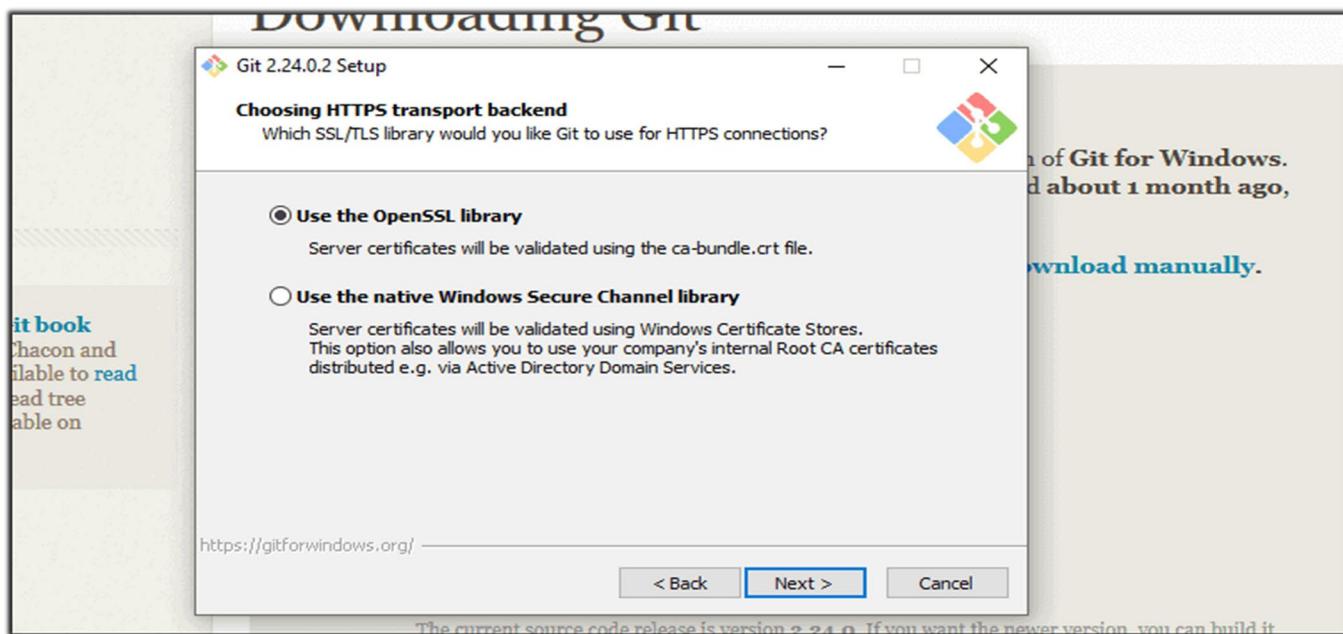
- ❖ The installer will offer to create a start menu folder. Simply click **Next**.



- ❖ The next step allows you to choose a different name for your initial branch. The default is 'master.' Unless you're working in a team that requires a different name, leave the default option and click **Next**.



- ❖ The next option relates to server certificates. Most users should use the default. Click **Next**.

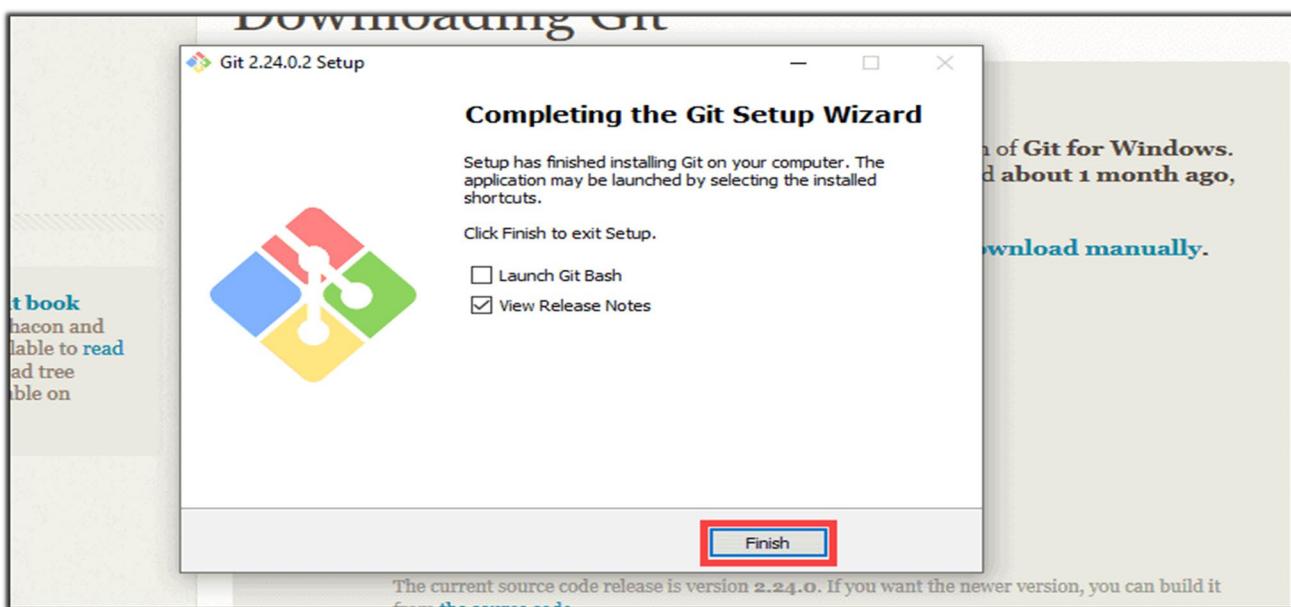


- ❖ Choose the terminal emulator you want to use. The default MinTTY is recommended, for its features. Click **Next**.



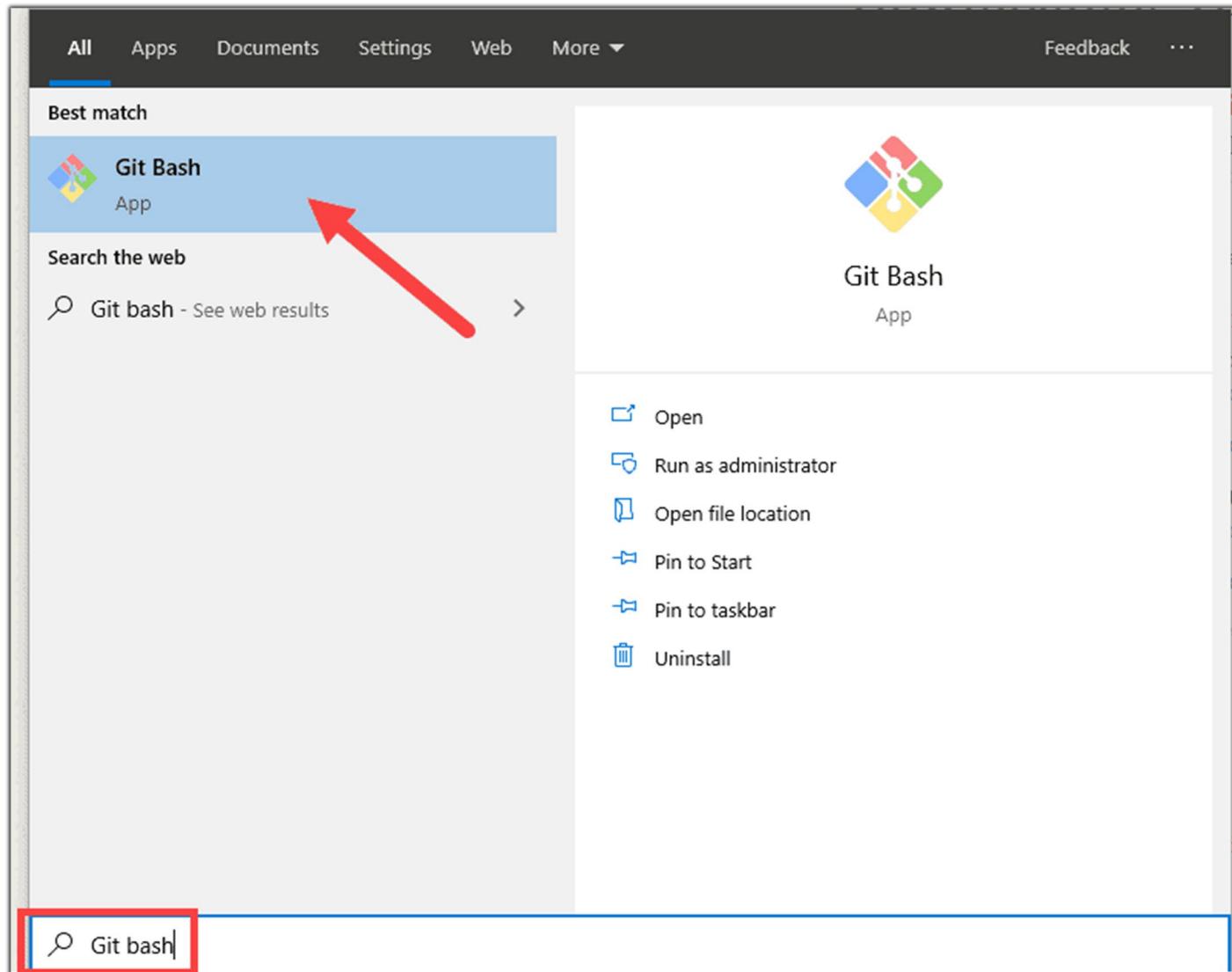
## Complete GIT Installation Process

- ❖ Once the installation is complete, tick the boxes to view the Release Notes or Launch GIT Bash, then click **Finish**.



## Launch GIT Bash Shell

- ❖ To launch **GIT Bash** open the **Windows Start** menu, type **git bash** and press **Enter** (or click the application icon).



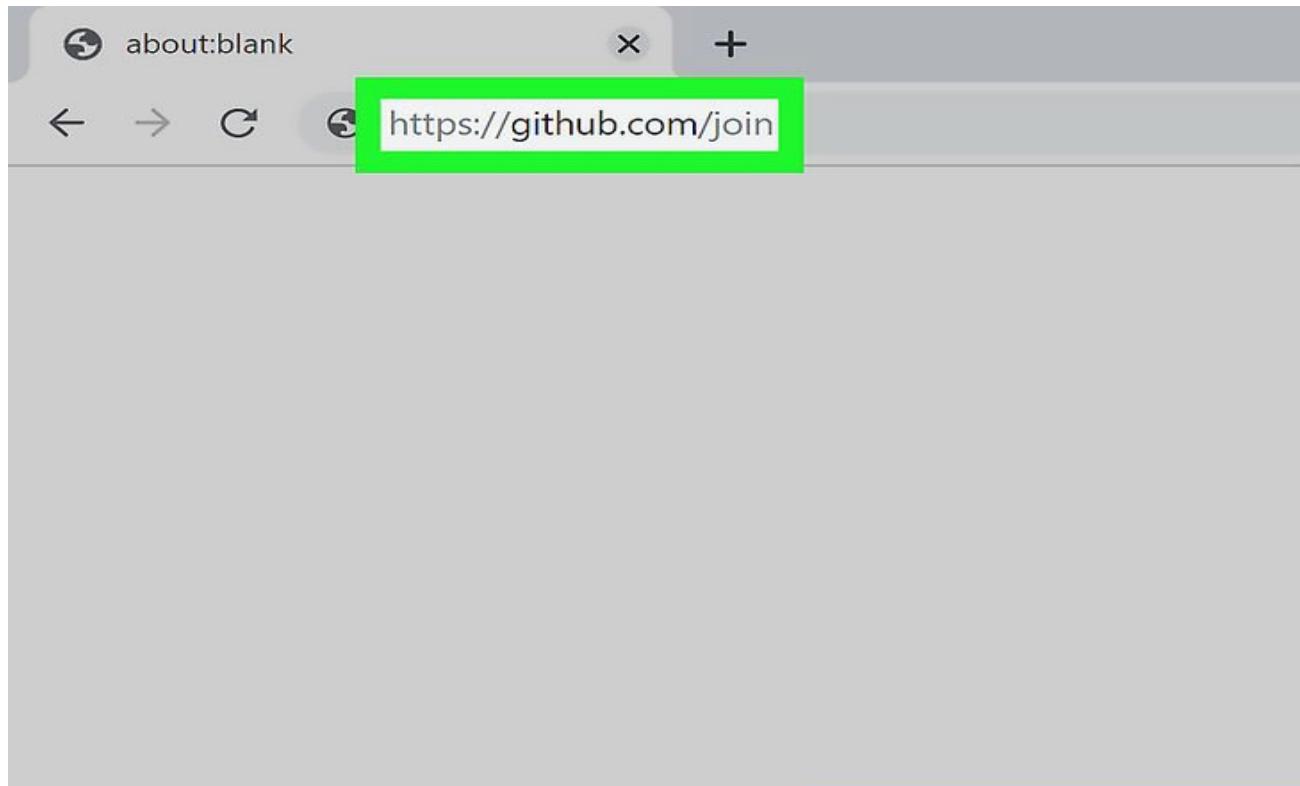
**Aim:** Setting up of Github Account**Description:**

- GitHub is an open-source repository hosting service, sort of like a cloud for code
- GitHub is a hosting site where developers and programmers can upload the code they create and work collaboratively to improve it.
- With GitHub, developers can build code, track changes, and innovate solutions to problems

**Objectives:**

To *make your account on github follow the following steps:*

- ❖ Go to <https://github.com/join> in a web browser. You can use any web browser on your computer, phone, or tablet to join.



## ❖ Enter your personal details.

In addition to creating a username and entering an email address, you'll also have to create a password.

*The login form will appear on the same page. Fill out the form with your details to create an account on GitHub.*

Create your personal account

**Username \***

2Alisha 

This will be your username. You can add the name of your organization later.

**Email address \***

Alishagupta314@gmail.com 

We'll occasionally send updates about your account to this inbox. We'll never share your email address with anyone.

**Password \***

\*\*\*\*\*

Make sure it's at least 15 characters OR [at least 8 characters including a number and a lowercase letter](#). [Learn more](#).

Verify account 

- ❖ Click the **Create an account** button. It's below the form.

We'll occasionally send updates about your account to this inbox. We'll never share your email address with anyone.

**Password \***

.....

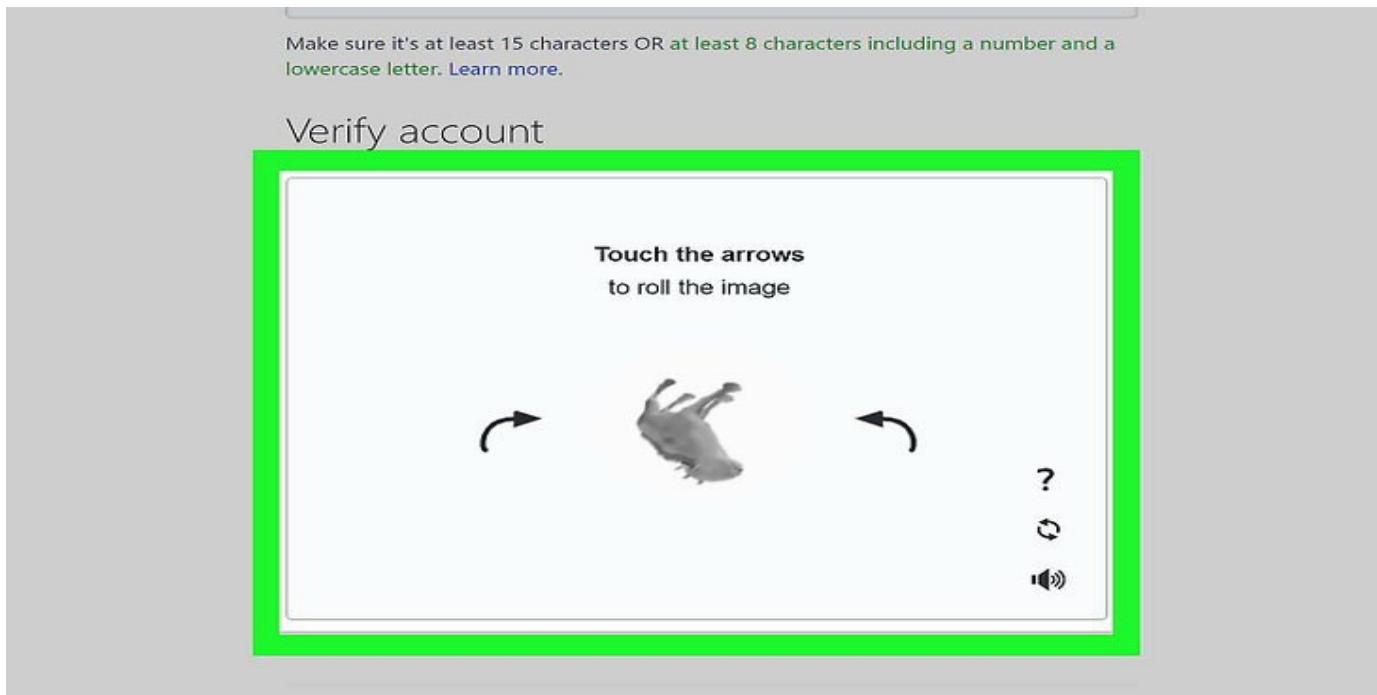
Make sure it's at least 15 characters OR [at least 8 characters including a number and a lowercase letter](#). [Learn more](#).

Verify account

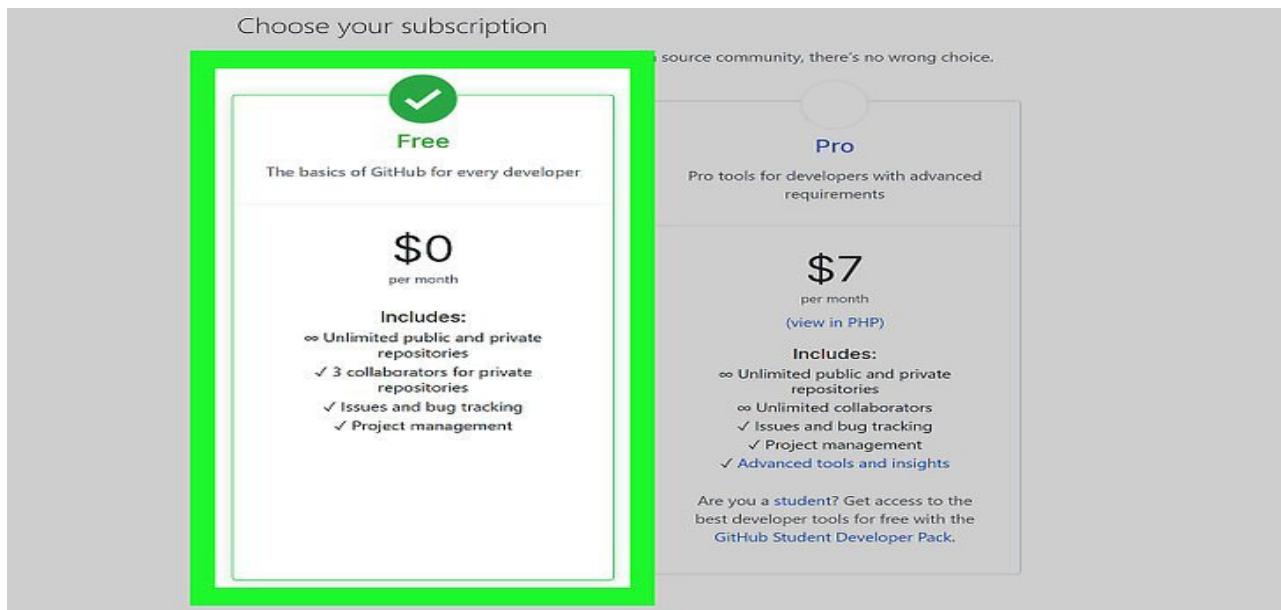
By clicking "Create an account" below, you agree to our [Terms of Service](#) and [Privacy Statement](#). We'll occasionally send you account-related emails.

**Create an account**

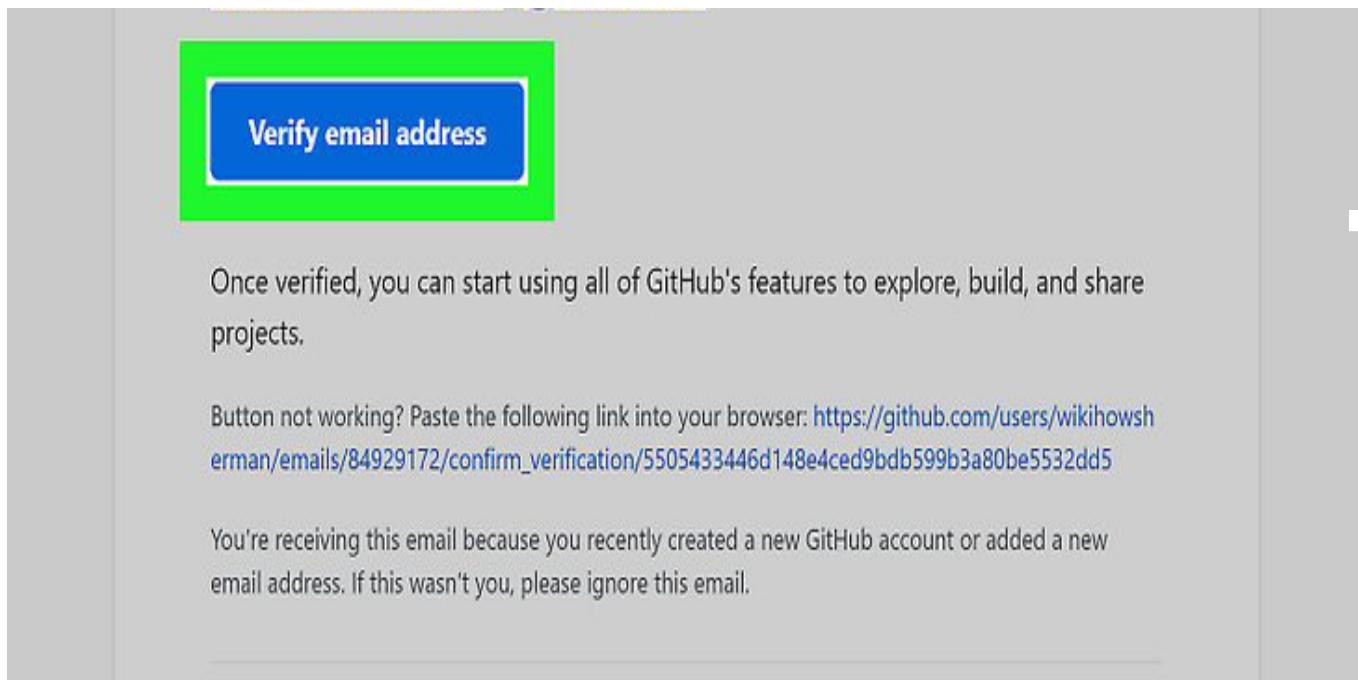
- ❖ Complete the CAPTCHA puzzle. The instructions vary by puzzle, so just follow the on-screen instructions to confirm that you are a human.



- ❖ Click the **Choose** button for your desired plan. Once you select a plan, GitHub will send an email confirmation message to the address you entered.

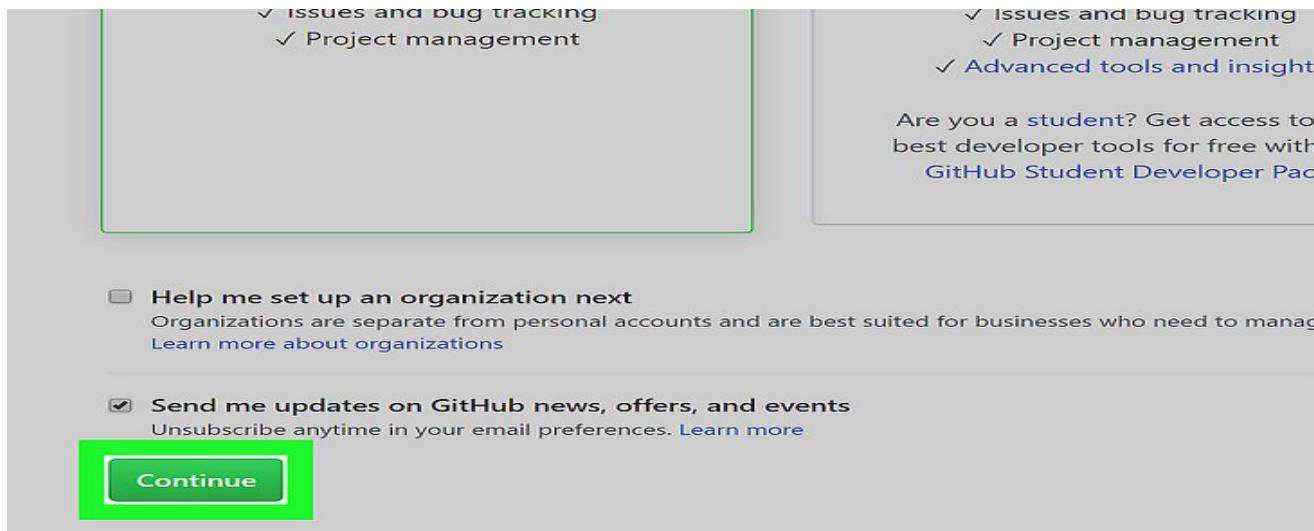


- ❖ Click the **Verify email address** button in the message from GitHub. This confirms your email address and returns you to the sign-up process.



❖ Review your plan selection and click **Continue**. You can also choose whether you want to receive updates from GitHub via email by checking or unchecking the "Send me updates" box.

➤ If you chose a paid plan, you'll have to enter your payment information.



The screenshot shows two plan options:

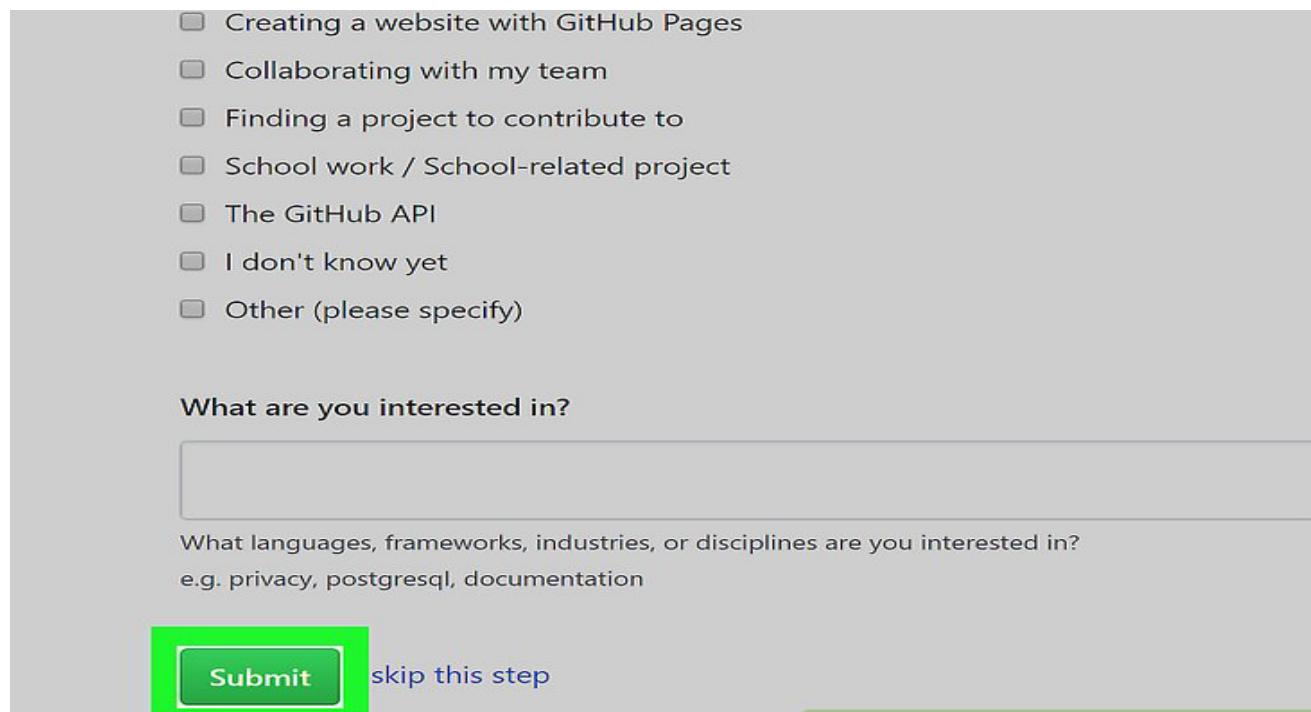
- Free plan:** Includes Issues and bug tracking, Project management.
- Paid plan:** Includes Issues and bug tracking, Project management, and Advanced tools and insight. It also features a promotional message for GitHub Student Developer Pack.

Below the plans, there are two checkboxes:

- Help me set up an organization next (unchecked)
- Send me updates on GitHub news, offers, and events (checked)

A green button labeled "Continue" is visible at the bottom left.

❖ Select your preferences and click **Submit**



On this page, users can select their interests:

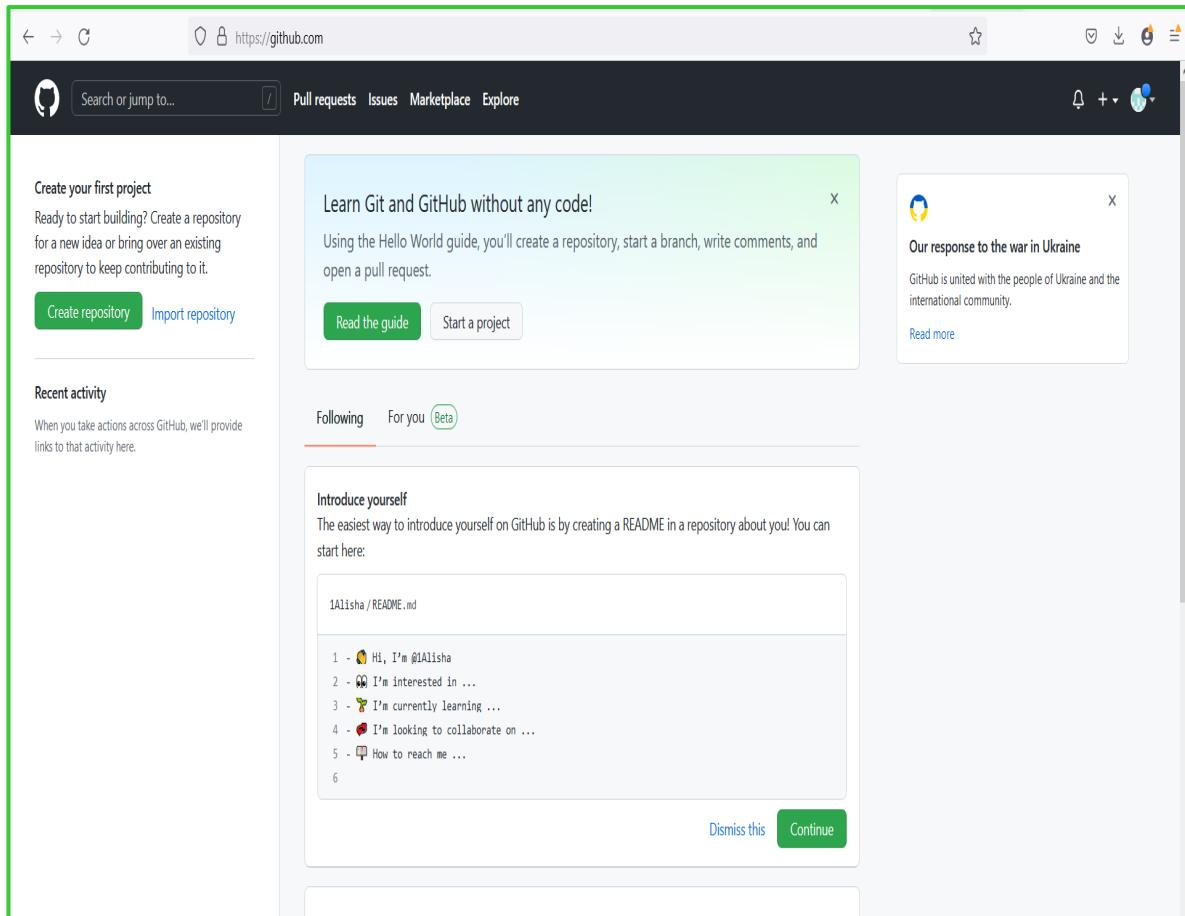
- Creating a website with GitHub Pages
- Collaborating with my team
- Finding a project to contribute to
- School work / School-related project
- The GitHub API
- I don't know yet
- Other (please specify)

Below the interests, there's a section for "What are you interested in?" with a text input field. A note says: "What languages, frameworks, industries, or disciplines are you interested in? e.g. privacy, postgresql, documentation".

At the bottom, there are two buttons: a green "Submit" button and a blue "skip this step" link.

## ❖ GitHub Account Dashboard

- Now that the GitHub account is all set up, you can log in through your credentials on the ***GitHub's website***.



**Aim:** Program to generate logs**Description:**

- Git log is a utility tool to review and read a history of everything that happens to a repository. Multiple options can be used with a git log to make history more specific.
- The basic git log command will display the most recent commits and the status of the head. It will use as:

```
$ git log  
< >
```

- The above command will display the last commits. Consider the below output:

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git log
commit Od3835a76b82a4dc7ca97bcfbeb4e39b26a680 (HEAD -> master)
Author: ImDwivedil <himanshudubey481@gmail.com>
Date:   Fri Nov 8 15:49:51 2019 +0530

    newfile2 Re-added

commit 56afce0ea387ab840819686ec9682bb07d72add6 (tag: -d, tag: --delete, tag: --
d, tag: projectv1.1, origin/master, testing)
Author: ImDwivedil <himanshudubey481@gmail.com>
Date:   Wed Oct 9 12:27:43 2019 +0530

    Added an empty newfile2

commit Od5191fe05e4377abef613d2758ee0dbab7e8d95
Author: ImDwivedil <himanshudubey481@gmail.com>
Date:   Sun Oct 6 17:37:09 2019 +0530

    added a new image to project

commit 828b9628a873091ee26ba53c0fcfc0f2a943c544 (tag: olderversion)
Author: ImDwivedil <s2317024+ImDwivedil@users.noreply.github.com>
Date:   Thu Oct 3 11:17:25 2019 +0530

    Update design2.css

commit 0a1a475d0b15ecec744567c910eb0d8731ae1af3 (test)
Author: ImDwivedil <s2317024+ImDwivedil@users.noreply.github.com>
Date:   Tue Oct 1 12:30:40 2019 +0530

    CSS file

    See the proposed CSS file.

commit f1ddc7c9e765bd688e2c5503b2c88cb1dc835891
Author: ImDwivedil <himanshudubey481@gmail.com>
Date:   Sat Sep 28 12:31:30 2019 +0530
```

The above command is listing all the recent commits. Each commit contains some unique sha-id, which is generated by the SHA algorithm. It also includes the date, time, author, and some additional details.

## git init

Initializes a new git repository. If we want control to place a project under revision control, this is the first command we need to learn.

```
MINGW64/c/Users/shubh/Desktop/f1
```

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (master)
$ git init
Initialized empty Git repository in C:/Users/shubh/Desktop/f1/.git/
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (master)
$ |
```

## git config

A convenient way to set configuration options for git installation.

### Set your global username/email configuration

Open Git Bash and begin creating a username and email for working on Git Bash.

#### Set your username:

```
git config --global user.name "FIRST_NAME LAST_NAME"
```

#### Set your email address:

```
git config --global user.email "MY_NAME@example.com"
```

```
MINGW64/c/Users/shubh/Desktop/f1
```

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (master)
$ git init
Initialized empty Git repository in C:/Users/shubh/Desktop/f1/.git/
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (master)
$ git config --global user.name "Alisha"
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (master)
$ git config --global user.email "alishagupta314@gmail.com"
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (master)
$ git config user.name
Alisha
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (master)
$ git config user.email
alishagupta314@gmail.com
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (master)
$ git config user.name
Alisha
$ |
```

## git status

Displays the state of the working directory and the staged snapshot .

```
MINGW64:/c/Users/shubh/Desktop/f1
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (master)
$ git init
initialized empty Git repository in C:/Users/shubh/Desktop/f1/.git/
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (master)
$ git config --global user.name "Alisha"
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (master)
$ git config --global user.email "alishagupta314@gmail.com"
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (master)
$ git config user.name
Alisha
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (master)
$ git config user.email
alishagupta314@gmail.com
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (master)
$ git status
On branch master
No commits yet
nothing to commit (create/copy files and use "git add" to track)
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (master)
$
```

## git add

Moves changes from the working directory to the staging area.

```
MINGW64:/c/Users/shubh/Desktop/f1
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (master)
$ git init
initialized empty Git repository in C:/Users/shubh/Desktop/f1/.git/
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (master)
$ git config --global user.name "Alisha"
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (master)
$ git config --global user.email "alishagupta314@gmail.com"
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (master)
$ git config user.name
Alisha
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (master)
$ git config user.email
alishagupta314@gmail.com
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (master)
$ git status
On branch master
No commits yet
nothing to commit (create/copy files and use "git add" to track)
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (master)
$ git add .
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (master)
$ git status
On branch master
No commits yet
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:  A1.txt

shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (master)
$ git commit -m"new changes are added"
[master (root-commit) 52ca8c0] new changes are added
 1 file changed, 1 insertion(+)
 create mode 100644 A1.txt
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (master)
$
```

## git log

```
MINGW64:/c/Users/shubh/Desktop/f1
```

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (master)
$ git commit -m "New changes are added"
[master (root-commit) 52ca8c0] new changes are added
 1 file changed, 1 insertion(+)
create mode 100644 A1.txt
```

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (master)
$ git log
commit 52ca8c0313df5f28581acba3591e34cb31fbe01e (HEAD -> master)
Author: IAlisha <alishagupta314@gmail.com>
Date:   Wed Mar 30 20:50:50 2022 +0530
```

new changes are added

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (master)
$ git branch
* master
```

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (master)
$ git branch "activity1"
```

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (master)
$ git checkout "activity1"
Switched to branch 'activity1'
```

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (activity1)
$ git branch
* activity1
  master
```

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (activity1)
$ git branch "activity2"
```

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (activity1)
$ git checkout
```

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (activity1)
$ git merge
fatal: No remote for the current branch.
```

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (activity1)
$ git merge "activity1"
git: 'mergeactivity1' is not a git command. See 'git --help'.
```

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (activity1)
$ git merge "activity1"
Already up to date.
```

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (activity1)
$ git log
commit 52ca8c0313df5f28581acba3591e34cb31fbe01e (HEAD -> activity1, master, activity2)
Author: IAlisha <alishagupta314@gmail.com>
Date:   Wed Mar 30 20:50:50 2022 +0530
```

new changes are added

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/f1 (activity1)
$
```

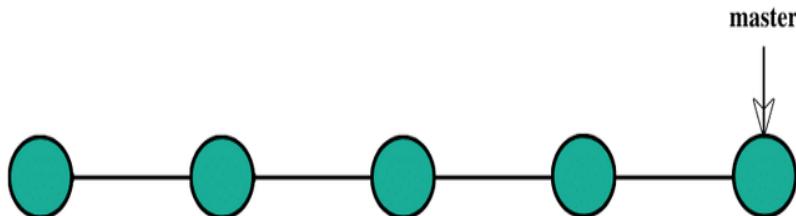
**Aim:** Create and Visualize branches**Description:**

Suppose if a team is working on a project and a branch is created for every member working on the project.

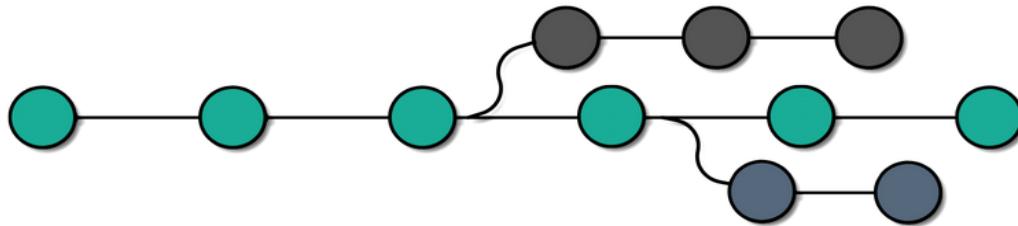
Hence every member will work on their branches hence every time the best branch is merged to the master branch of the project.

The branching makes it version controlling system and makes it very easy to maintain a project source code.

**The primary or default branch in Git is the master branch (similar to a trunk of the tree).** As soon as the repository creates, so does the main branch (*or the default branch*).



Branches give you the freedom to independently work on different modules (*not necessarily though*) and merge the modules when you finish developing them



## Syntax:

- ❖ List all of the branches in your repository

```
git branch
```

- ❖ Create a new branch

```
git branch branch_name
```

## *Navigating between Branches*

To navigate between the branches **git checkout** is used.

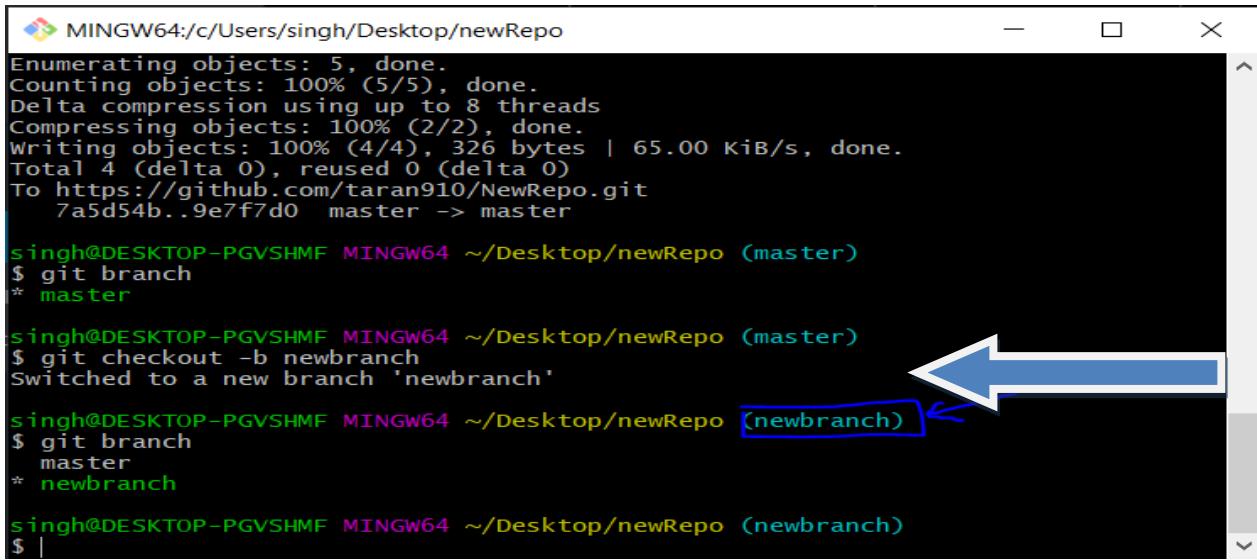
- ❖ To create a new branch and switch on it:

```
git checkout -b new_branch_name
```

- ❖ To simply switch to a branch

```
git checkout branch_name
```

After checkout to branch you can see a \* on the current branch



```
MINGW64:/c/Users/singh/Desktop/newRepo
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 326 bytes | 65.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To https://github.com/taran910/NewRepo.git
  7a5d54b..9e7f7d0  master -> master

singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (master)
$ git branch
* master

singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (master)
$ git checkout -b newbranch
Switched to a new branch 'newbranch'

singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (newbranch)
$ git branch
  master
* newbranch

singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (newbranch)
$ |
```

Now the same **commit add** and **commit actions** can be performed on this branch also.

```
singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (newbranch)
$ git add .

singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (newbranch)
$ git commit -m "branch commit"
[newbranch 3eb93e9] branch commit
 1 file changed, 1 insertion(+)
 create mode 100644 branch file.txt
```

## Merge any two branches

To merge a branch in any branch:

- First reach to the target branch

```
git checkout branch_name
```

- Merge the branch to target branch

```
git merge new_branch
```

```
singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (newbranch)
$ git checkout master
Switched to branch 'master'

singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (master)
$ git merge newbranch
Updating 9e7f7d0..3eb93e9
Fast-forward
 branch file.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 branch file.txt

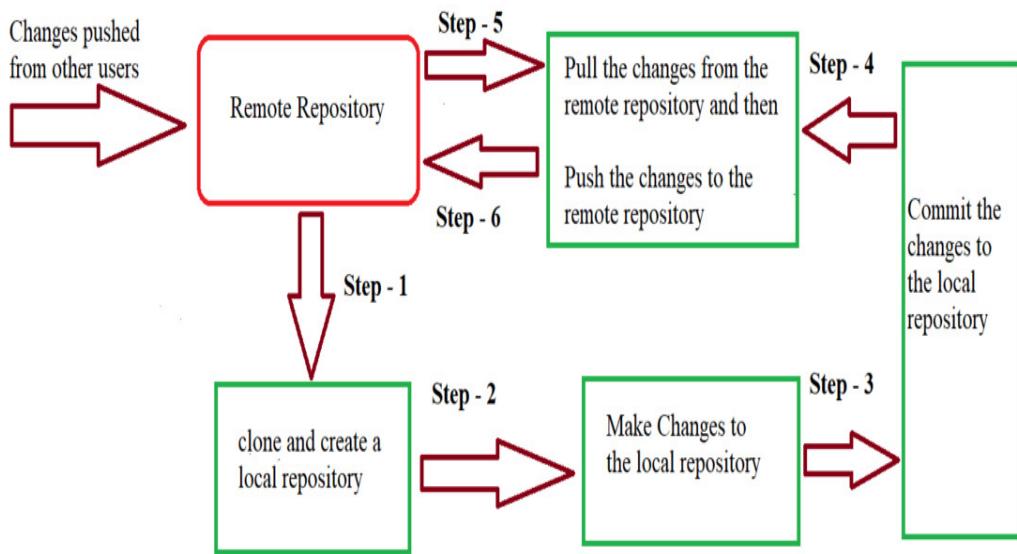
singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (master)
$ |
```



## Aim: GIT lifecycle Description

### Description :

- ❖ Git is used in our day-to-day work, we use git for keeping a track of our files,
- ❖ Let us look at the Life Cycle that git has and understand more about its life cycle.
- ❖ Let us see some of the basic steps that we follow while working with Git –



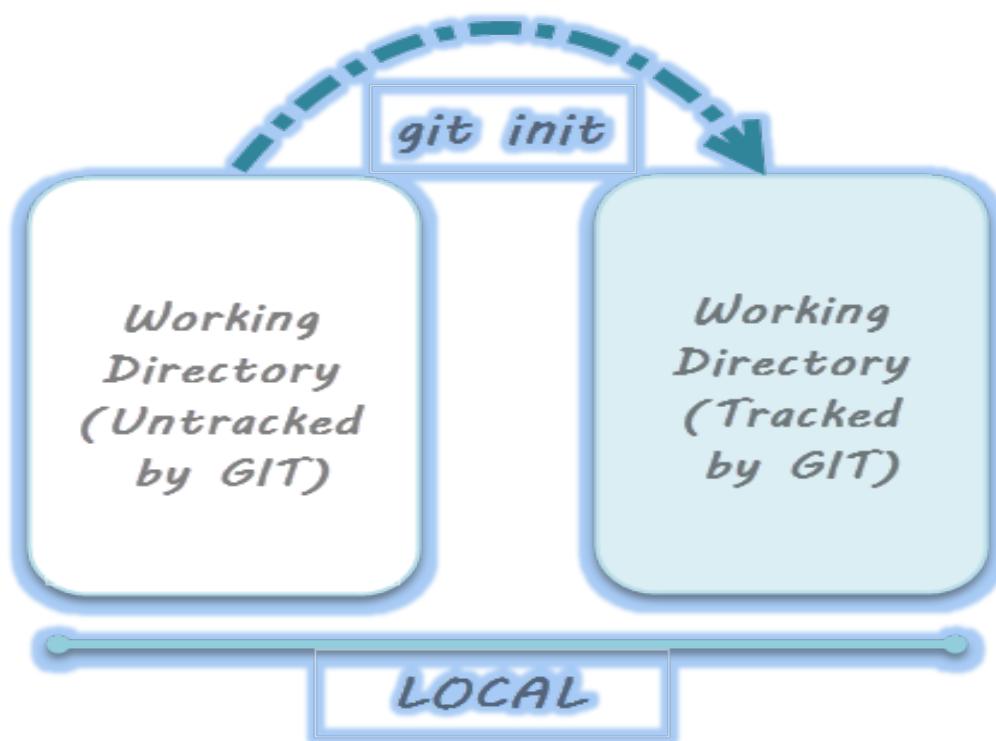
- **In Step - 1**, We first clone any of the code residing in the remote repository to make our own local repository.
- **In Step-2** we edit the files that we have cloned in our local repository and make the necessary changes in it.
- **In Step-3** we commit our changes by first adding them to our staging area and committing them with a commit message.
- **In Step - 4 and Step-5** we first check whether there are any of the changes done in the remote repository by some other users and we first pull that changes.
- If there are no changes we directly proceed with **Step - 6** in which we push our changes to the remote repository and we are done with our work.

When a directory is made a git repository, there are mainly 3 states which make the essence of Git Version Control System. The three states are –

- Working Directory
- Staging area
- GIT Directory

## Working Directory

- Whenever we want to initialize our local project directory to make it a git repository, we use the ***git init*** command.
- After this command, git becomes aware of the files in the project although it doesn't track the files yet.
- The files are further tracked in the staging area.



## Staging Area

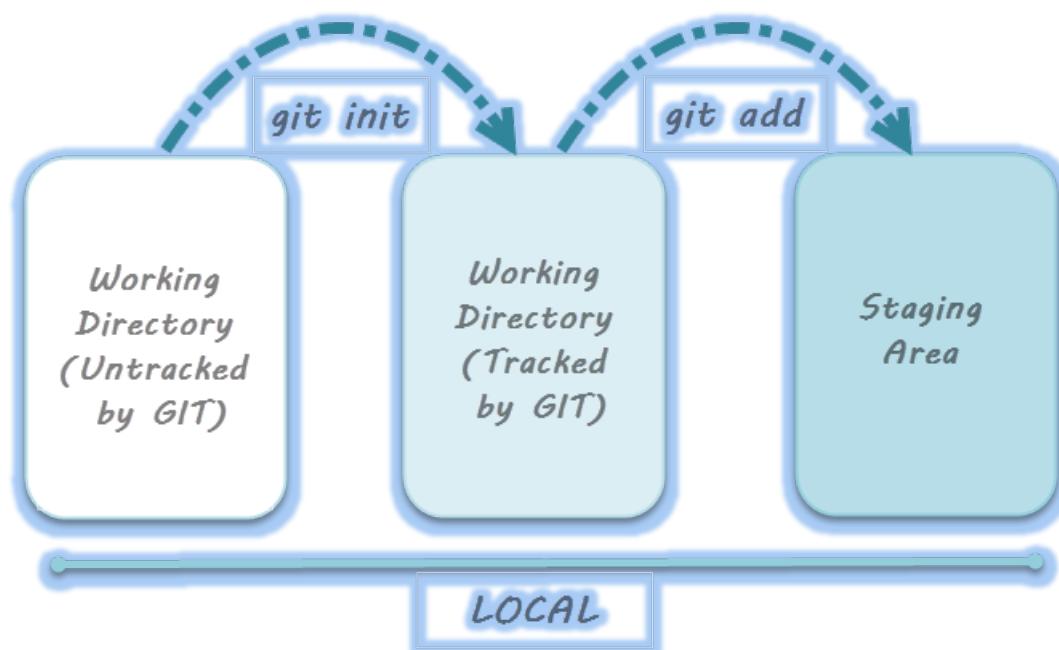
Now, to track the different versions of our files we use the command **git add** which copies the version of your file from your working directory to the staging area.

*// to specify which file to add to the staging area*

*git add <filename>*

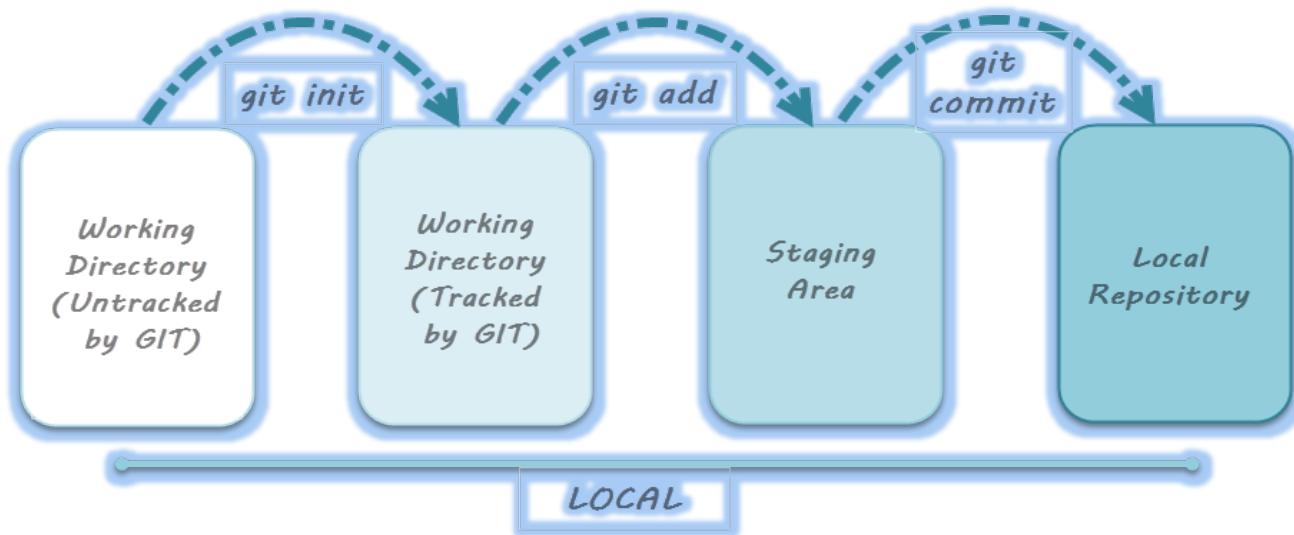
*// to add all files of the working directory to the staging area*

*git add .*

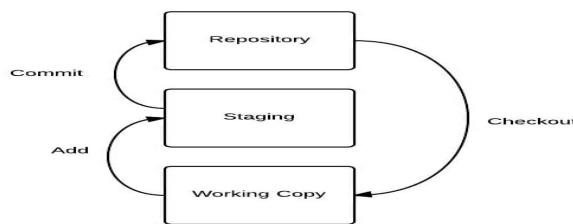


### 3. Git Directory

Now since we have all the files that are to be tracked and are ready in the staging area, we are ready to commit our files using the ***git commit*** command.



- So, we commit this group of files along with a commit message explaining what is the commit about.
- Now, a snapshot of the files in the commit is recorded by Git.
- The information related to this commit (*names of files committed, date and time of commit, author of commit, commit message*) is stored in the Git directory.
- Thus, Git directory is the database where metadata about project files' history will be tracked.



(Reference from the site: <https://www.toolsqa.com/git/merge-branch-in-git/>)

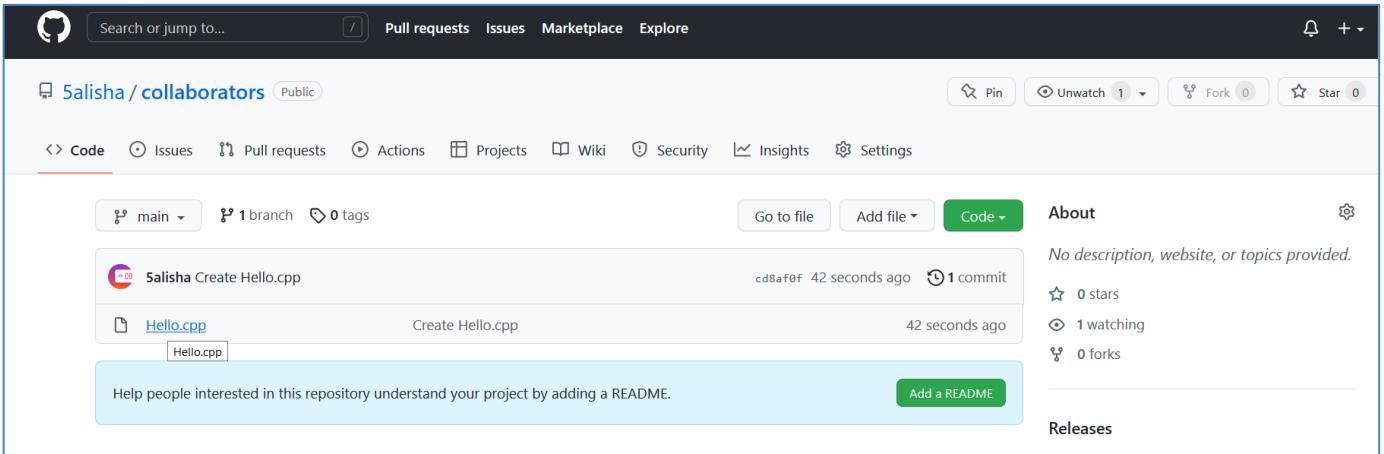
## Aim: Add collaborators on Github Repository

### Description:

Collaboration is the way different people can work on the same project together. It is like creating a group in GitHub just like Groups in other social media. The people added to the collaborator's list can be able to push, merge, and do other kinds of similar things on the project.

### Steps:

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

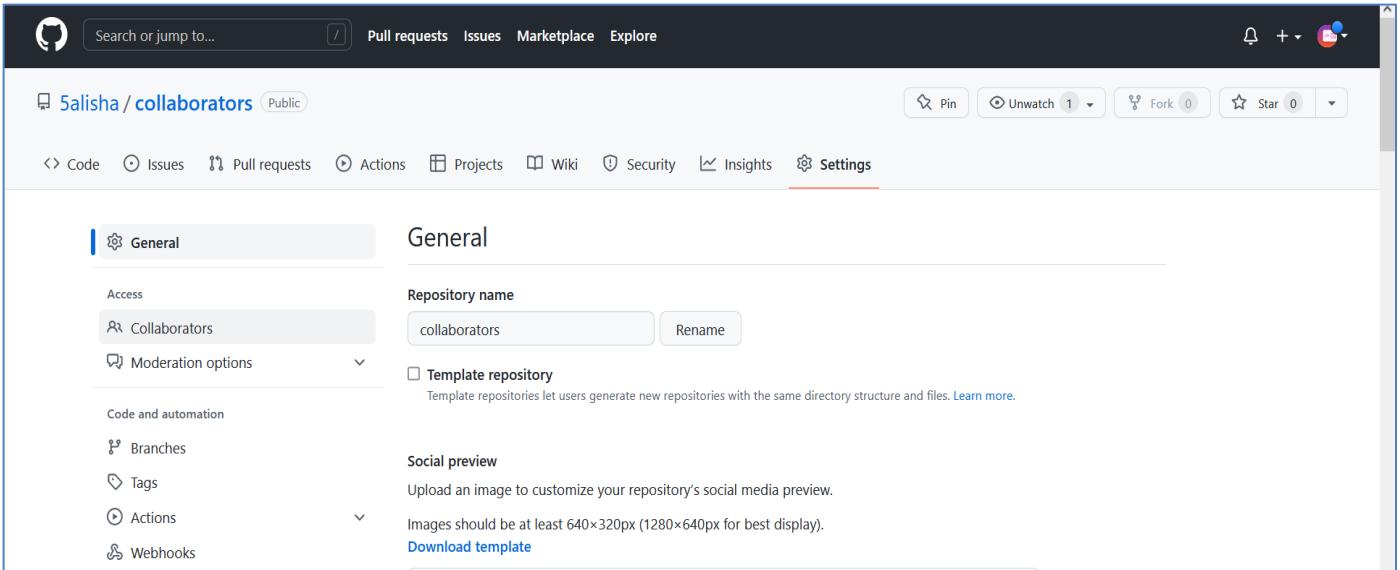


The screenshot shows the GitHub repository page for 'Salisha / collaborators'. The repository is public and contains one commit. The commit details are as follows:

- Author: Salisha
- Commit message: Create Hello.cpp
- Date: 42 seconds ago
- SHA: cd8af0f
- Commits: 1

The repository has 0 stars, 1 watching, and 0 forks. There is a button to 'Add a README'.

- Under your repository name, click **Settings**. In the "Access" section of the sidebar, click **Collaborators**



The screenshot shows the GitHub repository settings page for 'Salisha / collaborators'. The 'General' tab is selected in the sidebar. The 'Access' section is highlighted.

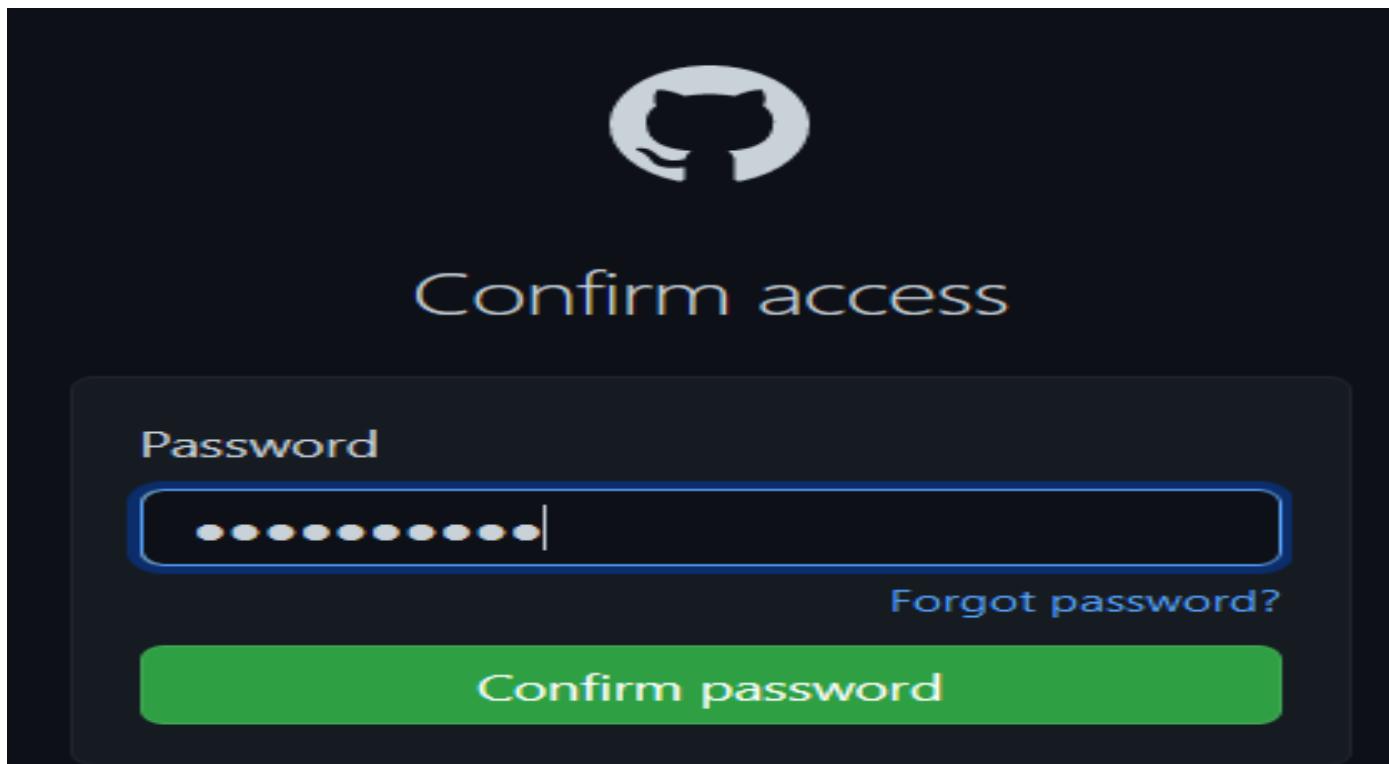
**General** tab settings:

- Repository name:** collaborators
- Template repository:** (checkbox)
- Social preview:** (instructions and download template link)

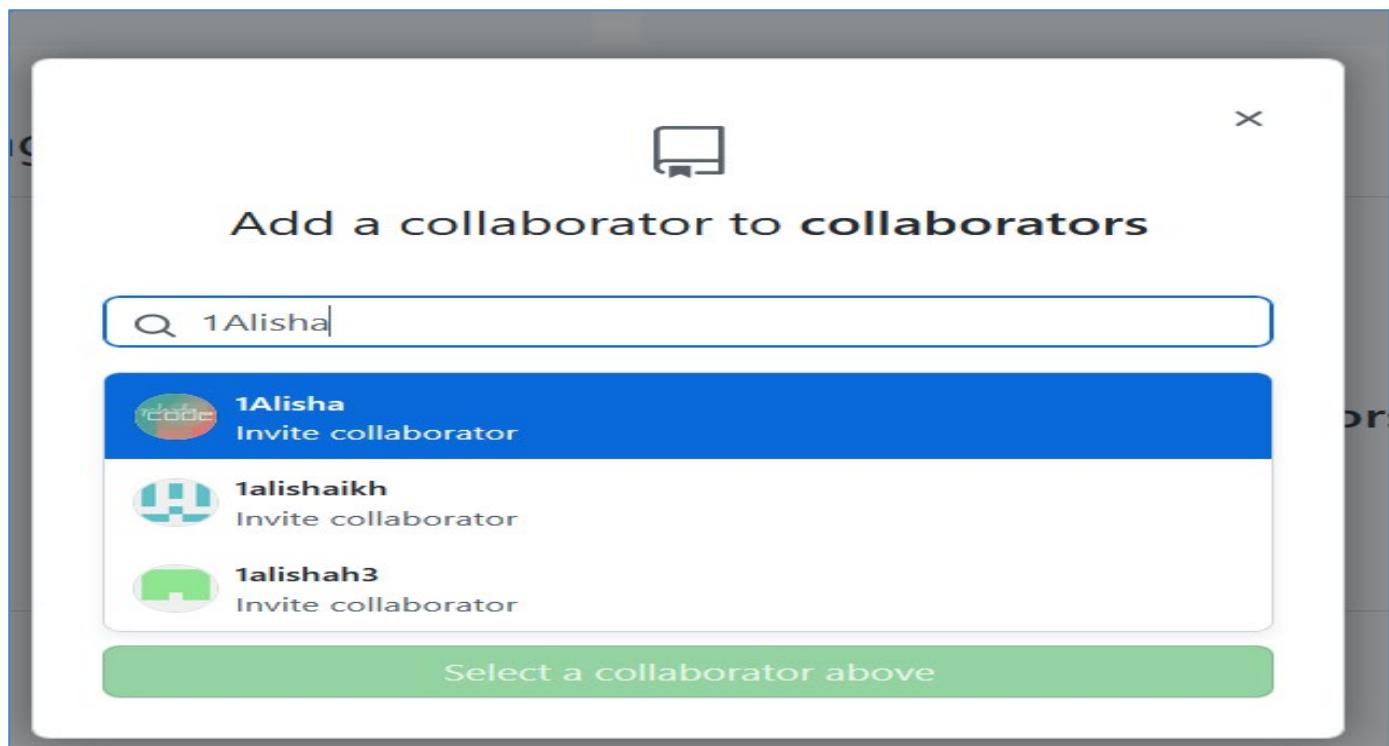
**Access** sidebar options:

- General
- Collaborators (selected)
- Moderation options
- Code and automation
- Branches
- Tags
- Actions
- Webhooks

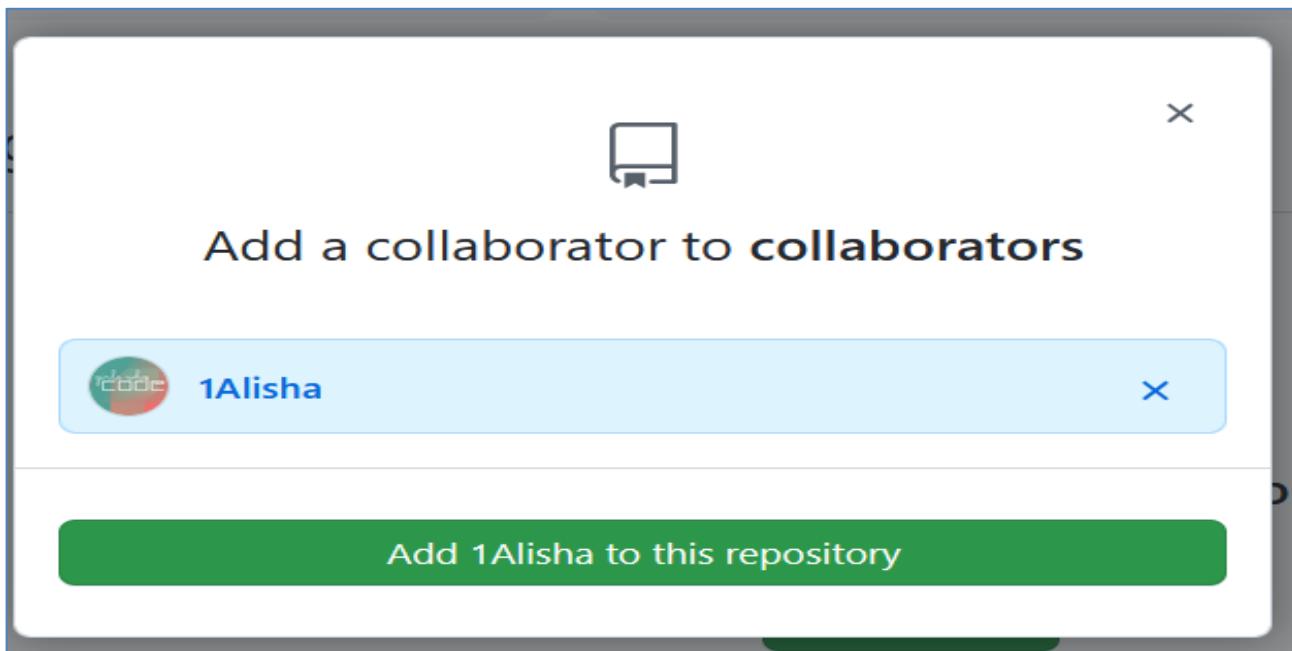
- Then a confirm password page may appear, enter your password for the confirmation.



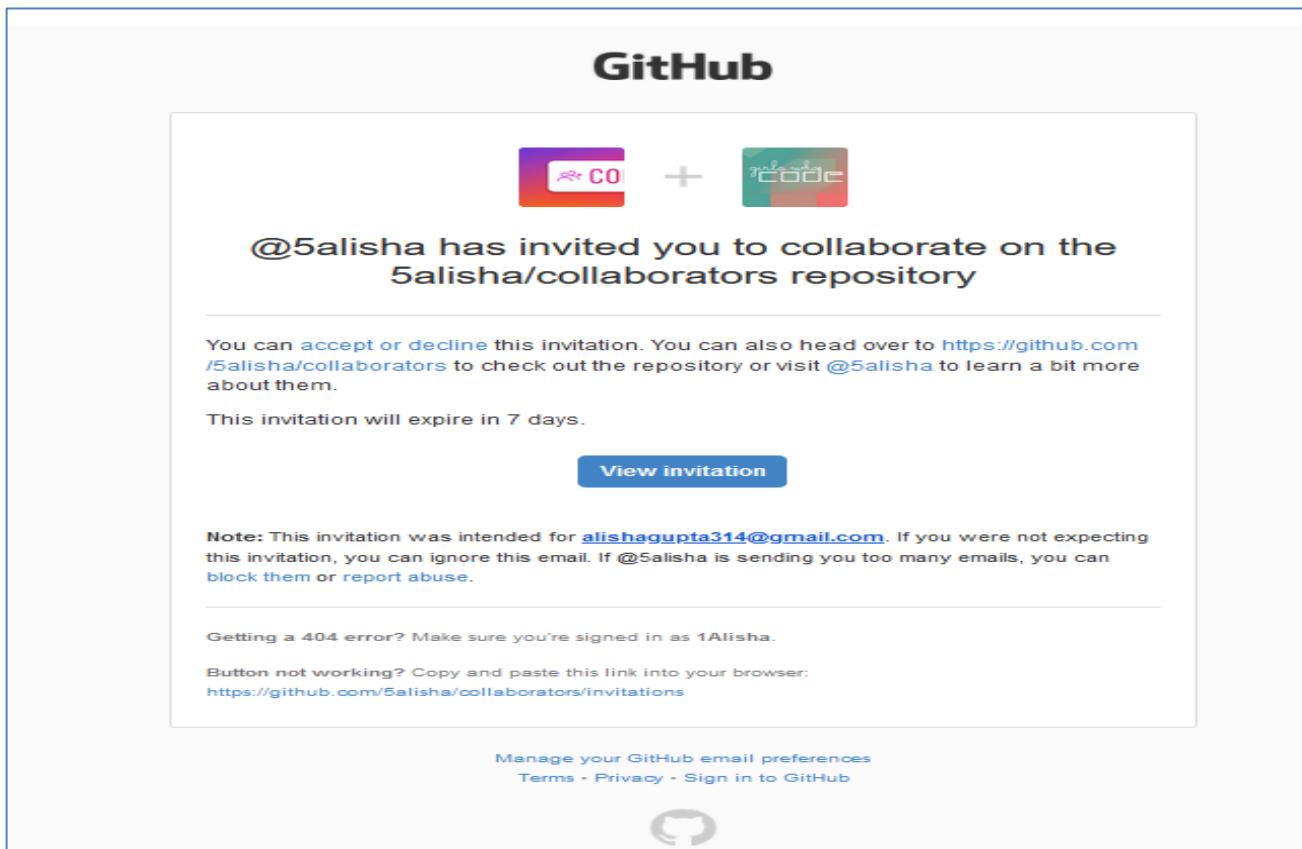
- Click **Invite a collaborator**. In the search field, start typing the name of person you want to invite, then click a name in the list of matches.



- Click Add NAME to REPOSITORY.



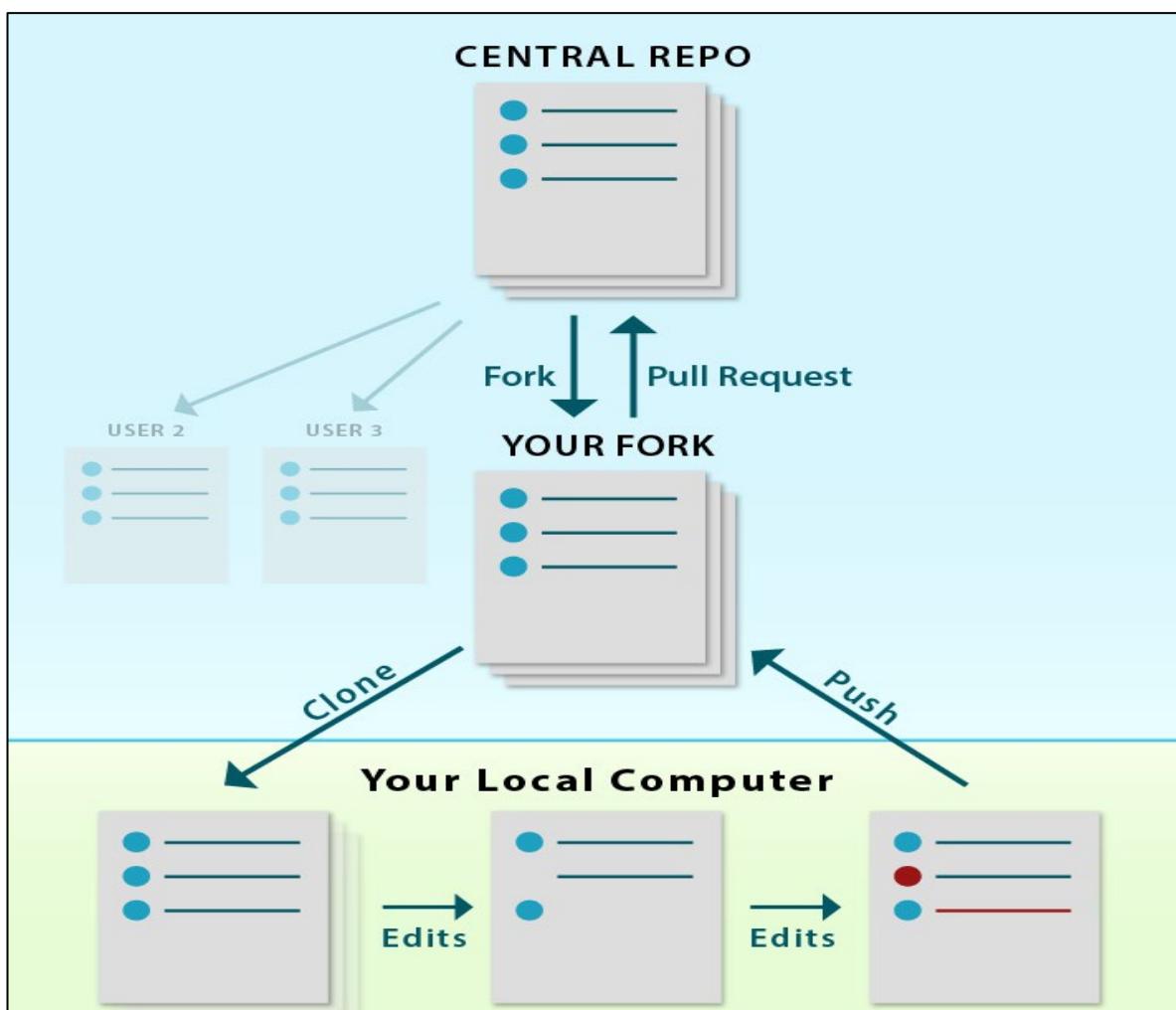
- The user will receive an email inviting them to the repository. Once invitation is accepted , the user will have collaborator access to the repository.



**Aim:** Fork and commit

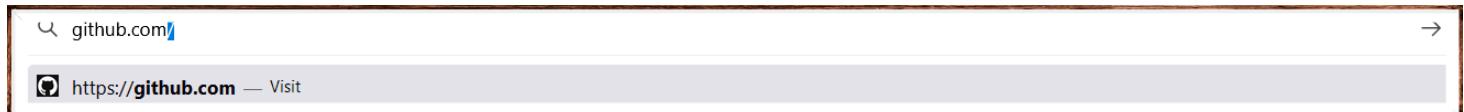
**Description:**

- ❖ When a repository is forked, an exact copy of the repository is made in user's account. Thus, user can freely modify it as he/she wishes. The forking workflow is simple to learn and let' get started.
- ❖ Forking is an excellent tool for copying source code from someone's repository to your repository and contributing to it. The forking workflow is simple to learn and let' get started.

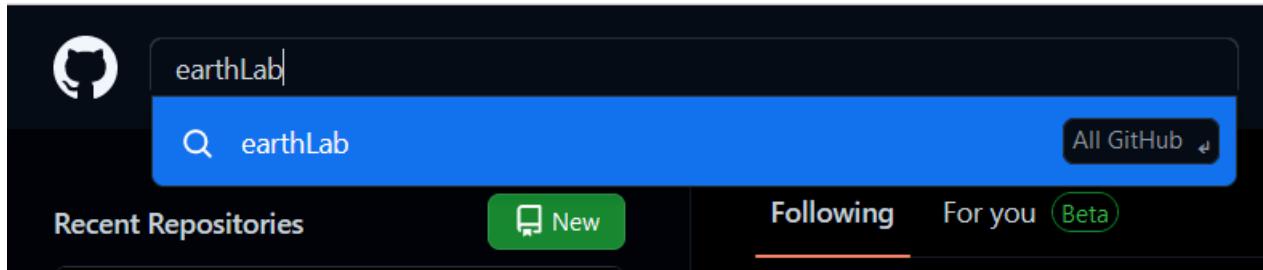


## Steps to be followed:

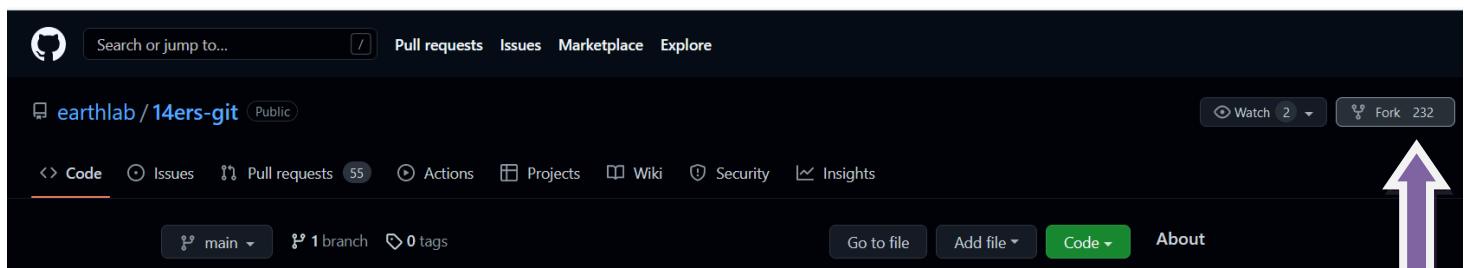
- On the github.com website, login to your github account.



- Navigate to desired repository that you want to fork.



- Click fork button.

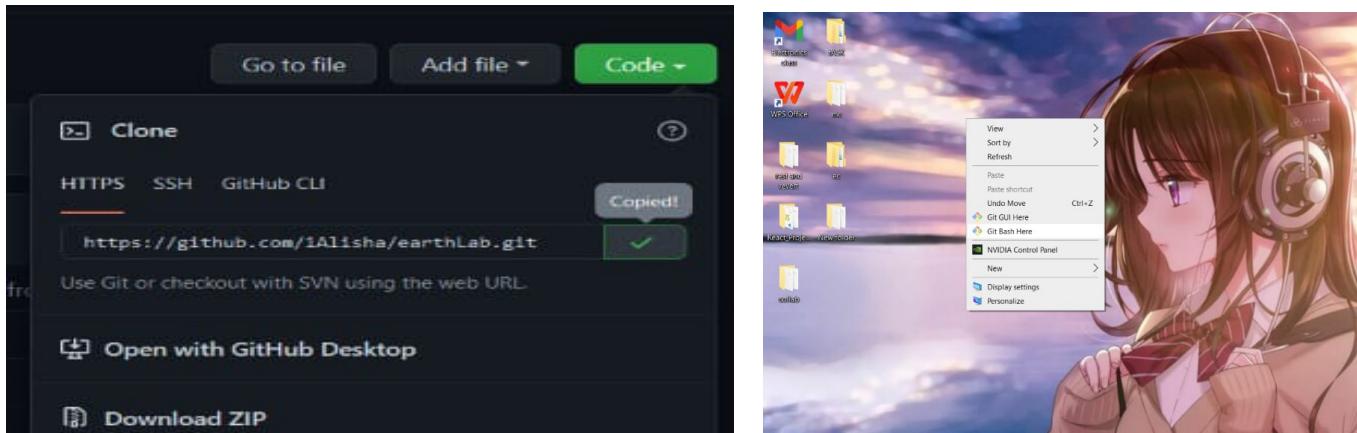


- Fork the repository into your account.

A screenshot of the "Create a new fork" dialog box. It shows the "Owner" dropdown set to "1Alisha" and the "Repository name" field set to "earthLab". A note says "By default, forks are named after the parent repository. You can customize the name to distinguish it further." Below the form is a note: "You are creating a fork in your personal account." At the bottom is a green "Create fork" button.

The left side of the screen shows a progress message: "Forking earthlab/14ers-git" and "It should only take a few seconds." with a "Refresh" button.

- Clone the fork of your repository, so that you can edit the contents locally.



- Cloning the “earthLab” repository into Desktop folder.

```
MINGW64:/c/Users/shubh/Desktop/earthLab
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop (master)
$ git clone https://github.com/1Alisha/earthLab.git
cloning into 'earthLab'...
remote: Enumerating objects: 72, done.
remote: Counting objects: 100% (25/25), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 72 (delta 19), reused 19 (delta 19), pack-reused 47
Receiving objects: 100% (72/72), 9.98 KiB | 123.00 KiB/s, done.
Resolving deltas: 100% (35/35), done.

shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop (master)
$ cd earthLab

shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/earthLab (main)
```

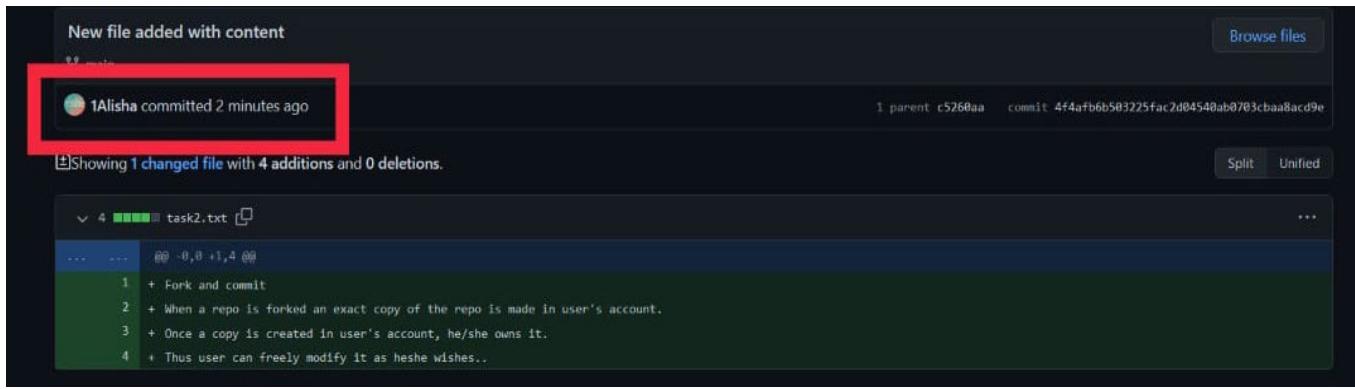
- Make edits to your local cloned copy of the repository on your computer.  
 ➤ Add, Commit , Push those edits back to your fork on github, go on github ,we see that repository has a new commit

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/earthLab (main)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/earthLab (main)
$ git remote
origin

shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/earthLab (main)
$ git push -u origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 416 bytes | 416.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/1Alisha/earthLab.git
  c5260aa..4f4afb6 main -> main
branch 'main' set up to track 'origin/main'.
```



New file added with content

1Alisha committed 2 minutes ago

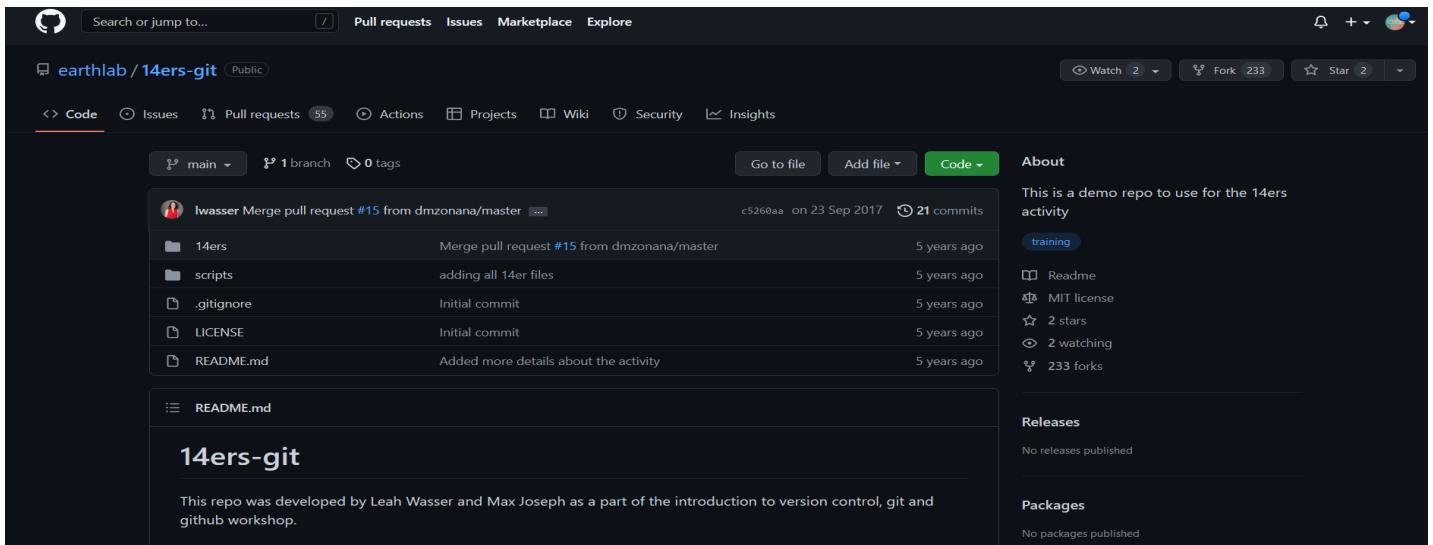
Showing 1 changed file with 4 additions and 0 deletions.

```

task2.txt
@@ -0,0 +1,4 @@
1 + Fork and commit
2 + When a repo is forked an exact copy of the repo is made in user's account.
3 + Once a copy is created in user's account, he/she owns it.
4 + Thus user can freely modify it as he/she wishes..

```

- Suggest the changes that you made, to be added to the earthLab Central repository using a pull request if it accepts the request then the changes will be reflected in the central repository otherwise it will not be .
- Here , we have not suggested the changes to the earthLab repo, so no changes will be reflected in it.



earthlab / 14ers-git Public

Code Issues Pull requests 55 Actions Projects Wiki Security Insights

main · 1 branch · 0 tags

lwasser Merge pull request #15 from dmzonana/master ... c5260aa on 23 Sep 2017 21 commits

14ers	Merge pull request #15 from dmzonana/master	5 years ago	
scripts	adding all 14er files	5 years ago	
.gitignore	Initial commit	5 years ago	
LICENSE	Initial commit	5 years ago	
README.md	Added more details about the activity	5 years ago	

About

This is a demo repo to use for the 14ers activity

training

Readme MIT license 2 stars 2 watching 233 forks

Releases

No releases published

Packages

No packages published

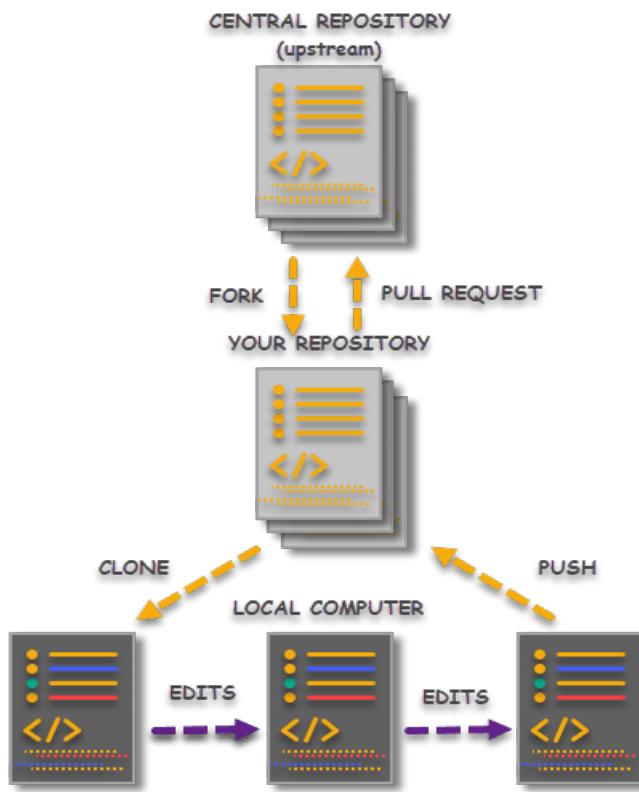
## Tips:

- User can change the name of the forked repository and it will still be connected to the central repository from which it was forked.
- If the user modifies his/her forked repository, the changes will not be reflected in the central repository until user merge his/her fork with the central repository.

## Updating your fork:

It may happen that while you are editing your fork ,other changes are made to the original repository. To fetch These changes into your fork, use these commands in your fork workspace

- Add the original repository as remote repository called “upstream”
- Fetch all the changes from the upstream repository
- Switch to the master branch of your fork
- Merge changes from the upstream repository into your fork



**Note:** Both git fork and clone creates copies of a repository, but they offer drastically different levels of access, isolation & control over the target repository.

**Aim:** Merge conflict created due to own activity

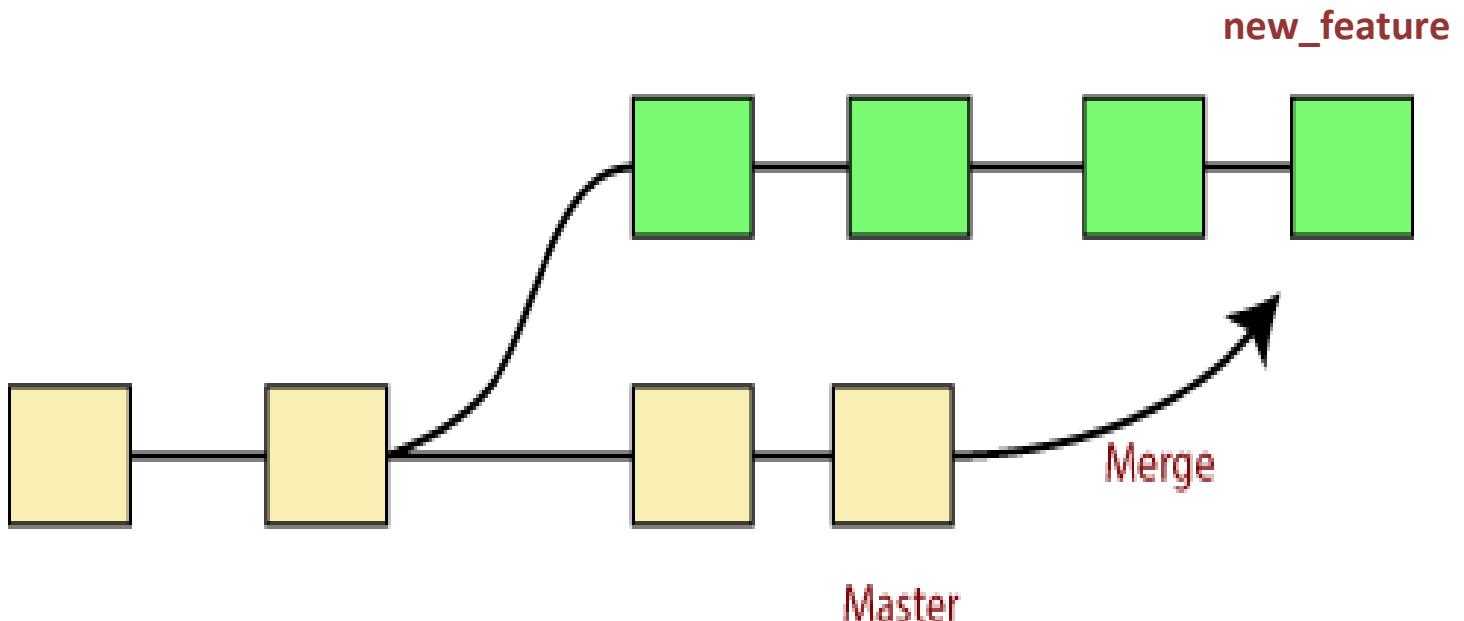
**Description:**

A merge conflict is an event that takes place when Git is unable to automatically resolve differences in code between two commits. Git can merge the changes automatically only if the commits are on different lines or branches. The Git merge command combines separate branches and resolves any conflicting edits.

Let's understand with the help of following example:

➤ Here, we have two branches :

- master
- new\_feature



## Steps to be followed to resolve merge conflicts created due to own activity:

- Initialized empty git repository
- Create a new file code.html and add content in it ; stage it and commit it.
- Push the changes on github.

```
MINGW64:/c/Users/shubh/Desktop/mini
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/mini (master)
$ git init
Initialized empty Git repository in C:/Users/shubh/Desktop/mini/.git/
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/mini (master)
$ vi code.html
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/mini (master)
$ git add --a
warning: LF will be replaced by CRLF in code.html.
The file will have its original line endings in your working directory
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/mini (master)
$ git commit -m "master"
[master (root-commit) 82564f6] master
 1 file changed, 12 insertions(+)
 create mode 100644 code.html
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/mini (master)
$ git remote add origin https://github.com/1A1isha/mini.git
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/mini (master)
$ git push -u origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 290 bytes | 290.00 KiB/s,
done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
```

- Create a new branch(new\_feature) and modify code.html & commit it.

```
MINGW64:/c/Users/shubh/Desktop/mini
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/mini (master)
$ git checkout -b new_feature
Switched to a new branch 'new_feature'

shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/mini (new_feature)
$ vi code.html

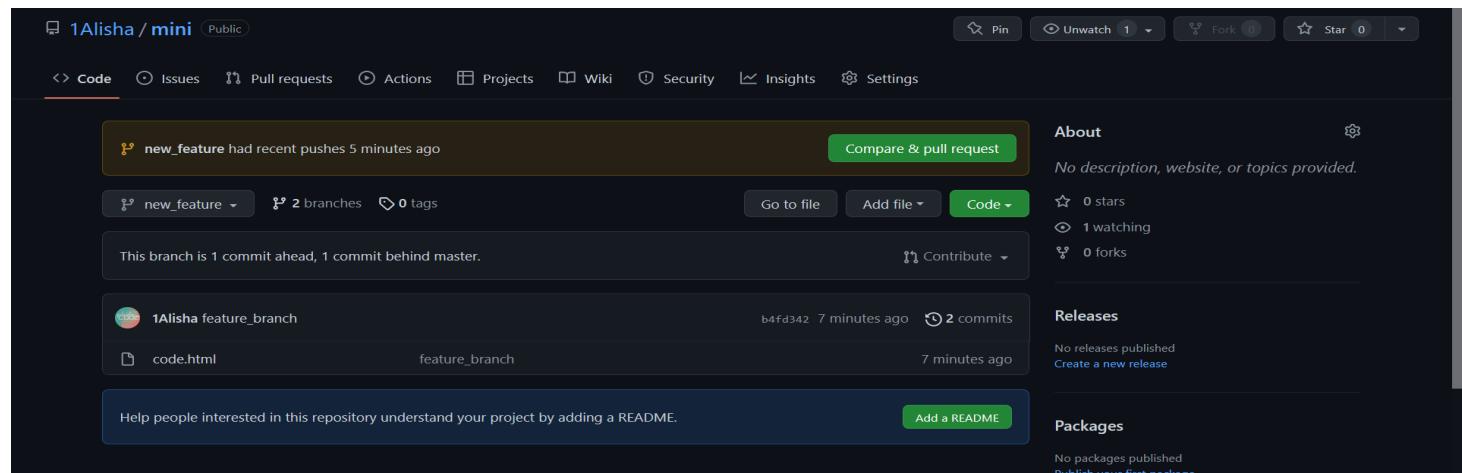
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/mini (new_feature)
$ git add --a
warning: LF will be replaced by CRLF in code.html.
The file will have its original line endings in your working directory

shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/mini (new_feature)
$ git commit -m "feature_branch"
[new_feature b4fd342] feature_branch
 1 file changed, 1 insertion(+), 1 deletion(-)
```

- Push the changes on github

```
MINGW64:/c/Users/shubh/Desktop/mini
shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/mini (new_feature)
$ git push -u origin new_feature
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 322 bytes | 322.00 KiB/s,
done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'new_feature' on GitHub by visiting:
remote:     https://github.com/1Alisha/mini/pull/new/new_feature
remote:
To https://github.com/1Alisha/mini.git
 * [new branch]      new_feature -> new_feature
branch 'new_feature' set up to track 'origin/new_feature'.
```

- On the github, we can see that new\_feature branch had recent pushes 5 minutes ago.



1Alisha / mini Public

**Code** Issues Pull requests Actions Projects Wiki Security Insights Settings

**About**  
No description, website, or topics provided.

**Branches**  
new\_feature (selected) 2 branches 0 tags

This branch is 1 commit ahead, 1 commit behind master.

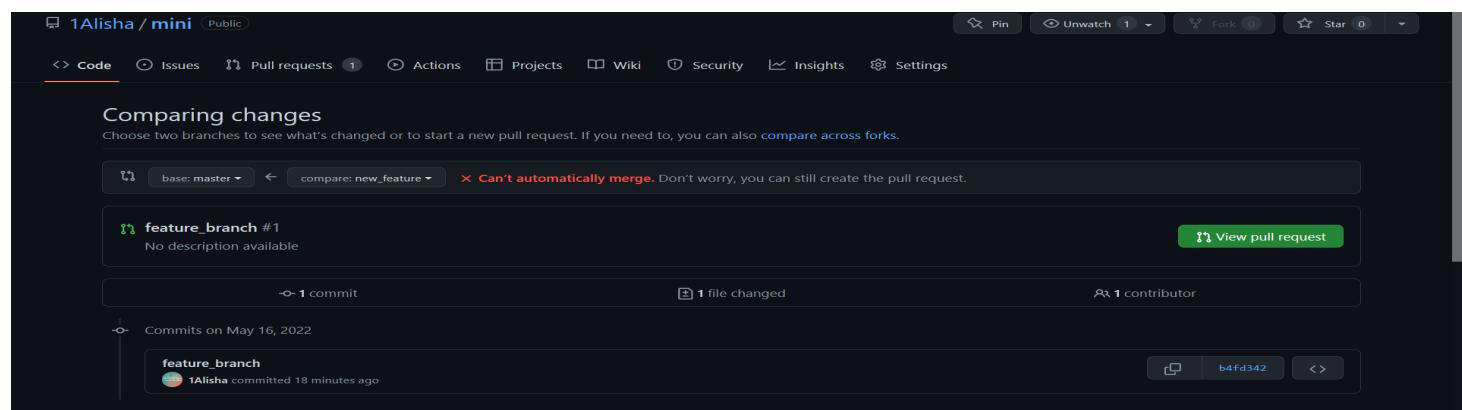
**Commits**  
1Alisha feature\_branch b4fd342 7 minutes ago 2 commits  
code.html feature\_branch 7 minutes ago

**Publish** Help people interested in this repository understand your project by adding a README. Add a README

**Releases**  
No releases published Create a new release

**Packages**  
No packages published Publish your first package

- Now, try to merge master branch and new\_feature branch, it will show merge conflict



1Alisha / mini Public

**Code** Issues Pull requests 1 Actions Projects Wiki Security Insights Settings

**Comparing changes**  
Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks.

base: master compare: new\_feature Can't automatically merge. Don't worry, you can still create the pull request.

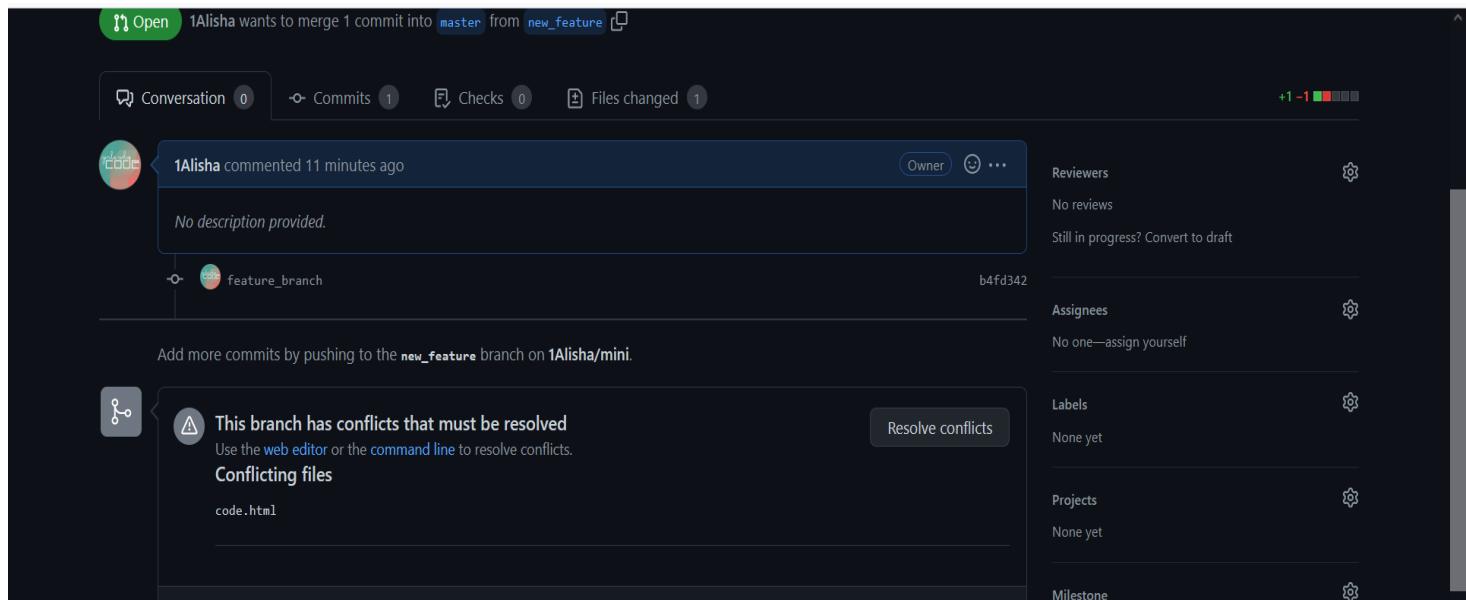
**feature\_branch #1**  
No description available

-> 1 commit 1 file changed 1 contributor

Commits on May 16, 2022

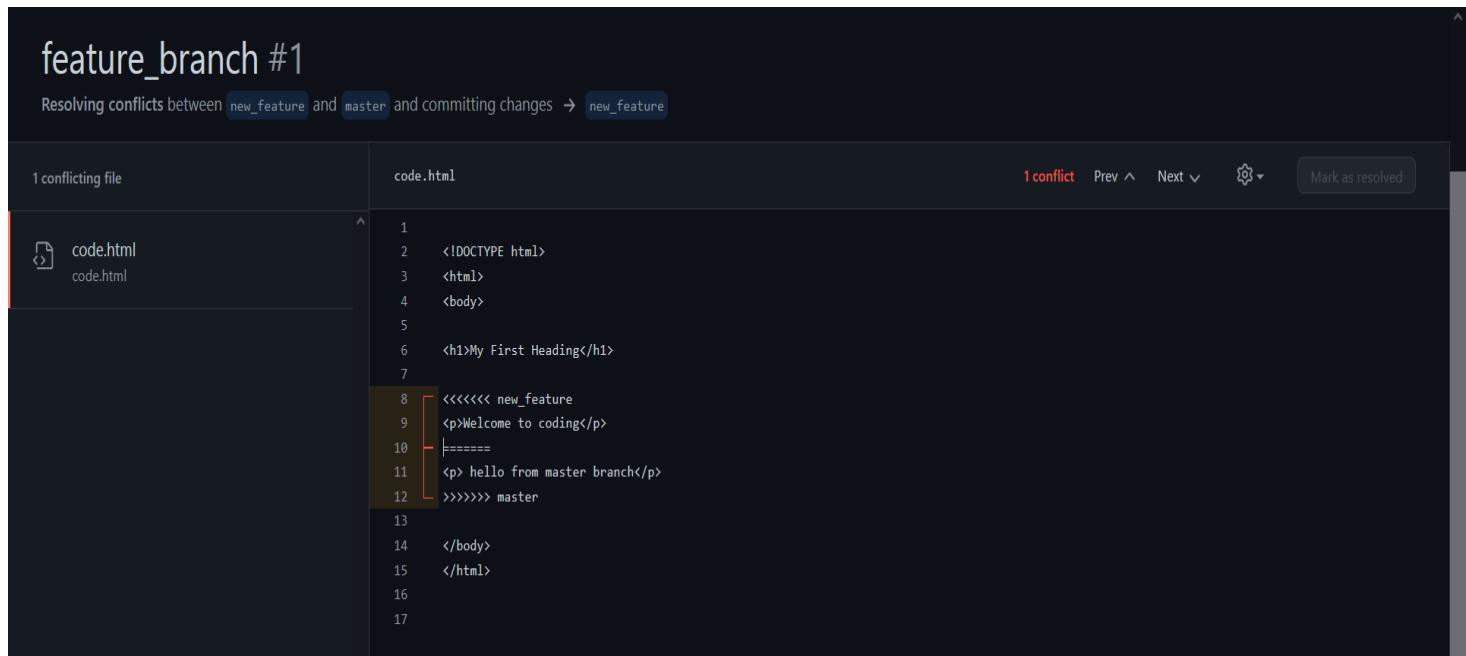
feature\_branch 1Alisha committed 18 minutes ago

- This branch has conflicts that must be resolved for merging both branches.



The screenshot shows a GitHub pull request interface. At the top, it says "1Alisha wants to merge 1 commit into `master` from `new_feature`". Below this, there are tabs for "Conversation" (0), "Commits" (1), "Checks" (0), and "Files changed" (1). A comment from "1Alisha" is shown, stating "No description provided." and "This branch has conflicts that must be resolved". A "Resolve conflicts" button is present. The right side of the screen displays review settings for "Reviewers", "Assignees", "Labels", "Projects", and "Milestone". The commit hash is b4fd342. A note at the bottom says "Add more commits by pushing to the `new_feature` branch on `1Alisha/min`".

- Think of these new lines as "conflict dividers".
- The ===== line is the "center" of the conflict.
- All the content between the center and the <<<<< HEAD line is content that exists in the current branch main which the HEAD ref is pointing to.
- Alternatively all content between the center and >>>>> new\_branch\_to\_merge\_later is content that is present in our merging branch.



The screenshot shows a GitHub conflict resolution interface for the file "code.html". It indicates "1 conflicting file" and "Resolving conflicts between `new_feature` and `master` and committing changes → `new_feature`". The file content is as follows:

```

1
2  <!DOCTYPE html>
3  <html>
4  <body>
5
6  <h1>My First Heading</h1>
7
8  <===== new feature
9  <p>Welcome to coding</p>
10 <=====
11 <p> hello from master branch</p>
12 >>>>> master
13
14  </body>
15  </html>
16
17

```

A red bracket highlights the conflict area between line 8 and line 12. The GitHub interface includes navigation buttons for "1 conflict", "Prev ⌂", "Next ⌂", and "Mark as resolved".

- Let's resolve the merge conflict on git bash using command "git mergetool"

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/mini (master)
$ git merge new_feature
Auto-merging code.html
CONFLICT (content): Merge conflict in code.html
Automatic merge failed; fix conflicts and then commit t
he result.

shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/mini (master|ME
RGING)
$ git mergetool

This message is displayed because 'merge.tool' is not c
onfigured.
See 'git mergetool --tool-help' or 'git help config' fo
r more details.
'git mergetool' will now attempt to use one of the foll
owing tools:
opendiff kdiff3 tkdiff xxdiff meld tortoisemerge gvimdi
ff diffuse diffmerge ecmerge p4merge araxis bc codecomp
are smerge emerge vimdiff nvimdiff
Merging:
code.html

Normal merge conflict for 'code.html':
{local}: modified file
{remote}: modified file
Hit return to start merge resolution tool (vimdiff):
4 files to edit
```

Using command "git mergetool" this window will open , here delete the content which you don't want to keep using backspace key. Enter :wq to come out this window.



```
- <!DOCTYPE html> - <!DOCTYPE html> - <!DOCTYPE html>
<html> <html> <html>
<body> <body> <body>
<h1>My First Hea <h1>My First Hea <h1>My First Heading</h1>
<h1>My First Heading</h1>

<h1>My First Heading</h1>

<p>Welcome to coding</p>

</body>
</html>

~
~
```

- After, editing the file using mergetool command, commit the changes and push them on github.

```

MINGW64/c/Users/shubh/Desktop/mini
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/mini (master|MERGING)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)

Changes to be committed:
  modified:   code.html

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

shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/mini (master|MERGING)
$ git add --a

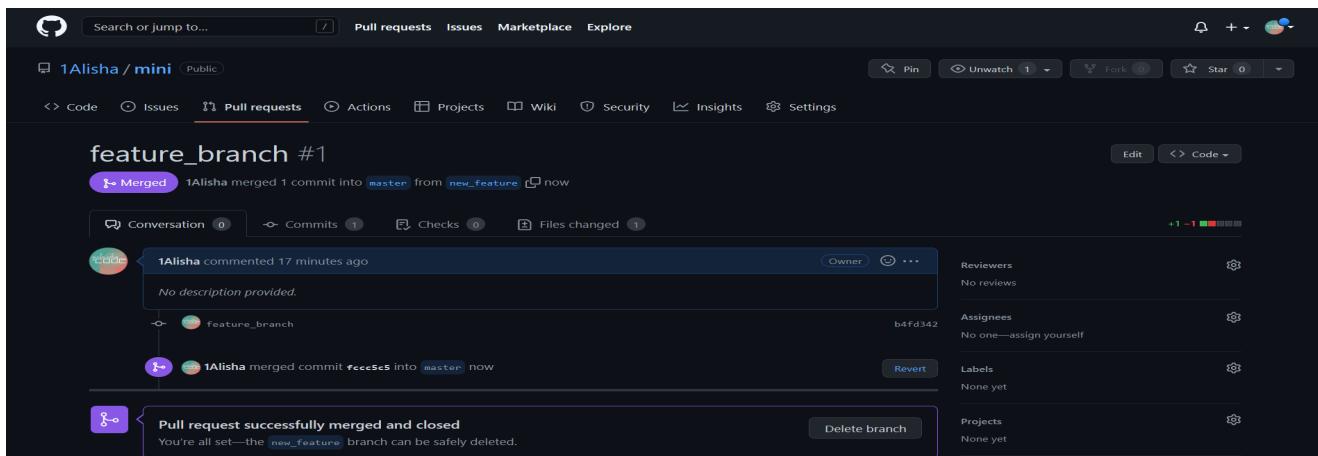
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/mini (master|MERGING)
$ git commit -m "merge conflict resolved"
[master fccc5c5] merge conflict resolved

shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/mini (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/mini (master)
$ git push -u origin master
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 4 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 498 bytes | 498.00 KiB/s, done.
Total 4 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/1Alisha/mini.git
  b40d621..fccc5c5  master -> master
branch 'master' set up to track 'origin/master'.

```

- On the github, it is showing that 1Alisha merged 1 commit into master from new\_feature branch.

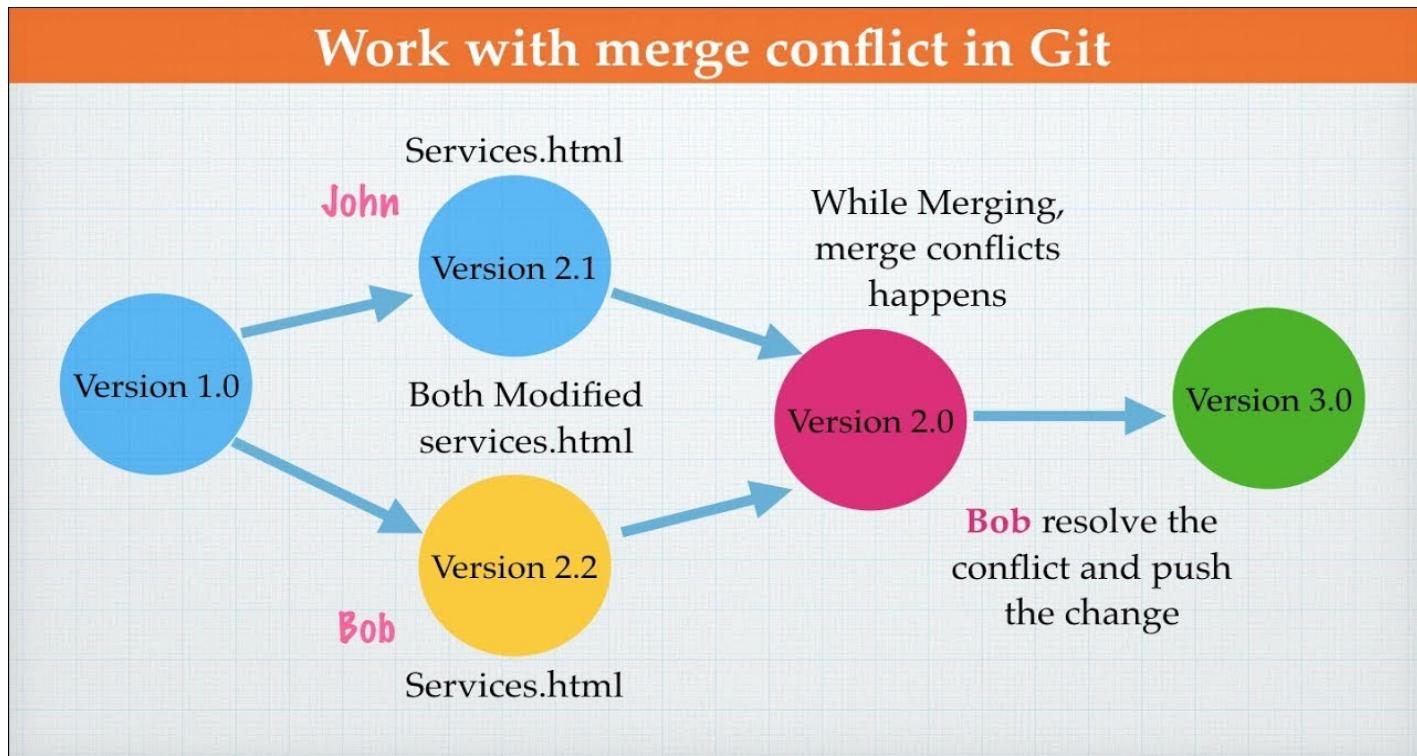


- Hence, the merge conflict due to own activity has been resolved.

**Aim:** Merge conflict created due to Collaborators

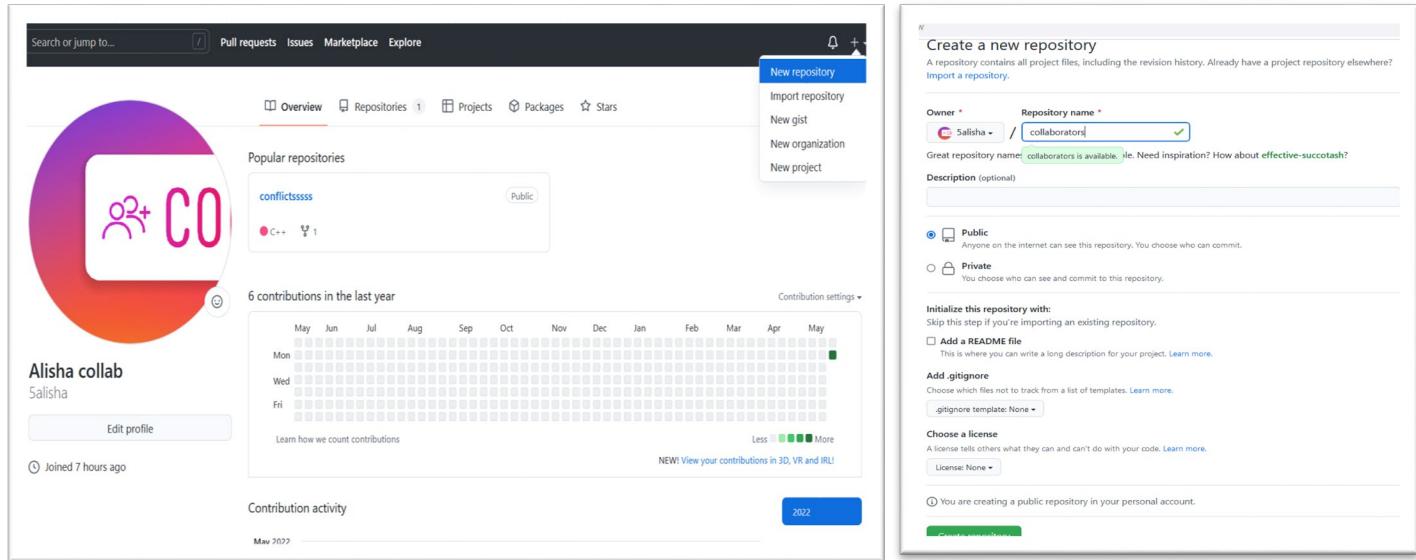
**Description:** When multiple contributors work on the same part of a code or work with numerous branches, merge conflicts are bound to happen. The primary goal of `git merge` is to resolve or *warn* about these conflicts automatically.

Let's understand with the help of following example:



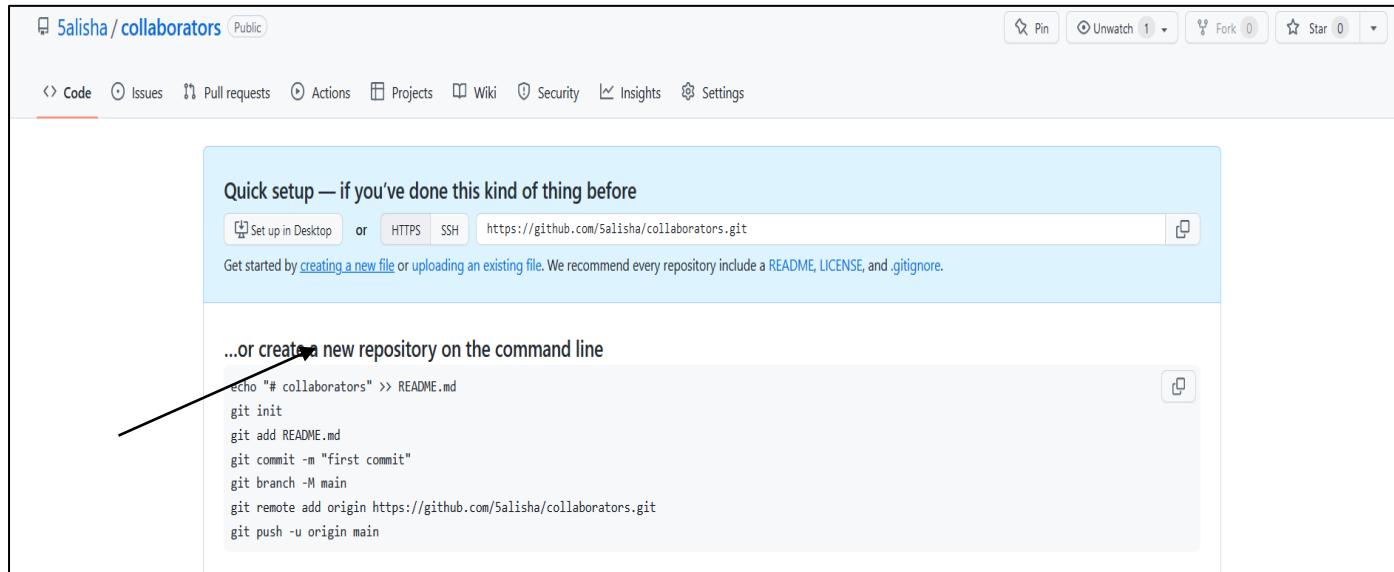
- Let's assume there are two developers: Bob and John.
- Both of them pull the same code file from the remote repository and try to make various amendments in that file.
- After making the changes, John pushes the file back to the remote repository from his local repository.
- Now, when Bob tries to push that file after making the changes from his end, he is unable to do so, as the file has already been changed in the remote repository.
- To prevent such conflicts, developers work in separate isolated branches. The Git merge command combines separate branches and resolves any conflicting edits.
- Hence, Bob resolve the conflict and push the change.

- ❖ On the github, a new repository named “Collaborators ” has been created as the remote Repository.



The image shows two side-by-side screenshots. On the left is a screenshot of a GitHub user profile for 'Alisha collab'. It features a large circular profile picture with a 'CO' logo, a repository named 'conflictsssss' (Public), and a contribution calendar for May 2022 showing activity on Monday, Wednesday, and Friday. On the right is a screenshot of the 'Create a new repository' dialog. The 'Repository name' field contains 'collaborators' with a green checkmark. Other fields include 'Owner' set to 'Salisha', 'Visibility' set to 'Public', and 'Description' left blank. Buttons for 'Add a README file', 'Add .gitignore', 'Choose a license', and 'Create repository' are visible.

- ❖ After creating repo , this window will open where a file named hello .cpp has been created.



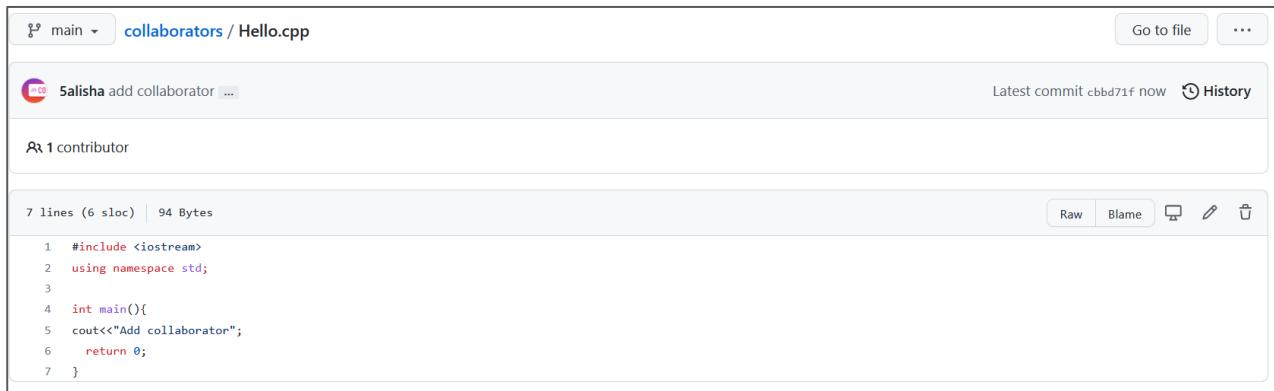
This screenshot shows the 'Code' tab of the 'Salisha / collaborators' repository. A blue box highlights the 'Quick setup — if you've done this kind of thing before' section, which contains instructions for setting up the repository on desktop or via HTTPS/SSH, along with a link to the repository's URL. Below this, another blue box highlights the command-line setup instructions:

```
...or create a new repository on the command line
echo "# collaborators" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/5alisha/collaborators.git
git push -u origin main
```



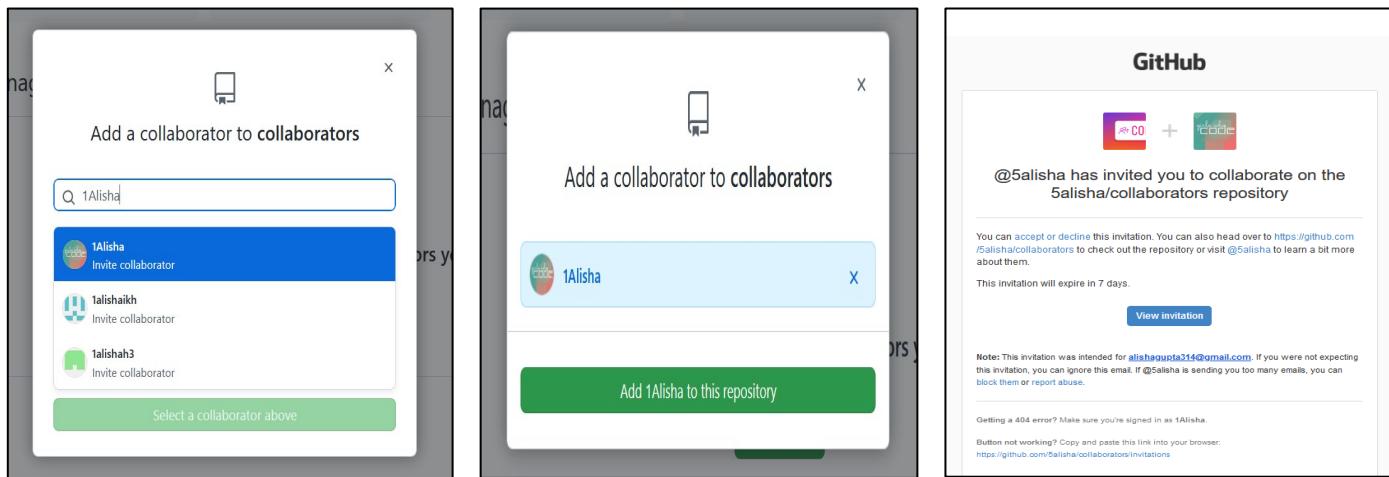
This screenshot shows the repository's history. A commit titled 'Create Hello.cpp' was made by 'Salisha' at 'now'. The commit message is 'Create Hello.cpp'. The commit hash is 'cd8af0f'. The commit details show a single file named 'Hello.cpp' with the content 'Hello.cpp'.

- ❖ A simple c++ code has been added in the file

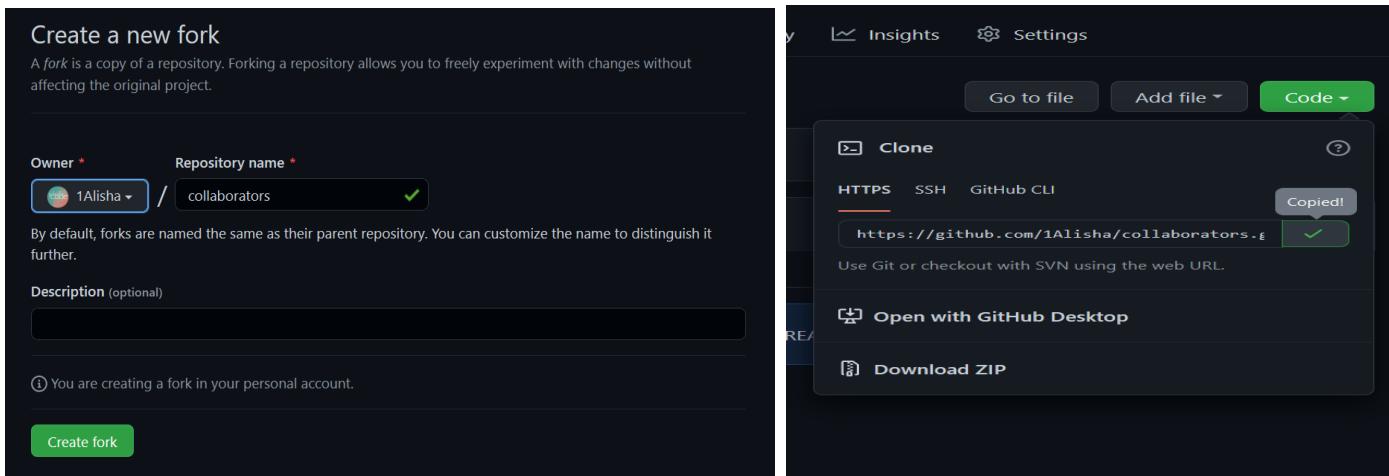


```
#include <iostream>
using namespace std;
int main(){
cout<<"Add collaborator";
return 0;
}
```

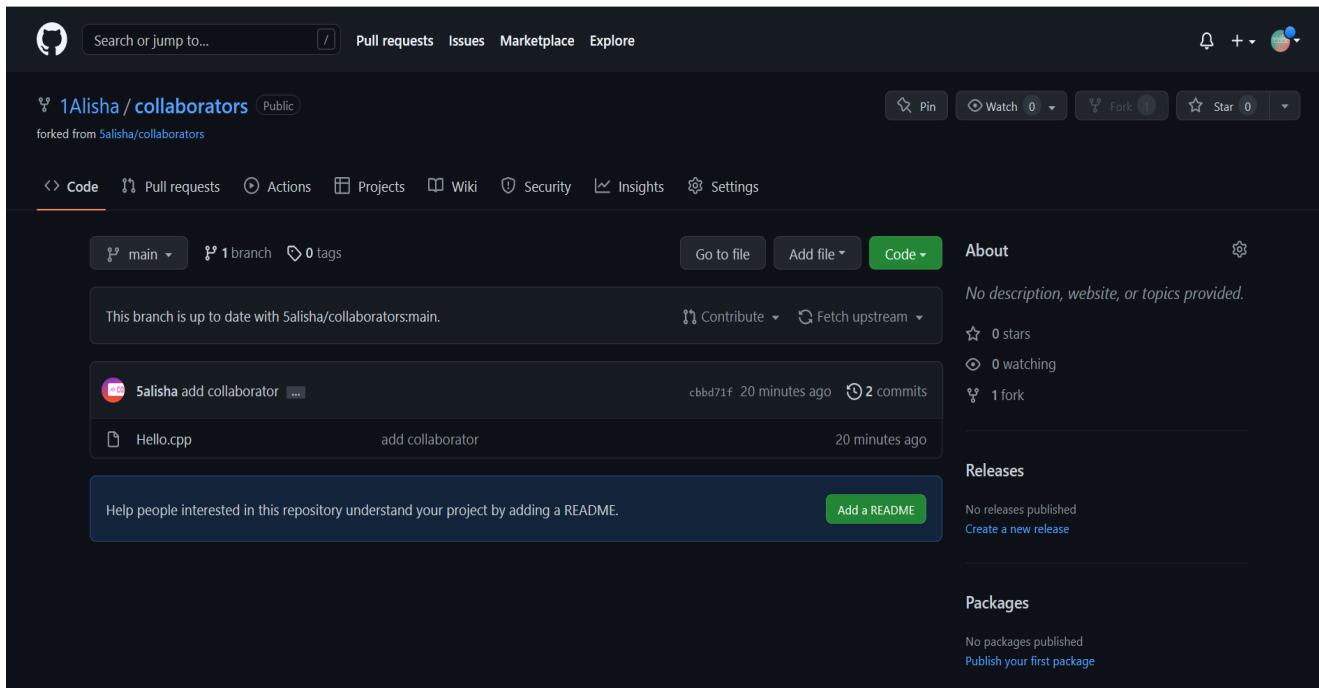
- ❖ Add collaborator, “1Alisha”. As you can see @5alisha has invited 1Alisha to collaborate.



- ❖ 1Alisha has forked the repo and cloned the repo using the url.

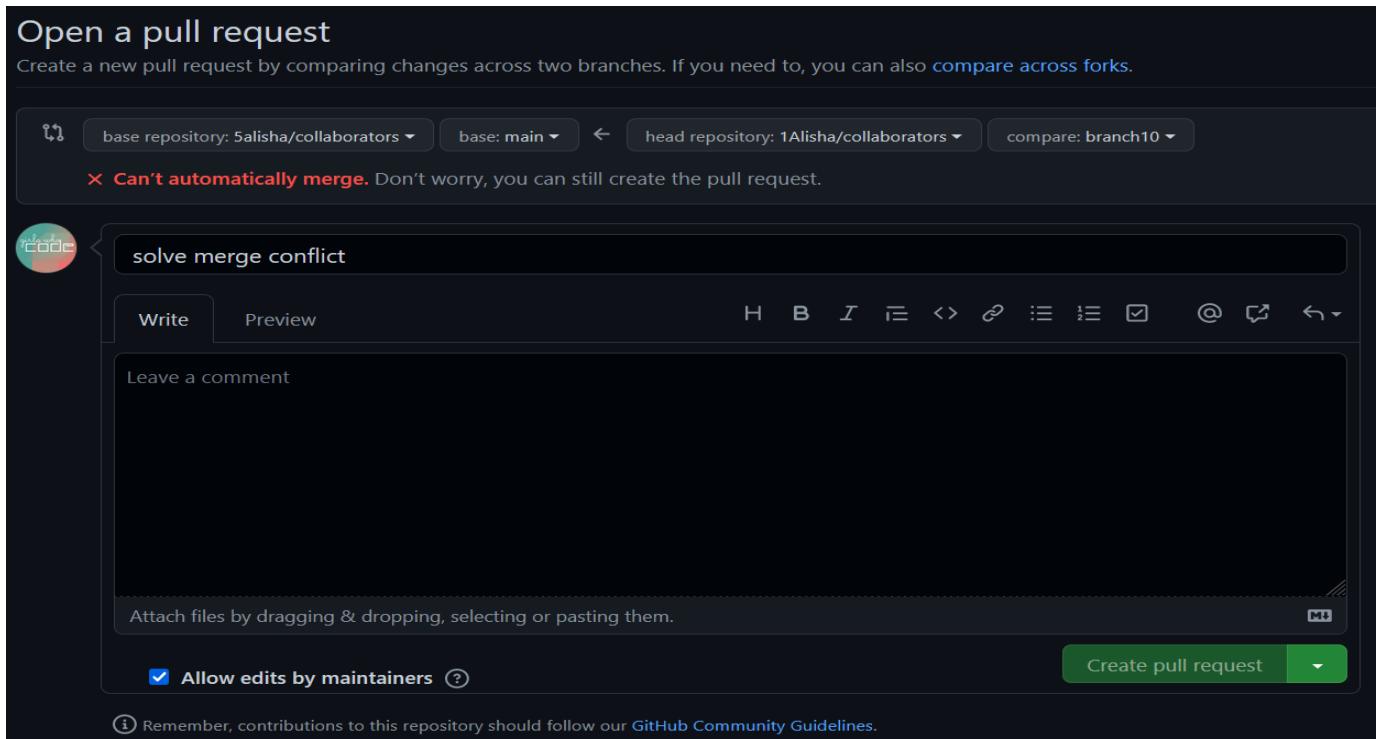


- ❖ Upto here main branch of 1Alisha is upto date with 5alisha/collaborators:main



This screenshot shows a GitHub repository page for '1Alisha / collaborators'. The repository is public and has been forked from '5alisha/collaborators'. The 'Code' tab is selected, showing the 'main' branch. A message indicates that the branch is up-to-date with '5alisha/collaborators:main'. The commit history shows a single commit by 'Salisha' adding a collaborator, made 20 minutes ago. The README section is empty, with a button to 'Add a README'. Repository statistics show 0 stars, 0 watching, and 1 fork.

- ❖ After making a new branch “branch10” and adding different content in same hello.cpp file by 1Alisha, creates a merge conflict.



This screenshot shows the GitHub interface for opening a pull request. The 'base repository' is set to '5alisha/collaborators', 'base' to 'main', 'head repository' to '1Alisha/collaborators', and 'compare' to 'branch10'. A note states 'Can't automatically merge. Don't worry, you can still create the pull request.' The pull request form includes fields for 'solve merge conflict' (with 'Write' selected), 'Preview', and 'Leave a comment'. There is also a note at the bottom: 'Remember, contributions to this repository should follow our GitHub Community Guidelines.' and a 'Create pull request' button.

- ❖ So, we will create a pull request to @5alisha to merge 1commit into @5alisha:main from 1Alisha:branch10

# solve merge conflict #1

[Open](#) 1Alisha wants to merge 1 commit into `1Alisha:main` from `1Alisha:branch10`

Conversation 0    Commits 1    Checks 0    Files changed 1

 1Alisha commented now

No description provided.

 solve merge conflict

3793e98

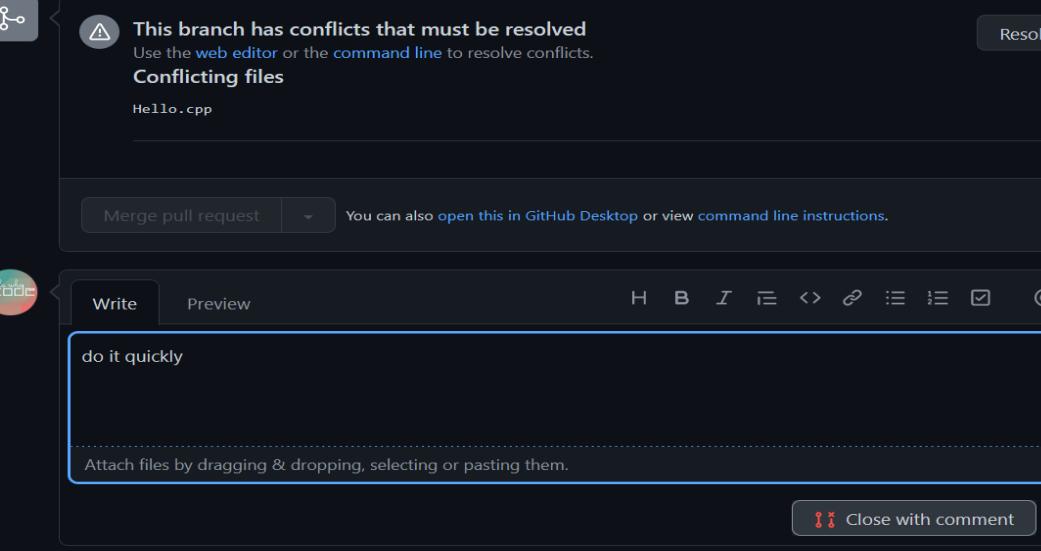
Add more commits by pushing to the `branch10` branch on [1Alisha/collaborators](#).

 **This branch has conflicts that must be resolved**  
Use the [web editor](#) or the [command line](#) to resolve conflicts.  
**Conflicting files**  
`Hello.cpp`

[Resolve conflicts](#)

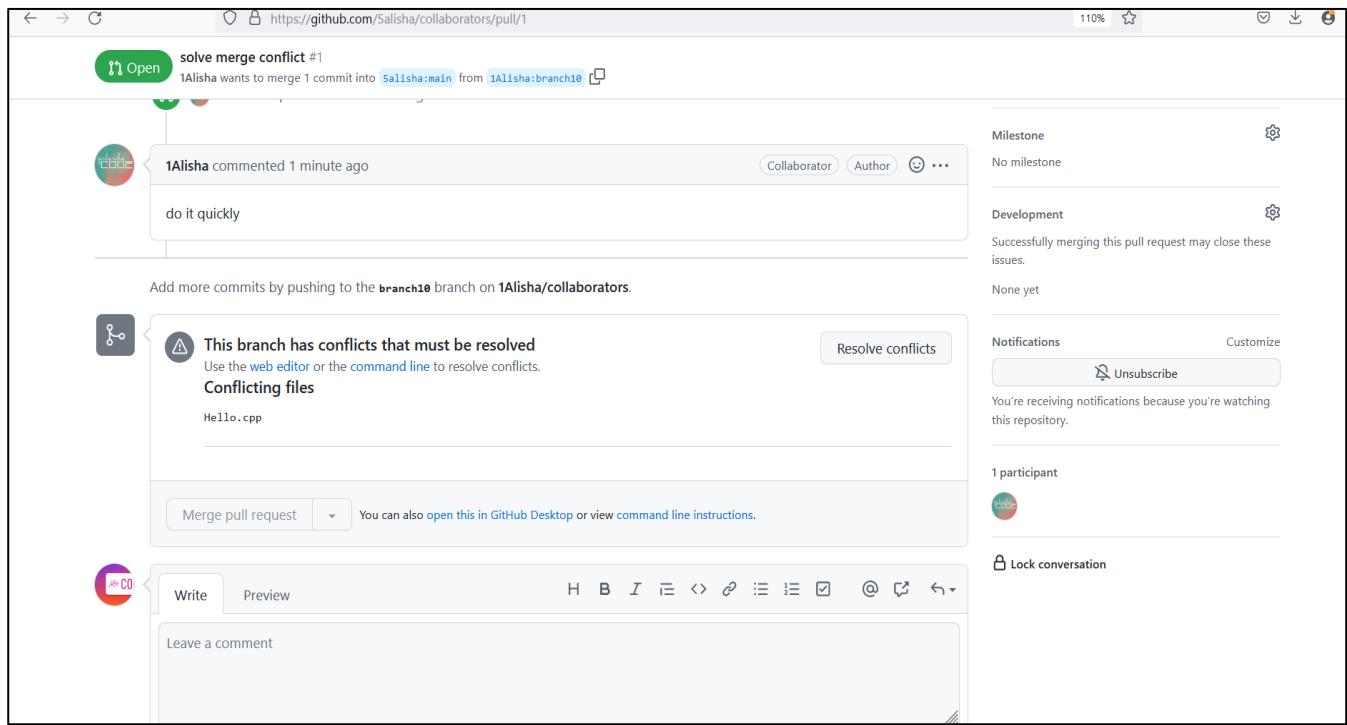
Merge pull request ▾ You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

- ❖ Along with the merge pull request , we can also comment or attach files by dragging & dropping or passing them.



The screenshot shows a GitHub merge conflict resolution interface. At the top, there's a green button labeled "Open" and a status message: "solve merge conflict #1 1Alisha wants to merge 1 commit into Salisha:main from 1Alisha:branch10". Below this, a warning message says "This branch has conflicts that must be resolved" with a link to "Use the web editor or the command line to resolve conflicts." A "Resolve conflicts" button is on the right. On the left, there's a "Conflict files" section showing "Hello.cpp". A "Merge pull request" button with a dropdown arrow is on the left, and a note "You can also open this in GitHub Desktop or view command line instructions." is below it. The main area has tabs "Write" and "Preview" (selected). The "Write" tab contains the text "do it quickly". A "Close with comment" button with a red icon is at the bottom left, and a "Comment" button is at the bottom right. A "ProTip!" link at the bottom right suggests adding comments to specific lines under "Files changed".

- ❖ @5alisha had opened the merge pull request



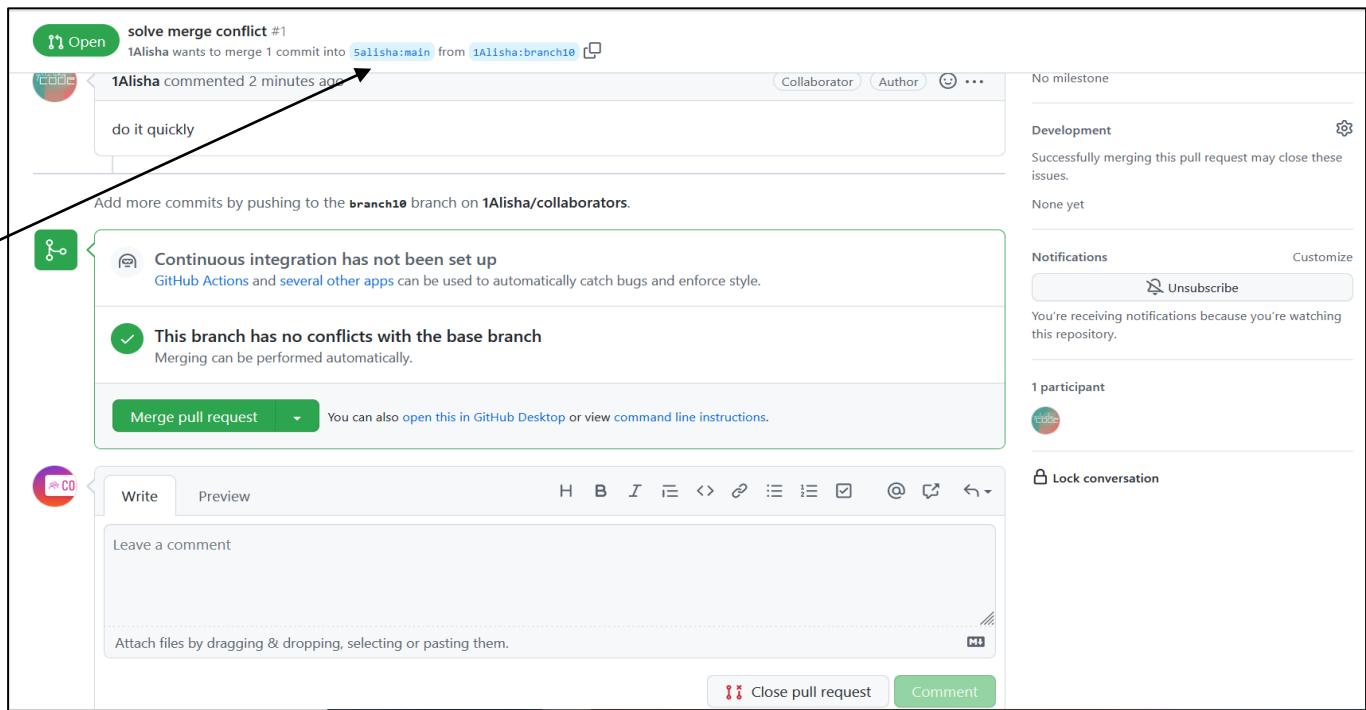
- ❖ Now, 5@alisha can keep the changes she want and delete other changes to resolve the merge conflict created due to collaboration.

```

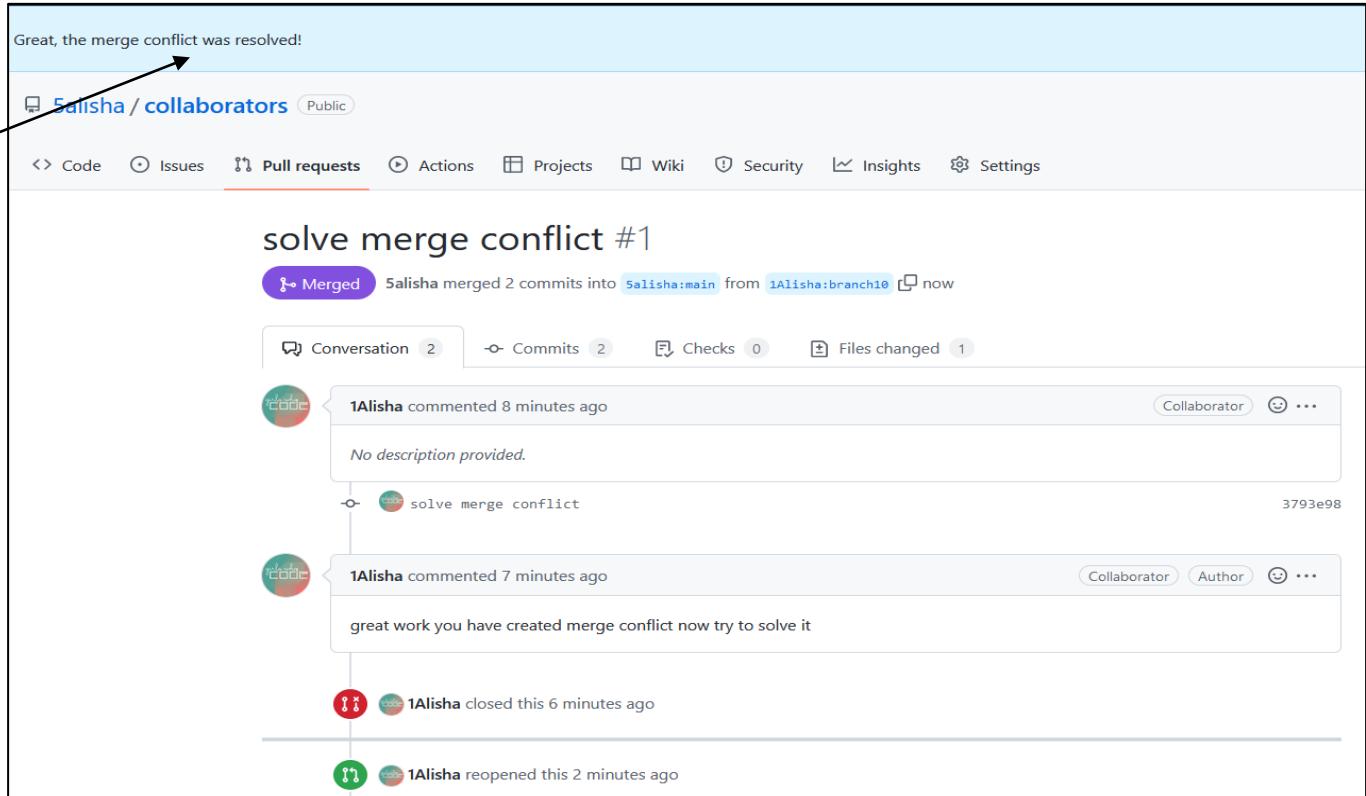
1 #include <iostream>
2 using namespace std;
3
4 int main(){
5     <<<<< branch10
6     cout<<"Solve merge conflict";
7     =====
8     cout<<"rm -rf .git is used to remove .git folder";
9     >>>> main
10    return 0;
11 }
12

```

- ❖ After performing the above step, merging can be done easily and pull request can be closed after merging.

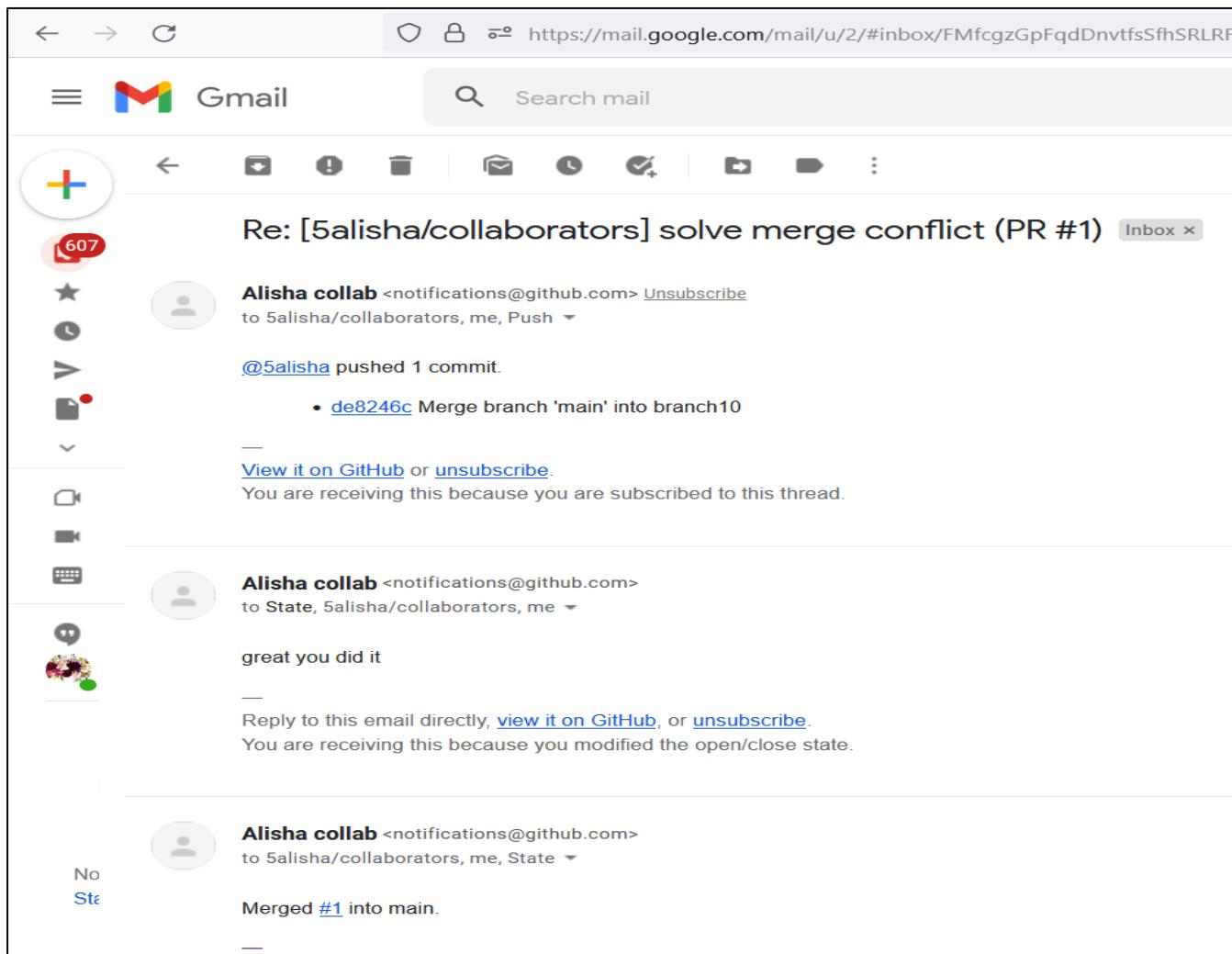


A screenshot of a GitHub pull request page titled "solve merge conflict #1". The pull request is from user "1Alisha" to branch "Salisha:main" from "1Alisha:branch10". A comment from "1Alisha" says "do it quickly". Below the comment, a note says "Add more commits by pushing to the branch10 branch on 1Alisha/collaborators.". A green button at the bottom right says "Merge pull request". On the right side, there are sections for "Development" (noting successful merge), "Notifications" (with an "Unsubscribe" button), and "1 participant" (with a profile picture). A large arrow points from the "do it quickly" comment towards the "Merge pull request" button.



A screenshot of a GitHub pull request page titled "solve merge conflict #1". The status is now "Merged" (indicated by a purple button). The message is "Salisha merged 2 commits into Salisha:main from 1Alisha:branch10 now". The conversation shows a comment from "1Alisha" saying "No description provided." followed by a link to "solve merge conflict". Another comment from "1Alisha" says "great work you have created merge conflict now try to solve it". The pull request was closed 6 minutes ago and reopened 2 minutes ago. A large arrow points from the "Great, the merge conflict was resolved!" message at the top left towards the "Merged" status.

- ❖ Notifications received on email during collaboration have been shown here



The screenshot shows a Gmail inbox with a red notification badge of 607 on the left sidebar. The main area displays three notifications from 'Alisha collab <notifications@github.com>'.

**Re: [5alisha/collaborators] solve merge conflict (PR #1) Inbox ×**

**Alisha collab <notifications@github.com>** [Unsubscribe](#)  
to 5alisha/collaborators, me, Push

@5alisha pushed 1 commit.  
• [de8246c](#) Merge branch 'main' into branch10

[View it on GitHub](#) or [unsubscribe](#).  
You are receiving this because you are subscribed to this thread.

**Alisha collab <notifications@github.com>**  
to State, 5alisha/collaborators, me

great you did it

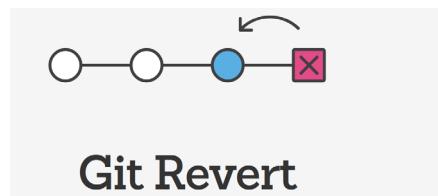
Reply to this email directly, [view it on GitHub](#), or [unsubscribe](#).  
You are receiving this because you modified the open/close state.

**Alisha collab <notifications@github.com>**  
to 5alisha/collaborators, me, State

Merged [#1](#) into main.

**Aim:** Revert and Reset**Description for Revert:**

- ❖ When you revert a git command, the changes from targeted commits are removed from local workspace. A new commit is also created to reflect the new state of your repository
- ❖ When you git revert a commit, only the changes associated with that commit are undone.



- ❖ For example, if a past commit added a file named index.html to the repo, a git revert on that commit will remove the index.html file from the repo. If a past commit added a new line of code to a Java file, a git revert on that commit will remove the added line.

**Steps to be followed to revert a git command:**

- ❖ Initialize the empty git repository, create a file student.txt and add the content .
- ❖ Here, in student.txt file changes like name , LastName, roll no and subject have been staged and committed.

```
MINGW64:/c/Users/shubh/Desktop/Revertt
shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Revertt (master)
$ git init
Initialized empty Git repository in c:/Users/shubh/Desktop/Revertt/.git/
shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Revertt (master)
$ touch student.txt

shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Revertt (master)
$ vi st*
shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Revertt (master)
$ cat st*
Alisha

shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Revertt (master)
$ git add --a
warning: LF will be replaced by CRLF in student.txt.
The file will have its original line endings in your working
directory

shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Revertt (master)
$ git commit -m"Name added"
[master (root-commit) 715fc0] Name added
 1 file changed, 1 insertion(+)
```

- ❖ Copy the checksum of last commit and run the command
  - ❖ `git revert d16e0d8`

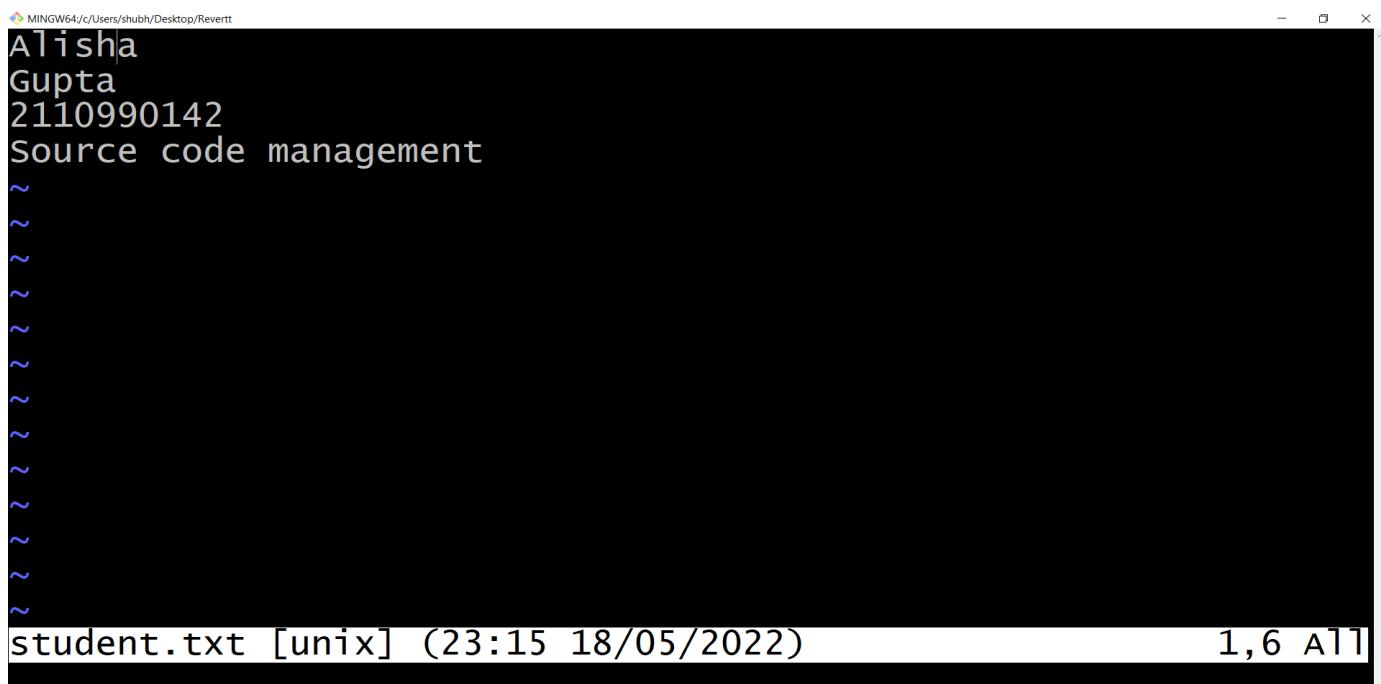
- ❖ After executing git revert command when we use git log –online command a new commit is also created to reflect the new state of repository .

```
shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Revertt (master)
$ git log --oneline
d16e0d8 (HEAD -> master) Revert "subject added"
8f09a9f subject added
a0e1c2b roll no
252c9ab LastName added
715fca0 Name added

shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Revertt (master)
$ cat st*
Alisha
Gupta
2110990142

shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Revertt (master)
$ git status
on branch master
nothing to commit, working tree clean
```

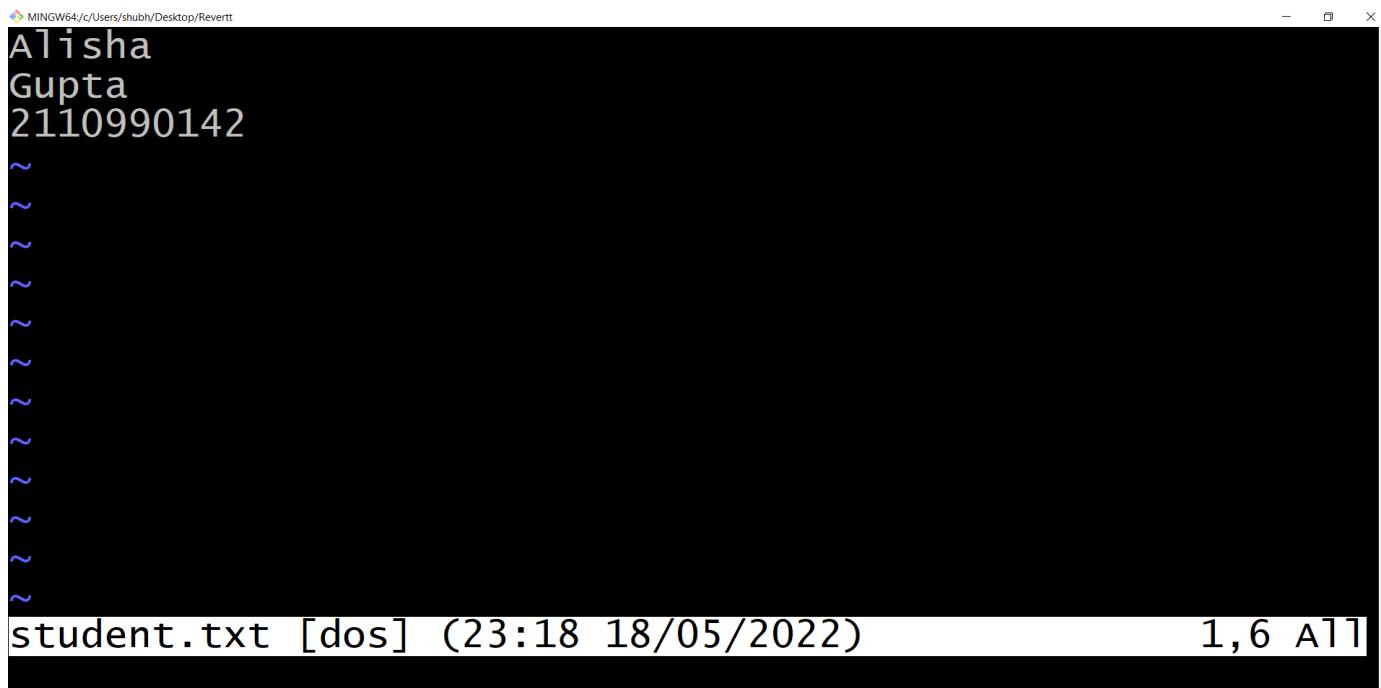
- ❖ Before executing revert command student.txt file has 4 commits.



MINGW64/c/Users/shubh/Desktop/Revertt

```
Alisha
Gupta
2110990142
Source code management
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
student.txt [unix] (23:15 18/05/2022) 1,6 All
```

- ❖ After executing the git revert command, last commit “subject added” has been removed.



MINGW64/c/Users/shubh/Desktop/Revertt

```
Alisha
Gupta
2110990142
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
student.txt [dos] (23:18 18/05/2022) 1,6 All
```

## Description for reset:

Practically, you can think of it as a "rollback"—it points your local environment back to a previous commit. By "local environment," we mean your local repository, staging area, and working directory. Git reset will delete/undo changes which are committed in local repository. It will undo changes in three ways:

### --soft

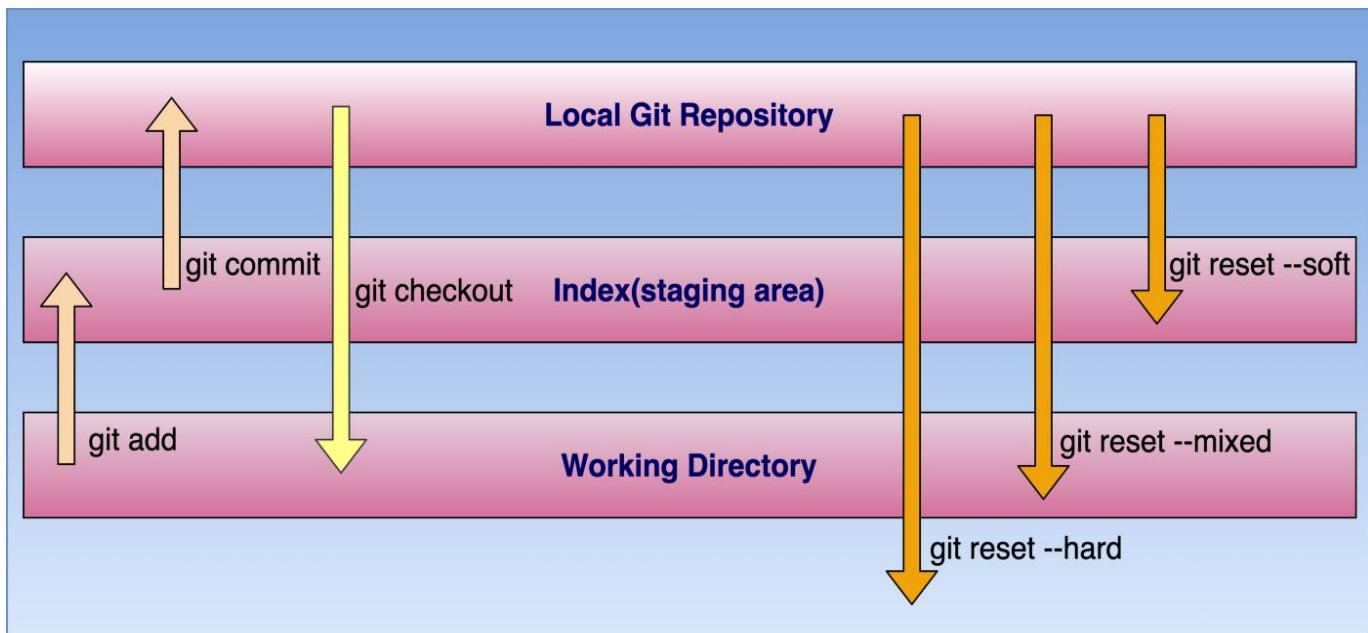
Does not touch the index file or the working tree at all (but resets the head to <commit>, just like all modes do). This leaves all your changed files "Changes to be committed", as git status would put it.

### --mixed

Resets the index but not the working tree (i.e., the changed files are preserved but not marked for commit) and reports what has not been updated. This is the default action.

### --hard

Resets the index and working tree. Any changes to tracked files in the working tree since <commit> are discarded. Any untracked files or directories in the way of writing any tracked files are simply deleted.



## git reset --soft Head~1

- **git reset** changes where the current branch is pointing to (**HEAD**).
- **HEAD** is a pointer or a reference to the last commit in the current branch.
- **HEAD~3** would mean behind 3 commits from **HEAD**.

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/Revertt (master)
$ git status
On branch master
nothing to commit, working tree clean

shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/Revertt (master)
$ cat st*
Alisha
Gupta
2110990142
a1ishagupta0142.be21@chitkara.edu.in
Monika sthi
SCM

shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/Revertt (master)
$ git log --oneline
b781be9 (HEAD -> master) 1st subject added
6ff4d67 mentor added
b854bb6 mail id added
d16e0d8 Revert "subject added"
8f09a9f subject added
a0e1c2b roll no
252c9ab LastName added
715fcfa0 Name added
```

```
MINGW64:/c/Users/shubh/Desktop/Revertt
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/Revertt (master)
$ git reset --soft Head~2

shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/Revertt (master)
$ git log --oneline
b854bb6 (HEAD -> master) mail id added
d16e0d8 Revert "subject added"
8f09a9f subject added
a0e1c2b roll no
252c9ab LastName added
715fcfa0 Name added

shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/Revertt (master)
$ cat st*
Alisha
Gupta
2110990142
a1ishagupta0142.be21@chitkara.edu.in
Monika sthi
SCM

shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/Revertt (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   student.txt
```

## git reset --mixed Head~1

Suppose we have three commits in the order -C1 - C2 – C3. Assume **HEAD** is pointing to **C3** commit and the index matches **C3** commit. When we execute **git reset --mixed HEAD~1** then **HEAD** points to **C2** commit,

- If we run **git commit** at this point, nothing will happen since the index matches **HEAD**. But the changes are still there in the working directory, but since they're not in the index, **git status** show them as unstaged.
- To commit them, we would use **git add** , and then we'll use **git commit -m “<message>”** to commit changes as we do.

**Note that --mixed is the default option.**

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/Revertt (master)
$ git log --oneline
e9b7259 (HEAD -> master) using reset soft
b854bb6 mail id added
d16e0d8 Revert "subject added"
8f09a9f subject added
a0e1c2b roll no
252c9ab LastName added
715fcfa0 Name added

shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/Revertt (master)
$ cat st*
Alisha
Gupta
2110990142
alishagupta0142.be21@chitkara.edu.in
Monika sthi
SCM

shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/Revertt (master)
$ git reset --mixed Head~2
Unstaged changes after reset:
M      student.txt
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/Revertt (master)
$ cat st*
Alisha
Gupta
2110990142
alishagupta0142.be21@chitkara.edu.in
Monika sthi
SCM

shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/Revertt (master)
$ git log --oneline
d16e0d8 (HEAD -> master) Revert "subject added"
8f09a9f subject added
a0e1c2b roll no
252c9ab LastName added
715fcfa0 Name added

shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/Revertt (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>" to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   student.txt

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

## git reset --hard Head~1

- Assuming **HEAD** is pointing to **C3** and the index matches **C3**
- **--hard** flag modifies HEAD, index, and working directory.
- If we're at C3 and when we run **git reset --hard HEAD~1**, then the changes committed to C3 , uncommitted changes that we have in the staging area, all the changes in the working directory will be removed and the working directory will match match C2 commit.

```
shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Revertt (master)
$ git status
On branch master
nothing to commit, working tree clean

shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Revertt (master)
$ git log --oneline
edfbc88 (HEAD -> master) using reset mixe
d16e0d8 Revert "subject added"
8f09a9f subject added
a0e1c2b roll no
252c9ab LastName added
715fcfa0 Name added

shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Revertt (master)
$ cat st*
Alisha
Gupta
2110990142
alishagupta0142.be21@chitkara.edu.in
Monika sthi
SCM

shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Revertt (master)
$ git reset --hard Head~2
HEAD is now at 8f09a9f subject added
```

```
MINGW64:/c/Users/shubh/Desktop/Revertt
shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Revertt (master)
$ cat st*
Alisha
Gupta
2110990142
Source code management

shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Revertt (master)
$ git status
On branch master
nothing to commit, working tree clean

shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Revertt (master)
$ git reset edfbc88 --hard
HEAD is now at edfbc88 using reset mixe

shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Revertt (master)
$ git log --oneline
edfbc88 (HEAD -> master) using reset mixe
d16e0d8 Revert "subject added"
8f09a9f subject added
a0e1c2b roll no
252c9ab LastName added
715fcfa0 Name added
```

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/Revertt (master)
$ cat st*
Alisha
Gupta
2110990142
alishagupta0142.be21@chitkara.edu.in
Monika sthi
SCM
```

- Using **git reset edfbc88 --hard** , deleted commits can be restored.



Checksum of last commit

A Project report

On

**“Hosting a website”**

with

**Source Code Management**

(CS181)

Submitted by

Team Member 1 Name Roll No. 2110990142

Team Member 2 Name Roll No. 2110990143

Team Member 3 Name Roll No. 2110990144

Team Member 3 Name Roll No. 2110990145



**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>Beta</b>	Group No	<b>G-2</b>
Course Coordinator	<b>Dr. Neeraj Singla</b>		
Faculty Name	<b>Dr. Deepak Thakur</b>		

## **Submission**

**Name: Alisha Gupta**

**Signature:**

**Date: May23, 2022**

## Table of content

S. No.	Title	Page No.
1	Version control with Git	61-65
2	Problem Statement	66
3	Objective	67
5	Concepts and commands	68-78
6	Workflow and Discussion	79-88
7	Reference	89

We all have maintained several versions, whether it's **adding documentation to the code, rearranging the methods, adding or removing some classes or resources**, Most of the time these versions would have been maintained by giving a suitable name to the file as shown below

1. abc\_1.java
2. abc\_14Mar.java

Naturally, we all must've wished for an organized way of storing these versions to see progressive changes made to the code. Some of the problems that one can face in code management are mentioned below

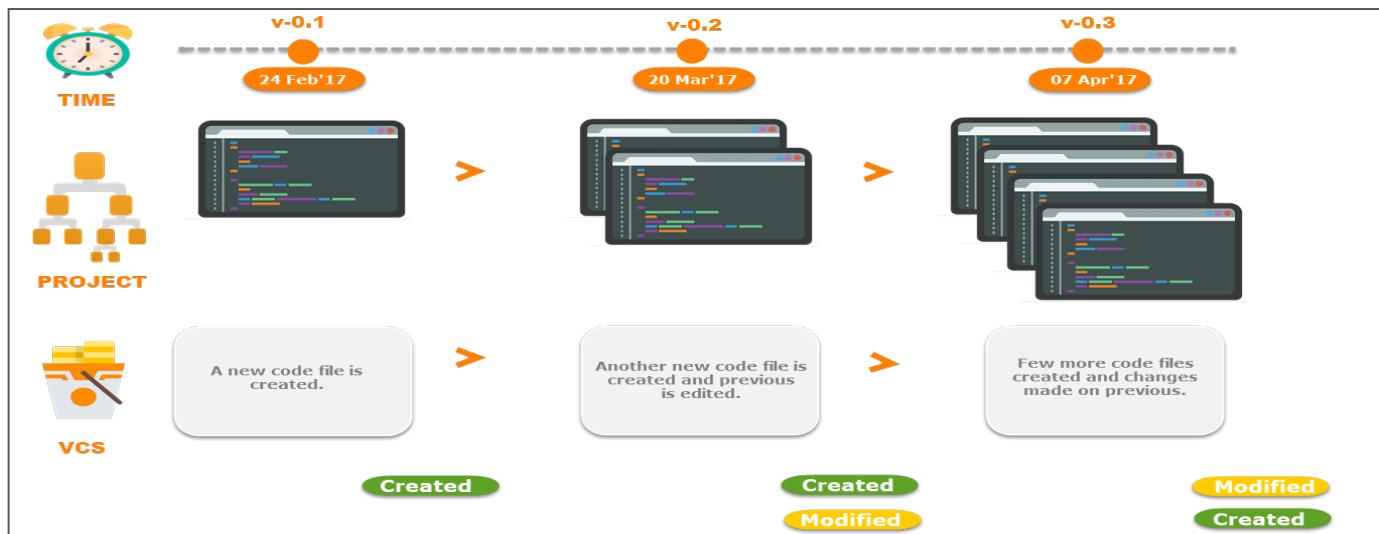
## Challenges in Code Management

3. Maintaining multiple copies of the same code files based on changes done on different dates
4. Trace the file's history to find the version of the code which introduced the bug
5. Maintain back-up of your files

## What is Version Control System?

We can think of a **Version Control System** as a kind of **database**. Version Control Systems (VCS) are tools that help teams manage and track changes in code over time. It lets us save a snapshot of the complete project at any point in time. Every change made to the project files is tracked, along with

- Who made the change
- Why the change was made (references to problems fixed or features added in the change)



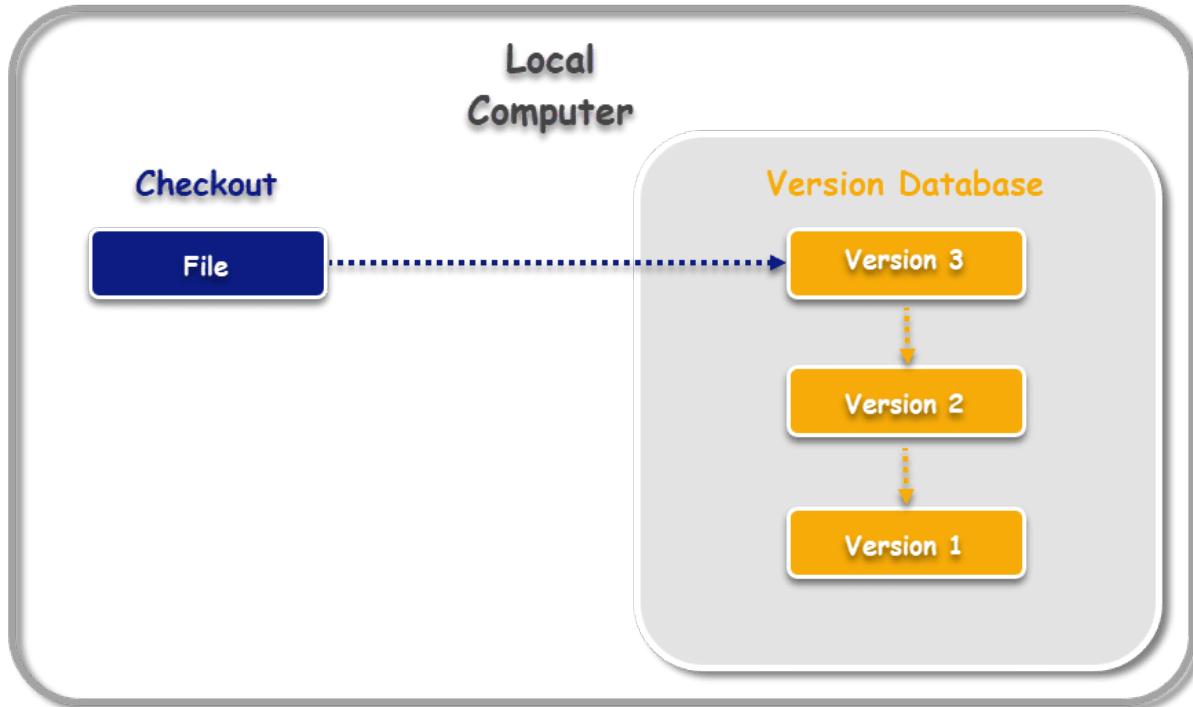
**Note:** Snapshot is the entire state of the project at a given point of time.

In general, there are three types of **Version Control Systems**

- Local Version Control System
- Central Version Control System
- Distributed Version Control Systems

### Local VCS

- ❖ Usually, **Local VCS** would store the changes to a file in a **Local Database**.
- ❖ **VCS**, in general, have got advance these days.
- ❖ Instead of storing the new versions of the complete file, VCS now store only the difference between two versions. This saves memory cost.



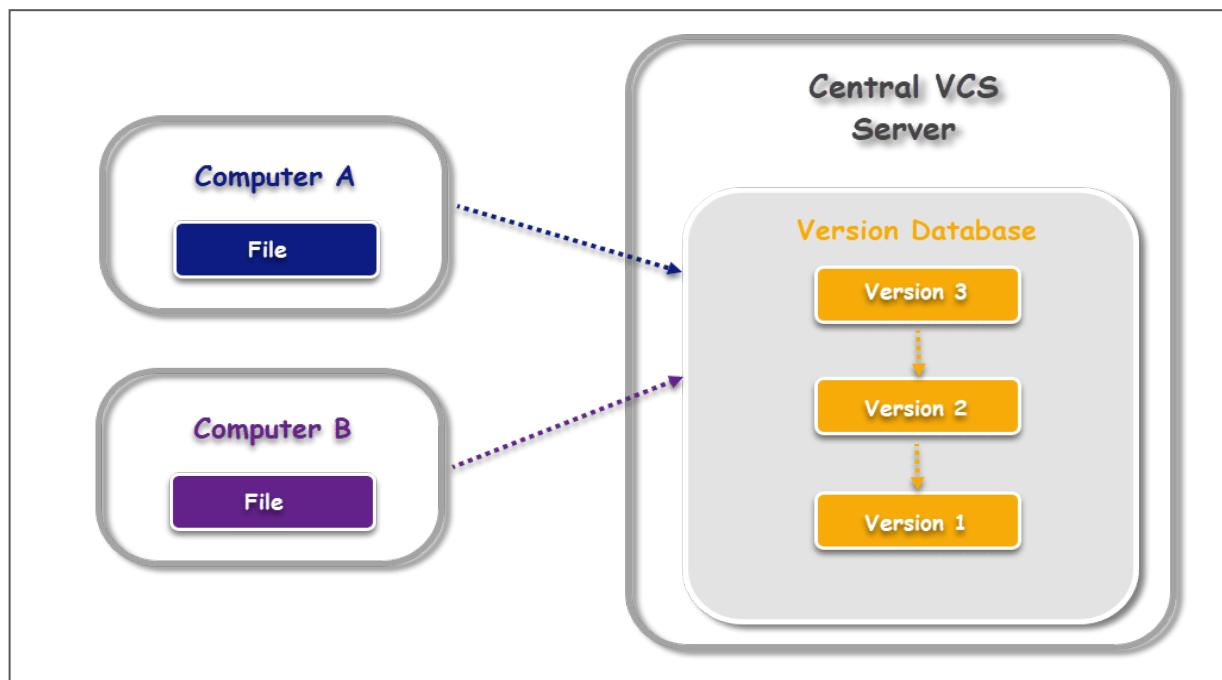
As shown in the image above, the local database stores **Version 1**, **Version 2**, and so on. These versions are groups of related files that have undergone changes since their last edit.

**Note:** Local VCS had a major drawback of being useful to only a single developer at a time. Thus, several developers working on different systems in a project could not collaborate effectively by using local VCS.

## Central VCS

The developers needed to collaborate with other developers on other systems. The localized version control system failed in this case. To deal with this problem, Centralized Version Control Systems were developed.

- Everyone on the system has information about the work what others are doing on the project.
- Administrators have control over other developers.
- It is easier to deal with a centralized version control system than a localized version control system.
- A local version control system facilitates with a server software component which stores and manages the different versions of the files.



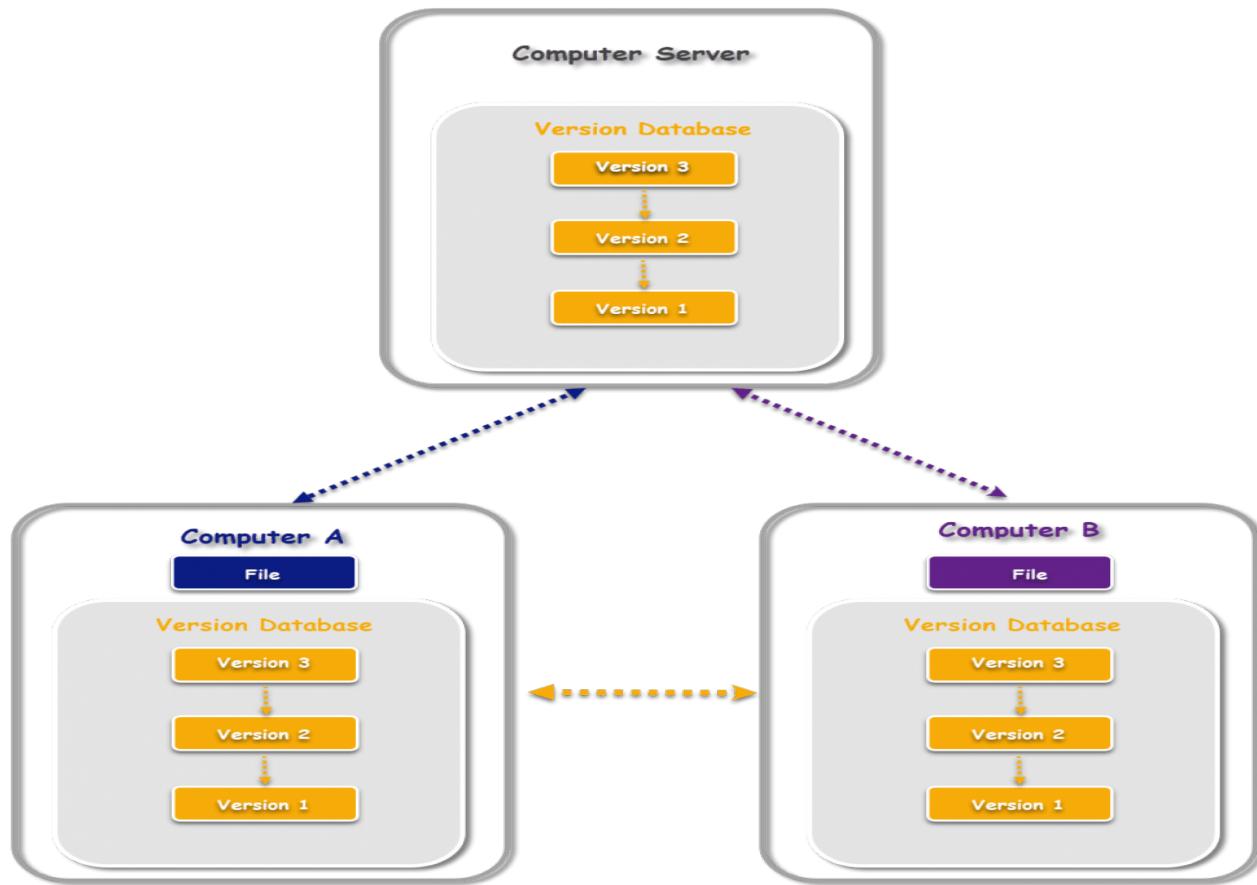
These systems have a single server that contains the versioned files, and some clients to check out files from a central place.

### Few drawbacks of Central VCS

- ❖ Though central VCS helped developers and teams collaborate, there's a risk of an outage of the central server.
- ❖ Teams would neither be able to collaborate nor record changes of whatever they're working on due to such outages.
- ❖ Such servers are also prone to hacking. In such a case, it'd be pain-staking for teams to recover their work. Therefore it's not advisable to store all changes to code in a single place.

## Distributed Version Control Systems (D-VCS)

- Distributed VCS introduced significant improvement over the risks posed by Central VCS.
- In Distributed VCS, every local machine would have a copy of the entire code-base along with its history. Thus Distributed VCS moves from the client-server approach of Central VCS to peer-to-peer approach for version control a



- ❖ Centralized Version Control System uses a central server to store all the database and team collaboration, but due the failure of the central server, developers do not prefer it
- ❖ In a Distributed Version Control System, the user has a local copy of a repository. So, the clients don't just check out the latest snapshot of the files even they can fully mirror the repository.
- ❖ Developers can record changes in their local machine while working offline and upload the new history of versions to the central server for others to view and use after connecting to the network.

**NOTE:** Git is a Distributed Version Control system.

## 2. Problem statement

We have to edit and send our project to other team members which is very lengthy process:

So we want an alternate solution for this to make changes conveniently and accurately. Also we want that if we make the changes it should reflect to all the other participants and they should be able to revert and edit.

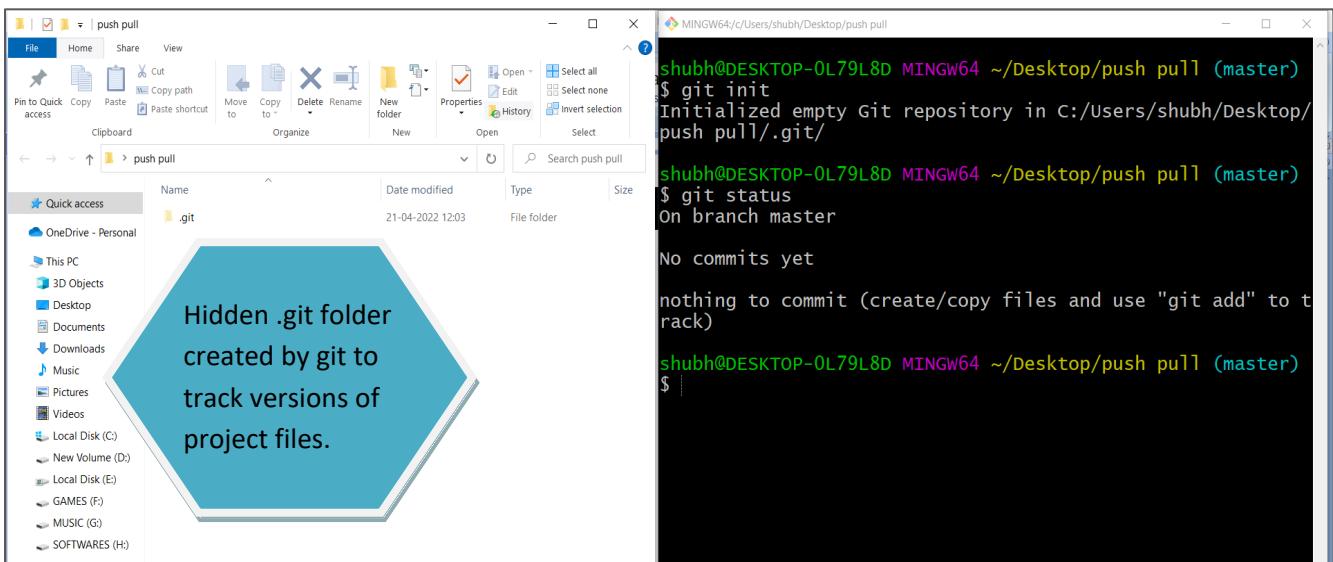
❖ Following points are the Objectives of Git:

- To track changes in the code using Git and do collaborations.
- Git keeps track of changes to files and allows multiple users to coordinate updates to those files.
- To make our code more public for everyone to use.
- To do distributed development as 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. Concepts and Commands

### ❖ git init

- Initializes empty git repository



### ❖ git status

- Displays the state of the working directory and the staging area.

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/push pull (master)
$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/Project 2.0 (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

### ❖ git add --a

- This command updates the current content of the working tree to the staging area.

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/Revertt (master)
$ git add --a
```

### ❖ **git config --global user.name "FIRST\_NAME LAST\_NAME"**

➤ Set username

```
shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/f1 (master)
$ git config --global user.name "1Alisha"

shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/f1 (master)
$ git config --global user.email "alishagupta314@gmail.com"

shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/f1 (master)
$ git config user.name
1Alisha
```

### ❖ **git config --global user.email "MY\_NAME@example.com"**

➤ Set your email address

```
shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/f1 (master)
$ git config user.email
alishagupta314@gmail.com
```

### ❖ **pwd**

➤ Shows the present working directory

```
MINGW64:/c/Users/shubh/Desktop/Revertt
shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Revertt (master)
$ pwd
/c/Users/shubh/Desktop/Revertt
```

### ❖ **ls**

➤ lists the current directory contents

```
shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Revertt (master)
$ ls
student.txt
```

### ❖ **git commit -m “message for the commit”**

➤ commit the staged snapshot

```
MINGW64:/c/Users/shubh/Desktop/Project 2.0
shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Project 2.0 (master)
$ git commit -m"Logo image added"
[master e497cd7] logo image added
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 sss.jpeg
```

## git log

- commit the staged snapshot

```
shubh@DESKTOP-OL79L8D MINGW64 /e/Evening_class_scm (super)
$ git log
commit 3066ad7722b9517a88c70f600e80c8599a5b8cc8 (HEAD -> super)
Author: 1Alisha <alishagupta314@gmail.com>
Date:   Wed Apr 13 06:19:04 2022 +0530
```

## ❖ git log --oneline

- commit the staged snapshot

```
shubh@DESKTOP-OL79L8D MINGW64 /e/Evening_class_scm (super)
$ git log --oneline
3066ad7 (HEAD -> super) men section on super branch
b4629b3 toy section nd women section modified
f0cbabe (master) women and toy section adeded
6341bcd men section added
54ae60d home page on master branch
50578d8 new content added on master branch
ea085ea file name changeeg to home.txt
3c65142 file name changed to contactus
6d873a3 new text files added
```

## ❖ git log --oneline graph

- commit the staged snapshot

```
shubh@DESKTOP-OL79L8D MINGW64 /e/Evening_class_scm (master)
$ git log --oneline --graph
* 8a5718e (HEAD -> master) Merge branch 'super'
| \
| * 29f2150 (super) women section on super branch ....
* | fa215fd home and contactus on master branch
| /
* 3066ad7 men section on super branch
* b4629b3 toy section nd women section modified
* f0cbabe women and toy section adeded
* 6341bcd men section added
* 54ae60d home page on master branch
* 50578d8 new content added on master branch
* ea085ea file name changeeg to home.txt
* 3c65142 file name changed to contactus
* 6d873a3 new text files added
```

## ❖ rm -rf .git

- Deletes the .git folder.



### ❖ git branch “branch\_name”

- Creates a new branch

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/Final (master)
$ git branch branch1
```

### ❖ Git checkout branch\_name

- Swapping between different branches

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/Final (master)
$ git checkout branch1
Switched to branch 'branch1'
```

### ❖ git branch

- Shows all branches present in that directory

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/Final (branch1)
$ git branch
* branch1
  branch3
  master
```

### ❖ git merge branch\_name

- Merge two branches

```
shubh@DESKTOP-OL79L8D MINGW64 /e/Evening_class_scm (master)
$ git merge tree
Auto-merging home.txt
CONFLICT (content): Merge conflict in home.txt
Automatic merge failed; fix conflicts and then commit the result.
```

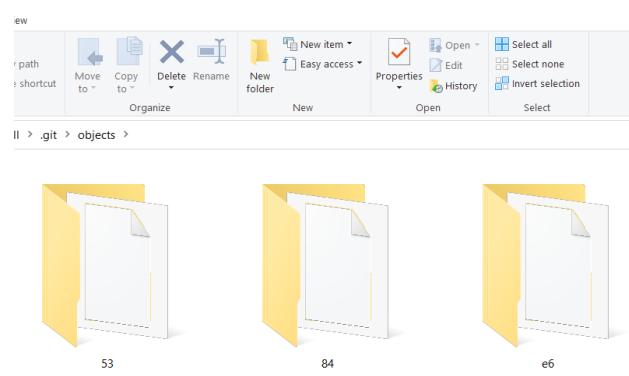
### ❖ git cat-file checksum -p

- Show the pointer to TREE object *git cat-file commit*

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/push pull (master)
$ git cat-file 53539c12e4db7d001113efdff3fd74e4ec30b905 -p
tree 84feb77cc36c1f8da2b093b0665aa3b9cdf87c54
author 1Alisha <alishagupta314@gmail.com> 1650526404 +0530
committer 1Alisha <alishagupta314@gmail.com> 1650526404 +0530

aboutus and home created

shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/push pull (master)
$ git cat-file 84feb77cc36c1f8da2b093b0665aa3b9cdf87c54 -p
100644 blob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391 Aboutus.txt
100644 blob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391 home.txt
```



## ❖ mv file1.txt file2.txt

- Renames the file using this command we have to stage the file again

```
shubh@DESKTOP-0L79L8D MINGW64 /e/Evening_class_scm (master)
$ mv aboutus.txt.txt contactus.txt

shubh@DESKTOP-0L79L8D MINGW64 /e/Evening_class_scm (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    deleted:    aboutus.txt.txt

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

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

## ❖ git mv file1.txt file2.txt

- Renames the file and shows the status as modified

```
shubh@DESKTOP-0L79L8D MINGW64 /e/Evening_class_scm (master)
$ git mv home.txt.txt home.txt

shubh@DESKTOP-0L79L8D MINGW64 /e/Evening_class_scm (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    renamed:   home.txt.txt -> home.txt

shubh@DESKTOP-0L79L8D MINGW64 /e/Evening_class_scm (master)
$ git commit -m 'file name changeg to home.txt'
[master ea085ea] file name changeg to home.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
 rename home.txt.txt => home.txt (100%)
```

## ❖ git restore --staged file1.txt

- Unstage the file

```
shubh@DESKTOP-0L79L8D MINGW64 /e/Evening_class_scm (master)
$ git restore --staged home.txt

shubh@DESKTOP-0L79L8D MINGW64 /e/Evening_class_scm (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   home.txt
```

### ❖ **git branch -d branch\_name**

➤ Deletes the branch after merging

### ❖ **git branch -D branch\_name**

➤ deletes the branch without merging

```
shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Final (master)
$ git branch -d branch1
error: The branch 'branch1' is not fully merged.
If you are sure you want to delete it, run 'git branch -D branch1'

shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Final (master)
$ git merge branch1
Updating fa6e7bf..a06925a
Fast-forward
 oops.cpp | 2 ++
 1 file changed, 1 insertion(+), 1 deletion(-)

shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Final (master)
$ git branch -d branch1
Deleted branch branch1 (was a06925a).

shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Final (master)
$ git status
On branch master
nothing to commit, working tree clean

shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Final (master)
$ git branch -D branch2
Deleted branch branch2 (was fa6e7bf).
```

```
shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Final (branch1)
$ git branch -d branch1
error: Cannot delete branch 'branch1' checked out at 'c
:/Users/shubh/Desktop/Final'

shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Final (branch1)
$ git checkout master
Switched to branch 'master'

shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Final (master)
$ git branch -d branch1
error: The branch 'branch1' is not fully merged.
If you are sure you want to delete it, run 'git branch -D branch1'.

shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Final (master)
$ git merge branch1
Updating fa6e7bf..a06925a
Fast-forward
 oops.cpp | 2 ++
 1 file changed, 1 insertion(+), 1 deletion(-)
```

### ❖ **git remote add origin "url"**

➤ Create a new, empty Git repository on your remote server

```
shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Project 2.0 (master)
$ git remote add origin https://github.com/1Alisha/Project2.0.git
```

### ❖ **git push -u origin master**

➤ Uploads content from a local repository to a remote repository.

```
shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Project 2.0 (master)
$ git push -u origin master
Enumerating objects: 13, done.
Counting objects: 100% (13/13), done.
Delta compression using up to 4 threads
Compressing objects: 100% (13/13), done.
Writing objects: 100% (13/13), 68.03 KiB | 7.56 MiB/s, done.
Total 13 (delta 4), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (4/4), done.
To https://github.com/1Alisha/Project2.0.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
```

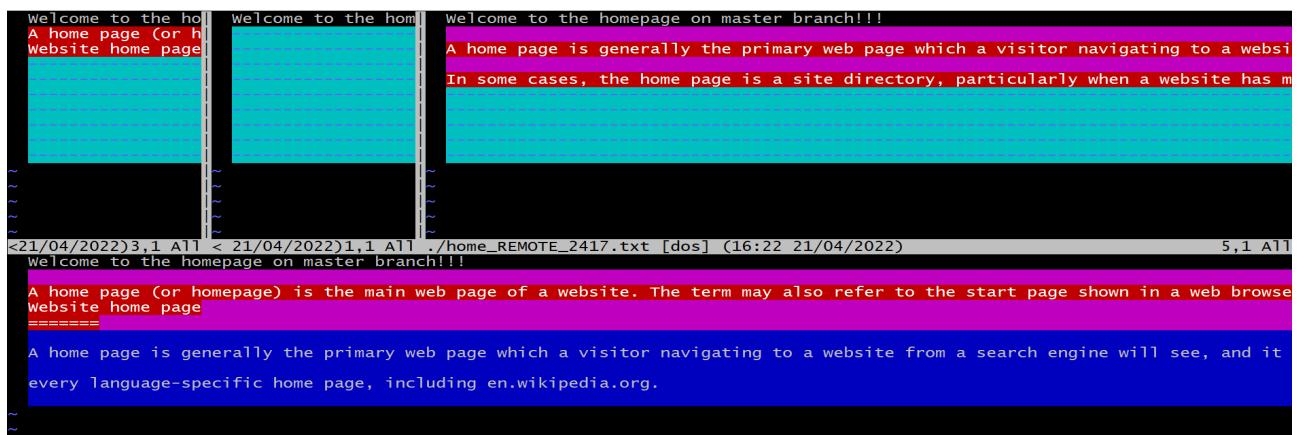
## ❖ git branch pull “url”

- To *pull* the data from the edited *branch*.

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/Project 2.0 (master)
$ git pull https://github.com/1Alisha/Project2.0.git
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 5 (delta 2), reused 4 (delta 2), pack-reused 0
Unpacking objects: 100% (5/5), 205.54 KiB | 880.00 KiB/s, done.
From https://github.com/1Alisha/Project2.0
 * branch HEAD      -> FETCH_HEAD
Updating 9b8352f..7492b19
Fast-forward
 #Kurtis.html      |  2 +-+
 Screenshot (17).png | Bin 0 -> 209423 bytes
 2 files changed, 1 insertion(+), 1 deletion(-)
 create mode 100644 Screenshot (17).png
```

## ❖ git mergetool

- To remove conflicts between two branches



## ❖ git clone “url”

- deletes the branch without merging

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop (master)
$ git clone https://github.com/1Alisha/earthLab.git
Cloning into 'earthLab'...
remote: Enumerating objects: 72, done.
remote: Counting objects: 100% (25/25), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 72 (delta 19), reused 19 (delta 19), pack-reused 47
Receiving objects: 100% (72/72), 9.98 KiB | 123.00 KiB/s, done.
Resolving deltas: 100% (35/35), done.
```

### ❖ cd

- To change this current working directory

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop (master)
$ cd earthLab

shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/earthLab (main)
```

### ❖ touch file.txt

- To create a file

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/earthLab (main)
$ touch task2.txt
```

### ❖ vi file.txt

- to either create a new file, or to edit an existing one

```
MINGW64:c/Users/shubh/Desktop/Revertt
Alisha
Gupta
2110990142
Source code management
~
~
```

### ❖ cat file.txt

```
#include <iostream.h>
using namespace std;

int main(){
cout<<"hello from b2 branch"

return 0,
```

- To view the content of file

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/Revertt (master)
$ cat student.txt
Alisha
Gupta
2110990142
alishagupta0142.be21@chitkara.edu.in
```

### ❖ **git remote add upstream “url”**

- Add the original repository as remote repository called “upstream”

```
shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/earthLab (main)
$ git remote add upstream "https://github.com/1Alisha/earthLab.git"
```

### ❖ **git fetch upstream**

- Fetch all the changes from the upstream repository

```
shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/earthLab (main)
$ git fetch upstream
From https://github.com/1Alisha/earthLab
 * [new branch]      main            -> upstream/main
```

### ❖ **git checkout master**

- Switch to the master branch of your fork

```
shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/earthLab (main)
$ git checkout main
Already on 'main'
Your branch is up to date with 'origin/main'.
```

### ❖ **git revert checksum**

- The changes from targeted commits are removed from your local workspace.

```
Revert "subject added"
This reverts commit 8f09a9f7a46ea15f32071e7ccbf00ed59e6a3ba1.
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
# Changes to be committed:
#       modified:   student.txt
#
```

### ❖ **.git reset --soft Head~1**

```
shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Reverttt (master)
$ git reset --soft Head~2
```

### ❖ **.git reset --mixed Head~1**

```
shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Reverttt (master)
$ git reset --mixed Head~2
Unstaged changes after reset:
M          student.txt
```

## ❖ git reset --hard Head~1

```
shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Revertt (master)
$ git reset --hard Head~2
HEAD is now at 8f09a9f subject added
```

## ❖ git reset checksum --hard

```
shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Revertt (master)
$ git reset edfbcc88 --hard
HEAD is now at edfbcc88 using reset mixe
```

## ❖ mkdir

- To make a new directory

```
shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Final (master)
$ mkdir scm

shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Final (master)
$ ls
oops.cpp    scm/
```

## ❖ git add .

- This adds the untracked file in staging area

```
shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Final (master)
$ git add .
```

## ❖ :wq + enter

- To exit any command

```
~
oops.cpp [dos] (17:32 20/05/2022) 7,14-21 All
:wq
```

## ❖ git remote -v

- This shows which remote is associated with which URL

```
shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Project 2.0 (master)
$ git remote -v
origin https://github.com/1Alisha/Project2.0.git (fetch)
origin https://github.com/1Alisha/Project2.0.git (push)
```

## ❖ git branch -r

- This shows which remote is linked with which branch

```
shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Project 2.0 (master)
$ git branch -r
origin/master
```

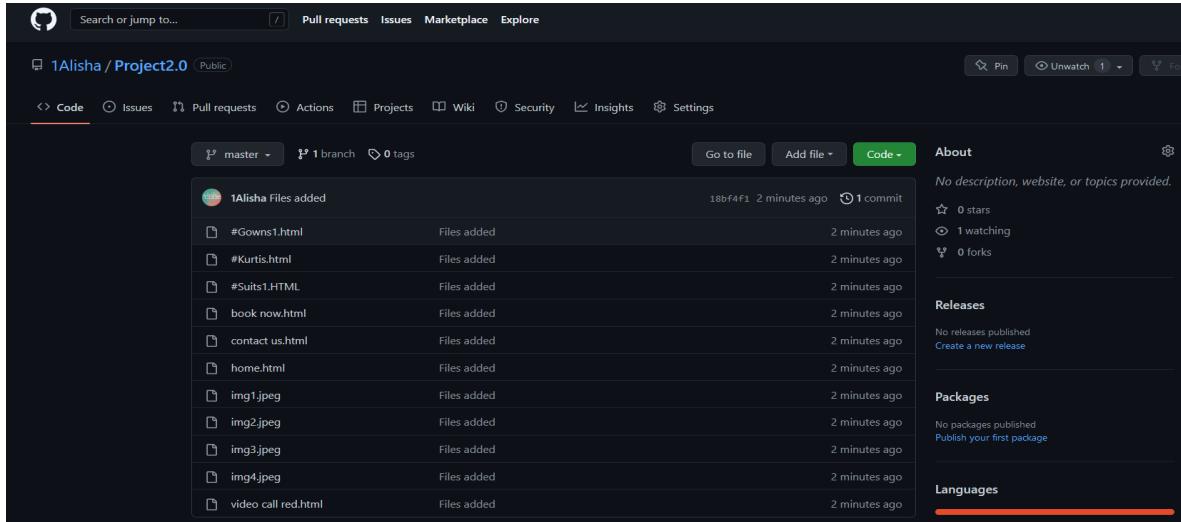
❖ **git branch -a**

- Shows the master branch of local repository

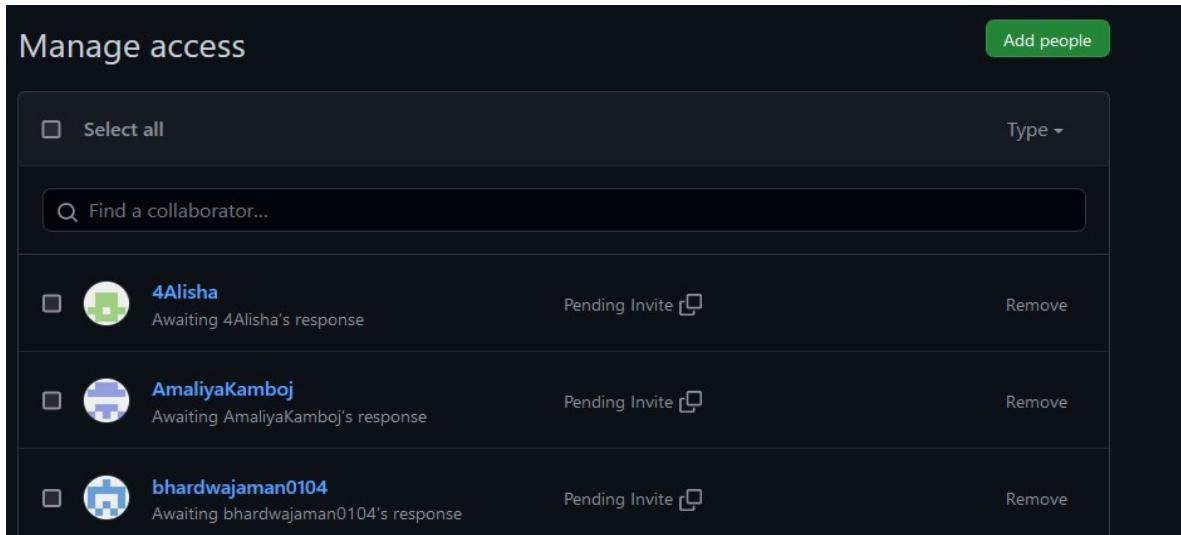
```
shubh@DESKTOP-0L79L8D MINGW64 ~/Desktop/Project 2.0 (master)
$ git branch -a
* master
  remotes/origin/master
```

## 5. Workflow and Discussions

- ❖ Firstly make a 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 to create a new repository



- Add your team members as collaborators



- After accepting the invitation, all members are ready to contribute to the project

## My snapshot's of collaboration ->

MINGW64:/c/Users/shubh/Desktop/Project 2.0

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/Project 2.0 (master)
$ git init
Initialized empty Git repository in C:/Users/shubh/Desktop/Project 2.0/.git/
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/Project 2.0 (master)
$ git add --a
g
```

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/Project 2.0 (master)
```

```
$ git commit -m"Files added"
[master (root-commit) 18bf4f1] Files added
11 files changed, 2349 insertions(+)
create mode 100644 #Gowns1.html
create mode 100644 #Kurtis.html
create mode 100644 #Suits1.HTML
create mode 100644 book now.html
create mode 100644 contact us.html
create mode 100644 home.html
create mode 100644 img1.jpeg
create mode 100644 img2.jpeg
create mode 100644 img3.jpeg
create mode 100644 img4.jpeg
create mode 100644 video call red.html
```

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/Project 2.0 (master)
```

```
$ git remote add origin https://github.com/1Alisha/Project2.0.git
```

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/Project 2.0 (master)
```

```
$ git push -u origin master
Enumerating objects: 13, done.
Counting objects: 100% (13/13), done.
Delta compression using up to 4 threads
Compressing objects: 100% (13/13), done.
Writing objects: 100% (13/13), 68.03 KiB | 7.56 MiB/s, done.
Total 13 (delta 4), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (4/4), done.
To https://github.com/1Alisha/Project2.0.git
 * [new branch] master -> master
branch 'master' set up to track 'origin/master'.
```

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/Project 2.0 (master)
```

```
$ git add --a
```

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/Project 2.0 (master)
```

```
$ git commit -m"css added"
[master 9b8352f] css added
2 files changed, 241 insertions(+)
create mode 100644 site.css
create mode 100644 styles.css
```

```
shubh@DESKTOP-OL79L8D MINGW64 ~/Desktop/Project 2.0 (master)
```

```
$ git push -u origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 1.74 KiB | 1.74 MiB/s, done.
Total 4 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/1Alisha/Project2.0.git
  18bf4f1..9b8352f master -> master
branch 'master' set up to track 'origin/master'.
```

Search or jump to... Pull requests Issues Marketplace Explore

1Alisha / Project2.0 Public

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

master 1 branch 0 tags Go to file Add file Code

1Alisha Files added 18bf4f1 2 minutes ago 1 commit

📁 #Gowns1.html	Files added	2 minutes ago	
📁 #Kurtis.html	Files added	2 minutes ago	
📁 #Suits1.HTML	Files added	2 minutes ago	
📁 book now.html	Files added	2 minutes ago	
📁 contact us.html	Files added	2 minutes ago	
📁 home.html	Files added	2 minutes ago	
📁 img1.jpeg	Files added	2 minutes ago	
📁 img2.jpeg	Files added	2 minutes ago	
📁 img3.jpeg	Files added	2 minutes ago	
📁 img4.jpeg	Files added	2 minutes ago	
📁 video call red.html	Files added	2 minutes ago	

About No description, website, or topics provided.

0 stars 1 watching 0 forks

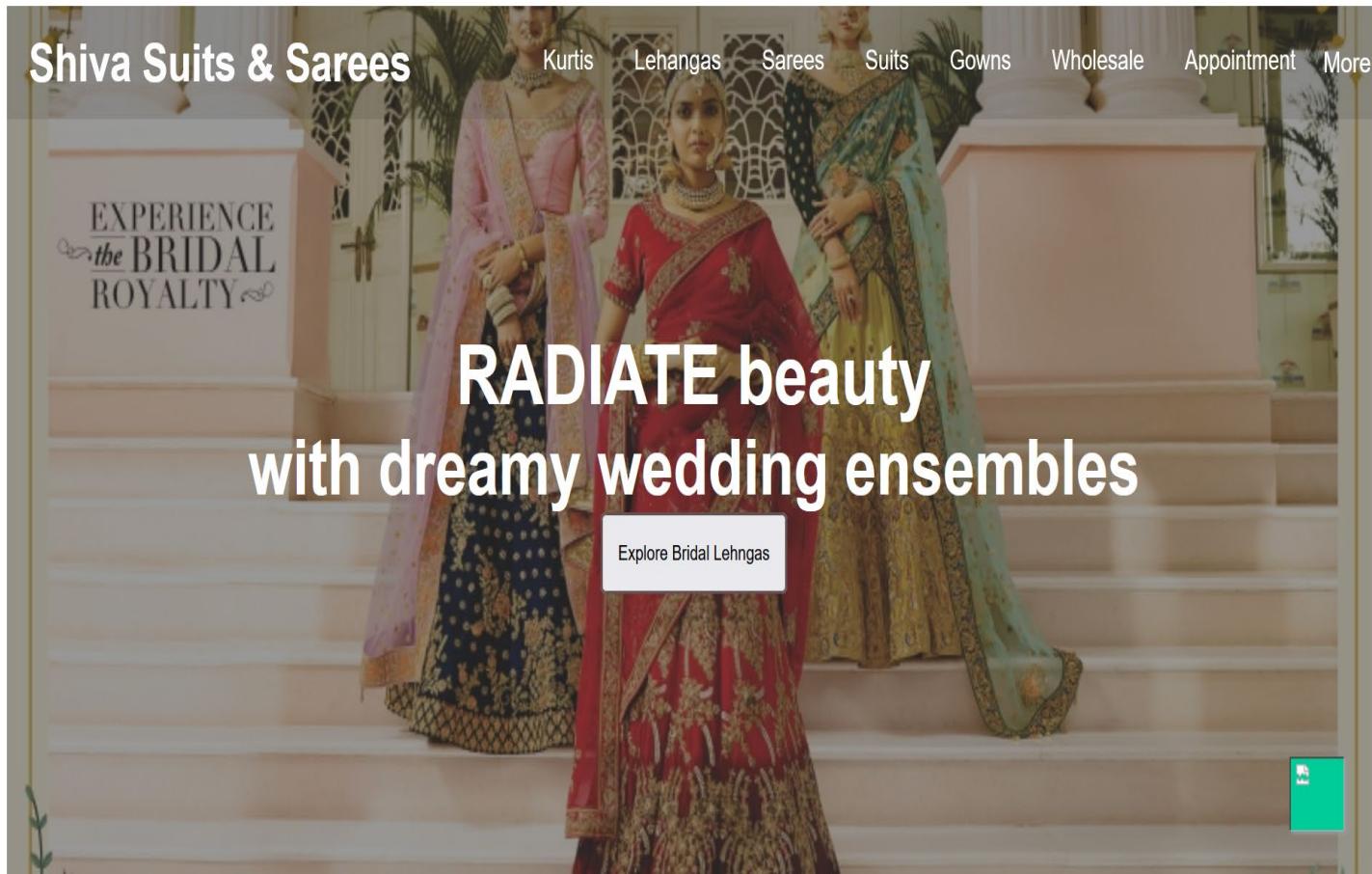
Releases No releases published Create a new release

Packages No packages published Publish your first package

Languages

This is how my uploaded Web pages looks like:

### Home page of the website



## Gown's page



Stunning Festive  
₹ 45,000



Trendy Attire  
₹ 42,000



Radiate Joy  
₹ 50,000



Stunning  
₹ 35,000



Beautiful  
₹ 40,000



Festivities  
₹ 42,000



Enchanting Style  
₹ 45,000



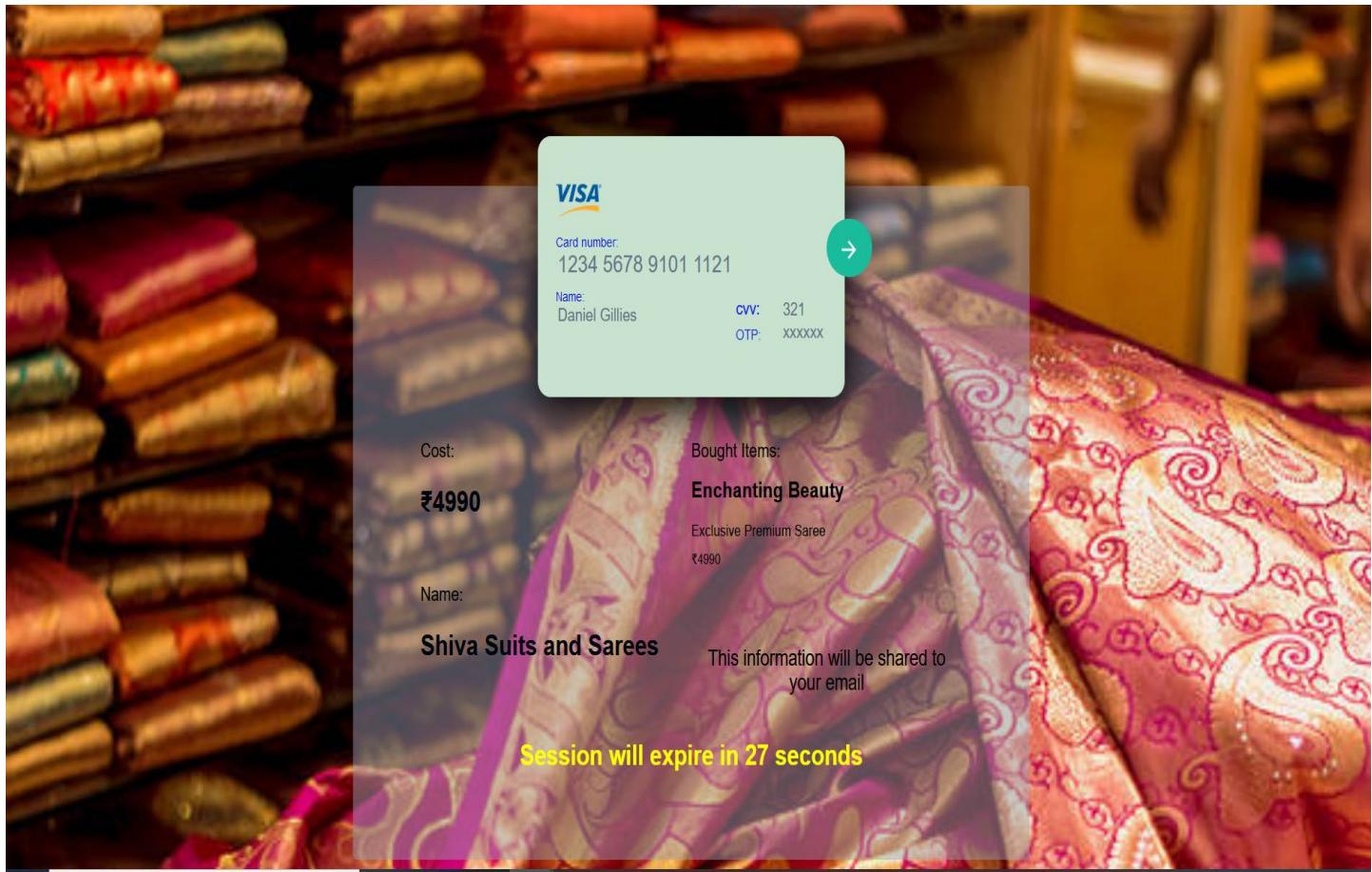
Enquire now

Exuberant Choice  
₹ 47,599



Blooming Bright  
₹ 40,000

## Payment gateway for the website



## Footer

### SUBSCRIBE TO OUR NEWSLETTER

Subscribe to our newsletter to receive exclusive offers and the latest news on our products and services



Shiva Suits and Sarees is a definition of the elegant traditional wear. Be it a Banaras weave saree, a designer Bridal Lehenga or the

chic/gown, every outfit is designed and crafted to perfection.

Shiva Suits and Sarees is a world of Traditional and semi traditional outfits, housed under one roof.

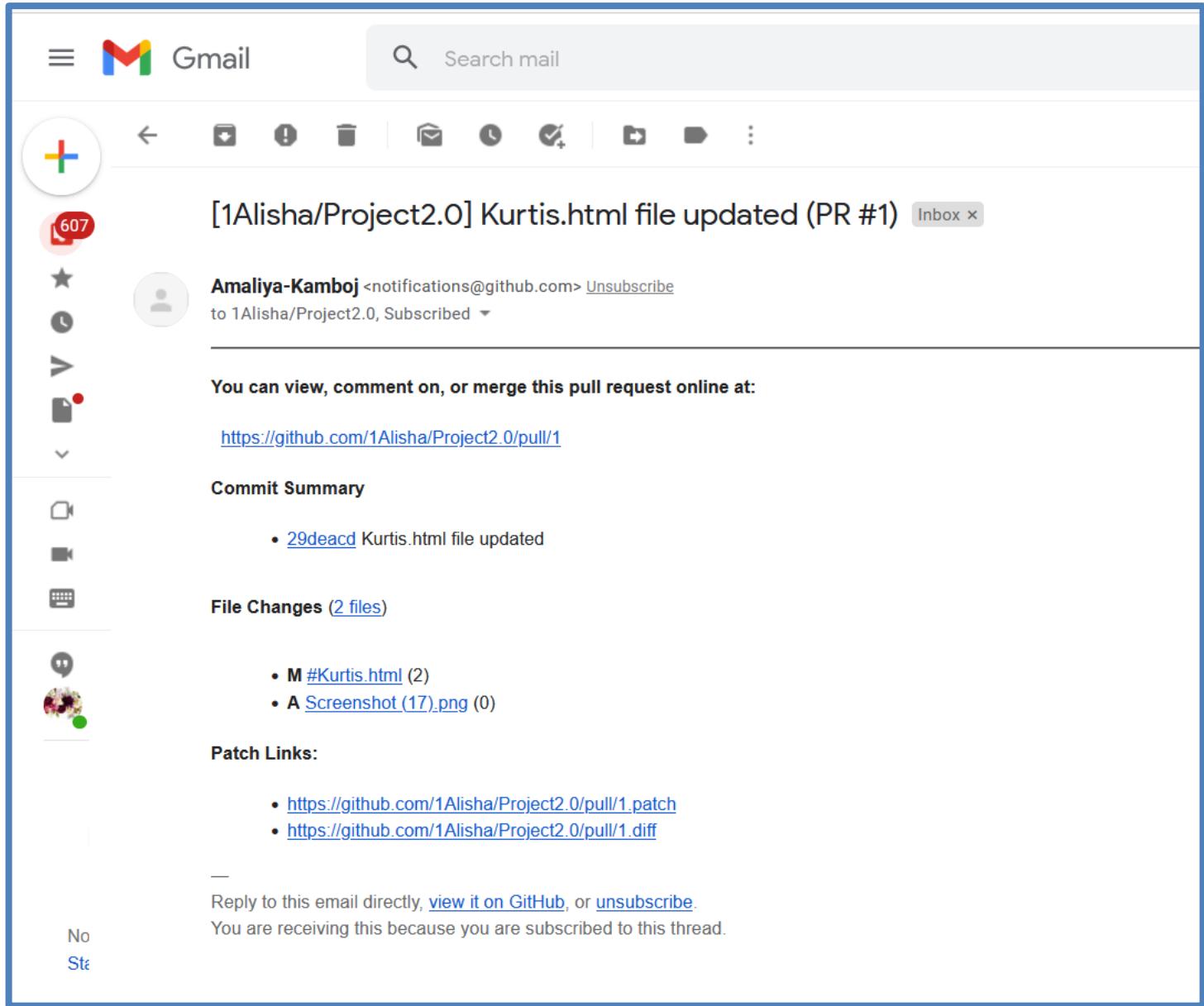
Follow us on



Only home page, Gown's page, payment gateway, Navbar is inserted in it.

Now collaborators have to do fork, clone “Project2.0” repository in localdrive, add other web pages, commit it, pull it in their fork repository and send me ‘pull request’ so that I can add those web pages in my website.

## Amaliya's snapshot's of collaboration ->



The screenshot shows a Gmail inbox with a single email from Amaliya-Kamboj. The subject of the email is "[1Alisha/Project2.0] Kurtis.html file updated (PR #1)". The email body contains a link to view the pull request online at <https://github.com/1Alisha/Project2.0/pull/1>. It also includes a Commit Summary with one commit: "29deacd Kurtis.html file updated". The File Changes section shows two files: "#Kurtis.html" (2) and "Screenshot (17).png" (0). The Patch Links section provides two links: <https://github.com/1Alisha/Project2.0/pull/1.patch> and <https://github.com/1Alisha/Project2.0/pull/1.diff>.

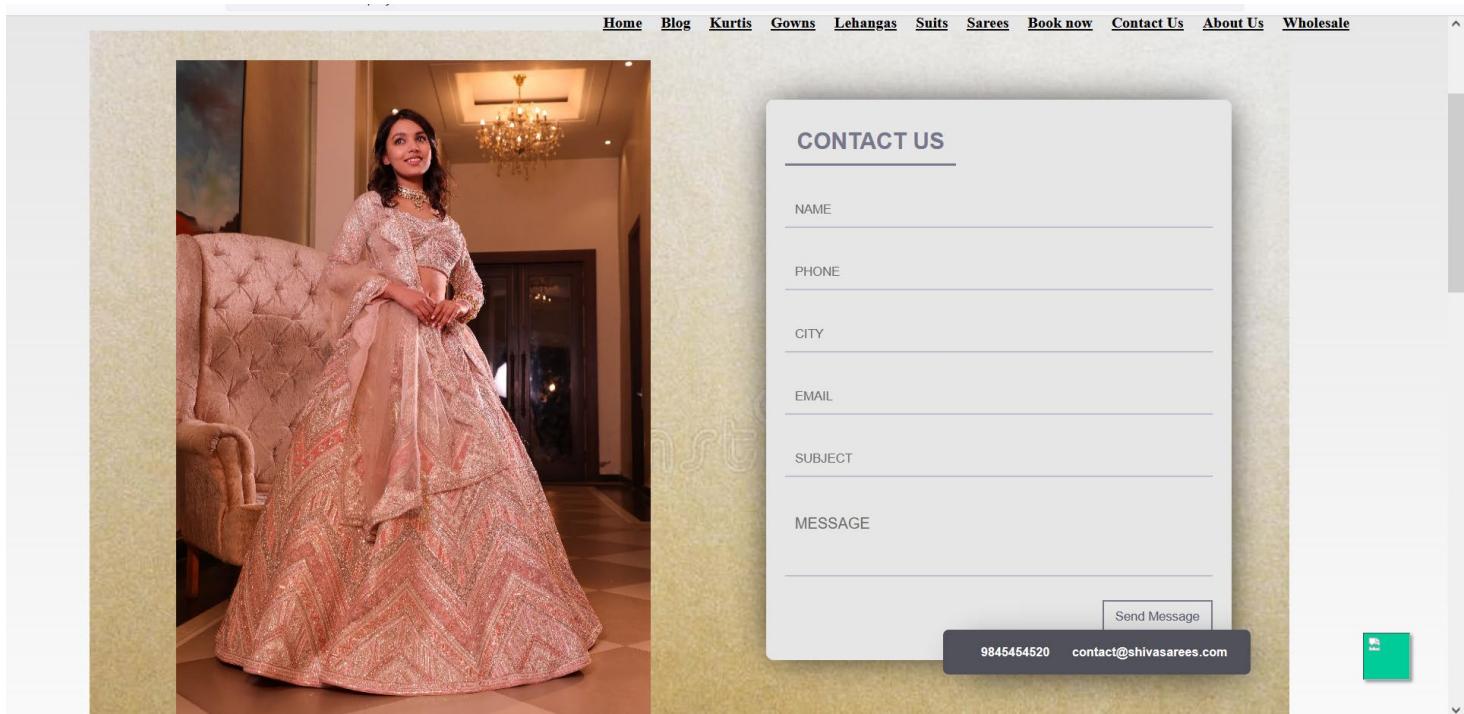
Kurti's page and contact us page added by Amaliya



Flawless Beauty

Enchanting Beauty ₹999 Enquire now

The Stunner



[Home](#) [Blog](#) [Kurtis](#) [Gowns](#) [Lehangas](#) [Suits](#) [Sarees](#) [Book now](#) [Contact Us](#) [About Us](#) [Wholesale](#)

**CONTACT US**

NAME

PHONE

CITY

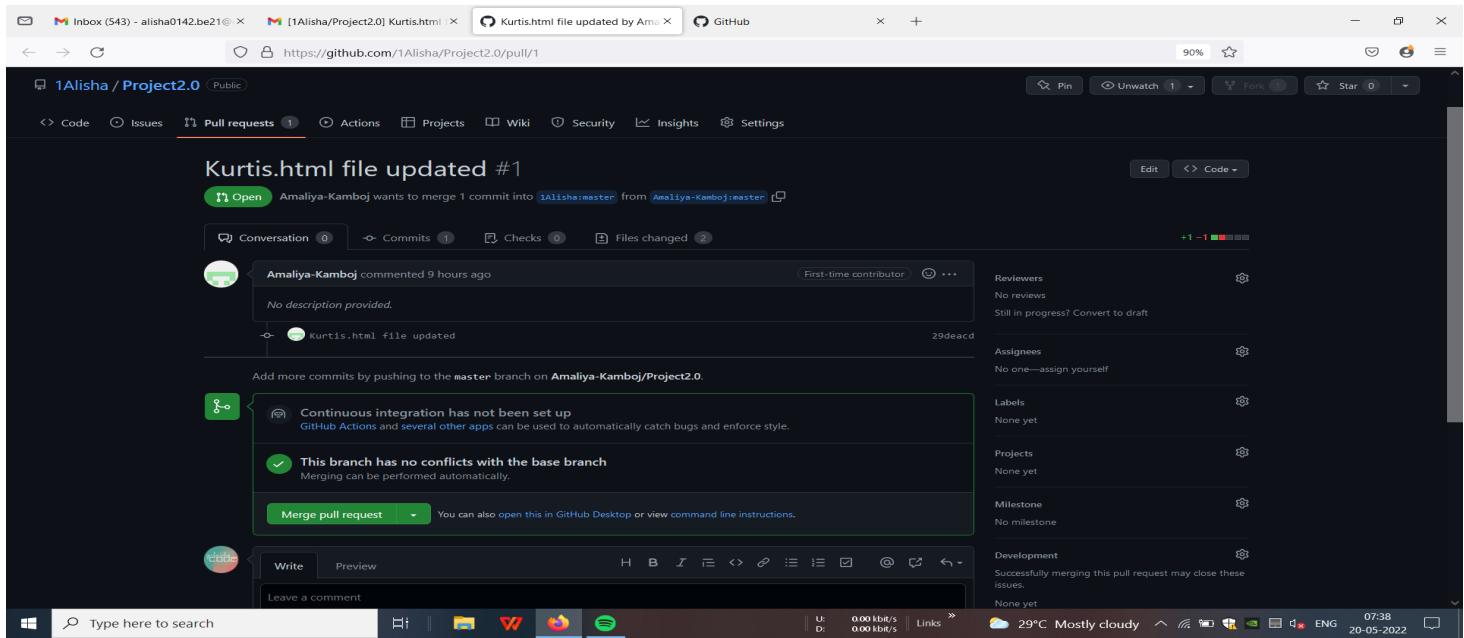
EMAIL

SUBJECT

MESSAGE

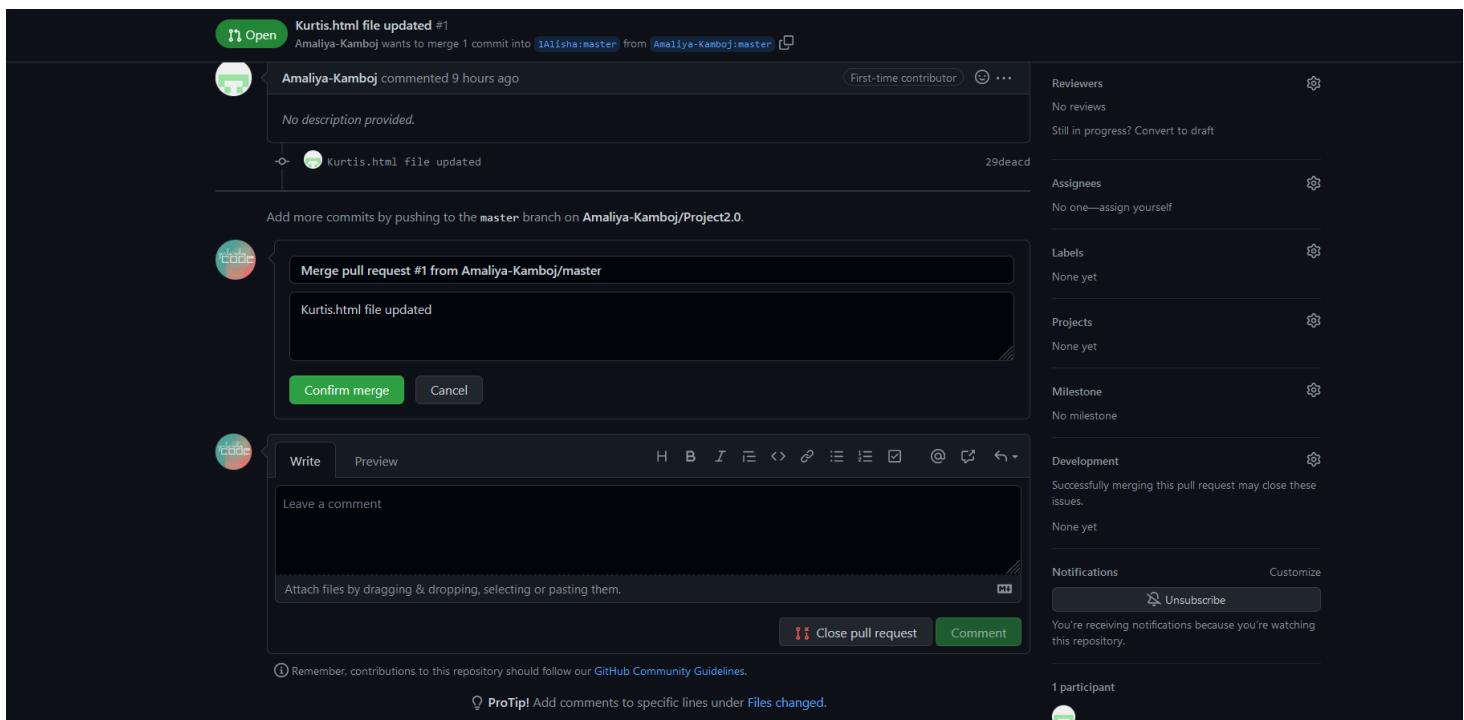
9845454520 contact@shivasarees.com

## Opened the pull request sent by Amaliya.



The screenshot shows a GitHub pull request page for a repository named '1Alisha / Project2.0'. The pull request is titled 'Kurtis.html file updated #1'. It is from the user 'Amaliya-Kamboj' to the branch '1Alisha:master'. The commit 'Kurtis.html file updated' has a green checkmark next to it, indicating no conflicts. The merge button is visible at the bottom left. The GitHub interface includes a navigation bar with 'Code', 'Issues', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', 'Insights', and 'Settings'. On the right side, there are sections for 'Reviewers', 'Assignees', 'Labels', 'Projects', 'Milestone', and 'Development'.

Added Kurtis.html file in the website and confirmed merging of 1Alisha:master from AmaliyaKamboj:master.



The screenshot shows the same GitHub pull request page after the merge. A modal window titled 'Merge pull request #1 from Amaliya-Kamboj/master' is open, showing the commit 'Kurtis.html file updated'. The 'Confirm merge' button is highlighted in green. The GitHub interface includes a navigation bar with 'Code', 'Issues', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', 'Insights', and 'Settings'. On the right side, there are sections for 'Reviewers', 'Assignees', 'Labels', 'Projects', 'Milestone', and 'Development'.

## Now to Publish and print network graphs.

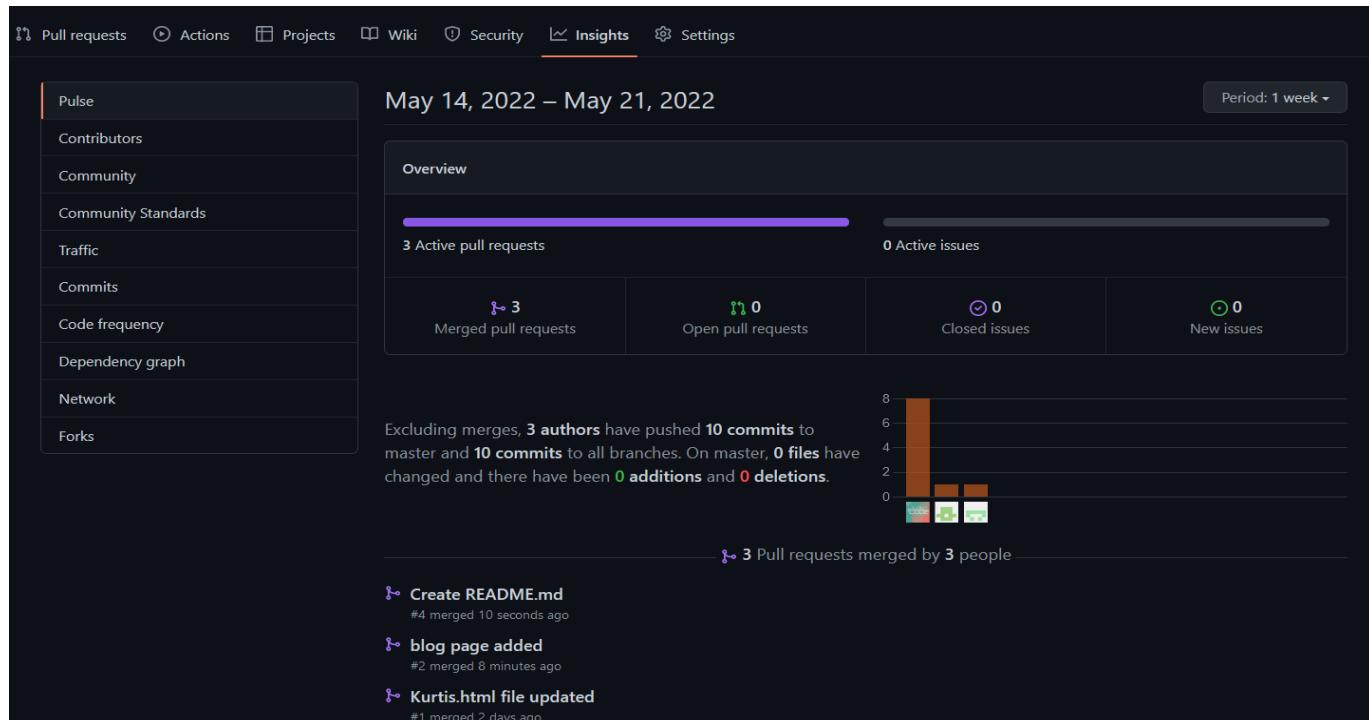
The network graph is one of the useful features for developers on GitHub. It is used to display the branch history of the entire repository network, including branches of the root repository and branches of forks that contain commits unique to the network. A repository's graphs give you information on 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
- 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.



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**.

S.no.	Links
1.	<a href="https://stackoverflow.com/questions/3528245/whats-the-difference-between-git-reset-mixed-soft-and-hard">https://stackoverflow.com/questions/3528245/whats-the-difference-between-git-reset-mixed-soft-and-hard</a>
2.	<a href="https://initialcommit.com/blog/Technical-Guide-VCS-Internals">https://initialcommit.com/blog/Technical-Guide-VCS-Internals</a>
3.	<a href="https://www.geeksforgeeks.org/version-control-systems/">https://www.geeksforgeeks.org/version-control-systems/</a>
4.	<a href="https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control">https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control</a>
5.	<a href="https://dzone.com/articles/top-20-git-commands-with-examples">https://dzone.com/articles/top-20-git-commands-with-examples</a>
6.	<a href="https://www.toolsqa.com/git/what-is-git/">https://www.toolsqa.com/git/what-is-git/</a>
7.	<a href="https://www.javatpoint.com/git-reset">https://www.javatpoint.com/git-reset</a>

