A Project report

with

Source Code Management

(CS181)

Submitted by

BY-ANUSHKA MALHOTRA

ROLL NO.2110990242

Group-8-A

Department of Computer Science & Engineering

Chitkara University Institute of Engineering and Technology

Jan- June
(2021-2022)

Institute/School Name	**Chitkara University Institute of Engineering and Technology**		
Department Name	**Department of Computer Science & Engineering**		
Programme Name	**Bachelor of Engineering (B.E.), Computer Science & Engineering**		
Course Name	**Source Code Management**	Session	**2021-22**
Course Code	**CS181**	Semester/Batch	**2nd/2021**
Vertical Name	**Beta**	Group No	G08-A
Course Coordinator	**Dr. Neeraj Singla**		
Faculty Name	**Dr. Monit Kapoor**		

CHITKARA UNIVERSITY

INDEX

What is GIT and why is it used?

Git is a version control system that is widely used in the programming world. It is used for tracking changes in the source code during software development. It was developed in 2005 by Linus Torvalds, the creator of the Linux operating system kernel.

Git is a speedy and efficient distributed VCS tool that can handle projects of any size, from small to very large ones. Git provides cheap local branching, convenient staging areas, and multiple workflows. It is free, open-source software that lowers the cost because developers can use Git without paying money. It provides support for non-linear development. Git enables multiple developers or teams to work separately without having an impact on the work of others.

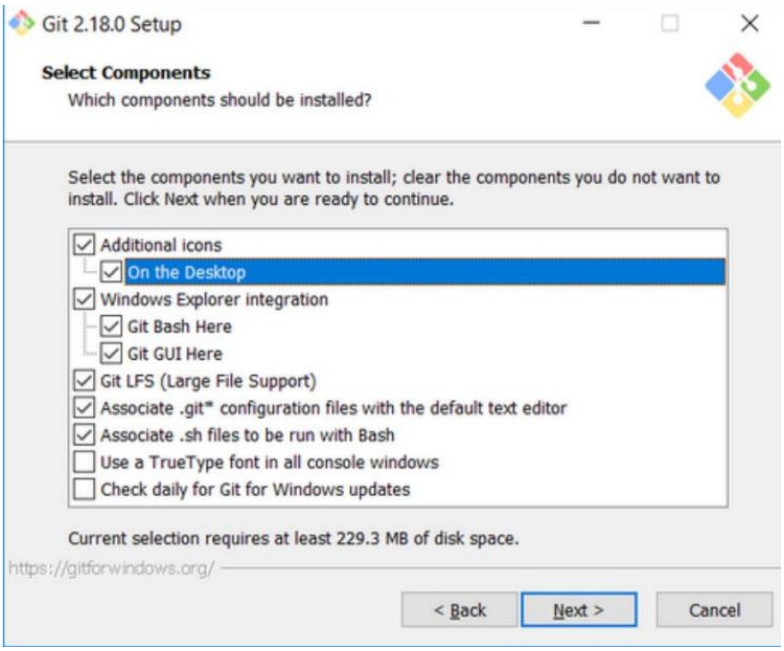
Git is an example of a distributed version control system (DVCS) (hence Distributed Version Control System).

What is GITHUB?

It is the world's largest open-source software developer community platform where the users upload their projects using the software Git.

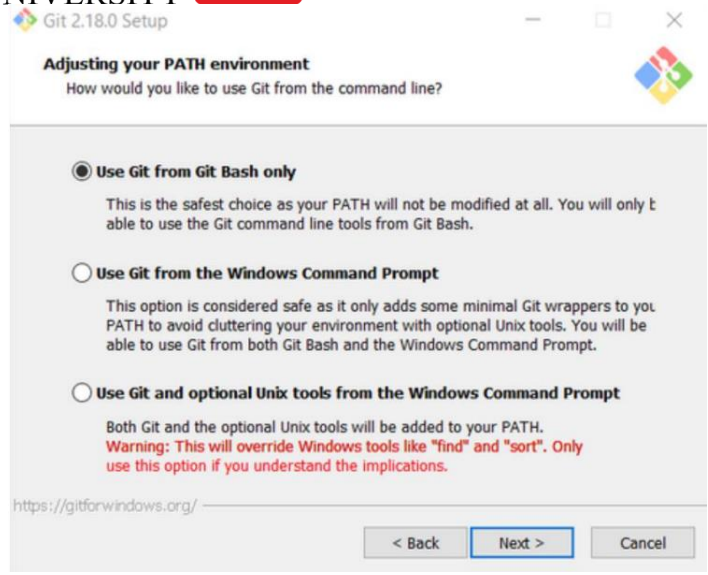
EXPERIMENT 1

Aim: Setting up the git client. Git Installation: Download the Git installation program (Windows, Mac, or Linux) from Git - Downloads (git-scm.com). When running the installer, various screens appear (Windows screens shown). Generally, you can accept the default selections, except in the screens below where you do not want the default selections: In the Select Components screen, make sure Windows Explorer Integration is selected as shown

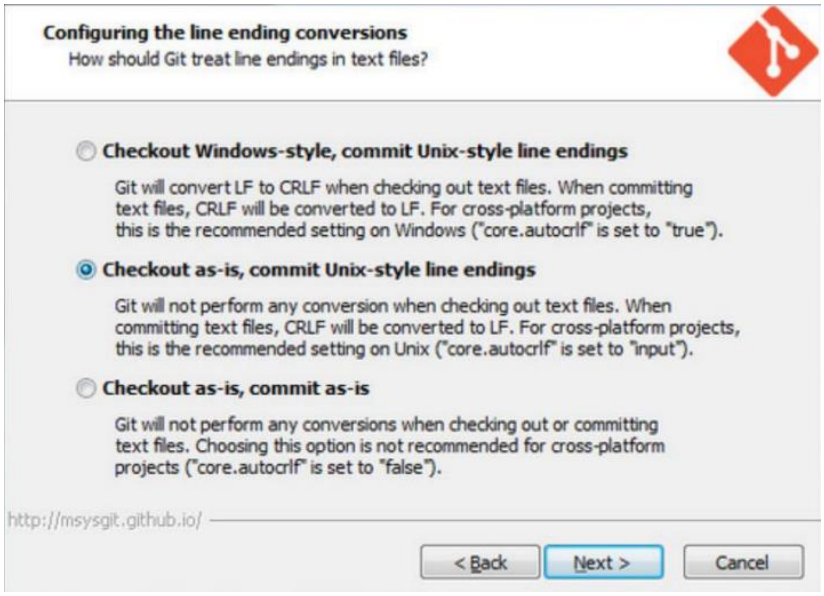


In the choosing the default editor is used by Git dialog, it is strongly recommended that you DO NOT select default VIM editor- it is challenging to learn how to use it, and there are better modern editors available. Instead, choose Notepad++ or Nano – either of those is much easier to use. It is strongly recommended that you select Notepad++

In the Adjusting your PATH screen, all three options are acceptable: 1. Use Git from Git Bash only: no integration, and no extra command in your command path. 2. Use Git from the windows Command Prompt: add flexibility – you can simply run git from a windows command prompt, and is often the setting for people in industry – but this does add some extra commands. 3. Use Git and optional Unix tools from the Windows Command Prompt: this is also a robust choice and useful if you like to use Unix like commands like grep.



In the Configuring the line ending screen, select the middle option (Checkout-as-is, commit Unix-style line endings) as shown. This helps migrate files towards the Unix-style (LF) terminators that most modern IDE's and editors support. The Windows convention (CR-LF line termination) is only important for Notepad.



Configuring Git to ignore certain files: This part is extra important and required so that your repository does not get cluttered with garbage files. By default, Git tracks all files in a project. Typically, this is not what you want; rather, you want Git to ignore certain files such as .bak files created by an editor or .class files created by the Java compiler. To have Git automatically ignore particular files, create a file named .gitignore (note that the filename begins with a dot) in the C:\users\name folder (where name is your MSOE login name). NOTE: The .gitignore file must NOT have any file extension (e.g. .txt). Windows normally tries to place a file extension (.txt) on a file you create from File Explorer - and then it (by default) HIDES the file extension. To avoid this, create the file from within a useful editor (e.g.Notepad++ or UltraEdit) and save the file without a file extension)

Note: You can always edit this file and add additional patterns for other types of files you might want to ignore. Note that you can also have a .gitignore files in any folder naming additional files to ignore. This is useful for projectspecific build products.

Once Git is installed, there is some remaining custom configuration you must do. Follow the steps below:

a. From within File Explorer, right-click on any folder. A context menu appears containing the commands " Git Bash here" and "GitGUI here". These commands permit you to launch either Git client. For now, select Git Bash here.

b. Enter the command (replacing name as appropriate) git config -- global core.excludesfile c:/users/name/. gitignore This tells Git to use the .gitignore file you created in step 2 NOTE: TO avoid typing errors, copy and paste the commands shown here into the Git Bash window, using the arrow keys to edit the red text to match your information.

c. Enter the command git config --global user.Email "name@msoe.edu" This links your Git activity to your email address. Without this, your commits will often show up as "unknown login". Replace name with your own MSOE email name.

d Enter the command git config --global user.name "Your Name" Git usesthisto log your activity. Replace "Your Name" by your actual first and last name.

e. Enter the command git config --global push.default simple This ensures that all pushes go back to the branch from which they were pulled. Otherwise pushes will go to the master branch, forcing a merge.

EXPERIMENT – 2

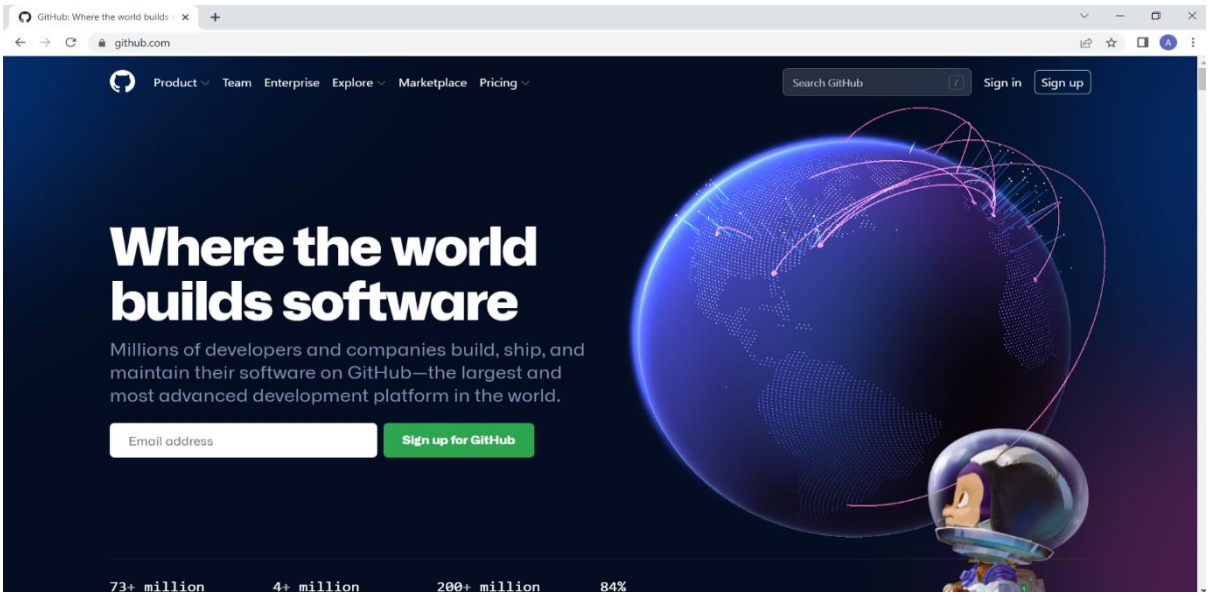
Aim

Setting up GitHub Account The first steps in starting with GitHub are to create an account, choose a product that fits your needs best, verify your email, set up two-factor authentication, and view your profile.

There are several types of accounts on GitHub. Every person who uses GitHub has their own user account, which can be part of multiple organisations and teams.

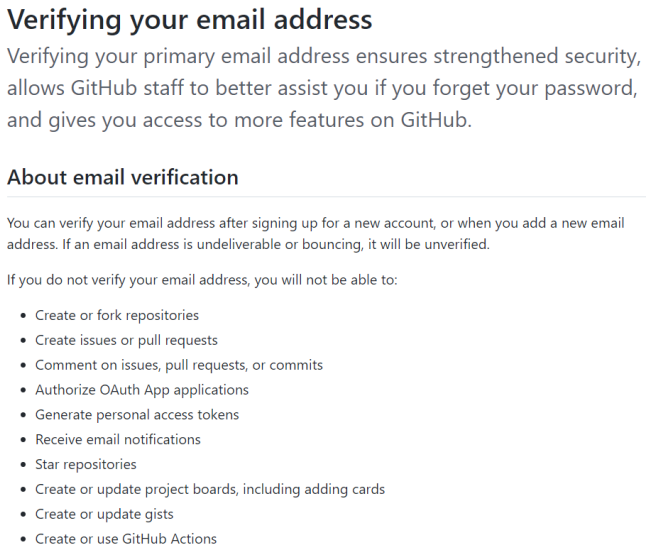
Your user account is your identity on GitHub.com and represents you as an individual.

1. Creating an account: To sign up for an account on GitHub.com, navigate to https://github.com/ and follow the prompts. To keep your GitHub account secure you should use a strong and unique password. For more information, see "Creating a strong password"

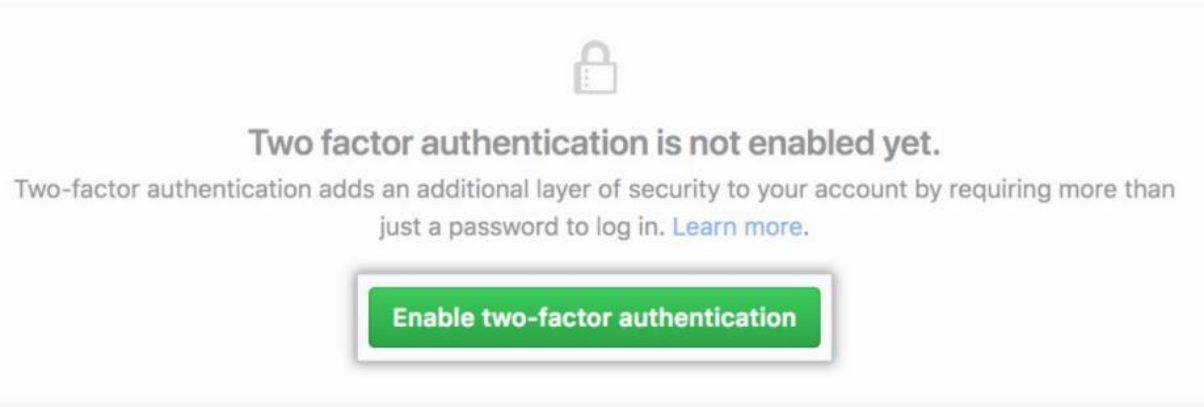


2. Choosing your GitHub product: You can choose GitHub Free or GitHub Pro to get access to different features for your personal account. You can upgrade at any time if you are unsure at first which product you want. For more information on all GitHub's plans, see "GitHub's products". Experiment No. 02 Page 8 of 16 CS181

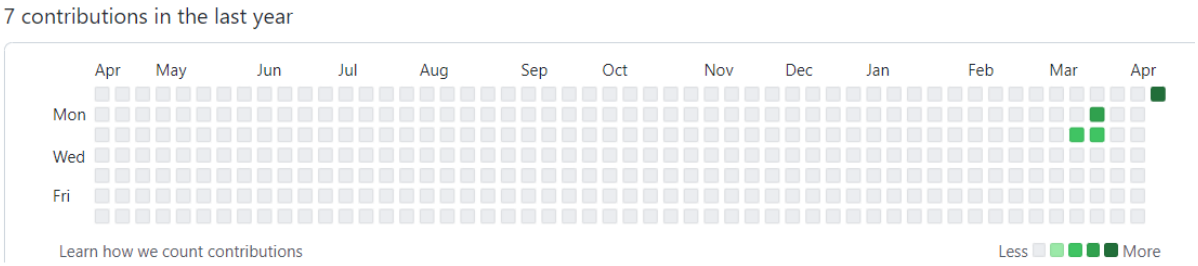
3. Verifying your email address: To ensure you can use all the features in your GitHub plan, verify your email address after signing up for a new account. For more information, see “Verifying your email address”.



4. Configuring two-factor authentication: Two-factor authentication, or 2FA, is an extra layer of security used when logging into websites or apps. We strongly urge you to configure 2FA for safety of your account. For more information, see “About two factor authentication."



5. Viewing your GitHub profile and contribution graph: Your GitHub profile tells people the story of your work through the repositories and gists you've pinned, the organisation memberships you've chosen to publicize, the contributions you've made, and the projects you've created. For more information, see "About your profile" and "Viewing contributions on your profile."



CHITKARA UNIVERSITY

EXPERIMENT 3

Aim: Program to generate logs

Basic Git init

Git init command creates a new Git repository. It can be used to convert an existing, undersigned project to a Git repository or initialize a new, empty repository. Most other Git commands are not available outside of an initialize repository, so this is usually the first command you'll run in a new project.

Basic Git status

The git status command displays the state of the working directory and the staging area. It lets you see which changes have been staged, which haven't, and which files aren't being tracked by Git. Status output does not show you any information regarding the committed project history.

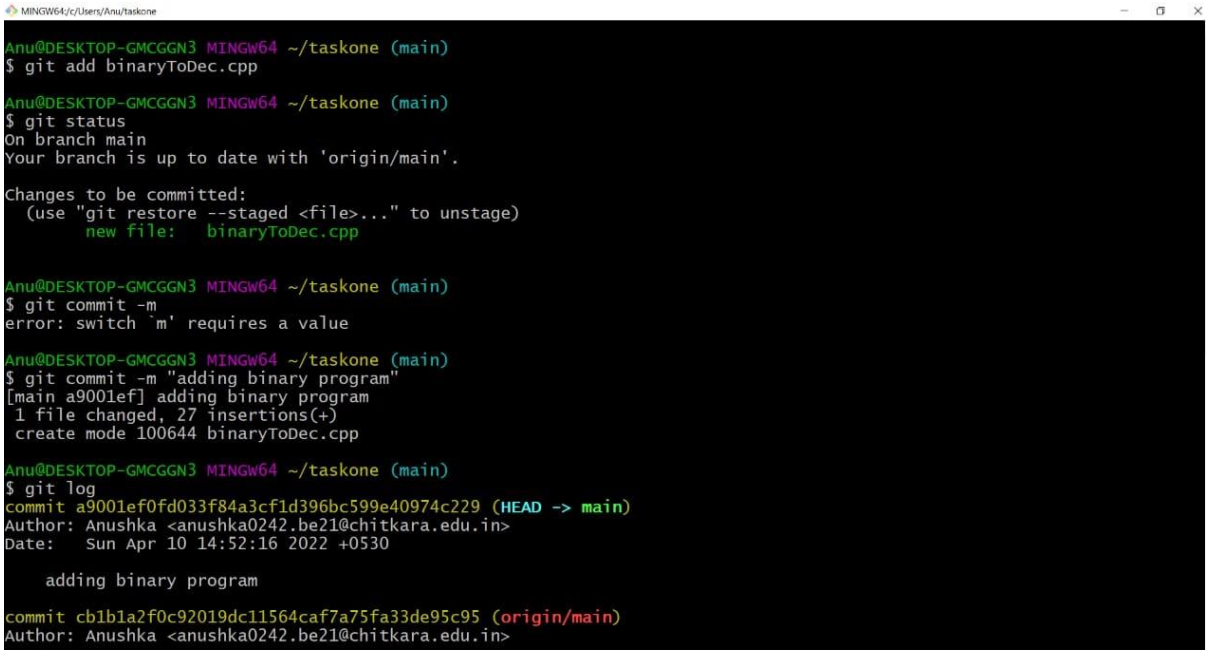
Basic Git commit

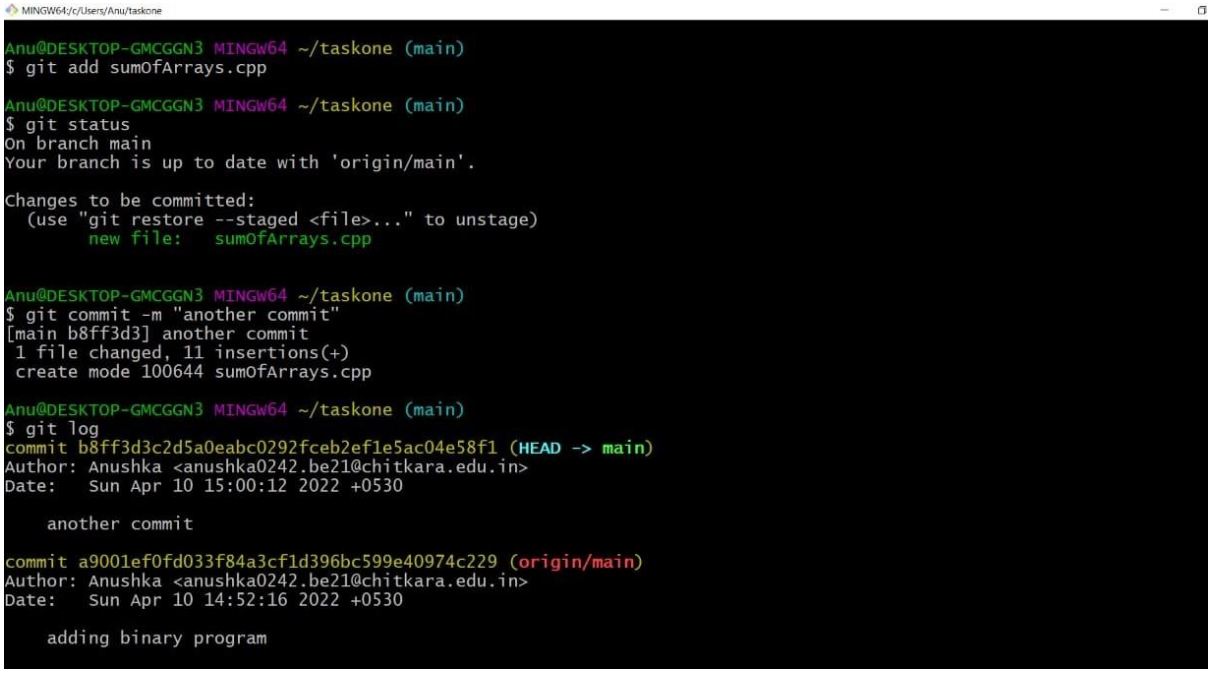
The git commit command captures a snapshot of the project's currently staged changes. Committed snapshots can be thought of as "safe" versions of a project—Git will never change them unless you explicitly ask it to. Prior to the execution of git commit, The git add command is used to promote or 'stage' changes to the project that will be stored in a commit. These two commands git commit and git add are two of the most frequently used

Basic Git add command

The git add command adds a change in the working directory to the staging area. It tells Git that you want to include updates to a particular file in the next commit. However, git add doesn't really affect the repository in any significant way—changes are not actually recorded until you run git commit.

Basic Git log Git log command is one of the most usual commands of git. It is the most useful command for Git. Every time you need to check the history, you have to use the git log command. The basic git log command will display the most recent commits and the status of the head. It will use as:





CHITKARA UNIVERSITY

EXPERIMENT 4

Aim: Create and visualize branches in Git

How to create branches?

The main branch in git is called master branch. But we can make branches out of this main master branch. All the files present in master can be shown in branch but the files which are created in branch are not shown in master branch. We also can merge both the parent(master) and child (other branches).

1. For creating a new branch: git branch “name of branch”

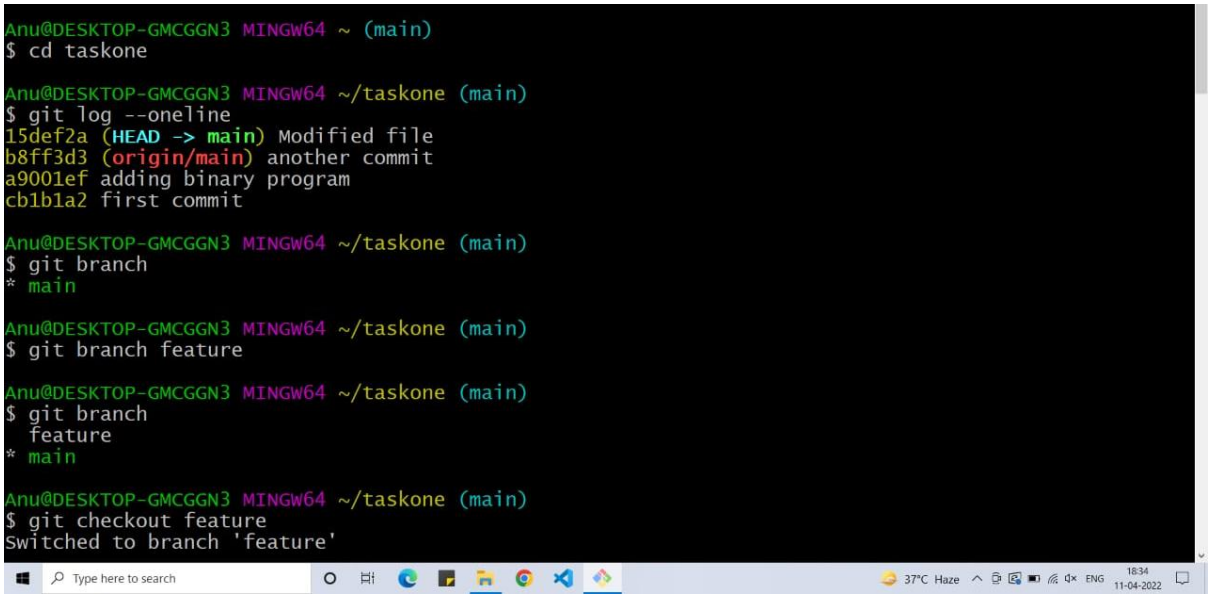
2. To check how many branches we have : git branch

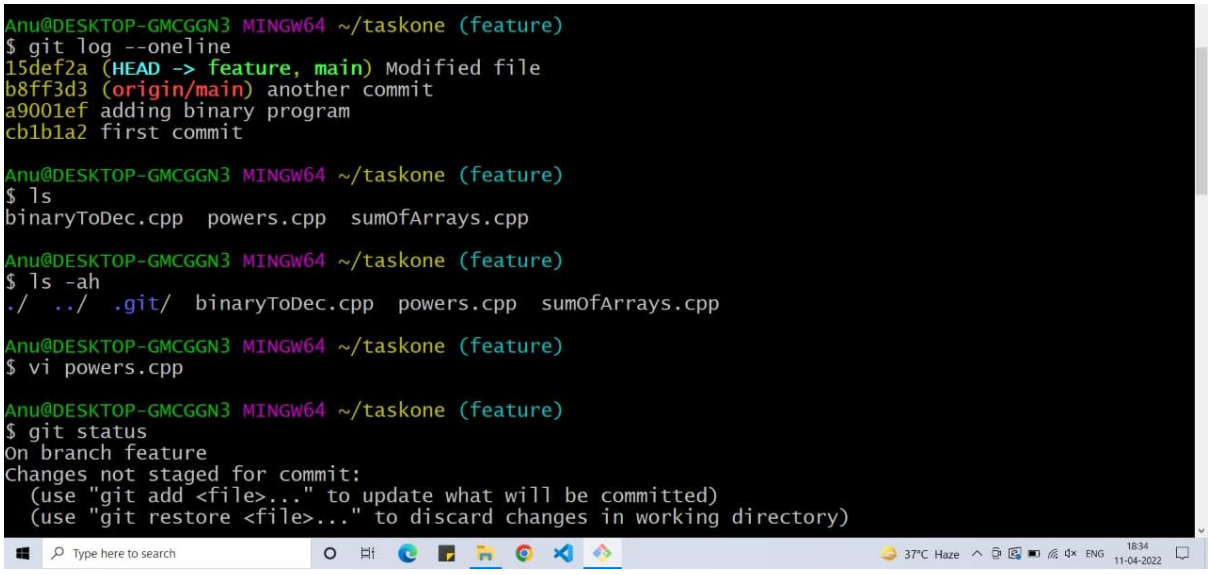
3. To change the present working branch: git checkout “name of the branch”
Visualizing Branches: To visualize, we have to create a new file in the new branch “activity1” instead of the master branch. After this we have to do three step architecture i.e. working directory, staging area and git repository.

After this I have done the 3 Step architecture which is tracking the file, send it to stagging area and finally we can rollback to any previously saved version of this file.

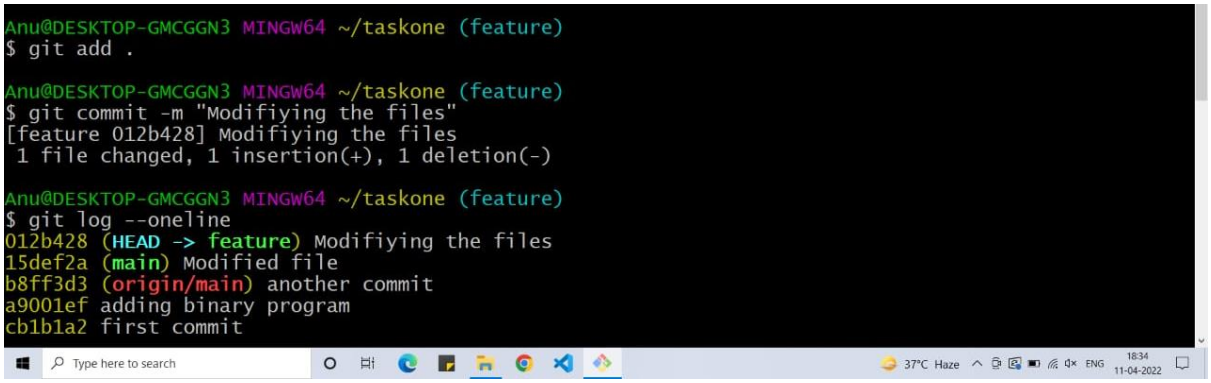
After this we will change the branch from activity1 to master, but when we switch back to master branch the file we created i.e. “hello” will not be there. Hence the new file will not be shown in the master branch. In this way we can create and change different branches. We can also merge the branches by using the git merge command.

In this way we can create and change different branches. We can also merge the branches by using git merge command.


Anu@DESKTOP-GMCGGN3 MINGW64 ~ (main)
$ cd taskone
Anu@DESKTOP-GMCGGN3 MINGW64 ~/taskone (main)
$ git log --oneline
15def2a (HEAD -> main) Modified file
b8ff3d3 (origin/main) another commit
a9001ef adding binary program
cb1b1a2 first commit
Anu@DESKTOP-GMCGGN3 MINGW64 ~/taskone (main)
$ git branch
* main
Anu@DESKTOP-GMCGGN3 MINGW64 ~/taskone (main)
$ git branch feature
Anu@DESKTOP-GMCGGN3 MINGW64 ~/taskone (main)
$ git branch
feature
* main
Anu@DESKTOP-GMCGGN3 MINGW64 ~/taskone (main)
$ git checkout feature
Switched to branch 'feature'
Type here to search
37°C Haze
ENG
18:34
11-04-2022


Anu@DESKTOP-GMCGGN3 MINGW64 ~/taskone (feature)
$ git log --oneline
15def2a (HEAD -> feature, main) Modified file
b8ff3d3 (origin/main) another commit
a9001ef adding binary program
cb1b1a2 first commit
Anu@DESKTOP-GMCGGN3 MINGW64 ~/taskone (feature)
$ ls
binaryToDec.cpp powers.cpp sumOfArrays.cpp
Anu@DESKTOP-GMCGGN3 MINGW64 ~/taskone (feature)
$ ls -ah
./ ../ .git/ binaryToDec.cpp powers.cpp sumOfArrays.cpp
Anu@DESKTOP-GMCGGN3 MINGW64 ~/taskone (feature)
$ vi powers.cpp
Anu@DESKTOP-GMCGGN3 MINGW64 ~/taskone (feature)
$ git status
On branch feature
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)
Type here to search
37°C Haze
ENG
18:34
11-04-2022

```
Anu@DESKTOP-GMCGGN3 MINGW64 ~/taskone (feature)
$ git add .

Anu@DESKTOP-GMCGGN3 MINGW64 ~/taskone (feature)
$ git commit -m "Modifiying the files"
[feature 012b428] Modifiying the files
 1 file changed, 1 insertion(+), 1 deletion(-)

Anu@DESKTOP-GMCGGN3 MINGW64 ~/taskone (feature)
$ git log --oneline
012b428 (HEAD -> feature) Modifiying the files
15def2a (main) Modified file
b8ff3d3 (origin/main) another commit
a9001ef adding binary program
cb1b1a2 first commit
```




```
MINGW64:/c/Users/Anu/taskone
Anu@DESKTOP-GMCGGN3 MINGW64 ~/taskone (feature)
$ cat powers.cpp
#include<iostream>
using namespace std;
//To find power of a number
int power(int num1, int num2) {
    int ans = 1;

    for(int i = 1; i <= num2; i++) {
        ans = ans * num1;
    }
    return ans;
}
int main()
{
    int c , d;
    cin>> c >> d;
//last comment is added to this file
    int answer = power(c,d);
    cout << " answer is " << answer << endl;
    return 0;
}

Anu@DESKTOP-GMCGGN3 MINGW64 ~/taskone (feature)
$ git log --oneline
012b428 (HEAD -> feature) Modifiying the files
15def2a (main) Modified file
b8ff3d3 (origin/main) another commit
a9001ef adding binary program
```

```
Anu@DESKTOP-GMCGGN3 MINGW64 ~/taskone (feature)
$ git checkout feature
Already on 'feature'

Anu@DESKTOP-GMCGGN3 MINGW64 ~/taskone (feature)
$ git log --oneline
012b428 (HEAD -> feature) Modifiying the files
15def2a (main) Modified file
b8ff3d3 (origin/main) another commit
a9001ef adding binary program
cb1b1a2 first commit
```

```
Anu@DESKTOP-GMCGGN3 MINGW64 ~/taskone (feature)
$ git push -u origin feature
Enumerating objects: 15, done.
Counting objects: 100% (15/15), done.
Delta compression using up to 8 threads
Compressing objects: 100% (14/14), done.
Writing objects: 100% (15/15), 1.63 KiB | 238.00 KiB/s, done.
Total 15 (delta 5), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (5/5), done.
remote:
remote: Create a pull request for 'feature' on GitHub by visiting
:
remote:      https://github.com/Group08-Chitkara-University/21109
```

EXPERIMENT-5

Aim: Git lifecycle description

Git is used in our day-to-day work, we use Git for keeping a track of our files, working in a collaboration with our team, to go back to our previous code versions if we face some error. Git helps us in many ways. Let us look at the Lifecycle description that git has and understand more about its life cycle. Let us see some of the basic steps that we have to follow while working with Git-

• Step 1- We first clone any of the code residing in the remote repository to make our won local repository.

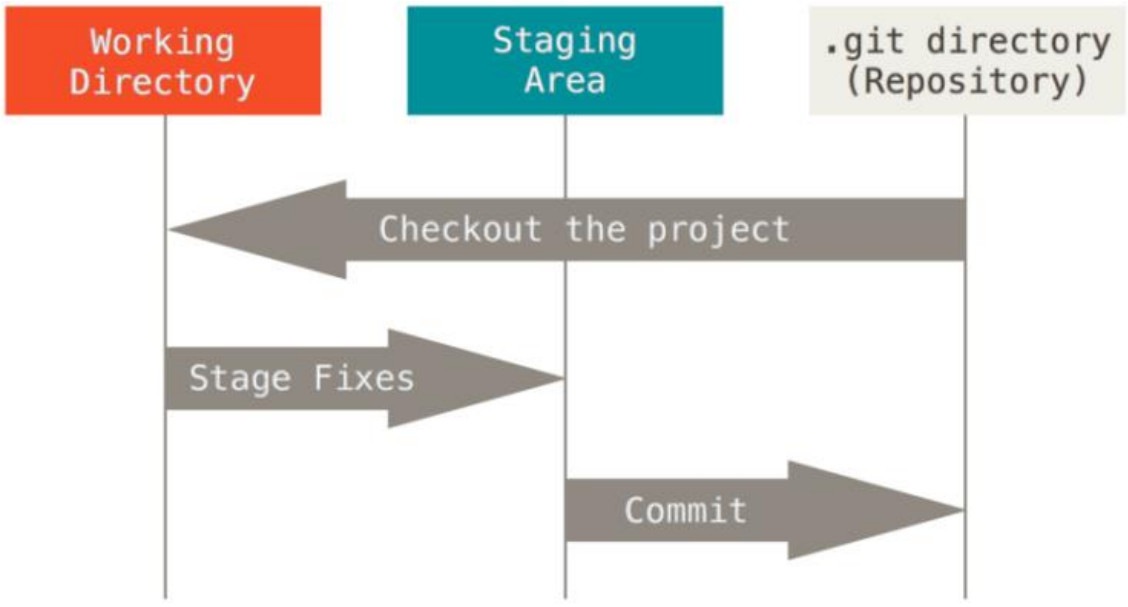
• Step 2- We edit the files that we have cloned in our local repository and make the necessary changes in it.

• Step 3- We commit our changes by first adding them to our staging area and committing them with a commit message.

• 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.

• Step 6- If there are no changes 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:



1. Working Directory Whenever we want to initialize aur local project directory to make a Git repository, we use the git init command. After this command, git becomes aware of the files in the project although it does not track the files yet. The files are further tracked in the staging area.

2. Staging Area Now, to track files the different versions of our files we use the command git add. We can term a staging area as a place where different versions of our files are stored. git add command copies the version of your file from your working directory to the staging area. We can, however, choose which files we need to add to the staging area because in our working directory there are some files that we don't want to get tracked, examples include node modules, temporary files, etc. Indexing in Git is the one that helps Git in understanding which files need to be added or sent. You can find your staging area in the .git folder inside the index file.

git add<filename>

git add .

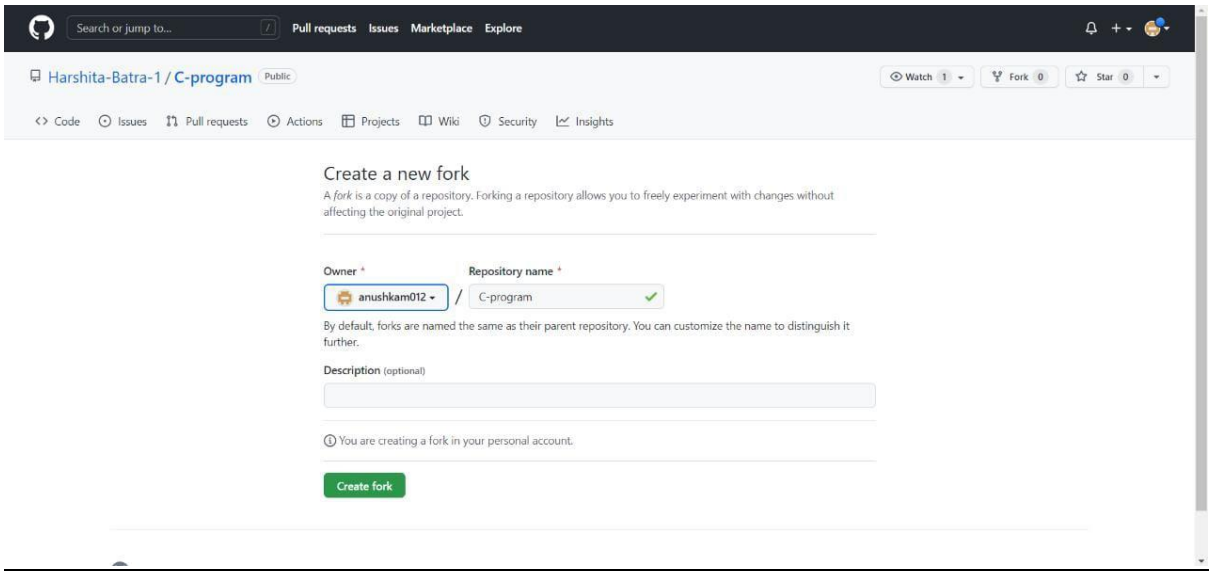
3. Git Directory

Now since we have all the files that are to be tracked and are ready in the staging area, we are ready to commit aur files using the git commit command. Commit helps us in keeping the track of the metadata of the files in our staging area. We specify every commit with a message which tells what the commit is about. Git preserves the information or the metadata of the files that were committed in a Git Directory which helps Git in tracking files basically it preserves the photocopy of the committed files. Commit also stores the name of the author who did the commit, files that are committed, and the date at which they are committed along with the commit message. git commit -m

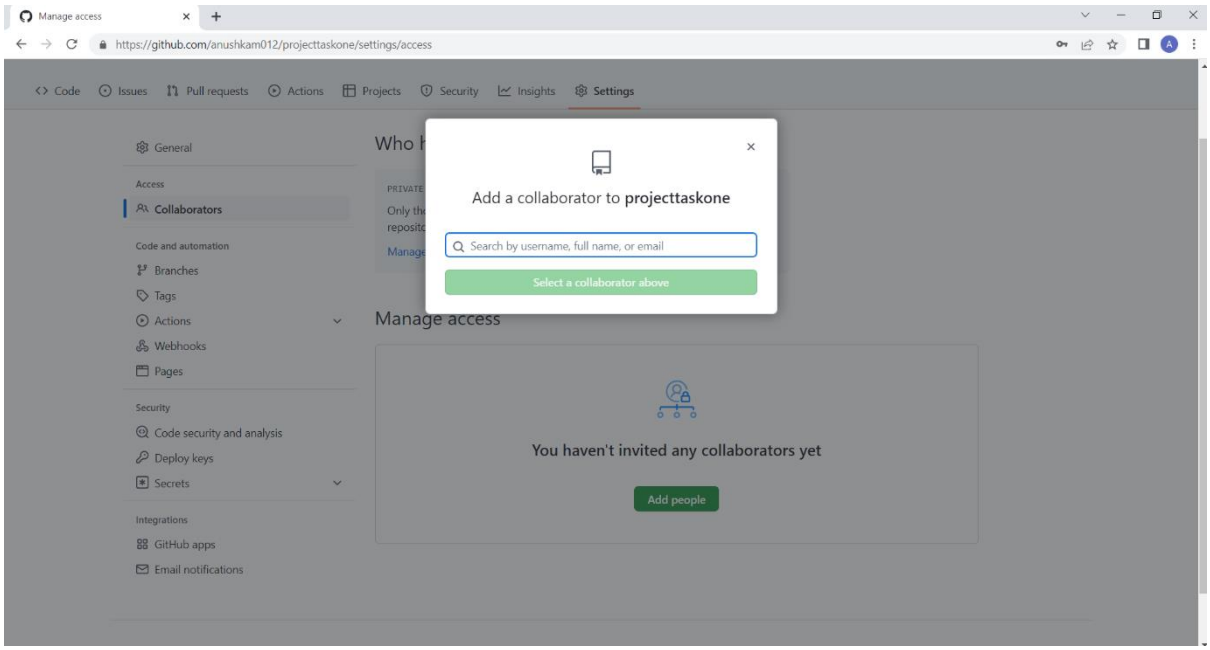
CHITKARA UNIVERSITY

TASK 1.2

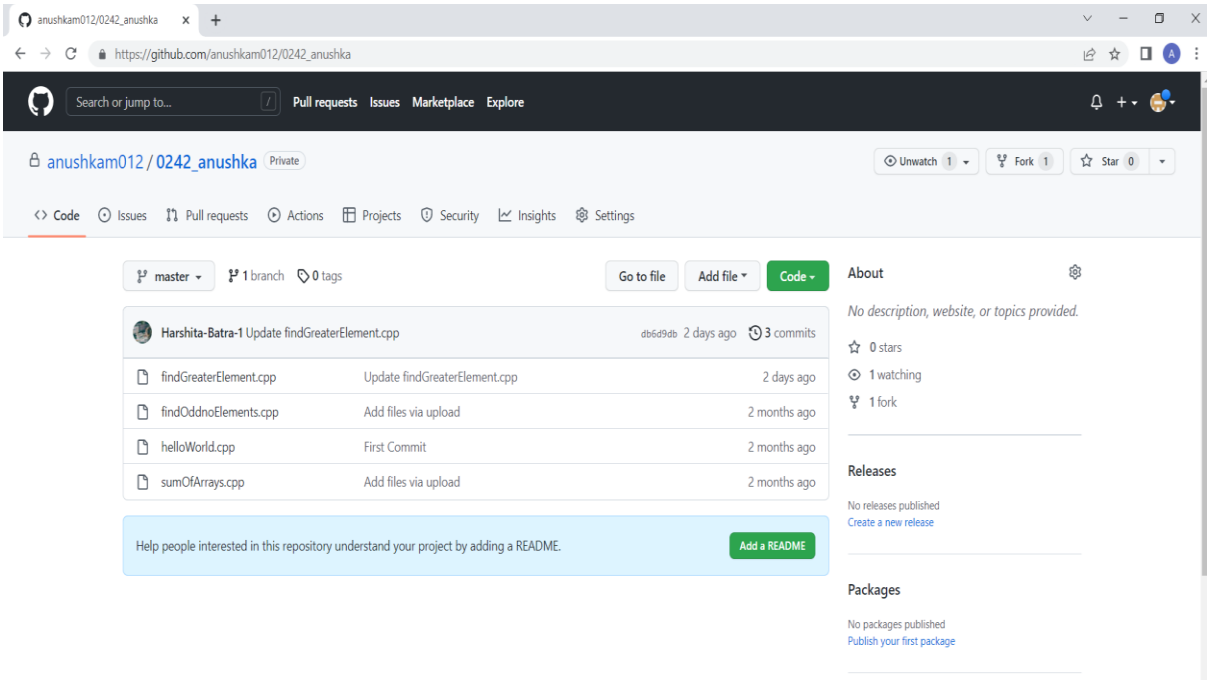
1. Add Collaborators on Github Repository



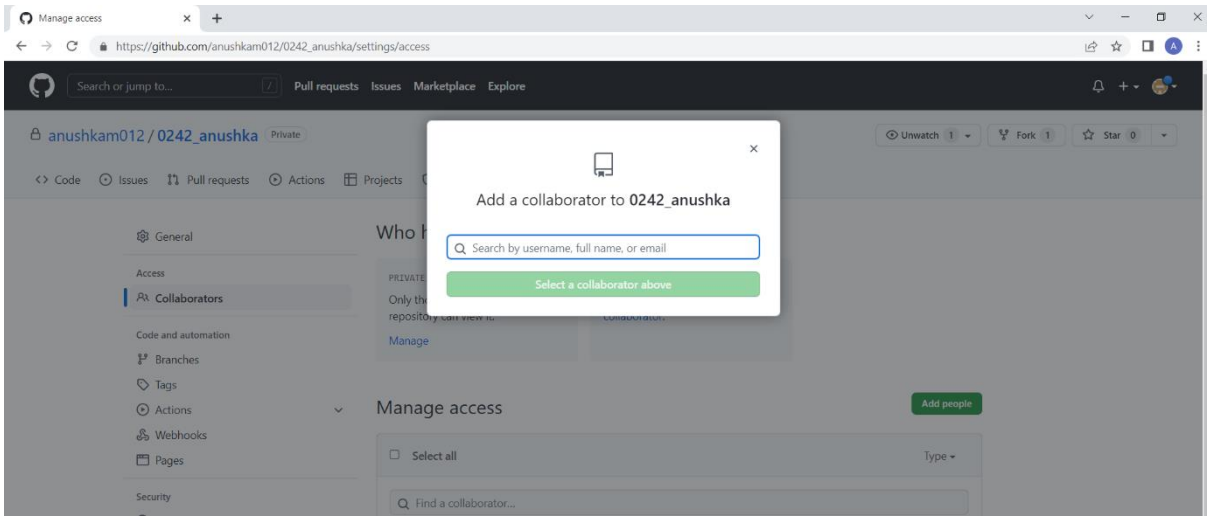
1. On GitHub.com, navigate to the main page of the repository.



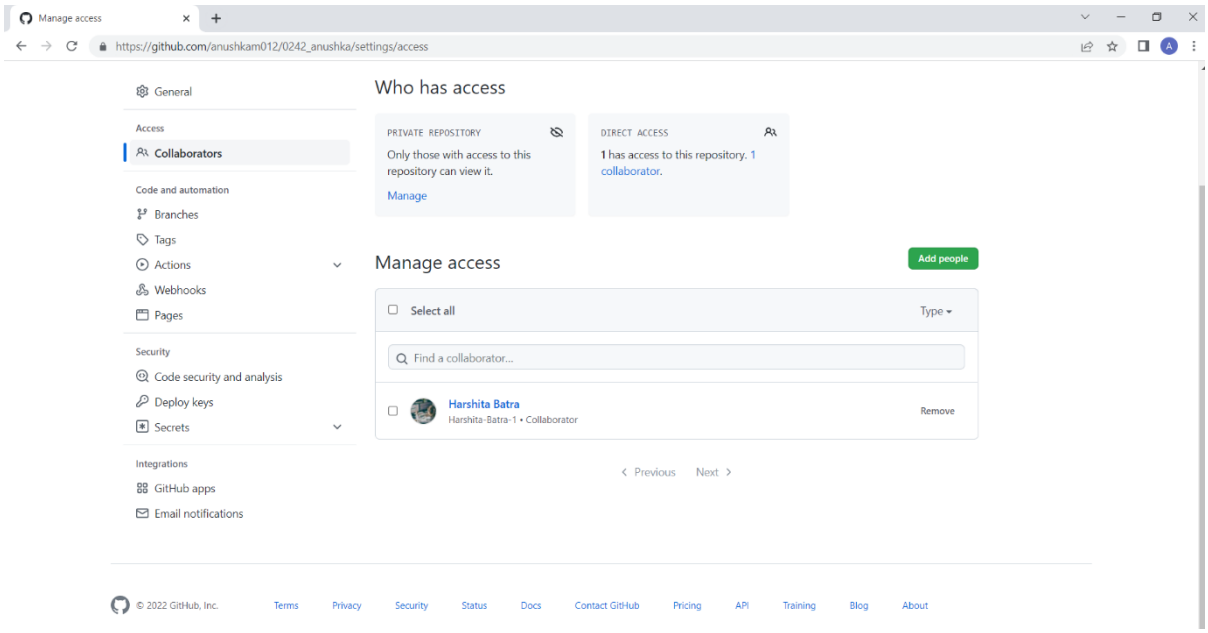
2. Under our repository name, click Settings.



3. In the "Access" section of the sidebar, click Collaborators & teams.



4. Click Invite a collaborator.

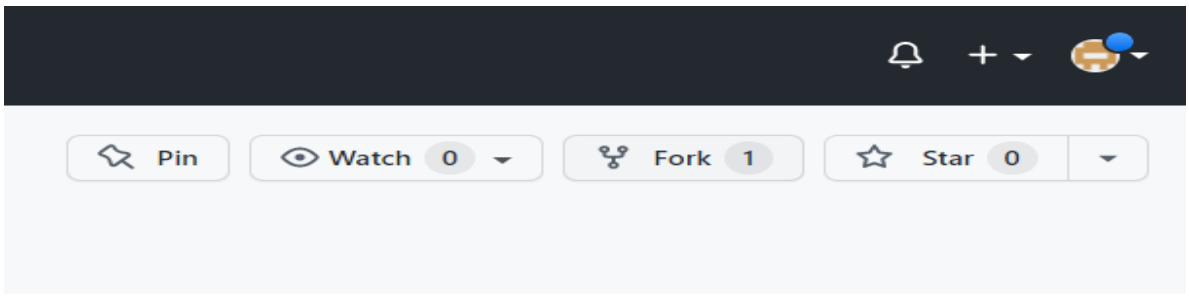


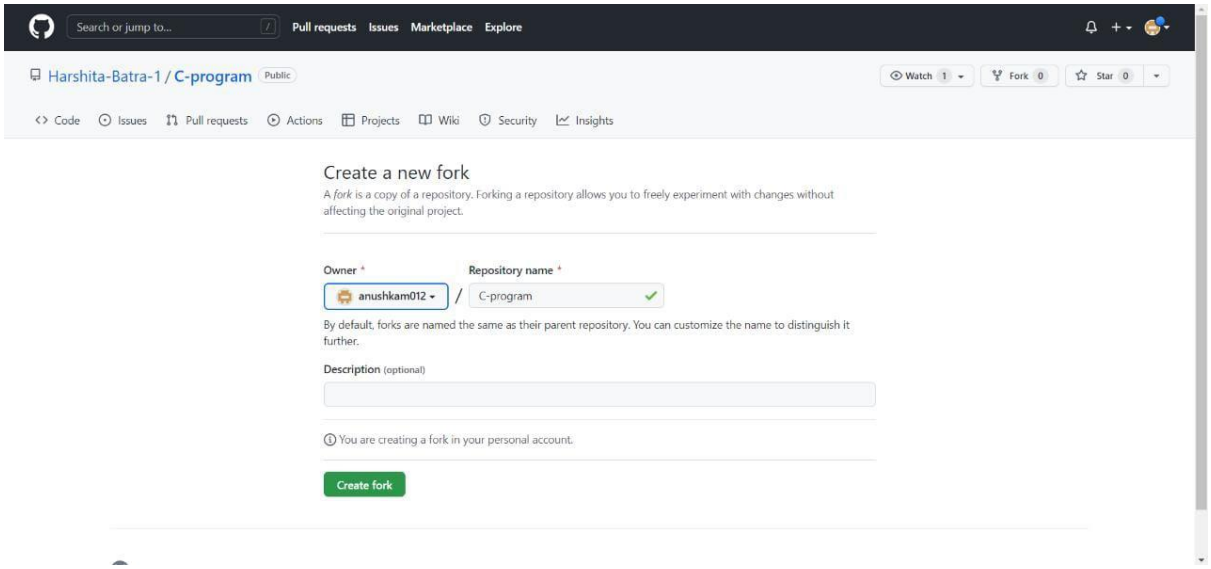
2. Fork And Commit

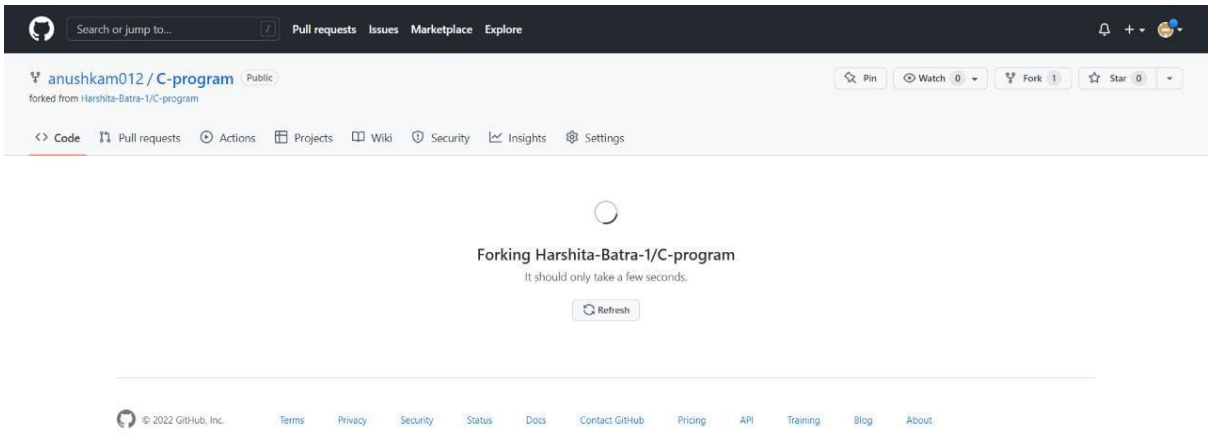
A fork is a copy repository. Forking a repository allows you to freely experiment with changes without affecting the original project. Most commonly, forks are used to either propose changes to someone else's project to which you do not have write access, or to use someone else's project as a starting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

To follow along, browse to a public repository that you want to fork. At the top right of the page, you will find the Fork button. Click on the button and wait for a few seconds. You will see that the newly forked repository gets created under your GitHub account.

Fork Button







Once you have navigated to the directory where you want to put your repository, you can use **git clone** command.

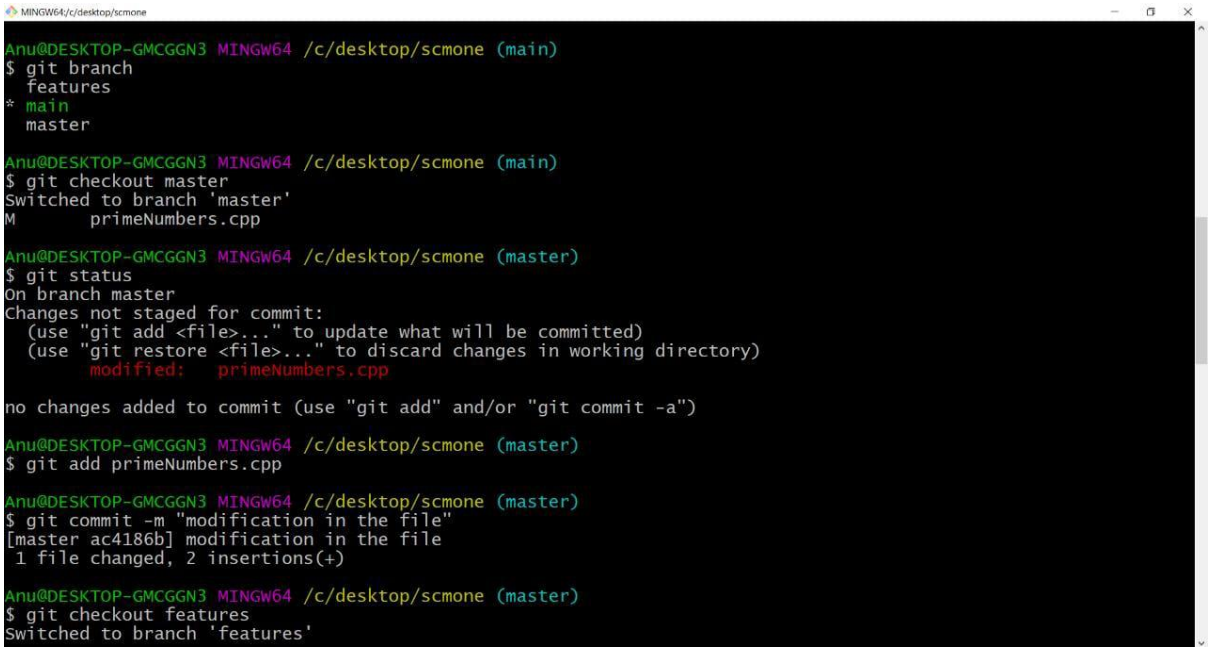
The `git clone` command copies your repository from GitHub to your local computer. Note that this is a git specific command.

```
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/forkingcfile
$ git clone https://github.com/anushkam012/C-program.gi
t
Cloning into 'C-program'...
remote: Enumerating objects: 19, done.
remote: Counting objects: 100% (19/19), done.
remote: Compressing objects: 100% (13/13), done.
remote: Total 19 (delta 3), reused 11 (delta 1), pack-reused 0
Receiving objects: 100% (19/19), 31.16 KiB | 437.00 KiB/s, done.
Resolving deltas: 100% (3/3), done.
```

4. **Merge and Resolve conflicts created due to own Activity and Collaborators activity :-**

Let us now look into the Git commands that may play a significant role in resolving conflicts.

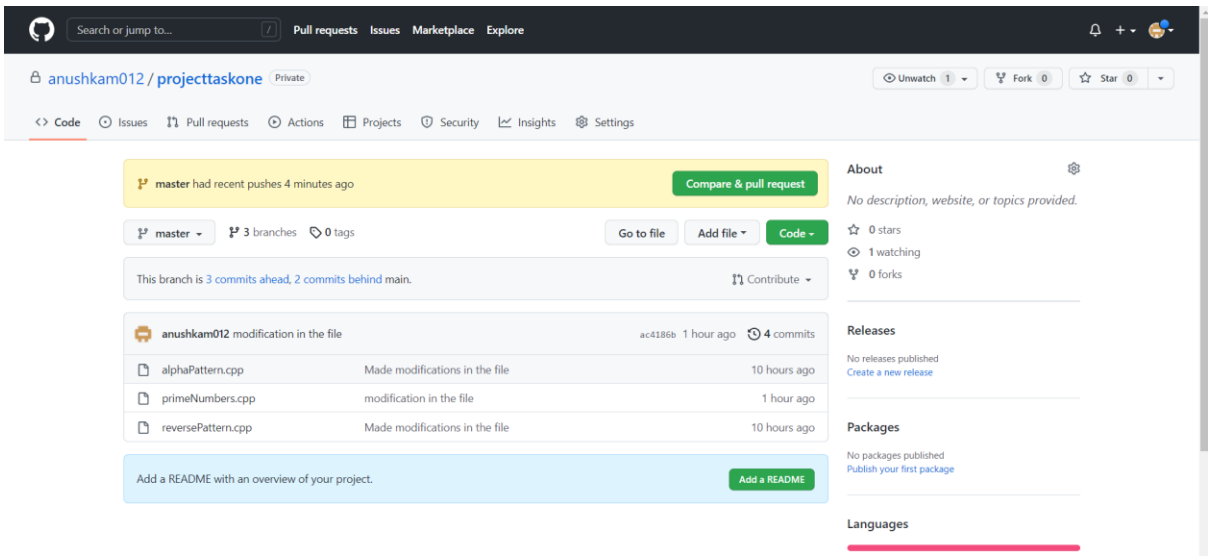
To merge branches locally, use **git checkout** to switch to the branch you want to merge into. This branch is typically the *main* branch. Next, use **git merge** and specify the name of the other branch to bring into this branch. This example merges the features branch into the master branch. This is known as a fast-forward merge.

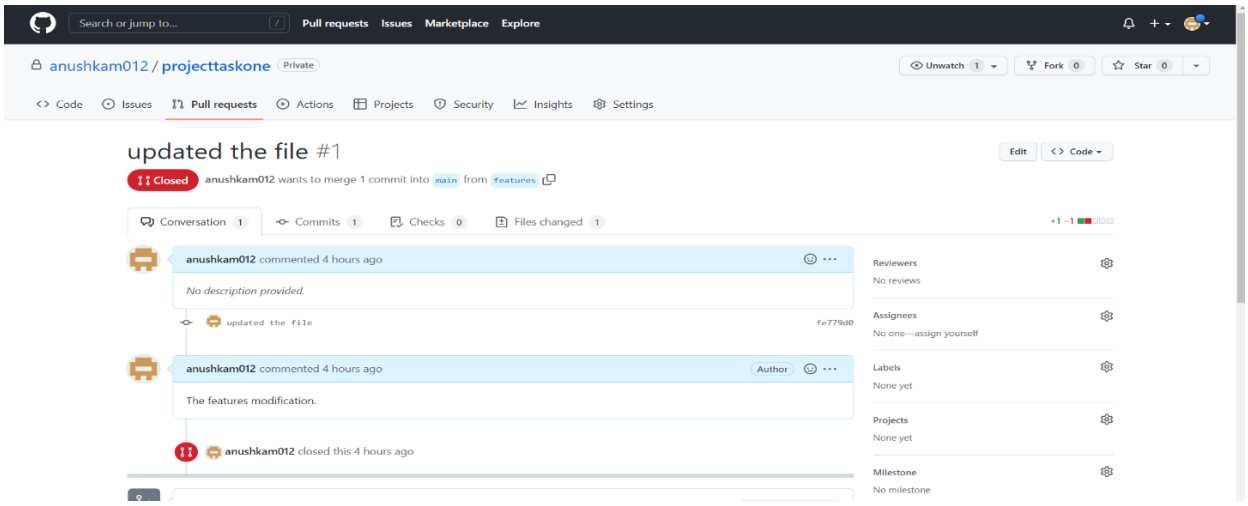


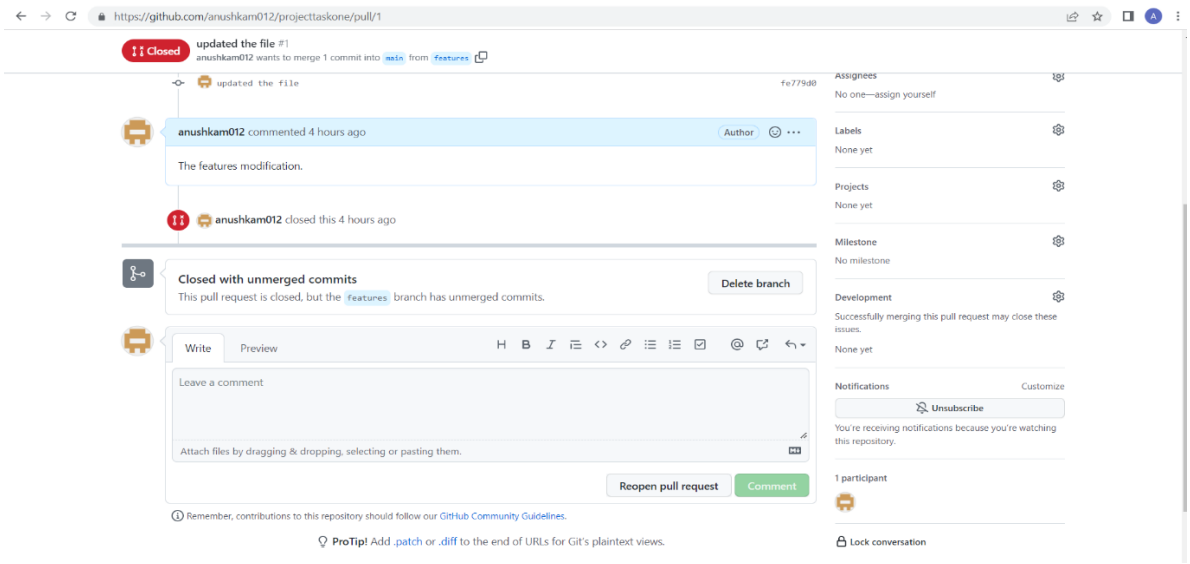
```
MINGW64:/c/desktop/scmone

Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/scmone (master)
$ git checkout features
Switched to branch 'features'
Your branch is ahead of 'origin/features' by 1 commit.
  (use "git push" to publish your local commits)

Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/scmone (features)
$ git merge master
Updating 3fc3fab..ac4186b
Fast-forward
 primeNumbers.cpp | 2 ++
 1 file changed, 2 insertions(+)

Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/scmone (features)
$ git push -u origin master
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 8 threads
Compressing objects: 100% (7/7), done.
Writing objects: 100% (7/7), 1.25 KiB | 255.00 KiB/s, done.
Total 7 (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 'master' on GitHub by visiting:
remote:      https://github.com/anushkam012/projecttaskone/pull/new/master
remote:
To https://github.com/anushkam012/projecttaskone.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.

Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/scmone (features)
$ cat primeNumbers.cpp
```

```
MINGW64:/c/desktop/scmone

Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/scmone (features)
$ cat primeNumbers.cpp
//Program for printing all prime numbers.
//NOTE-Prime number is a number that is greater than 1 and divided by 1 or itself. In other words, prime number
s can't be divided by other numbers than itself or 1. For example 2, 3, 5, 7, 11, 13, 17, 19, 23.... are the pr
ime numbers.

#include<iostream>
using namespace std;

int main(){

    int N, i, j, isPrime, n;

    cin >> N;
    for(i = 2; i <= N; i++){
        isPrime = 0;

        for(j = 2; j <= i/2; j++){

            if(i % j == 0){
                isPrime = 1;
                break;
            }
        }

        if(isPrime==0 && N!= 1)
            cout << i << ""<<endl;
    }

   return 0;
}
```

```
    for(i = 2; i <= N; i++){
        isPrime = 0;

        for(j = 2; j <= i/2; j++){

            if(i % j == 0){
                isPrime = 1;
                break;
            }
        }

        if(isPrime==0 && N!= 1)
            cout << i << ""<<endl;
    }

   return 0;
}
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/scmone (features)
$ git log --oneline
ac4186b (HEAD -> features, origin/master, master) modification in the file
3fc3fab Made modifications in the file
fe779d0 (origin/features) updated the file
f9126c6 (origin/main) initial commit

Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/scmone (features)
$ git log --oneline --graph
* ac4186b (HEAD -> features, origin/master, master) modification in the file
* 3fc3fab Made modifications in the file
* fe779d0 (origin/features) updated the file
* f9126c6 (origin/main) initial commit

Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/scmone (features)
$
```



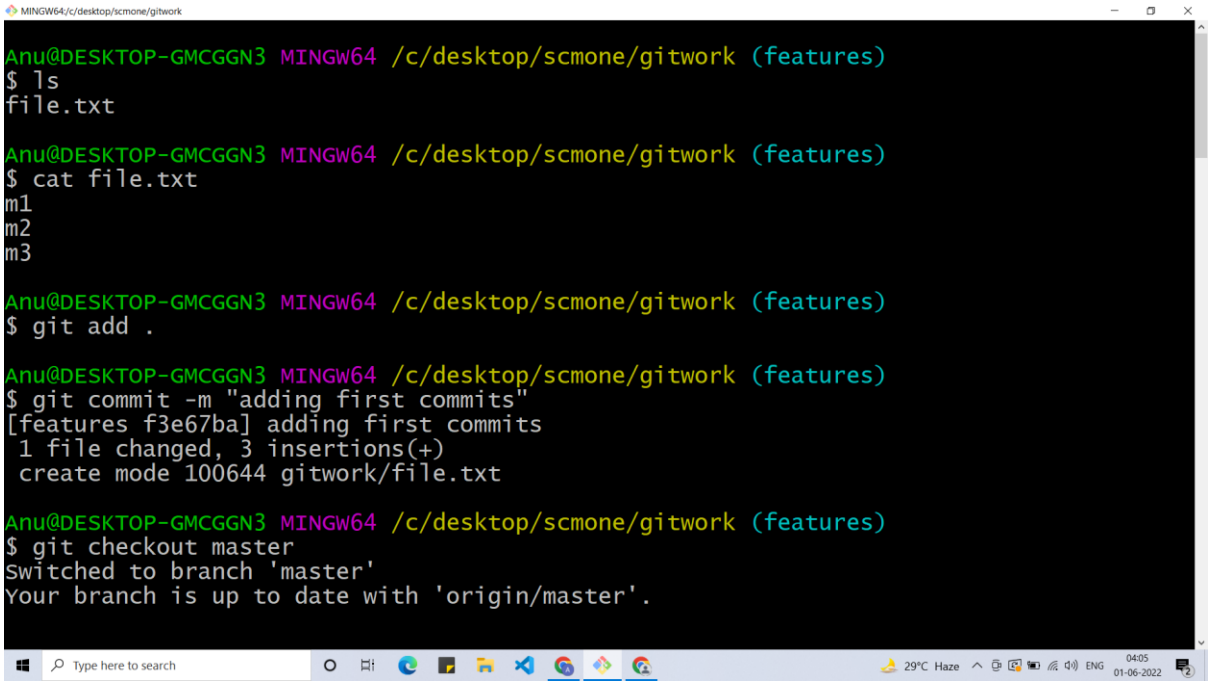



https://github.com/anushkam012/projecttaskone/pull/1
Closed
updated the file #1
anushkam012 wants to merge 1 commit into main from features
updated the file
fe779d0
anushkam012 commented 4 hours ago
Author
The features modification.
anushkam012 closed this 4 hours ago
Closed with unmerged commits
This pull request is closed, but the features branch has unmerged commits.
Delete branch
Write
Preview
Leave a comment
Attach files by dragging & dropping, selecting or pasting them.
Reopen pull request
Comment
Remember, contributions to this repository should follow our GitHub Community Guidelines.
ProTip! Add .patch or .diff to the end of URLs for Git's plaintext views.
Assignees
No one—assign yourself
Labels
None yet
Projects
None yet
Milestone
No milestone
Development
Successfully merging this pull request may close these issues.
None yet
Notifications
Customize
Unsubscribe
You're receiving notifications because you're watching this repository.
1 participant
Lock conversation

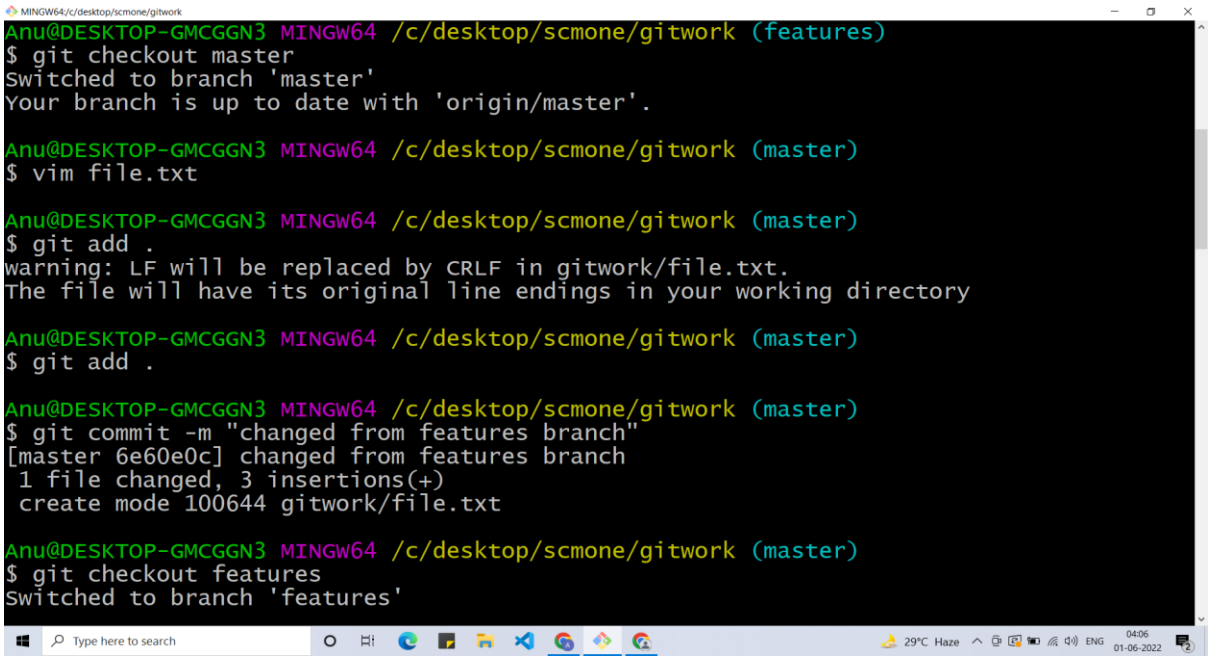
Resolving Conflicts

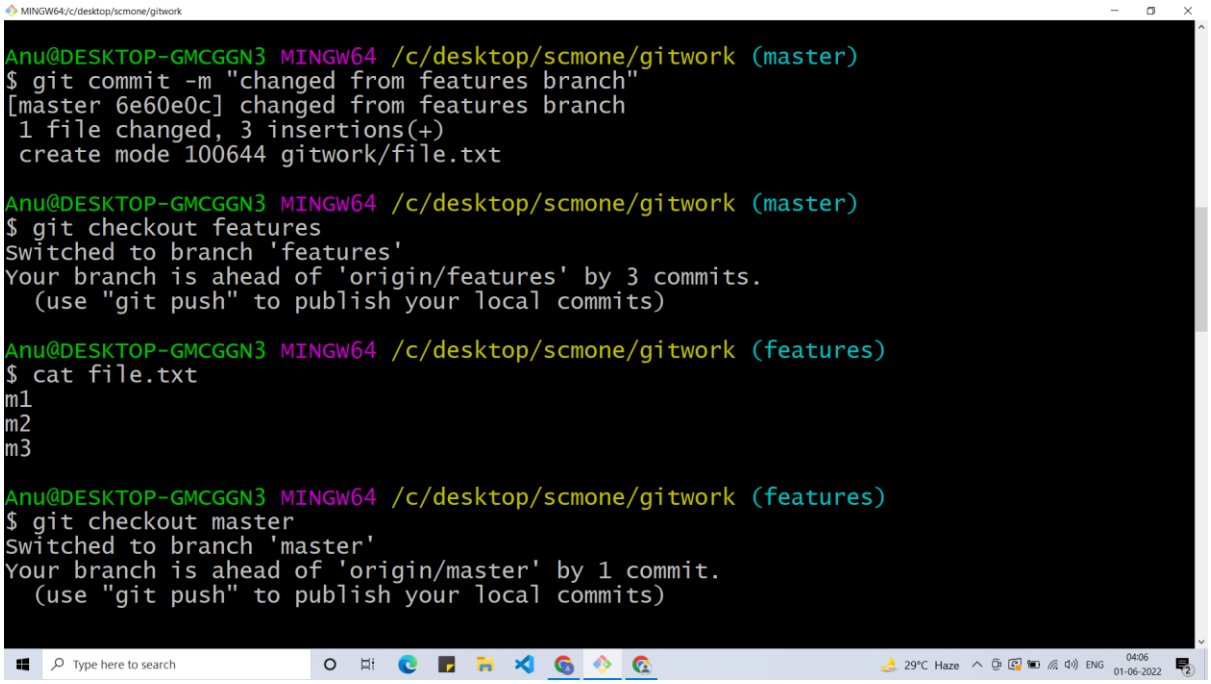
The git mergetool helps the developer to resolve conflicts in an efficient way.

1. Invoke gitbash.
2. Create a directory in your local computer.
3. Create an empty Git repository under this directory.
4. Create a new file 'file.txt'.
5. Add a new line to the file -> 'First Commit.'
6. Add the file to staging area, then commit it.

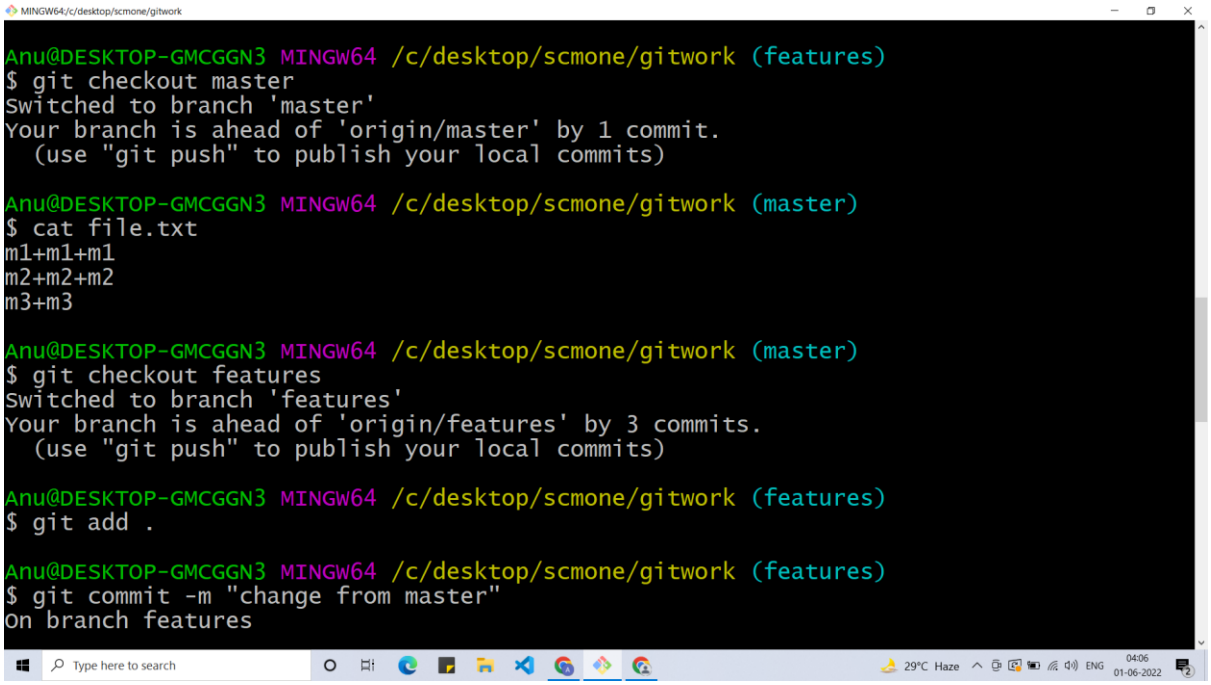


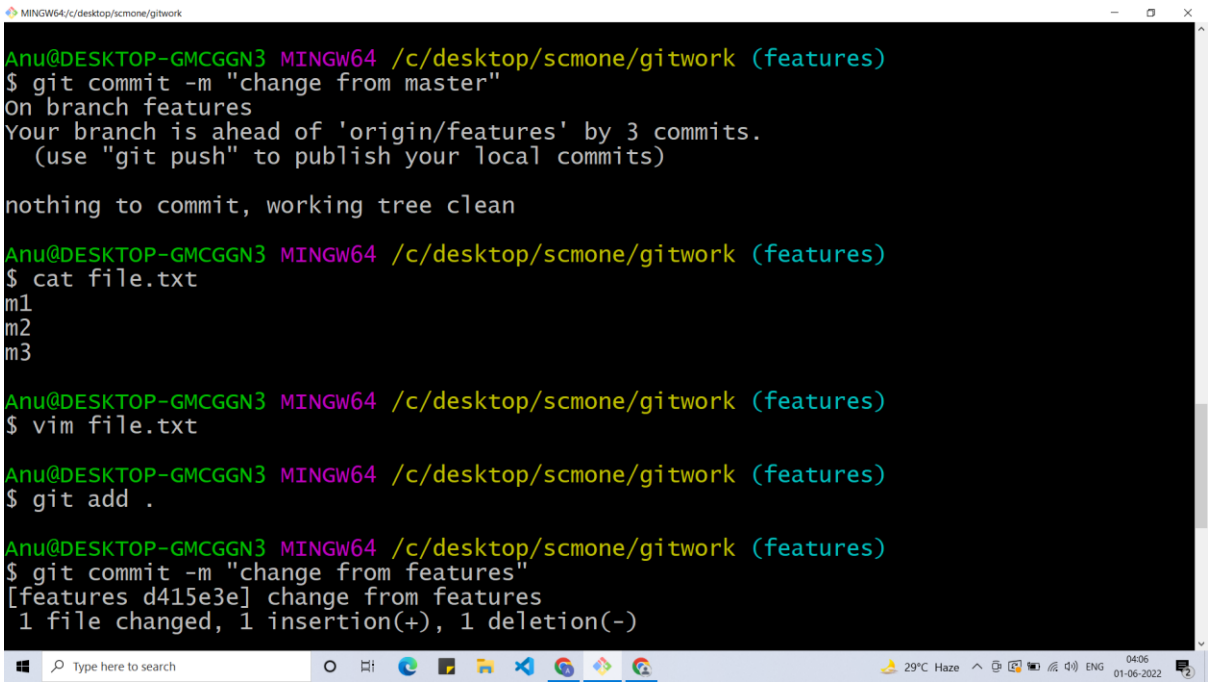
7. Now switch to another branch and then commit the file which is the second commit.





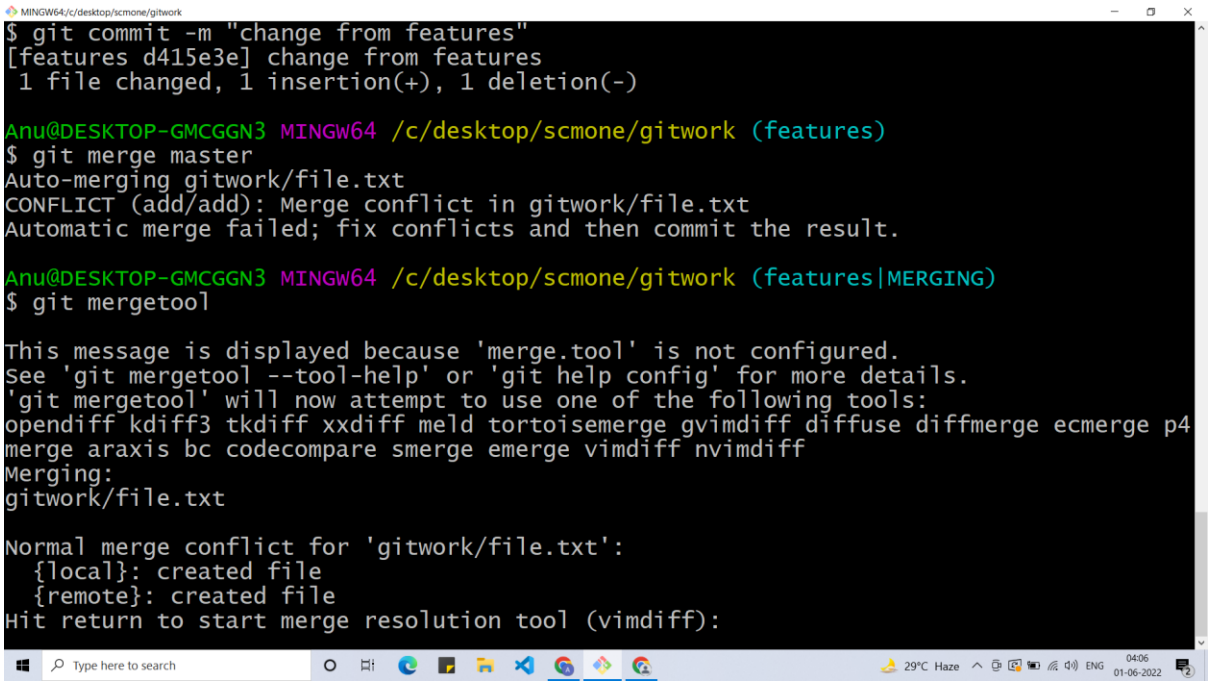
8. Now checkout to the another branch.





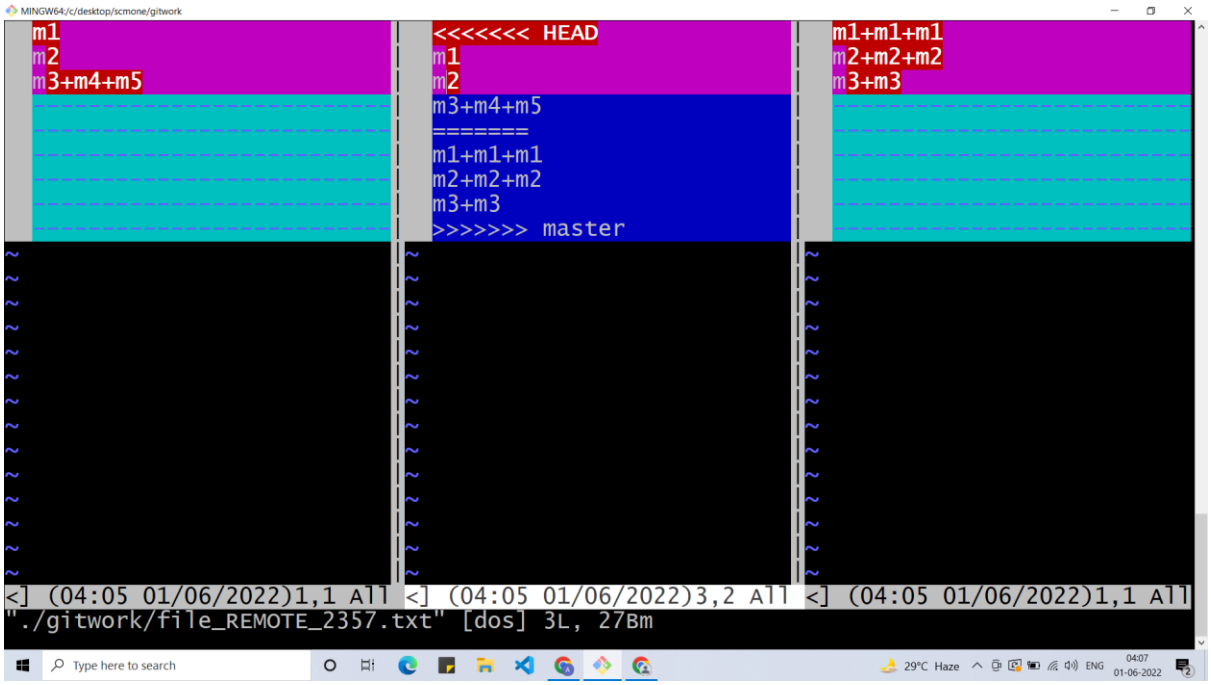
9. Finally, merge the branches using the command below. You will see an error.

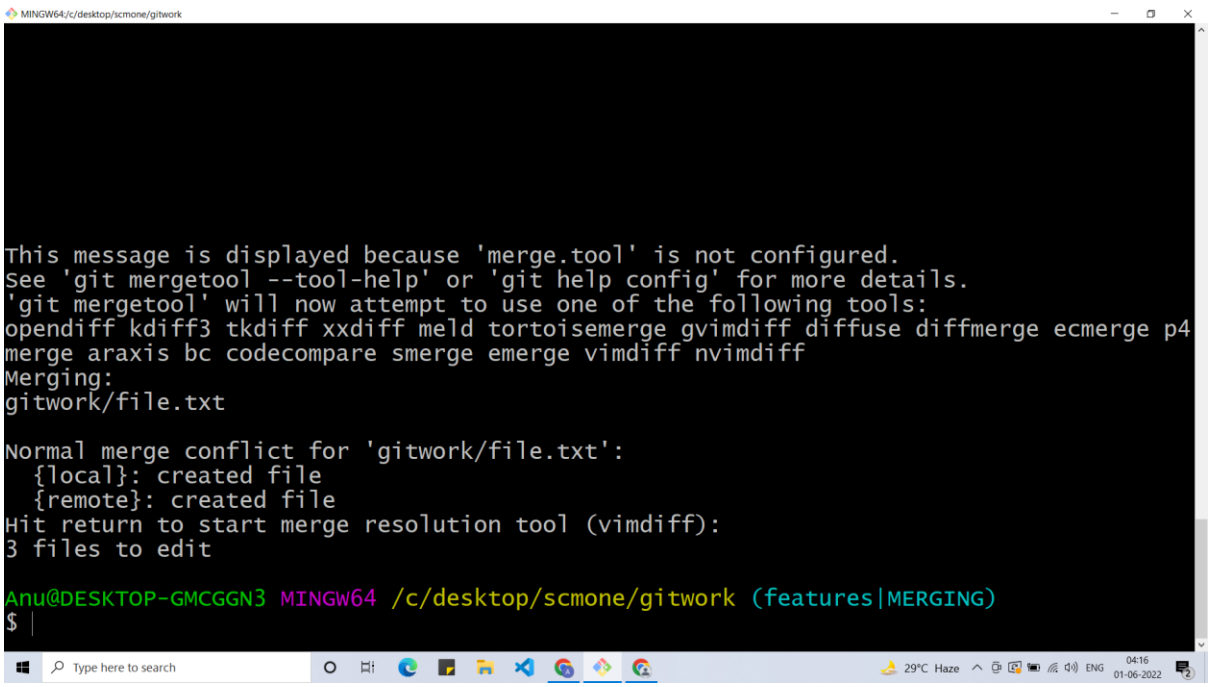
10. Now we have a conflict, because we changed the same line of a file in both branches. As Git cannot resolve this conflict automatically, we have to solve it by ourselves. Here, we will use Git Merge tool to resolve this. We will use 'git mergetool' command.



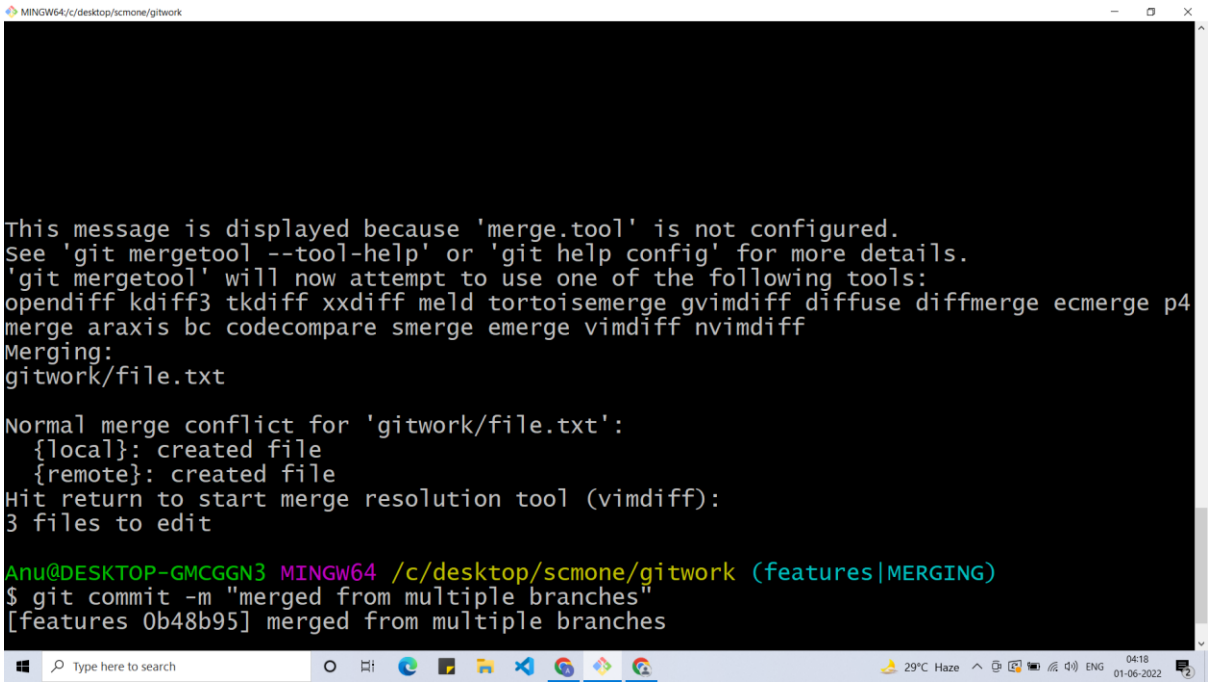
11. As you can see in the MERGED pane , the content where the conflict is present is wrapped around the lines <<<<<<<HEAD and >>>>>>>master separated by ==========

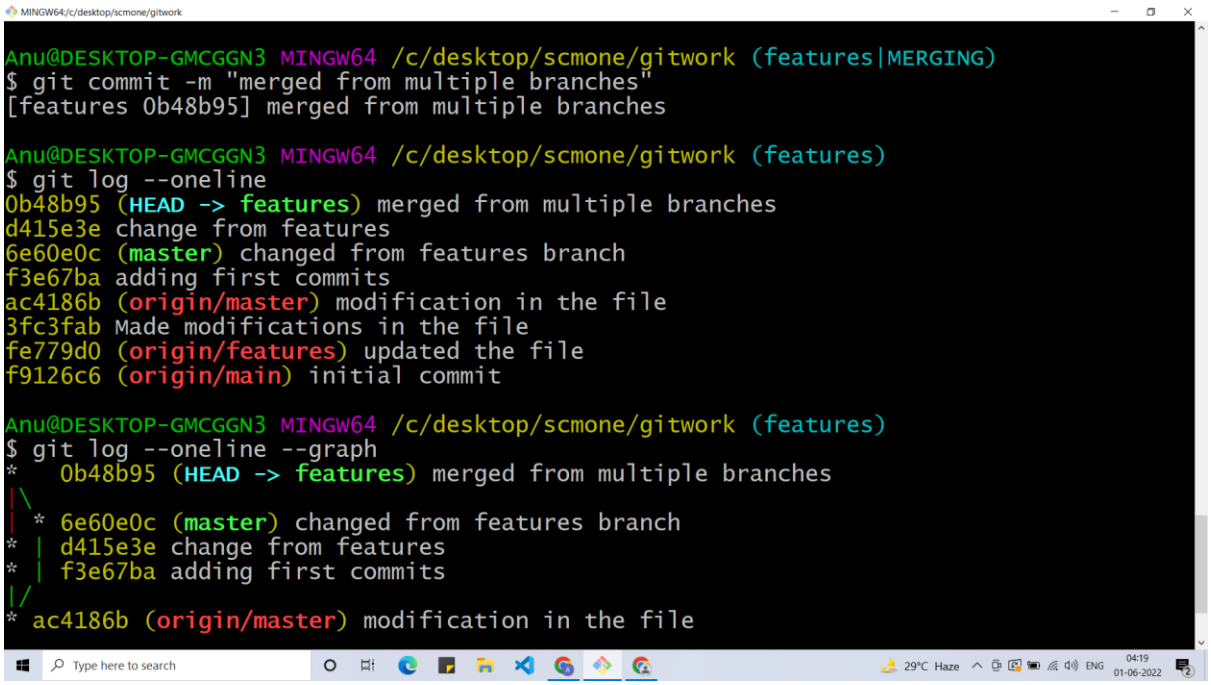
We can manually resolve the merge conflict by editing the content in the bottom pane, and then saving the file using :wqa (Write and Quit all files).





12. Once the conflict resolution is successful, the merged file will be staged for commit.



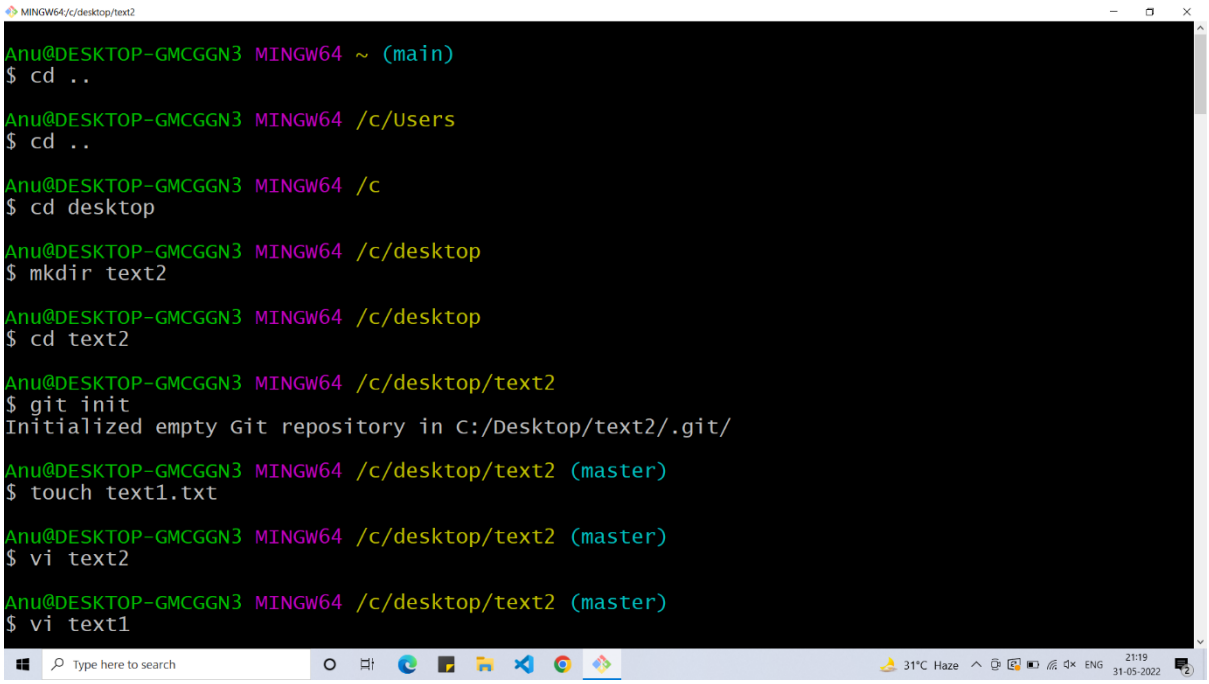


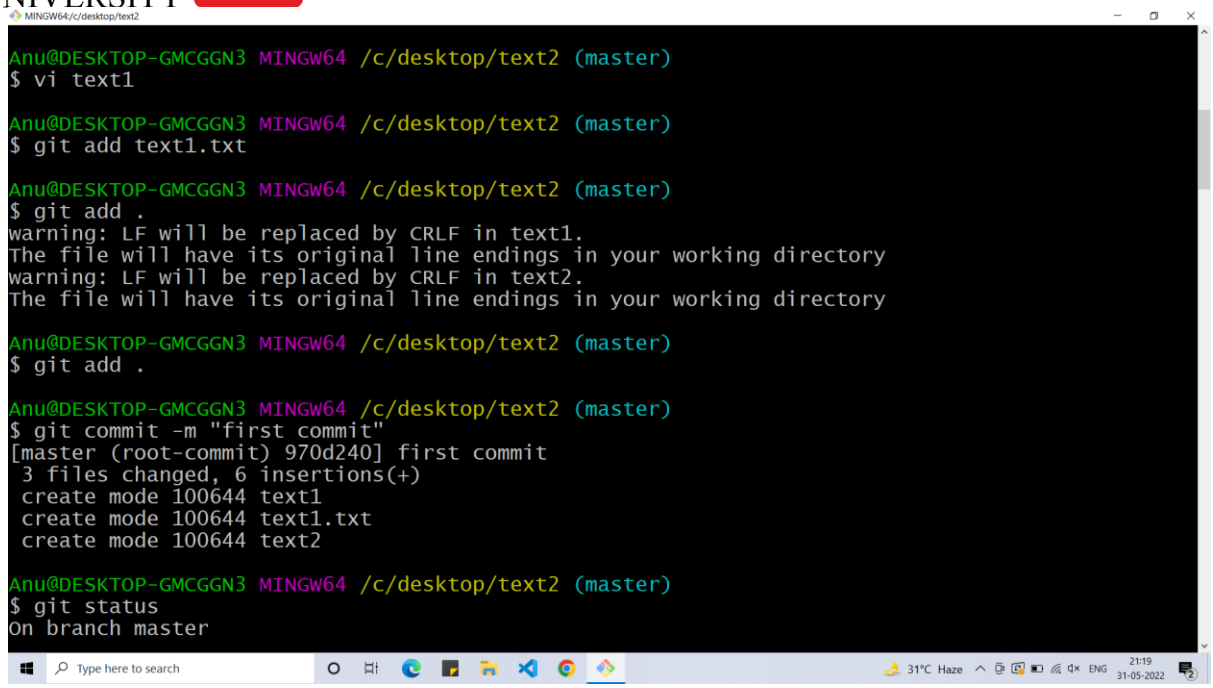
5. Git Reset :-

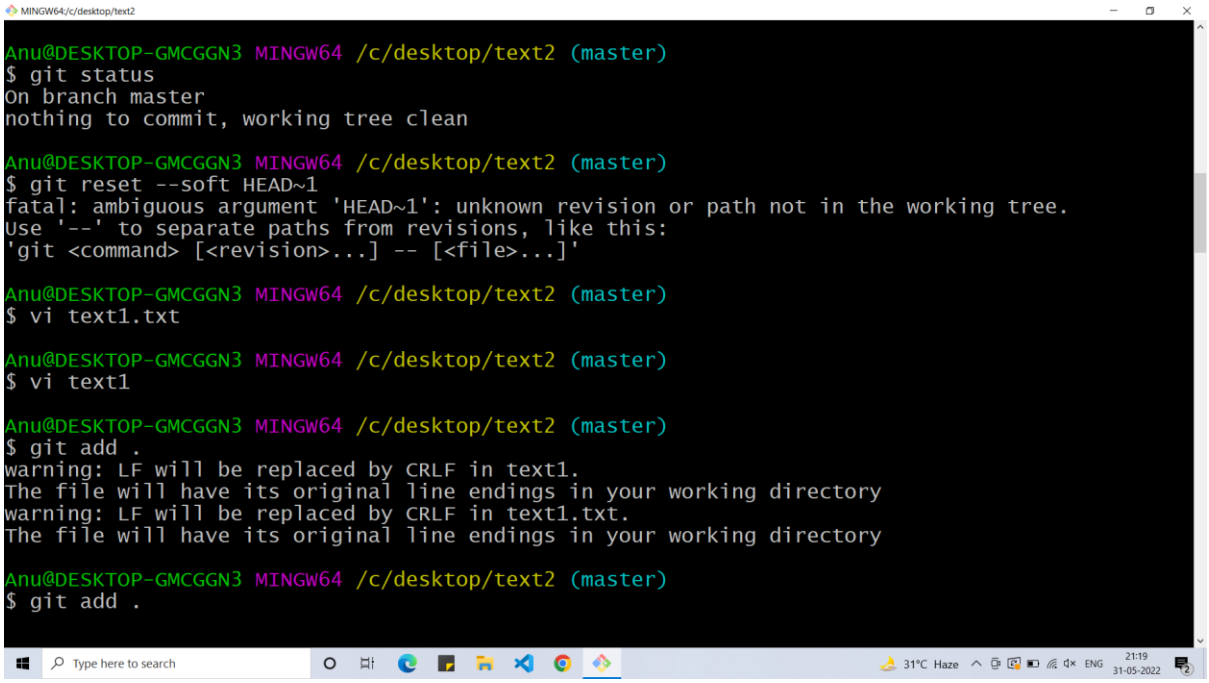
Git reset is a powerful command that is used to undo local changes to the state of a Git repo. Git reset operates on "The Three Trees of Git". These trees are the Commit History (HEAD), the Staging Index, and the Working Directory.

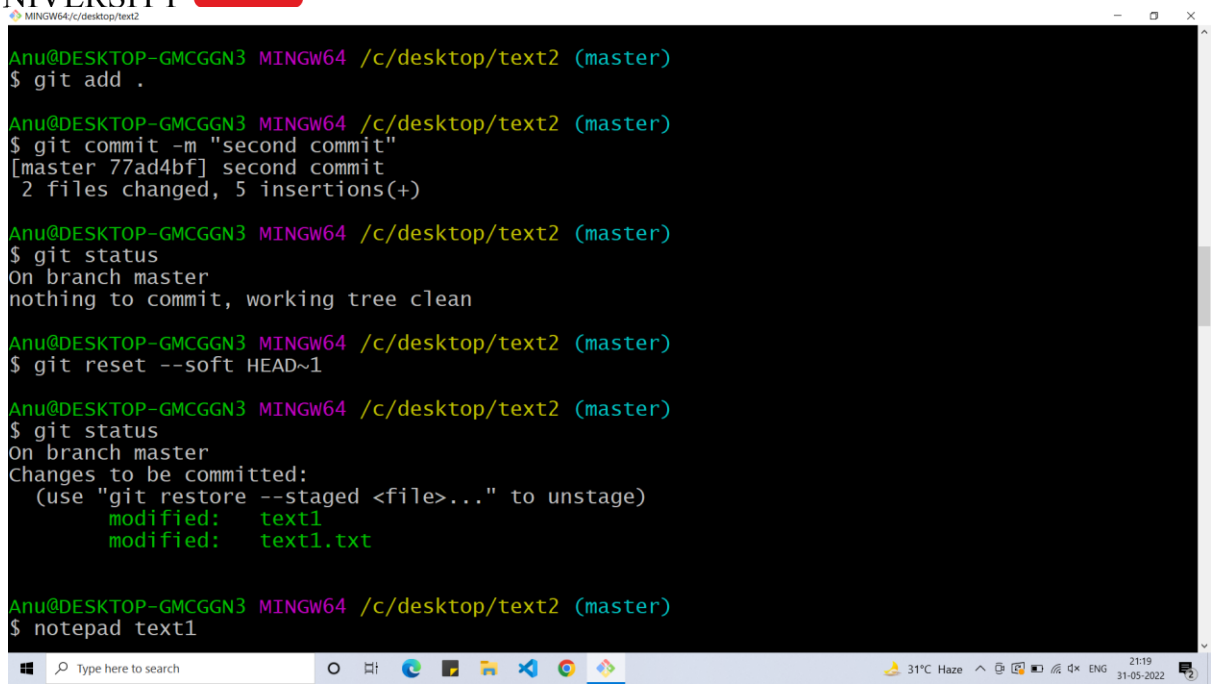
The easiest way to undo the last Git commit is to execute the “git reset” command with the “–soft” option that will preserve changes done to your files.

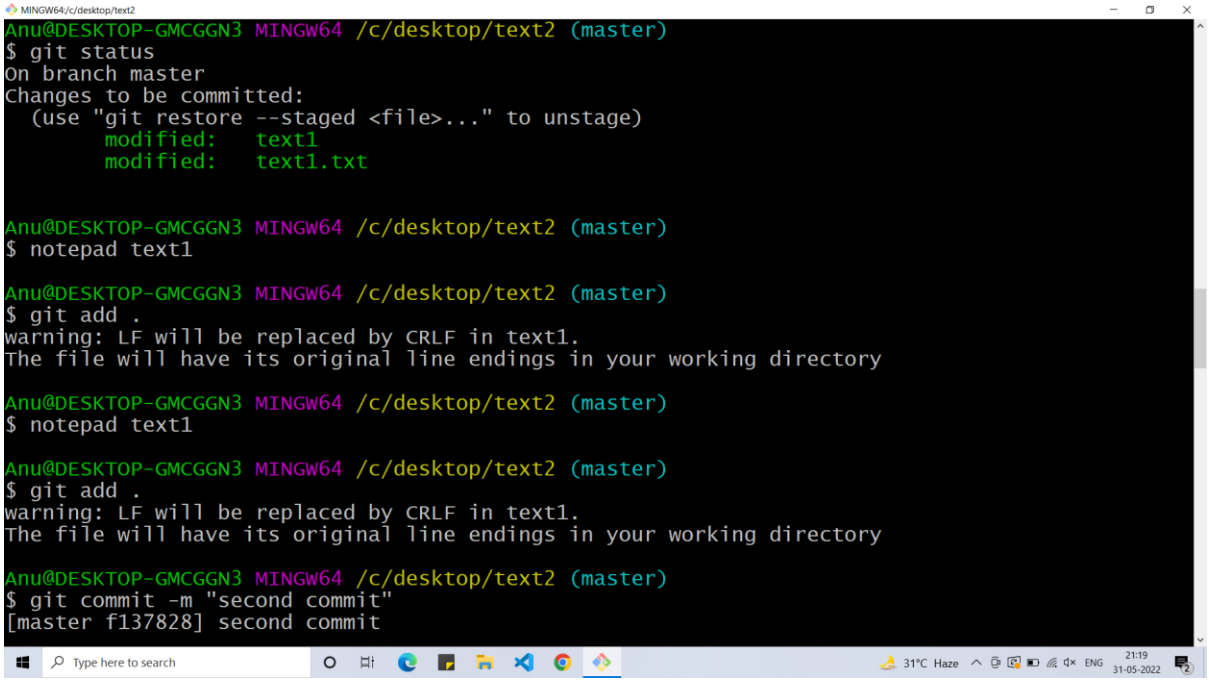
Git reset --hard , which will completely destroy any changes and remove them from the local directory.

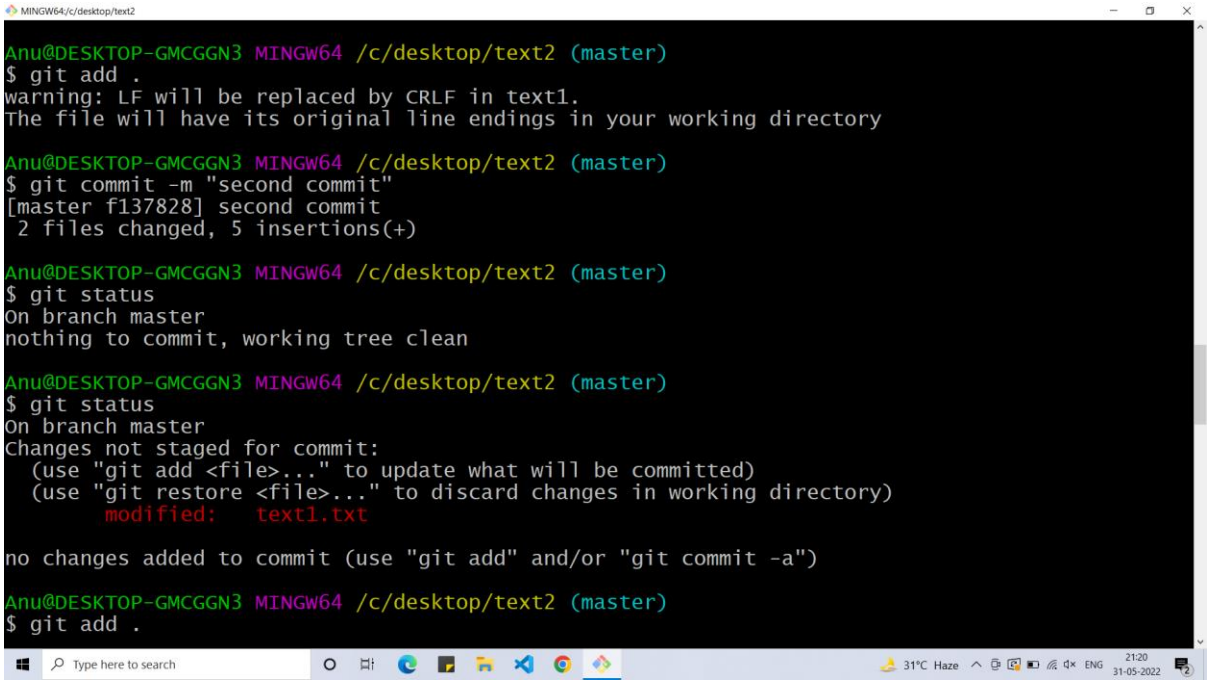


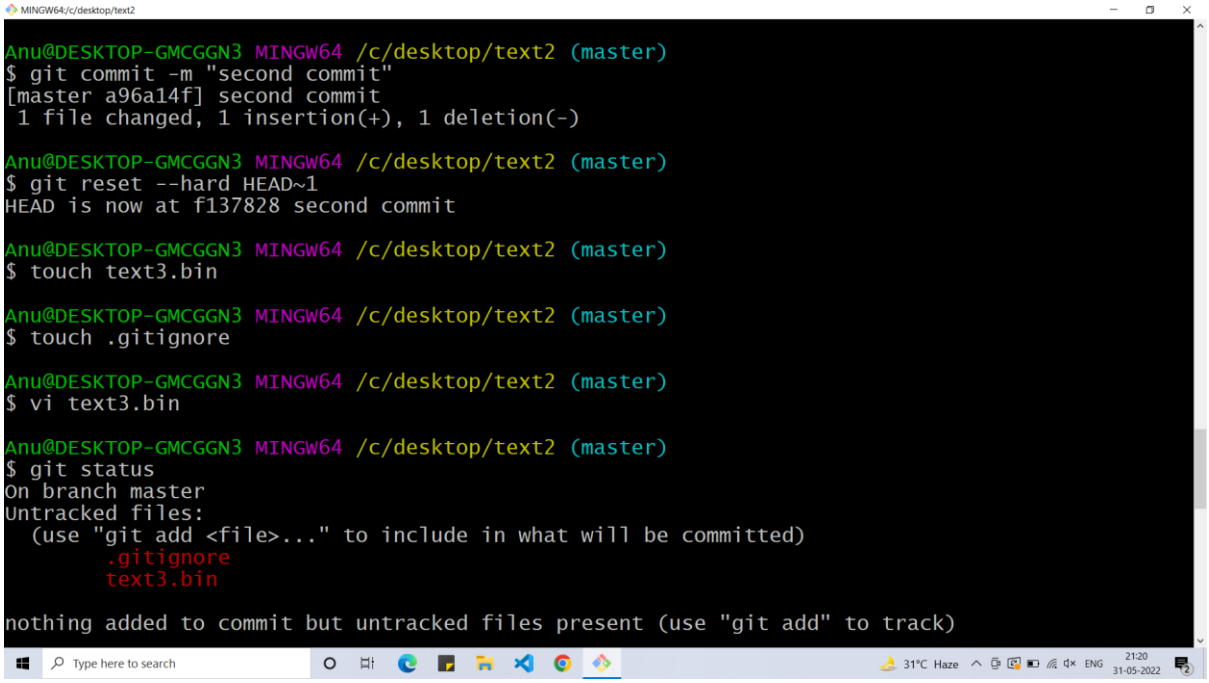

MINGW64:/c/desktop/text2
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ vi text1
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ git add text1.txt
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ git add .
warning: LF will be replaced by CRLF in text1.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in text2.
The file will have its original line endings in your working directory
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ git add .
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ git commit -m "first commit"
[master (root-commit) 970d240] first commit
3 files changed, 6 insertions(+)
create mode 100644 text1
create mode 100644 text1.txt
create mode 100644 text2
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ git status
On branch master
Type here to search
31°C Haze
ENG
21:19
31-05-2022

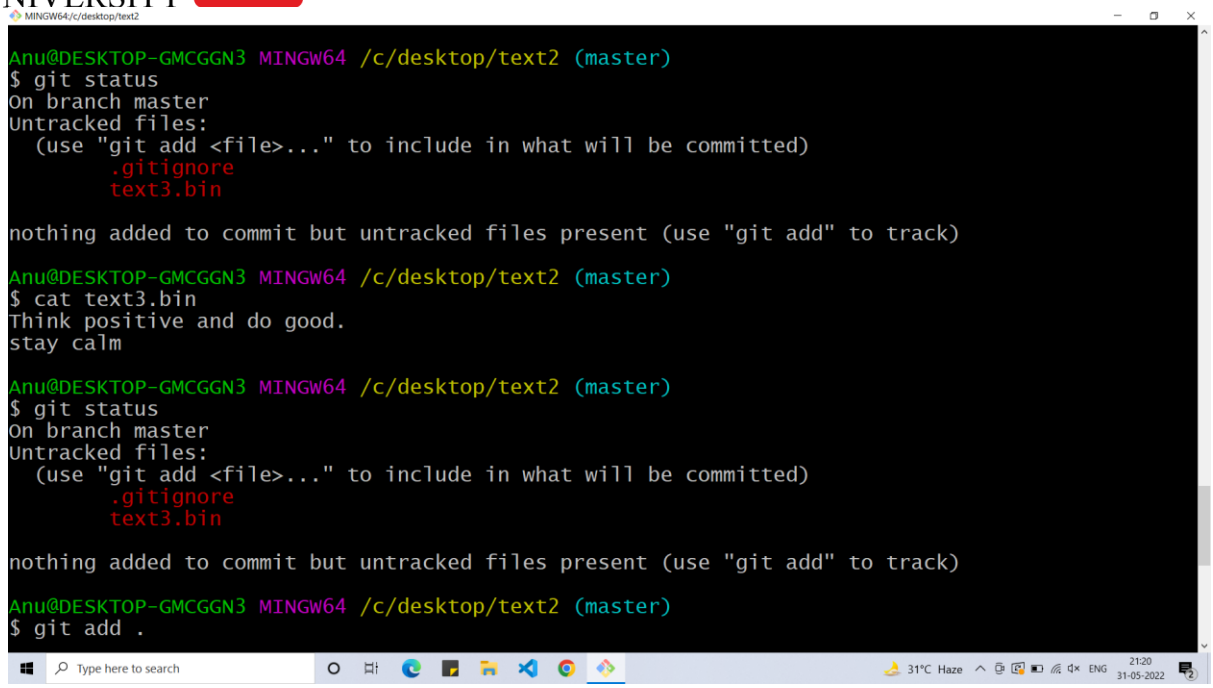

MINGW64:/c/desktop/text2
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ git status
On branch master
nothing to commit, working tree clean
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ git reset --soft HEAD~1
fatal: ambiguous argument 'HEAD~1': unknown revision or path not in the working tree.
Use '--' to separate paths from revisions, like this:
'git <command> [<revision>...] -- [<file>...]'
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ vi text1.txt
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ vi text1
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ git add .
warning: LF will be replaced by CRLF in text1.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in text1.txt.
The file will have its original line endings in your working directory
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ git add .
Type here to search
31°C Haze
ENG
21:19
31-05-2022

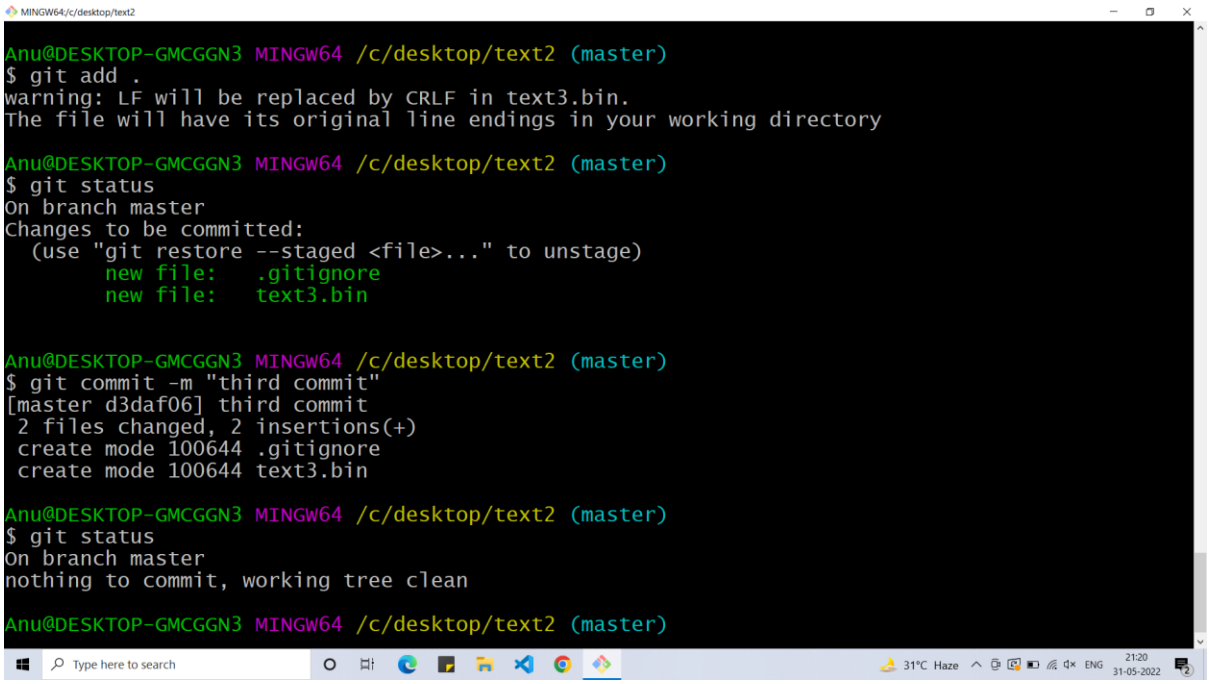

MINGW64:/c/desktop/text2
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ git add .
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ git commit -m "second commit"
[master 77ad4bf] second commit
2 files changed, 5 insertions(+)
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ git status
On branch master
nothing to commit, working tree clean
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ git reset --soft HEAD~1
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ git status
On branch master
Changes to be committed:
(use "git restore --staged <file>..." to unstage)
modified: text1
modified: text1.txt
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ notepad text1
Type here to search
31°C Haze
ENG
21:19
31-05-2022

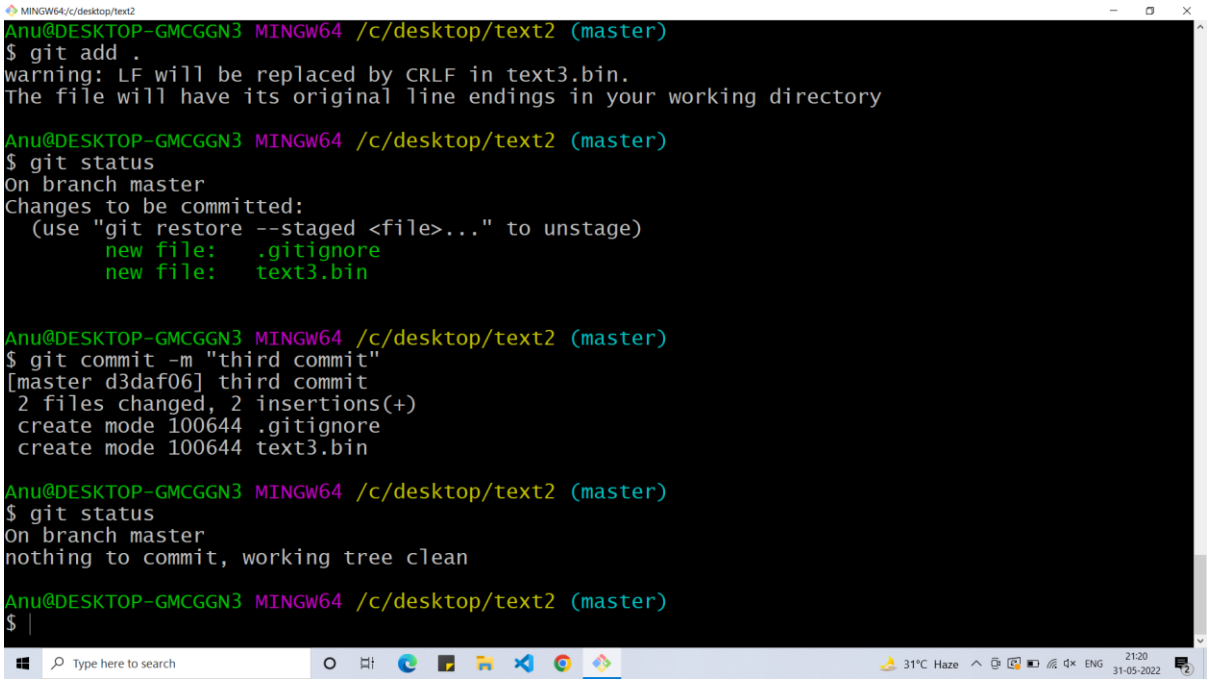

MINGW64:/c/desktop/text2
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ git status
On branch master
Changes to be committed:
(use "git restore --staged <file>..." to unstage)
modified: text1
modified: text1.txt
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ notepad text1
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ git add .
warning: LF will be replaced by CRLF in text1.
The file will have its original line endings in your working directory
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ notepad text1
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ git add .
warning: LF will be replaced by CRLF in text1.
The file will have its original line endings in your working directory
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ git commit -m "second commit"
[master f137828] second commit
Type here to search
31°C Haze
ENG
21:19
31-05-2022


MINGW64:/c/desktop/text2
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ git add .
warning: LF will be replaced by CRLF in text1.
The file will have its original line endings in your working directory
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ git commit -m "second commit"
[master f137828] second commit
2 files changed, 5 insertions(+)
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ git status
On branch master
nothing to commit, working tree clean
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (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: text1.txt
no changes added to commit (use "git add" and/or "git commit -a")
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ git add .
Type here to search
31°C Haze
ENG
21:20
31-05-2022


MINGW64:/c/desktop/text2
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ git commit -m "second commit"
[master a96a14f] second commit
1 file changed, 1 insertion(+), 1 deletion(-)
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ git reset --hard HEAD~1
HEAD is now at f137828 second commit
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ touch text3.bin
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ touch .gitignore
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ vi text3.bin
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ git status
On branch master
Untracked files:
(use "git add <file>..." to include in what will be committed)
.gitignore
text3.bin
nothing added to commit but untracked files present (use "git add" to track)
Type here to search
31°C Haze
ENG
21:20
31-05-2022

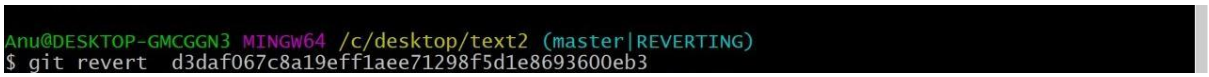

MINGW64:/c/desktop/text2
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ git status
On branch master
Untracked files:
(use "git add <file>..." to include in what will be committed)
.gitignore
text3.bin
nothing added to commit but untracked files present (use "git add" to track)
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ cat text3.bin
Think positive and do good.
stay calm
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ git status
On branch master
Untracked files:
(use "git add <file>..." to include in what will be committed)
.gitignore
text3.bin
nothing added to commit but untracked files present (use "git add" to track)
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ git add .
Type here to search
31°C Haze
ENG
21:20
31-05-2022


MINGW64:/c/desktop/text2
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ git add .
warning: LF will be replaced by CRLF in text3.bin.
The file will have its original line endings in your working directory
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ git status
On branch master
Changes to be committed:
(use "git restore --staged <file>..." to unstage)
new file: .gitignore
new file: text3.bin
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ git commit -m "third commit"
[master d3daf06] third commit
2 files changed, 2 insertions(+)
create mode 100644 .gitignore
create mode 100644 text3.bin
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
$ git status
On branch master
nothing to commit, working tree clean
Anu@DESKTOP-GMCGGN3 MINGW64 /c/desktop/text2 (master)
Type here to search
31°C Haze
ENG
21:20
31-05-2022



6. *Git revert*

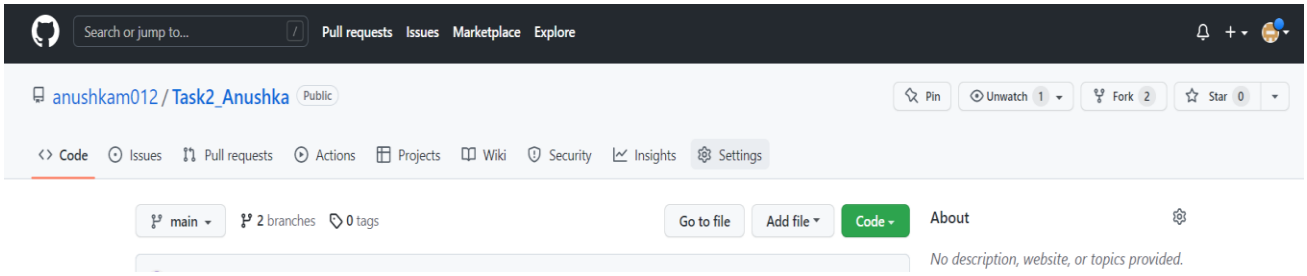
The git revert command is used for undoing changes to a repository's commit history.



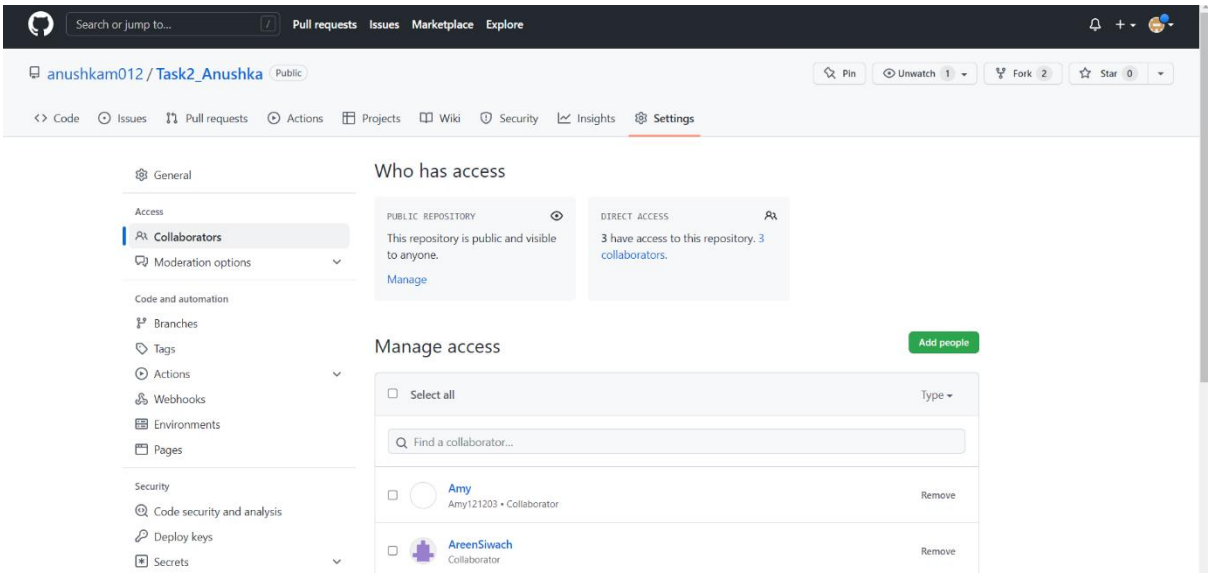
TASK 2-Project

1. Create a distributed Repository and add members in project team

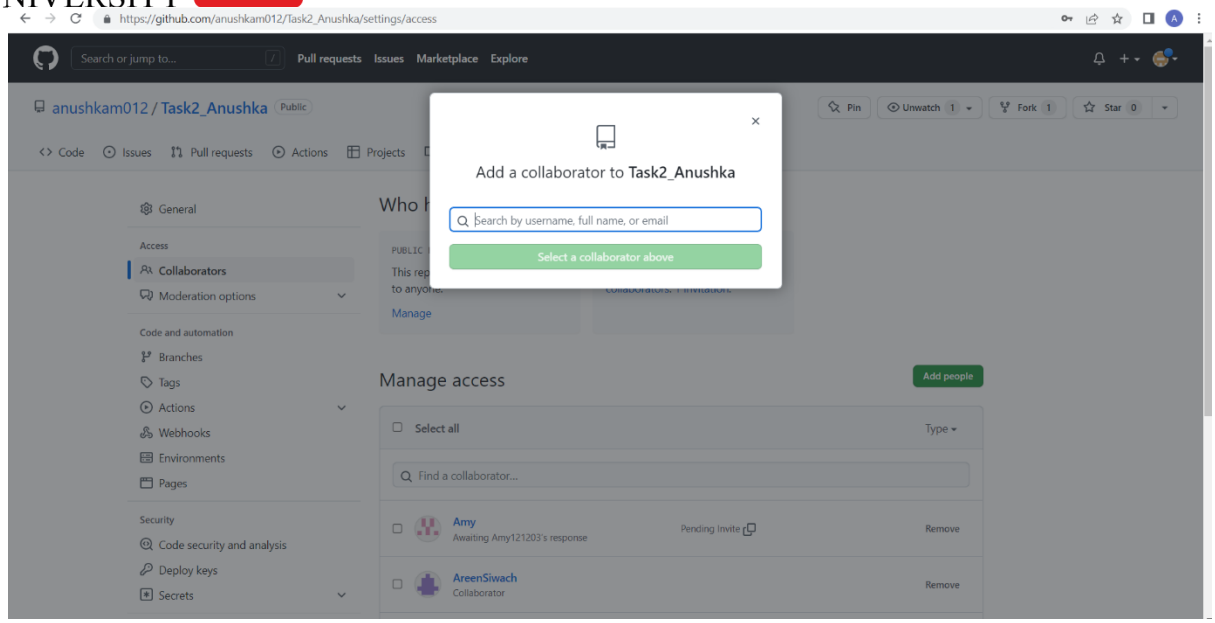
- To add members to your repository open your repository and select settings option in the navigation bar.

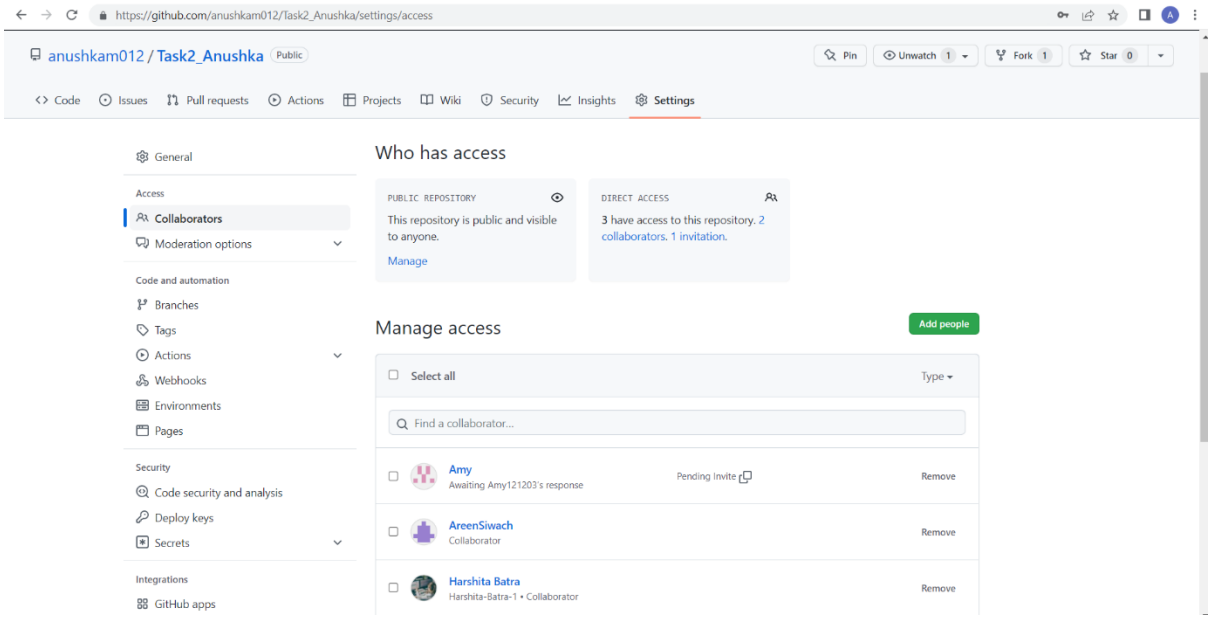


- Click on Collaborators option under the access tab.



- After clicking on collaborators Github asks you to enter your password to confirm the access to the repository.
- After entering the password you can manage access and add/remove team members to your project.
- To add members click on the add people option and search the id of your respective team member.





- To accept the invitation from your team member, open your email registered with Github.
- You will receive an invitation mail from the repository owner. Open the email and click on accept invitation.
- You will be redirected to Github where you can either select to accept or decline the invitation.

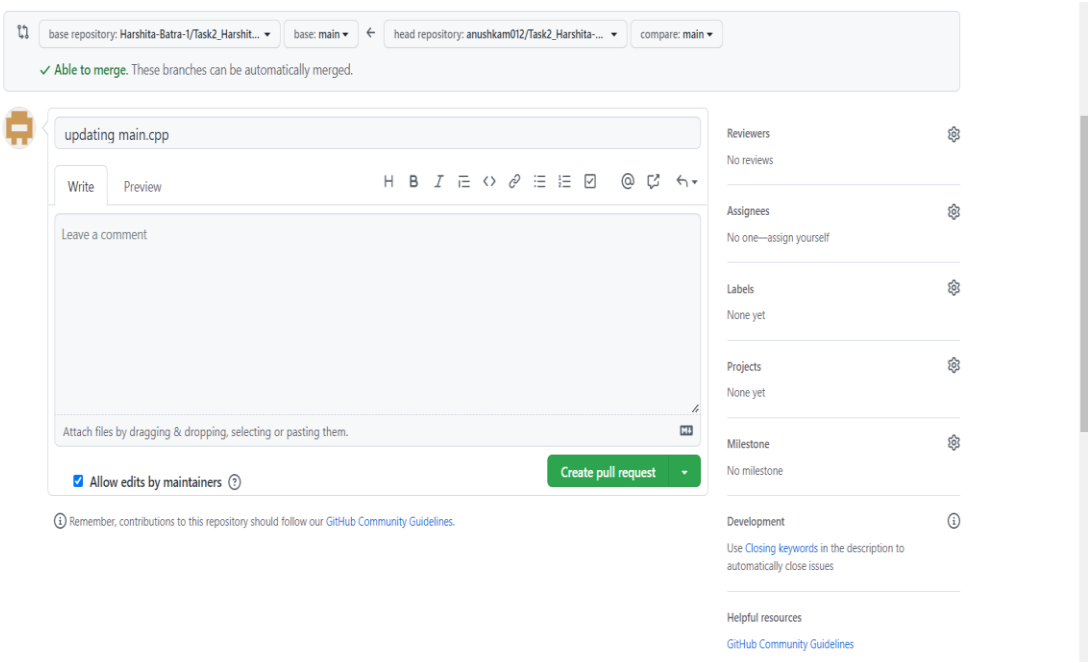
- You will be shown the option that you are now allowed to push.
- Now all members are ready to contribute to the project.

2. **Open and close a pull request.**

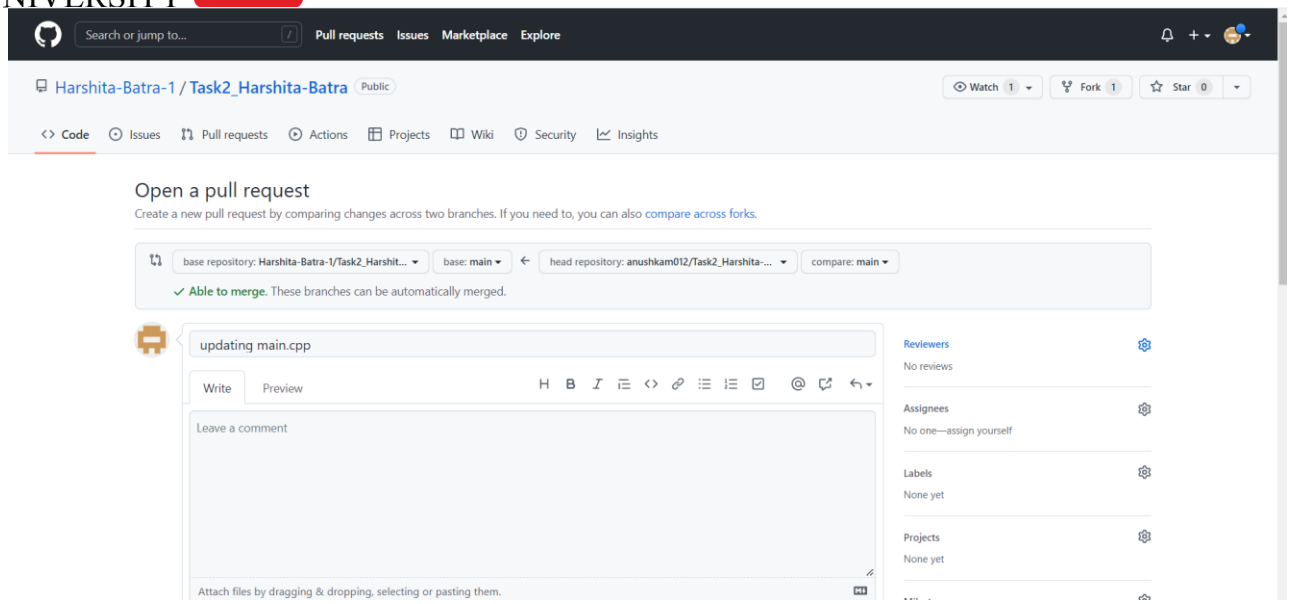
Each project member shall create a pull request on a team members repo and close pull requests generated by team members on own Repo as a maintainer.

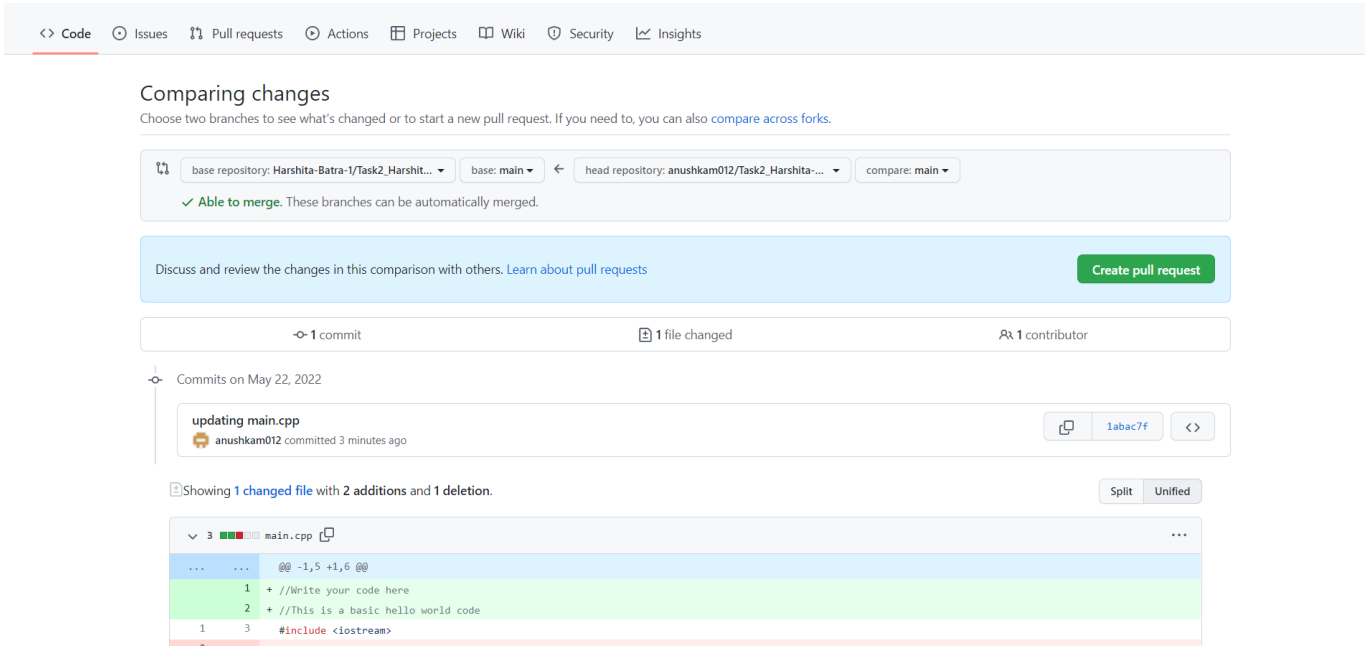
To create a pull request on a team member's repository and close requests by any other team members as a maintainer follow the procedure given below:-

- Do the required changes in the repository, add and commit these changes in the local repository in a new branch.
- Push the modified branch using git push origin *branchname*.
- Open a pull request by following the procedure from the above experiment.
- The pull request will be created and will be visible to all the team members.
- Ask your team member to login to his/her Github account.
- They will notice a new notification in the pull request menu.
- Click on it. The pull request generated by you will be visible to them.

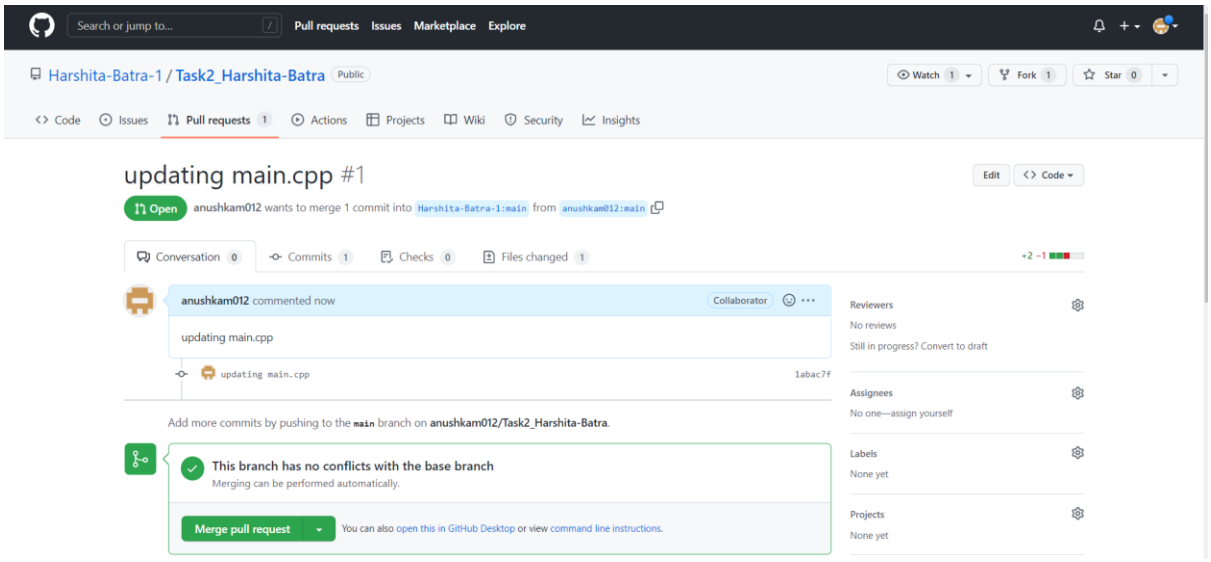


- Click on the pull request. Two option will be available, either to close the pull request or Merge the request with the main branch.

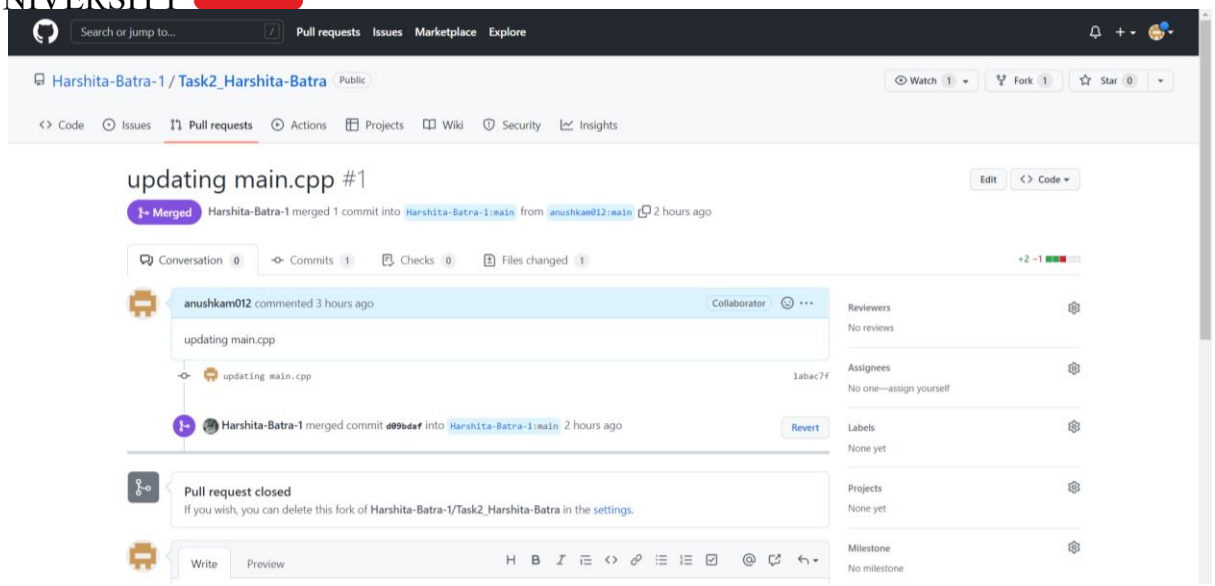

Search or jump to...
Pull requests
Issues
Marketplace
Explore
Harshita-Batra-1 / Task2_Harshita-Batra
Public
Watch 1
Fork 1
Star 0
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Open a pull request
Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks.
base repository: Harshita-Batra-1/Task2_Harshit...
base: main
head repository: anushkam012/Task2_Harshita-...
compare: main
Able to merge. These branches can be automatically merged.
updating main.cpp
Write
Preview
Leave a comment
Attach files by dragging & dropping, selecting or pasting them.
Reviewers
No reviews
Assignees
No one—assign yourself
Labels
None yet
Projects
None yet

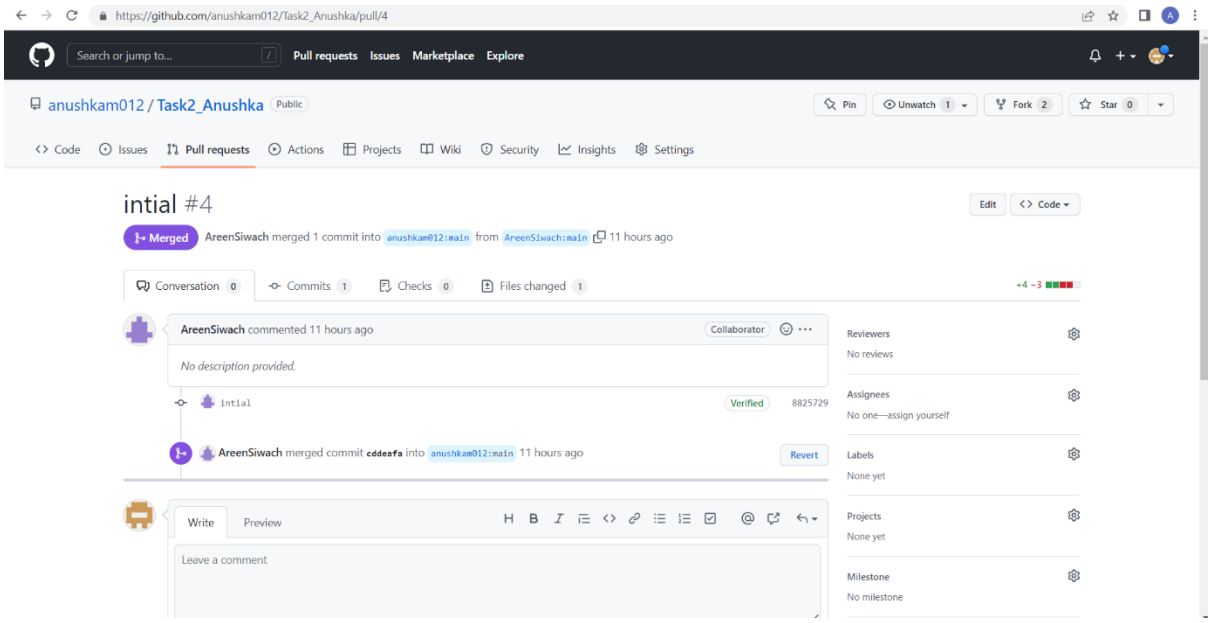

Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Comparing changes
Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks.
base repository: Harshita-Batra-1/Task2_Harshit...
base: main
head repository: anushkam012/Task2_Harshita-...
compare: main
Able to merge. These branches can be automatically merged.
Discuss and review the changes in this comparison with others. Learn about pull requests
Create pull request
1 commit
1 file changed
1 contributor
Commits on May 22, 2022
updating main.cpp
anushkam012 committed 3 minutes ago
1abac7f
Showing 1 changed file with 2 additions and 1 deletion.
Split
Unified
3
main.cpp
@@ -1,5 +1,6 @@
+ //Write your code here
+ //This is a basic hello world code
#include <iostream>

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

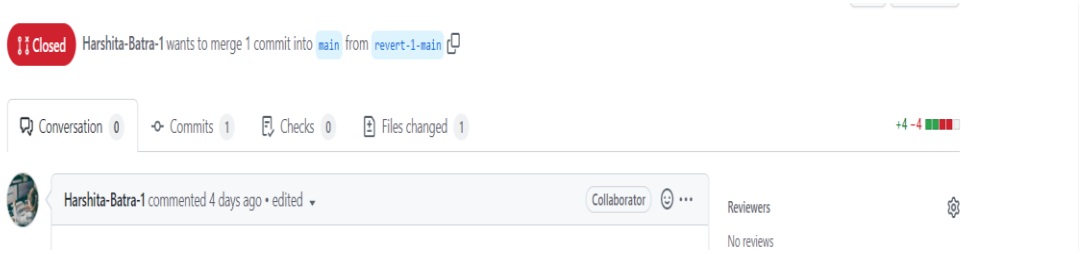


- The process is similar to closing and merging the pull request by you. It simply includes an external party to execute.
- The result of merging the pull request is shown below.


Search or jump to...
Pull requests
Issues
Marketplace
Explore
Harshita-Batra-1 / Task2_Harshita-Batra
Public
Watch 1
Fork 1
Star 0
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
updating main.cpp #1
Edit
Code
Merged
Harshita-Batra-1 merged 1 commit into Harshita-Batra-1:main from anushkam012:main 2 hours ago
Conversation 0
Commits 1
Checks 0
Files changed 1
+2 −1
anushkam012 commented 3 hours ago
Collaborator
updating main.cpp
updating main.cpp
1abac7f
Harshita-Batra-1 merged commit d09bdaf into Harshita-Batra-1:main 2 hours ago
Revert
Pull request closed
If you wish, you can delete this fork of Harshita-Batra-1/Task2_Harshita-Batra in the settings.
Write
Preview
Reviewers
No reviews
Assignees
No one—assign yourself
Labels
None yet
Projects
None yet
Milestone
No milestone


https://github.com/anushkam012/Task2_Anushka/pull/4
Search or jump to...
Pull requests
Issues
Marketplace
Explore
anushkam012 / Task2_Anushka
Public
Pin
Unwatch 1
Fork 2
Star 0
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
intial #4
Edit
Code
Merged
AreenSiwach merged 1 commit into anushkam012:main from AreenSiwach:main 11 hours ago
Conversation 0
Commits 1
Checks 0
Files changed 1
+4 −3
AreenSiwach commented 11 hours ago
Collaborator
No description provided.
intial
Verified
8825729
AreenSiwach merged commit cddeafa into anushkam012:main 11 hours ago
Revert
Write
Preview
Leave a comment
Reviewers
No reviews
Assignees
No one—assign yourself
Labels
None yet
Projects
None yet
Milestone
No milestone

The result of closing the request is shown below



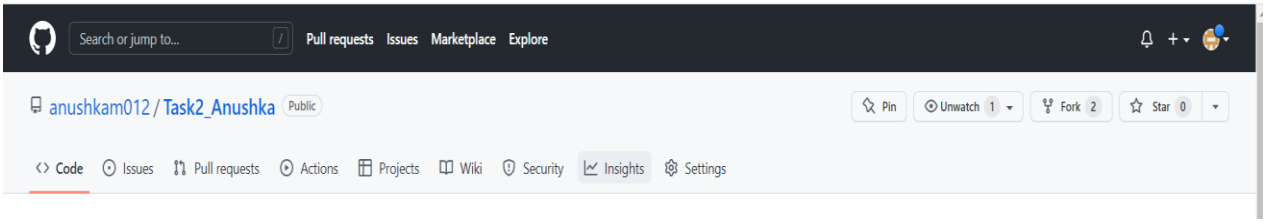
- Thus, we conclude opening and closing of pull request. We also conclude merging of the pull request to the main branch.

3. Publish and print network graphs

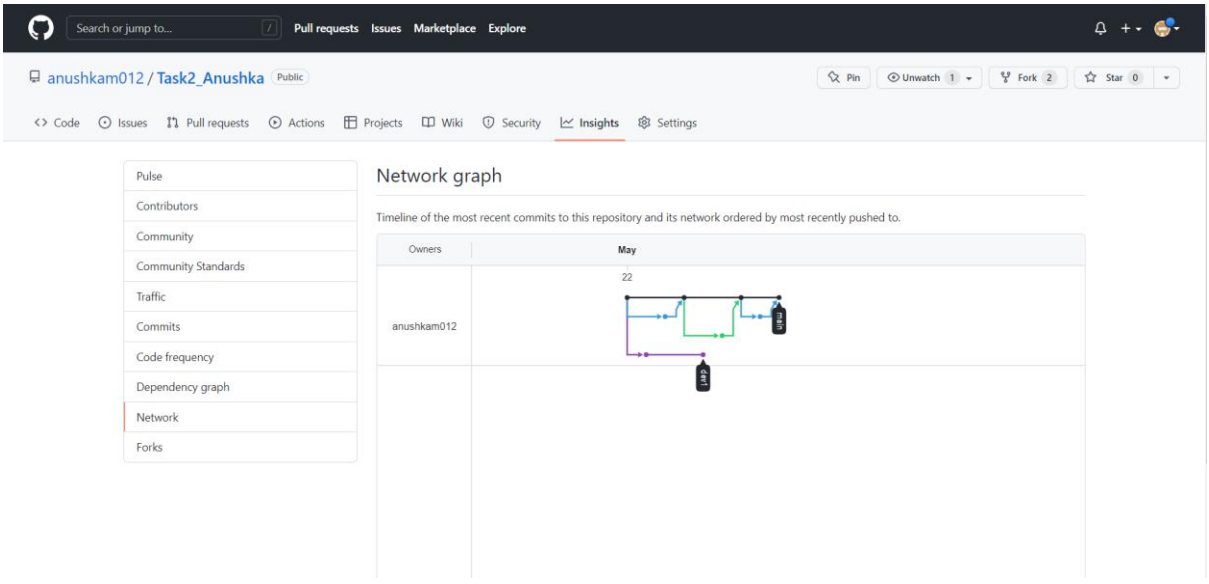
The network graph displays the branch history of the entire repository network, including branches of the root repository and branches of forks that contain commits unique to the network.

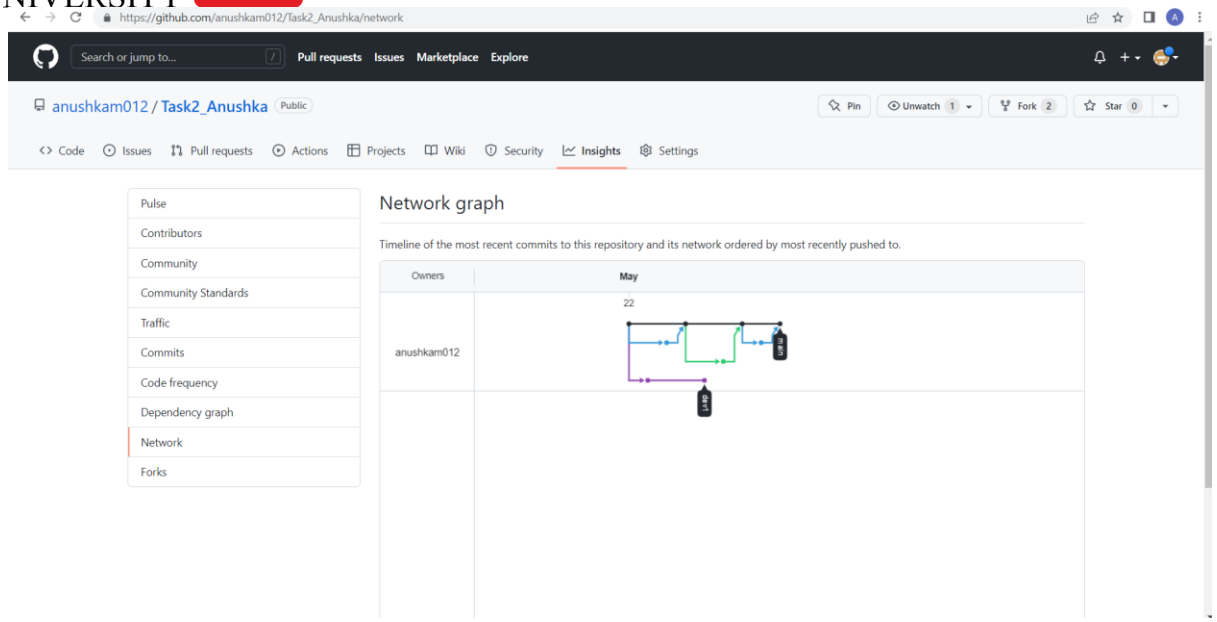
Accessing the network graph

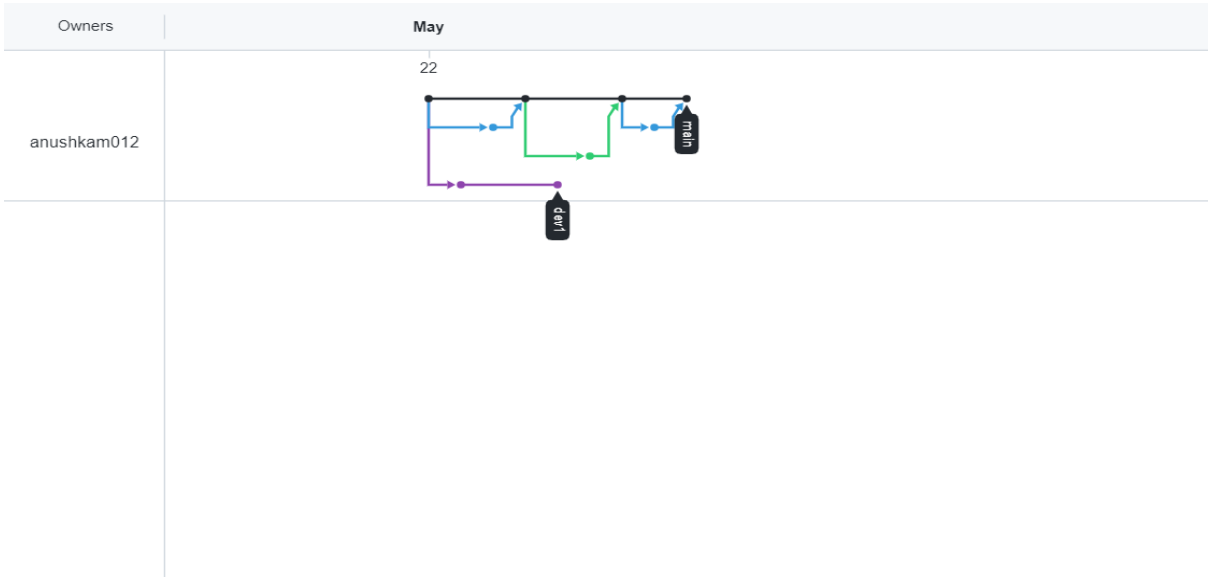
1. On GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click **Insights**.



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




https://github.com/anushkam012/Task2_Anushka/network
Search or jump to...
Pull requests
Issues
Marketplace
Explore
anushkam012 / Task2_Anushka
Public
Pin
Unwatch 1
Fork 2
Star 0
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
Pulse
Contributors
Community
Community Standards
Traffic
Commits
Code frequency
Dependency graph
Network
Forks
Network graph
Timeline of the most recent commits to this repository and its network ordered by most recently pushed to.
Owners
May
22
anushkam012
main
day1


Owners
May
22
anushkam012
main
day1

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**.