Subject Name: **Source Code Management**

Subject Code: **CS181**

Cluster: **Beta**

Department: **CSE**

Submitted by- Akshay Sharma

Roll no. :-2110990130

Submitted to- Dr. Monit Kapoor

INDEX

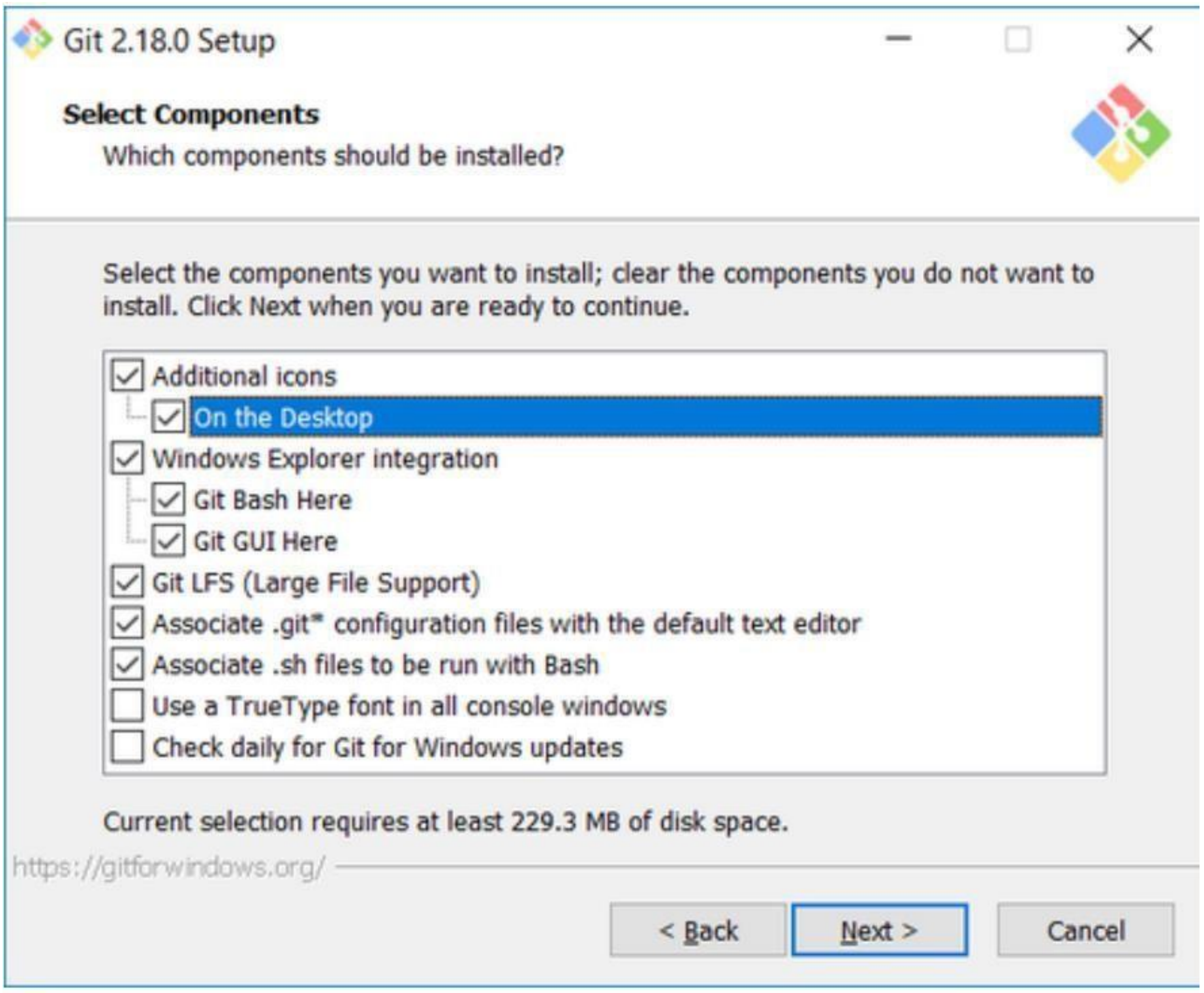
Experiment No. 01

Aim: Setting up the git client.

Git Installation: Download the Git installation program (Windows, Mac, or Linux) from Git - Downloads (git-scm.com).

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:


Git 2.18.0 Setup

Select Components

Which components should be installed?

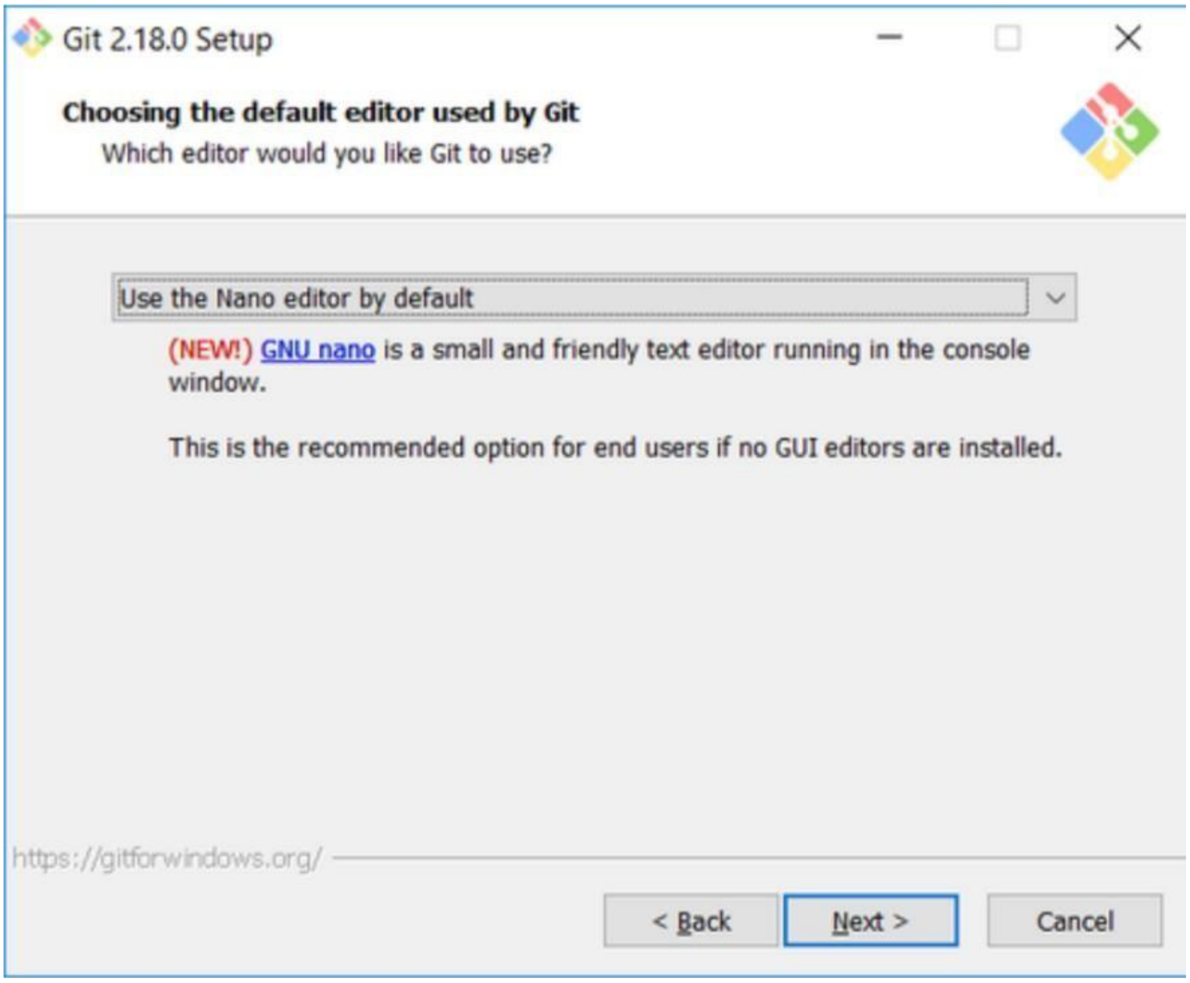
Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

- [✓] Additional icons
  - [✓] On the Desktop
- [✓] Windows Explorer integration
  - [✓] Git Bash Here
  - [✓] Git GUI Here
- [✓] Git LFS (Large File Support)
- [✓] Associate .git* configuration files with the default text editor
- [✓] Associate .sh files to be run with Bash
- [ ] Use a TrueType font in all console windows
- [ ] Check daily for Git for Windows updates

Current selection requires at least 229.3 MB of disk space.

https://gitforwindows.org/

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++.



Git 2.18.0 Setup

Choosing the default editor used by Git

Which editor would you like Git to use?

Use the Nano editor by default

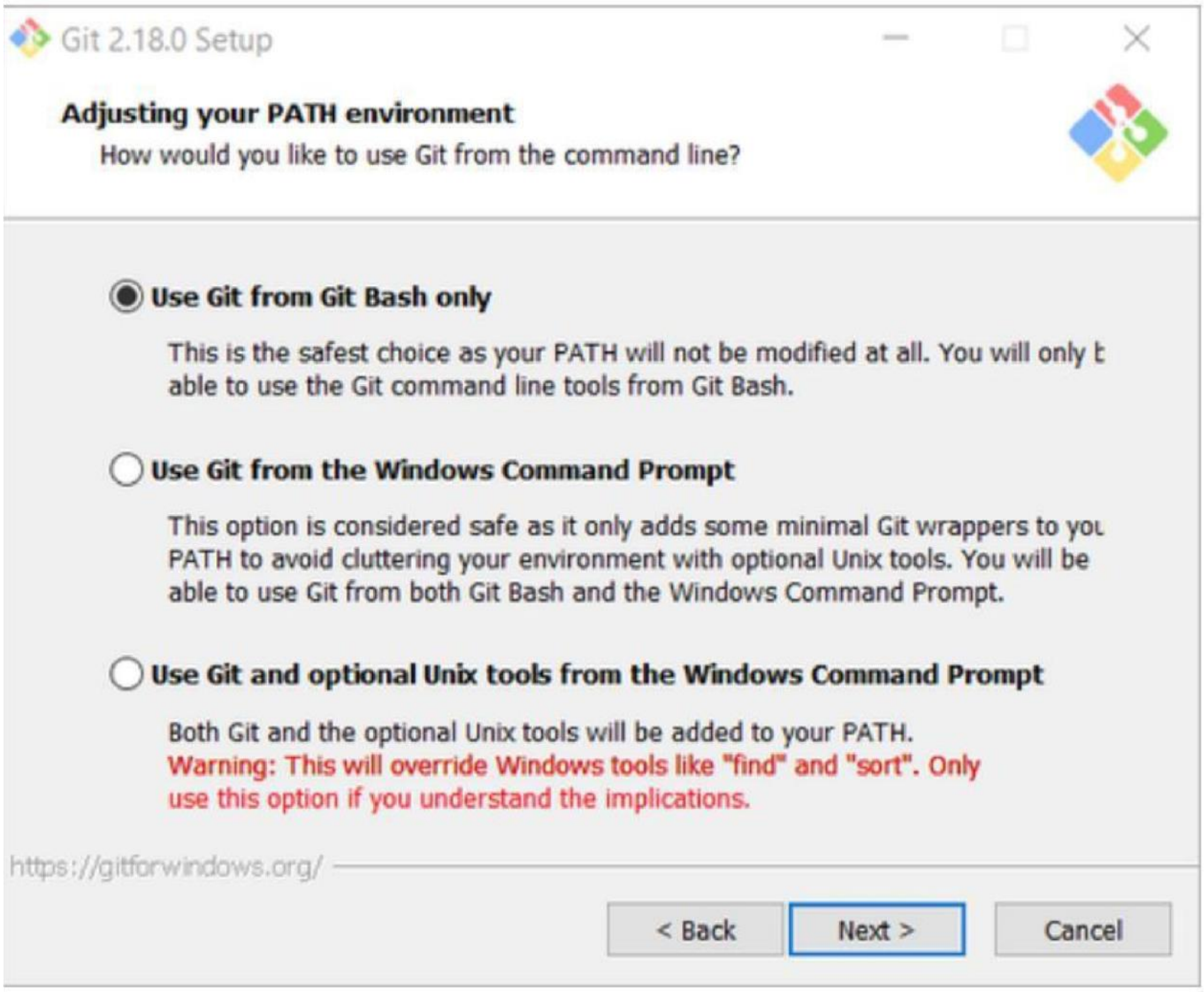
(NEW!) GNU nano is a small and friendly text editor running in the console window.

This is the recommended option for end users if no GUI editors are installed.

https://gitforwindows.org/

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.


Git 2.18.0 Setup

Adjusting your PATH environment

How would you like to use Git from the command line?

[✓ **Use Git from Git Bash only**]

This is the safest choice as your PATH will not be modified at all. You will only b
able to use the Git command line tools from Git Bash.

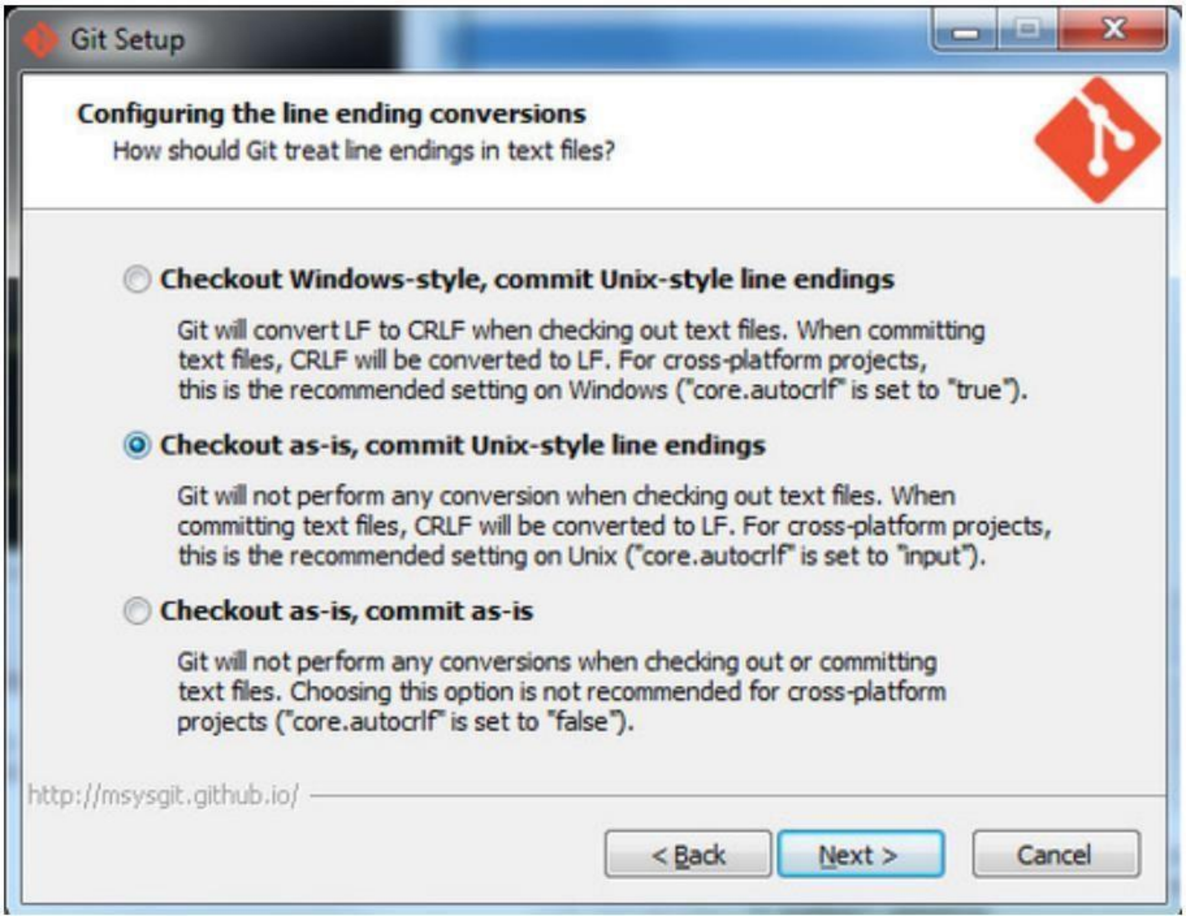
Use Git from the Windows Command Prompt

This option is considered safe as it only adds some minimal Git wrappers to you
PATH to avoid cluttering your environment with optional Unix tools. You will be
able to use Git from both Git Bash and the Windows Command Prompt.

Use Git and optional Unix tools from the Windows Command Prompt

Both Git and the optional Unix tools will be added to your PATH.
Warning: This will override Windows tools like "find" and "sort". Only
use this option if you understand the implications.

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 the line ending conversions
How should Git treat line endings in text files?

( ) **Checkout Windows-style, commit Unix-style line endings**
Git will convert LF to CRLF when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Windows ("core.autocrlf" is set to "true").

[✓ **Checkout as-is, commit Unix-style line endings**]
Git will not perform any conversion when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Unix ("core.autocrlf" is set to "input").

( ) **Checkout as-is, commit as-is**
Git will not perform any conversions when checking out or committing text files. Choosing this option is not recommended for cross-platform projects ("core.autocrlf" is set to "false").

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 “akshay0130.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 "AKSHAY3183"`

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.

Experiment No. 02

Aim

Setting up GitHub Account

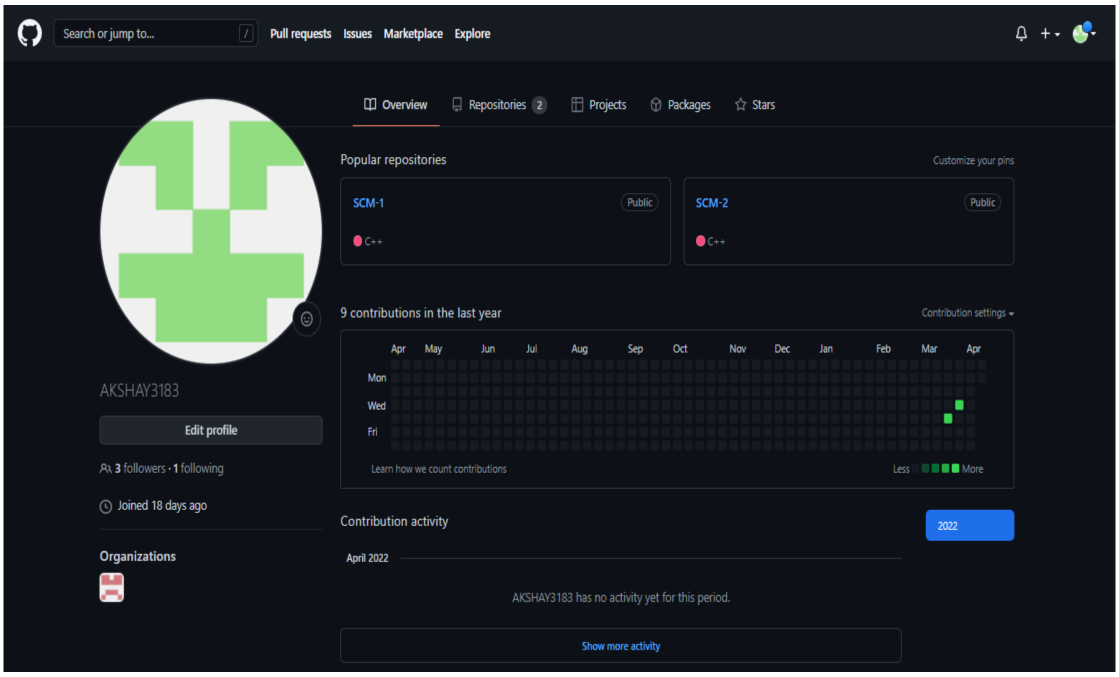
The first steps in starting with GitHub are to create an account, choose a product that fits your needs best, verify your email, set up two-factor authentication, and view your profile.

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.

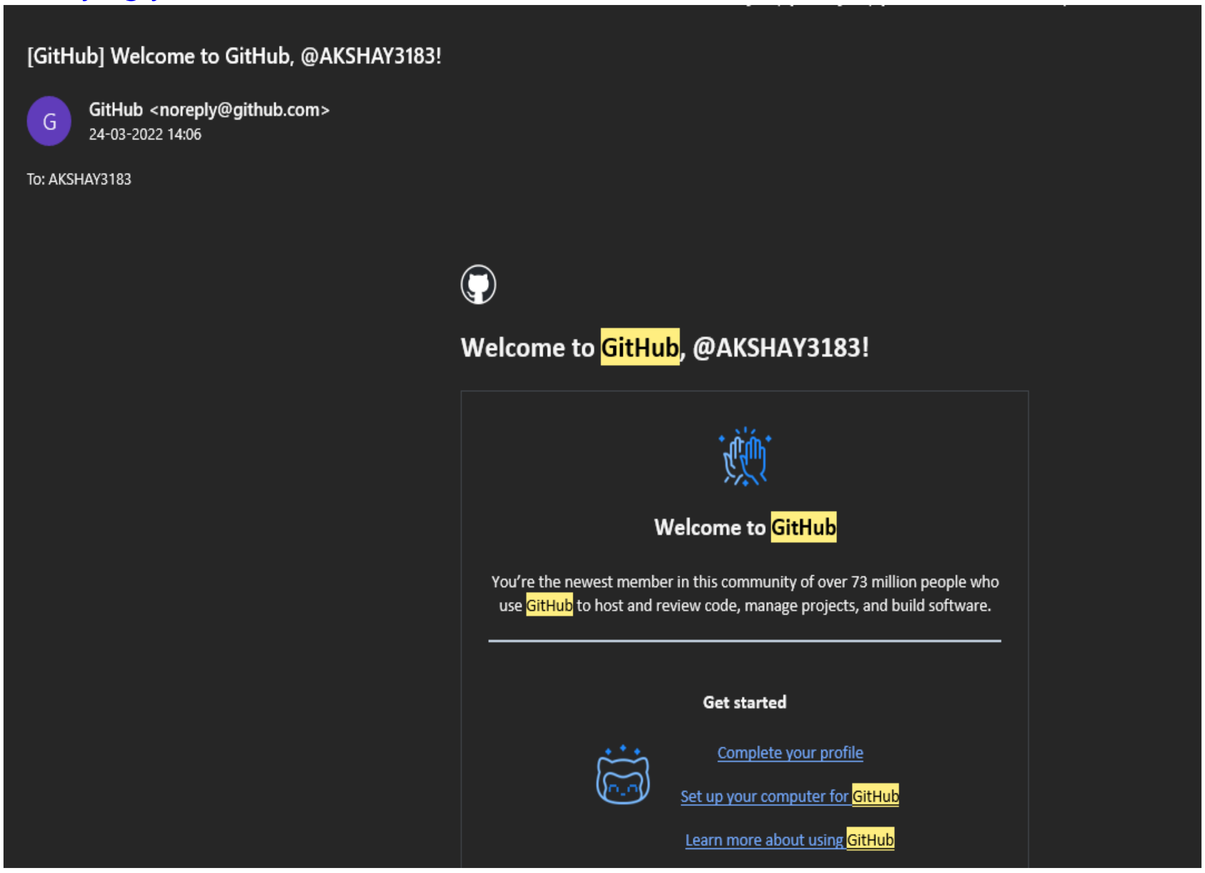
   For more information, see "Creating a strong password".



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"



[GitHub] Welcome to GitHub, @AKSHAY3183!

GitHub <noreply@github.com>
24-03-2022 14:06

To: AKSHAY3183

Welcome to GitHub, @AKSHAY3183!

Welcome to GitHub

You're the newest member in this community of over 73 million people who use GitHub to host and review code, manage projects, and build software.

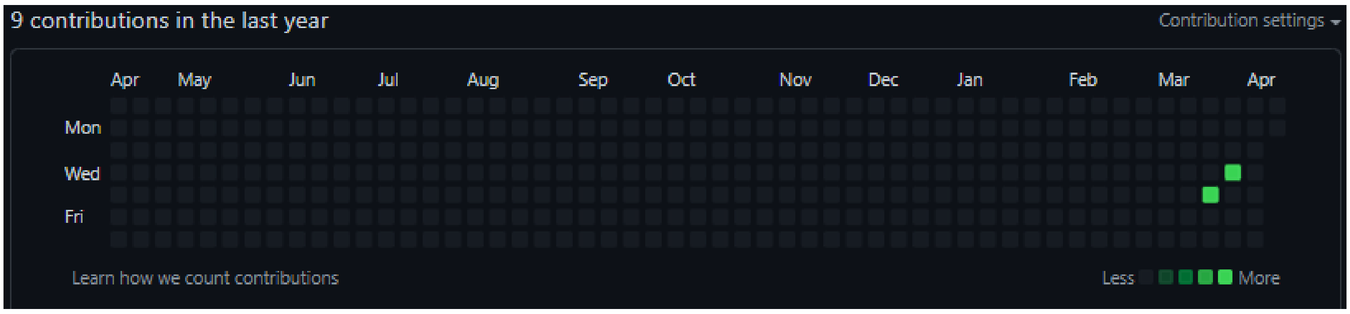
Get started

Complete your profile

Set up your computer for GitHub

Learn more about using GitHub

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."


9 contributions in the last year

Experiment No. 03

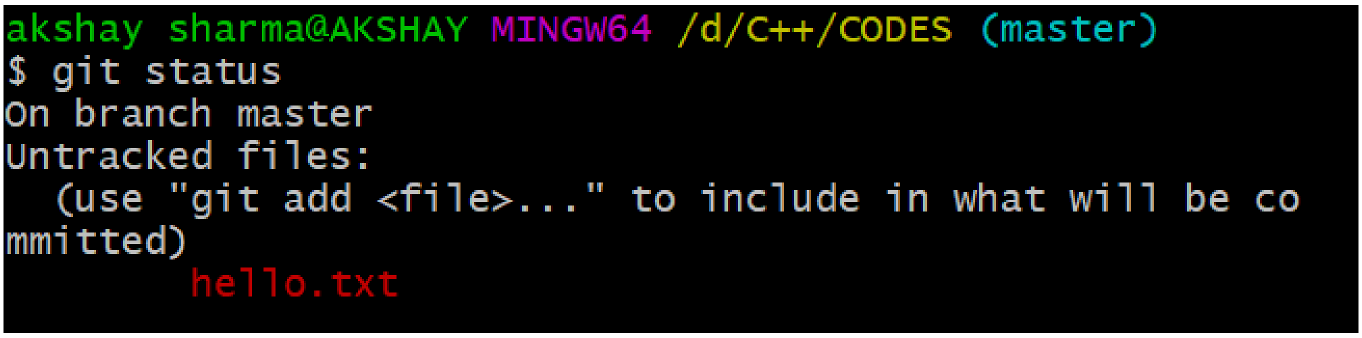
Aim: Program to generate logs

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.

Basic Git status

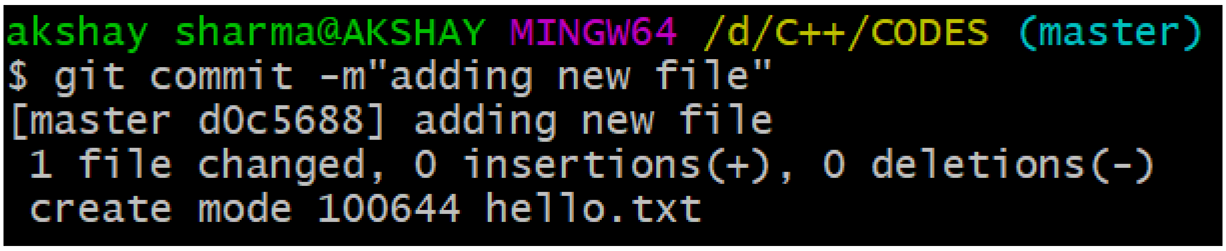
The git status command displays the state of the working directory and the staging area. It lets you see which changes have been staged, which haven't, and which files aren't being tracked by Git. Status output does not show you any information regarding the committed project history.



```
akshay sharma@AKSHAY MINGW64 /d/C++/CODES (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be co
mmitted)
        hello.txt
```

Basic Git commit

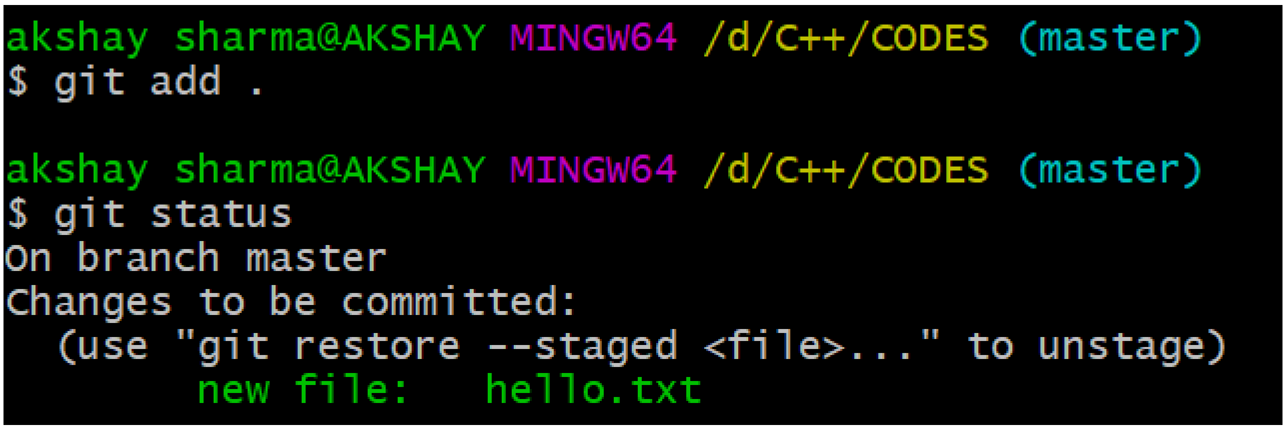
The `git commit` command captures a snapshot of the project's currently staged changes. Committed snapshots can be thought of as "safe" versions of a project—Git will never change them unless you explicitly ask it to. Prior to the execution of `git commit`, The git add command is used to promote or 'stage'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



```
akshay sharma@AKSHAY MINGW64 /d/C++/CODES (master)
$ git commit -m"adding new file"
[master d0c5688] adding new file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 hello.txt
```

Basic Git add command

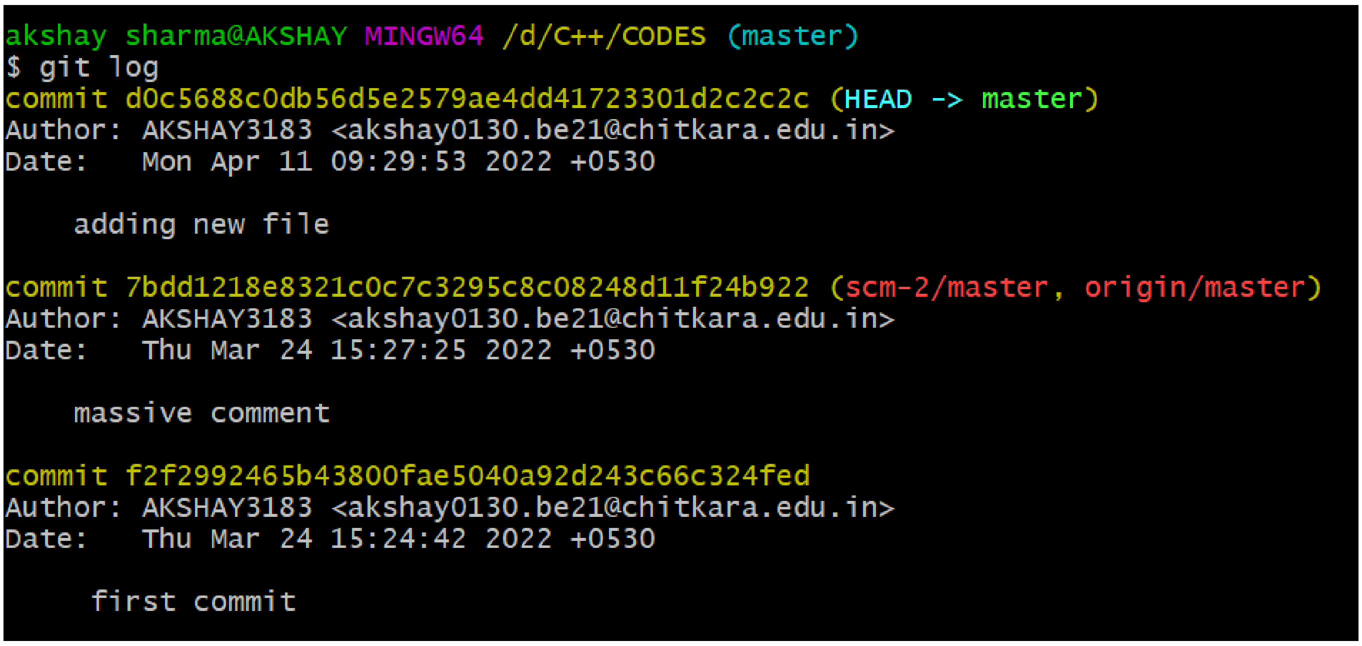
The git add command adds a change in the working directory to the staging area. It tells Git that you want to include updates to a particular file in the next commit. However, git add doesn't really affect the repository in any significant way—changes are not actually recorded until you run git commit



```
akshay sharma@AKSHAY MINGW64 /d/C++/CODES (master)
$ git add .

akshay sharma@AKSHAY MINGW64 /d/C++/CODES (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   hello.txt
```

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:



```
akshay sharma@AKSHAY MINGW64 /d/C++/CODES (master)
$ git log
commit d0c5688c0db56d5e2579ae4dd41723301d2c2c2c (HEAD -> master)
Author: AKSHAY3183 <akshay0130.be21@chitkara.edu.in>
Date:   Mon Apr 11 09:29:53 2022 +0530

    adding new file

commit 7bdd1218e8321c0c7c3295c8c08248d11f24b922 (scm-2/master, origin/master)
Author: AKSHAY3183 <akshay0130.be21@chitkara.edu.in>
Date:   Thu Mar 24 15:27:25 2022 +0530

    massive comment

commit f2f2992465b43800fae5040a92d243c66c324fed
Author: AKSHAY3183 <akshay0130.be21@chitkara.edu.in>
Date:   Thu Mar 24 15:24:42 2022 +0530

     first commit
```

Experiment No. 04

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"

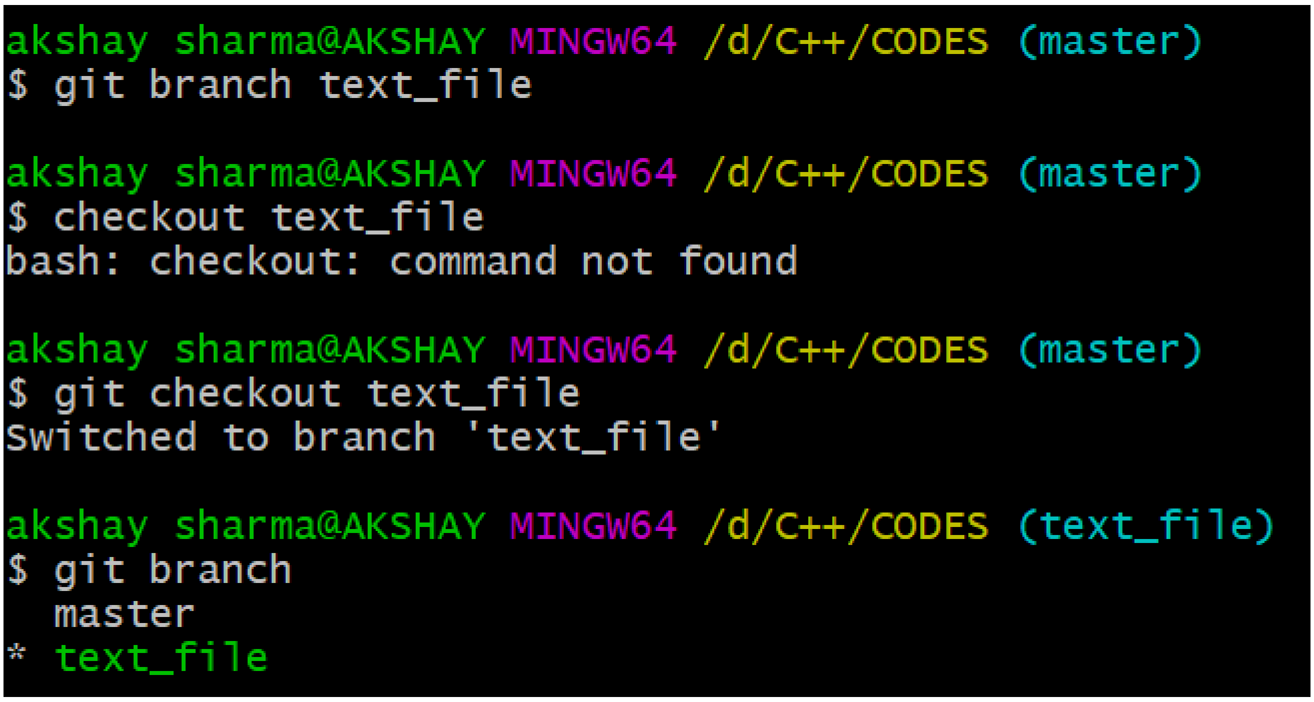
Visualizing Branches:

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.

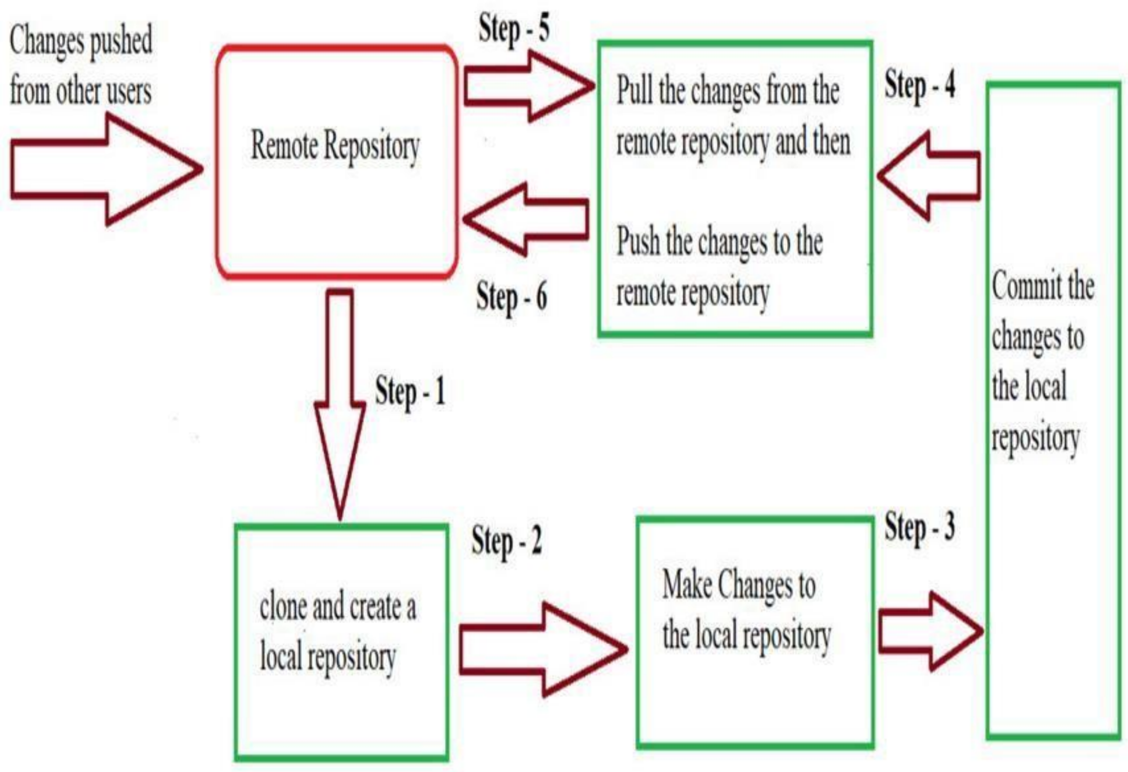
In this way we can create and change different branches. We can also merge the branches by using git merge command.


```
akshay sharma@AKSHAY MINGW64 /d/C++/CODES (master)
$ git branch text_file

akshay sharma@AKSHAY MINGW64 /d/C++/CODES (master)
$ checkout text_file
bash: checkout: command not found

akshay sharma@AKSHAY MINGW64 /d/C++/CODES (master)
$ git checkout text_file
Switched to branch 'text_file'

akshay sharma@AKSHAY MINGW64 /d/C++/CODES (text_file)
$ git branch
  master
* text_file
```

Experiment No. 05

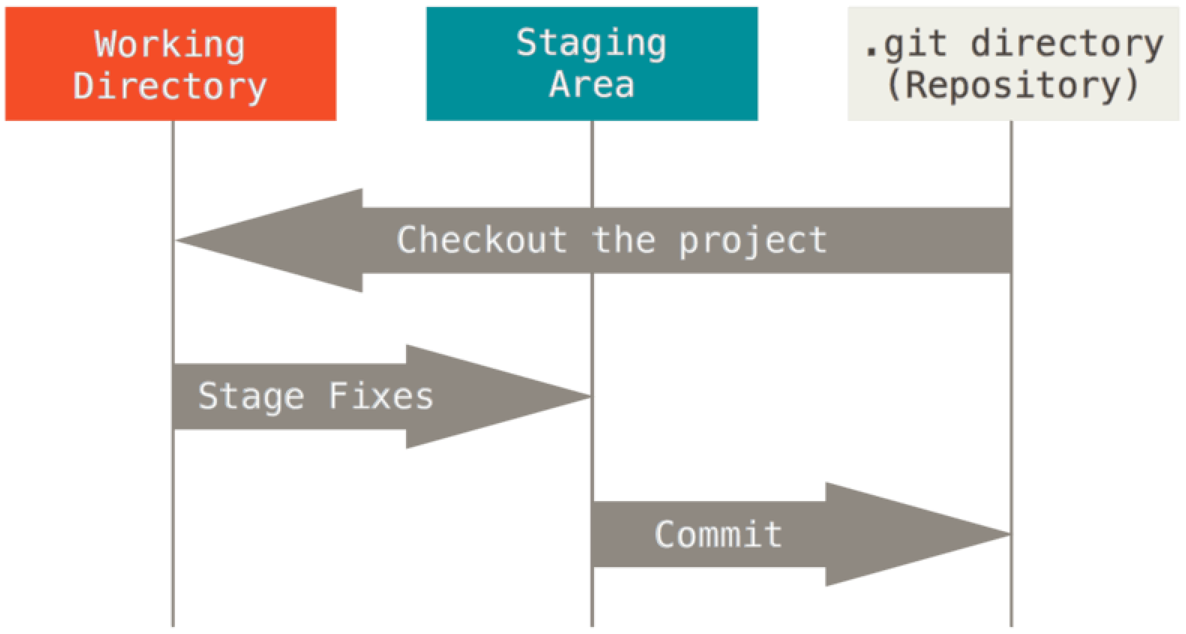
Aim: Git lifecycle description

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-


Changes pushed from other users
Remote Repository
Step - 5
Pull the changes from the remote repository and then
Push the changes to the remote repository
Step - 4
Step - 6
Commit the changes to the local repository
Step - 1
Step - 2
Step - 3
clone and create a local repository
Make Changes to the local repository

- **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 <Message>

Experiment No. 06

Aim: Add collaborators on GitHub Repo

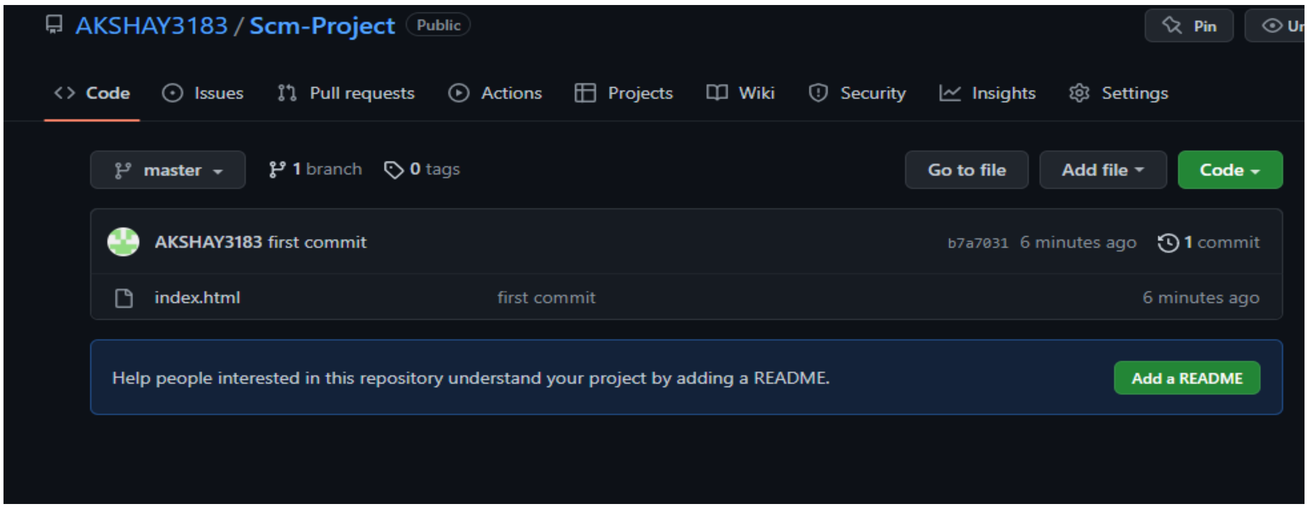
In GitHub, We can invite other GitHub users to become collaborators to our private repositories(which expire after 7 days if not accepted, restoring any unclaimed licenses). Being a collaborator, of a personal repository you can pull (read) the contents of the repository and push (write) changes to the 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:

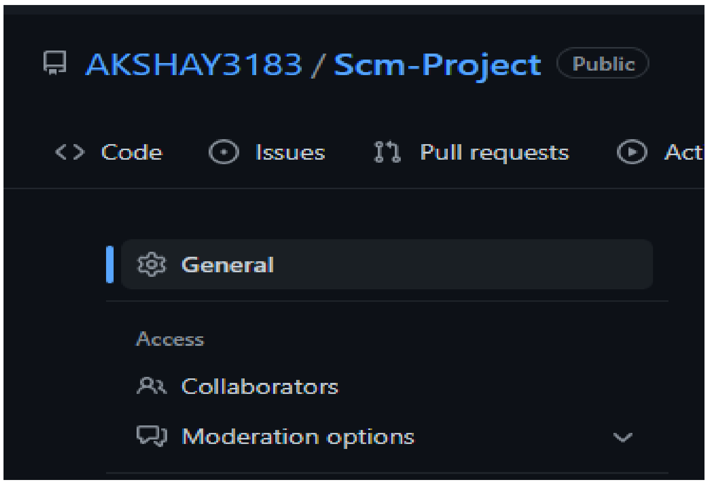
Step 1: Get the GitHub username of people who you want to add as collaborators, in case they aren't on GitHub, they will need to sign-up.

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

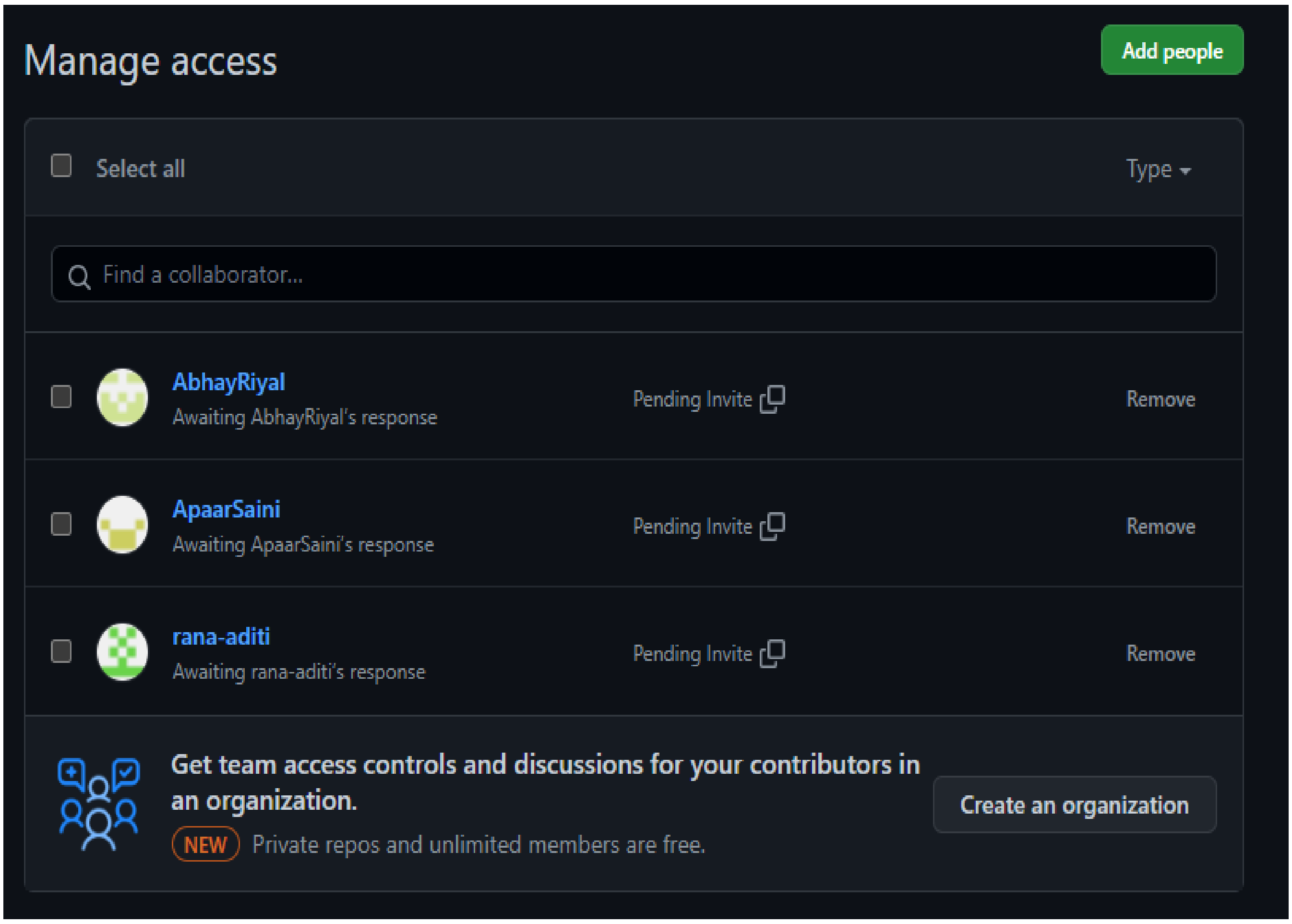


<u>Step 5:</u> It might ask your GitHub account password saying "Sudo mode" is going to be activated, if asked, enter your password.

<u>Step 6:</u> Click Add People

<u>Step 7:</u> 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

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



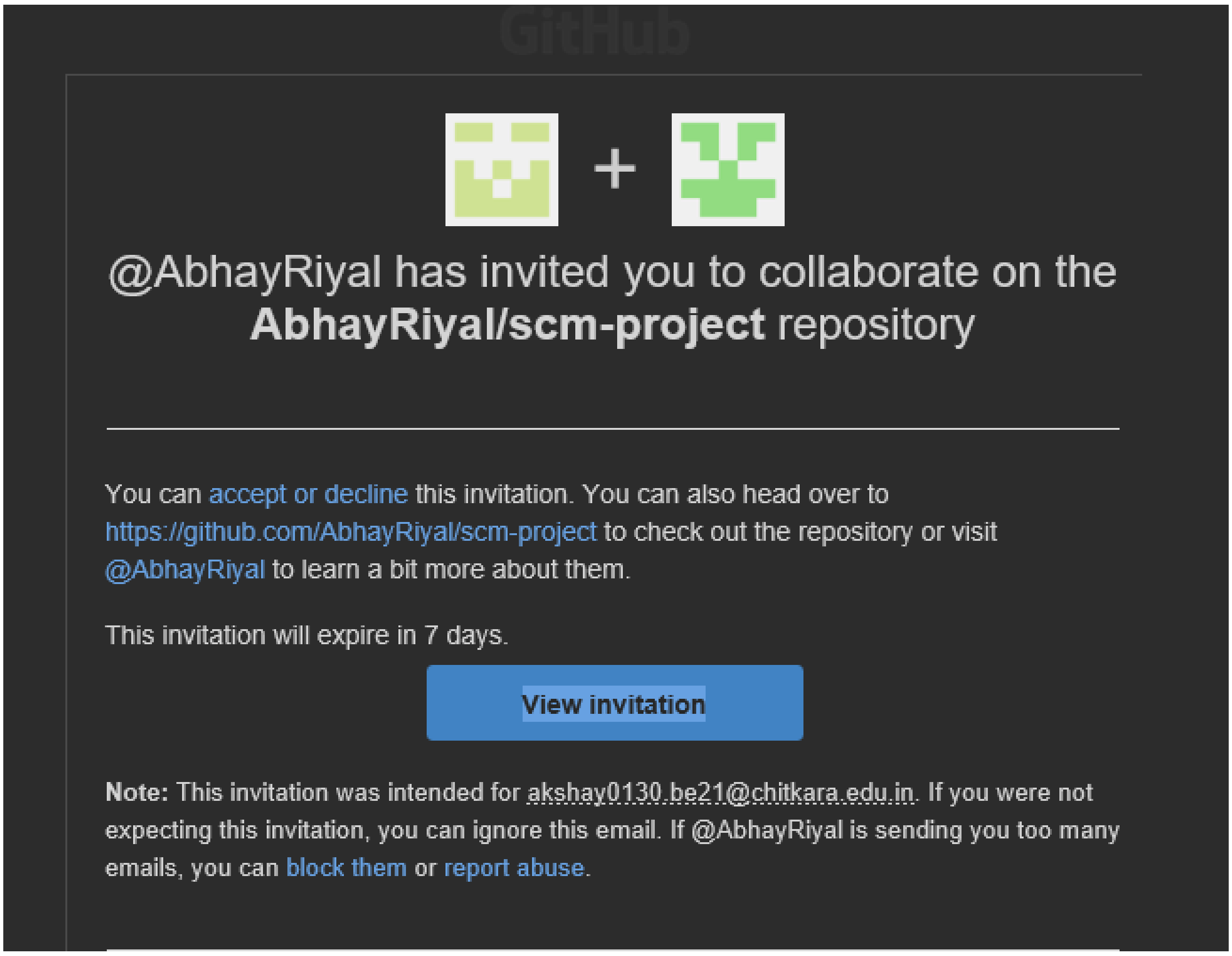
AbhayRiyal
Awaiting AbhayRiyal's response
Pending Invite

ApaarSaini
Awaiting ApaarSaini's response
Pending Invite

rana-aditi
Awaiting rana-aditi's response
Pending Invite

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

<u>Step 9:</u> 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!


GitHub

@AbhayRiyal has invited you to collaborate on the **AbhayRiyal/scm-project** repository

You can accept or decline this invitation. You can also head over to https://github.com/AbhayRiyal/scm-project to check out the repository or visit @AbhayRiyal to learn a bit more about them.

This invitation will expire in 7 days.

View invitation

Note: This invitation was intended for akshay0130.be21@chitkara.edu.in. If you were not expecting this invitation, you can ignore this email. If @AbhayRiyal is sending you too many emails, you can block them or report abuse.

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 thefollowing reasons:

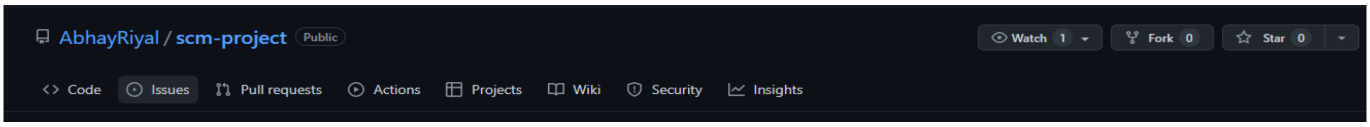
1. You have your own copy of the project on which you may test your own changeswithout changing the original project.

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.

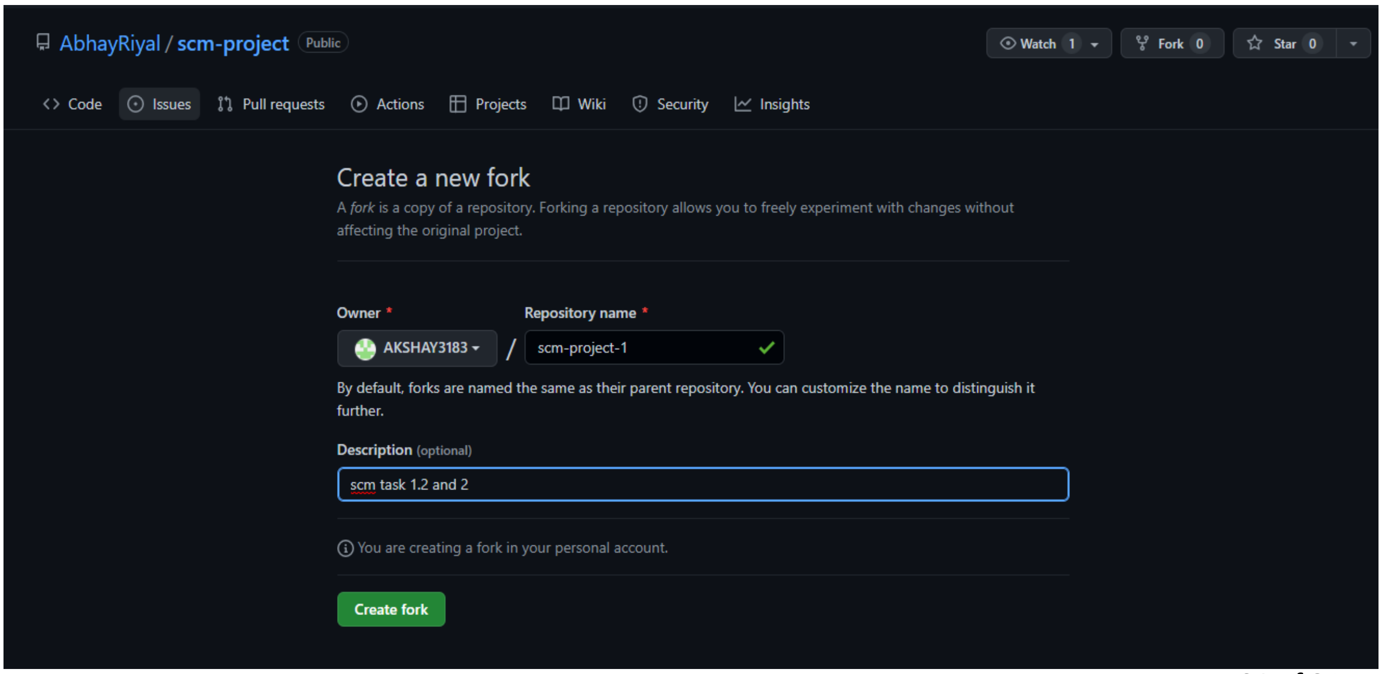
3. When you clone an Open Source project, which isn't yours, you don't have the rightto push code directly into the project.

Steps to fork a repository:

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



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



Create a new fork

A *fork* is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project.

Owner * AKSHAY3183 / Repository name * scm-project-1

By default, forks are named the same as their parent repository. You can customize the name to distinguish it further.

Description (optional) scm task 1.2 and 2

You are creating a fork in your personal account.

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 onlogical units of change.

Over time, commits should tell a story of the history of your repository and how it cameto 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 READMEfile 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 addcommit title and its description

5. You can give it a title and then click the “Commit changes” button on the bottom tocomplete 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.

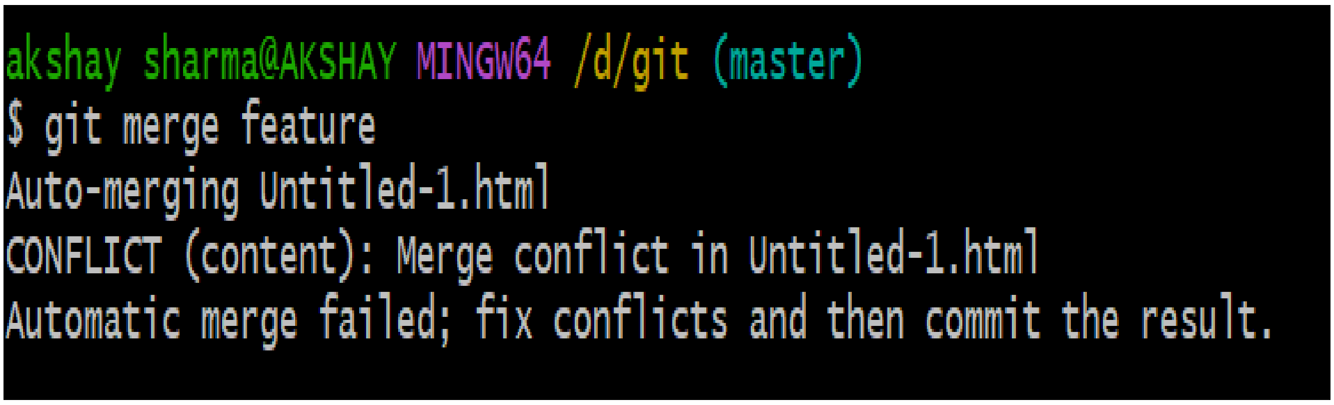
2. After editing the file, we can use the git add a command to stage the new merged content.

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

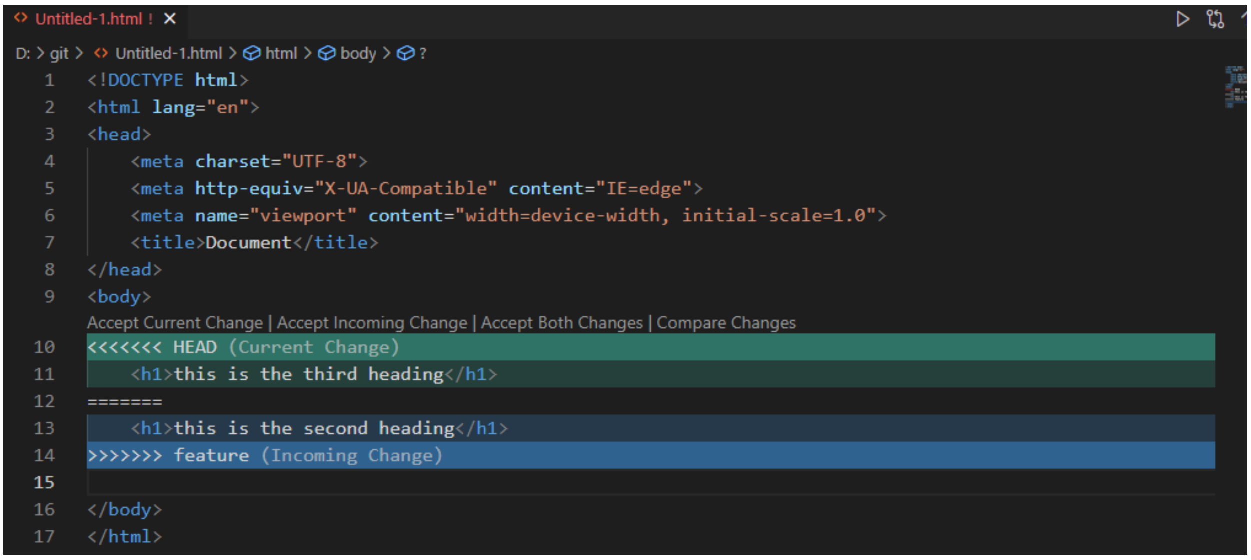
Let us now look into the Git commands that may play a significant role in resolving conflicts.

1. Merge conflict occurs.



```
akshay sharma@AKSHAY MINGW64 /d/git (master)
$ git merge feature
Auto-merging Untitled-1.html
CONFLICT (content): Merge conflict in Untitled-1.html
Automatic merge failed; fix conflicts and then commit the result.
```

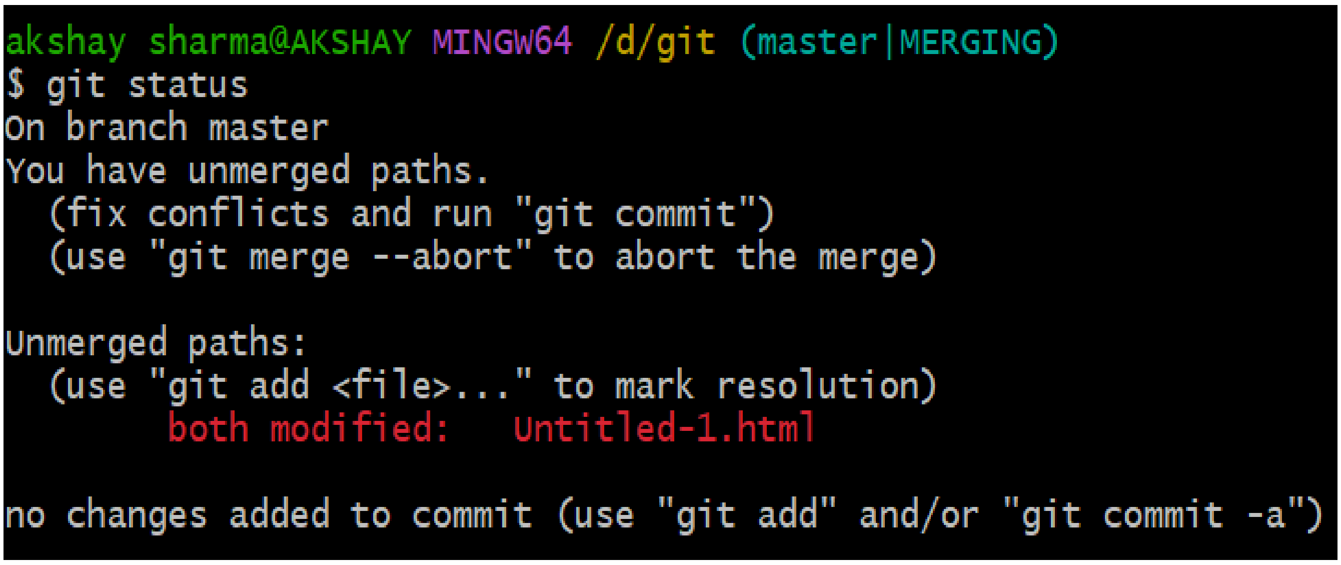
2. Resolving conflict through code editor.



Untitled-1.html !
D: > git > Untitled-1.html > html > body > ?
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes

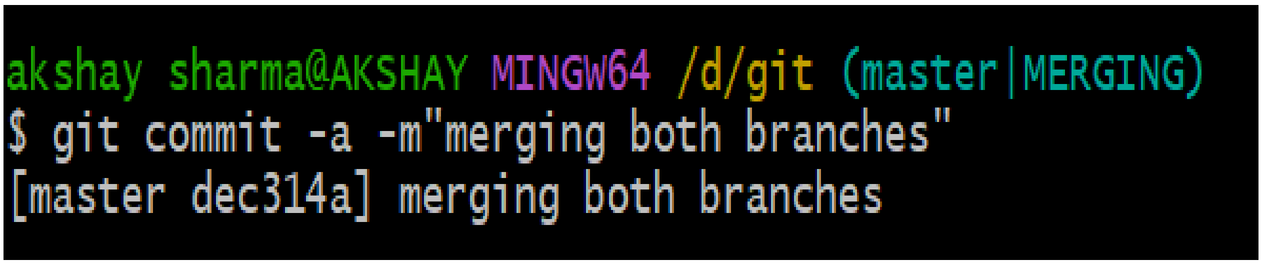
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
<<<<<<< HEAD (Current Change)
    <h1>this is the third heading</h1>
=======
    <h1>this is the second heading</h1>
>>>>>>> feature (Incoming Change)

</body>
</html>
```

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



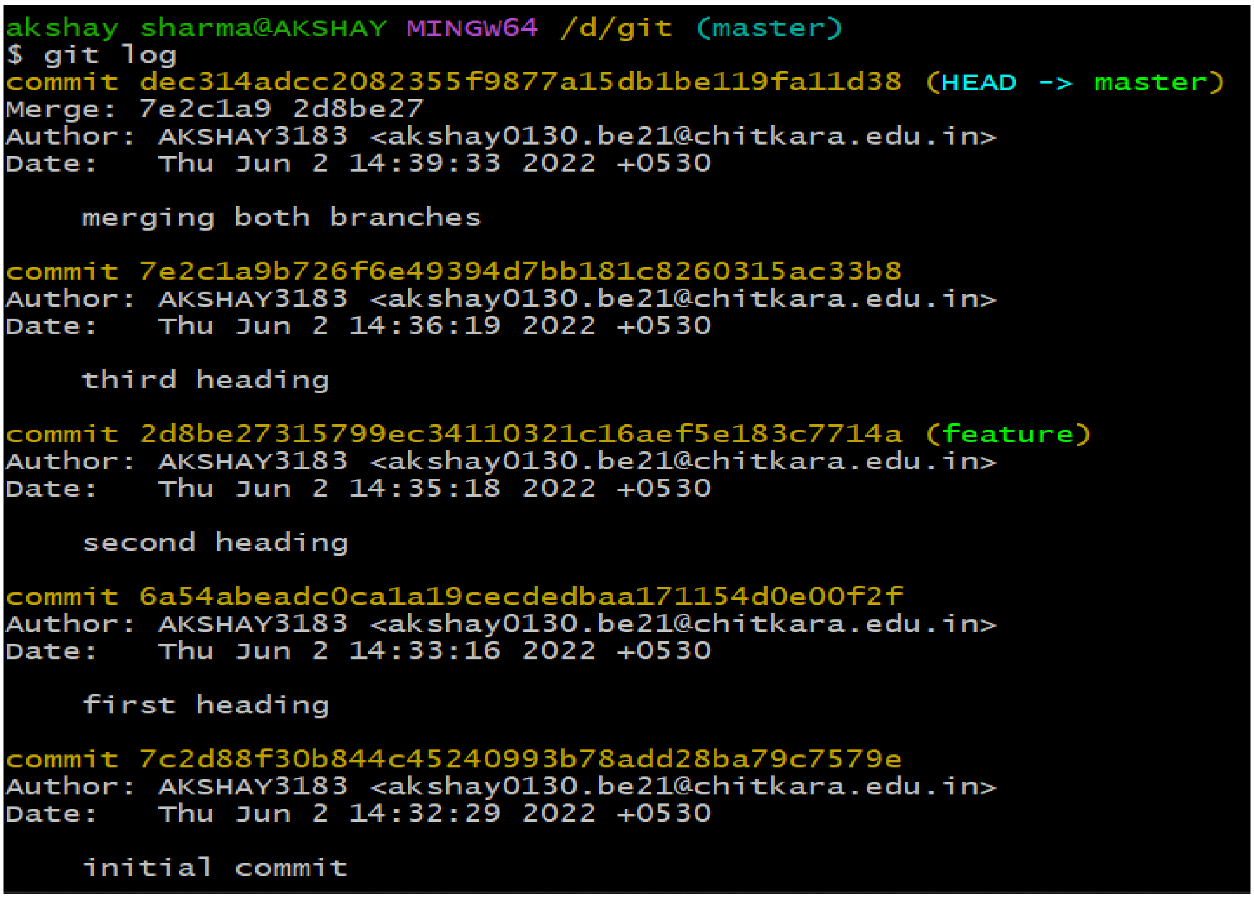
```
akshay sharma@AKSHAY MINGW64 /d/git (master|MERGING)
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both modified:   Untitled-1.html

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

1. Merging of branches happen.



```
akshay sharma@AKSHAY MINGW64 /d/git (master|MERGING)
$ git commit -a -m"merging both branches"
[master dec314a] merging both branches
```

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



```
akshay sharma@AKSHAY MINGW64 /d/git (master)
$ git log
commit dec314adcc2082355f9877a15db1be119fa11d38 (HEAD -> master)
Merge: 7e2c1a9 2d8be27
Author: AKSHAY3183 <akshay0130.be21@chitkara.edu.in>
Date:   Thu Jun 2 14:39:33 2022 +0530

    merging both branches

commit 7e2c1a9b726f6e49394d7bb181c8260315ac33b8
Author: AKSHAY3183 <akshay0130.be21@chitkara.edu.in>
Date:   Thu Jun 2 14:36:19 2022 +0530

    third heading

commit 2d8be27315799ec34110321c16aef5e183c7714a (feature)
Author: AKSHAY3183 <akshay0130.be21@chitkara.edu.in>
Date:   Thu Jun 2 14:35:18 2022 +0530

    second heading

commit 6a54abeadc0ca1a19cecdedbaa171154d0e00f2f
Author: AKSHAY3183 <akshay0130.be21@chitkara.edu.in>
Date:   Thu Jun 2 14:33:16 2022 +0530

    first heading

commit 7c2d88f30b844c45240993b78add28ba79c7579e
Author: AKSHAY3183 <akshay0130.be21@chitkara.edu.in>
Date:   Thu Jun 2 14:32:29 2022 +0530

    initial commit
```

Experiment No. 09

Aim: Reset and Revert

While Working with Git in certain situations we want to undo changes in the workingarea 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, forthis we use "**RESET**"

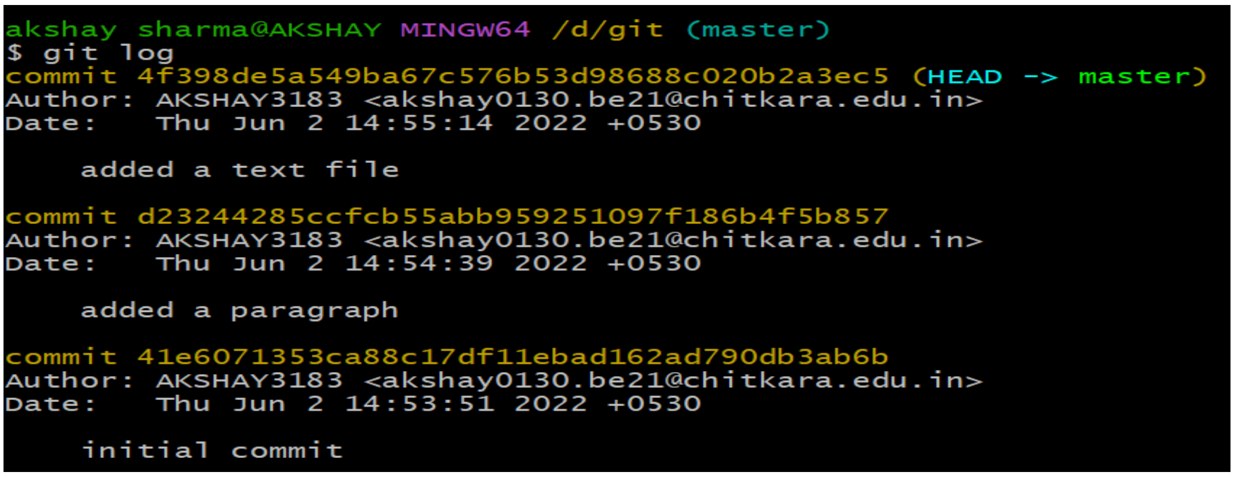
2) Make a new commit that changes the effect of the previous commit withoutactually removing it from history, for this, we use "**REVERT**"

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 ***textFile2*** with git add

Currently *git status* should show:

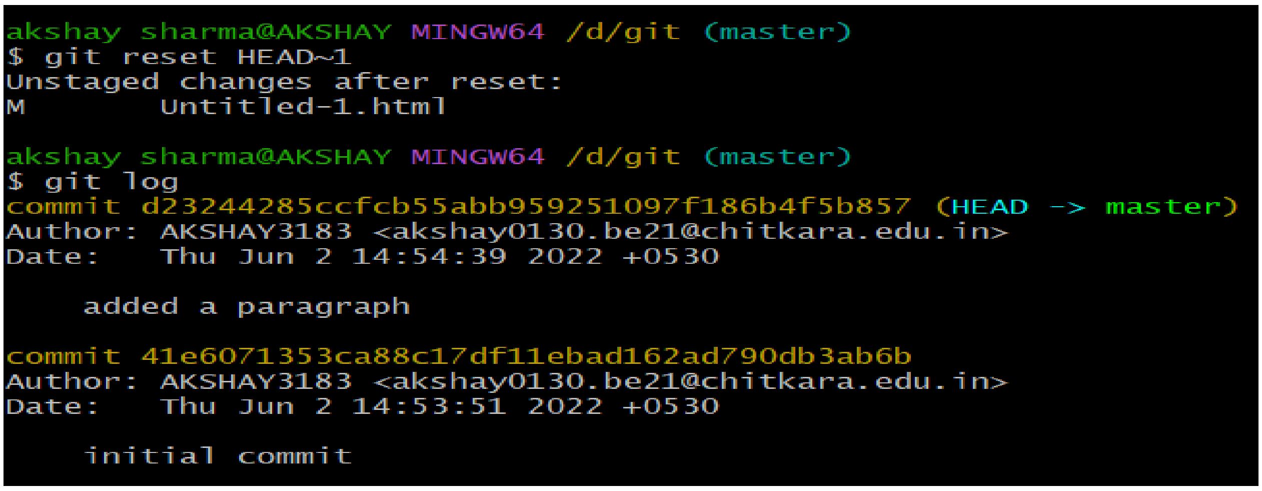


```
akshay sharma@AKSHAY MINGW64 /d/git (master)
$ git log
commit 4f398de5a549ba67c576b53d98688c020b2a3ec5 (HEAD -> master)
Author: AKSHAY3183 <akshay0130.be21@chitkara.edu.in>
Date:   Thu Jun 2 14:55:14 2022 +0530

    added a text file

commit d23244285ccfcb55abb959251097f186b4f5b857
Author: AKSHAY3183 <akshay0130.be21@chitkara.edu.in>
Date:   Thu Jun 2 14:54:39 2022 +0530

    added a paragraph

commit 41e6071353ca88c17df11ebad162ad790db3ab6b
Author: AKSHAY3183 <akshay0130.be21@chitkara.edu.in>
Date:   Thu Jun 2 14:53:51 2022 +0530

    initial commit
```

To unstage, we will run

Git reset HEAD textFile2

After which git status will change to:-



```
akshay sharma@AKSHAY MINGW64 /d/git (master)
$ git reset HEAD~1
Unstaged changes after reset:
M       Untitled-1.html

akshay sharma@AKSHAY MINGW64 /d/git (master)
$ git log
commit d23244285ccfcb55abb959251097f186b4f5b857 (HEAD -> master)
Author: AKSHAY3183 <akshay0130.be21@chitkara.edu.in>
Date:   Thu Jun 2 14:54:39 2022 +0530

    added a paragraph

commit 41e6071353ca88c17df11ebad162ad790db3ab6b
Author: AKSHAY3183 <akshay0130.be21@chitkara.edu.in>
Date:   Thu Jun 2 14:53:51 2022 +0530

    initial commit
```

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

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!

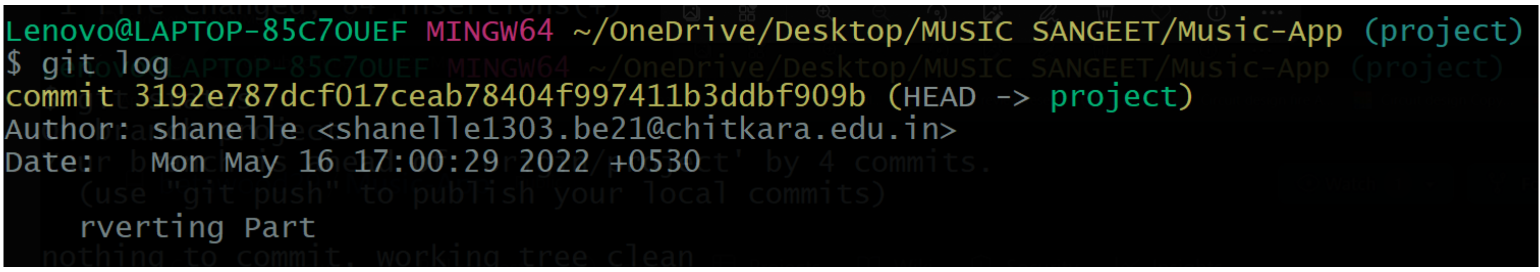
Git reset —mixed HEAD~n: This is the default option which is the same as *git reset HEAD~n* it only removes the commits and unstaged the files but the changes made bythose commits still exist in the working directory

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

Git Revert

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

For example, if the git log currently is

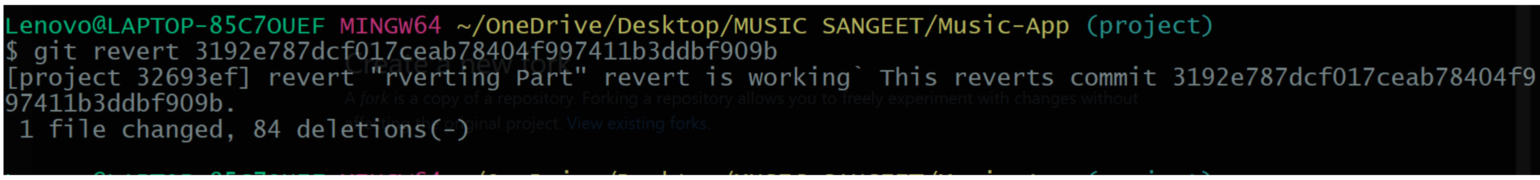


```
Lenovo@LAPTOP-85C7OUEF MINGW64 ~/OneDrive/Desktop/MUSIC SANGEET/Music-App (project)
$ git log
commit 3192e787dcf017ceab78404f997411b3ddbf909b (HEAD -> project)
Author: shanelle <shanelle1303.be21@chitkara.edu.in>
Date:   Mon May 16 17:00:29 2022 +0530

    rverting Part
```

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

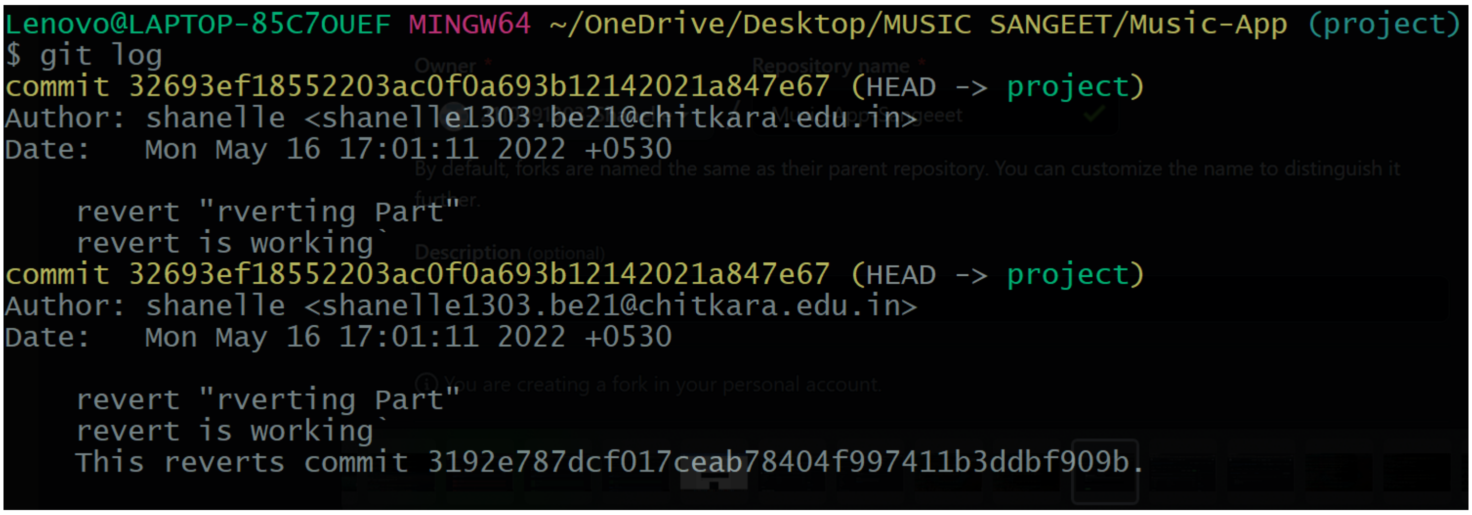
Git revert [commit id without brackets]

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



```
Lenovo@LAPTOP-85C7OUEF MINGW64 ~/OneDrive/Desktop/MUSIC SANGEET/Music-App (project)
$ git revert 3192e787dcf017ceab78404f997411b3ddbf909b
[project 32693ef] revert "rverting Part" revert is working` This reverts commit 3192e787dcf017ceab78404f9
97411b3ddbf909b.
 1 file changed, 84 deletions(-)
```

The git log now should be



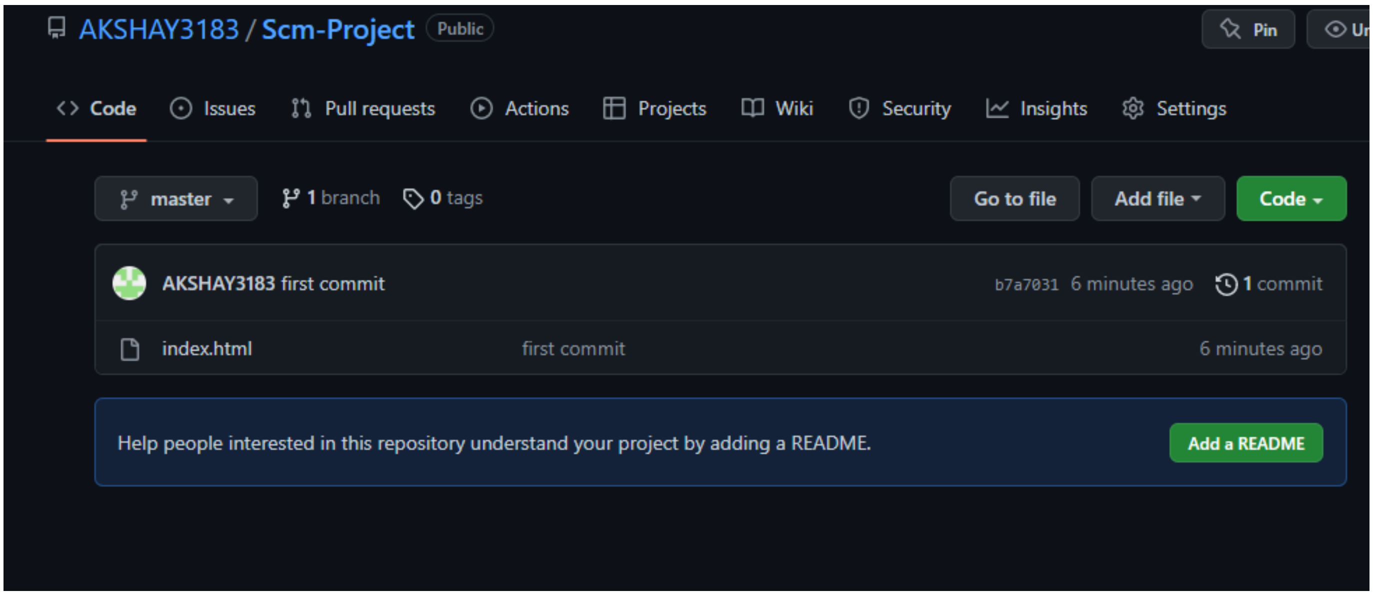
```
Lenovo@LAPTOP-85C7OUEF MINGW64 ~/OneDrive/Desktop/MUSIC SANGEET/Music-App (project)
$ git log
commit 32693ef18552203ac0f0a693b12142021a847e67 (HEAD -> project)
Author: shanelle <shanelle1303.be21@chitkara.edu.in>
Date:   Mon May 16 17:01:11 2022 +0530

    revert "rverting Part"
    revert is working`
commit 32693ef18552203ac0f0a693b12142021a847e67 (HEAD -> project)
Author: shanelle <shanelle1303.be21@chitkara.edu.in>
Date:   Mon May 16 17:01:11 2022 +0530

    revert "rverting Part"
    revert is working`
    This reverts commit 3192e787dcf017ceab78404f997411b3ddbf909b.
```

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 wrongcommits in the history.

Final source code management project :-

Steps-

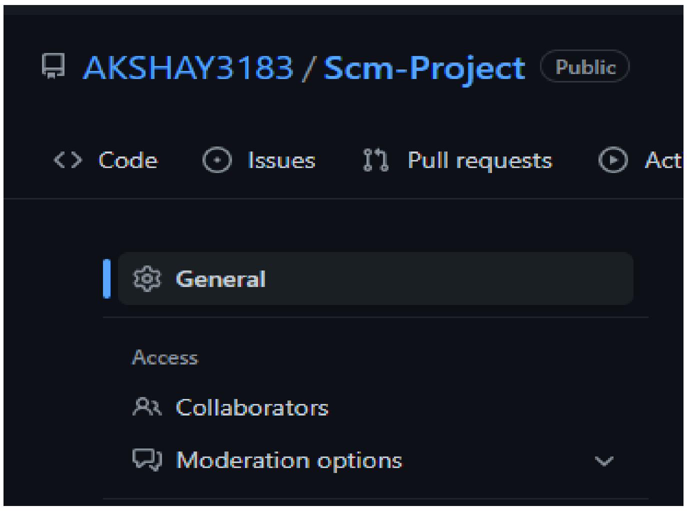
1. Creating a distributed repository and adding people as collaborators.

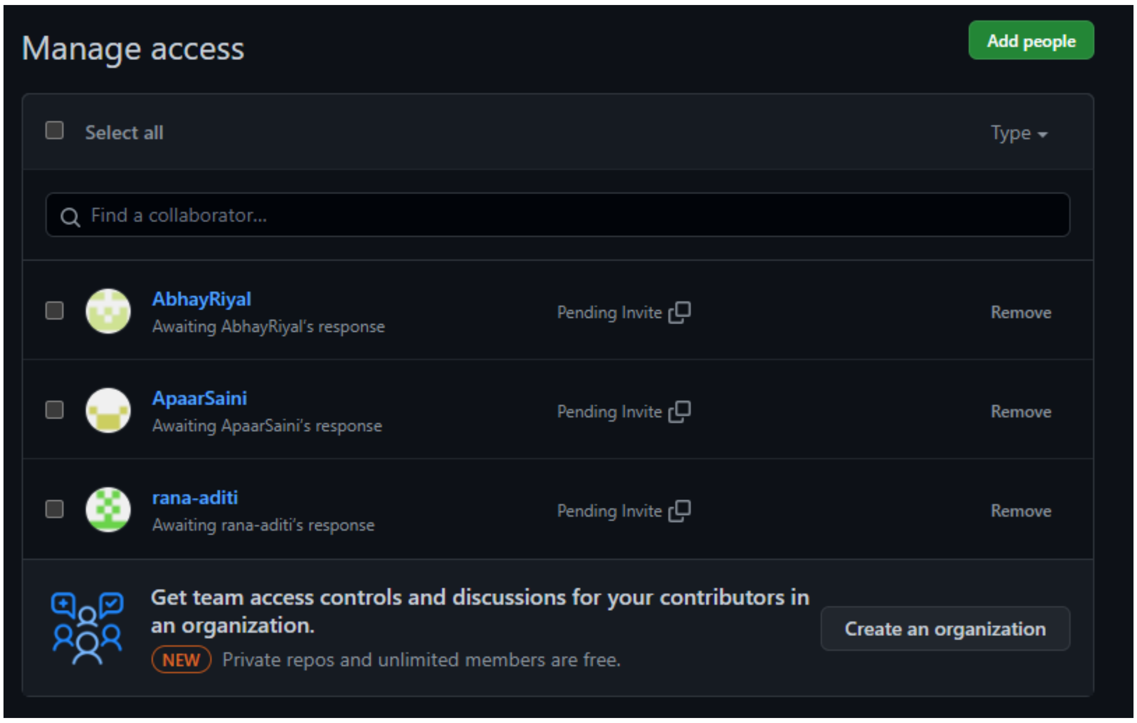


AKSHAY3183 first commit b7a7031 6 minutes ago 1 commit

index.html first commit 6 minutes ago

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




Manage access

Add people

Select all

Type

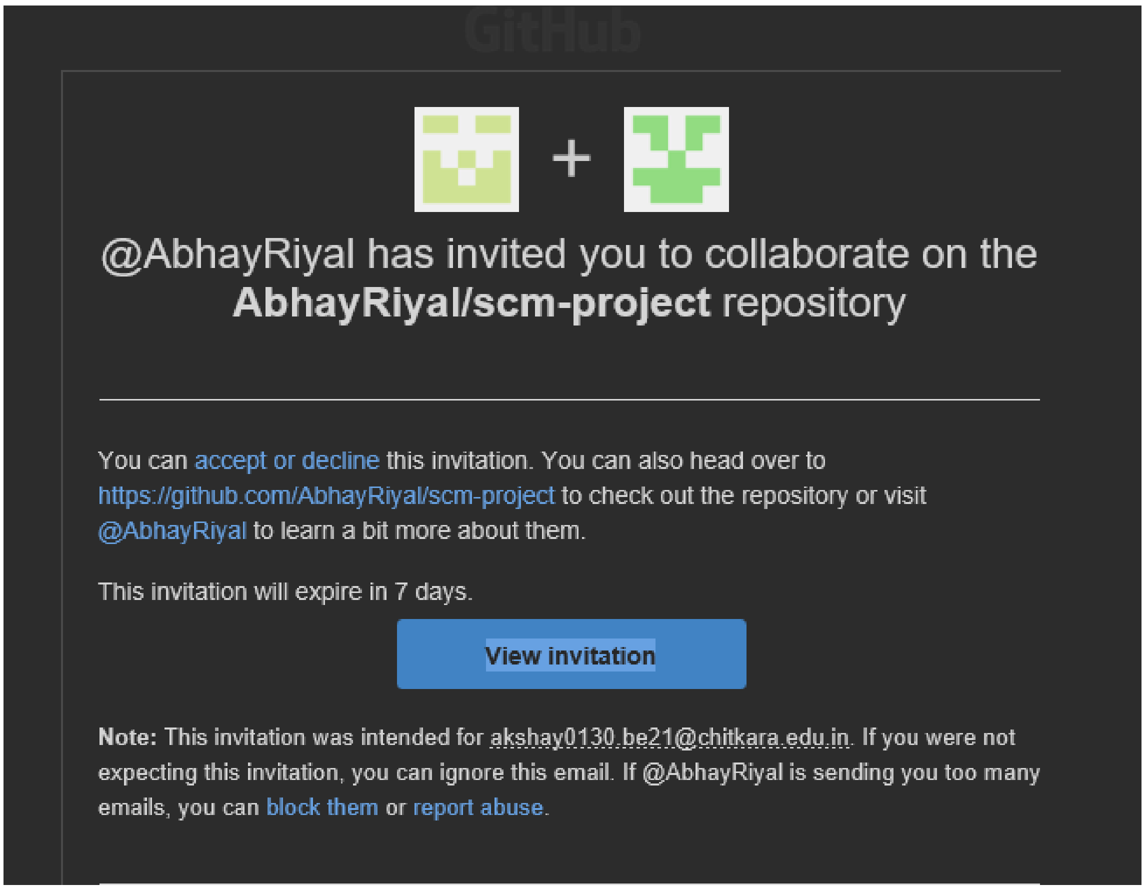
Find a collaborator...

AbhayRiyal Awaiting AbhayRiyal's response	Pending Invite	Remove
ApaarSaini Awaiting ApaarSaini's response	Pending Invite	Remove
rana-aditi Awaiting rana-aditi'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


GitHub

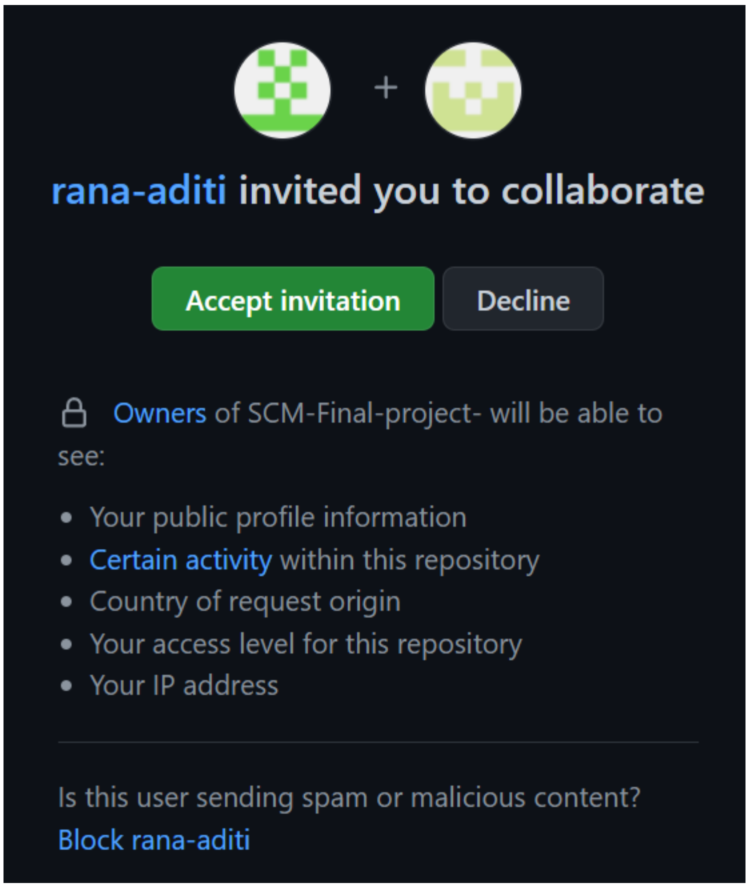
@AbhayRiyal has invited you to collaborate on the **AbhayRiyal/scm-project** repository

You can accept or decline this invitation. You can also head over to https://github.com/AbhayRiyal/scm-project to check out the repository or visit @AbhayRiyal to learn a bit more about them.

This invitation will expire in 7 days.

View invitation

Note: This invitation was intended for akshay0130.be21@chitkara.edu.in. If you were not expecting this invitation, you can ignore this email. If @AbhayRiyal is sending you too many emails, you can block them or report abuse.



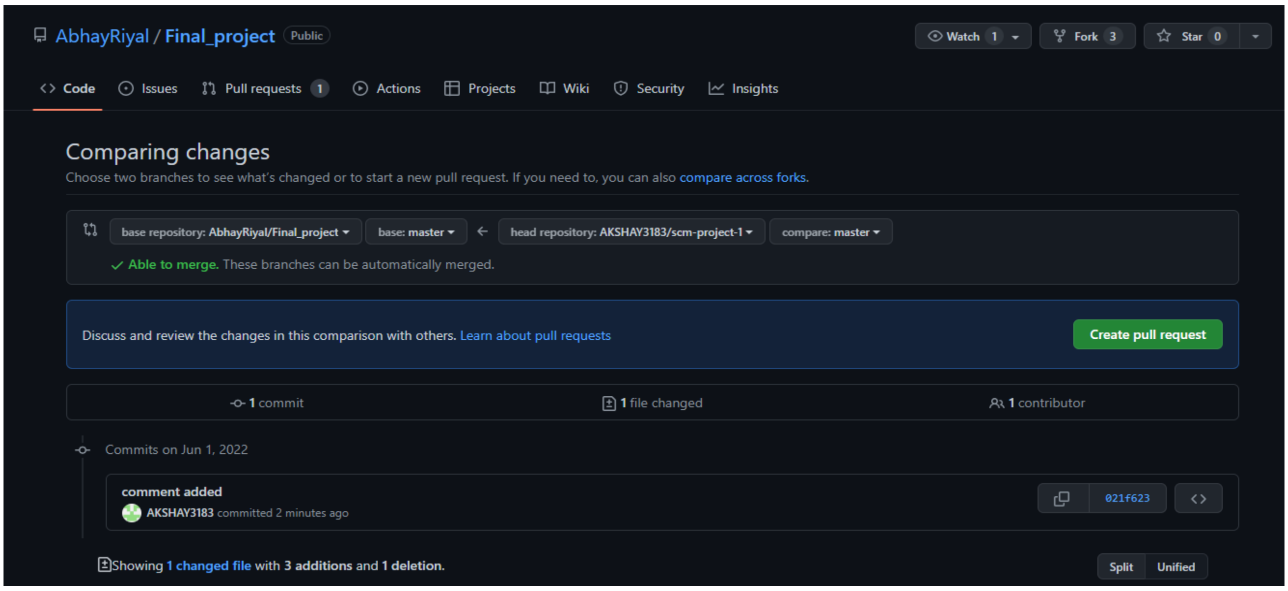
rana-aditi invited you to collaborate

Owners of SCM-Final-project- will be able to see:

- Your public profile information
- Certain activity within this repository
- Country of request origin
- Your access level for this repository
- Your IP address

Is this user sending spam or malicious content?
Block rana-aditi

1. Opening pull request for other team members projects.



AbhayRiyal / **Final_project** Public

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/Final_project   base: master ← head repository: AKSHAY3183/scm-project-1   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

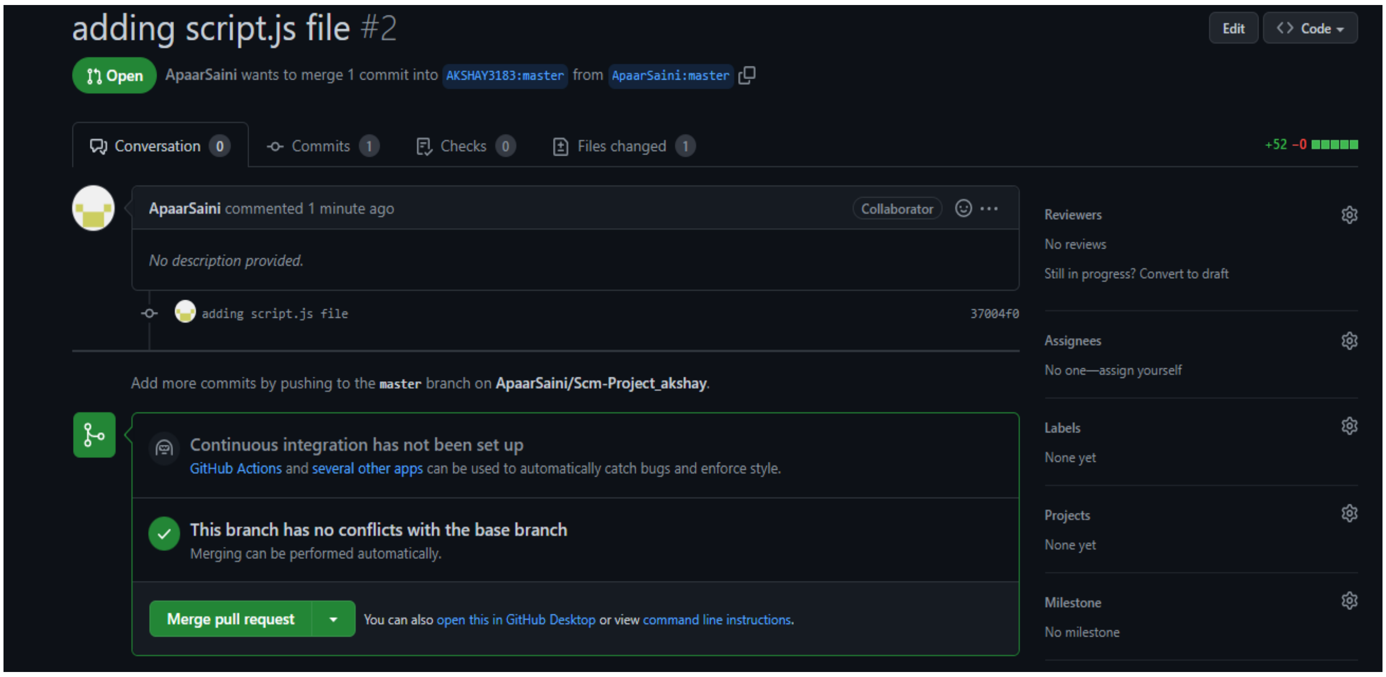
1 commit   1 file changed   1 contributor

Commits on Jun 1, 2022

comment added
AKSHAY3183 committed 2 minutes ago   021f623

Showing 1 changed file with 3 additions and 1 deletion.

1. Closing and merging pull requests as a maintainer.



adding script.js file #2

Open ApaarSaini wants to merge 1 commit into AKSHAY3183:master from ApaarSaini:master

Conversation 0 · Commits 1 · Checks 0 · Files changed 1 · +52 −0

ApaarSaini commented 1 minute ago — Collaborator

No description provided.

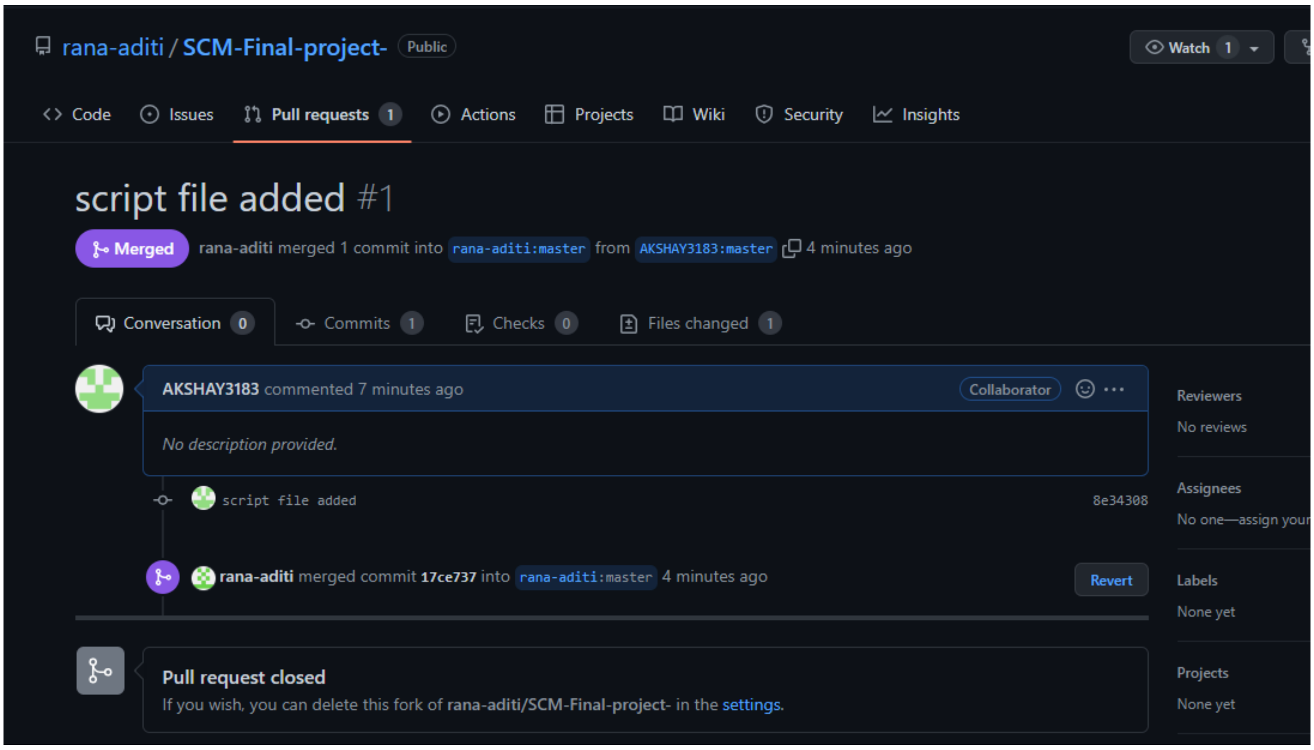
adding script.js file 37004f0

Add more commits by pushing to the master branch on ApaarSaini/Scm-Project_akshay.

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.



rana-aditi / SCM-Final-project- Public

script file added #1

Merged rana-aditi merged 1 commit into rana-aditi:master from AKSHAY3183:master 4 minutes ago

Conversation 0 · Commits 1 · Checks 0 · Files changed 1

AKSHAY3183 commented 7 minutes ago — Collaborator

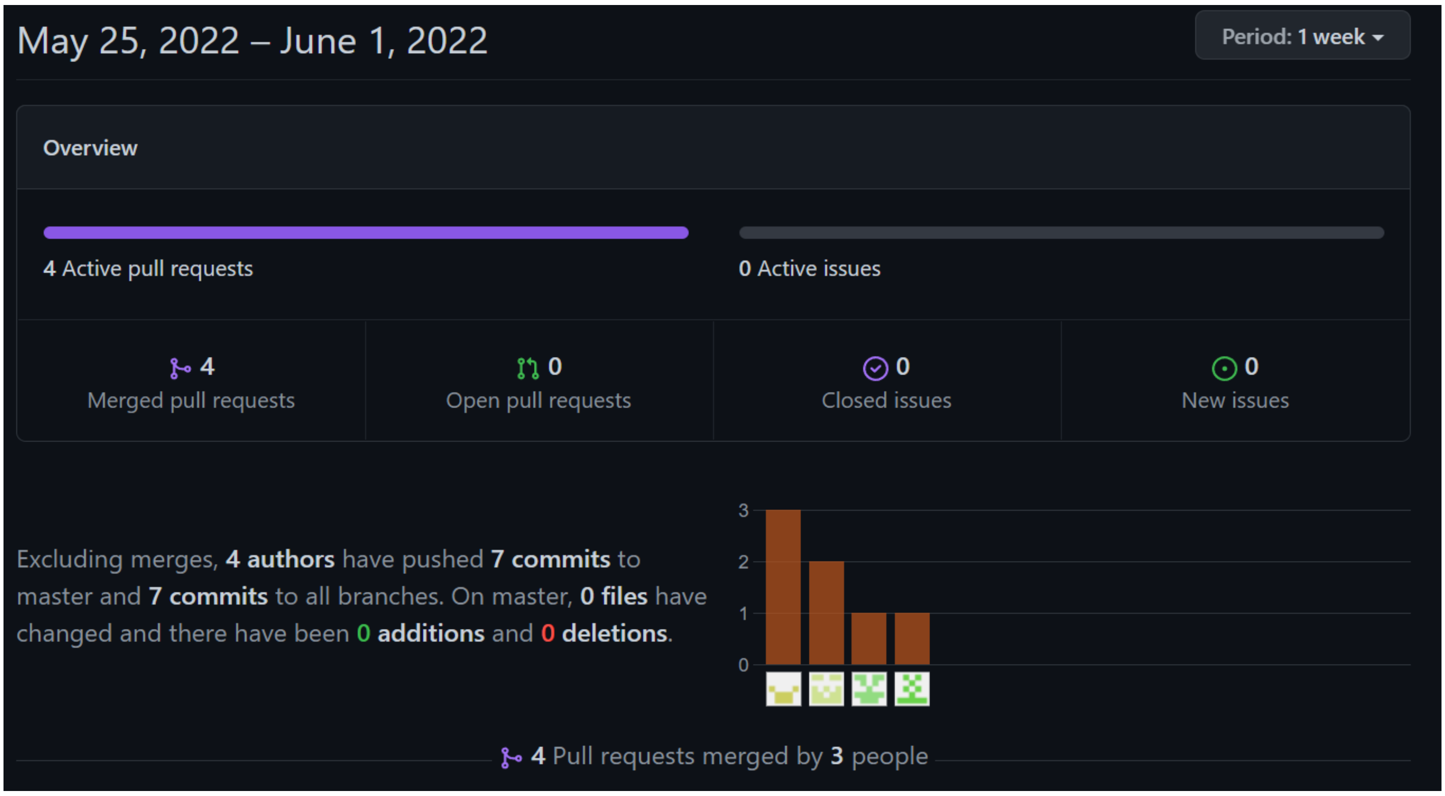
No description provided.

script file added 8e34308

rana-aditi merged commit 17ce737 into rana-aditi:master 4 minutes ago

Pull request closed
If you wish, you can delete this fork of rana-aditi/SCM-Final-project- in the settings.

2. Network graph as maintainer.


May 25, 2022 – June 1, 2022

Period: 1 week

Overview

4 Active pull requests

0 Active issues

4 Merged pull requests

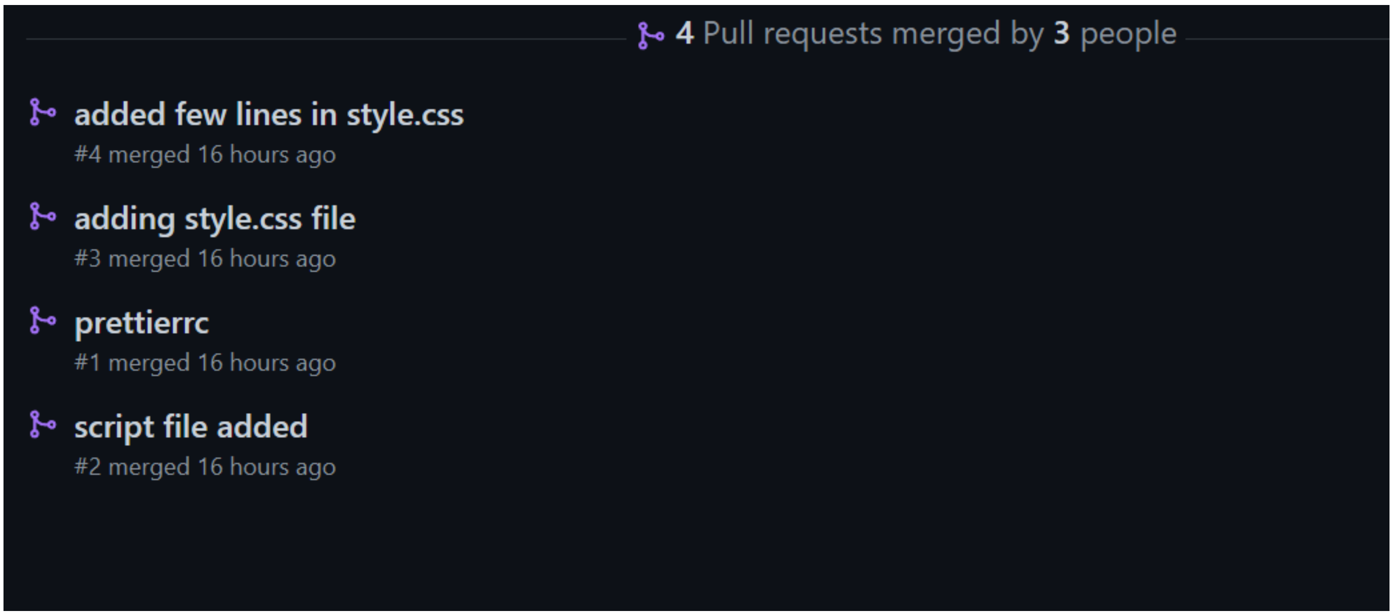
0 Open pull requests

0 Closed issues

0 New issues

Excluding merges, **4 authors** have pushed **7 commits** to master and **7 commits** to all branches. On master, **0 files** have changed and there have been **0 additions** and **0 deletions**.

4 Pull requests merged by **3** people


4 Pull requests merged by **3** people

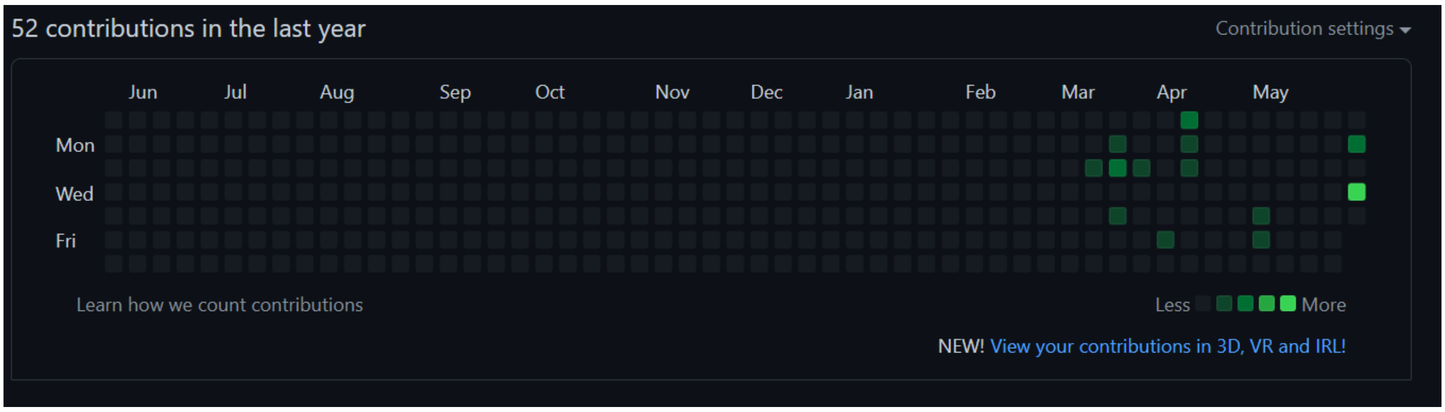
added few lines in style.css
#4 merged 16 hours ago

adding style.css file
#3 merged 16 hours ago

prettierrc
#1 merged 16 hours ago

script file added
#2 merged 16 hours ago

3. Contribution graph for whole year.


52 contributions in the last year

Contribution settings
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Jan
Feb
Mar
Apr
May
Mon
Wed
Fri
Learn how we count contributions
Less
More
NEW! View your contributions in 3D, VR and IRL!