

Subject Name: **Source Code Management**

Subject Code: **CS181**

Cluster: **Beta**

Department: **CSE**



Submitted by-

Name: Aditi Rana

Roll No. : 2110990080

Submitted to- Dr. Monit Kapoor

INDEX

Serial Number	Program Title	Page Number
1.	Task 1.1	3-21
2.	Task 1.2	22-34
3.	Task 2	35-39

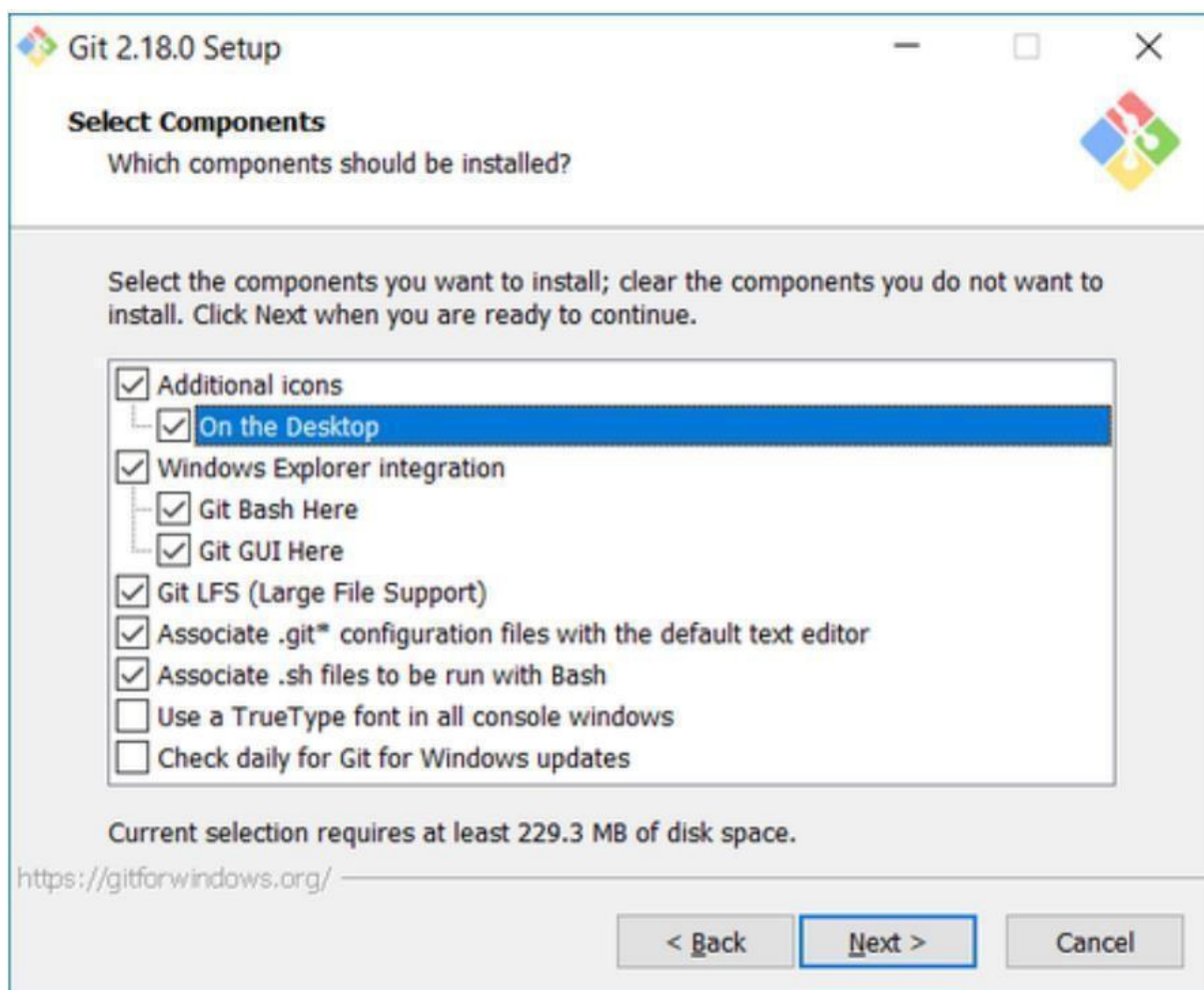
Experiment No. 01

Aim: Setting up the git client.

Git Installation: Download the Git installation program (Windows, Mac, or Linux) from [Git - Downloads \(git-scm.com\)](https://git-scm.com/downloads).

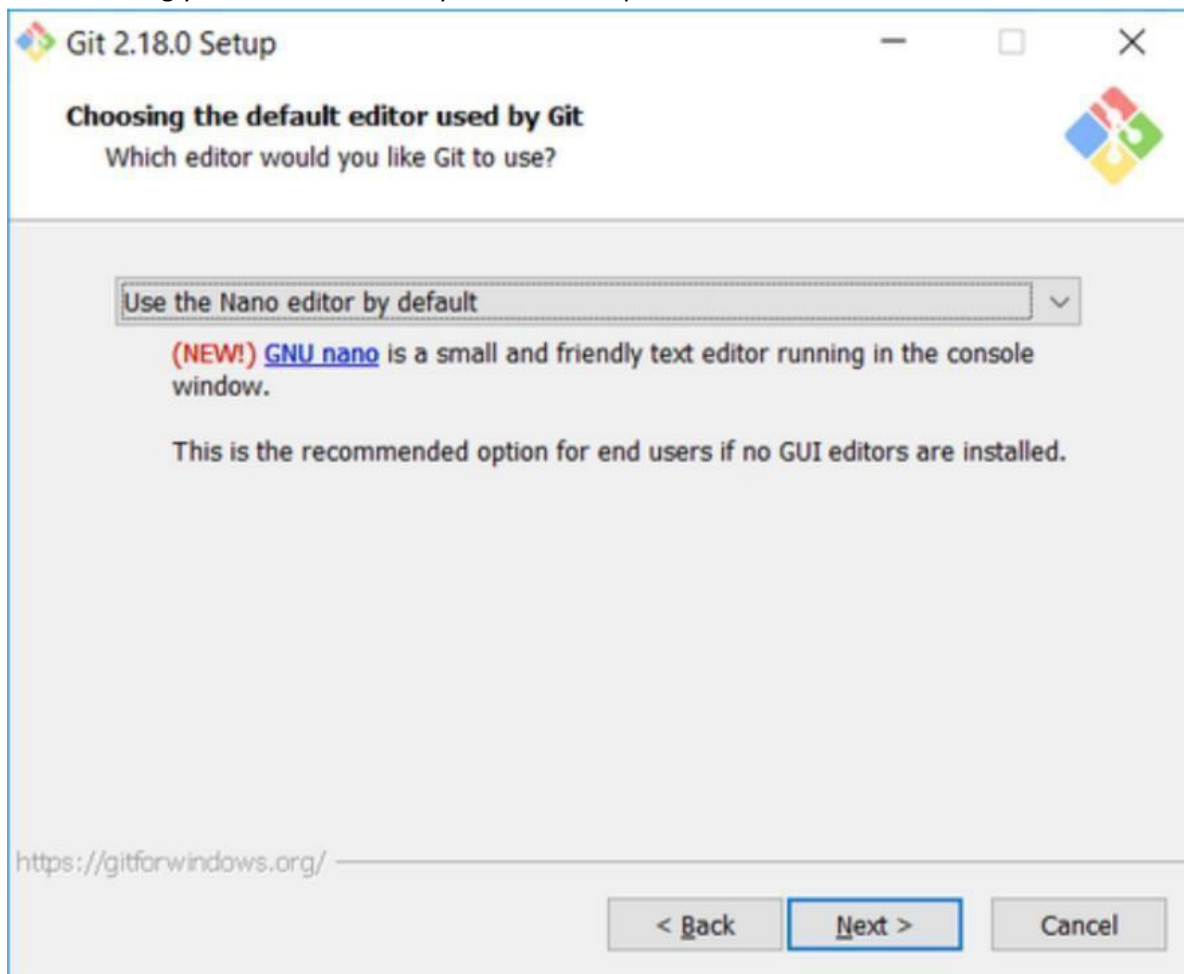
When running the installer, various screens appear (Windows screens shown). Generally, you can accept the default selections, except in the screens below where you do not want the default selections:

In the Select Components screen, make sure Windows Explorer Integration is selected as shown:



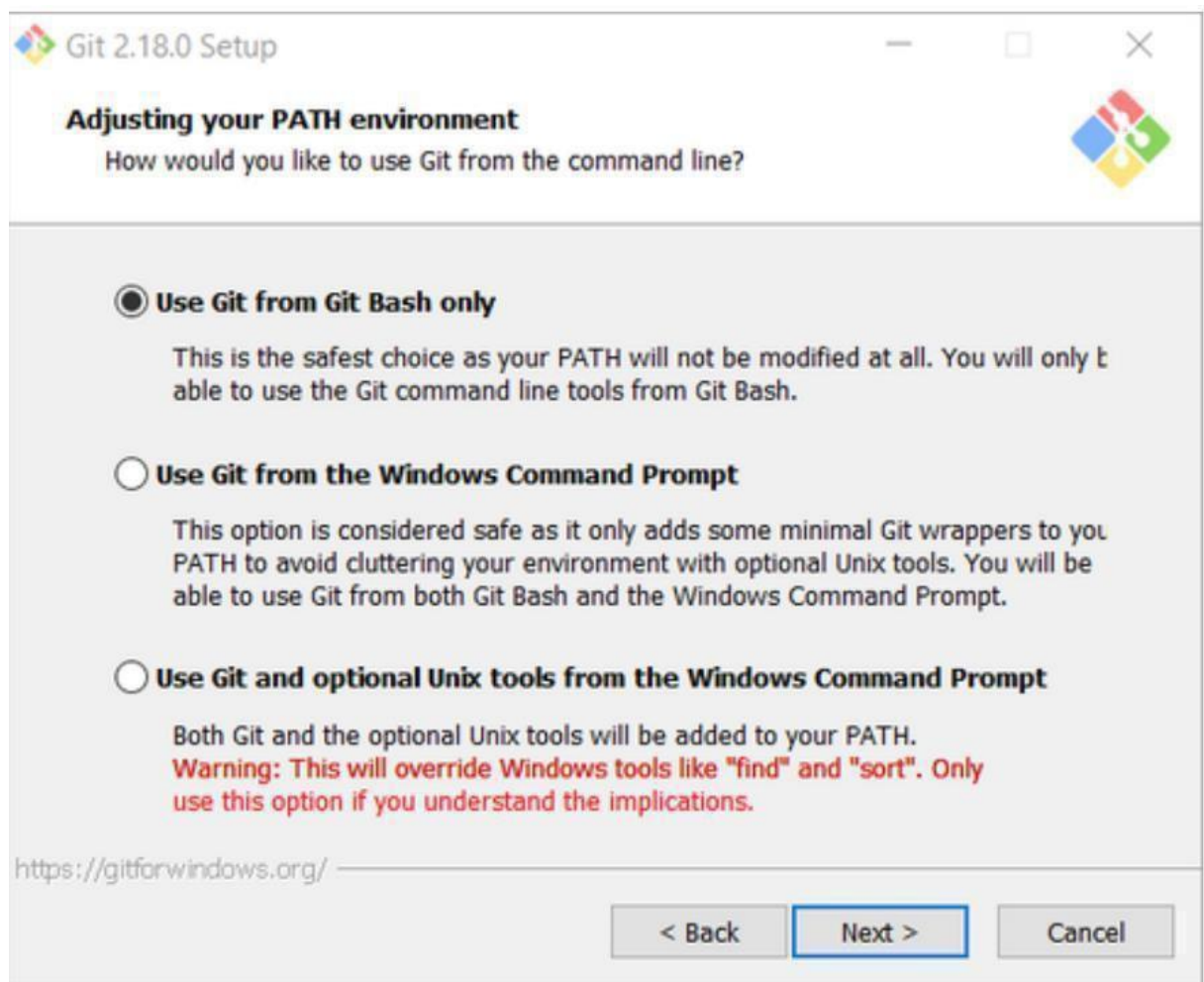
In the choosing the default editor is used by Git dialog, it is strongly recommended that you DO NOT select default VIM editor- it is challenging to learn how to use it, and there are better

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).

Edit this file and add the lines below (just copy/paste them from this screen); these are patterns for files to be ignored (taken from examples provided at <https://github.com/github/gitignore>.)

```
#Lines (like this one) that begin with # are comments; all other lines are rules

# common build products to be ignored at MSOE
*.o
*.obj
*.class
*.exe

# common IDE-generated files and folders to ignore
workspace.xml
bin
/
out
/
.classpath
# uncomment following for courses in which Eclipse .project files are not checked
in # .project

#ignore automatically generated files created by some common applications,
operating systems
*.bak
*.log
*.ldb
~*
.DS_Store*
.*
Thumbs.d
b

# Any files you do not want to ignore must be specified starting with ! # For example,
if you didn't want to ignore .classpath, you'd uncomment the following rule: #
!.classpath
```

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 project-specific 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 "Git GUI 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 "aditi0080.be21@chitkara.edu.in"`

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 "rana-aditi"`

Git uses this to 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.

Aim

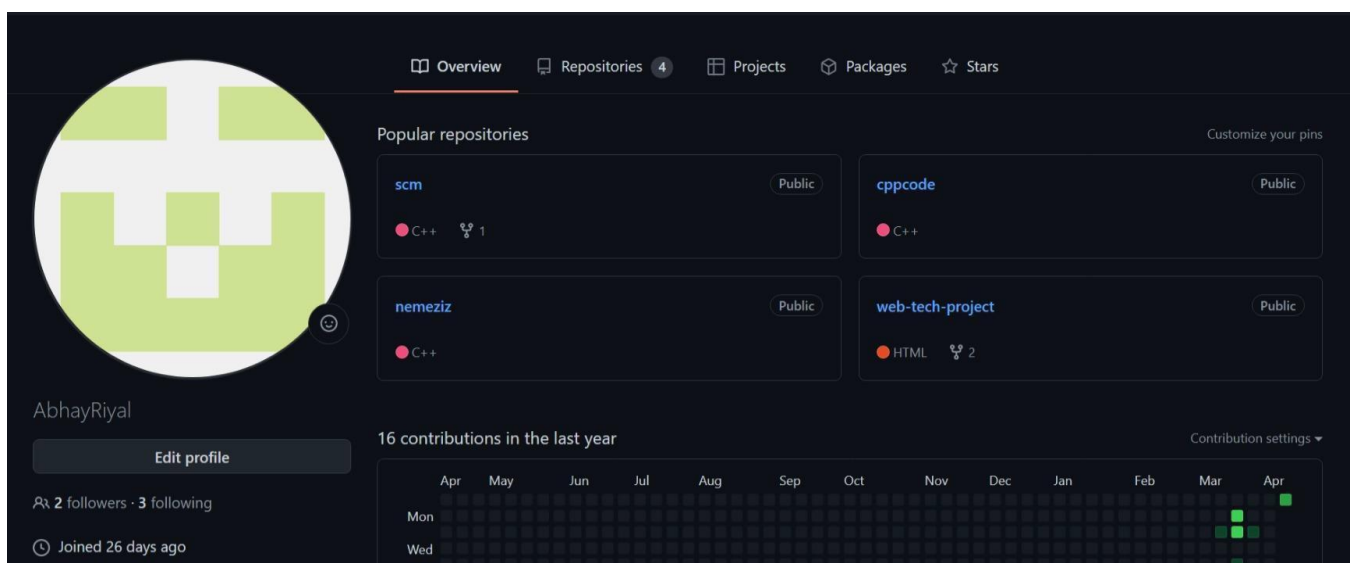
Setting up GitHub Account

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.

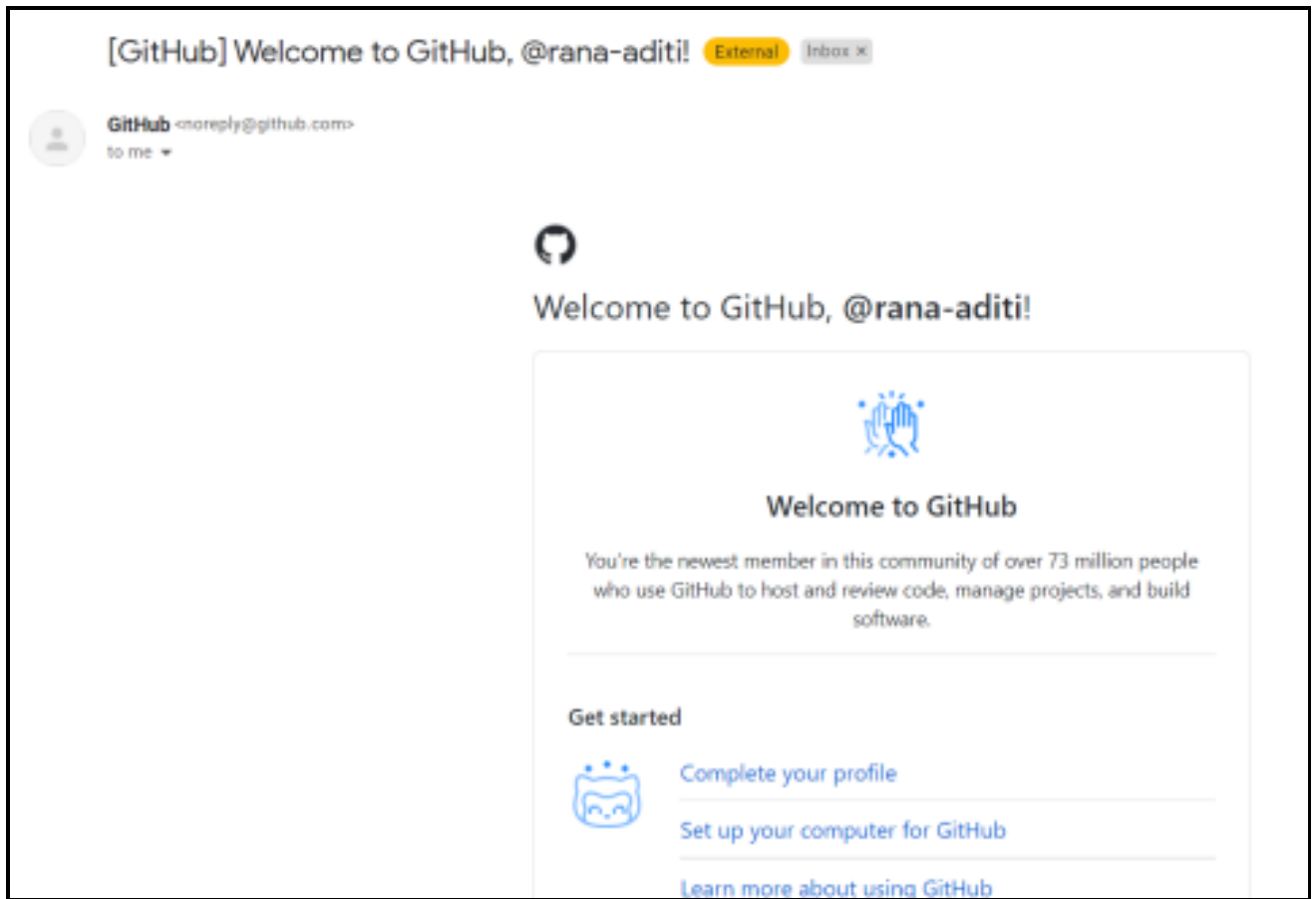
To keep your GitHub account secure you should use a strong and unique password. For more information, see "[Creating a strong password](#)".



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](#)".

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 twofactor authentication.”](#)

Two-factor authentication



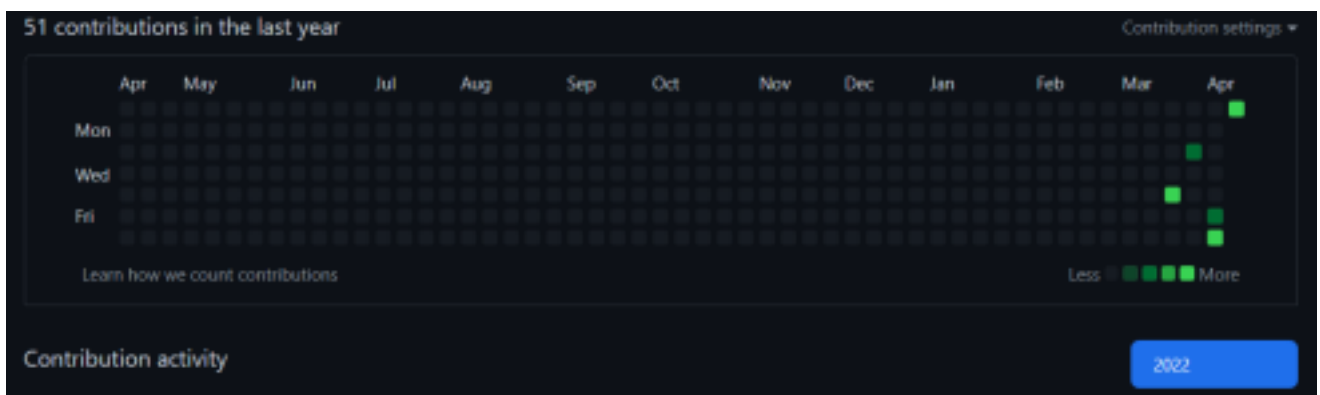
Two factor authentication is not enabled yet.

Two-factor authentication adds an additional layer of security to your account by requiring more than just a password to sign in.

[Enable two-factor authentication](#)

[Learn more](#)

4. **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](#)."




Experiment No. 03

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.



```
MINGW64/d/scmproject
Dell@DESKTOP-0B7Q4VN MINGW64 /
$ cd d:

Dell@DESKTOP-0B7Q4VN MINGW64 /d
$ mkdir scmproject

Dell@DESKTOP-0B7Q4VN MINGW64 /d
$ git init
Initialized empty Git repository in D:/scmproject/.git/
```

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.

```

MINGW64/d/scmproject
Dell@DESKTOP-OB7Q4VN MINGW64 /d/scmproject (master)
$ cat file.txt

hello

Dell@DESKTOP-OB7Q4VN MINGW64 /d/scmproject (master)
$ git status
warning: could not open directory '$RECYCLE.BIN/S-1-5-21-2771725337-3240630128-580306671-1001/': Permission denied
warning: could not open directory '$RECYCLE.BIN/S-1-5-21-2771725337-3240630128-580306671-500/': Permission denied
warning: could not open directory 'Recovery/': Permission denied
warning: could not open directory 'System Volume Information/': Permission denied
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    ../$RECYCLE.BIN/
    ../C++/
    ../ENGLISH ACTIVITY.pptx
    ../Mock Test Parents' Notice 19-10-2020.docx
    ../Rudra's Documents/
    ../awdcodes/
    ../cn solutions/
    ../codes/
    ../git/
    ../oops/
    ../

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

```

Basic Git commit

The `git commit` command captures a snapshot of the project's currently staged changes.

```

MINGW64/d/scmproject
$ git add .
warning: LF will be replaced by CRLF in scmproject/file.txt.
The file will have its original line endings in your working directory

Dell@DESKTOP-OB7Q4VN MINGW64 /d/scmproject (master)
$ git commit -m"first commit"
[master (root-commit) a5bd62f] first commit
1 file changed, 3 insertions(+)
create mode 100644 scmproject/file.txt

```

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:

```
 Dell@DESKTOP-0B7Q4VN MINGW64 /d/scmproject (master)
$ git log
commit a5bd62f4d8d366ff746b689adb2941fe6fbbaa724 (HEAD -> master)
Author: rana-aditi <aditi0080.be21@chitkara.edu.in>
Date:   Sun Apr 10 18:26:36 2022 +0530

    first commit

Dell@DESKTOP-0B7Q4VN MINGW64 /d/scmproject (master)
$
```


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 staging 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.

 MINGW64:/d/scmproject

```
De1l@DESKTOP-OB7Q4VN MINGW64 /d/scmproject (master)
$ git branch first-branch

De1l@DESKTOP-OB7Q4VN MINGW64 /d/scmproject (master)
$ git branch
feature
first-branch
* master

De1l@DESKTOP-OB7Q4VN MINGW64 /d/scmproject (master)
$ git checkout first-branch
Switched to branch 'first-branch'

De1l@DESKTOP-OB7Q4VN MINGW64 /d/scmproject (first-branch)
$
```

MINGW64:/d/scmproject

```
De11@DESKTOP-OB7Q4VN MINGW64 /d/scmproject (master)
$ git branch first-branch
```

```
De11@DESKTOP-OB7Q4VN MINGW64 /d/scmproject (master)
$ git branch
  feature
  first-branch
* master
```

```
De11@DESKTOP-OB7Q4VN MINGW64 /d/scmproject (master)
$ git checkout first-branch
Switched to branch 'first-branch'
```

```
De11@DESKTOP-OB7Q4VN MINGW64 /d/scmproject (first-branch)
$ git merge
fatal: No remote for the current branch.
```

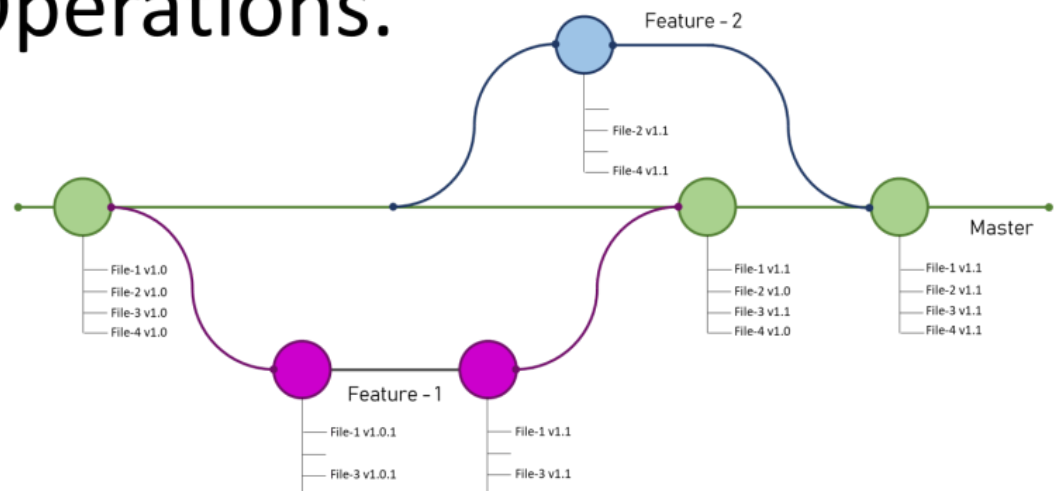
```
De11@DESKTOP-OB7Q4VN MINGW64 /d/scmproject (first-branch)
$ git merge first-branch
Already up to date.
```

```
De11@DESKTOP-OB7Q4VN MINGW64 /d/scmproject (first-branch)
$ git merge feature
Already up to date.
```

```
De11@DESKTOP-OB7Q4VN MINGW64 /d/scmproject (first-branch)
$ git merge master
Already up to date.
```

```
De11@DESKTOP-OB7Q4VN MINGW64 /d/scmproject (first-branch)
$ |
```

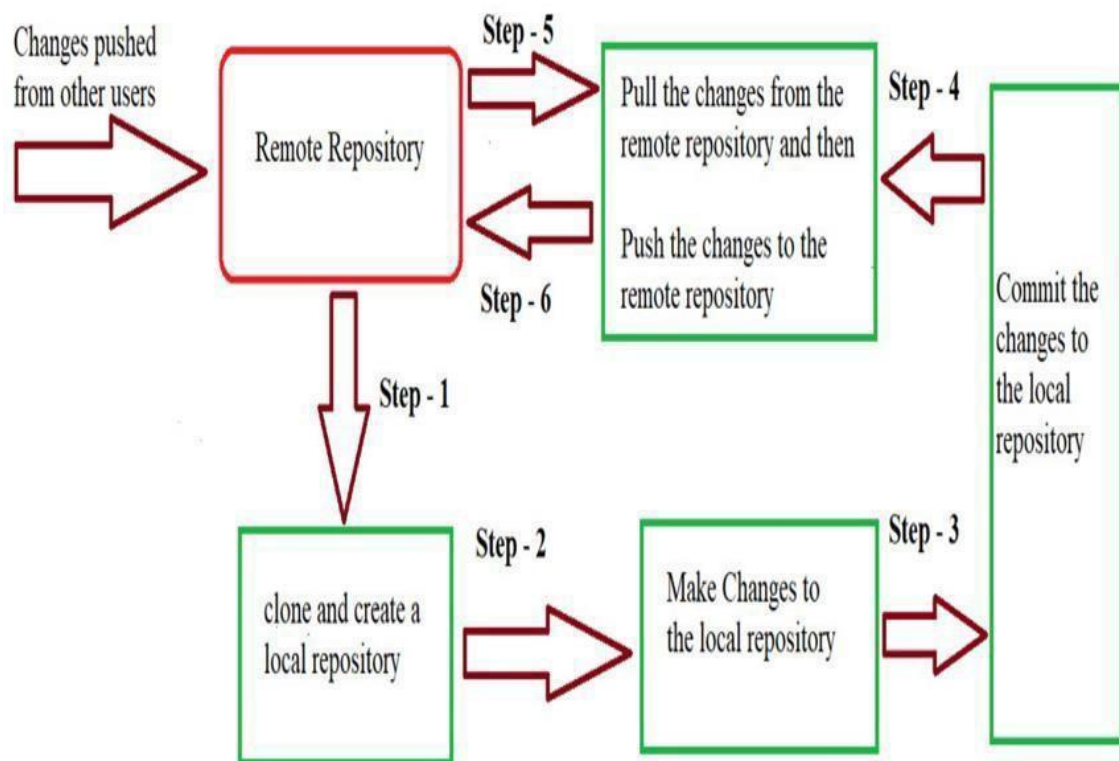
GIT Branch and its Operations.



Experiment No. 05

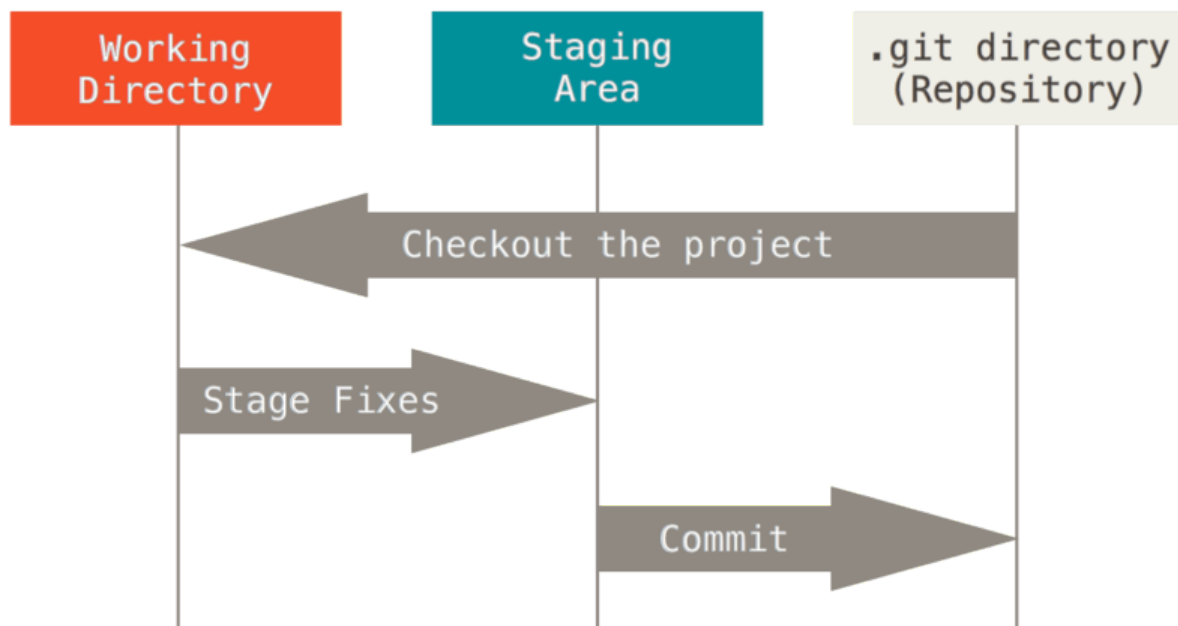
Aim: Git lifecycle description

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 own 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 our 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 our 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 <Message>`.

Experiment No. 06

Aim: Add collaborators on GitHub Repo

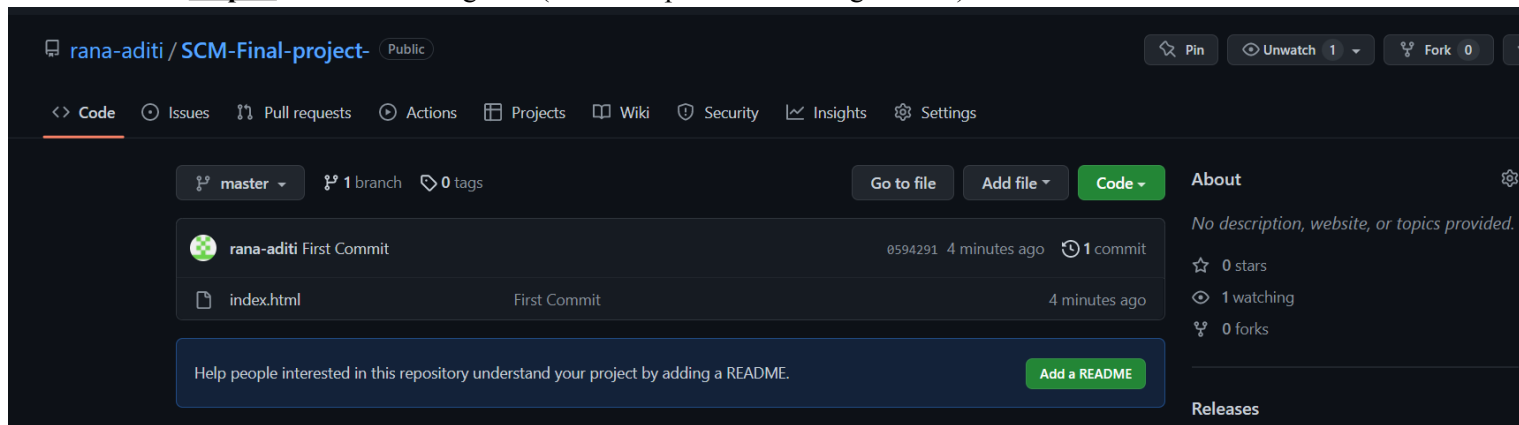
In GitHub, We can invite other GitHub users to become collaborators to our private repositories(which expire after 7 days if not accepted, restoring any unclaimed licenses). Being a collaborator, of a personal repository you can pull (read) the contents of the repository and push (write) changes to the repository. You can add unlimited collaborators on public and private repositories(with some per-day limit restrictions). But, in a private repository, the owner of the repo can only grant write access to the collaborators, and they can't have read-only access.

Steps to add collaborators:

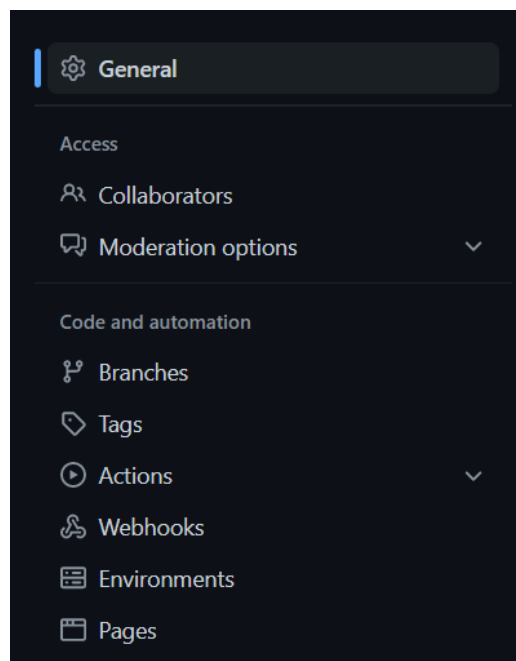
Step 1: Get the GitHub username of people who you want to add as collaborators, in case they aren't on GitHub, they will need to sign-up.

Step 2: Open the repo on which you wish to add collaborators

Step 3: Go to the settings tab (The last option in the image below)



Step 4: Select Collaborators on left side-bar.

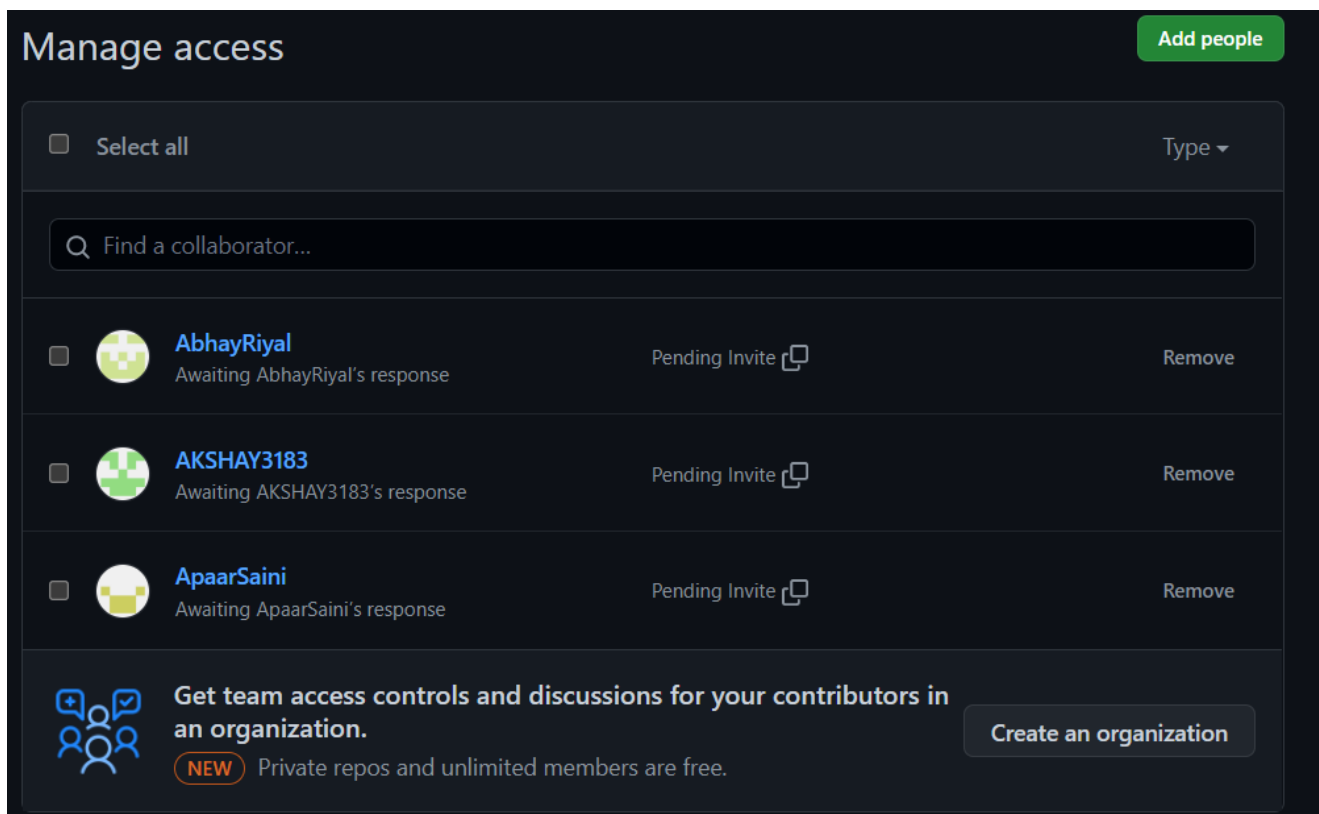


Step 5: It might ask your GitHub account password saying “Sudo mode” is going to be activated, if asked, enter your password.

Step 6: Click Add People

Step 7: A pop-up will appear in which you need to enter the username or email of the person you want to add as a collaborator

Step 8: Insert their username/email and click on the add button.



The screenshot displays the 'Manage access' page in GitHub. At the top right is a green 'Add people' button. Below the header, there's a 'Select all' checkbox and a 'Type' dropdown menu. A search bar labeled 'Find a collaborator...' is present. The main area lists three pending invitations:

Collaborator	Status	Action
<input type="checkbox"/> AbhayRiyal Awaiting AbhayRiyal's response	Pending Invite	Remove
<input type="checkbox"/> AKSHAY3183 Awaiting AKSHAY3183's response	Pending Invite	Remove
<input type="checkbox"/> ApaarSaini Awaiting ApaarSaini's response	Pending Invite	Remove

At the bottom, there's a promotional banner for creating an organization, stating 'Get team access controls and discussions for your contributors in an organization.' with a 'NEW' tag and a 'Create an organization' button. It also mentions 'Private repos and unlimited members are free.'

Step 9: After this, they will receive an email that they are being invited to a GitHub repository. Once they accept the invitation they will be added as a collaborator to the Repository!

AKSHAY3183 invited you to AKSHAY3183/Scm-Project

External

Inbox x



AKSHAY3183 <noreply@github.com>
to me ▾

4:52 PM (3 minutes ago)



GitHub



@AKSHAY3183 has invited you to collaborate on
the
AKSHAY3183/Scm-Project repository

You can [accept or decline](#) this invitation. You can also head over to
<https://github.com/AKSHAY3183/Scm-Project> to check out the repository or visit
[@AKSHAY3183](#) to learn a bit more about them.

This invitation will expire in 7 days.

[View invitation](#)

Experiment No. 07

Aim: Fork and Commit

Forking a repository means creating a copy of the repo. When you fork a repo, you create your own copy of the repository on your GitHub account. This is done for the following reasons:

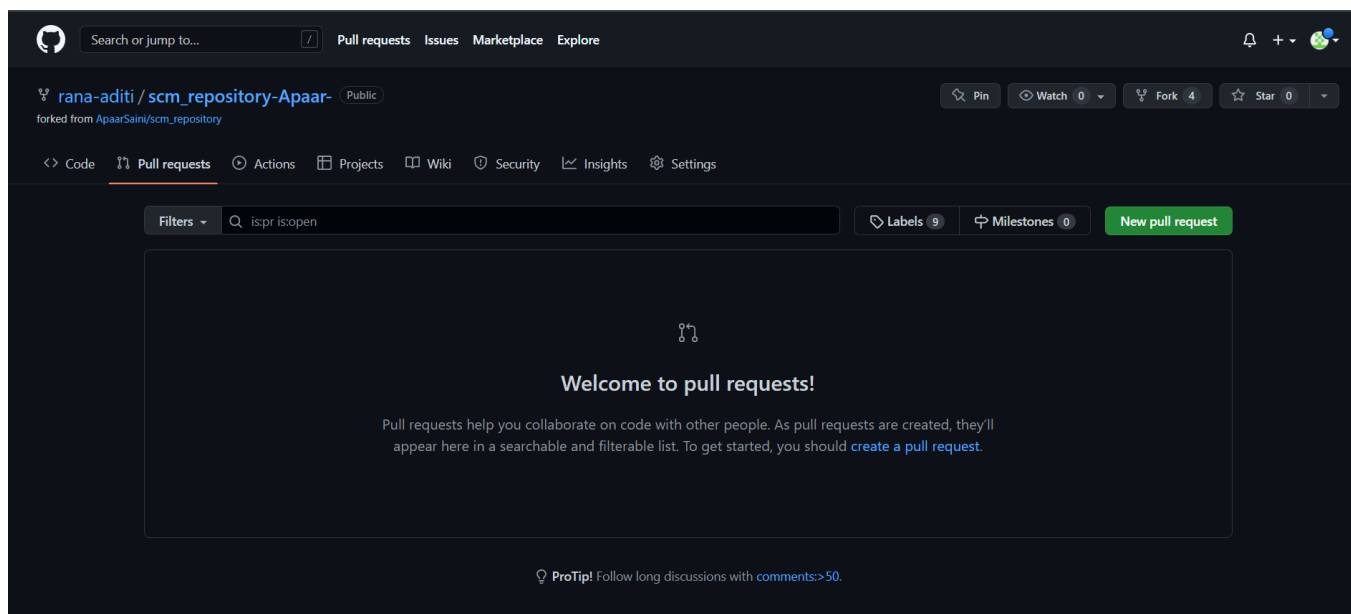
1. You have your own copy of the project on which you may test your own changes without changing the original project.
2. This helps the maintainer of the project to better check the changes you made to the project and has the power to either accept, reject or suggest something.
3. When you clone an Open Source project, which isn't yours, you don't have the right to push code directly into the project.

Steps to fork a repository:

1. Go to the repository that you wish to fork on GitHub, and click the fork button on top as shown in the image below



2. Once you click fork, you can optionally add a description and then click the “Create fork” button.



3. On clicking “Create fork” it will redirect you to the page of the fork of the repo on your account on which you can make changes without affecting the original repo!

MAKING COMMITS:

Commit is like a snapshot of your repository. These commits are snapshots of your entire repository at specific times. You should make new commits often, based on logical units of change.

Over time, commits should tell a story of the history of your repository and how it came to be the way that it currently is.

Commits include lots of metadata in addition to the contents and message, like the author, timestamp, and more.

Steps to make a commit in the forked repo:

1. Click on any file that you want to change in the original repo, I am taking the README file as an example here
2. On the file's page click the pencil icon to start editing as shown in the image below
3. You can write what you want to add in the file
4. When you are adding stuff to a file you can scroll down where there is the option to add commit title and its description
5. You can give it a title and then click the “Commit changes” button on the bottom to complete the commit!

Experiment No. 08

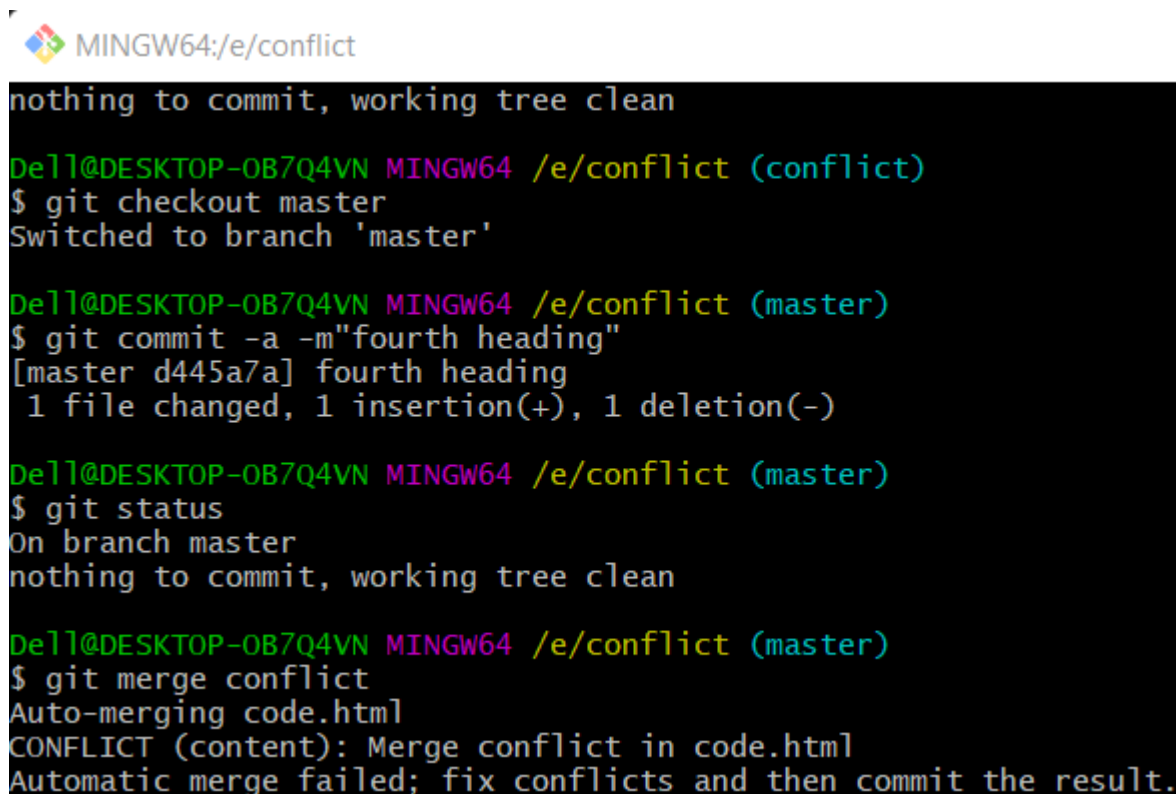
Aim: Merge and Resolve conflicts created due to own activity and collaborator's activity.

There are a few steps that could reduce the steps needed to resolve merge conflicts in Git.

1. The easiest way to resolve a conflicted file is to open it and make any necessary changes.
2. After editing the file, we can use the git add a command to stage the new merged content.
3. The final step is to create a new commit with the help of the git commit command.
4. Git will create a new merge commit to finalize the merge

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

1. Merge conflict occurs.



```
MINGW64:/e/conflict

nothing to commit, working tree clean

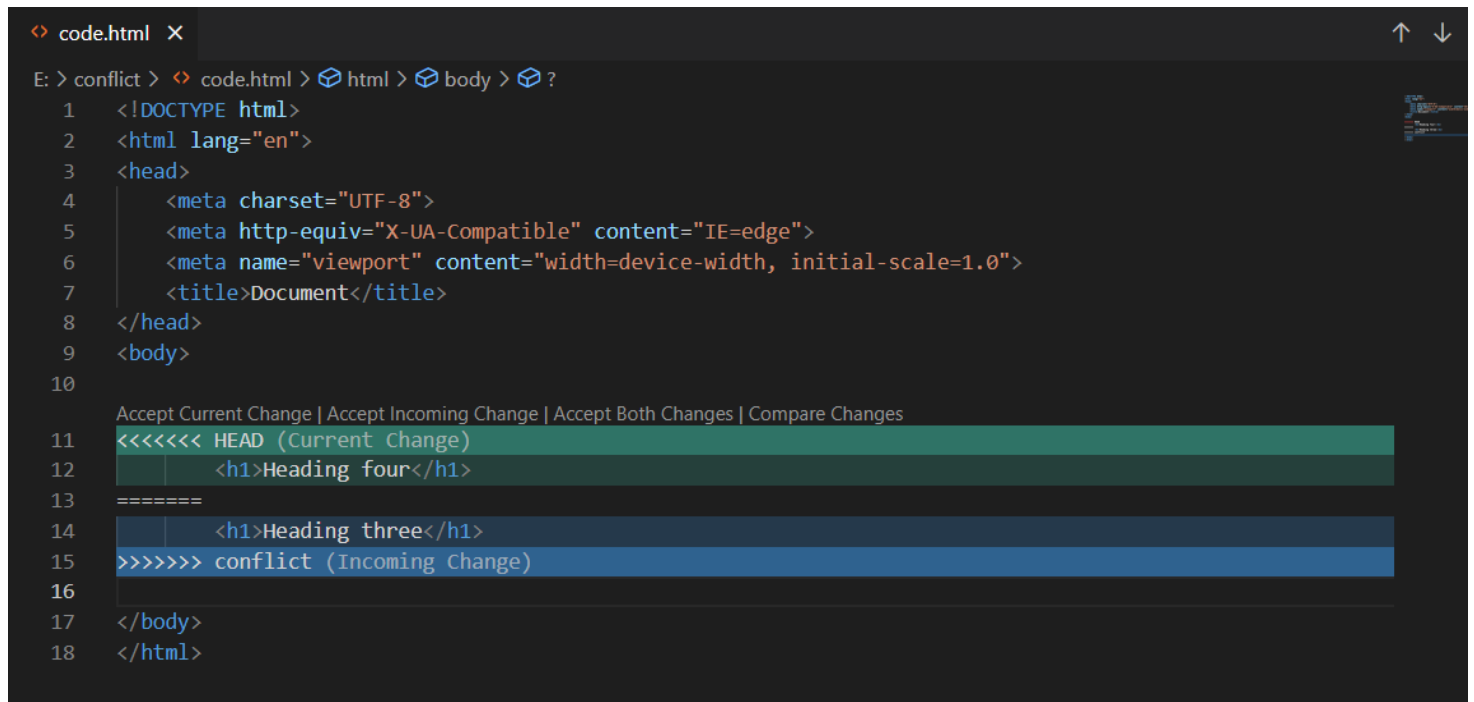
Dell@DESKTOP-OB7Q4VN MINGW64 /e/conflict (conflict)
$ git checkout master
Switched to branch 'master'

Dell@DESKTOP-OB7Q4VN MINGW64 /e/conflict (master)
$ git commit -a -m"fourth heading"
[master d445a7a] fourth heading
1 file changed, 1 insertion(+), 1 deletion(-)

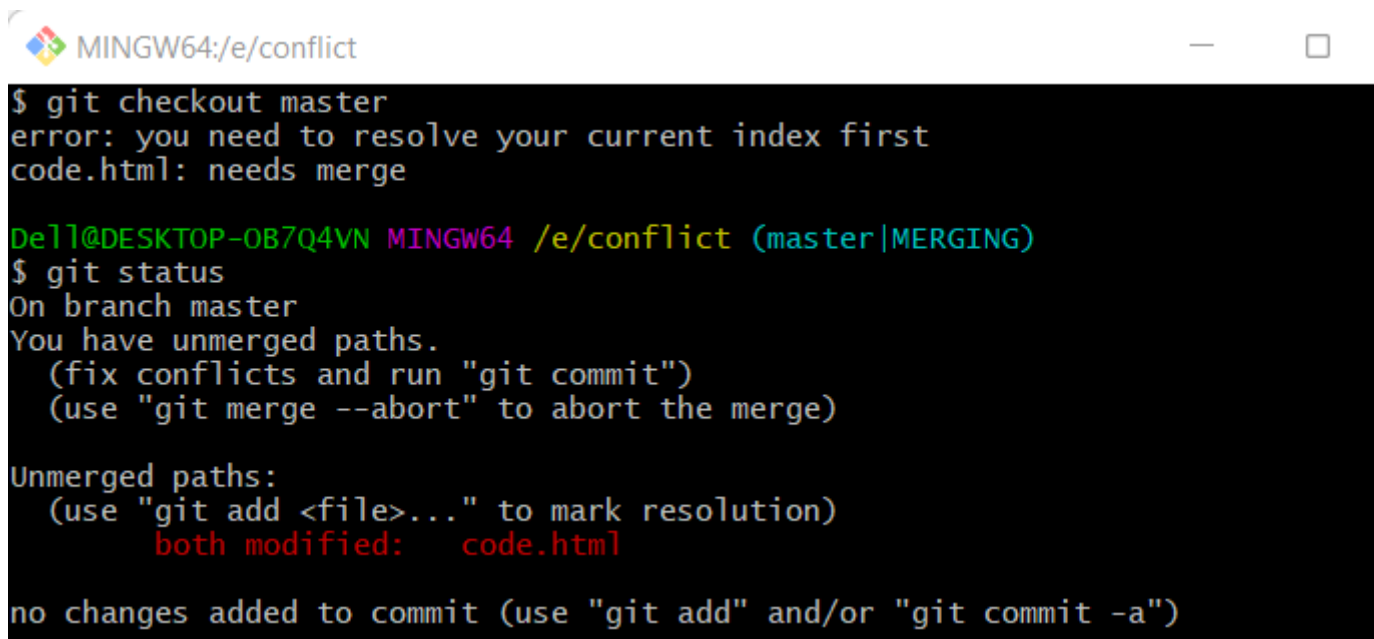
Dell@DESKTOP-OB7Q4VN MINGW64 /e/conflict (master)
$ git status
On branch master
nothing to commit, working tree clean

Dell@DESKTOP-OB7Q4VN MINGW64 /e/conflict (master)
$ git merge conflict
Auto-merging code.html
CONFLICT (content): Merge conflict in code.html
Automatic merge failed; fix conflicts and then commit the result.
```

2. Resolving conflict through code editor.



```
code.html X
E: > conflict > code.html > html > body > ?
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>Document</title>
8 </head>
9 <body>
10
11 Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
12 <<<<<< HEAD (Current Change)
13   <h1>Heading four</h1>
14 =====
15   <h1>Heading three</h1>
16 >>>>>> conflict (Incoming Change)
17 </body>
18 </html>
```



```
MINGW64:/e/conflict
$ git checkout master
error: you need to resolve your current index first
code.html: needs merge

De11@DESKTOP-OB7Q4VN MINGW64 /e/conflict (master|MERGING)
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
    both modified:   code.html

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

3. Adding files and making a commit for the merge.

4. Merging of branches happen.

```
De1l@DESKTOP-OB7Q4VN MINGW64 /e/conflict (master|MERGING)
$ git commit -m "conflict resolved"
[master dd3b7d8] conflict resolved
```

5. Snapshot git log after merging of feature and master branch.

```
De1l@DESKTOP-OB7Q4VN MINGW64 /e/conflict (master)
$ git log
commit dd3b7d84e31916ae8511db1cab2b73228068f0fd (HEAD -> master)
Merge: d445a7a e4af65c
Author: rana-aditi <aditi0080.be21@chitkara.edu.in>
Date: Thu Jun 2 14:02:17 2022 +0530

    conflict resolved

commit d445a7ae76c307a3e4d2f493908c1fabe08d3587
Author: rana-aditi <aditi0080.be21@chitkara.edu.in>
Date: Thu Jun 2 13:55:12 2022 +0530

    fourth heading

commit e4af65c380b31db6543e89238f25e56218751bdf (conflict)
Author: rana-aditi <aditi0080.be21@chitkara.edu.in>
Date: Thu Jun 2 13:54:07 2022 +0530

    third heading

commit 94ebdd5eb93cac2ddb5779482dc0d977e15f7dec
Author: rana-aditi <aditi0080.be21@chitkara.edu.in>
Date: Thu Jun 2 13:53:12 2022 +0530
```

Experiment No. 09

Aim: Reset and Revert

While Working with Git in certain situations we want to undo changes in the working area or index area, sometimes remove commits locally or remotely and we need to reverse those changes.

There are 2 ways to make these changes

- 1) Change previous commits, i.e, the commit with the wrong code is replaced, for this we use “**RESET**”
- 2) Make a new commit that changes the effect of the previous commit without actually removing it from history, for this, we use “**REVERT**”

Git reset

Git reset can be used to remove previous commits from your repository or unstage the currently staged files

To unstage a currently staged file, for example, let's stage a *code.html* with git add Currently *git status* should show:

```
De1l@DESKTOP-OB7Q4VN MINGW64 /e/reset (master)
$ git add .

De1l@DESKTOP-OB7Q4VN MINGW64 /e/reset (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   code.html
```

To unstage, we will run

Git reset HEAD code.html

After which git status will change to:

```
De1l@DESKTOP-OB7Q4VN MINGW64 /e/reset (master)
$ git reset f3fd0270664bc2368624bb4c93c48116517f58ce
Unstaged changes after reset:
M       code.html
```

Now for the other use of reset, that is to remove previous commits, let's make some commits, so the git log is like this

```
De11@DESKTOP-OB7Q4VN MINGW64 /e/reset (master)
$ git log
commit f3fd0270664bc2368624bb4c93c48116517f58ce (HEAD -> master)
Author: rana-aditi <aditi0080.be21@chitkara.edu.in>
Date: Thu Jun 2 14:37:50 2022 +0530

    third commit

commit 6feaf3468549c2c2a63e5a9709aae5133582ee8b
Author: rana-aditi <aditi0080.be21@chitkara.edu.in>
Date: Thu Jun 2 14:37:23 2022 +0530

    second commit

commit af3d9f9ada42e766226a786550c4f76c15f657b4
Author: rana-aditi <aditi0080.be21@chitkara.edu.in>
Date: Thu Jun 2 14:36:40 2022 +0530

    first commit
```

Now if we use

Git reset HEAD~1

This means to remove one commit from HEAD or top of all the commits, which means after this git log will change

In such a way, we can add HEAD~n to remove any n commits from the top. Reset has 3

ways it resets the commits

Git reset —hard HEAD~n: In this, it removes the commits from the history and also deletes the changes made by those commits

NOTE: this will remove **ALL** the changes made by those commits with no way to recover them, use with caution!

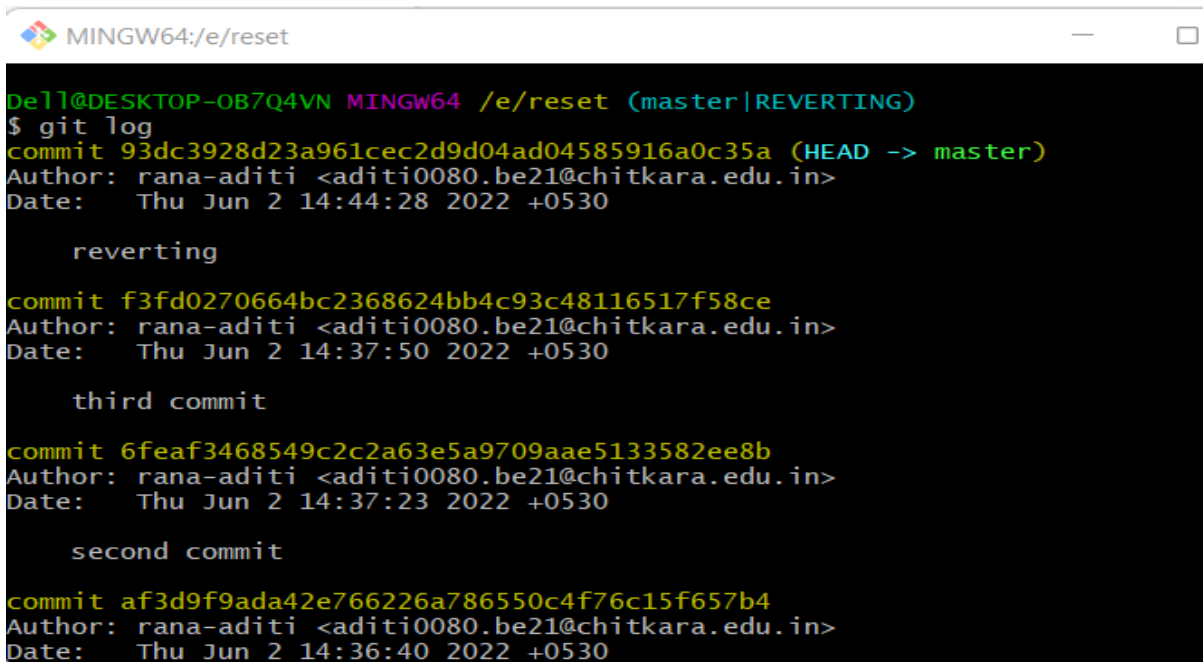
Git reset —mixed HEAD~n: This is the default option which is the same as *git reset HEAD~n* it only removes the commits and unstaged the files but the changes made by those commits still exist in the working directory

Git reset —soft HEAD~n: In this, it only removes the commits but would not unstage the file and the changes stay the same in the working directory, used for example to change the title of the previous commit.

Git Revert

Git revert is used to undo the changes made by some commit without actually removing the commit from the history, it just makes another commit with those changes reversed

For example, if the git log currently is



```
MINGW64:/e/reset
De11@DESKTOP-OB7Q4VN MINGW64 /e/reset (master|REVERTING)
$ git log
commit 93dc3928d23a961cec2d9d04ad04585916a0c35a (HEAD -> master)
Author: rana-aditi <aditi0080.be21@chitkara.edu.in>
Date: Thu Jun 2 14:44:28 2022 +0530

    reverting

commit f3fd0270664bc2368624bb4c93c48116517f58ce
Author: rana-aditi <aditi0080.be21@chitkara.edu.in>
Date: Thu Jun 2 14:37:50 2022 +0530

    third commit

commit 6feaf3468549c2c2a63e5a9709aae5133582ee8b
Author: rana-aditi <aditi0080.be21@chitkara.edu.in>
Date: Thu Jun 2 14:37:23 2022 +0530

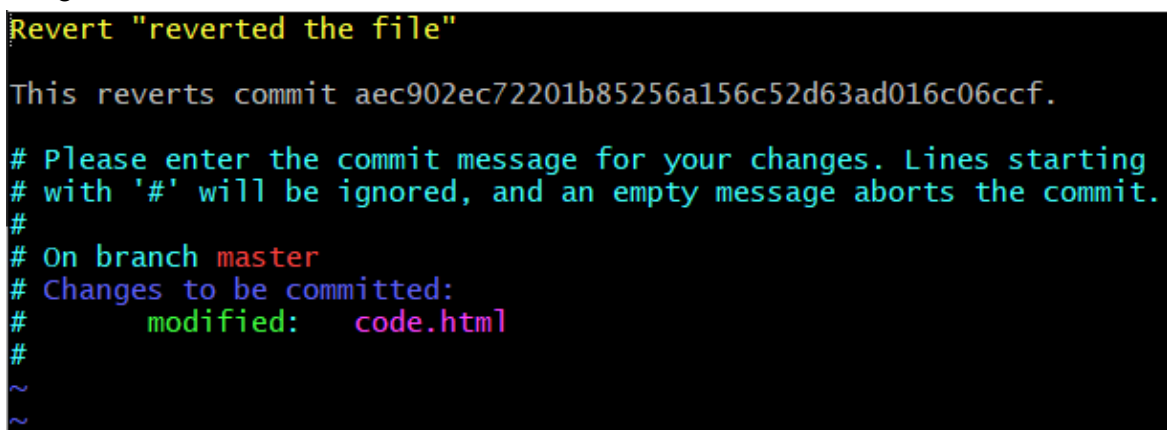
    second commit

commit af3d9f9ada42e766226a786550c4f76c15f657b4
Author: rana-aditi <aditi0080.be21@chitkara.edu.in>
Date: Thu Jun 2 14:36:40 2022 +0530
```

If we want to remove the topmost commit, then we will copy its commit ID/hash which is d33ff. In this case and use git revert as such

Git revert [commit id without brackets]

Which will open a text editor asking for a commit message, after you close the editor it will show the changes made



```
Revert "reverted the file"

This reverts commit aec902ec72201b85256a156c52d63ad016c06ccf.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
# Changes to be committed:
#   modified:   code.html
#
~
~
~
```

The git log now should be

```
De1l@DESKTOP-OB7Q4VN MINGW64 /e/reset (master)
$ git log
commit ee289750bbb47b63ed8c50ea8fbfa73841016a94 (HEAD -> master)
Author: rana-aditi <aditi0080.be21@chitkara.edu.in>
Date: Thu Jun 2 14:49:23 2022 +0530

    Revert "reverted the file"

    This reverts commit aec902ec72201b85256a156c52d63ad016c06ccf.

    This will revert the commit...

commit aec902ec72201b85256a156c52d63ad016c06ccf
Author: rana-aditi <aditi0080.be21@chitkara.edu.in>
Date: Thu Jun 2 14:48:58 2022 +0530

    reverted the file

commit 93dc3928d23a961cec2d9d04ad04585916a0c35a
Author: rana-aditi <aditi0080.be21@chitkara.edu.in>
Date: Thu Jun 2 14:44:28 2022 +0530

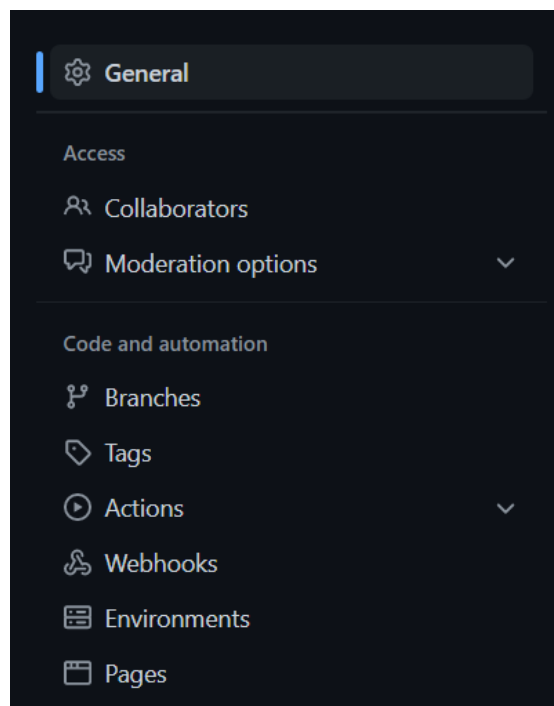
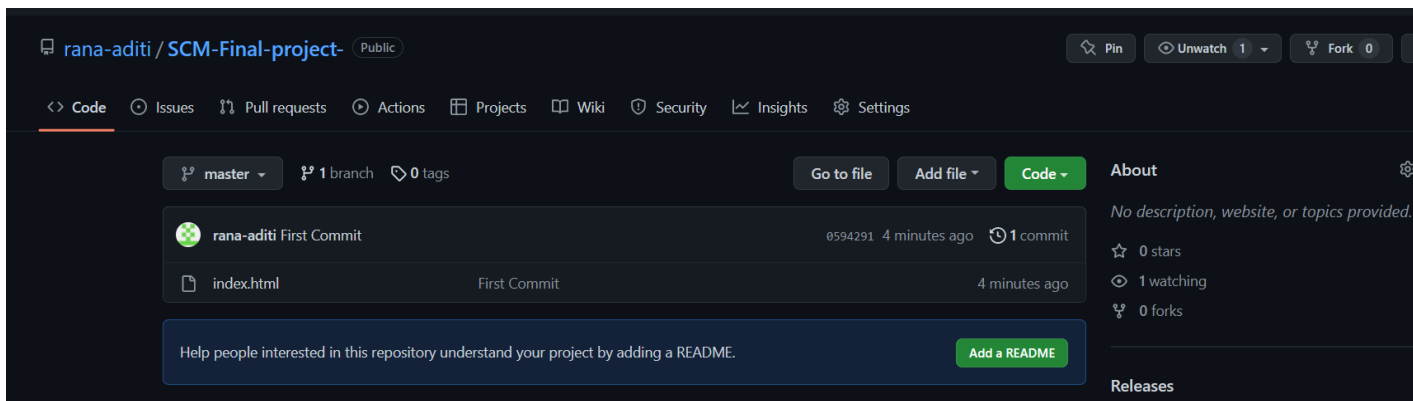
    reverting
```

The main difference between RESET and REVERT is that *Reset* changes the history and makes it so that the mistake or the wrong commit never happened while *Revert* fixes the mistake of the previous commit with another commit while keeping the wrong commits in the history.

Final source code management project :-

Steps-







1. Creating a distributed repository and adding people as collaborators.




Manage access

Add people

☐ Select all Type ▾

<input type="checkbox"/>	 AbhayRiyal Awaiting AbhayRiyal's response	Pending Invite 	Remove
<input type="checkbox"/>	 AKSHAY3183 Awaiting AKSHAY3183's response	Pending Invite 	Remove
<input type="checkbox"/>	 ApaarSaini Awaiting ApaarSaini's response	Pending Invite 	Remove

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

Create an organization


AKSHAY3183 invited you to AKSHAY3183/Scm-Project External Inbox x





AKSHAY3183 <noreply@github.com>
to me ▾

4:52 PM (3 minutes ago)





 + 

@AKSHAY3183 has invited you to collaborate on the **AKSHAY3183/Scm-Project** repository

You can [accept](#) or [decline](#) this invitation. You can also head over to <https://github.com/AKSHAY3183/Scm-Project> to check out the repository or visit [@AKSHAY3183](#) to learn a bit more about them.


This invitation will expire in 7 days.

View invitation

2.Opening pull request for other team members projects.

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: AbhayRiyal/scm-project

base: master

←

head repository: rana-aditi/scm-project-1-Abhay-

compare: master

✓ **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 Jun 1, 2022

prettierrc

 rana-aditi committed 6 minutes ago

 a6a3bb2 


Showing 1 changed file with 4 additions and 0 deletions.


Split

Unified


3.Closing and merging pull requests as a maintainer.

Add more commits by pushing to the **master** branch on **AKSHAY3183/SCM-Final-project**.



 Continuous integration has not been set up

GitHub Actions and several other apps can be used to automatically catch bugs and enforce style.

 This branch has no conflicts with the base branch

Merging can be performed automatically.

Merge pull request

▼

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

AKSHAY3183 / Scm-Project Public

added css #1

Merged rana-aditi merged 1 commit into AKSHAY3183:master from AbhayRiyal:master now

Conversation 0 Commits 1 Checks 0 Files changed 1 +119 -0

AbhayRiyal commented 1 minute ago Collaborator

No description provided.

added css 60c650d

rana-aditi merged commit 20eb9ed into AKSHAY3183:master now

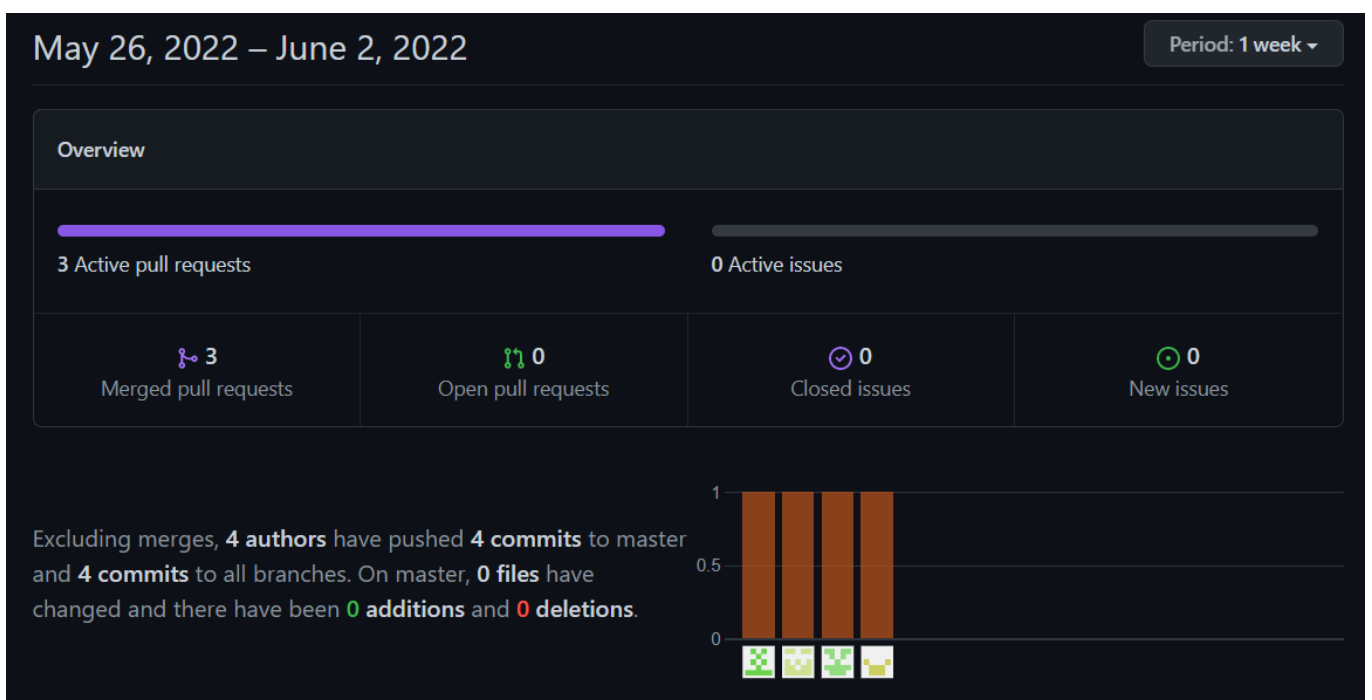
Revert

Reviews: No reviews


Assignees: No one—assign yourself

Labels: None yet


4. Network graph as maintainer.




3 Pull requests merged by 3 people

 adding prettier file

#3 merged 21 hours ago

 added css

#2 merged 21 hours ago

 script file added

#1 merged 21 hours ago

5. Contribution graph for whole year.

