TASK 1.1

SOURCE CODE MANAGEMENT

(CS-181)

Submitted By:
Aakriti Bahl
Roll no.-2110990008

Submitted To:
Dr. Monit Kapoor

Experiment 1

AIM: Setting up of GIT client.

THEORY:

- **GIT:** Git is a distributed, open-source version control system (VCS) that enables you to store code, track revision history, merge code changes, and revert to earlier code versions when needed. It was created by Linus Torvalds in 2005, and has been maintained by Junio Hamano since then.

PROCEDURE:

- On Mac (running OS X or greater) GIT is pre installed.
- Check the version by running **git –version** in the terminal.

EXPERIMENT 2

AIM: Setting up a GitHub account.

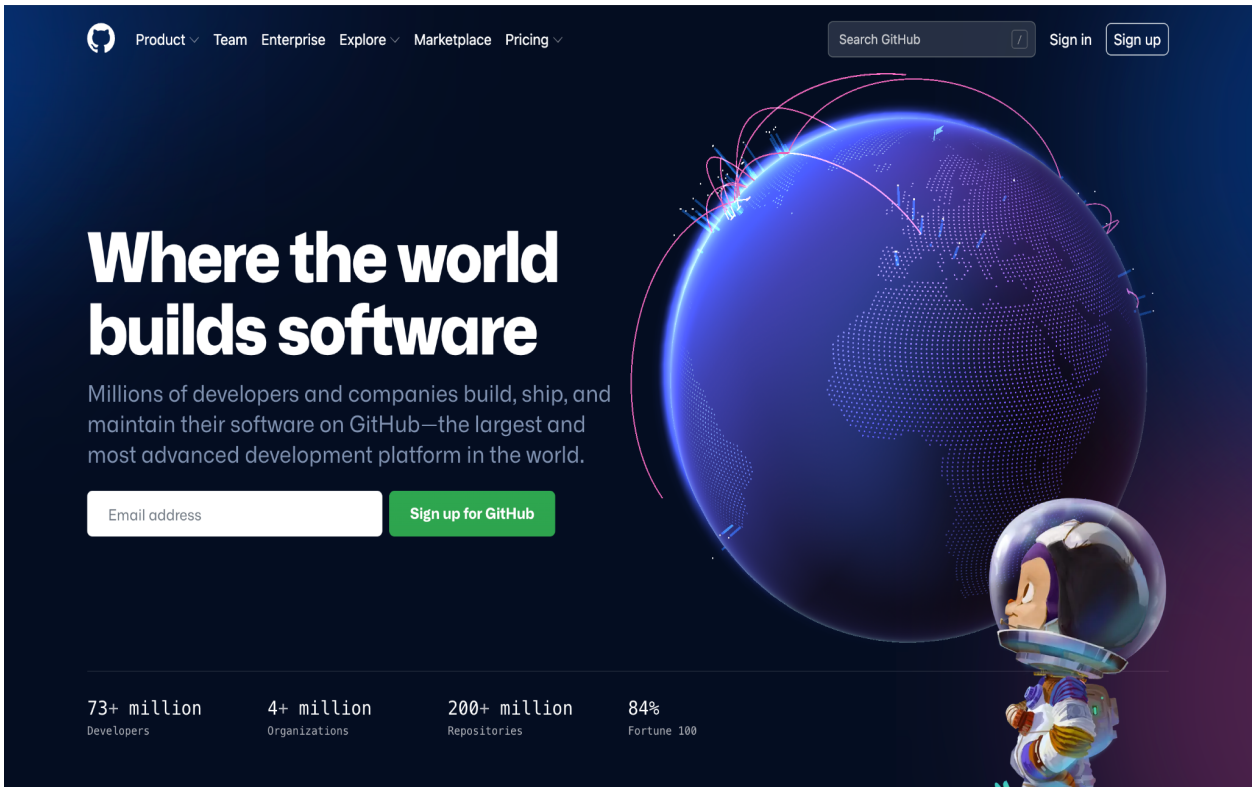
THEORY:

- **GITHUB:** Github is an online software development platform used for storing, tracking, and collaborating on software projects. GitHub is the largest host of source code in the world, and has been owned by Microsoft since 2018.

PROCEDURE:

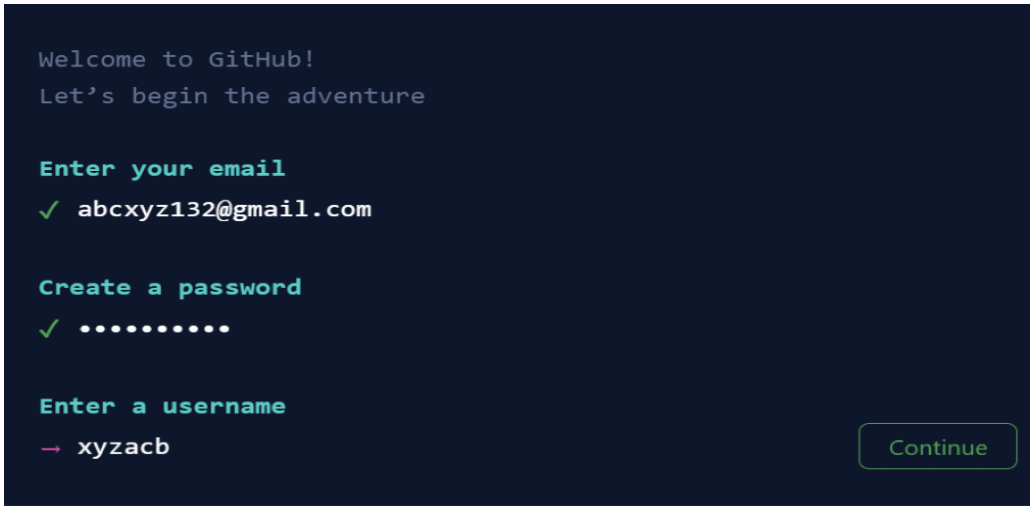
1. Search for GitHub on any search engine.

2. If you already have an account then click on **Sign in** or else



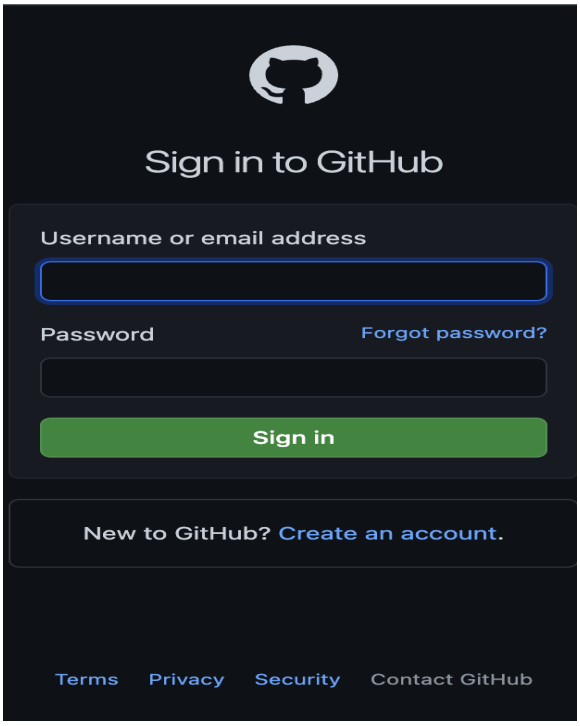
create an account by clicking on the **Sign up** option.

3. To create a new account:
    - Click on the Sign **up** option.

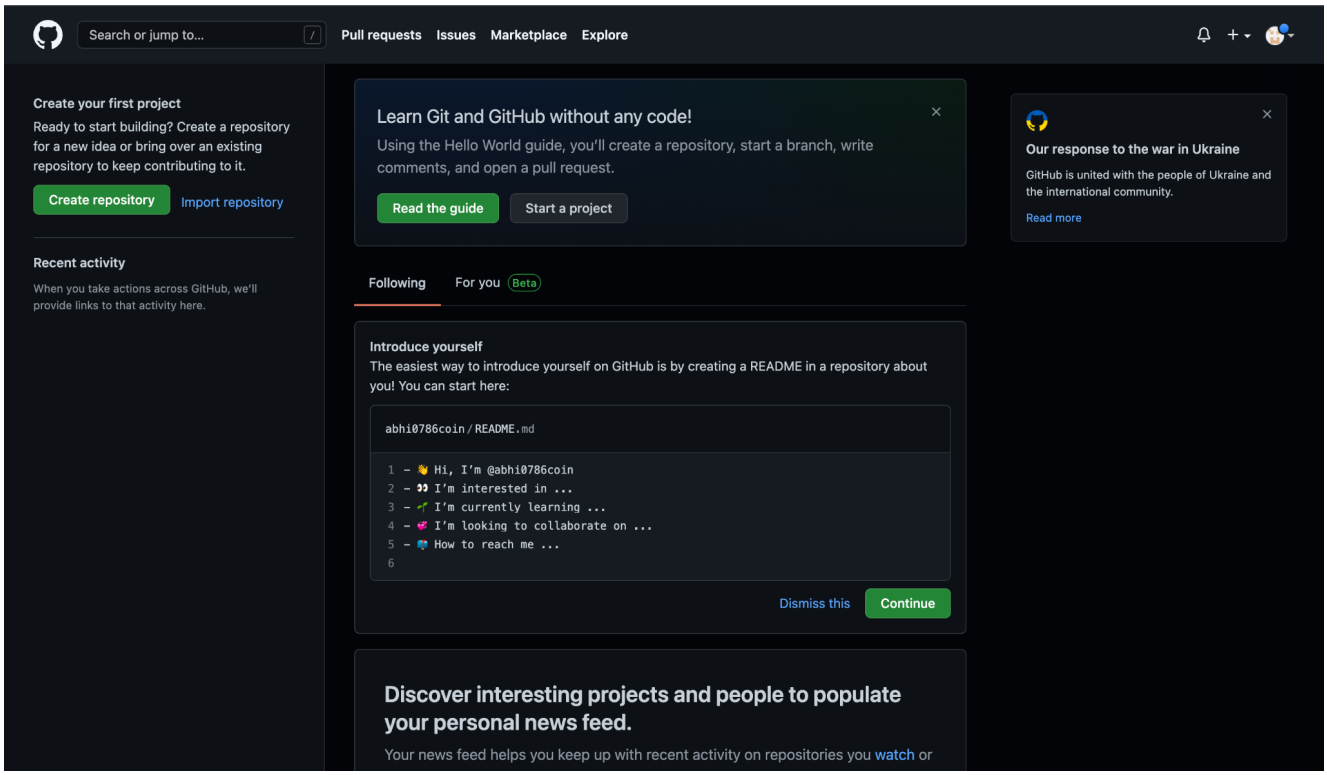


- Enter email address and then enter password.
- Enter a username and solve the captcha.
- Then enter the verification code sent on your email and your account will be created.
- Personalize your account according to your needs.
- Your account is set up.

4. If you already have an account then click on the Sign **in** option.



5. Interface of GitHub:

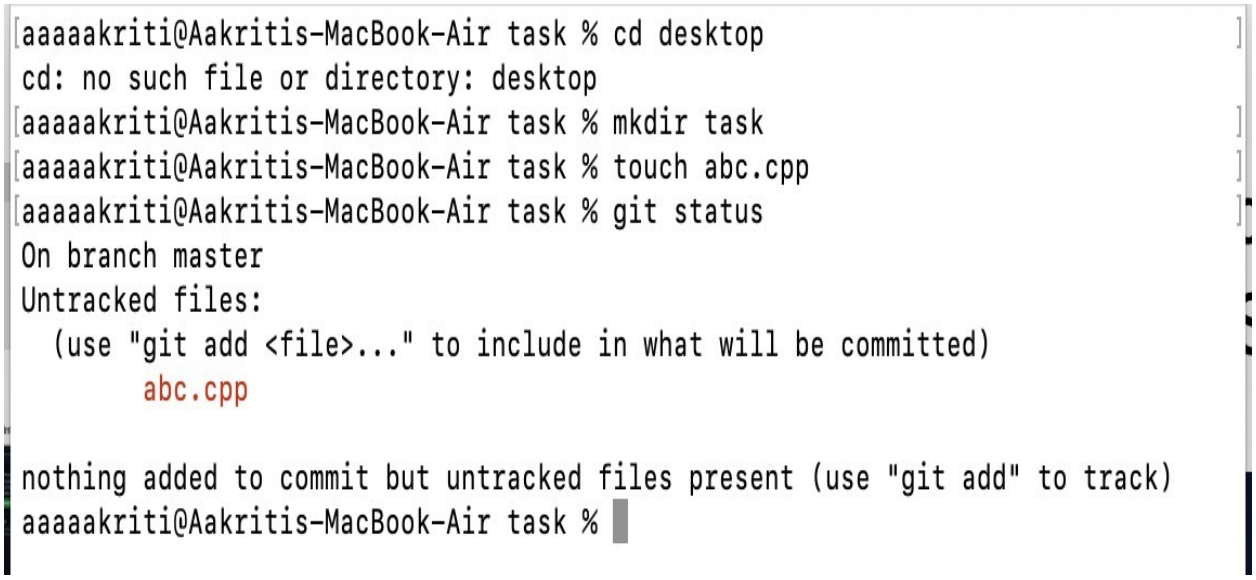


6. To link GitHub with Git Bash use the following commands.
    - For username:
      git config –global user.name"<Enter GitHub username>"
    - For email:
      git config –global user.email"<Enter GitHub email>"

EXPERIMENT 3

AIM: Generating Logs.
THEORY: 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. Generally git log is a record of commits.
PROCEDURE:

1. Start a repo on your machine by:
    - Create a directory on your machine.



    - Initialize a repo and check the status of files. As abc.cpp is in red color, means that it is unstaged.

```
aaaaakriti@Aakritis-MacBook-Air task % git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /Users/aaaaakriti/Desktop/task/.git/
aaaaakriti@Aakritis-MacBook-Air task % git status
On branch master

No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)
aaaaakriti@Aakritis-MacBook-Air task %
```

- Add the file using **git add .** command and then check the status of the file. Now the file name is shown in green which means that the file is staged and ready to commit.

```
aaaaakriti@Aakritis-MacBook-Air task % git add .
aaaaakriti@Aakritis-MacBook-Air task % git status
On branch master

No commits yet

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

aaaaakriti@Aakritis-MacBook-Air task %
```

2. Make some changes and commit at each step.
3. Run the **git log** command in the terminal to see the commits you made.

```
aaaaakriti@Aakritis-MacBook-Air task % git commit -m"sample commit"
[master (root-commit) 40f8e81] sample commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 abc.cpp
aaaaakriti@Aakritis-MacBook-Air task % git log
commit 40f8e81226b67e518a03006ed61831e61612fa8e (HEAD -> master)
Author: Aakriti Behl <aakriti0008.be21@chitkara.edu.in>
Date:   Mon Apr 11 21:16:41 2022 +0530

    sample commit
aaaaakriti@Aakritis-MacBook-Air task %
```

EXPERIMENT 4

AIM: To create and visualize branches.

```
aaaaakriti@Aakritis-MacBook-Air task %
aaaaakriti@Aakritis-MacBook-Air task % git branch
* master
aaaaakriti@Aakritis-MacBook-Air task % git branch example
```

THEORY: The main branch in git is called the master branch. But we can make branches out of this main master branch.

PROCEDURE:

1. To create a new branch use command:
   git branch <name of branch>
2. To check branches use command:
   git branch

```
aaaaakriti@Aakritis-MacBook-Air task %
aaaaakriti@Aakritis-MacBook-Air task % git branch
* master
aaaaakriti@Aakritis-MacBook-Air task % git branch example
aaaaakriti@Aakritis-MacBook-Air task % git branch
  example
* master
aaaaakriti@Aakritis-MacBook-Air task %
```

3. To change present working branch use command:
   git checkout <branch name>

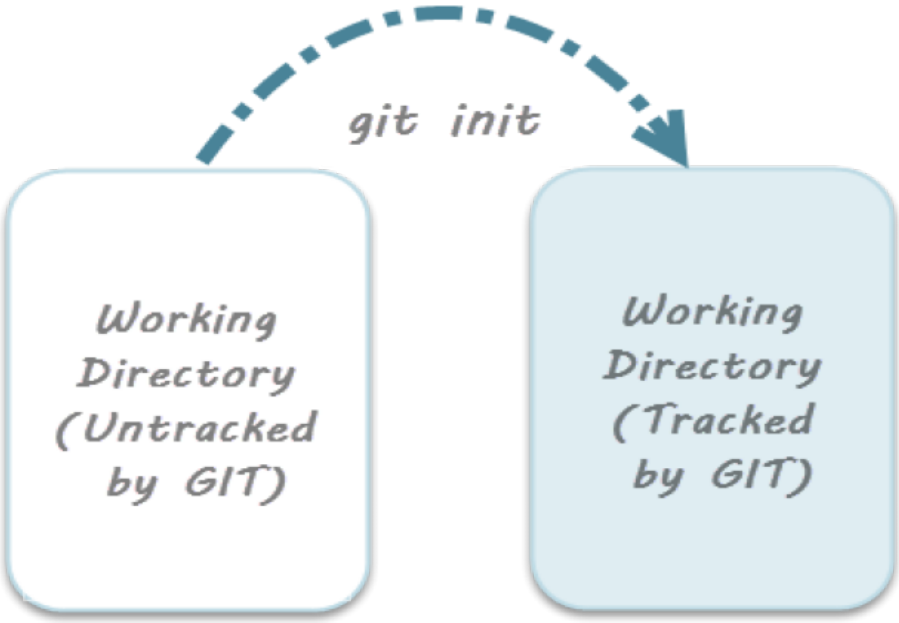
```
aaaaakriti@Aakritis-MacBook-Air task % git checkout example
Switched to branch 'example'
aaaaakriti@Aakritis-MacBook-Air task % git branch
* example
  master
aaaaakriti@Aakritis-MacBook-Air task %
```

EXPERIMENT 5

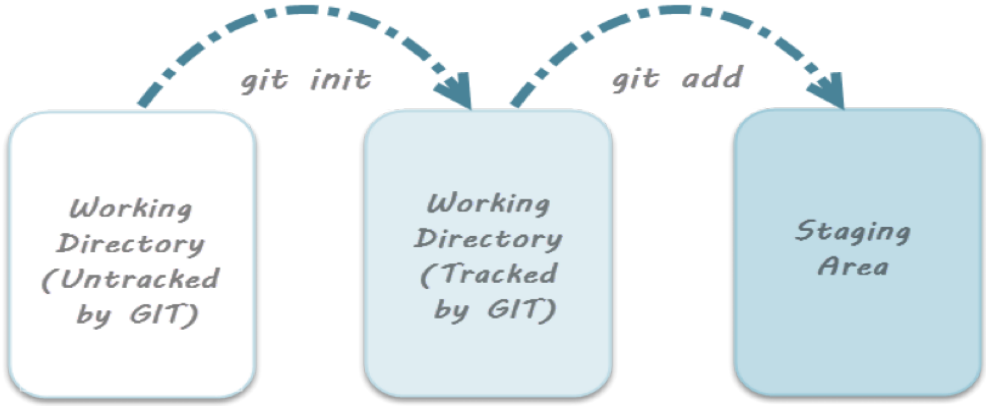
AIM: To explain the GIT lifecycle.
THEORY: When a project is under the Git version control system, they are present in three major Git states in addition to these basic ones. Here are the three Git states:

1. **Working Directory:** The git init command is used to initialize our local project directory to make it a git repository. After running this command, git becomes aware of the files in the project,

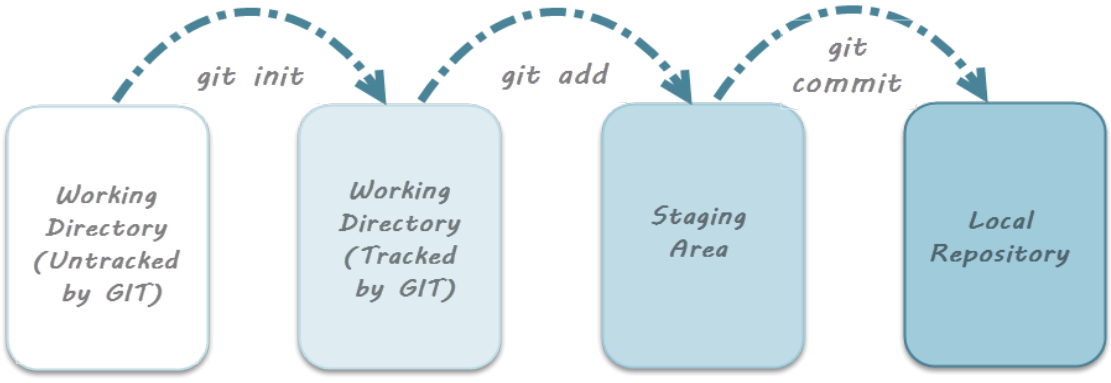
though it does not yet track them. In the staging area, the files are tracked further.



2. **Staging Area:** To keep track of the various versions of our files, we now use the command git add. A staging area is a location where different versions of our files are kept. The git add command moves your file's version from your working directory to the staging area. We can, however, choose which files to add to the staging area because there are some files in our working directory that we don't want tracked, such as node modules, env files, temporary files, and so on. Indexing in Git is the process by which Git determines which files need to be added or sent. Your staging area can be found inside the index file in the.git folder.



3. **GIT Directory:** Now that we have all of the files that need to be tracked ready in the staging area, we can commit them using the git commit command. Commit assists us in keeping track of the metadata associated with the files in our staging area. Every commit is accompanied by a message that describes what the commit is about. Git saves the information or metadata of the files that were committed in a Git Directory, which aids Git in file tracking and essentially saves a photocopy of the committed files. Commit also stores the name of the author who made the commit, the files that were committed, and the date they were committed, in addition to the commit message.



Subject Name: **Source Code Management**

Subject Code: **CS181**

Cluster: **Beta**

Department: **CSE**

Submitted By:
Aakriti Bahl
2110990008
G01-A

Submitted To:
Dr.Monit kapoor

Aim: Add collaborators on GitHub Repo

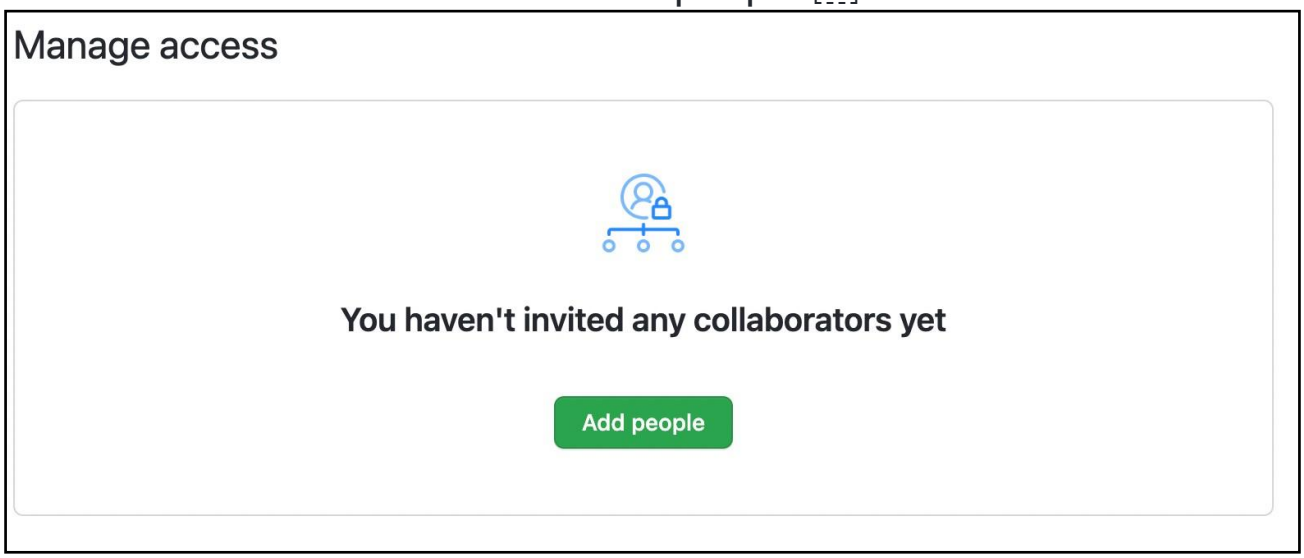
1. Ask for the username of the person you're inviting as a collaborator. If they don't have a username yet, they can sign up for GitHub For more information, see "Signing up for a new GitHub account".

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

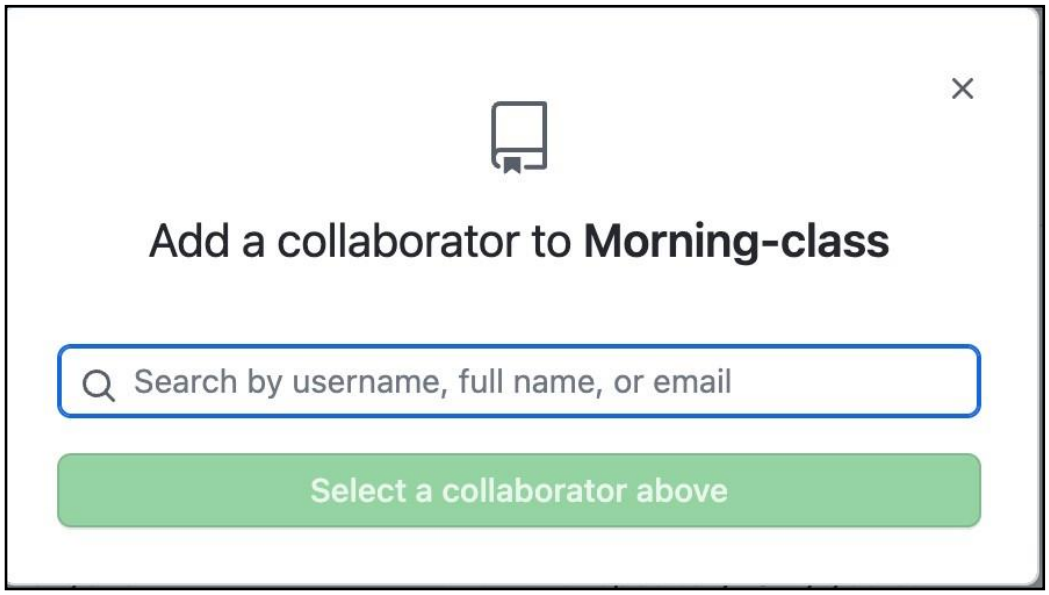
3. Under your repository name, click **Settings**.

4. In the “Access” section of sidebar, click **Collaborators & teams.**

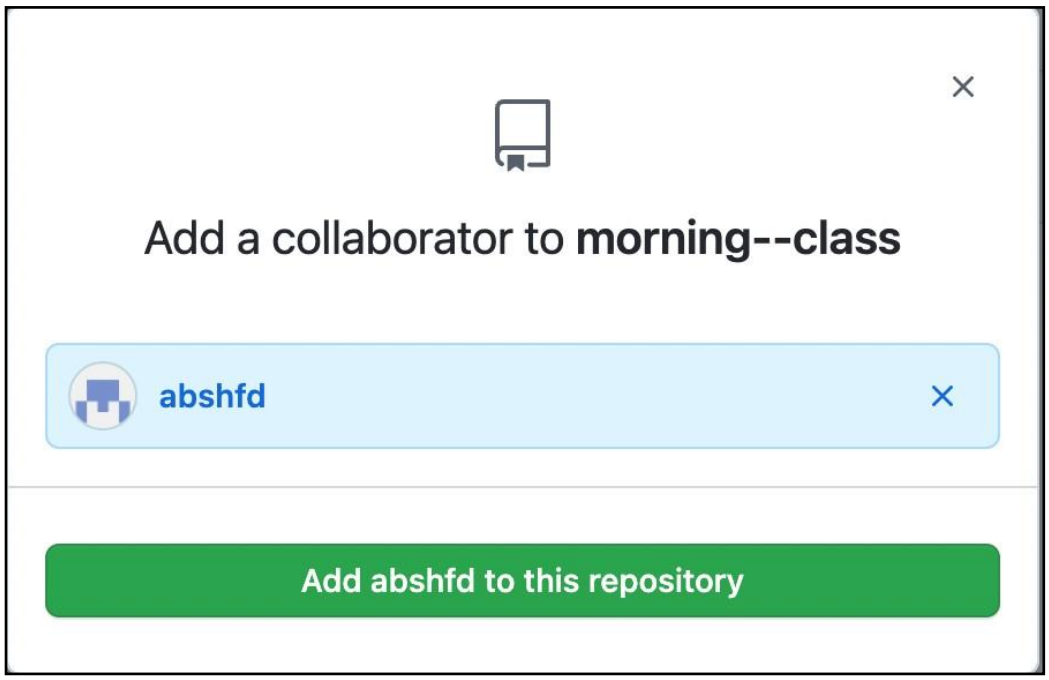
5. Click Invite a collaborator/Add people.



6. In the search field, start typing the name of person you want to invite , then click a name in the list of matches



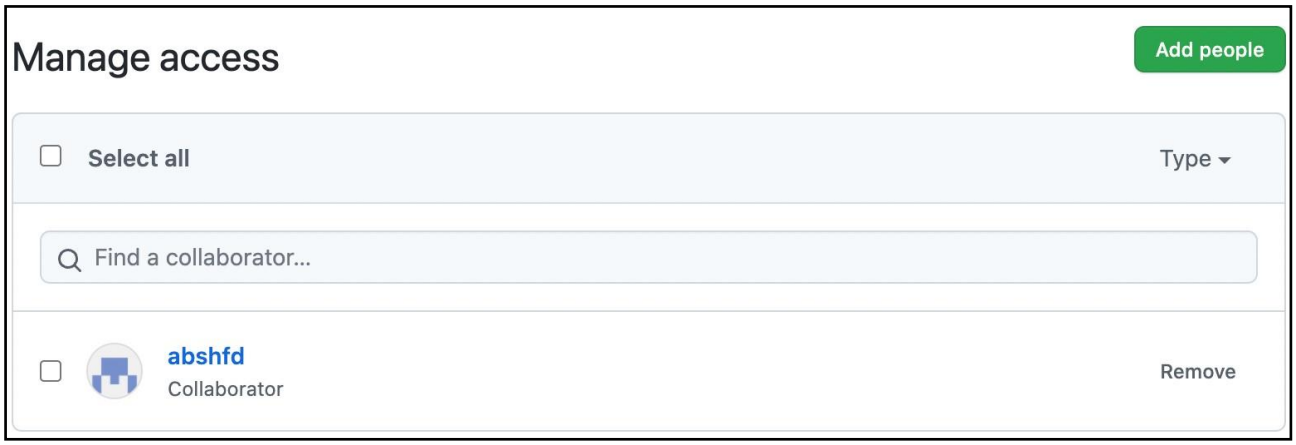
7. Click **Add NAME to REPOSITORY.**



8. The user will receive an email inviting them to the repository.

9. Once they accept your invitation, they will have collaborator access to your repository.

10. You can manage collaborator(add/remove) from Manage access.



(If we are collaborator and we want to push our changes to the cloud repo we can do by using:

Git push -u origin master

This will be discussed further in Experiment 8)

Aim: Fork and Commit

About forks

Most commonly, forks are used to either propose changes to someone else's project or to use someone else's project as a starting point for your own idea. You can fork a repository to create a copy of the repository and make changes without affecting the upstream repository. For more information, see "Working with forks."

Propose changes to someone else's project

For example, you can use forks to propose changes related to fixing a bug. Rather than logging an issue for a bug you've found, you can:

- Fork the repository.
- Make the fix.
- Submit a pull request to the project owner.

Prerequisites

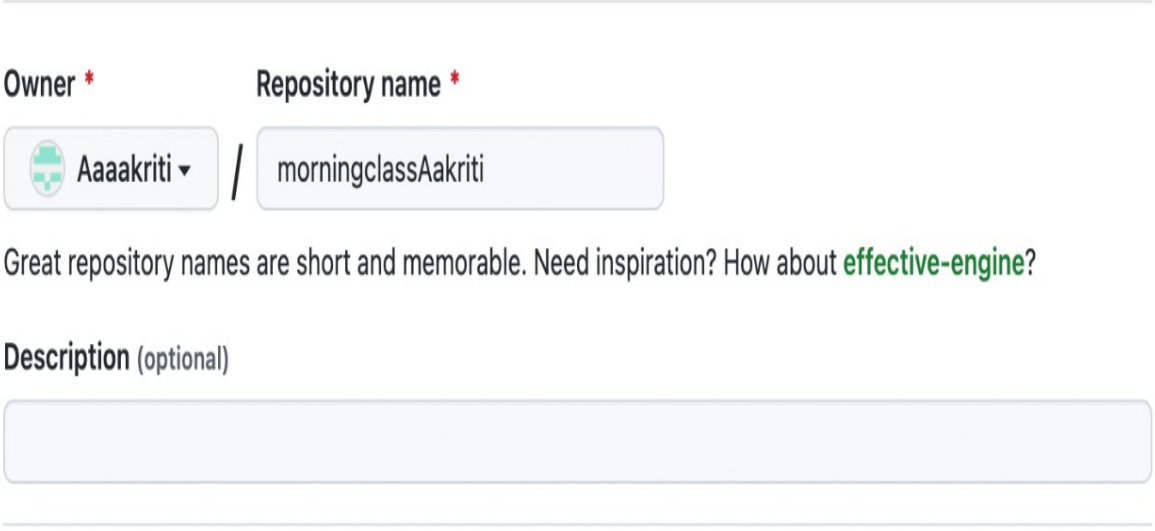
If you haven't yet, you should first set up Git. Don't forget to set up authentication to GitHub.com from Git as well.

Forking a repository

You might fork a project to propose changes to the upstream, or original, repository. In this case, it's good practice to regularly sync your fork with the upstream repository. To do this, you'll need to use Git on the command line. You can practice setting the upstream repository using the same deep1789/ morningclass19.4.22 repository you just forked.

1. On GitHub.com, navigate to the https://github.com/deep1789/ morningclass19.4.22 repository.
2. In the top-right corner of the page, click **Fork**.

3. Now click on create fork by renaming the existing name or using it as it is.

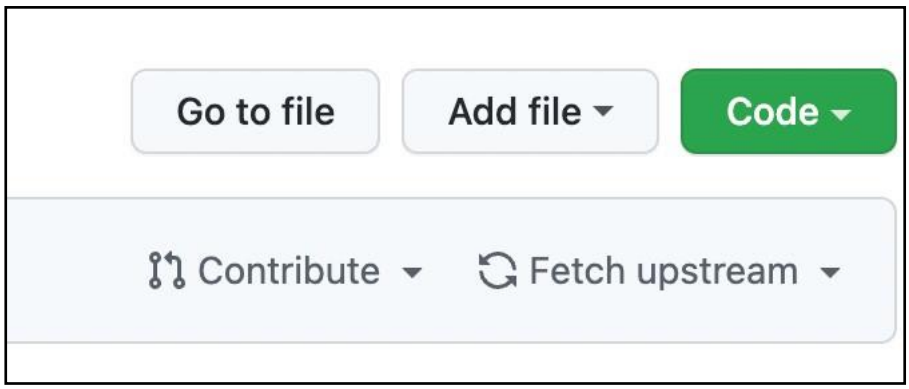


Cloning your forked repository

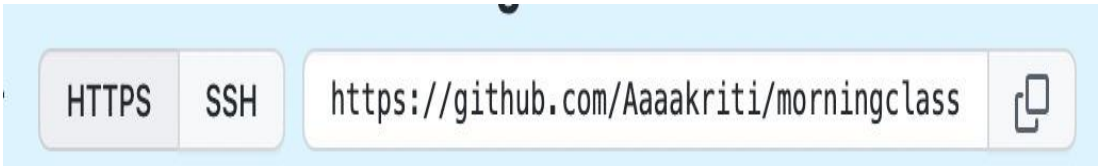
Right now, you have a fork of the morning-class repository, but you don't have the files in that repository locally on your computer.

1. On GitHub.com, navigate to **your fork** of the repository.

2. Above the list of files, click **Code**.



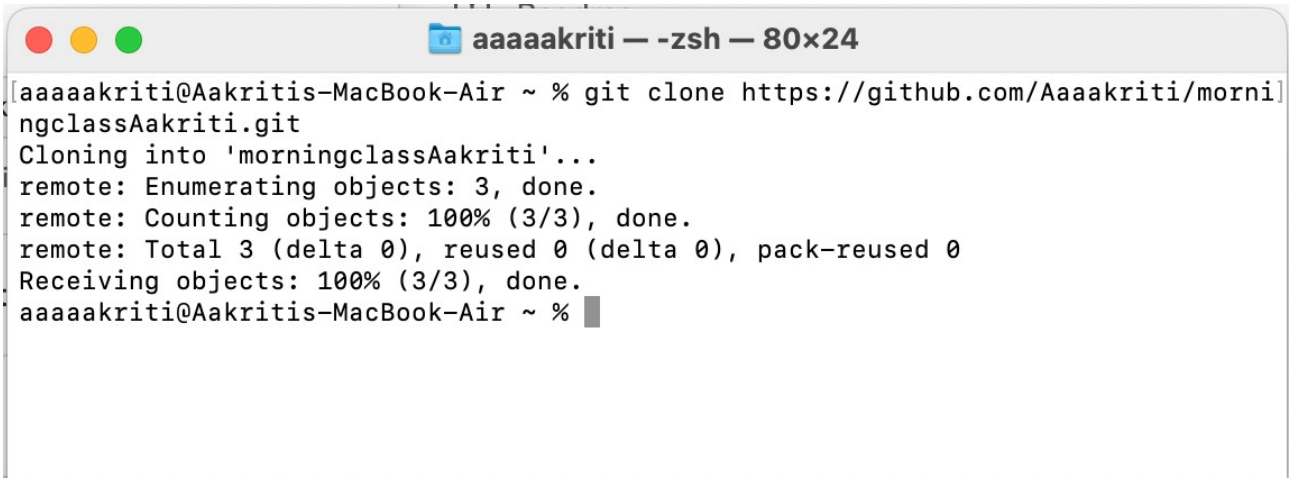
3. To clone the repository using HTTPS, under "Clone with HTTPS",copy the url, click . To clone the repository using an SSH key, including a certificate issued by your organisation's SSH certificate authority, click **Use SSH**, then click . To clone a repository using GitHub CLI, click **Use GitHub CLI,** then click.



4.Open Terminal.
5. Change the current working directory to the location where you want the cloned directory.

6.Type git clone, and then paste the URL you copied earlier. It will look like this, with your GitHub username instead of YOUR-USERNAME:
$ git clone https://github.com/YOUR-USERNAME/morningclassaak

Press **Enter**. Your local clone will be created.

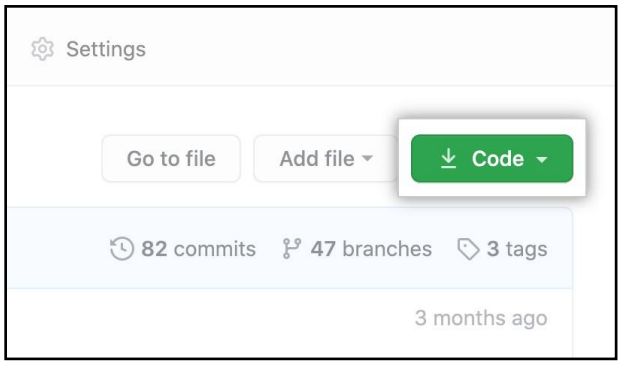


Configuring Git to sync your fork with the original repository

When you fork a project in order to propose changes to the original repository, you can configure Git to pull changes from the original, or upstream, repository into the local clone of your fork.

1. On GitHub.com, navigate to the morning-class repository.

2. Above the list of files, click **Code**.



3. To clone the repository using HTTPS, under "Clone with HTTPS", click .

4. To clone the repository using an SSH key, including a certificate issued by your organization's SSH certificate authority, click **Use SSH**, then click .

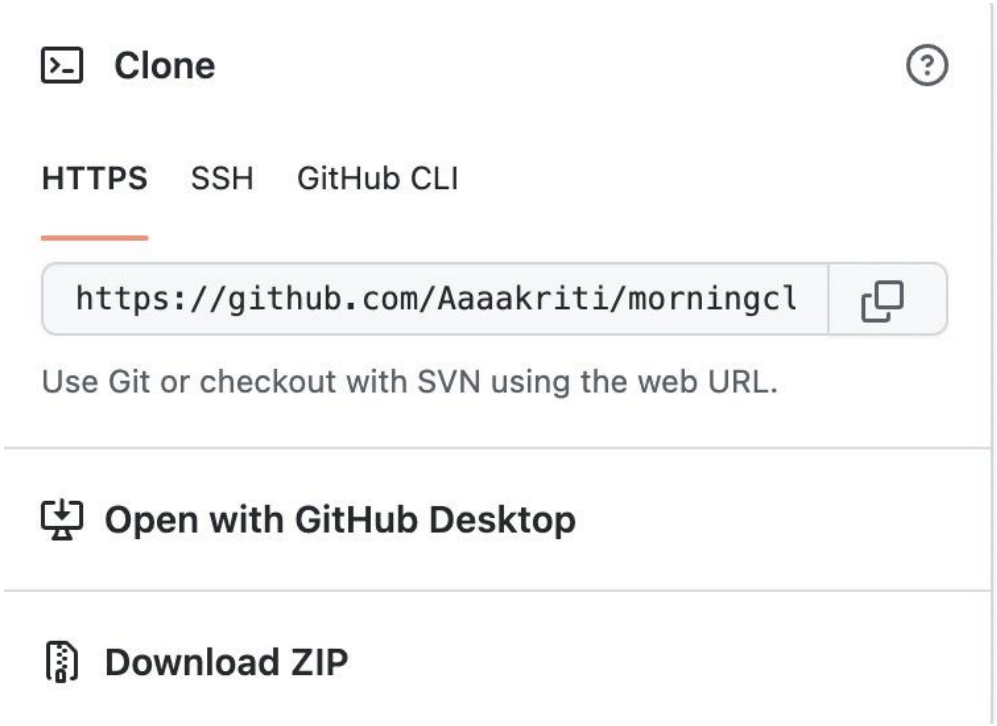
5. To clone a repository using GitHub CLI, click **Use GitHub CLI**,then click .

6. Open terminal.

```
aaaaakriti@Aakritis-MacBook-Air morningclassAaakriti % git remote -v
upstream        https://github.com/Aaaakriti/task2.git (fetch)
upstream        https://github.com/Aaaakriti/task2.git (push)
aaaaakriti@Aakritis-MacBook-Air morningclassAaakriti %
```

7.git remote -v and press **Enter**. You'll see the current configured remote repository for your fork.

8.Type git remote add upstream, and then paste the URL you copied in Step 5 and press **Enter**. It will look like this:



7. To verify the new upstream repository you've specified for your fork, type git remote -v again. You should see the URL for your fork as origin, and the URL for the original repository as upstream.

```
aaaaakriti@Aakritis-MacBook-Air morningclassAaakriti % git remote -v
upstream        https://github.com/Aaaakriti/task2.git (fetch)
upstream        https://github.com/Aaaakriti/task2.git (push)
aaaaakriti@Aakritis-MacBook-Air morningclassAaakriti %
```

Next steps

You can make any changes to a fork, including:

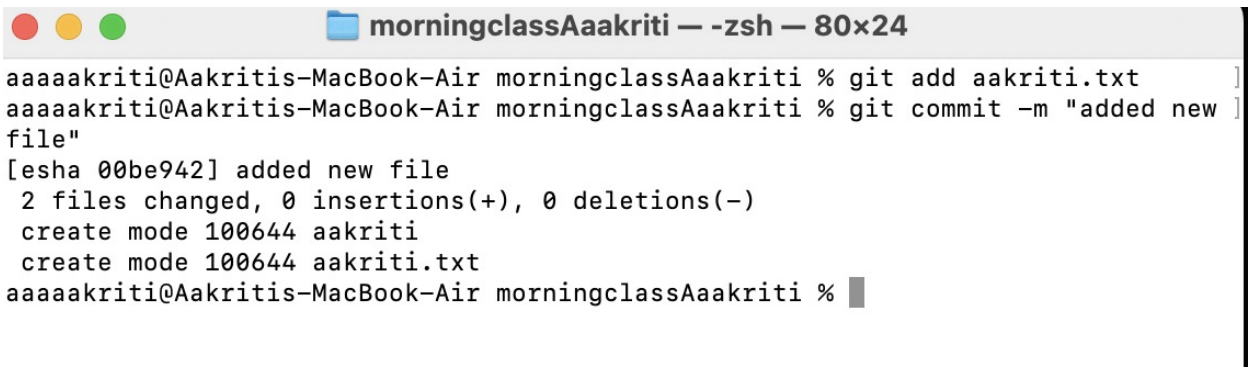
- **Creating branches:** *Branches* allow you to build new features or test out ideas without putting your main project at risk.
- **Opening pull requests:** If you are hoping to contribute back to the original repository, you can send a request to the original author to pull your fork into their repository by submitting a pull request.

Committing changes to club repo of host:

We can commit our changes to the original repo of the host if we are collaborator

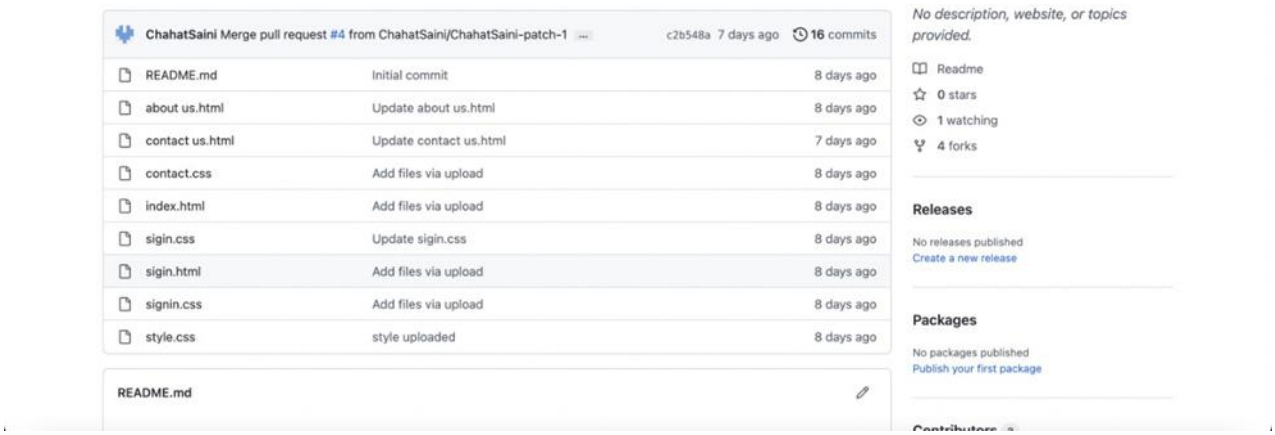
So morning—class-Aryan has included me as a collaborator .
First of all clone it and make a local repo.

Lets make a new file named aakriti.txt and take it to staging area
And then commit it to local repo.



Now you can push this commit to main repo using git push origin master

Now check in GitHub and search the repository the file has been added and commit has been done.



Aim: Merge and Resolve conflicts created due to own activity and collaborators activity.

FOR THIS EXP. I HAD COLLABORATE WITH JACK1299 IN REPO EXP8so things can go wrong, which usually starts with a **merge conflict**, due to both collaborators making incompatible changes to a file. While the error messages from merge conflicts can be daunting, getting things back to a normal state can be straightforward once you've got an idea where the problem lies.

A merge conflict occurs when both the owner and collaborator change the same lines in the same file without first pulling the changes that the other has made. This is most easily avoided by good communication about who is working on various sections of each file, and trying to avoid overlaps. But sometimes it happens, and *git* is there to warn you about potential problems. And git will not allow you to overwrite one person's changes to a file with another's changes to the same file if they were based on the same version.

The main problem with merge conflicts is that, when the Owner and Collaborator both make changes to the same line of a file, git doesn't know whose changes take precedence.

You have to tell git whose changes to use for that line.

Producing and resolving merge conflicts

First of all create a Repo and invite a collaborator as in exp6.

To illustrate this process, we're going to carefully create a merge conflict step by step, show how to resolve it, and show how to see the results of the successful merge after it is complete. First, we will walk through the exercise to demonstrate the issues.

Owner and collaborator ensure all changes are updated

First, start the exercise by ensuring that both the Owner and Collaborator have all of the changes synced to their local copies of the Owner's repository .This includes doing a git pull to ensure that you have all changes local, and make sure that the Git status in doesn't show any changes needing to be committed.

Owner makes a change and commits

From that clean slate, the Owner first modifies and commits a small change including their name on a specific line of the README.md file. Work to only change that one line, and add your username to the line in some form and commit the changes (but DO NOT push). We are now in the situation where the owner has unpushed changes that the collaborator can not yet see.

Collaborator makes a change and commits on the same line

Now the collaborator also makes changes to the same (line 2) of the README.md file copy of the project, adding their name to the line. They then commit. At this point, both the owner and collaborator have committed changes based on their shared version of the README.md file, but neither has tried to share their changes via GitHub.

Collaborator pushes the file to GitHub

Sharing starts when the Collaborator pushes their changes to the GitHub repo, which updates GitHub to their version of the file.

Owner trying to push changes

The owner is now one revision behind, but doesn't yet know it.
Owner pushes their changes and gets an error
At this point, the owner tries to push their change to the repository, which triggers an error from GitHub. While the error message is long, it basically tells you everything needed (that the owner's repository doesn't reflect the changes on GitHub, and that they need to pull before they can push).

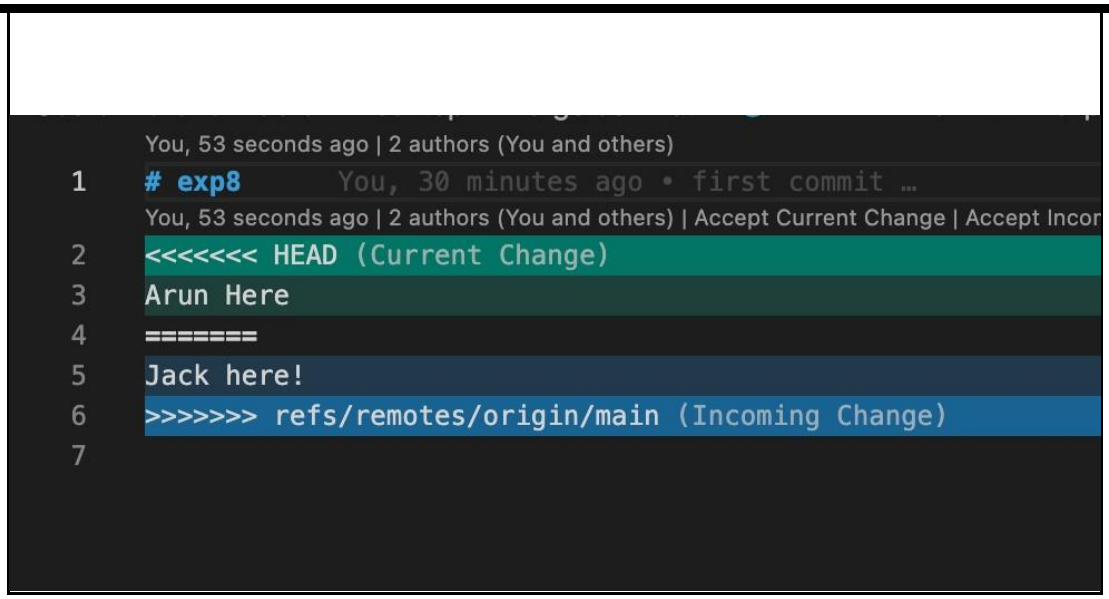
```
aaaaakriti@Aakritis-MacBook-Air merge conflict % git push origin main
To https://github.com/ Aaakriti  exp8.git
 ! [rejected]        main -> main (fetch first)
error: failed to push some refs to 'https://github.com/Arunch367/exp8.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Resolving Commit

To resolve the conflict, the Owner now needs to edit the file. Again, as indicated above, git has flagged the locations in the file where a conflict occurred with <<<<<<<, =======, and >>>>>>>. The Owner should edit the file, merging whatever changes are appropriate until the conflicting lines read how they should, and eliminate all of the marker lines with with <<<<<<<,.======= and >>>>>>>>

```
aaaaakriti@Aakritis-MacBook-Air merge conflict % git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 242 bytes | 80.00 KiB/s, done.
From https://github.com/Aaakriti /exp8
   2119e86..6fe0c24  main       -> origin/main
hint: You have divergent branches and need to specify how to reconcile them.
hint: You can do so by running one of the following commands sometime before
hint: your next pull:
hint:
hint:   git config pull.rebase false  # merge
hint:   git config pull.rebase true   # rebase
hint:   git config pull.ff only       # fast-forward only
hint:
hint: You can replace "git config" with "git config --global" to set a default
hint: preference for all repositories. You can also pass --rebase, --no-rebase,
hint: or --ff-only on the command line to override the configured default per
hint: invocation.
fatal: Need to specify how to reconcile divergent branches.
```

I



Accepted both changes then followed three stage architecture to commit this change

```
aaaaakriti@Aakritis-MacBook-Air morningclassAaakriti % git add Readme.md
aaaaakriti@Aakritis-MacBook-Air morningclassAaakriti % git conflict -m "resolved a merge of conflict"
```

Now push the changes it will be successful

Now both can view commit history. And collaborator can pull new conflict free repo.

Workflows to avoid merge conflicts

Some basic rules of thumb can avoid the vast majority of merge conflicts, saving a lot of time and frustration. These are words our teams live by:

- Communicate often
- Tell each other what you are working on • Pull immediately before you commit or push
- Commit often in small chunks.

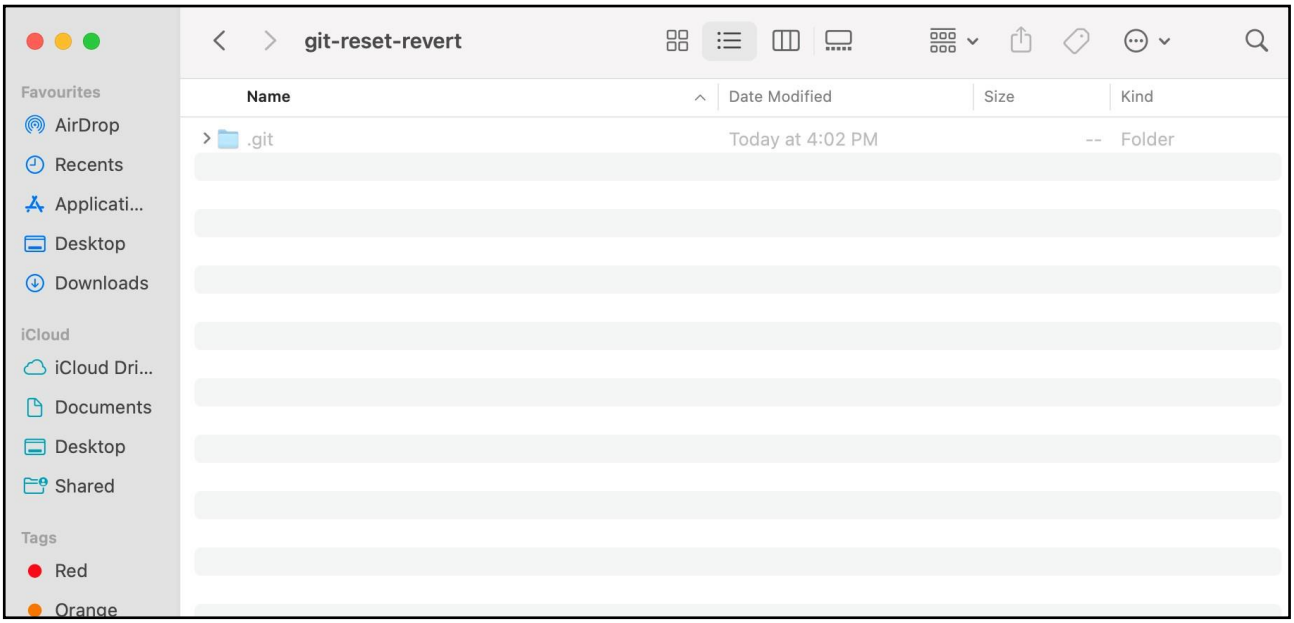
A good workflow is encapsulated as follows:

```
Pull -> Edit -> Add -> Pull -> Commit -> Push
```

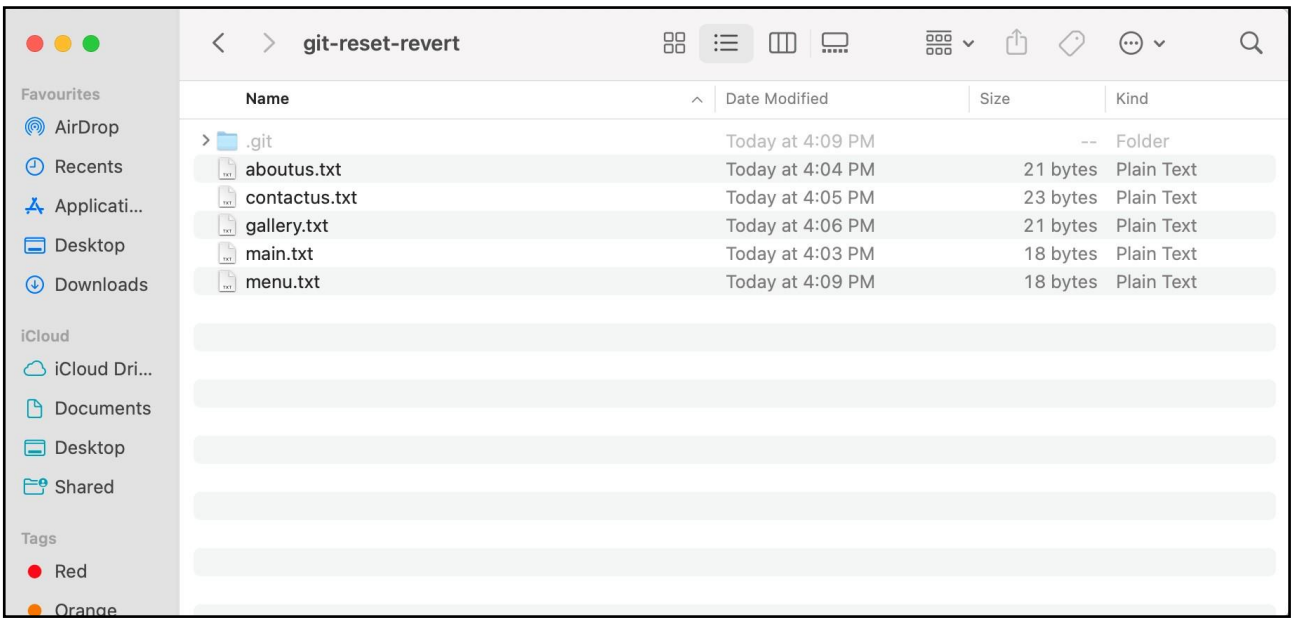
Always start your working sessions with a pull to get any outstanding changes, then start doing your editing and work. Stage your changes, but before you commit, Pull again to see if any new changes have arrived. If so, they should merge in easily if you are working in different parts of the program. You can then Commit and immediately Push your changes safely. Good luck, and try to not get frustrated. Once you figure out how to handle merge conflicts, they can be avoided or dispatched when they occur, but it does take a bit of practice.

Aim: Reset and Revert

First of all ,We will make a folder named git-reset-revert .Under this folder we initialise an empty repository using git init.



Add Some files to it and do some changes and commit it(following three step architecture .



In this we have added files and done commits .

```
git-reset-revert — -zsh — 111×24
Last login: Thu Jun  2 00:53:44 on ttys001
aaaaakriti@Aakritis-MacBook-Air git-reset-revert % git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /Users/aaaaakriti/Desktop/git-reset-revert/.git/
aaaaakriti@Aakritis-MacBook-Air git-reset-revert % echo this is main page >main.txt
```

Run Git log - - online command to get history of commit :

```
aaaaakriti@Aakritis-MacBook-Air git-reset-revert % git log
commit 9f06b55ed3632bc37e0acd0fbc0c1a45a41bc79e (HEAD -> master)
Author: Aaaakriti <aakriti0008.be21@chitkara.edu.in>
Date:   Thu Jun 2 01:05:57 2022 +0530
```

-

Now According to git log Head(-> master) is currently at latest (f3ddbbe) commit .

Git Reset:

Let's start with the Git command 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.

What happens if we want to roll back to a previous commit. Simple —we can just move the branch pointer. Git supplies the reset command to do this for us. For example, if we want to reset *master* to point to the commit three back from the current commit, we could use either of the following methods:

$ git reset 6b5b9c4 - - hard (using an absolute commit SHA1 value 6b5b9c4)

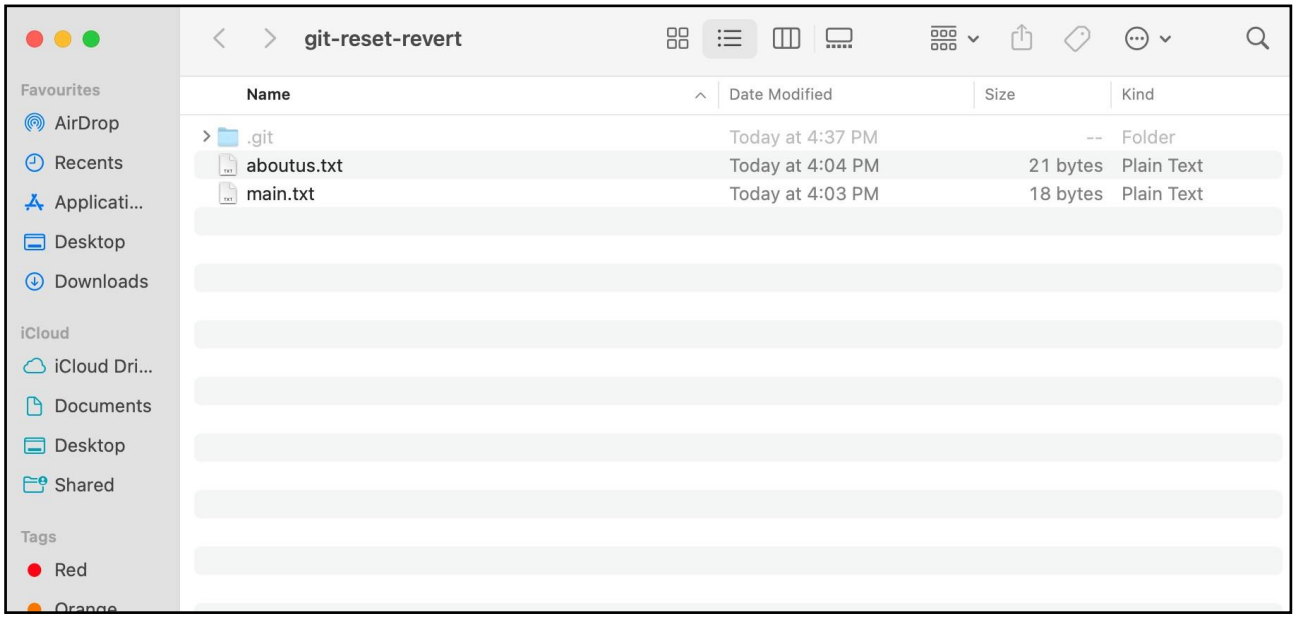
- - Hard will be discussed in below pages

or

$ git reset - - hard Head~3 (using a relative value ~3 before the “current”/“Head” tag)

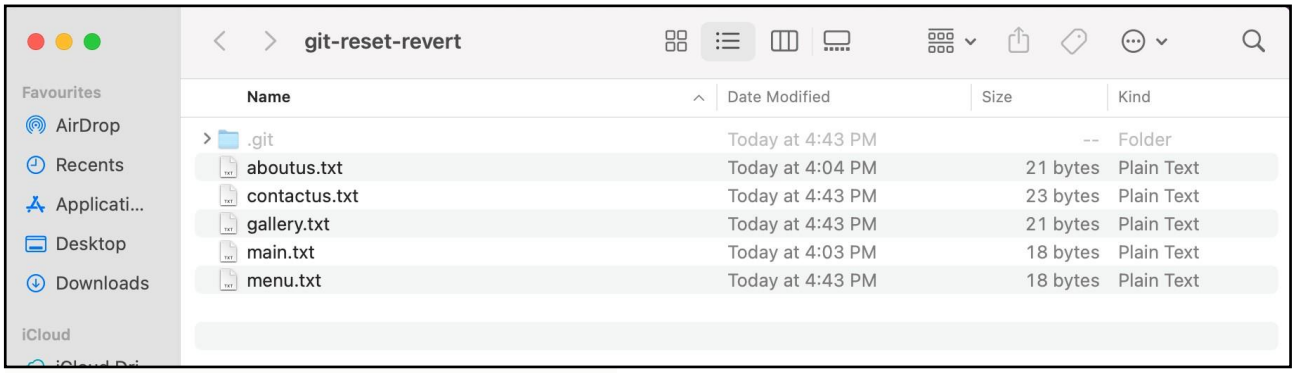
Below shows the results of this operation. After this, if we execute a git log command on the current branch (*master*), we'll see just two commit.

If we see in folder the three files will be deleted



If we Wish to go back again to Previous state with 5 files we will use Git reset f33ddbbe - - hard and now run git log command again and check the folder the files are back

```
HEAD is now at f3ddbbe Added menu page to the site
aaaaakriti@Aakritis-MacBook-Air git-reset-revert % git log --oneline
f3ddbbe (HEAD -> master) Added menu page to the site
7ebfbd0 Added gallery page to the site
312c3fc Added contactus page to the site
6b5b9c4 Added aboutus page to the site
1ab6339 Added main page to the site
```



***git reset* [<mode>] [<commit>]**

This form resets the current branch head to <commit> and possibly updates the index (resetting it to the tree of <commit>) and the working tree depending on <mode>. If <mode> is omitted, defaults to --mixed. The <mode> must be one of the following:

--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.

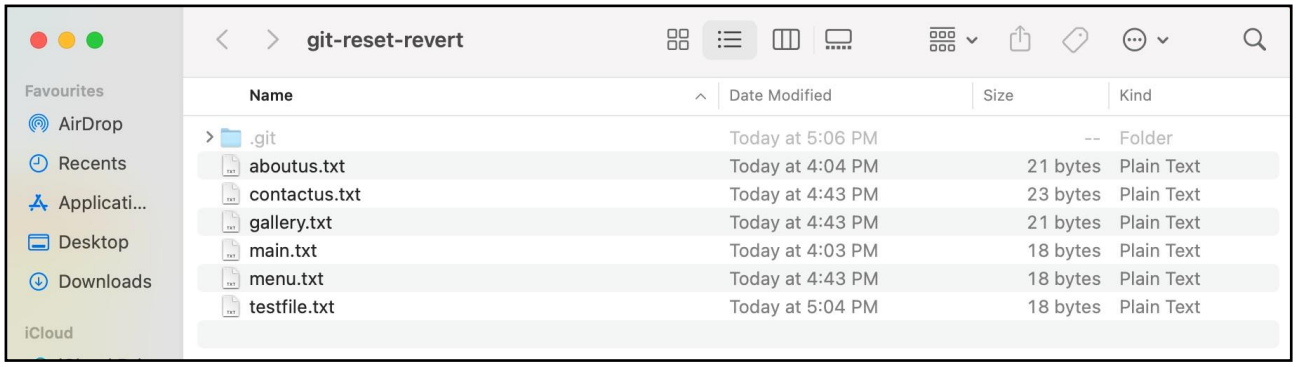
For e.g:

We add a file named test file.txt and add some content to it and take it to staging area and then commit it.

Now we run git log command again and suppose we don't want this file to staged we will use git reset Head - -mixed and then this file is removed from staging area but not from working directory.

```
aaaaakriti@Aakritis-MacBook-Air git-reset-revert % echo test1 test2 test3 >testfile.txt
aaaaakriti@Aakritis-MacBook-Air git-reset-revert % add textfile.txt
zsh: command not found: add
aaaaakriti@Aakritis-MacBook-Air git-reset-revert % git add testfile.txt
aaaaakriti@Aakritis-MacBook-Air git-reset-revert % git commit -m "added test page to the file"
[master 7796900] added test page to the file
 1 file changed, 1 insertion(+)
 create mode 100644 testfile.txt
```

We can see that test file is present in working directory.



Now if we use git reset head~1 - -soft then the latest commit will rollback to staging area(changes to be committed section)

--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.

For e.g:

We add a file named test file.txt and add some content to it and take it to staging area

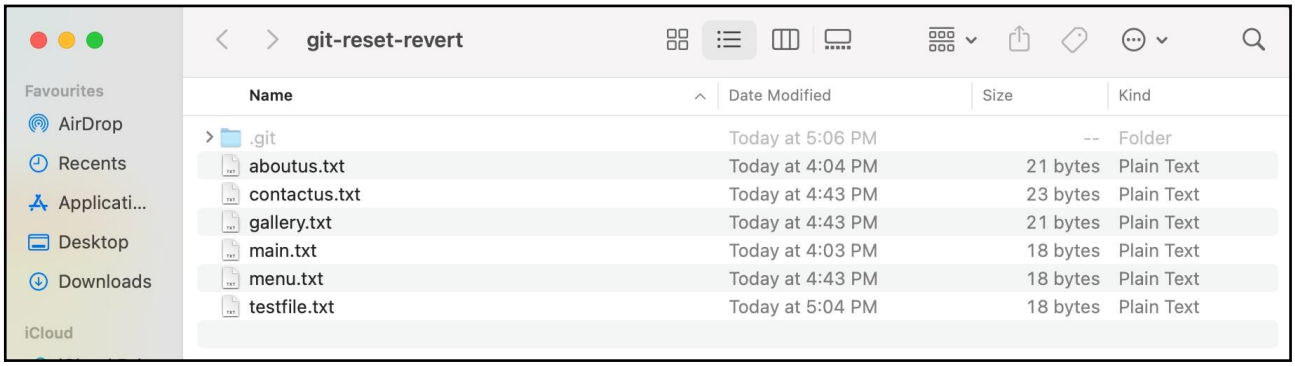
```
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   testfile.txt
```

```
aaaaakriti@Aakritis-MacBook-Air git-reset-revert % git reset head --mixed
aaaaakriti@Aakritis-MacBook-Air git-reset-revert % git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        testfile.txt

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

Now we run git log command again and suppose we don't want this file to staged we will use git reset Head - -mixed and then this file is removed from staging area but not from working directory.

We can see that test file is present in working directory.



Now if you want you can commit this by staging it and then committing it.

--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.

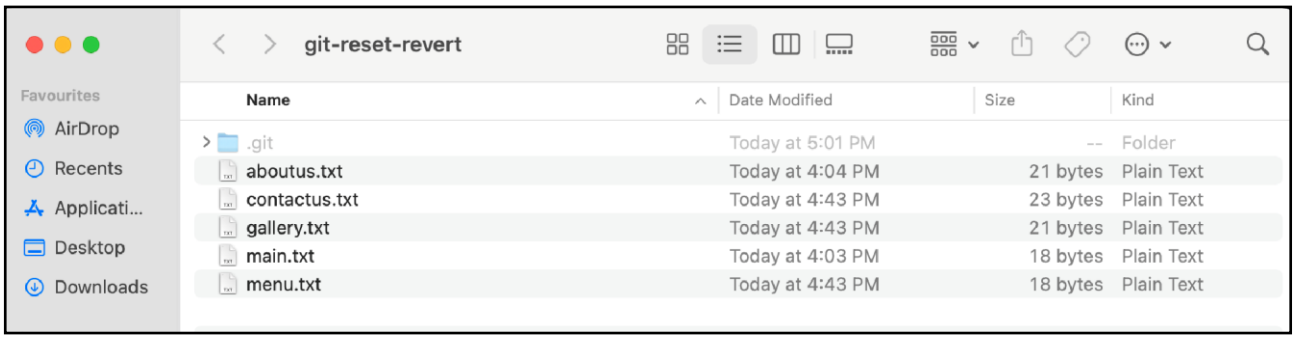
For e.g:

We add a file named test file.txt and add some content to it and take it to staging area

```
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   testfile.txt
```

Now we run git log command again and suppose we don't want this file we will use git reset Head - -hard and then this file is removed from staging as well as from our working directory.

```
aaaaakriti@Aakritis-MacBook-Air git-reset-revert % git reset head --hard
HEAD is now at 20d95ad this
aaaaakriti@Aakritis-MacBook-Air git-reset-revert % git status
On branch master
nothing to commit, working tree clean
aaaaakriti@Aakritis-MacBook-Air git-reset-revert %
```



Git Revert :

The git revert command is used for undoing changes to a repository's commit history. Other 'undo' commands like, git checkout and git reset, move the HEAD and branch ref pointers to a specified commit. Git revert also takes a specified commit, however, git revert does not move ref pointers to this commit. A revert operation will take the specified commit, inverse the changes from that commit, and create a new "revert commit". The ref pointers are then updated to point at the new revert commit making it the tip of the branch.

Below is git log of the commit we have done so far.

```
aaaaakriti@Aakritis-MacBook-Air git-reset-revert % git log --oneline
20d95ad (HEAD -> master) this
aaa9ef2 this is to be committed
7796900 added test page to the file
9f06b55 this is to be committed
aaaaakriti@Aakritis-MacBook-Air git-reset-revert %
```

Now suppose we revert 312c3fc commit by using git revert 312c3fc(opening editor window close it to proceed) :

```
aaaaakriti@Aakritis-MacBook-Air git-reset-revert % git revert 312c3fc
[master 1fdcd87] Revert "Added contactus page to the site"
 1 file changed, 1 deletion(-)
 delete mode 100644 contactus.txt
```

```
1   Revert "Added contactus page to the site"
2
3   This reverts commit 312c3fcf7fff94ec3f22970b397887d26bbb10b4.
4
5   # Please enter the commit message for your changes. Lines starting
6   # with '#' will be ignored, and an empty message aborts the commit.
7   #
8   # On branch master
9   # Changes to be committed:
10  #   deleted:    contactus.txt
```

Now do some changes in test file.txt and main.txt and commit it .

```
Last login: Thu Jun  2 01:20:39 on ttys002
aaaaakriti@Aakritis-MacBook-Air git-reset-revert % echo test4 >>testfile.t
xt
aaaaakriti@Aakritis-MacBook-Air git-reset-revert % git add testfile.txt
aaaaakriti@Aakritis-MacBook-Air git-reset-revert % git commit -m "added in
formation to test page"
[master 4786964] added information to test page
 1 file changed, 1 insertion(+)
aaaaakriti@Aakritis-MacBook-Air git-reset-revert % echo test4 >>main.txt
aaaaakriti@Aakritis-MacBook-Air git-reset-revert % git add main.txt
aaaaakriti@Aakritis-MacBook-Air git-reset-revert % git commit -m "Added in
firmation to main page"
[master f678a36] Added infirmation to main page
 1 file changed, 1 insertion(+)
aaaaakriti@Aakritis-MacBook-Air git-reset-revert %
```

Use git log again and suppose we don't want changes in testfile but want changes in main file so we use git revert head~1:

```
aaaaakriti@Aakritis-MacBook-Air git-reset-revert % git revert head~1
[master b3cdc82] Revert "Added information to test page"
 1 file changed, 1 deletion(-)
aaaaakriti@Aakritis-MacBook-Air git-reset-revert % git log --oneline
b3cdc82 (HEAD -> master) Revert "Added information to test page"
586474e Added information to main page
6639261 Added information to test page
1fdcd87 Revert "Added contactus page to the site"
7b4a4c1 Added test page to the site
f3ddbbe Added menu page to the site
7ebfbd0 Added gallery page to the site
312c3fc Added contactus page to the site
6b5b9c4 Added aboutus page to the site
1ab6339 Added main page to the site
```

As you can see changes in main.txt are still present beside the fact we revert the commit previous than this.

Task 2

SCM PROJECT

Subject Name: **Source Code Management**
Subject Code: **CS181**
Cluster: **Beta**
Department: **CSE**

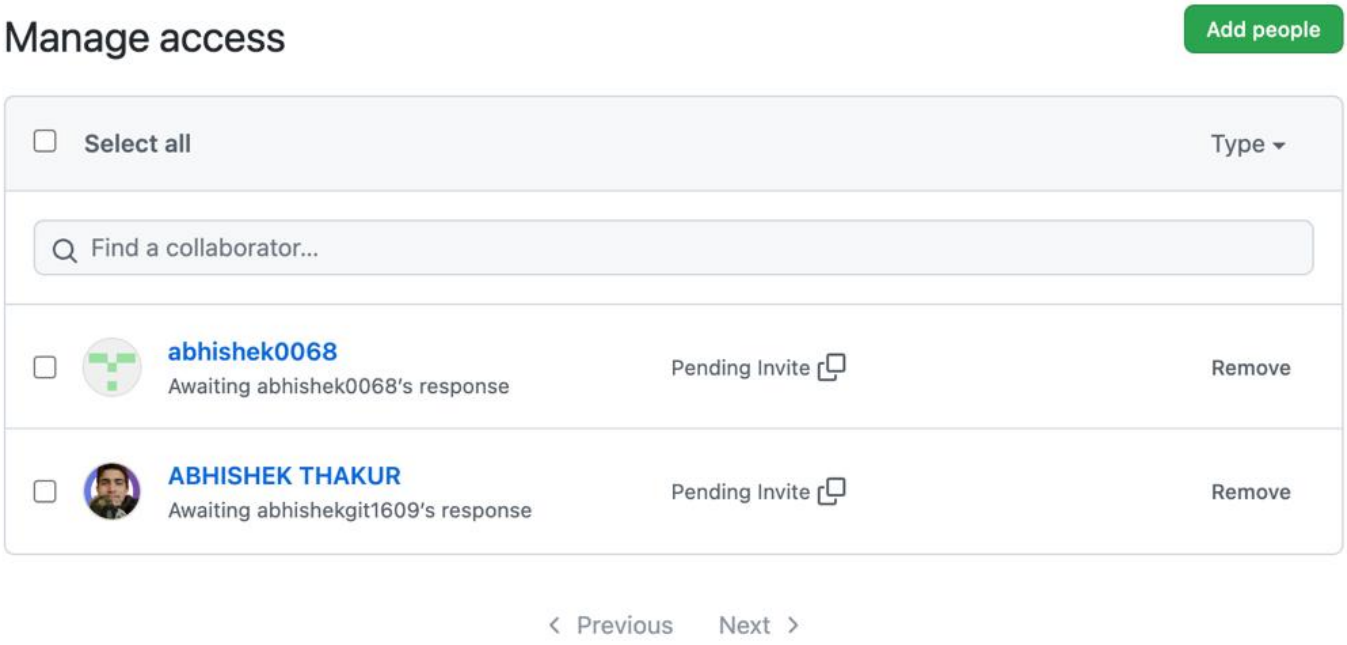
Made By: Aakriti Bahl
Roll No: 2110990008
Submitted To: Dr. Monit Kapoor
Team No: 12

INDEX

TO DO: Create a distributed repository and add team members.

STEPS:

1. We created a repository in the organization by the naming convention **<Roll No>.**
2. Then from the **collaborator** section of the **settings** panel I added my other teammates as collaborators in the repository.

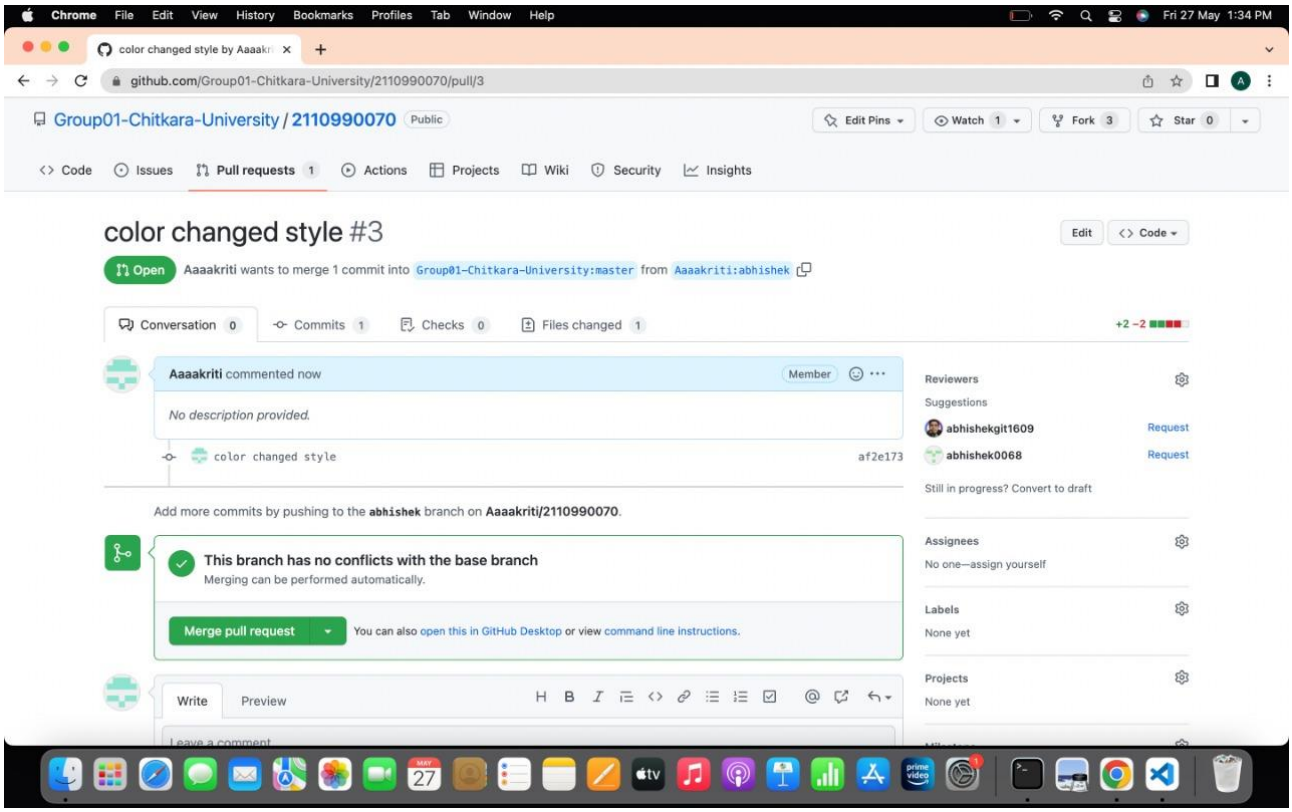


3. The distributed repository is now created, but if the members were added as collaborators then we could not generate any pull request except the ones which involved branches.
4. To avoid this issue I removed them as a collaborator. We then decided to **fork and commit** to each other's repo.
5. We then forked each other's repo and started working on the given code.
6. In my repository the code is in **Python** language and is a basic **ATM Project**. My teammates started working on the code by creating their own branches and then pushing and testing the code.

TO DO: Open and Close Pull Requests.

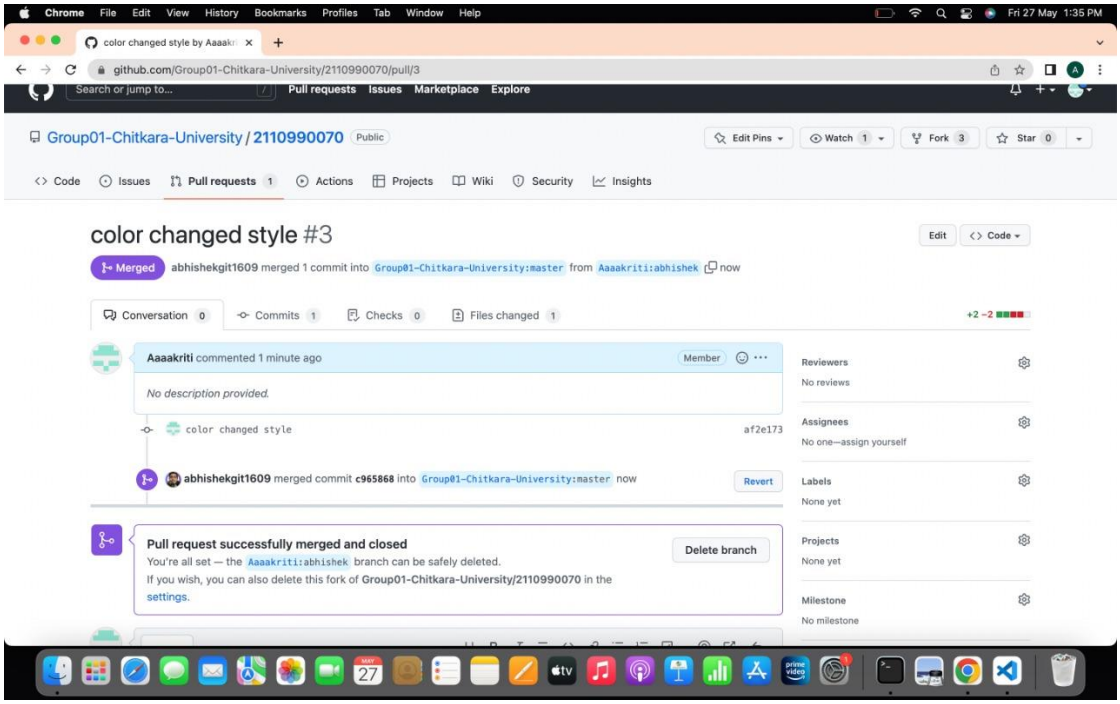
STEPS:

1. My teammates started pushing code in their own branches and then opened pull requests in my repository by **Contributing** to my code.

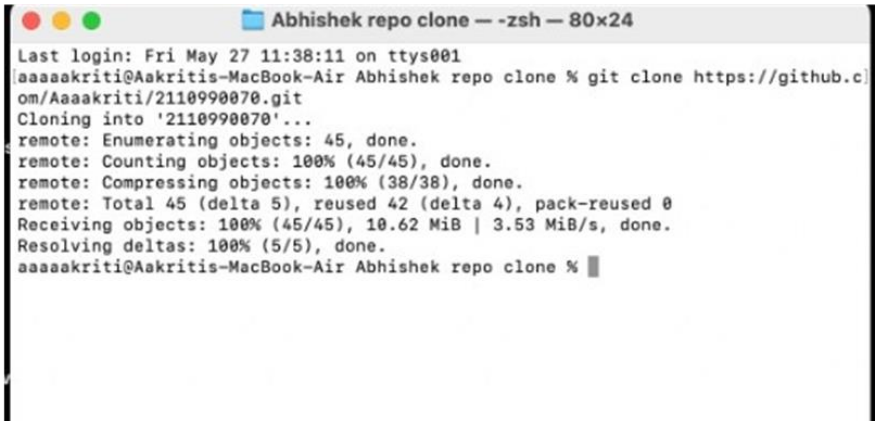


2. I closed them simultaneously as a maintainer of the repo.
3. As a member of the team and the requirements of the project work I also had to open and close pull requests in my teammates repositories. Below are the screenshots of the same.

   1. Contribution in **Abhishekgit1609**

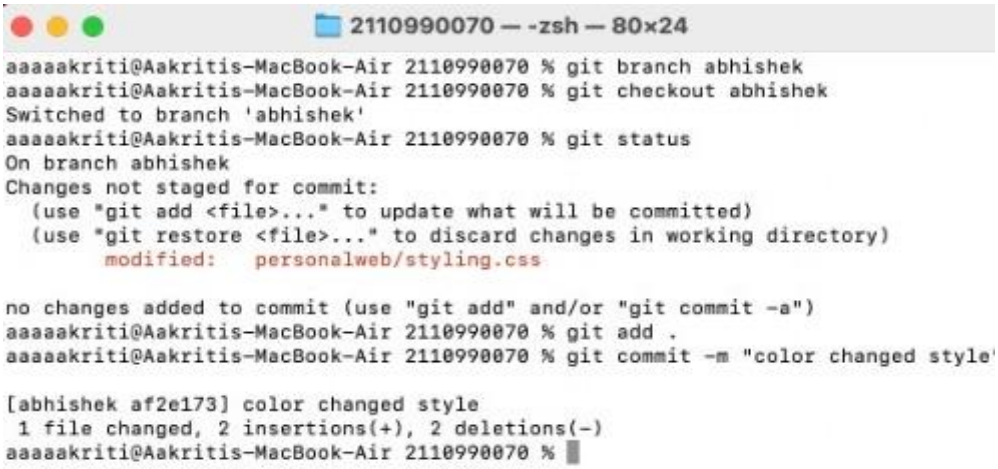


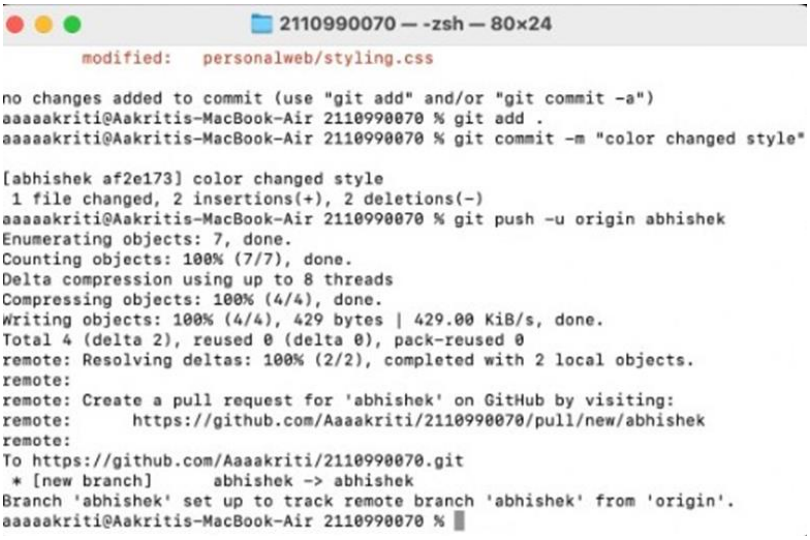
4. I worked with **abhishekgit1609, Abhijeet0003,Abhishek** on their repositories and modified their code, added new functionalities and tested their code.
5. To achieve this first I had to fork the repo into my account and then I cloned them onto my local machine.



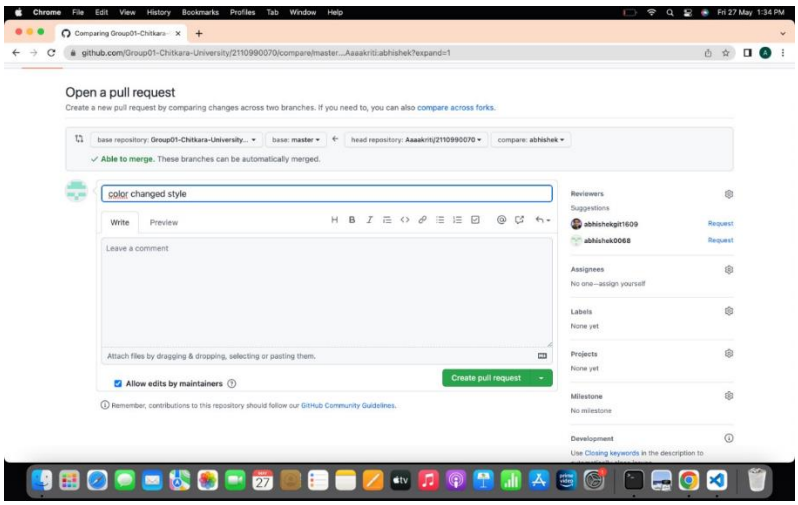
6. After cloning I reviewed the code, made changes on my own branch and then committed to the forked repo. From where I then open a pull request and it's

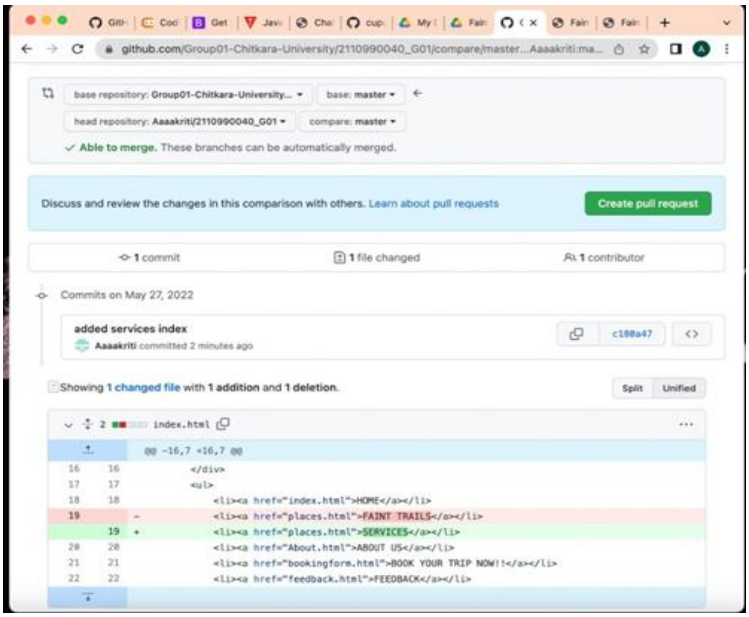
reflected on the repo owner's pull requests tab. [Screenshots attached for the same]. The changes made by me are also visible to the owner.

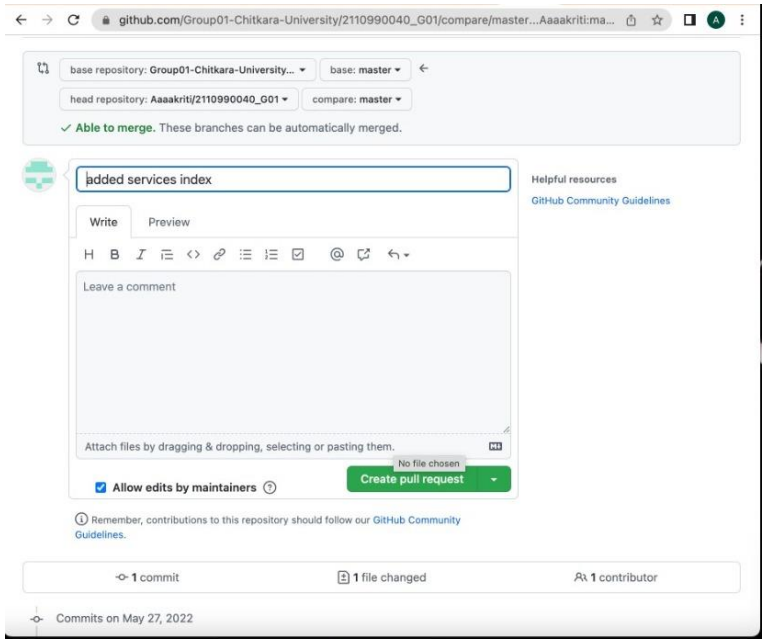


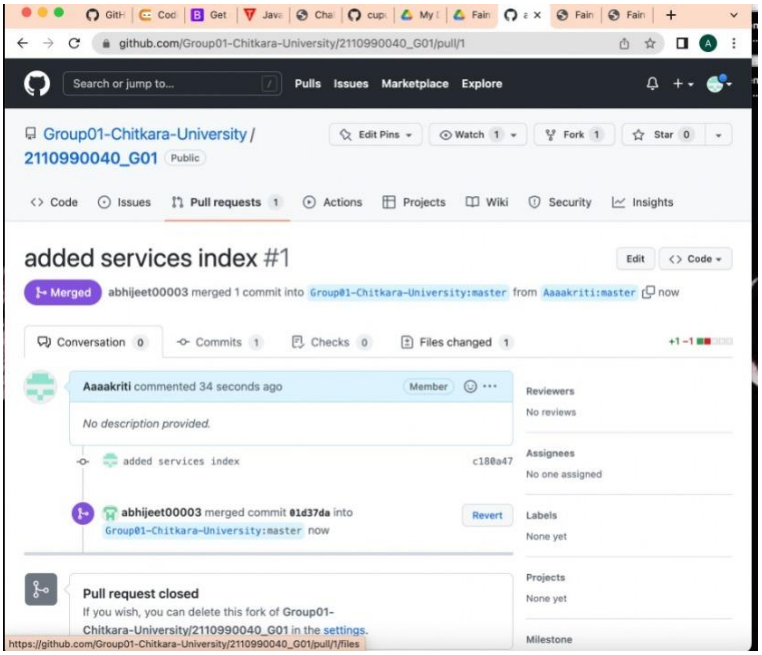


```
2110990070 — -zsh — 80×24
aaaaakriti@Aakritis-MacBook-Air 2110990070 % git push -u origin abhishek
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 429 bytes | 429.00 KiB/s, done.
Total 4 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
remote:
remote: Create a pull request for 'abhishek' on GitHub by visiting:
remote:      https://github.com/Aaaakriti/2110990070/pull/new/abhishek
remote:
To https://github.com/Aaaakriti/2110990070.git
 * [new branch]      abhishek -> abhishek
Branch 'abhishek' set up to track remote branch 'abhishek' from 'origin'.
aaaaakriti@Aakritis-MacBook-Air 2110990070 % git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.
aaaaakriti@Aakritis-MacBook-Air 2110990070 % git merge abhishek
Updating dce7bcf..af2e173
Fast-forward
 personalweb/styling.css | 4 ++--
 1 file changed, 2 insertions(+), 2 deletions(-)
aaaaakriti@Aakritis-MacBook-Air 2110990070 %
```





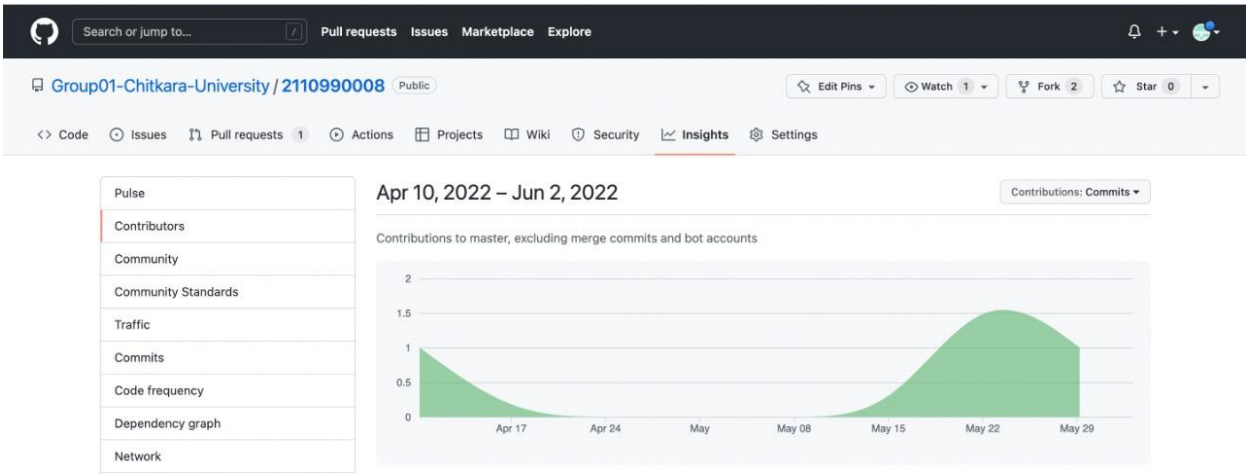




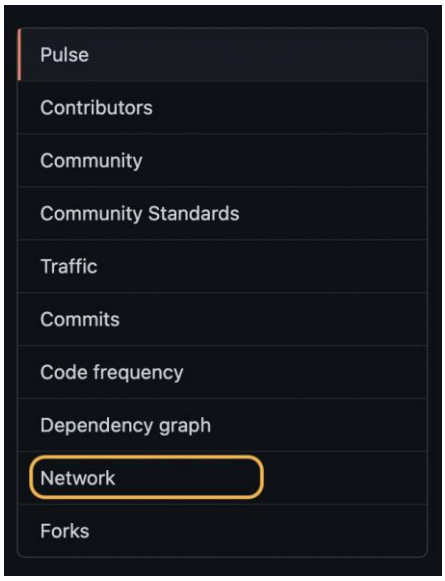
TO DO: Publish and Print Network Graphs

STEPS:

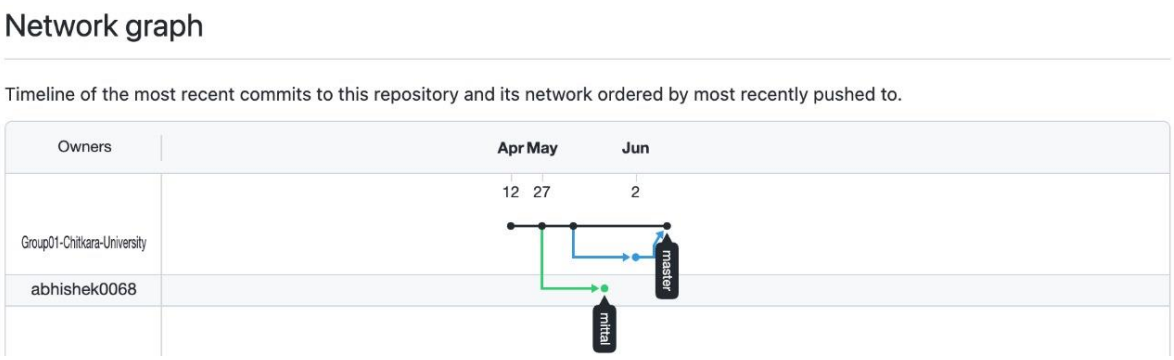
1. Go to the **Insights** tab of the Repository.



2. Click on the **Network** panel in the navigation panel.



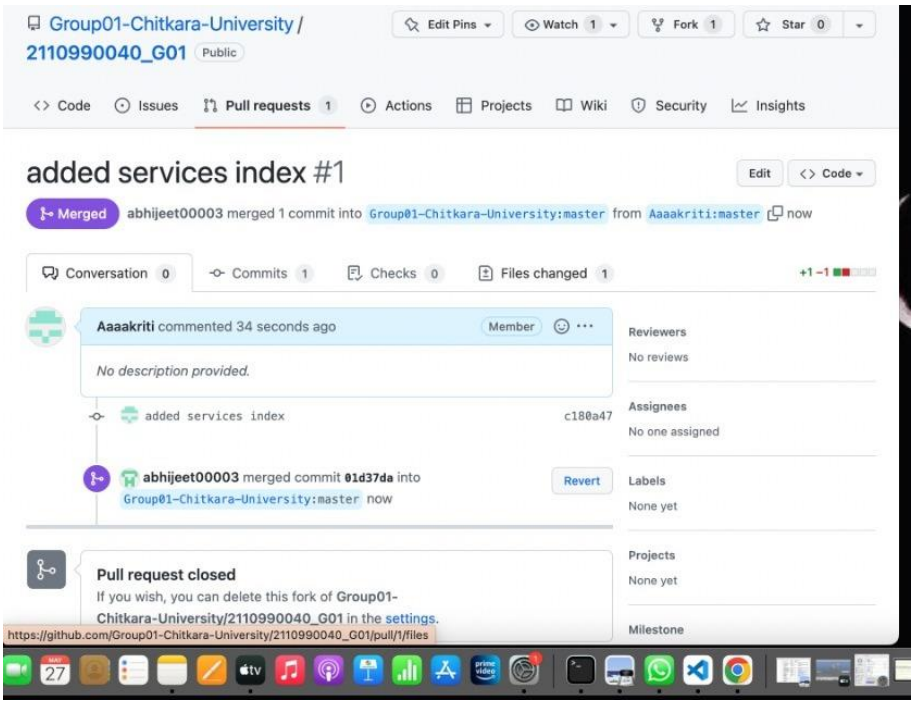
3. We can now visualize the network graph of our repository. Mine is attached below.



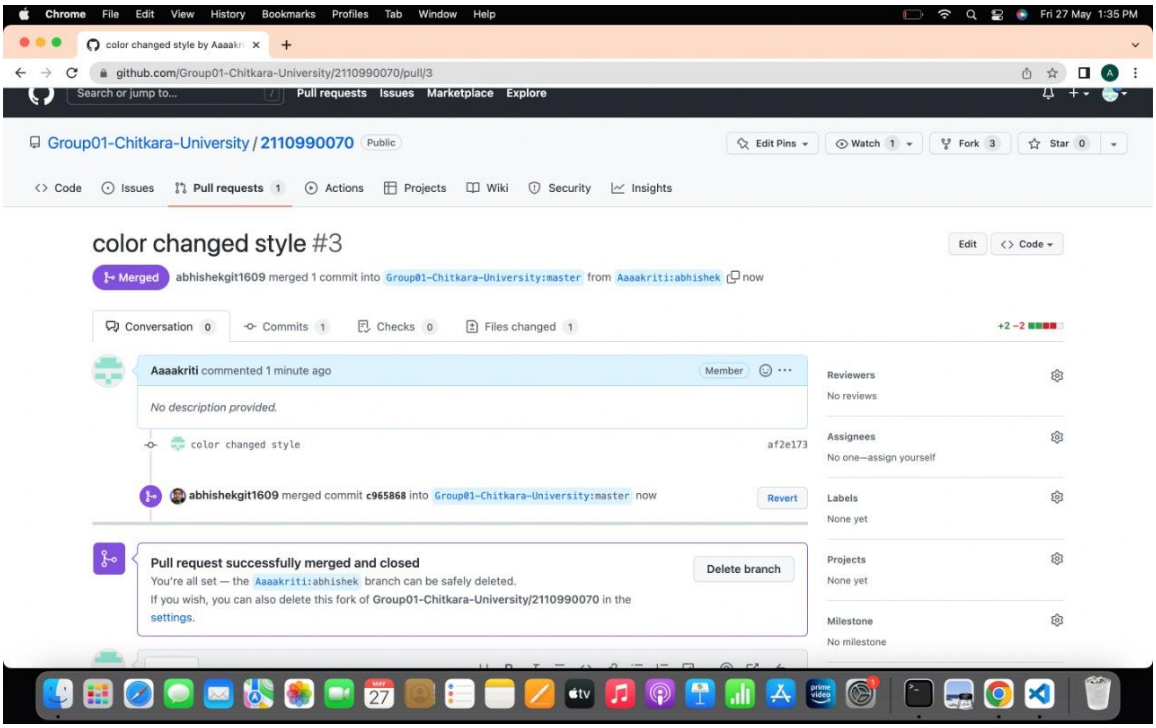
4. Through the network graph we can visualize the creating and merging of all the branches from the master branch and into the master branch. This network graph is very simple, they can be quite complex in bigger repositories.

TODO: Pull request from each member of the team.

- **Member #1**


Group01-Chitkara-University / 2110990040_G01 Public
Edit Pins
Watch 1
Fork 1
Star 0
Code
Issues
Pull requests 1
Actions
Projects
Wiki
Security
Insights
added services index #1
Edit
Code
Merged
abhijeet00003 merged 1 commit into Group01-Chitkara-University:master from Aaaakriti:master now
Conversation 0
Commits 1
Checks 0
Files changed 1
+1 −1
Aaaakriti commented 34 seconds ago
Member
No description provided.
added services index
c180a47
abhijeet00003 merged commit 01d37da into Group01-Chitkara-University:master now
Revert
Pull request closed
If you wish, you can delete this fork of Group01-Chitkara-University/2110990040_G01 in the settings.
Reviewers
No reviews
Assignees
No one assigned
Labels
None yet
Projects
None yet
Milestone
https://github.com/Group01-Chitkara-University/2110990040_G01/pull/1/files

- **Member #2**


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Fri 27 May 1:35 PM
color changed style by Aaaakri
github.com/Group01-Chitkara-University/2110990070/pull/3
Search or jump to...
Pull requests Issues Marketplace Explore
Group01-Chitkara-University / 2110990070 Public
Edit Pins
Watch 1
Fork 3
Star 0
Code
Issues
Pull requests 1
Actions
Projects
Wiki
Security
Insights
color changed style #3
Edit
Code
Merged
abhishekgit1609 merged 1 commit into Group01-Chitkara-University:master from Aaaakriti:abhishek now
Conversation 0
Commits 1
Checks 0
Files changed 1
+2 −2
Aaaakriti commented 1 minute ago
Member
No description provided.
color changed style
af2e173
abhishekgit1609 merged commit c965868 into Group01-Chitkara-University:master now
Revert
Pull request successfully merged and closed
You're all set — the Aaaakriti:abhishek branch can be safely deleted.
If you wish, you can also delete this fork of Group01-Chitkara-University/2110990070 in the settings.
Delete branch
Reviewers
No reviews
Assignees
No one—assign yourself
Labels
None yet
Projects
None yet
Milestone
No milestone

- **Screenshots of the total pull requests and contributors in my repository.**

000