Subject Name: **Source Code Management**

Subject Code: **CS181**

Cluster: **Beta**

Department: **CSE**

CHITKARA UNIVERSITY

Submitted by- Abhay Riyal

Submitted to- Dr. Monit Kapoor

INDEX

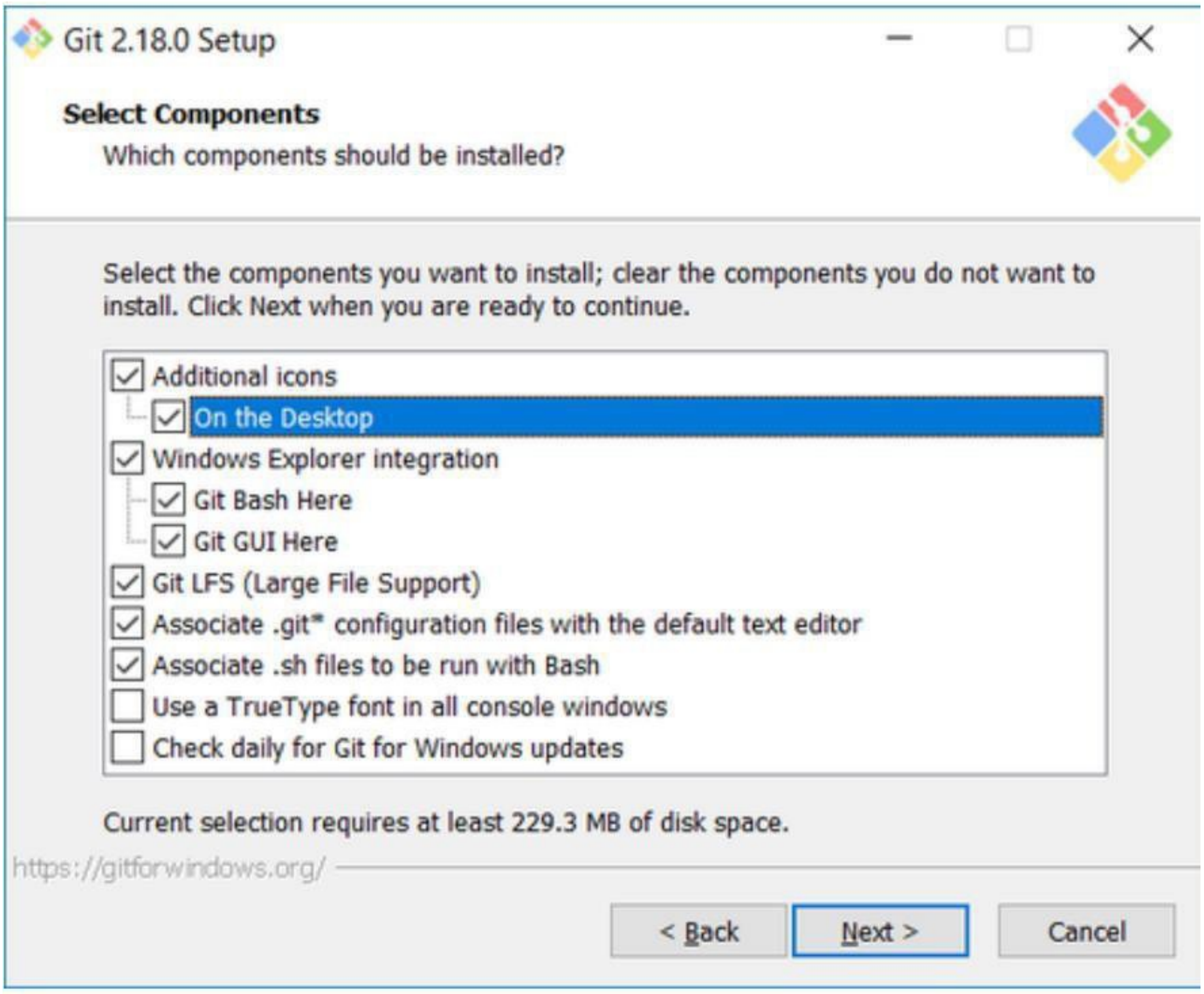
Experiment No. 01

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?

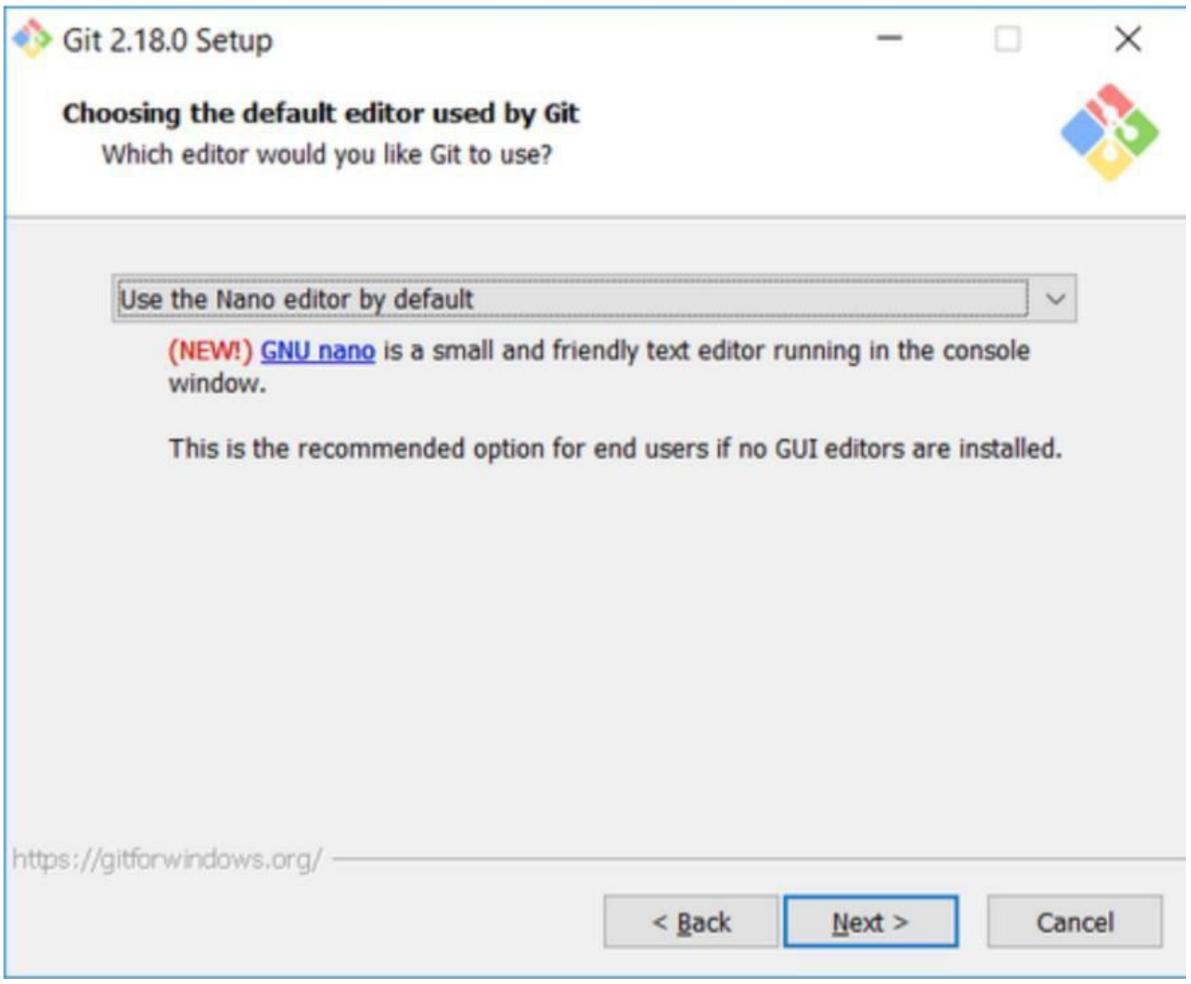
Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

- [✓] 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

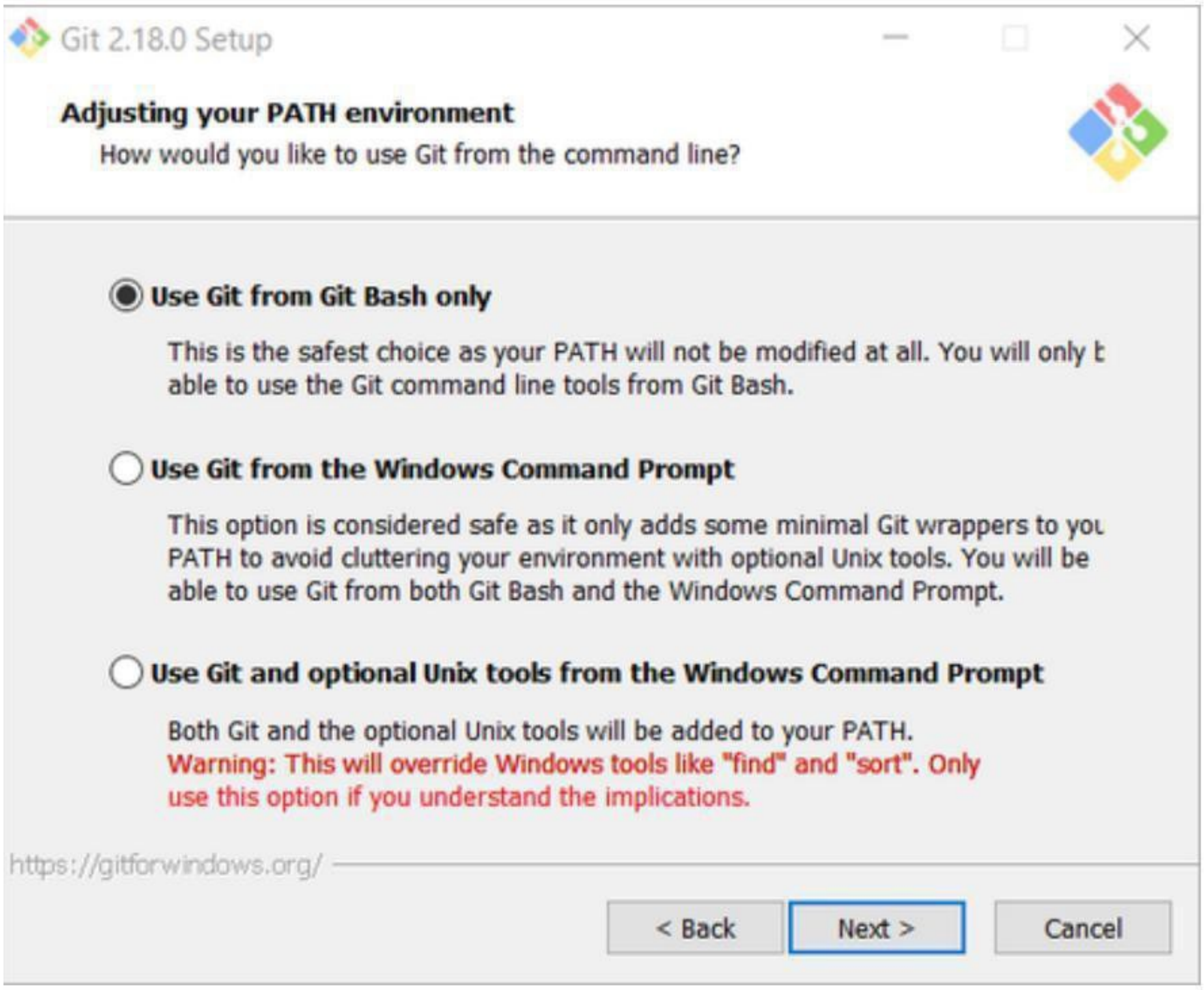
(NEW!) GNU nano is a small and friendly text editor running in the console window.

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 t able to use the Git command line tools from Git Bash.

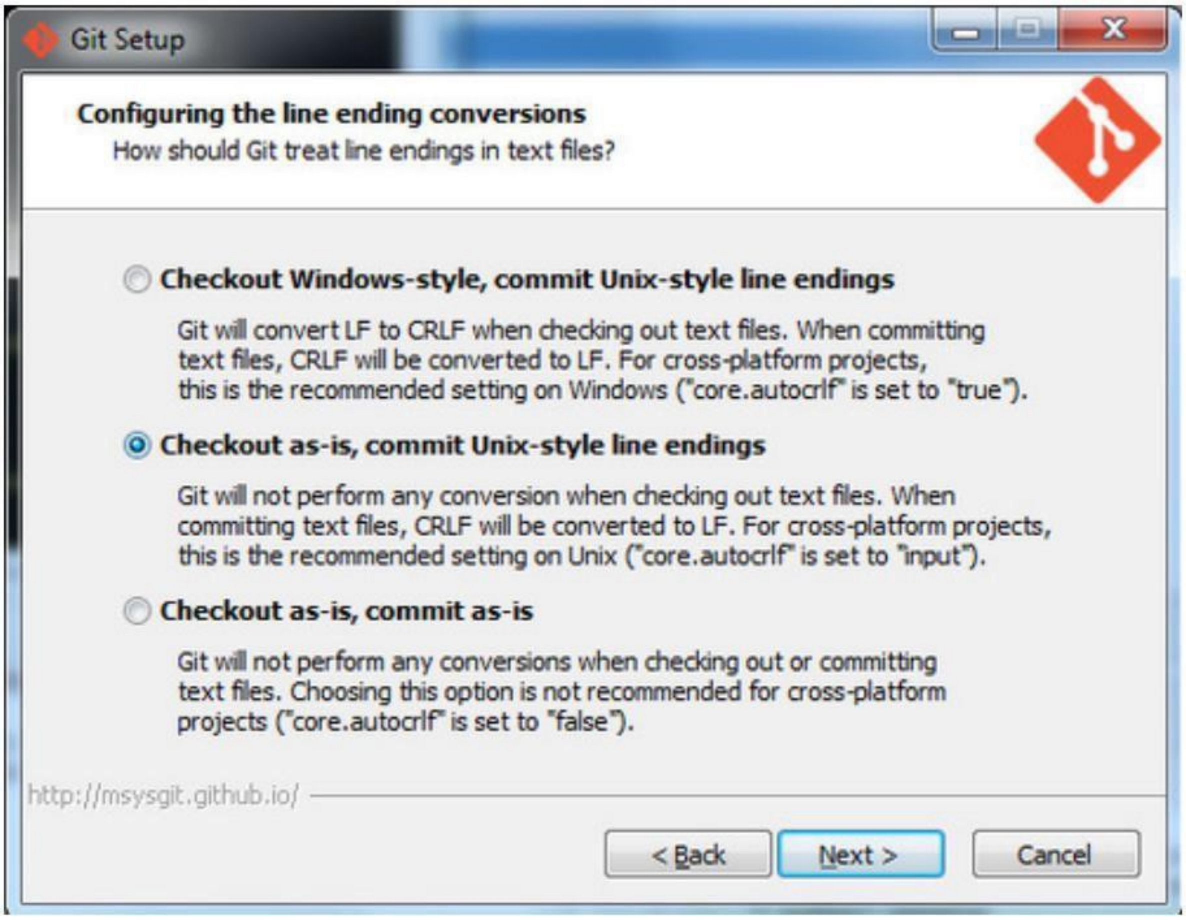
○ Use Git from the Windows Command Prompt

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.


Git Setup

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 “abhay0038.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 "AbhayRiyal"`

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

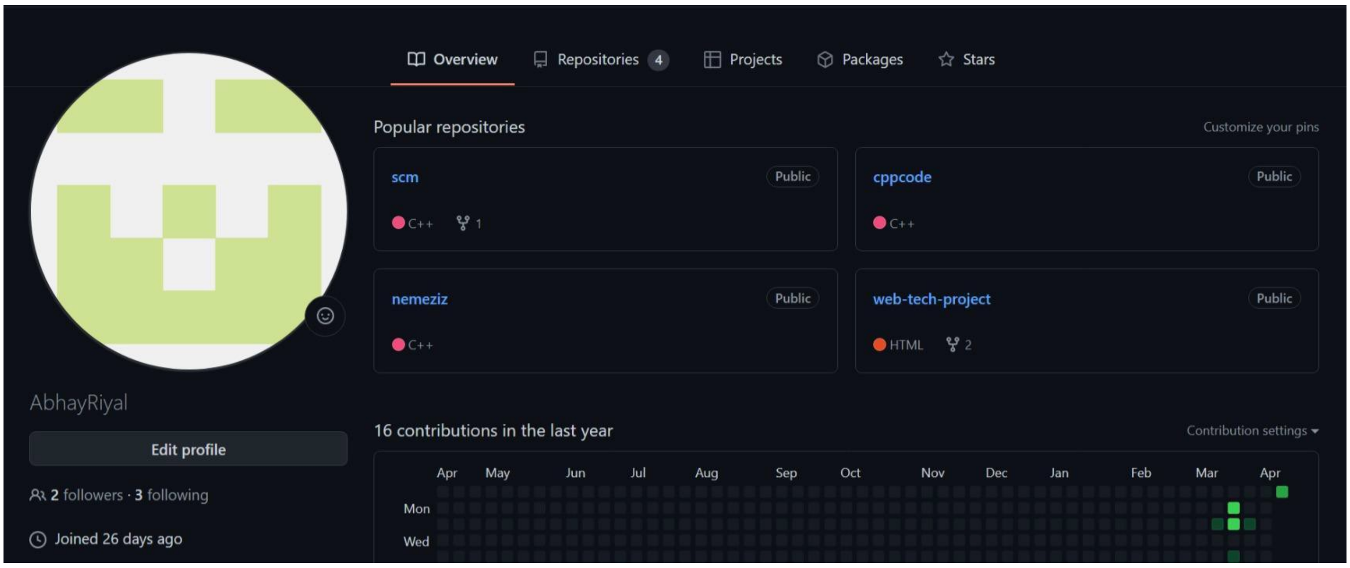
Setting up GitHub Account

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.

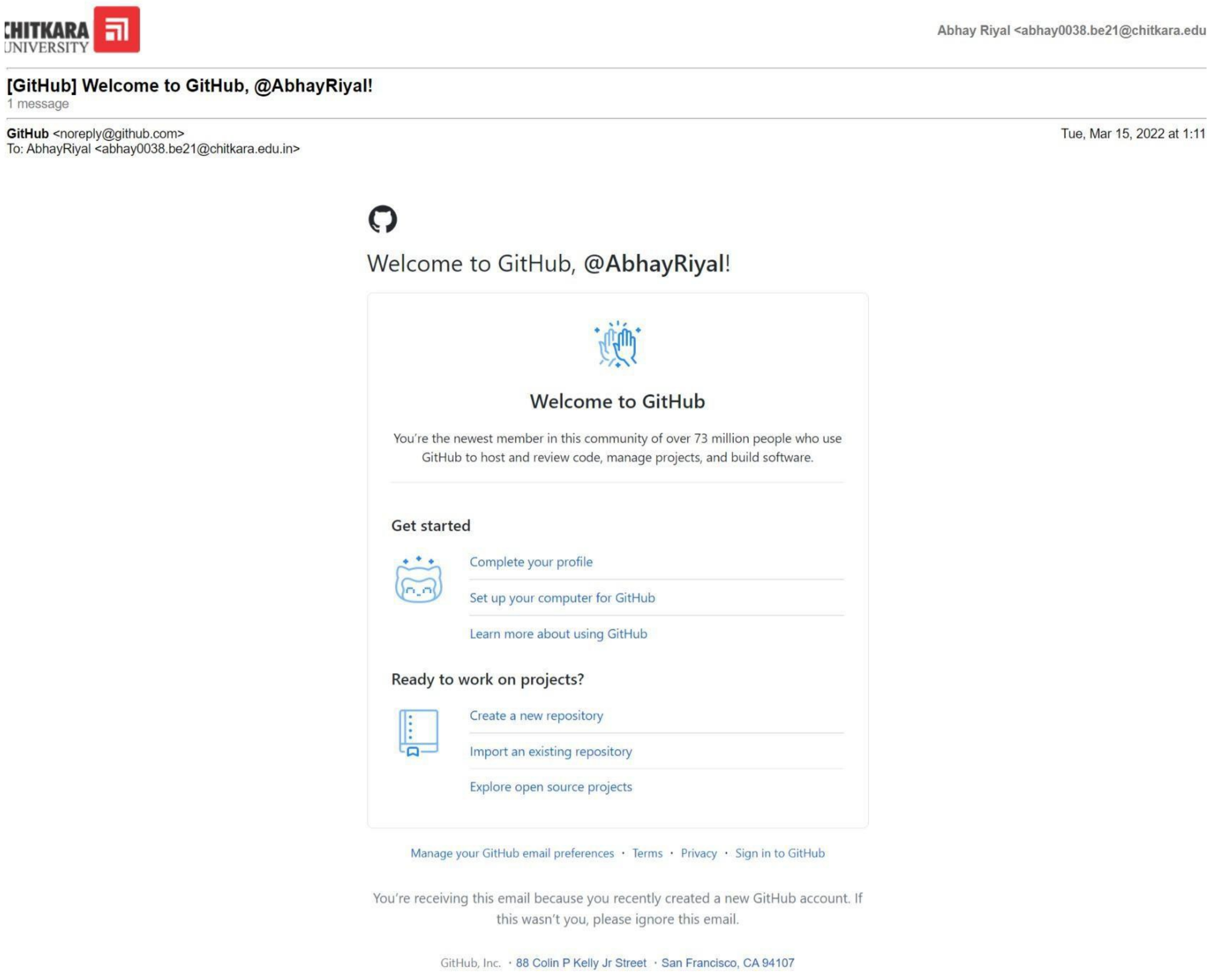
1. **Creating an account:** To sign up for an account on GitHub.com, navigate to https://github.com/ and follow the prompts.

   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".



Abhay Riyal <abhay0038.be21@chitkara.edu

[GitHub] Welcome to GitHub, @AbhayRiyal!
1 message

GitHub <noreply@github.com>
To: AbhayRiyal <abhay0038.be21@chitkara.edu.in>

Tue, Mar 15, 2022 at 1:11

Welcome to GitHub, **@AbhayRiyal**!

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

Ready to work on projects?

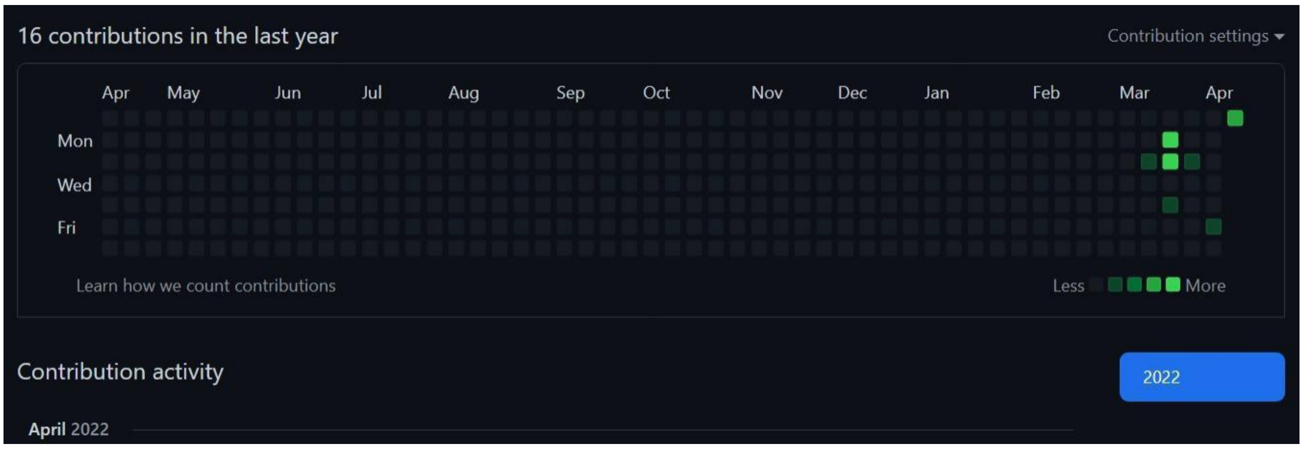
Create a new repository
Import an existing repository
Explore open source projects

Manage your GitHub email preferences · Terms · Privacy · Sign in to GitHub

You're receiving this email because you recently created a new GitHub account. If this wasn't you, please ignore this email.

GitHub, Inc. · 88 Colin P Kelly Jr Street · San Francisco, CA 94107

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

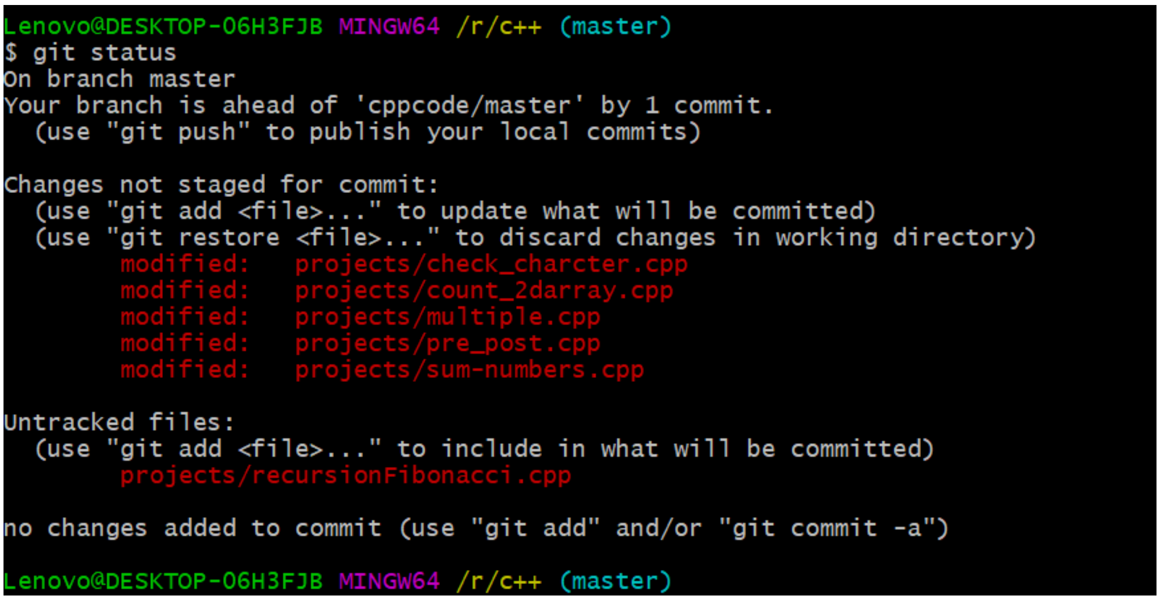
Aim: Program to generate logs

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.

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.

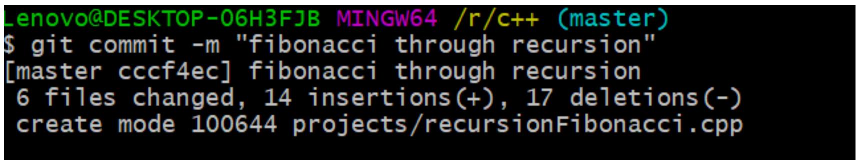


```
Lenovo@DESKTOP-06H3FJB MINGW64 /r/c++ (master)
$ git status
On branch master
Your branch is ahead of 'cppcode/master' by 1 commit.
  (use "git push" to publish your local commits)

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:   projects/check_charcter.cpp
        modified:   projects/count_2darray.cpp
        modified:   projects/multiple.cpp
        modified:   projects/pre_post.cpp
        modified:   projects/sum-numbers.cpp

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

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

Lenovo@DESKTOP-06H3FJB MINGW64 /r/c++ (master)
```

Basic Git commit

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

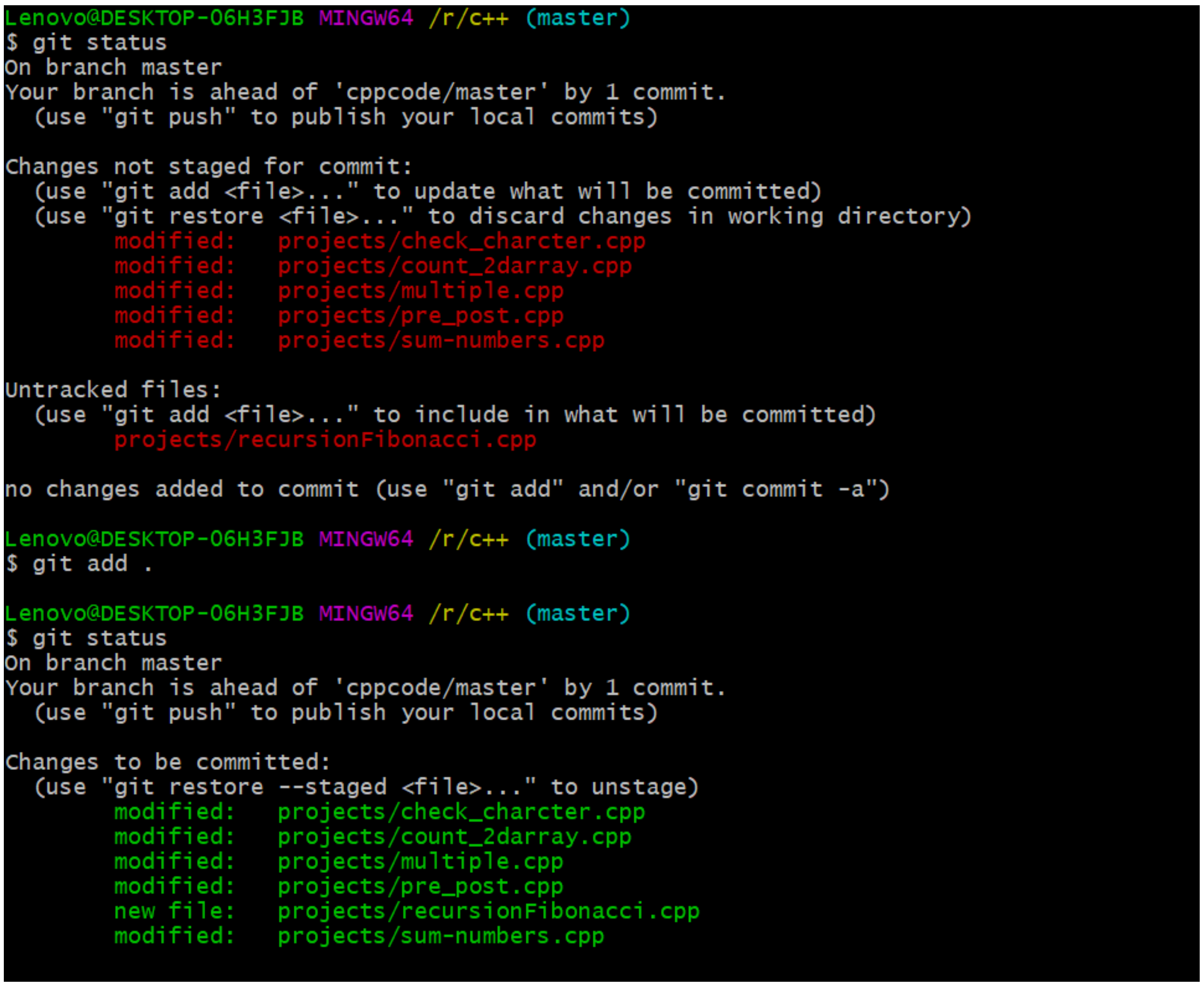
Committed snapshots can be thought of as “safe” versions of a project—Git will never change them unless you explicitly ask it to. Prior to the execution of `git commit`, The git add command is used to promote or 'stage' changes to the project that will be stored in a commit. These two commands `git commit`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



```
Lenovo@DESKTOP-06H3FJB MINGW64 /r/c++ (master)
$ git commit -m "fibonacci through recursion"
[master cccf4ec] fibonacci through recursion
 6 files changed, 14 insertions(+), 17 deletions(-)
 create mode 100644 projects/recursionFibonacci.cpp
```

Basic Git add command

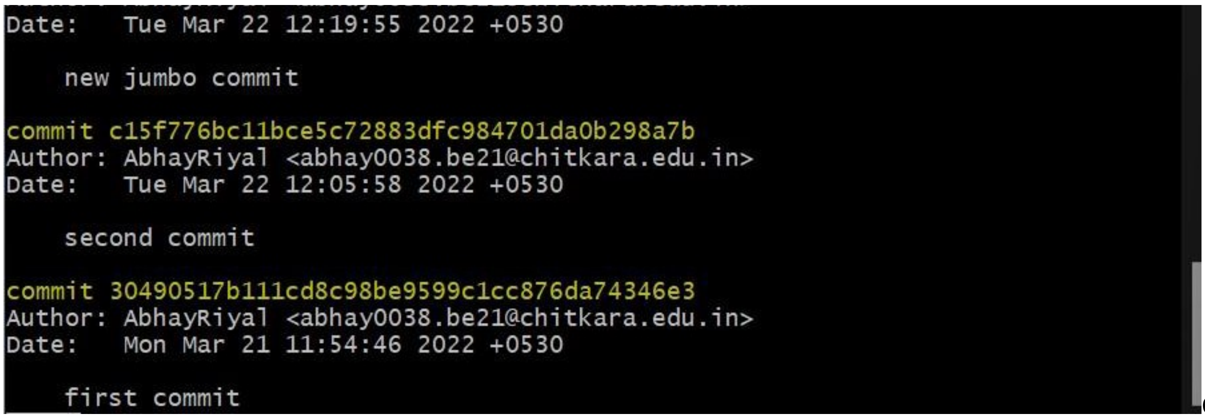
The git add command adds a change in the working directory to the staging area. It tells Git that you want to include updates to a particular file in the next commit. However, git add doesn't really affect the repository in any significant way—changes are not actually recorded until you run git commit



```
Lenovo@DESKTOP-06H3FJB MINGW64 /r/c++ (master)
$ git status
On branch master
Your branch is ahead of 'cppcode/master' by 1 commit.
  (use "git push" to publish your local commits)

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:   projects/check_charcter.cpp
        modified:   projects/count_2darray.cpp
        modified:   projects/multiple.cpp
        modified:   projects/pre_post.cpp
        modified:   projects/sum-numbers.cpp

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

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

Lenovo@DESKTOP-06H3FJB MINGW64 /r/c++ (master)
$ git add .

Lenovo@DESKTOP-06H3FJB MINGW64 /r/c++ (master)
$ git status
On branch master
Your branch is ahead of 'cppcode/master' by 1 commit.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   projects/check_charcter.cpp
        modified:   projects/count_2darray.cpp
        modified:   projects/multiple.cpp
        modified:   projects/pre_post.cpp
        new file:   projects/recursionFibonacci.cpp
        modified:   projects/sum-numbers.cpp
```

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:



```
Date:   Tue Mar 22 12:19:55 2022 +0530

    new jumbo commit

commit c15f776bc11bce5c72883dfc984701da0b298a7b
Author: AbhayRiyal <abhay0038.be21@chitkara.edu.in>
Date:   Tue Mar 22 12:05:58 2022 +0530

    second commit

commit 30490517b111cd8c98be9599c1cc876da74346e3
Author: AbhayRiyal <abhay0038.be21@chitkara.edu.in>
Date:   Mon Mar 21 11:54:46 2022 +0530

    first commit
```

cd

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"

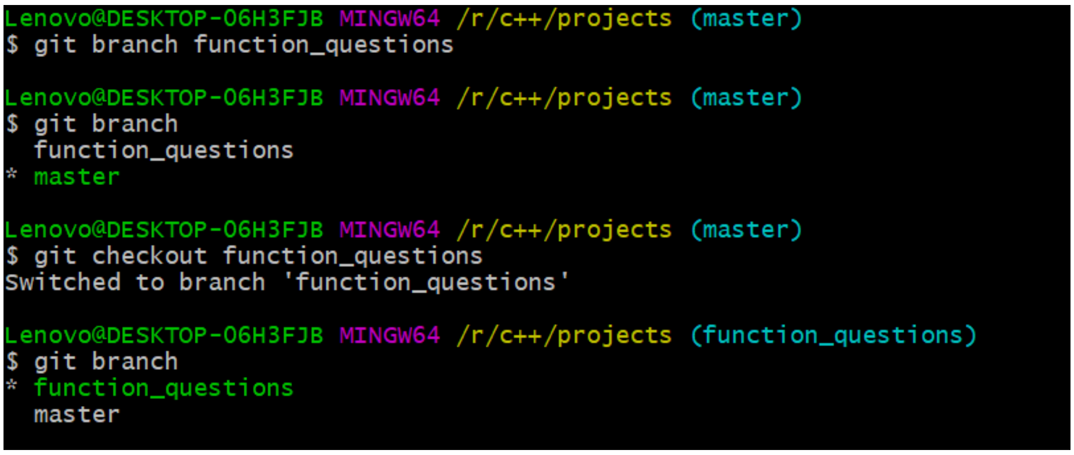
Visualizing Branches:

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.

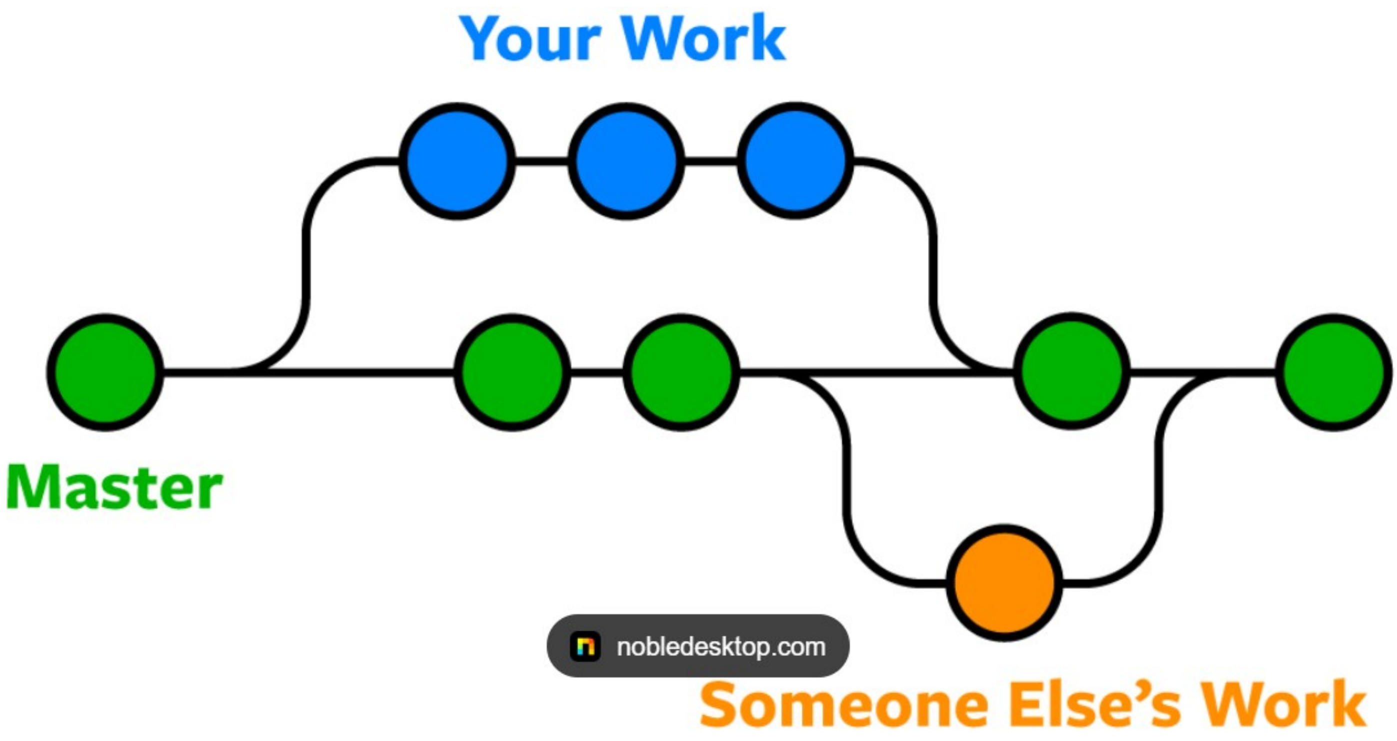
After this I have done the 3 Step architecture which is tracking the file, send it to stagging area and finally we can rollback to any previously saved version of this file.

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.



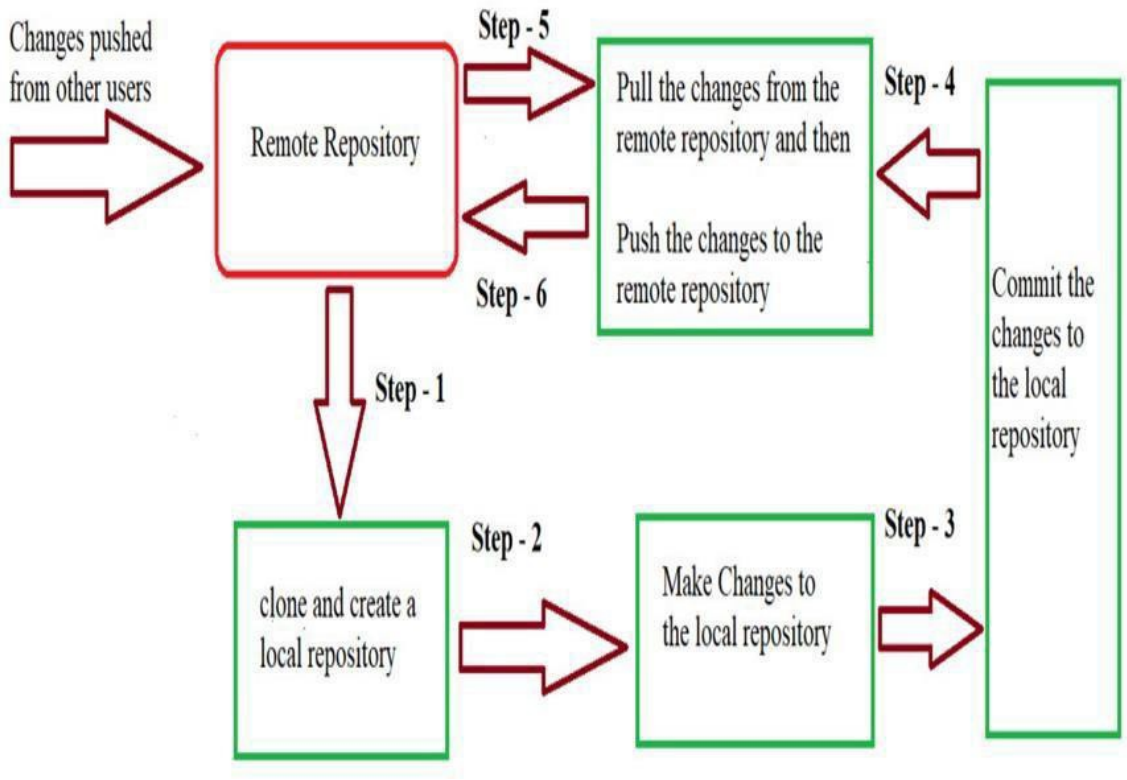
```
Lenovo@DESKTOP-06H3FJB MINGW64 /r/c++/projects (master)
$ git branch function_questions

Lenovo@DESKTOP-06H3FJB MINGW64 /r/c++/projects (master)
$ git branch
  function_questions
* master

Lenovo@DESKTOP-06H3FJB MINGW64 /r/c++/projects (master)
$ git checkout function_questions
Switched to branch 'function_questions'

Lenovo@DESKTOP-06H3FJB MINGW64 /r/c++/projects (function_questions)
$ git branch
* function_questions
  master
```


Your Work
Master
nobledesktop.com
Someone Else's Work

Experiment No. 05

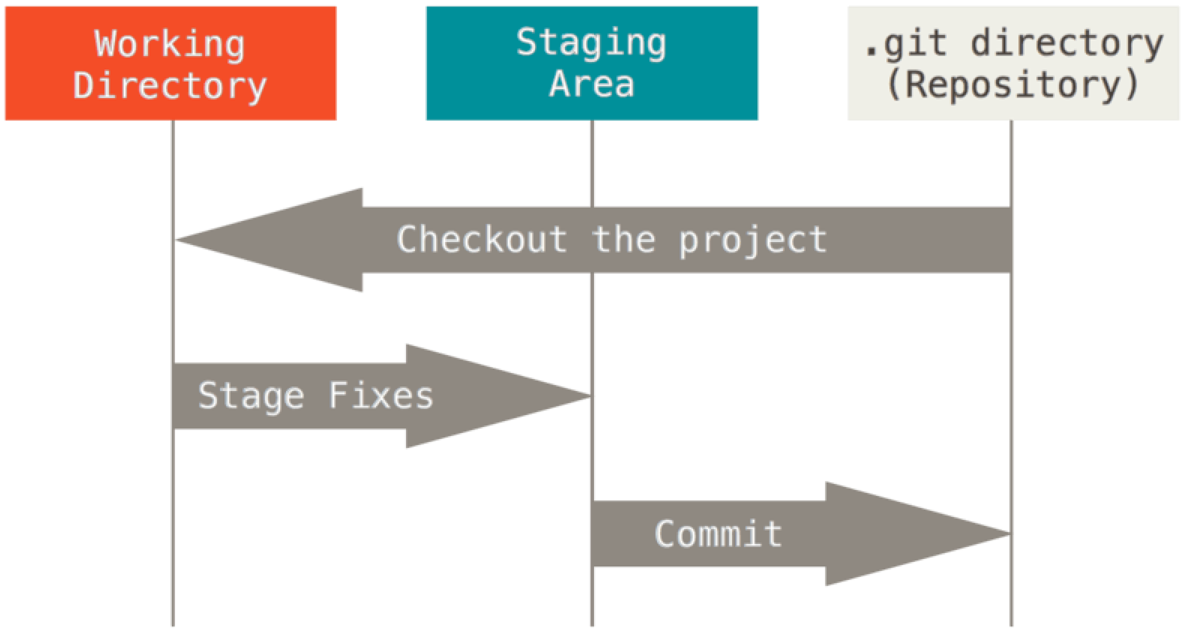
Aim: Git lifecycle description

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
clone and create a local repository
Make Changes to the local repository
Step - 3

- **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

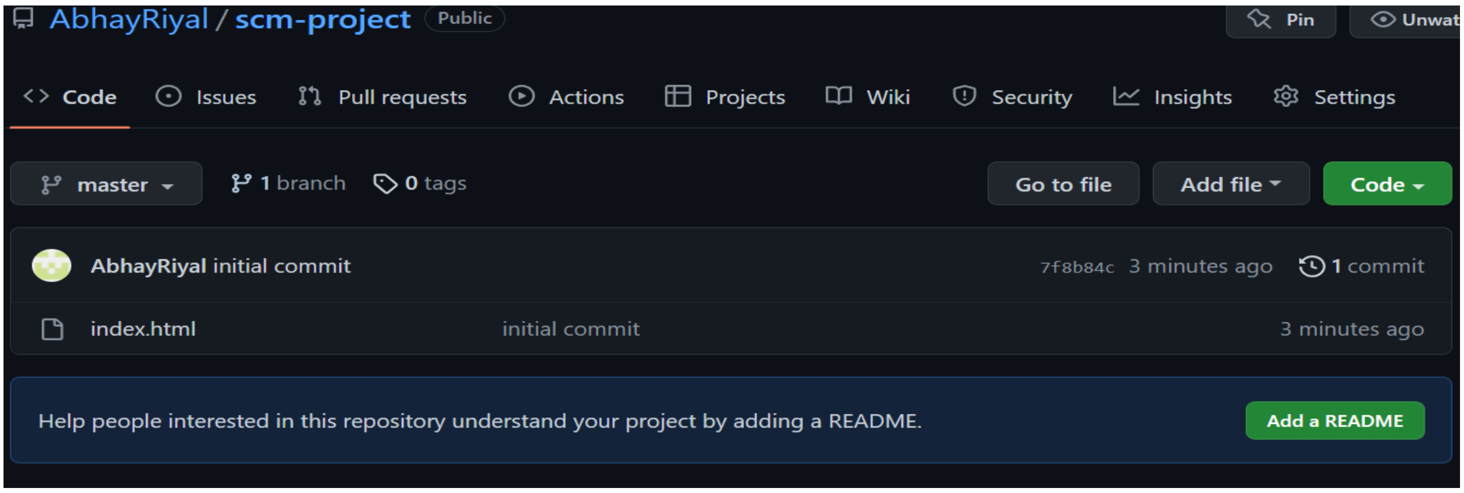
In GitHub, We can invite other GitHub users to become collaborators to our private repositories(which expire after 7 days if not accepted, restoring any unclaimed licenses). Being a collaborator, of a personal repository you can pull (read) the contents of the repository and push (write) changes to the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

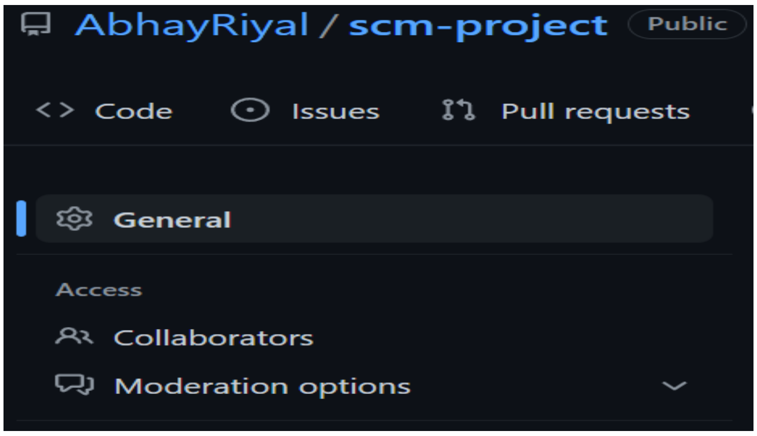
Step 2: Open the repo on which you wish to add collaborators

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



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

Step 4: Select Collaborators on left side-bar



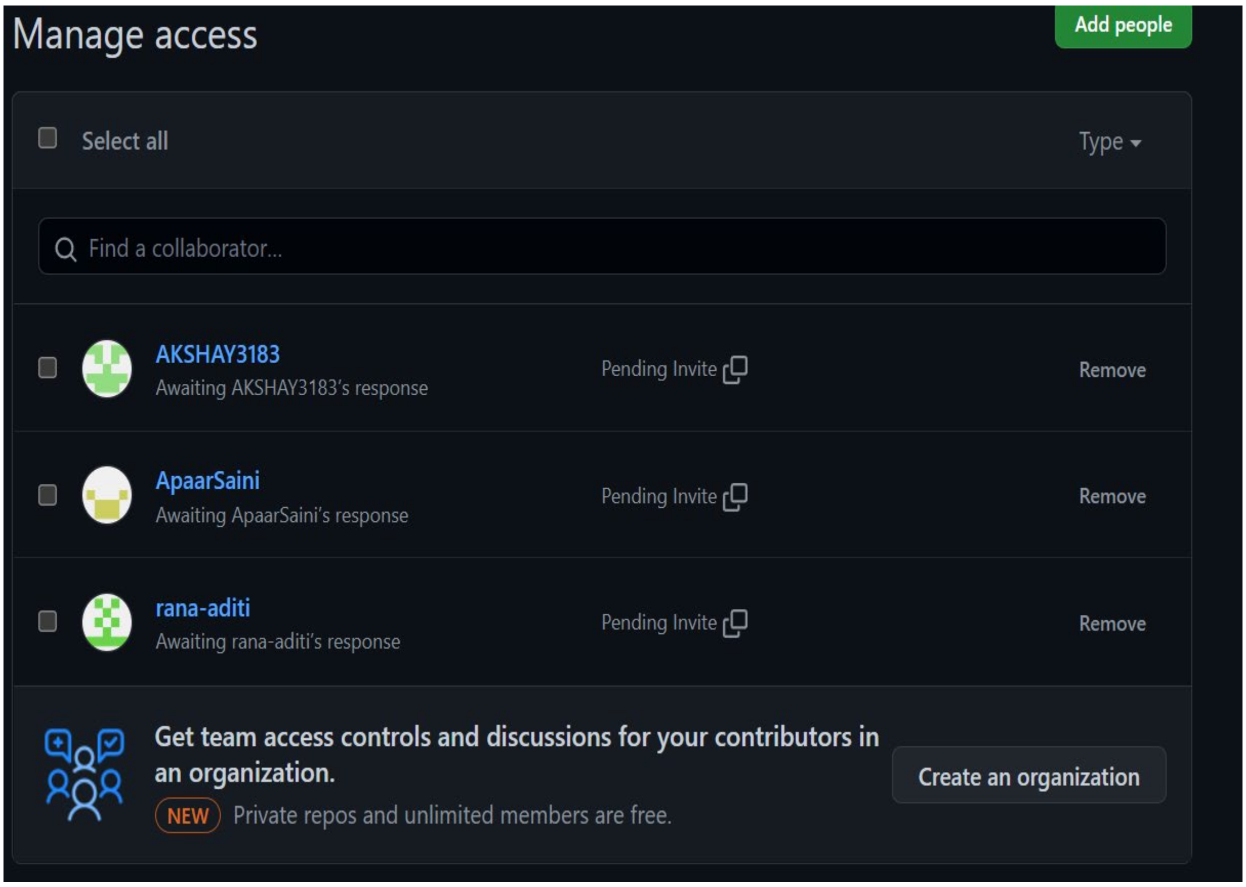
Step 5: It might ask your GitHub account password saying "Sudo mode" is going to be activated, if asked, enter

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 theperson you want to add as a collaborator

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



Manage access

Add people

Select all

Type

Find a collaborator...

AKSHAY3183
Awaiting AKSHAY3183's response
Pending Invite
Remove

ApaarSaini
Awaiting ApaarSaini's response
Pending Invite
Remove

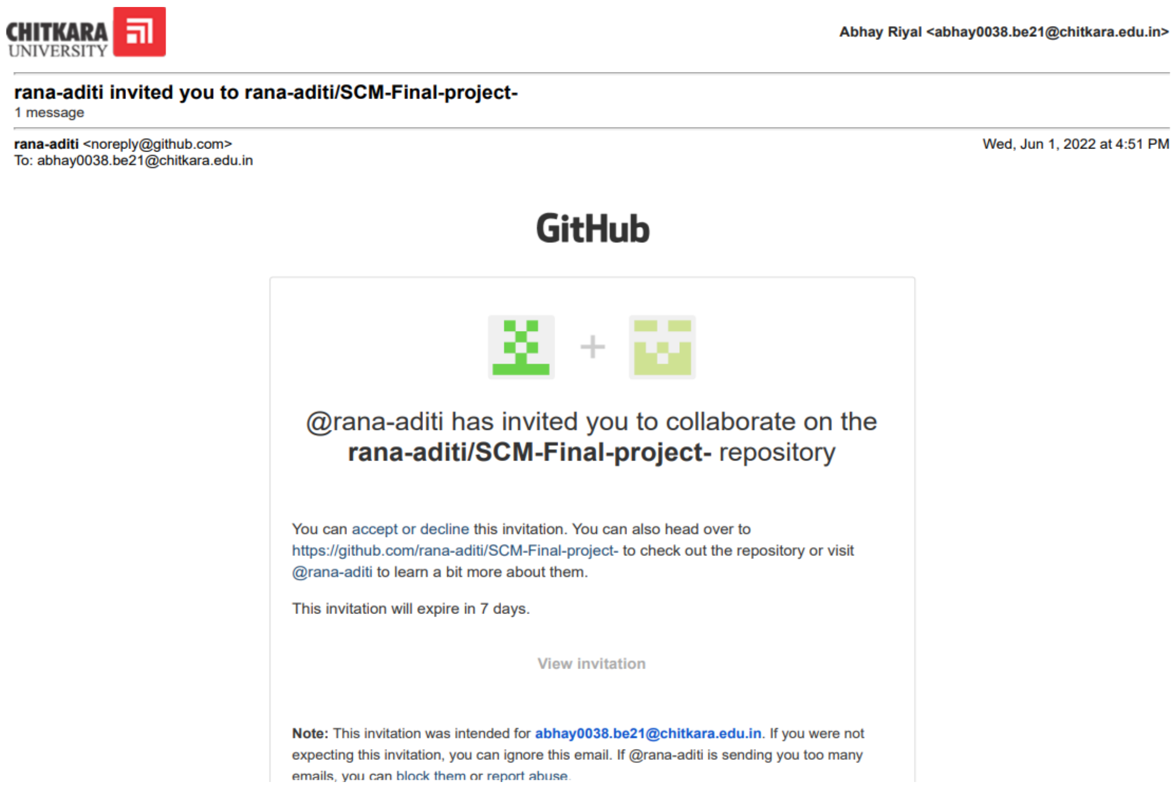
rana-aditi
Awaiting rana-aditi's response
Pending Invite
Remove

Get team access controls and discussions for your contributors in an organization.

NEW Private repos and unlimited members are free.

Create an organization

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 theRepository!



Abhay Riyal <abhay0038.be21@chitkara.edu.in>

rana-aditi invited you to rana-aditi/SCM-Final-project-
1 message

rana-aditi <noreply@github.com>
To: abhay0038.be21@chitkara.edu.in

Wed, Jun 1, 2022 at 4:51 PM

GitHub

@rana-aditi has invited you to collaborate on the **rana-aditi/SCM-Final-project-** repository

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

This invitation will expire in 7 days.

View invitation

Note: This invitation was intended for **abhay0038.be21@chitkara.edu.in**. If you were not expecting this invitation, you can ignore this email. If @rana-aditi 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:

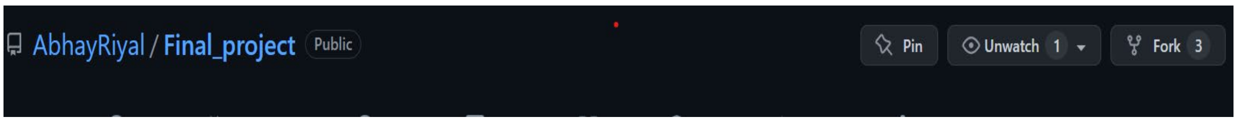
1. You have your own copy of the project on which you may test your own changeswithout changing the original project.

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.

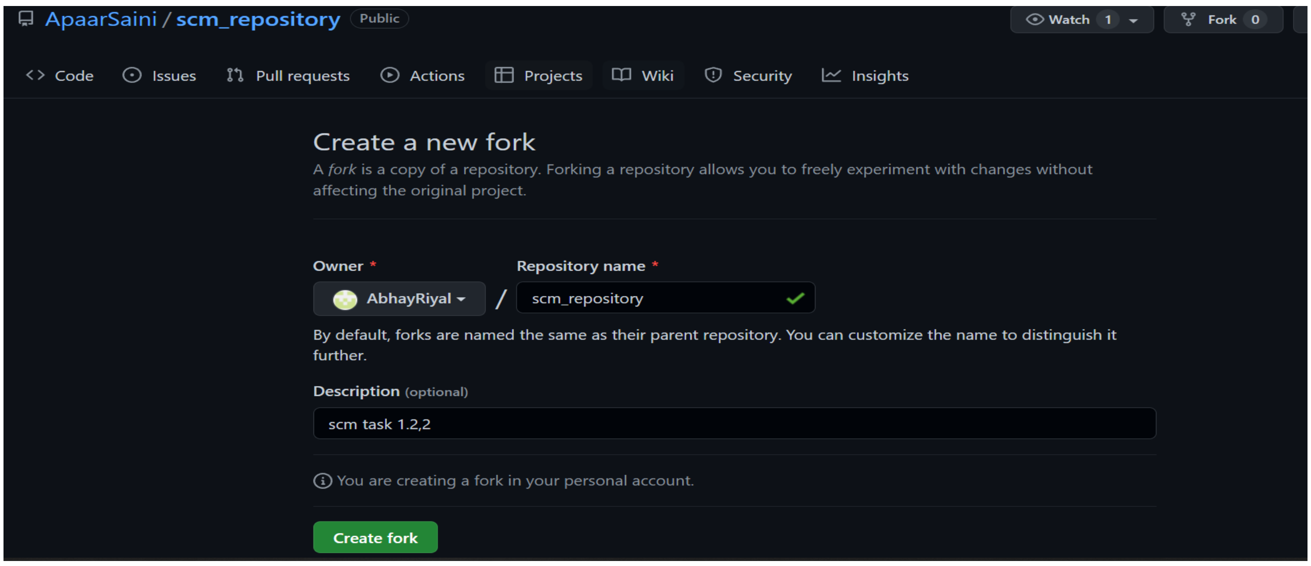
3. When you clone an Open Source project, which isn't yours, you don't have the rightto push code directly into the project.

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 * AbhayRiyal / Repository name * scm_repository

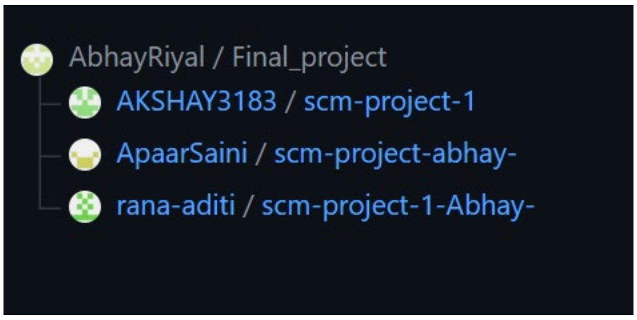
By default, forks are named the same as their parent repository. You can customize the name to distinguish it further.

Description (optional)

scm task 1.2,2

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!


AbhayRiyal / Final_project
AKSHAY3183 / scm-project-1
ApaarSaini / scm-project-abhay-
rana-aditi / scm-project-1-Abhay-

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 theauthor, 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 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.

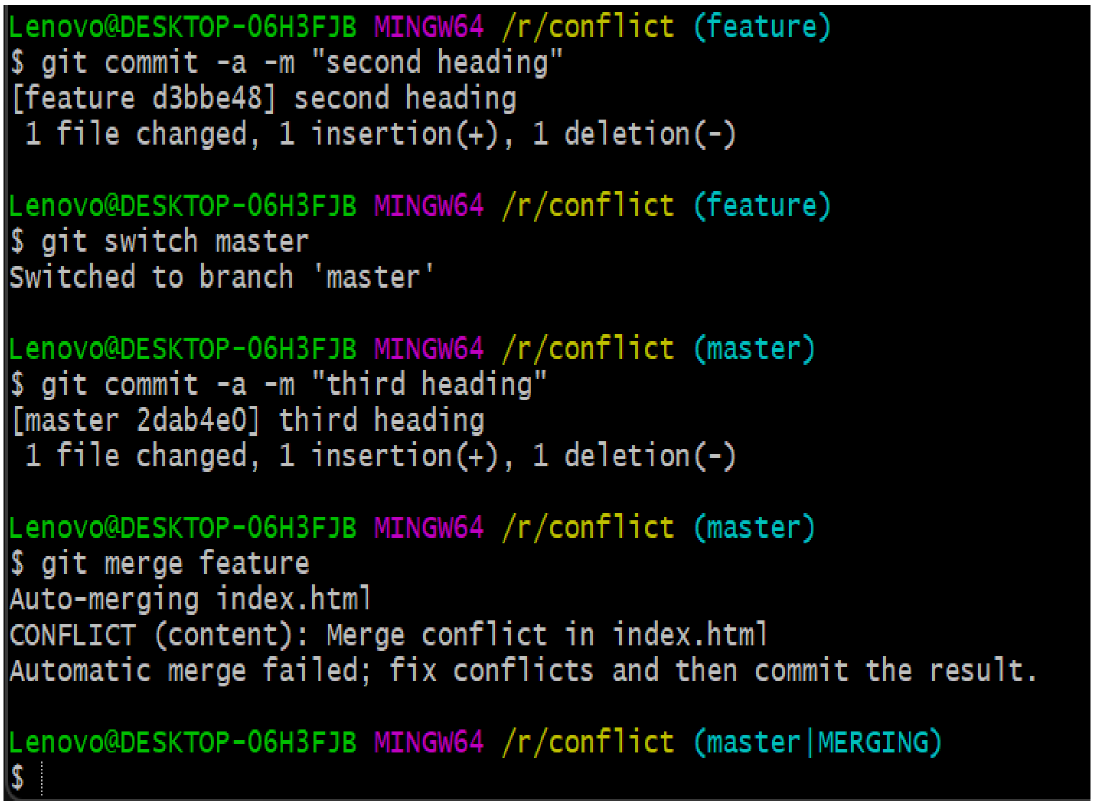
2. After editing the file, we can use the git add a command to stage the new merged content.

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

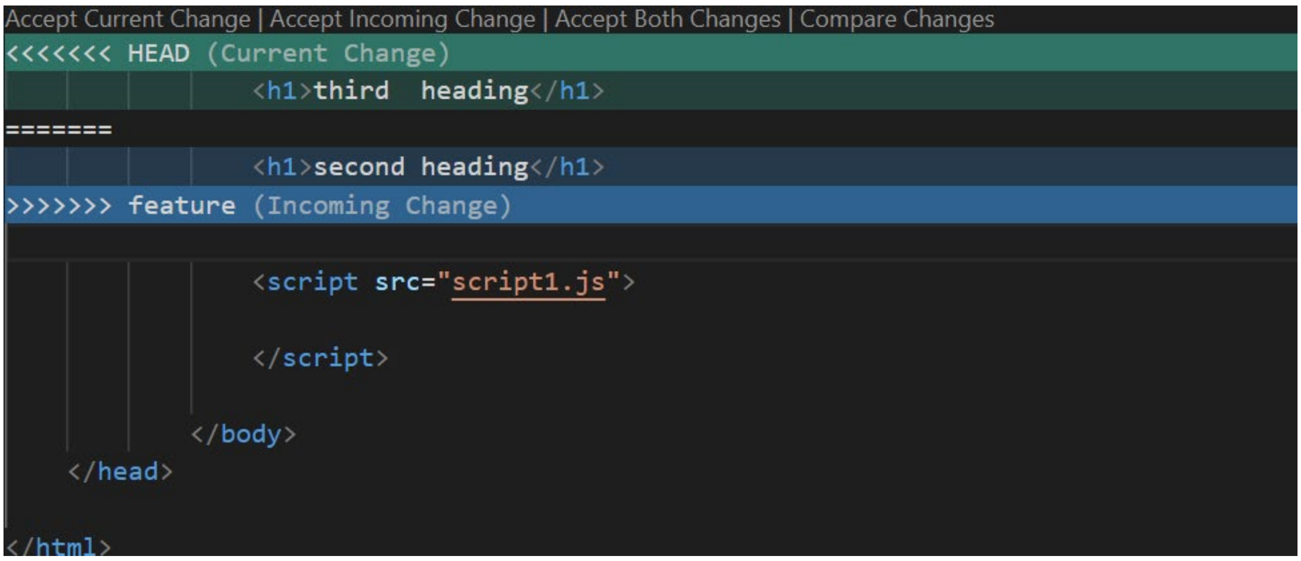
Let us now look into the Git commands that may play a significant role in resolving conflicts.

1. Merge conflict occurs.



```
Lenovo@DESKTOP-06H3FJB MINGW64 /r/conflict (feature)
$ git commit -a -m "second heading"
[feature d3bbe48] second heading
 1 file changed, 1 insertion(+), 1 deletion(-)

Lenovo@DESKTOP-06H3FJB MINGW64 /r/conflict (feature)
$ git switch master
Switched to branch 'master'

Lenovo@DESKTOP-06H3FJB MINGW64 /r/conflict (master)
$ git commit -a -m "third heading"
[master 2dab4e0] third heading
 1 file changed, 1 insertion(+), 1 deletion(-)

Lenovo@DESKTOP-06H3FJB MINGW64 /r/conflict (master)
$ git merge feature
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.

Lenovo@DESKTOP-06H3FJB MINGW64 /r/conflict (master|MERGING)
$
```

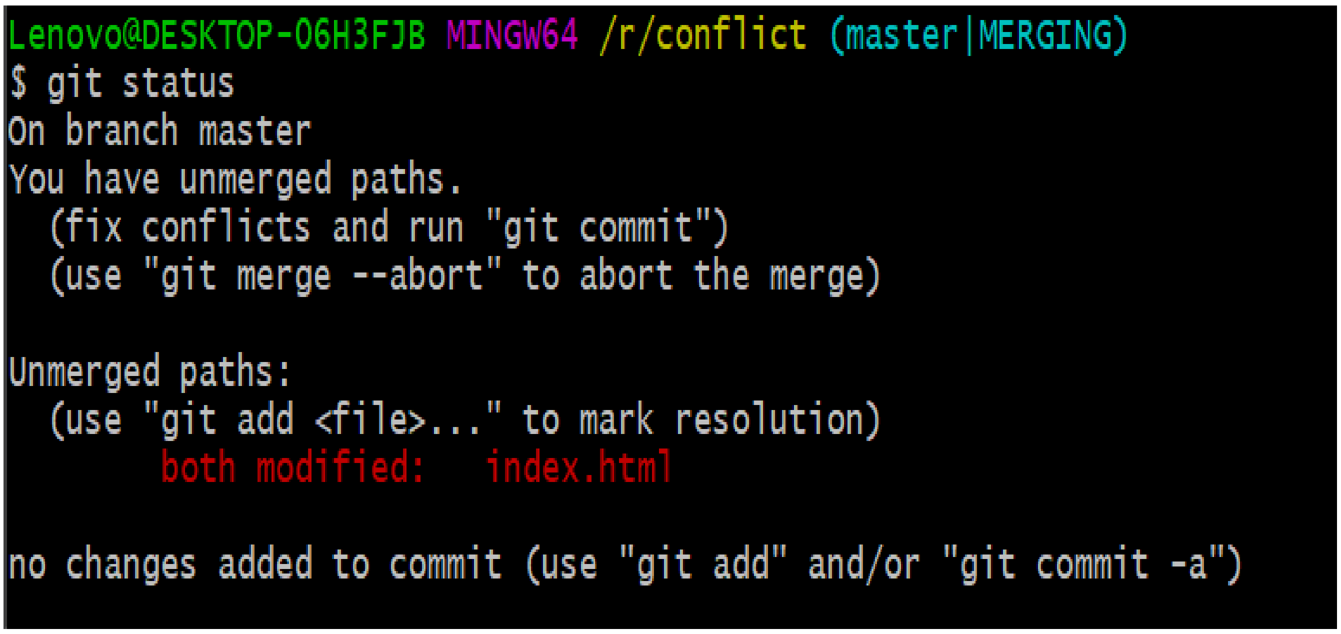
2. Resolving conflict through code editor.



Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes

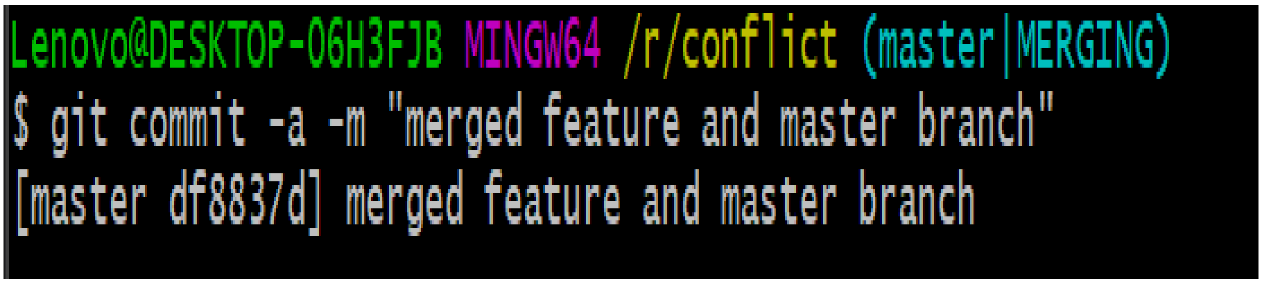
```
<<<<<<< HEAD (Current Change)
                <h1>third  heading</h1>
=======
                <h1>second heading</h1>
>>>>>>> feature (Incoming Change)

                <script src="script1.js">

                </script>

            </body>
    </head>

</html>
```

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



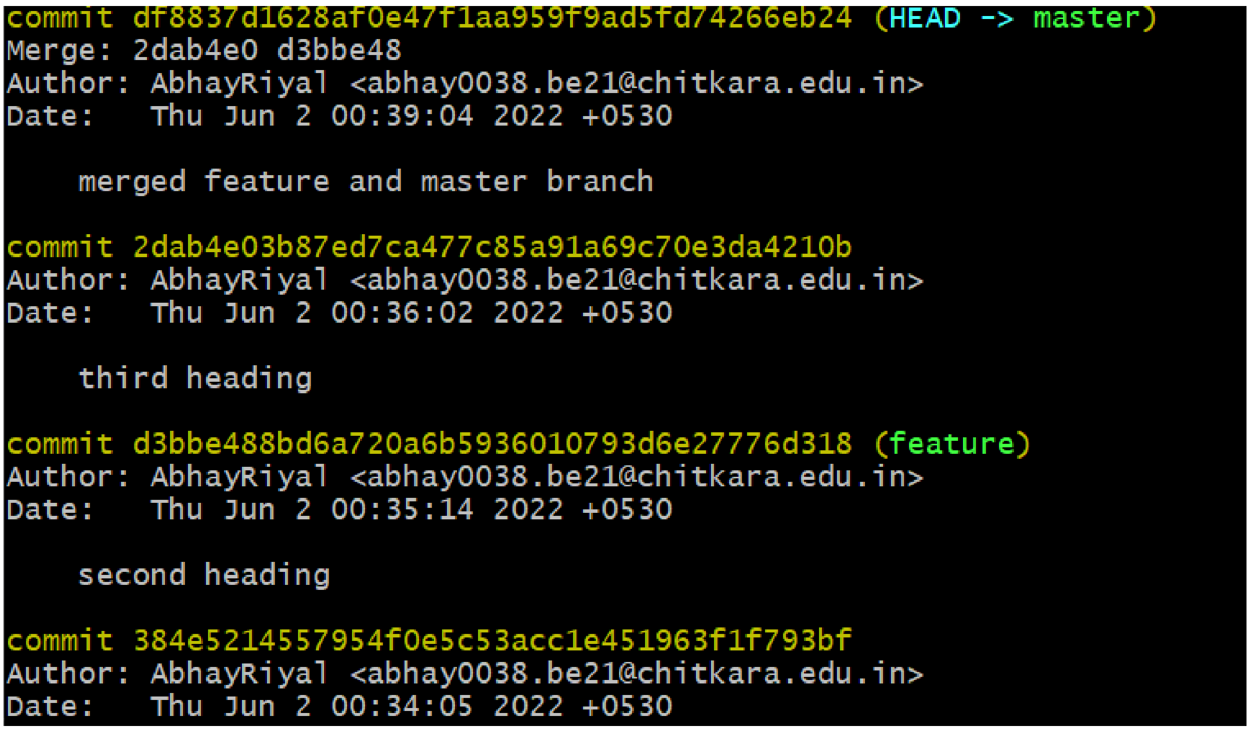
```
Lenovo@DESKTOP-06H3FJB MINGW64 /r/conflict (master|MERGING)
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

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

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

4. Merging of branches happen.



```
Lenovo@DESKTOP-O6H3FJB MINGW64 /r/conflict (master|MERGING)
$ git commit -a -m "merged feature and master branch"
[master df8837d] merged feature and master branch
```

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



```
commit df8837d1628af0e47f1aa959f9ad5fd74266eb24 (HEAD -> master)
Merge: 2dab4e0 d3bbe48
Author: AbhayRiyal <abhay0038.be21@chitkara.edu.in>
Date:   Thu Jun 2 00:39:04 2022 +0530

    merged feature and master branch

commit 2dab4e03b87ed7ca477c85a91a69c70e3da4210b
Author: AbhayRiyal <abhay0038.be21@chitkara.edu.in>
Date:   Thu Jun 2 00:36:02 2022 +0530

    third heading

commit d3bbe488bd6a720a6b5936010793d6e27776d318 (feature)
Author: AbhayRiyal <abhay0038.be21@chitkara.edu.in>
Date:   Thu Jun 2 00:35:14 2022 +0530

    second heading

commit 384e5214557954f0e5c53acc1e451963f1f793bf
Author: AbhayRiyal <abhay0038.be21@chitkara.edu.in>
Date:   Thu Jun 2 00:34:05 2022 +0530
```

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

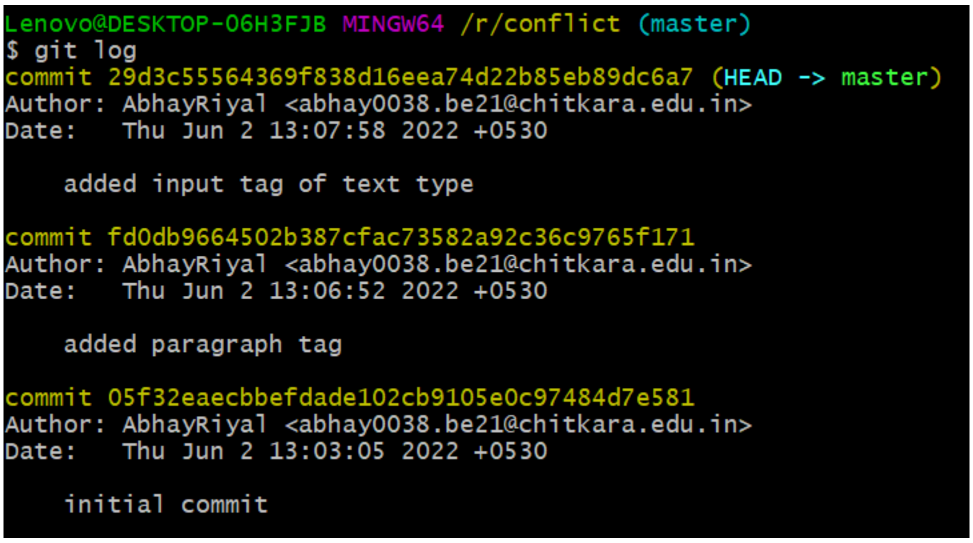
1) Change previous commits, i.e, the commit with the wrong code is replaced, forthis we use “**RESET**”
2) Make a new commit that changes the effect of the previous commit withoutactually removing it from history, for this, we use “**REVERT**”

Git reset

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

To stage a currently staged file we will rum git add and git commit..

git log should show:

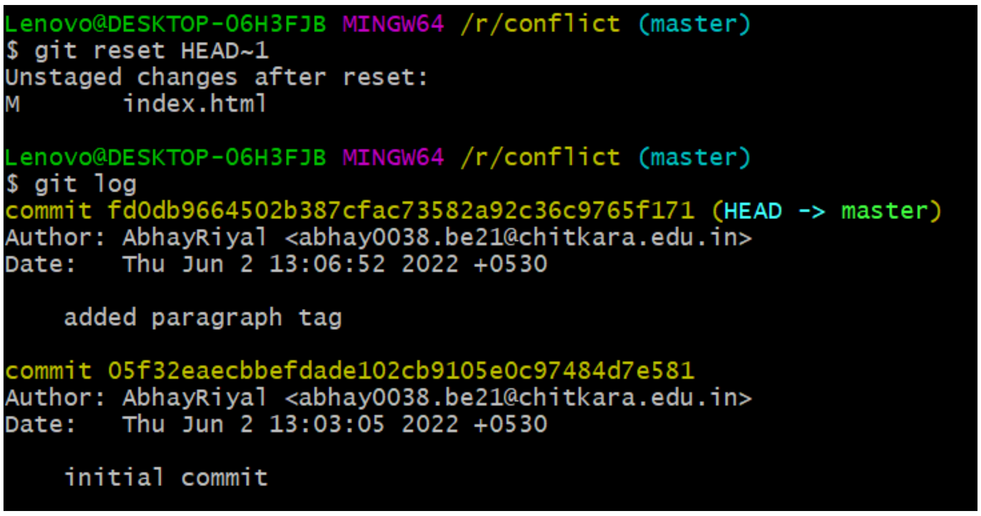


```
Lenovo@DESKTOP-06H3FJB MINGW64 /r/conflict (master)
$ git log
commit 29d3c55564369f838d16eea74d22b85eb89dc6a7 (HEAD -> master)
Author: AbhayRiyal <abhay0038.be21@chitkara.edu.in>
Date:   Thu Jun 2 13:07:58 2022 +0530

    added input tag of text type

commit fd0db9664502b387cfac73582a92c36c9765f171
Author: AbhayRiyal <abhay0038.be21@chitkara.edu.in>
Date:   Thu Jun 2 13:06:52 2022 +0530

    added paragraph tag

commit 05f32eaecbbefdade102cb9105e0c97484d7e581
Author: AbhayRiyal <abhay0038.be21@chitkara.edu.in>
Date:   Thu Jun 2 13:03:05 2022 +0530

    initial commit
```

To unstage, and remove previous commit from git log we will run

Git reset HEAD~1

After which git log will change to:



```
Lenovo@DESKTOP-06H3FJB MINGW64 /r/conflict (master)
$ git reset HEAD~1
Unstaged changes after reset:
M       index.html

Lenovo@DESKTOP-06H3FJB MINGW64 /r/conflict (master)
$ git log
commit fd0db9664502b387cfac73582a92c36c9765f171 (HEAD -> master)
Author: AbhayRiyal <abhay0038.be21@chitkara.edu.in>
Date:   Thu Jun 2 13:06:52 2022 +0530

    added paragraph tag

commit 05f32eaecbbefdade102cb9105e0c97484d7e581
Author: AbhayRiyal <abhay0038.be21@chitkara.edu.in>
Date:   Thu Jun 2 13:03:05 2022 +0530

    initial commit
```

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 torecover them, use with caution!

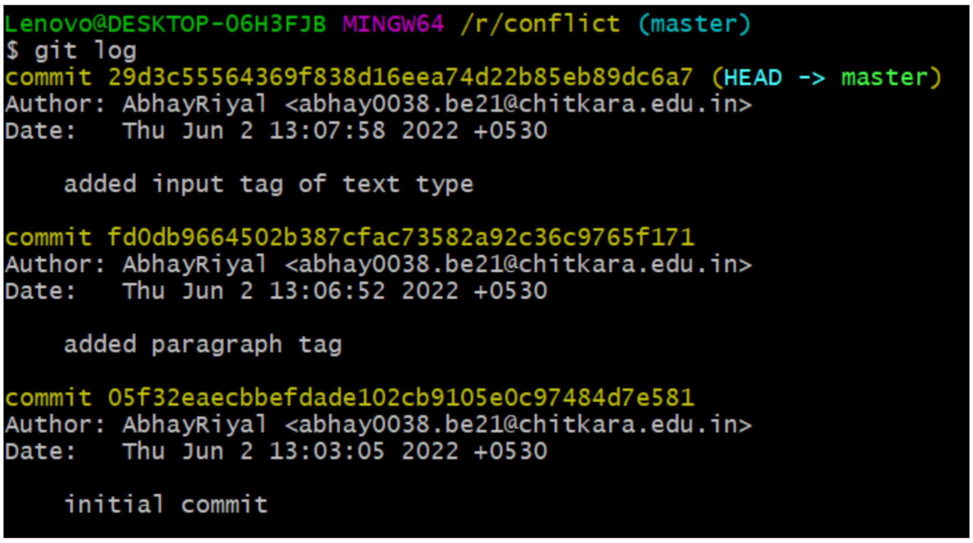
Git reset —mixed HEAD~n: This is the default option which is the same as *git reset HEAD~n* it only removes the commits and unstaged the files but the changes made by those commits still exist in the working directory

Git reset —soft HEAD~n: In this, it only removes the commits but would not 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



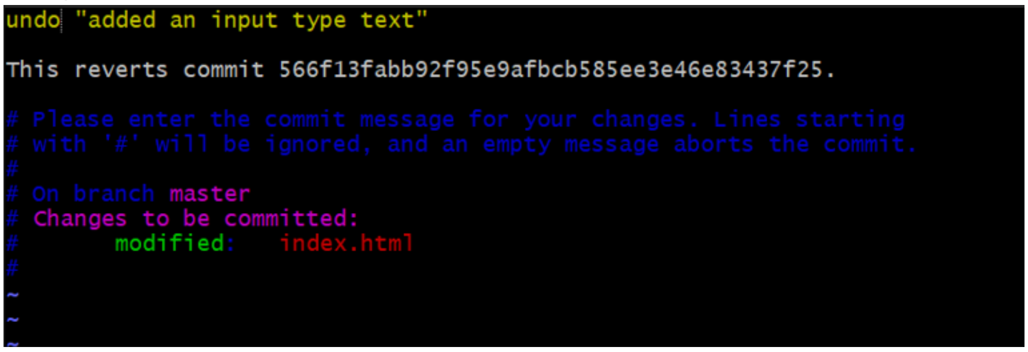
```
Lenovo@DESKTOP-06H3FJB MINGW64 /r/conflict (master)
$ git log
commit 29d3c55564369f838d16eea74d22b85eb89dc6a7 (HEAD -> master)
Author: AbhayRiyal <abhay0038.be21@chitkara.edu.in>
Date:   Thu Jun 2 13:07:58 2022 +0530

    added input tag of text type

commit fd0db9664502b387cfac73582a92c36c9765f171
Author: AbhayRiyal <abhay0038.be21@chitkara.edu.in>
Date:   Thu Jun 2 13:06:52 2022 +0530

    added paragraph tag

commit 05f32eaecbbefdade102cb9105e0c97484d7e581
Author: AbhayRiyal <abhay0038.be21@chitkara.edu.in>
Date:   Thu Jun 2 13:03:05 2022 +0530

    initial commit
```

If we want to remove the topmost commit, then we will copy its commit ID/hash.

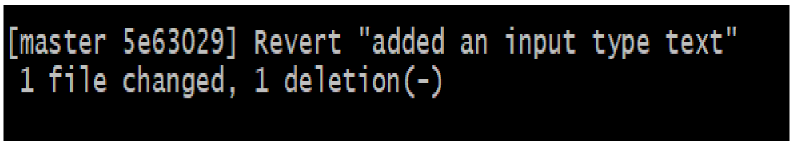
Use git revert as such.

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

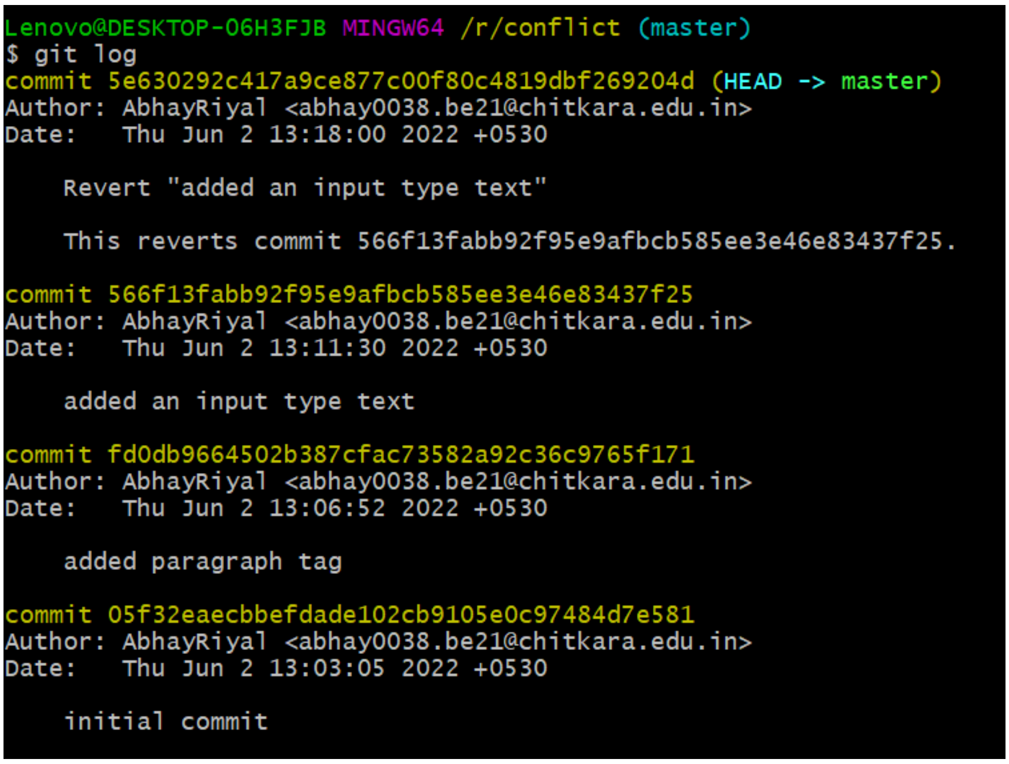


```
undo "added an input type text"

This reverts commit 566f13fabb92f95e9afbcb585ee3e46e83437f25.

# 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:   index.html
#
~
~
~
```



```
[master 5e63029] Revert "added an input type text"
 1 file changed, 1 deletion(-)
```

The git log now should be



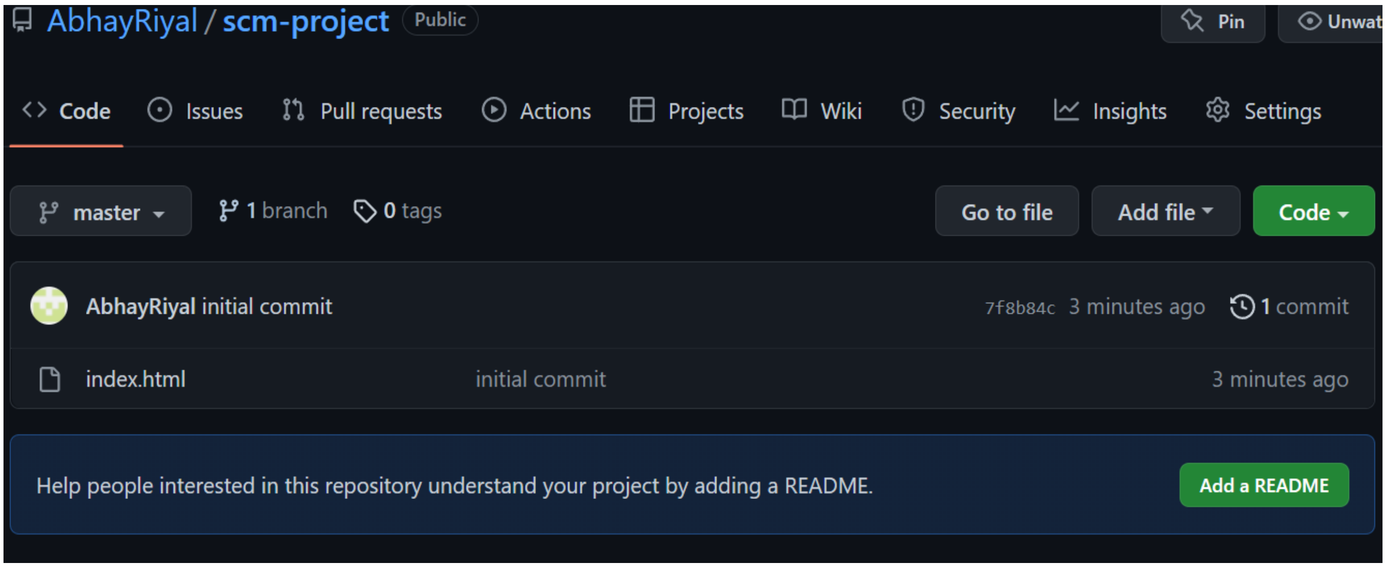
```
Lenovo@DESKTOP-06H3FJB MINGW64 /r/conflict (master)
$ git log
commit 5e630292c417a9ce877c00f80c4819dbf269204d (HEAD -> master)
Author: AbhayRiyal <abhay0038.be21@chitkara.edu.in>
Date:   Thu Jun 2 13:18:00 2022 +0530

    Revert "added an input type text"

    This reverts commit 566f13fabb92f95e9afbcb585ee3e46e83437f25.

commit 566f13fabb92f95e9afbcb585ee3e46e83437f25
Author: AbhayRiyal <abhay0038.be21@chitkara.edu.in>
Date:   Thu Jun 2 13:11:30 2022 +0530

    added an input type text

commit fd0db9664502b387cfac73582a92c36c9765f171
Author: AbhayRiyal <abhay0038.be21@chitkara.edu.in>
Date:   Thu Jun 2 13:06:52 2022 +0530

    added paragraph tag

commit 05f32eaecbbefdade102cb9105e0c97484d7e581
Author: AbhayRiyal <abhay0038.be21@chitkara.edu.in>
Date:   Thu Jun 2 13:03:05 2022 +0530

    initial commit
```

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-

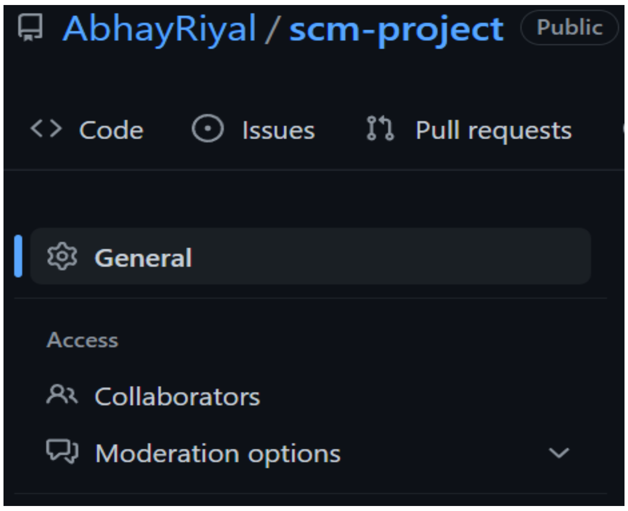
1. Creating a distributed repository and adding people as collaborators.

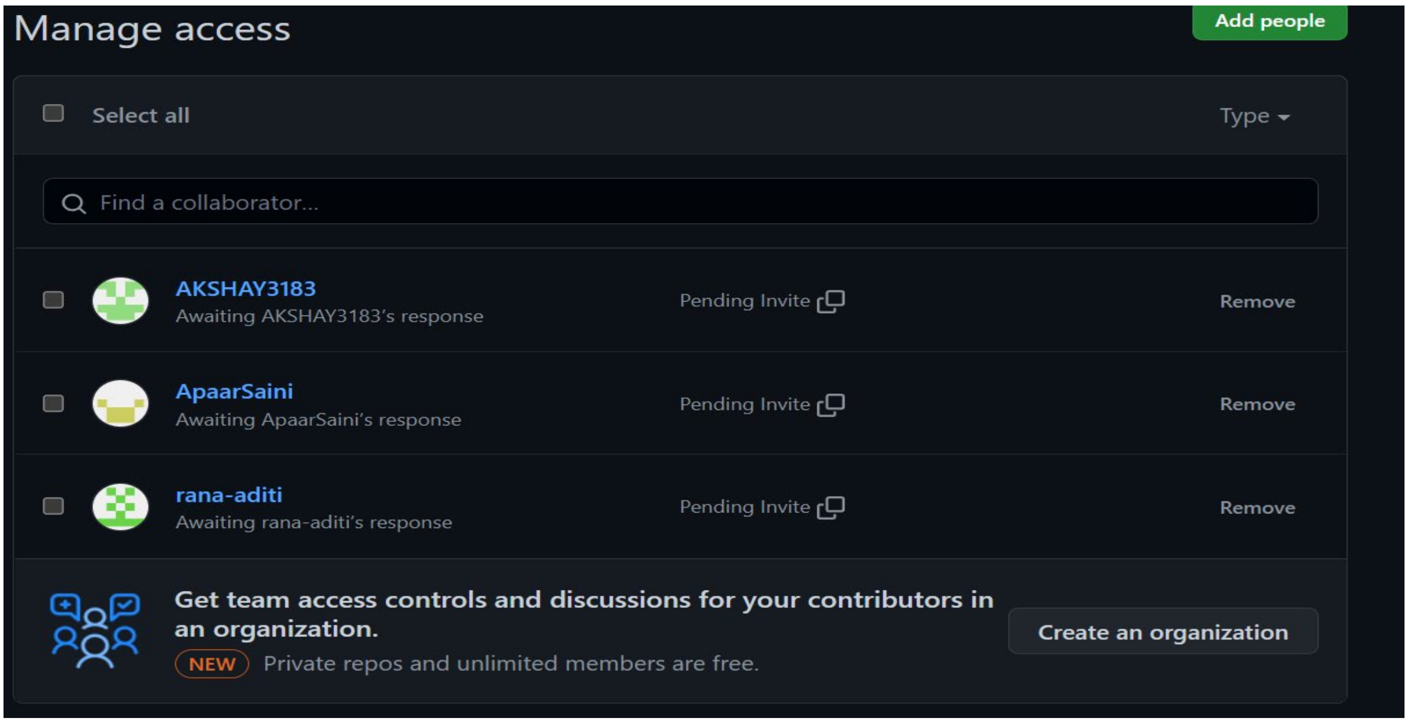


AbhayRiyal initial commit 7f8b84c 3 minutes ago 1 commit

index.html initial commit 3 minutes ago

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




Manage access

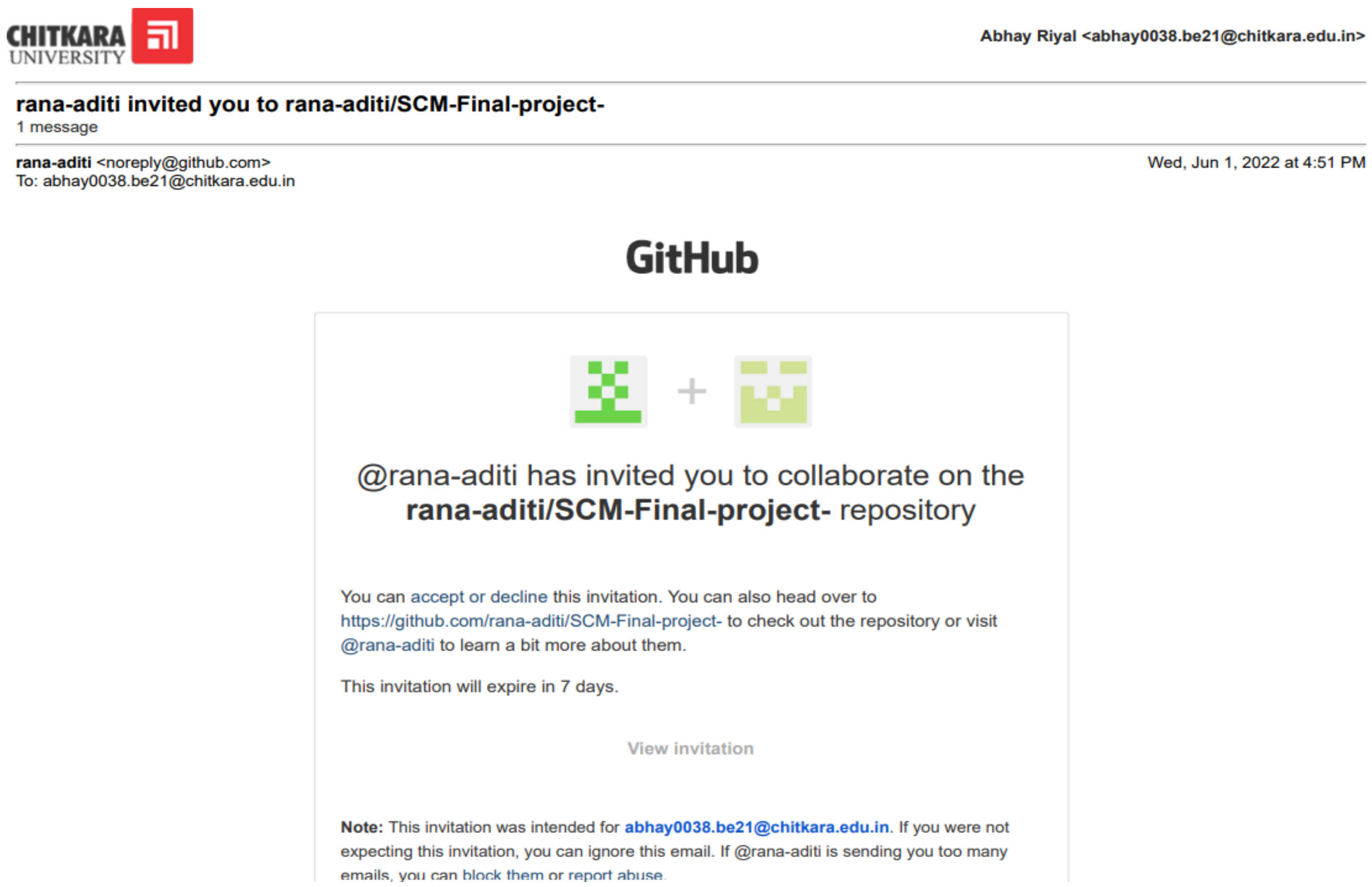
Select all

AKSHAY3183
Awaiting AKSHAY3183'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.


CHITKARA UNIVERSITY

Abhay Riyal <abhay0038.be21@chitkara.edu.in>

rana-aditi invited you to rana-aditi/SCM-Final-project-
1 message

rana-aditi <noreply@github.com>
To: abhay0038.be21@chitkara.edu.in

Wed, Jun 1, 2022 at 4:51 PM

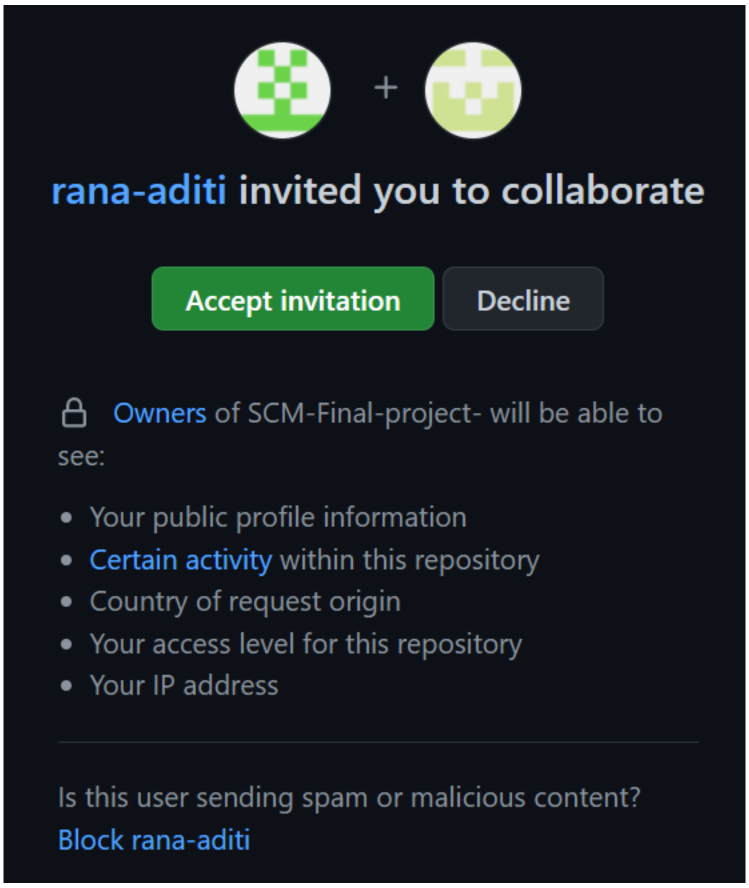
GitHub

@rana-aditi has invited you to collaborate on the **rana-aditi/SCM-Final-project-** repository

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

This invitation will expire in 7 days.

Note: This invitation was intended for **abhay0038.be21@chitkara.edu.in**. If you were not expecting this invitation, you can ignore this email. If @rana-aditi is sending you too many emails, you can block them or report abuse.


rana-aditi invited you to collaborate

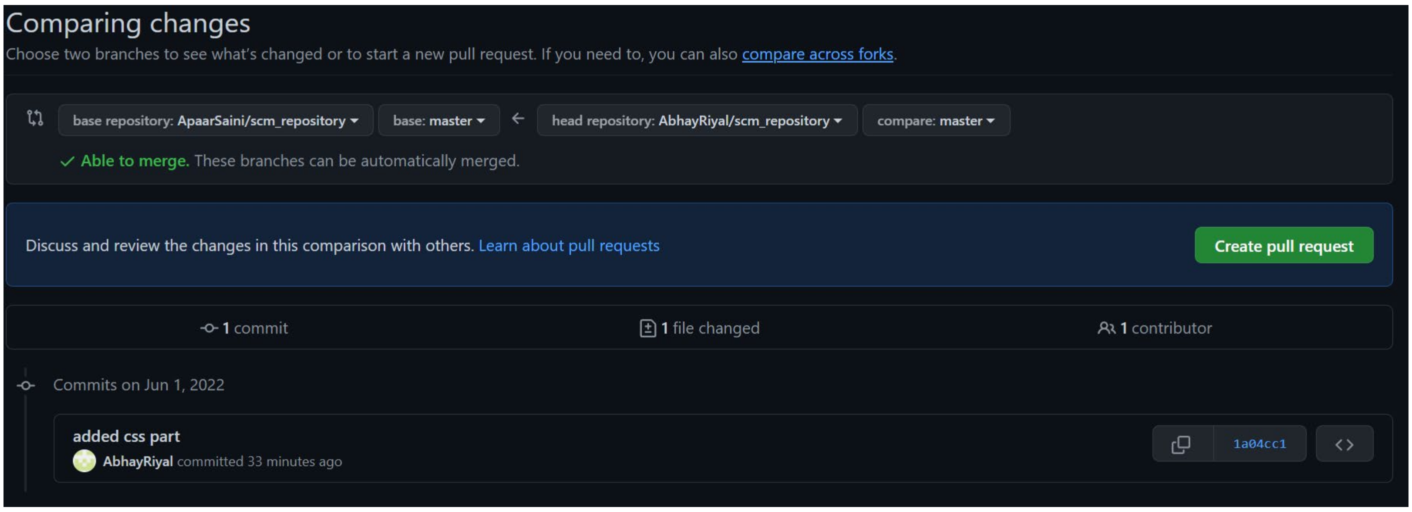
Owners of SCM-Final-project- will be able to see:

- 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

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: ApaarSaini/scm_repository | base: master ← head repository: AbhayRiyal/scm_repository | 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

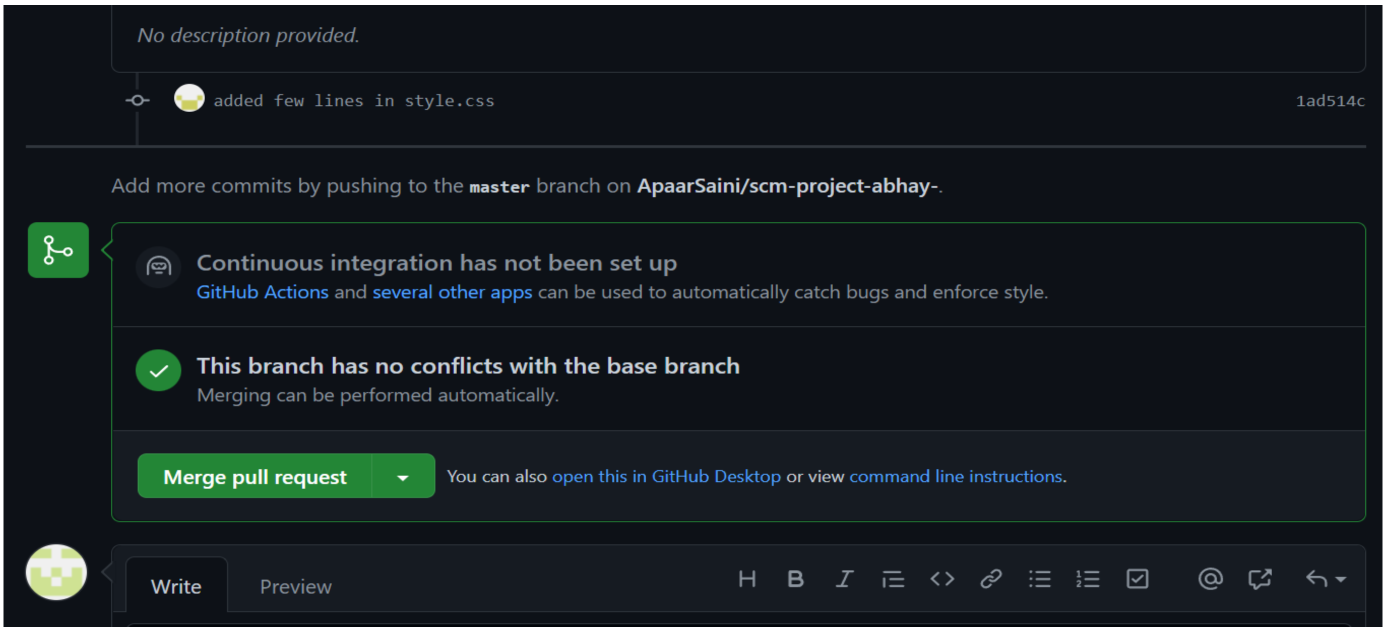
Commits on Jun 1, 2022

added css part

AbhayRiyal committed 33 minutes ago

1a04cc1

3. Closing and merging pull requests as a maintainer.



No description provided.

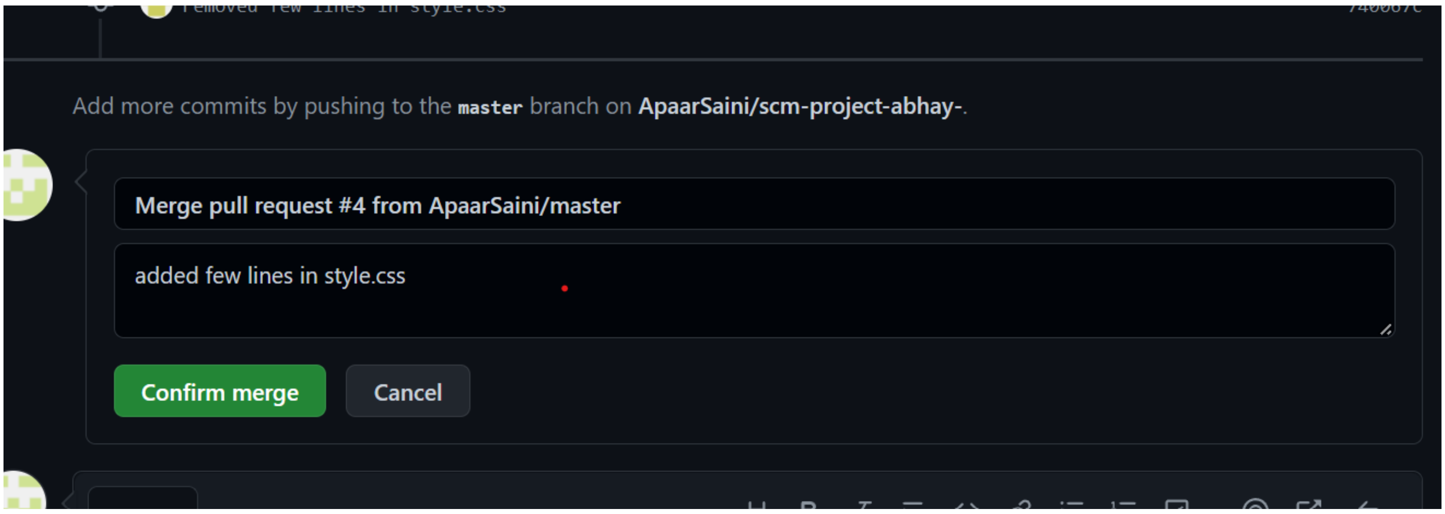
added few lines in style.css 1ad514c

Add more commits by pushing to the **master** branch on **ApaarSaini/scm-project-abhay-**.

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.

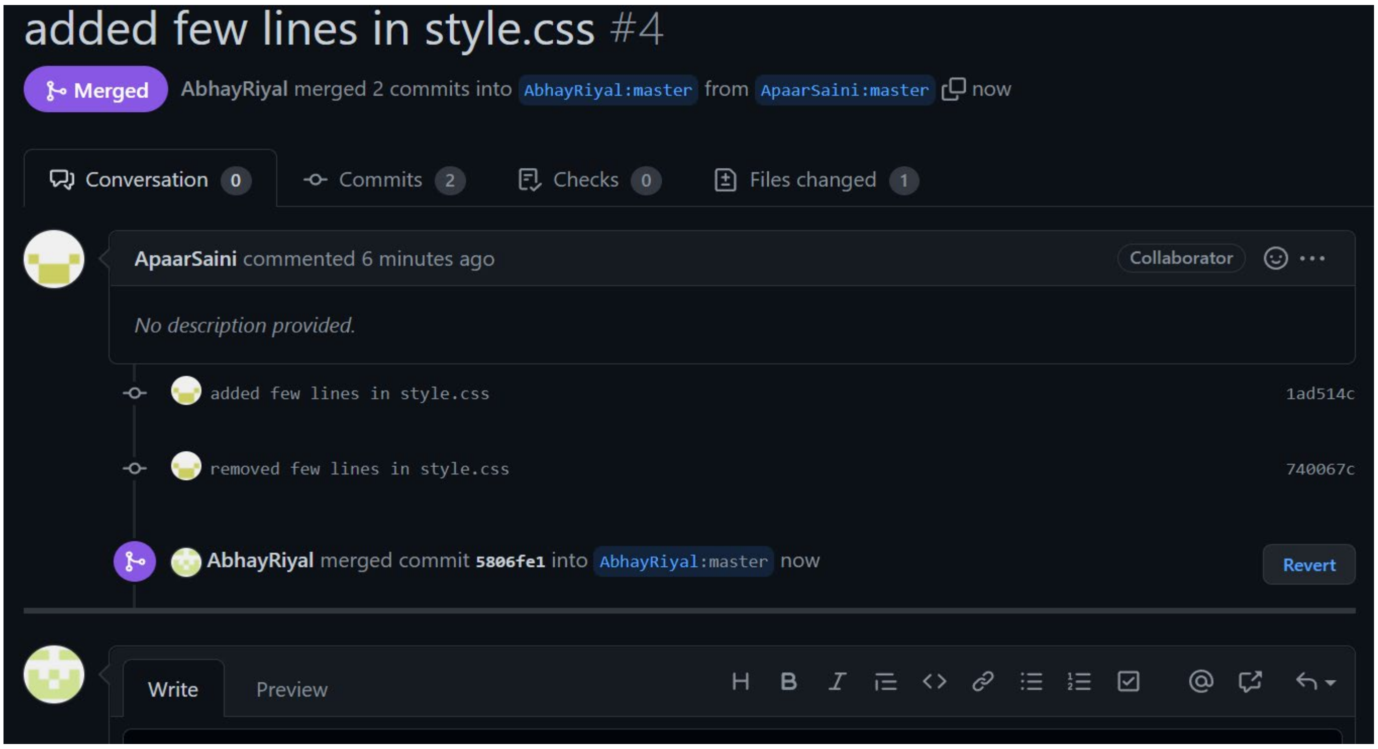
You can also open this in GitHub Desktop or view command line instructions.



Add more commits by pushing to the **master** branch on **ApaarSaini/scm-project-abhay-**.

Merge pull request #4 from ApaarSaini/master

added few lines in style.css



added few lines in style.css #4

Merged AbhayRiyal merged 2 commits into `AbhayRiyal:master` from `ApaarSaini:master` now

Conversation 0 · Commits 2 · Checks 0 · Files changed 1

ApaarSaini commented 6 minutes ago · Collaborator

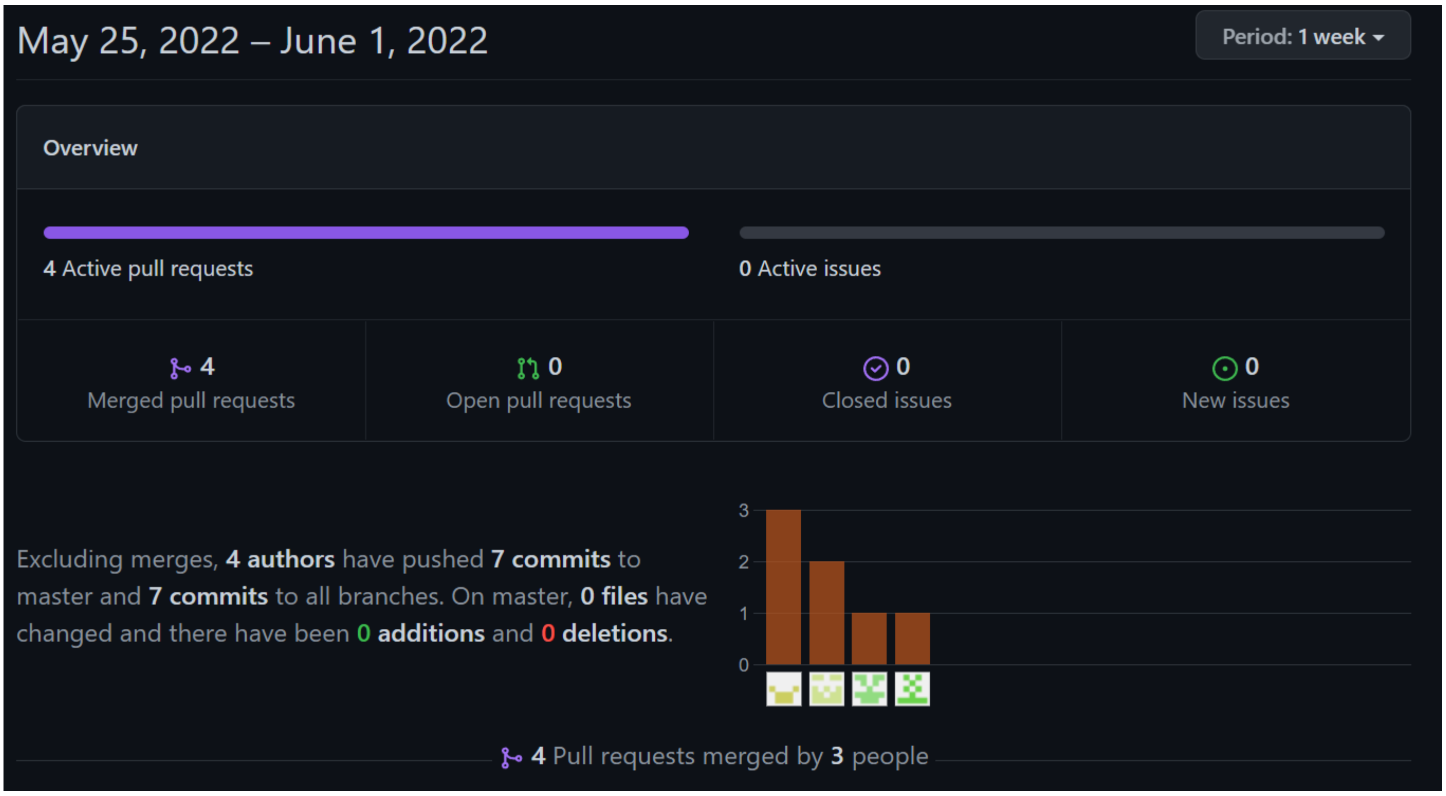
No description provided.

```
added few lines in style.css        1ad514c
removed few lines in style.css      740067c
```

AbhayRiyal merged commit **5806fe1** into `AbhayRiyal:master` now · Revert

Write · Preview

4. Network graph as maintainer.



May 25, 2022 – June 1, 2022 · Period: 1 week

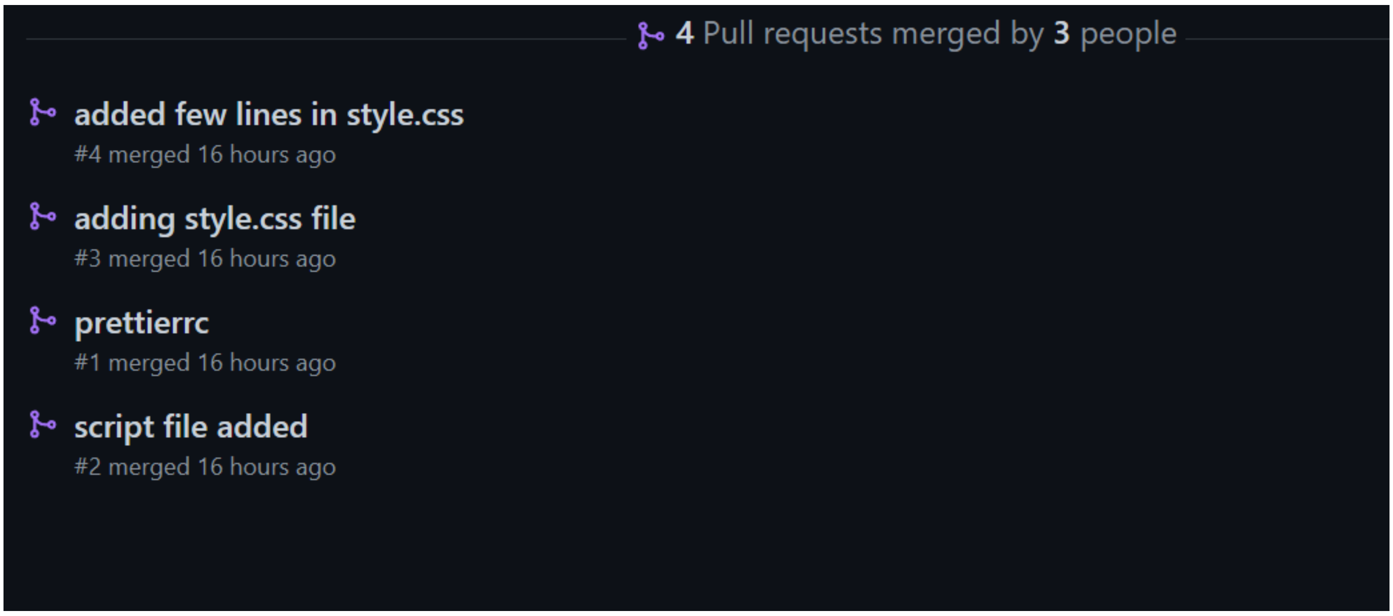
Overview

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

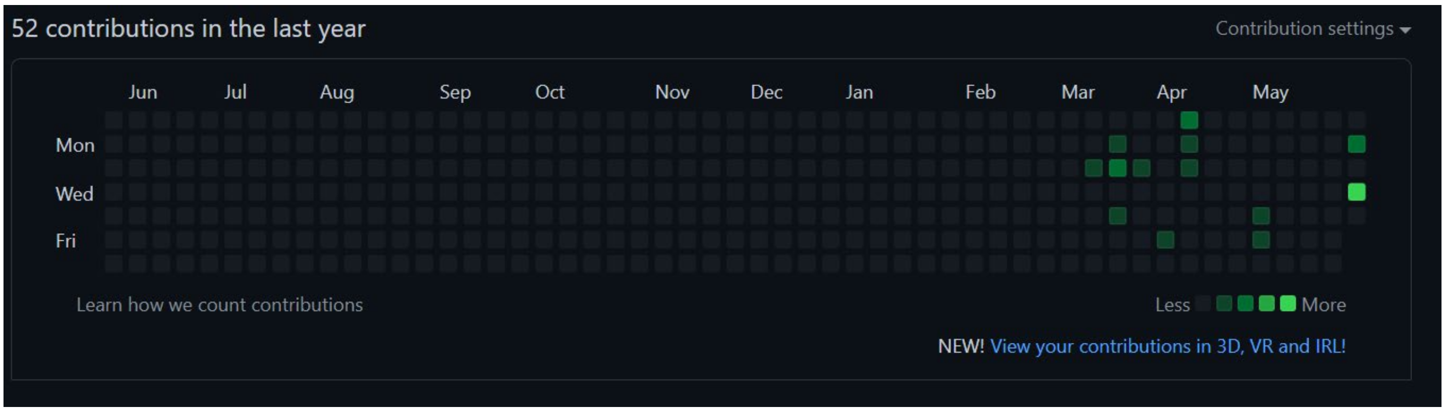
added few lines in style.css
#4 merged 16 hours ago

adding style.css file
#3 merged 16 hours ago

prettierrc
#1 merged 16 hours ago

script file added
#2 merged 16 hours ago

5. 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!