

Subject Name: **Source Code Management**

Subject Code: **CS181**

Cluster: **Beta**

Department: **DCSE**

Submitted By:

Aftab Alam
2110990103

Submitted To:

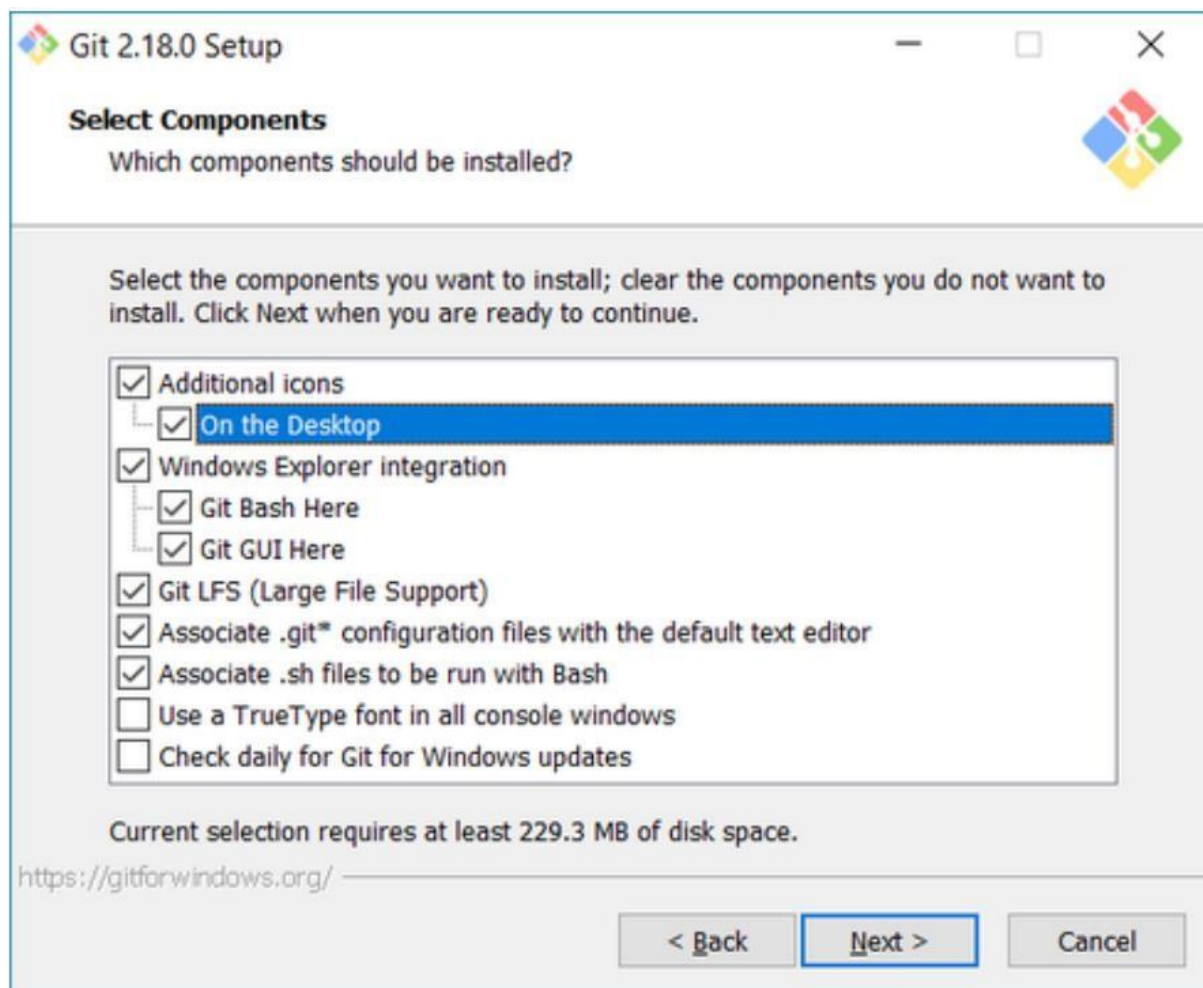
Dr. Monit Kapoor

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).

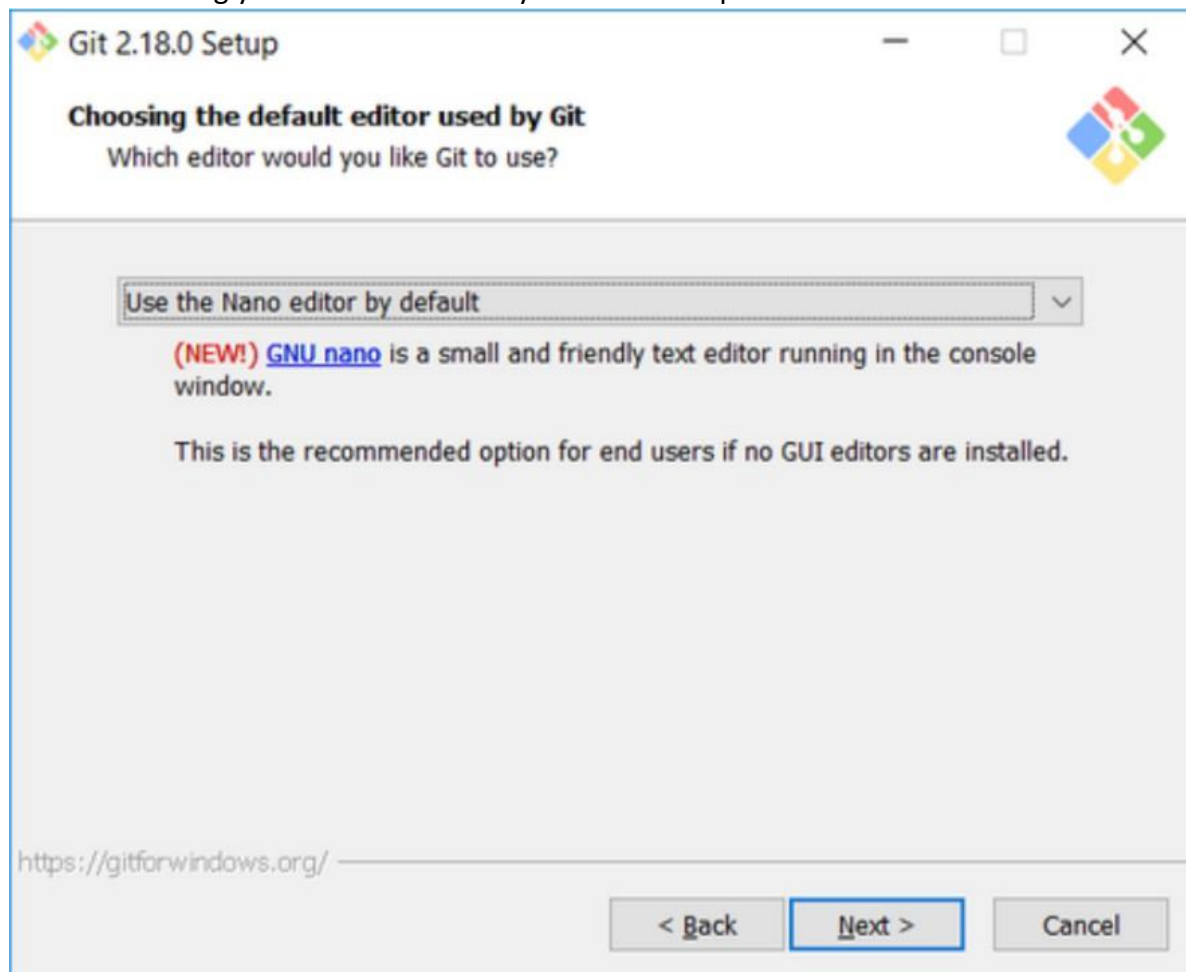
When running the installer, various screens appear (Windows screens shown). Generally, you can accept the default selections, except in the screens below where you do not want the default selections:

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



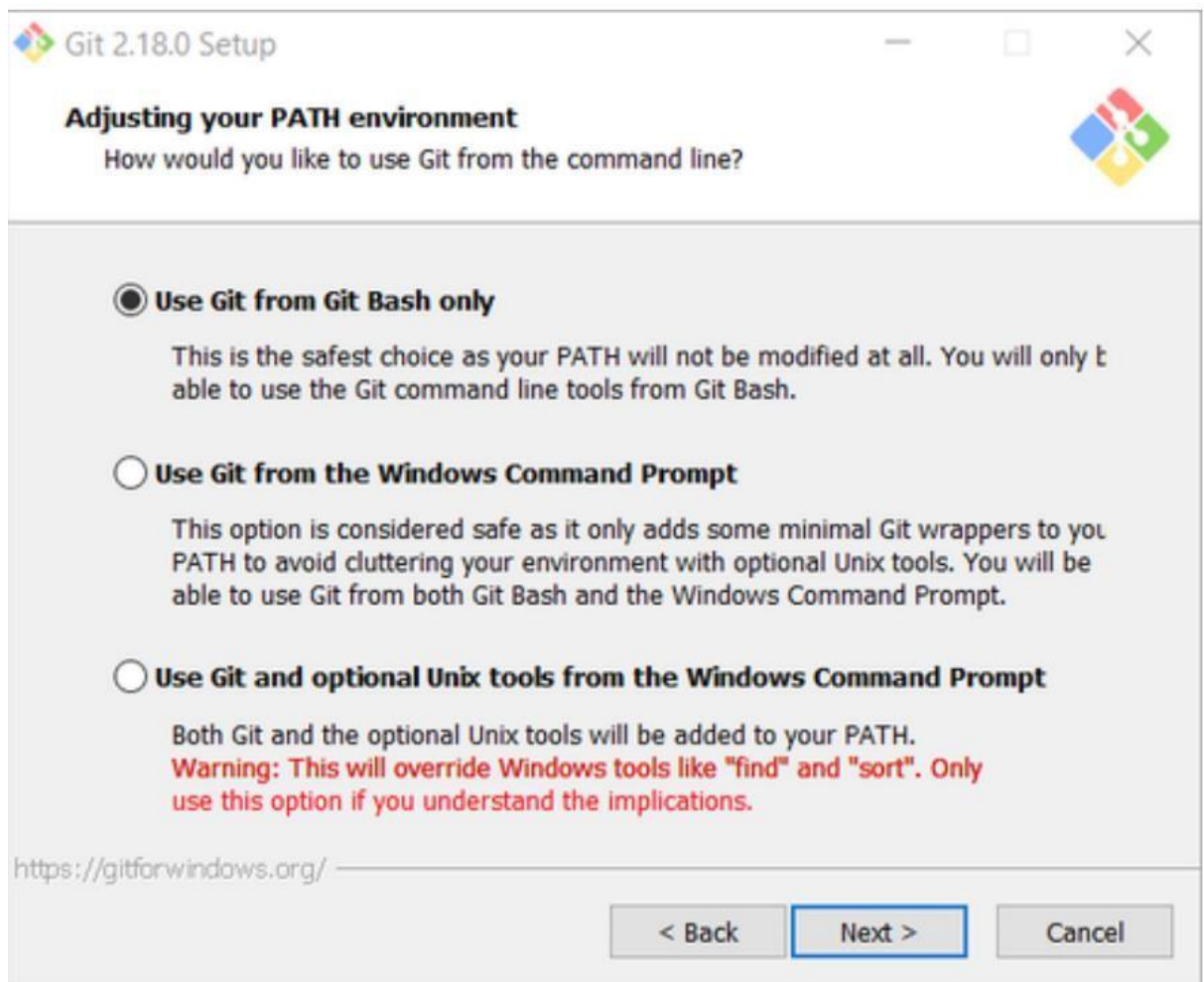
In the choosing the default editor is used by Git dialog, it is strongly recommended that you DO NOT select default VIM editor- it is challenging to learn how to use it, and there are better

modern editors available. Instead, choose Notepad++ or Nano – either of those is much easier to use. It is strongly recommended that you select Notepad++.

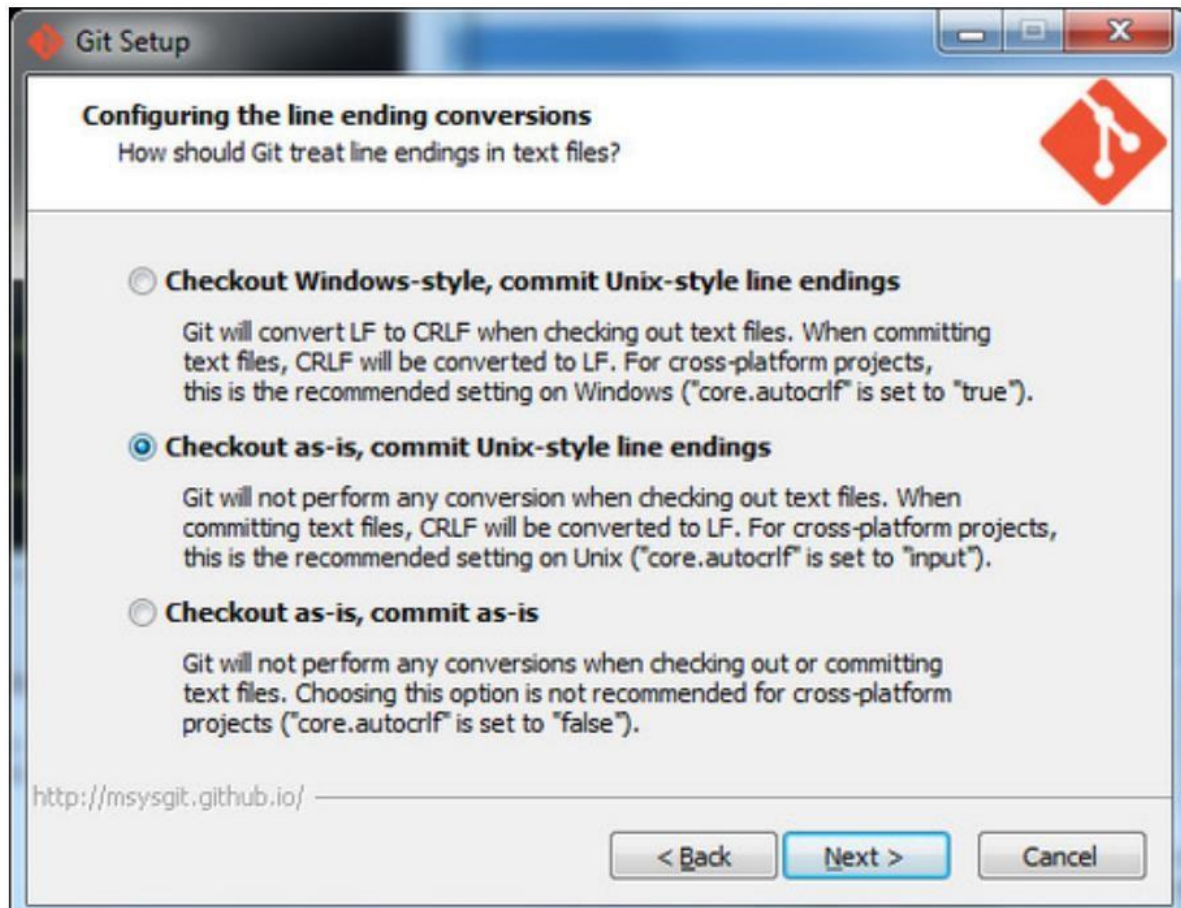


In the Adjusting your PATH screen, all three options are acceptable:

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 "name@msoe.edu"`

This links your Git activity to your email address. Without this, your commits will often show up as "unknown login". Replace name with your own MSOE email name.

d Enter the command `git config --global user.name "Your Name"`

Git 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

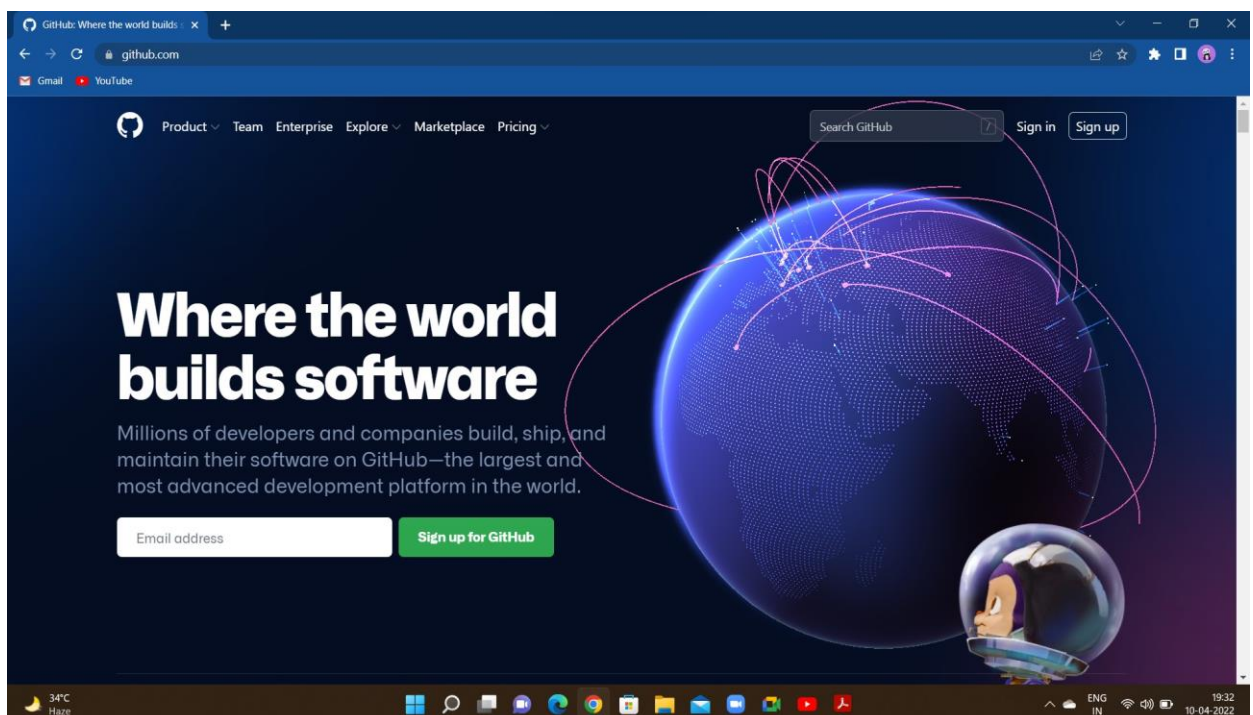
Setting up GitHub Account

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.

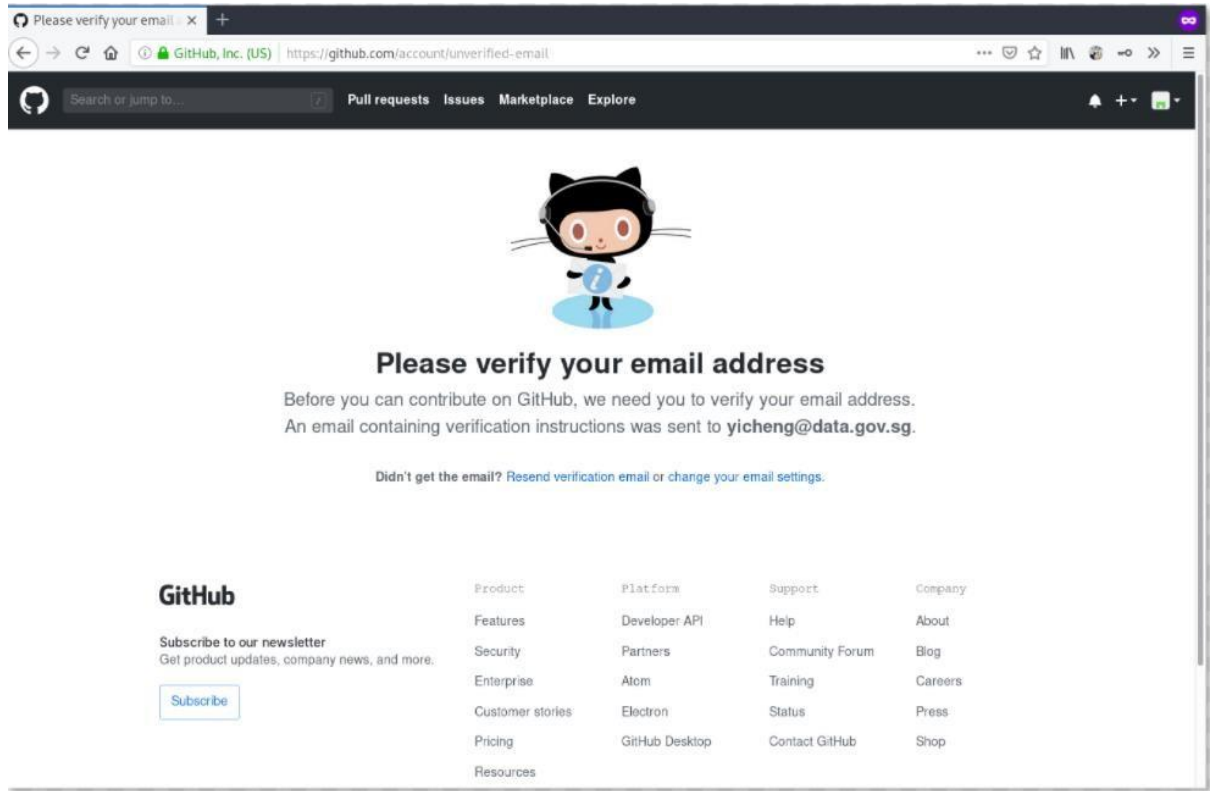
To keep your GitHub account secure you should use a strong and unique password. For more information, see "[Creating a strong password](#)".



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.

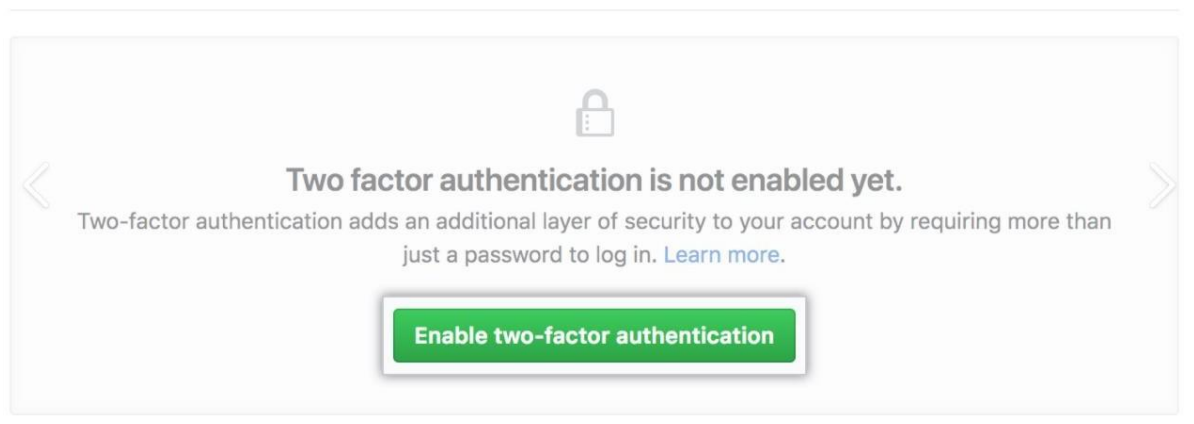
For more information on all GitHub's plans, see "[GitHub's products](#)".

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](#)".

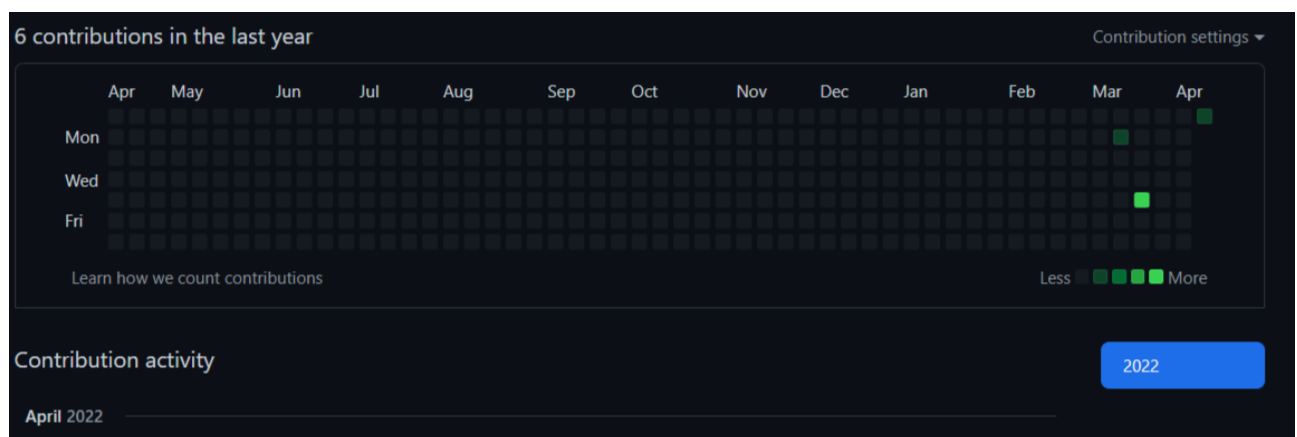


4. **Configuring two-factor authentication:** Two-factor authentication, or 2FA, is an extra layer of security used when logging into websites or apps. We strongly urge you to configure 2FA for safety of your account. For more information, see "[About two-factor authentication](#)."

Two-factor authentication



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



Aim: Program to generate logs

Basic Git init

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

Basic Git status

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

Basic Git commit

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

Basic Git add command

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

Basic Git log

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

```
MINGW64/c/Users/LENOVO/OneDrive/Desktop/SCM
LENOVO@LAPTOP-BOKSEHRI MINGW64 ~/OneDrive/Desktop/SCM (master)
$ git init
Initialized empty Git repository in C:/Users/LENOVO/OneDrive/Desktop/SCM/.git/
LENOVO@LAPTOP-BOKSEHRI MINGW64 ~/OneDrive/Desktop/SCM (master)
$ git status
On branch master
No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)
LENOVO@LAPTOP-BOKSEHRI MINGW64 ~/OneDrive/Desktop/SCM (master)
$ git add .
LENOVO@LAPTOP-BOKSEHRI MINGW64 ~/OneDrive/Desktop/SCM (master)
$ git status
On branch master
No commits yet

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

LENOVO@LAPTOP-BOKSEHRI MINGW64 ~/OneDrive/Desktop/SCM (master)
$ git commit -m "I am changing FUNCTION_OVERLOADING"
[master (root-commit) d52f33d] I am changing FUNCTION_OVERLOADING
1 file changed, 29 insertions(+)
create mode 100644 FUNCTION_OVERLOADING.cpp
LENOVO@LAPTOP-BOKSEHRI MINGW64 ~/OneDrive/Desktop/SCM (master)
$ git log
commit d52f33d2af4836651e11c1d685d26648766d025d (HEAD -> master)
Author: Aftab Alam <aftab0103.be21@chitkara.edu.in>
Date:   Sun Apr 10 20:50:57 2022 +0530

    I am changing FUNCTION_OVERLOADING
LENOVO@LAPTOP-BOKSEHRI MINGW64 ~/OneDrive/Desktop/SCM (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~/Users/LENOVO/OneDrive/Desktop/SCM
LENOVO@LAPTOP-BOKSEHRI MINGW64 ~/OneDrive/Desktop/SCM (master)
$ git branch
* master
LENOVO@LAPTOP-BOKSEHRI MINGW64 ~/OneDrive/Desktop/SCM (master)
$ git branch activity
LENOVO@LAPTOP-BOKSEHRI MINGW64 ~/OneDrive/Desktop/SCM (master)
$ git checkout activity
Switched to branch 'activity'
M   FUNCTION_OVERLOADING.cpp
LENOVO@LAPTOP-BOKSEHRI MINGW64 ~/OneDrive/Desktop/SCM (activity)
$ git status
On branch activity
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:   FUNCTION_OVERLOADING.cpp
no changes added to commit (use "git add" and/or "git commit -a")
LENOVO@LAPTOP-BOKSEHRI MINGW64 ~/OneDrive/Desktop/SCM (activity)
$ git add --a
LENOVO@LAPTOP-BOKSEHRI MINGW64 ~/OneDrive/Desktop/SCM (activity)
$ git branch
* activity
  master
LENOVO@LAPTOP-BOKSEHRI MINGW64 ~/OneDrive/Desktop/SCM (activity)
$ git commit -m "I am in activity"
[activity 9c52877] I am in activity
1 file changed, 3 insertions(+), 3 deletions(-)
LENOVO@LAPTOP-BOKSEHRI MINGW64 ~/OneDrive/Desktop/SCM (activity)
$ git log
commit 9c52877db93faea52b9a6ffdc1ccc6f11397cac (HEAD -> activity)
Author: Aftab Alam <aftab0103.be21@chitkara.edu.in>
Date:   Sun Apr 10 20:55:09 2022 +0530

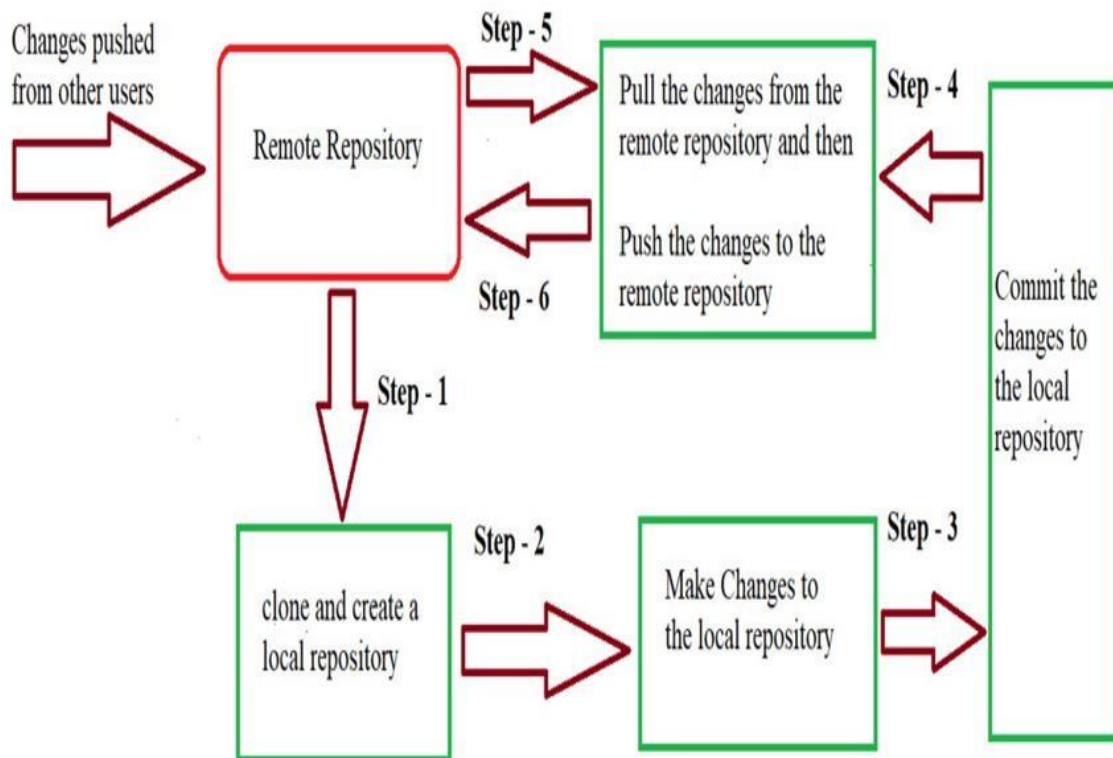
    I am in activity

commit d52f33d2af4836651e1c1d685d26648766d025d (master)
Author: Aftab Alam <aftab0103.be21@chitkara.edu.in>
Date:   Sun Apr 10 20:50:57 2022 +0530

    I am changing FUNCTION_OVERLOADING
LENOVO@LAPTOP-BOKSEHRI MINGW64 ~/OneDrive/Desktop/SCM (activity)
$
```

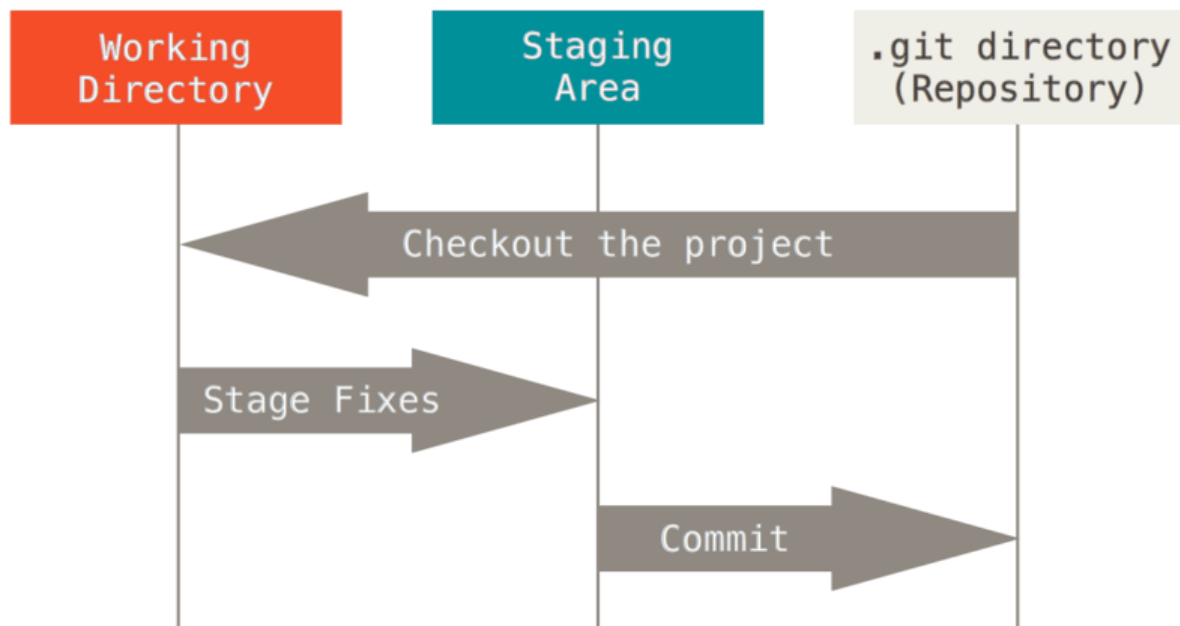
Aim: Git lifecycle description

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-



[This Photo](#) by Unknown Author is licensed under [CC BY-SA](#)

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

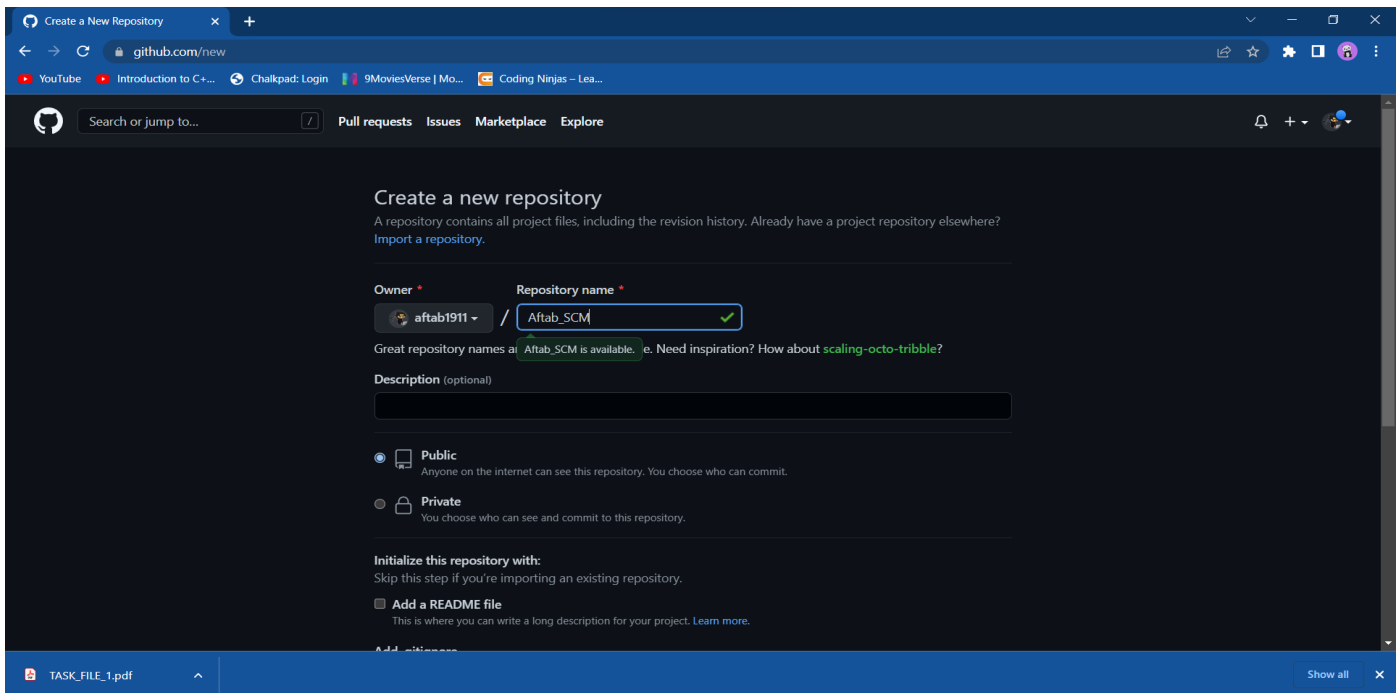
Now since we have all the files that are to be tracked and are ready in the staging area, we are ready to commit our files using the `git commit` command. Commit helps us in keeping the track of the metadata of the files in our staging area. We specify every commit with a message which tells what the 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>
```

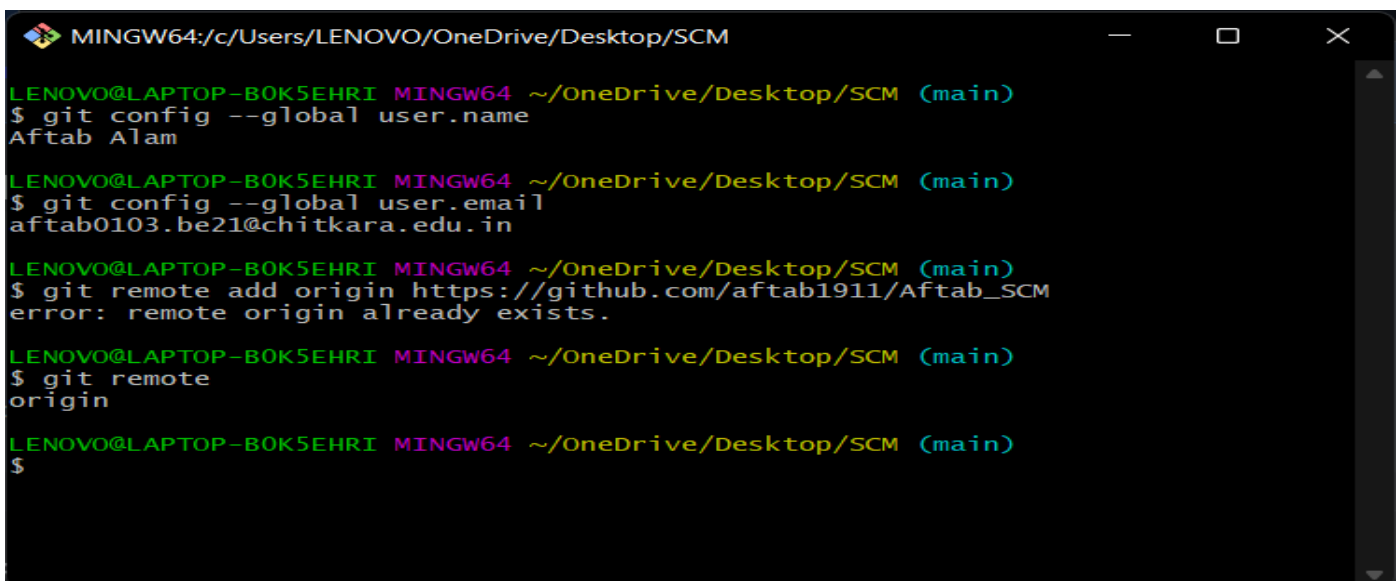
Experiment 6

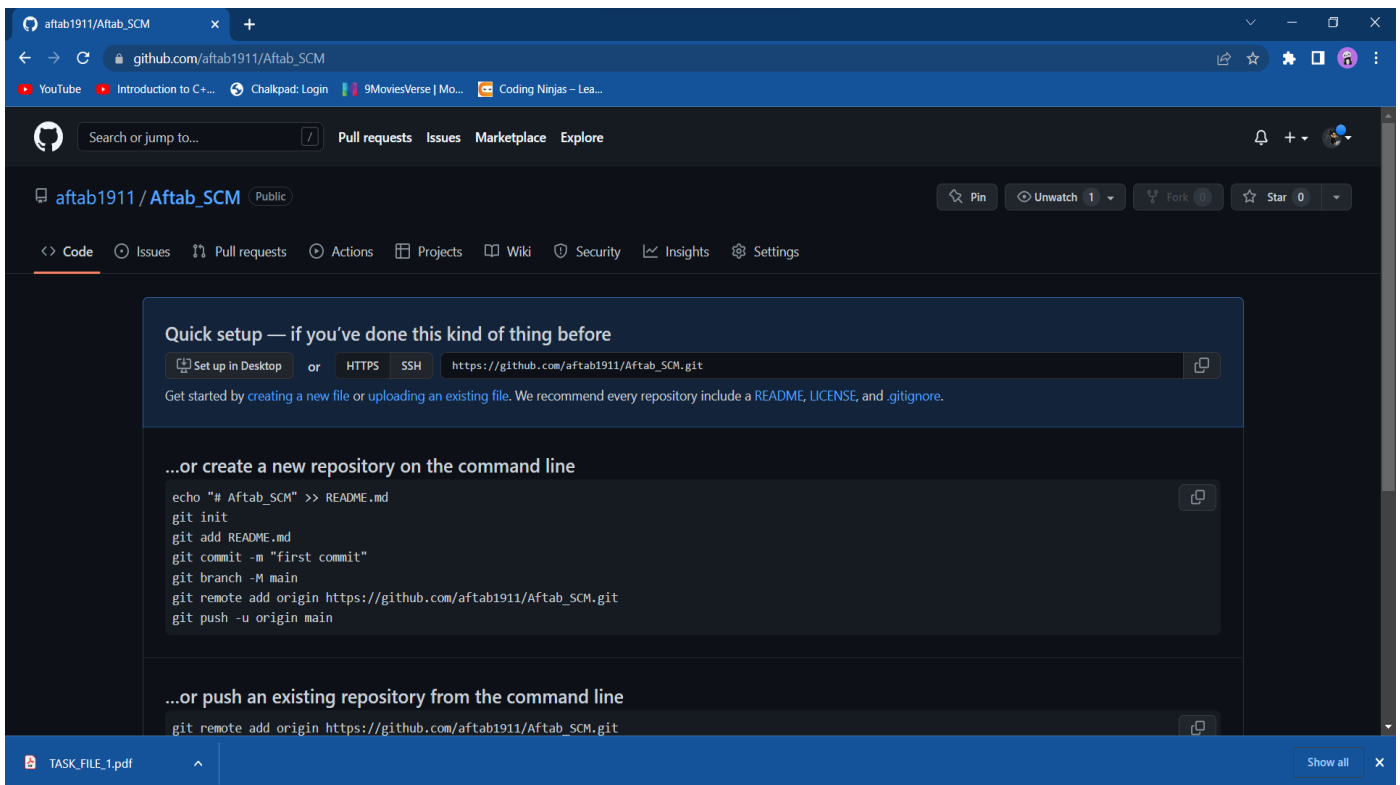
AIM → Add collaborators on GitHub Repo

1. Create a New Repository.

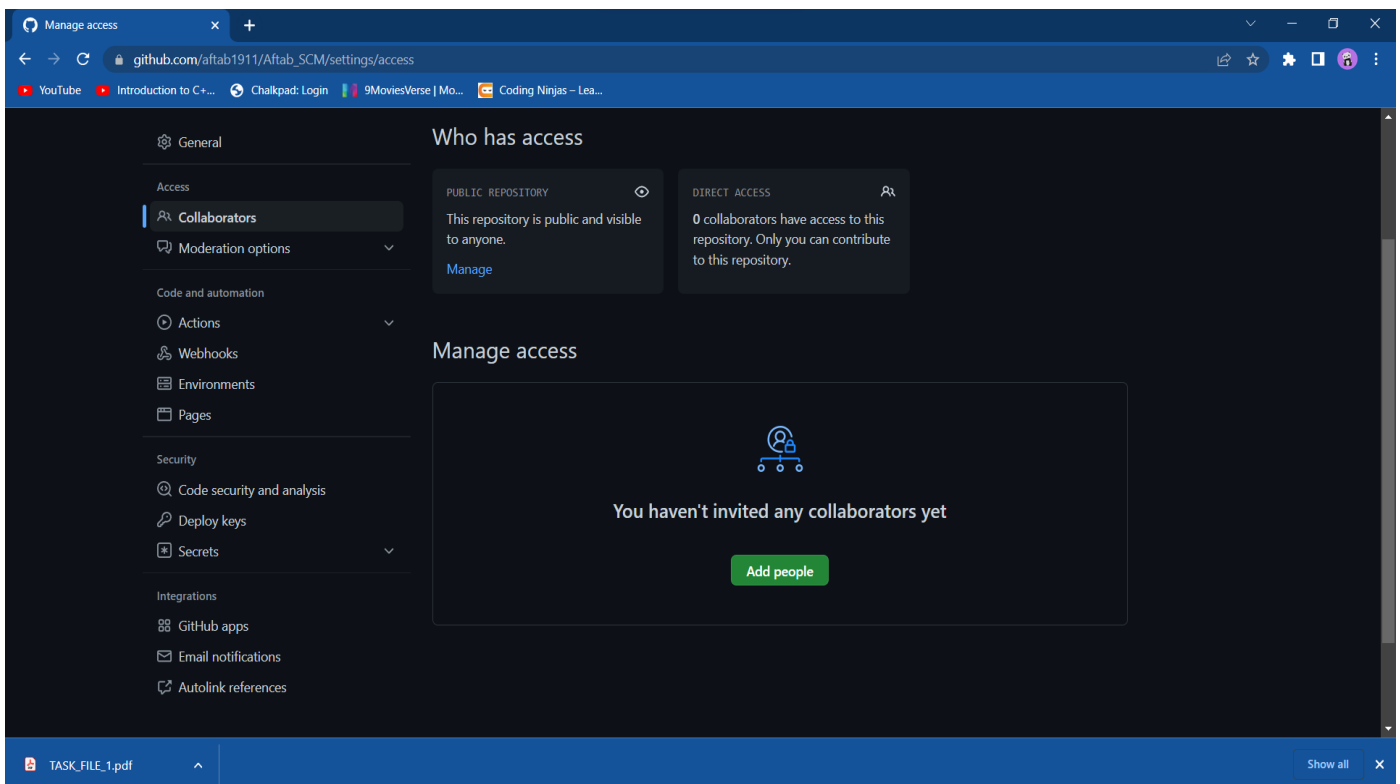


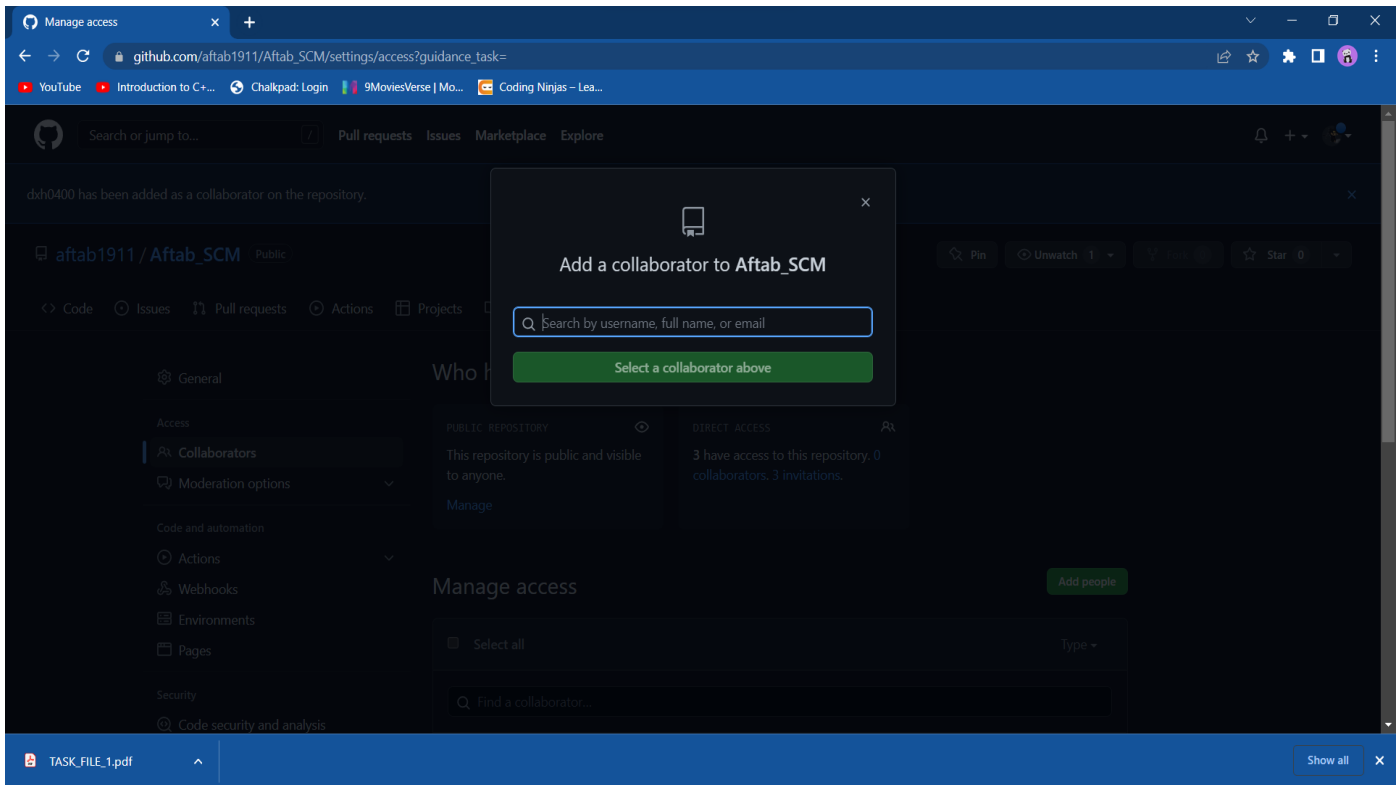
2. Now Copy the HTTP link of repo and paste it on your 'Git BASH', and merge the local repo in remote repo (i.e.) experiment 1.2.



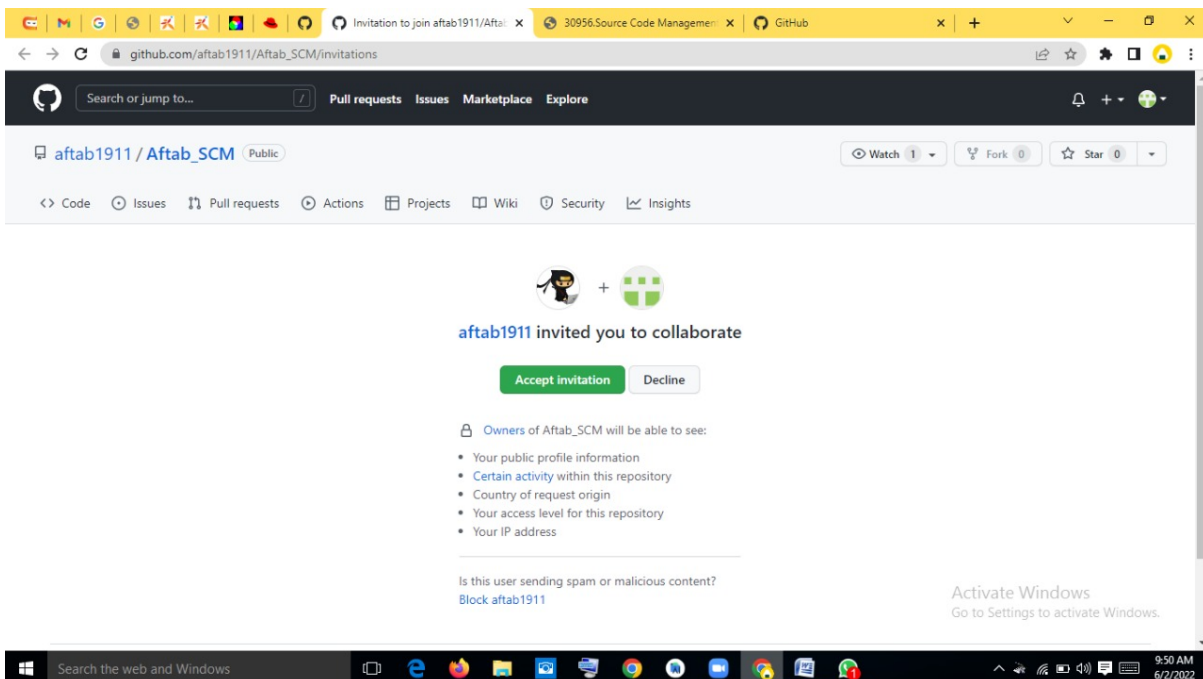


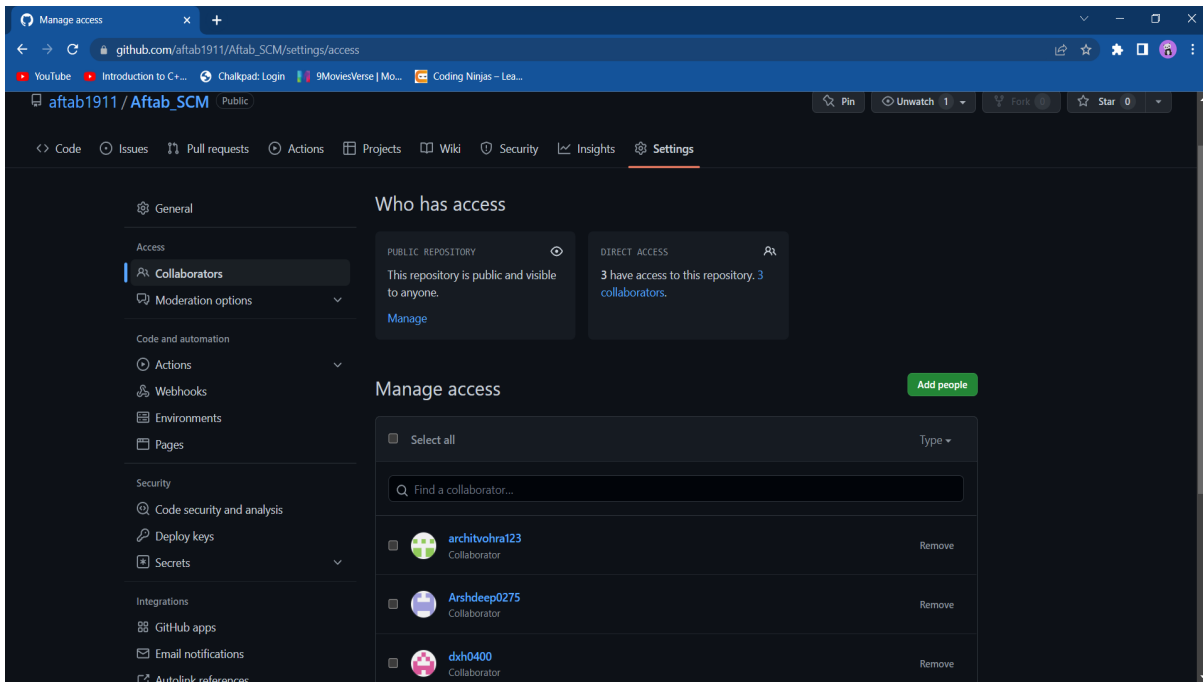
3. Go to Collaborators in Repo Setting, Add the username or email of Collaborator you want to add in your Repo.





4. Invitation Mail is sent to the Collaborator; the collaborator has to accept this Invitation.





5. New Collaborator has now access to the experiment (1.2)
6. To accept access to the Owner's repo, the Collaborator needs to go to <https://github.com/notifications> or check for email notification.
7. Once there she can accept access to the Owner's repo.
8. Next, the Collaborator needs to download a copy of the Owner's repository to her machine. This is called "cloning a repo".
9. The Collaborator can now make a change in her clone of the Owner's repository.
10. Note that we didn't have to create a remote called origin: Git uses this name by default when we clone a repository. (This is why origin was a sensible choice earlier when we were setting up remotes by hand.)

Experiment 7

AIM→ Fork and Commit

About forks

- A Fork is copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original Project.
- Most commonly, forks are used to either propose changes to someone else's project or to use someone else's project as a starting point for your own idea. You can fork a repository to create a copy of the repository and make changes without affecting the upstream repository.

Propose changes to someone else's project

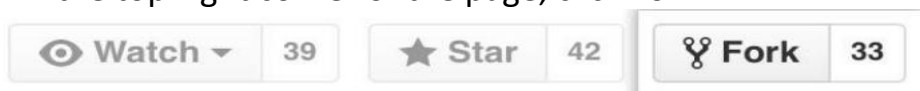
For example, you can use forks to propose changes related to fixing a bug. Rather than logging an issue for a bug you've found, you can:

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

Forking a repository

You might fork a project to propose changes to the upstream, or original, repository. In this case, it's good practice to regularly sync your fork with the upstream repository. To do this, you'll need to use Git on the command line.

1. On GitHub.com, navigate to your repository.
2. In the top-right corner of the page, click Fork.



Performing Fork & Commit Through GitBash

1. Create a New Repo (or Folder) and Initialize the Git repository by Using 'git init'. And add git origin of your remote Repo to local Repo.

2. Type “Git pull https://github.com/mehvish002/experiment-1.2-.git” on CLI.
Git pull → This command is used to fetch the remote repo or to clone the repo.
3. Create a new Branch “Feature-1”. Open and file and do changes in it and commit it.

```

MINGW64:/c/Users/LENOVO/OneDrive/Desktop/SCM

LENOVO@LAPTOP-BOK5EHRI MINGW64 ~/OneDrive/Desktop/SCM (code)
$ git init
Reinitialized existing Git repository in C:/Users/LENOVO/OneDrive/Desktop/SCM/.git/

LENOVO@LAPTOP-BOK5EHRI MINGW64 ~/OneDrive/Desktop/SCM (code)
$ git remote add origin https://github.com/aftab1911/Aftab_SCM
error: remote origin already exists.

LENOVO@LAPTOP-BOK5EHRI MINGW64 ~/OneDrive/Desktop/SCM (code)
$ git remote -v
origin https://github.com/Group08-Chitkara-University/2110990103.git (fetch)
origin https://github.com/Group08-Chitkara-University/2110990103.git (push)

LENOVO@LAPTOP-BOK5EHRI MINGW64 ~/OneDrive/Desktop/SCM (code)
$ git pull https://github.com/aftab1911/Aftab_SCM
fatal: unable to access 'https://github.com/aftab1911/Aftab_SCM/': Could not resolve host: github.com

LENOVO@LAPTOP-BOK5EHRI MINGW64 ~/OneDrive/Desktop/SCM (code)
$ git status
On branch code
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    FUNCTION_OVERLOADING.cpp

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        FUNCTION_OVERLOADING1.cpp

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

LENOVO@LAPTOP-BOK5EHRI MINGW64 ~/OneDrive/Desktop/SCM (code)
$

```

```

MINGW64:/c/Users/HP/OneDrive/Desktop/BTECH SEM 2/scm project

HP@Riddhi MINGW64 ~/OneDrive/Desktop/BTECH SEM 2/scm project (Feature-1)
$ git status
On branch Feature-1
nothing to commit, working tree clean

HP@Riddhi MINGW64 ~/OneDrive/Desktop/BTECH SEM 2/scm project (Feature-1)
$ git add -A

HP@Riddhi MINGW64 ~/OneDrive/Desktop/BTECH SEM 2/scm project (Feature-1)
$ git commit -m "changes have been made in Ft-1 Branch"
On branch Feature-1
nothing to commit, working tree clean

HP@Riddhi MINGW64 ~/OneDrive/Desktop/BTECH SEM 2/scm project (Feature-1)
$

```

```
MINGW64:/c/Users/LENOVO/OneDrive/Desktop/SCM

LENOVO@LAPTOP-B0K5EHRI MINGW64 ~/OneDrive/Desktop/SCM (code)
$ git status
On branch code
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    renamed:    FUNCTION_OVERLOADING.cpp -> FUNCTION_OVERLOADING1.cpp

LENOVO@LAPTOP-B0K5EHRI MINGW64 ~/OneDrive/Desktop/SCM (code)
$ git add -A

LENOVO@LAPTOP-B0K5EHRI MINGW64 ~/OneDrive/Desktop/SCM (code)
$ git commit -m "changes have been made in code Branch"
[code c26142d] changes have been made in code Branch
 1 file changed, 10 insertions(+), 4 deletions(-)
 rename FUNCTION_OVERLOADING.cpp => FUNCTION_OVERLOADING1.cpp (77%)

LENOVO@LAPTOP-B0K5EHRI MINGW64 ~/OneDrive/Desktop/SCM (code)
$
```

Experiment 8

AIM→ Merge and Resolve conflicts created due to own activity and collaborators activity.

Theory:

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

The simplest way to resolve a conflict, given that you know whose version of the file you want to keep, is to use the command line git program to tell git to use either your changes (the person doing the merge), or their changes (the other collaborator).

- keep your collaborators file: git checkout --theirs conflicted_file.Rmd
- keep your own file: git checkout --ours conflicted_file.Rmd

Once you have run that command, then run add, commit, and push the changes as normal.

Procedure for GitBash

1. Make changes in master branch and commit those changes, checkout to "Feature-1" branch and again make changes then commit them. Now checkout to master branch and merge the Feature-1 branch in master.

Commit in Master Branch

```
MINGW64/c/Users/LENOVO/OneDrive/Desktop/SCM (code)
$ git log
commit 124d0e76dfe021146eb71948bc11062 (HEAD -> code)
Author: Aftab Alam <aftab0103.be21@chitkara.edu.in>
Date:   Thu Jun 2 11:41:06 2022 +0530

    changes have been made in code Branch

commit bbd06e220de51248d1342b9282e5430ad1de377c (origin/main, main, Feature-1)
Author: Aftab Alam <aftab0103.be21@chitkara.edu.in>
Date:   Sun Apr 10 23:25:00 2022 +0530

    Practical File

commit 401babf8ac69e0aas5f17c26ceb871574f06991d
Author: Aftab Alam <aftab0103.be21@chitkara.edu.in>
Date:   Sun Apr 10 23:20:05 2022 +0530

    Practical File

commit 8b245e080e1b692352d9bdf6840f5023140e16fe
Author: Aftab Alam <aftab0103.be21@chitkara.edu.in>
Date:   Sun Apr 10 23:04:45 2022 +0530

    Delete TASK FILE.pdf

commit 012fc92180f1a85b9d3fd8f1feb3d6bcbfedbf64
Author: Aftab Alam <aftab0103.be21@chitkara.edu.in>
Date:   Sun Apr 10 23:01:36 2022 +0530

    Add Files via upload

commit dfd1eb4ccccc2463716de7ed7116bee3775bba0cc
Author: Aftab Alam <aftab0103.be21@chitkara.edu.in>
Date:   Sun Apr 10 23:00:51 2022 +0530

    Delete TASK FILE

commit 5d3c911109bb4c5b70cd186e507965c780272420
Author: Aftab Alam <aftab0103.be21@chitkara.edu.in>
Date:   Sun Apr 10 23:00:32 2022 +0530

    Create TASK FILE

commit 482f1802f4836051c111d685d26648766d025d
Author: Aftab Alam <aftab0103.be21@chitkara.edu.in>
Date:   Sun Apr 10 20:50:57 2022 +0530

    I am changing FUNCTION_OVERLOADING
LENOVOBLAPTOP-B0K5EH01 MINGW64 ~/OneDrive/Desktop/SCM (code)
$ |
```

Commit in Feature-1 Branch

```

MINGW64:/c/Users/LENOVO/OneDrive/Desktop/SCM

LENOVO@LAPTOP-B0K5EHRI MINGW64 ~/OneDrive/Desktop/SCM (feature-1)
$ git add -A

LENOVO@LAPTOP-B0K5EHRI MINGW64 ~/OneDrive/Desktop/SCM (feature-1)
$ git commit -m "conflict for solving merge (master)"
On branch feature-1
nothing to commit, working tree clean

LENOVO@LAPTOP-B0K5EHRI MINGW64 ~/OneDrive/Desktop/SCM (feature-1)
$ |

```

Due To My Activities , faces conflict during Merging

```

MINGW64:/c/Users/LENOVO/OneDrive/Desktop/SCM

LENOVO@LAPTOP-B0K5EHRI MINGW64 ~/OneDrive/Desktop/SCM (feature-1)
$ git merge code
Updating bbd06e2..c26142d
Fast-forward
 FUNCTION_OVERLOADING.cpp => FUNCTION_OVERLOADING1.cpp | 14 ++++++-----
 1 file changed, 10 insertions(+), 4 deletions(-)
 rename FUNCTION_OVERLOADING.cpp => FUNCTION_OVERLOADING1.cpp (77%)

LENOVO@LAPTOP-B0K5EHRI MINGW64 ~/OneDrive/Desktop/SCM (feature-1)
$ |

```

2. Use Command “Git mergetool” to solve the conflict git-mergetool - Run merge conflict resolution tools to resolve merge conflicts.

```

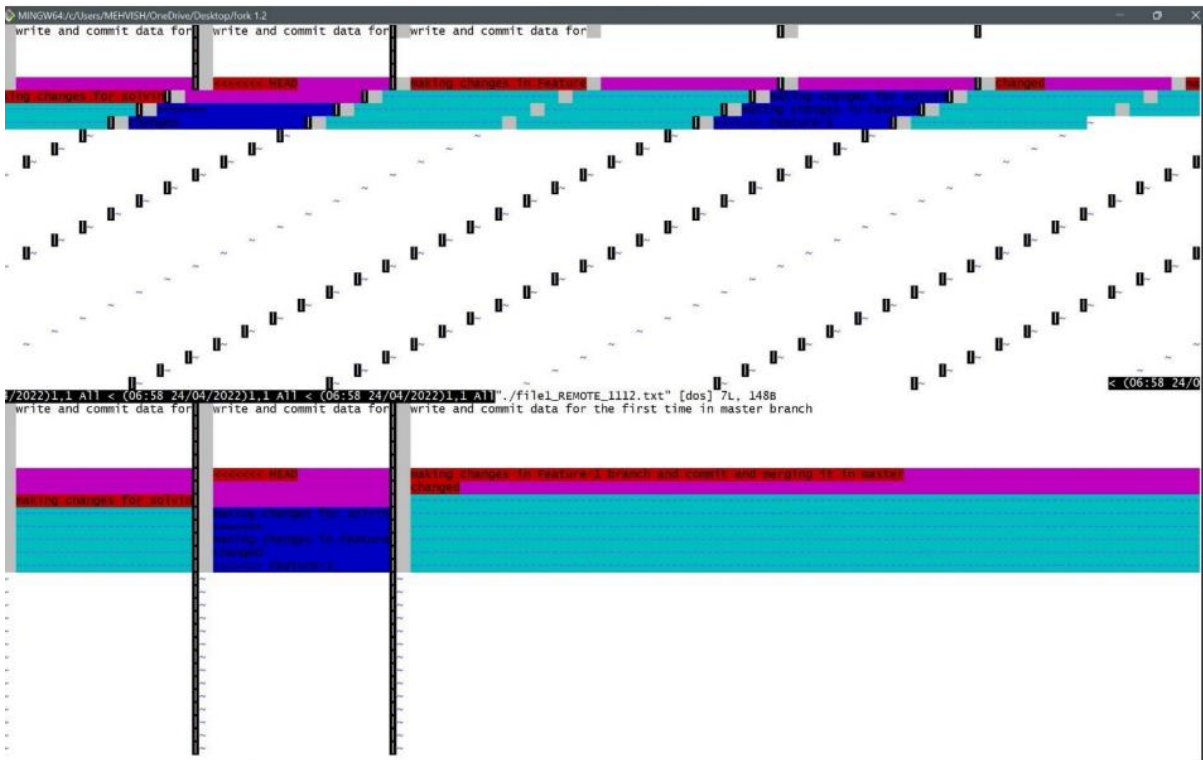
MINGW64:/c/Users/MEHVISH/OneDrive/Desktop/fork 1.2
write and commit data for the first time in master branch

This message is displayed because 'merge.tool' is not configured.
See 'git mergetool --tool-help' or 'git help config' for more details.
'git mergetool' will now attempt to use one of the following tools:
opendiff kdiff3 tkdiff xxdiff meld tortoisemerge gvimdiff diffuze diffmerge ecmerge p4merge araxis bc codecompare smerge emerge vimdiff nvimdiff
Merging:
file1.txt

Normal merge conflict for 'file1.txt':
{local}: created file
{remote}: created file
Hit return to start merge resolution tool (vimdiff): return
3 files to edit

```

3. Press “I” to insert, after insertion. Press “: wq”. The merge conflict is solved and our Feature-1 branch is merged to master branch



Now write the below command :

```
$ git commit -m "merge solved"
```

Experiment 9

AIM→ Reset & Revert

One of the lesser understood (and appreciated) aspects of working with Git is how easy it is to get back to where you were before—that is, how easy it is to undo even major changes in a repository. In this article, we'll take a quick look at how to reset, revert, and completely return to previous states, all with the simplicity and elegance of individual Git commands.

How to reset a Git commit

Let's start with the Git command reset. Practically, you can think of it as a "rollback"—it points your local environment back to a previous commit. By "local environment," we mean your local repository, staging area, and working directory.

If you want to review the changes of the file that were made between the current state and the state of a specific commit, then run the command below:

```
git diff <sha1-commit-hash> <file-path>
```

To reset a file to the state of a specific commit, run the [git reset](#) command:

```
git reset <sha1-commit-hash> <file-path>
```

You can also effectively use the [git checkout](#) command:

```
git checkout <sha1-commit-hash> -- <file1-path> <file2-path>
```

If you want to reset to the commit before a specific one, append ~1 (where 1 is the number of commits you want to go back, it can be any number):

```
git checkout <sha1-commit-hash>~1 -- <file1-path> <file2-path>
```

The git reset command is used for:

- Returning the overall working tree to the previous committed state. It will discard commits or clear the changes that were not committed.
- Changing which commit a branch HEAD is pointing at. It can adjust the commit history that already exists.
- For unstaging a file.

The git revert command helps to:

- Rollback the committed changes;
- Generate a new commit by inverting a specific commit. So, it can add a brand new commit history, yet can't modify the one that already exists
- Overwrite the files that are included in the working directory

Task 2 (Project)

Creating a distributed repository and adding members in project team:


Step 1 : Create a new repository and name it .

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?
[Import a repository.](#)

Owner *

Repository name *


 aftab1911 ▾

/


✓

Great repository names are short, lowercase, and contain only numbers, lowercase letters, and hyphens. Aftab_SCM is available. e. Need inspiration? How about [scaling-octo-tribble?](#)

Description (optional)

☒  **Public**

Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**

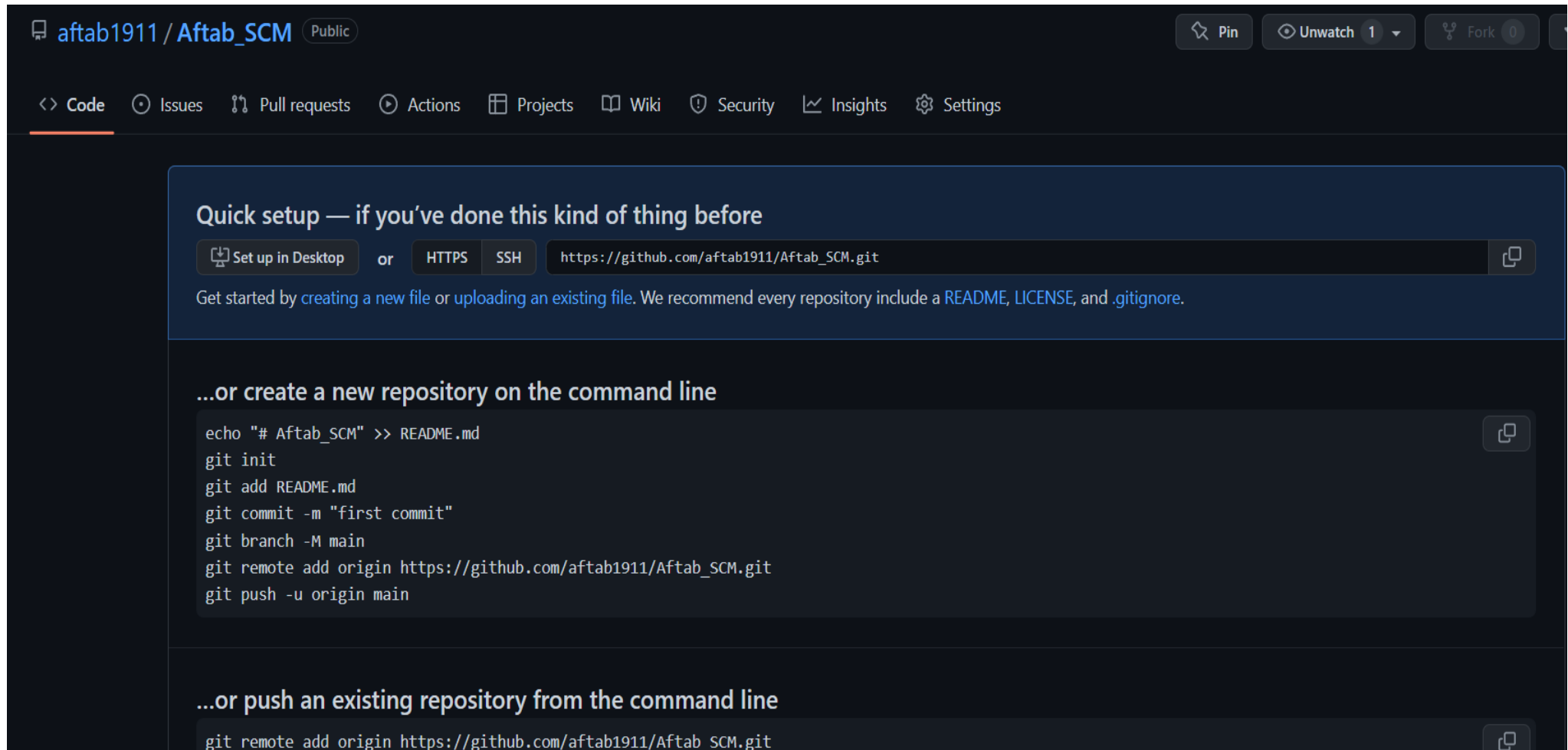
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☐ **Add a README file**
This is where you can write a long description for your project. [Learn more.](#)

Create repository

Step 2: Go to repository settings .



The screenshot shows the GitHub repository settings page for 'aftab1911 / Aftab_SCM'. The repository is public. The navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The main content area has a dark theme and includes a 'Quick setup' section with options to 'Set up in Desktop', 'HTTPS', or 'SSH'. The 'SSH' option is selected, showing the URL 'https://github.com/aftab1911/Aftab_SCM.git'. Below this, there is a section for creating a new repository on the command line with a list of commands: 'echo "# Aftab_SCM" >> README.md', 'git init', 'git add README.md', 'git commit -m "first commit"', 'git branch -M main', 'git remote add origin https://github.com/aftab1911/Aftab_SCM.git', and 'git push -u origin main'. At the bottom, there is a section for pushing an existing repository from the command line with the command 'git remote add origin https://github.com/aftab1911/Aftab_SCM.git'.

aftab1911 / Aftab_SCM Public

Pin Unwatch 1 Fork 0

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH `https://github.com/aftab1911/Aftab_SCM.git`

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

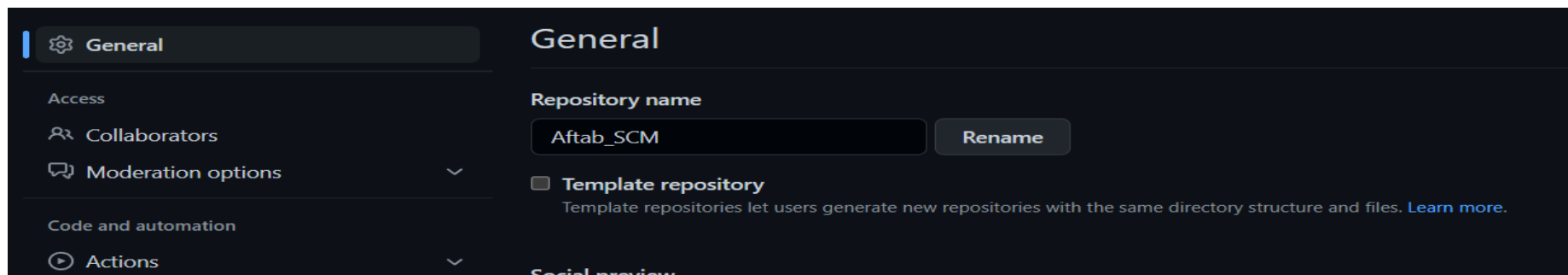
...or create a new repository on the command line

```
echo "# Aftab_SCM" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/aftab1911/Aftab_SCM.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/aftab1911/Aftab_SCM.git
```

Step 3: Now go to collaborators section



The screenshot shows the 'General' tab of the repository settings for 'Aftab_SCM'. The left sidebar has a 'General' tab selected, with other options like 'Access', 'Collaborators', 'Moderation options', 'Code and automation', and 'Actions'. The main content area shows the 'Repository name' as 'Aftab_SCM' with a 'Rename' button. There is a checkbox for 'Template repository' which is currently unchecked. Below this, there is a link to 'Learn more' about template repositories. The 'Social preview' section is partially visible at the bottom.

General

Access

Collaborators

Moderation options

Code and automation

Actions

Repository name

Aftab_SCM Rename

☐ **Template repository**

Template repositories let users generate new repositories with the same directory structure and files. [Learn more](#).

Social preview

Step 4: There y'll be having a add people option from where you can search your team mates .

The screenshot displays the GitHub repository settings for 'Aftab_SCM' at the path `github.com/aftab1911/Aftab_SCM/settings/access`. The left sidebar contains a navigation menu with the following sections:

- General**
 - Access
 - Collaborators** (selected)
 - Moderation options
 - Code and automation
 - Actions
 - Webhooks
 - Environments
 - Pages
 - Security
 - Code security and analysis
 - Deploy keys
 - Secrets
 - Integrations
 - GitHub apps
 - Email notifications
 - Autolink references

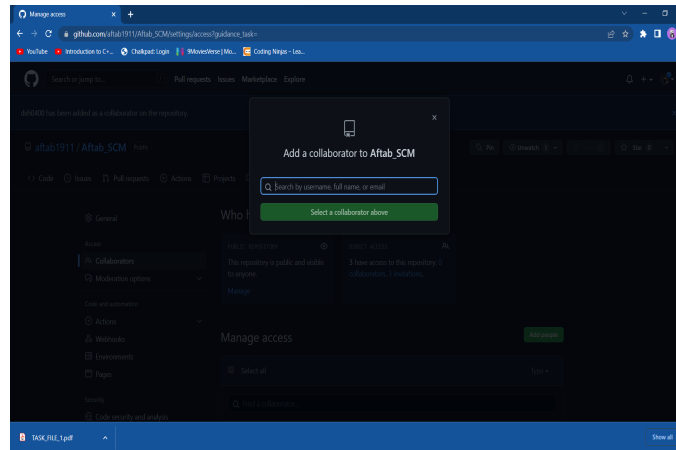
The main content area is titled 'Who has access' and contains two panels:

- PUBLIC REPOSITORY**: This repository is public and visible to anyone. [Manage](#)
- DIRECT ACCESS**: 0 collaborators have access to this repository. Only you can contribute to this repository.

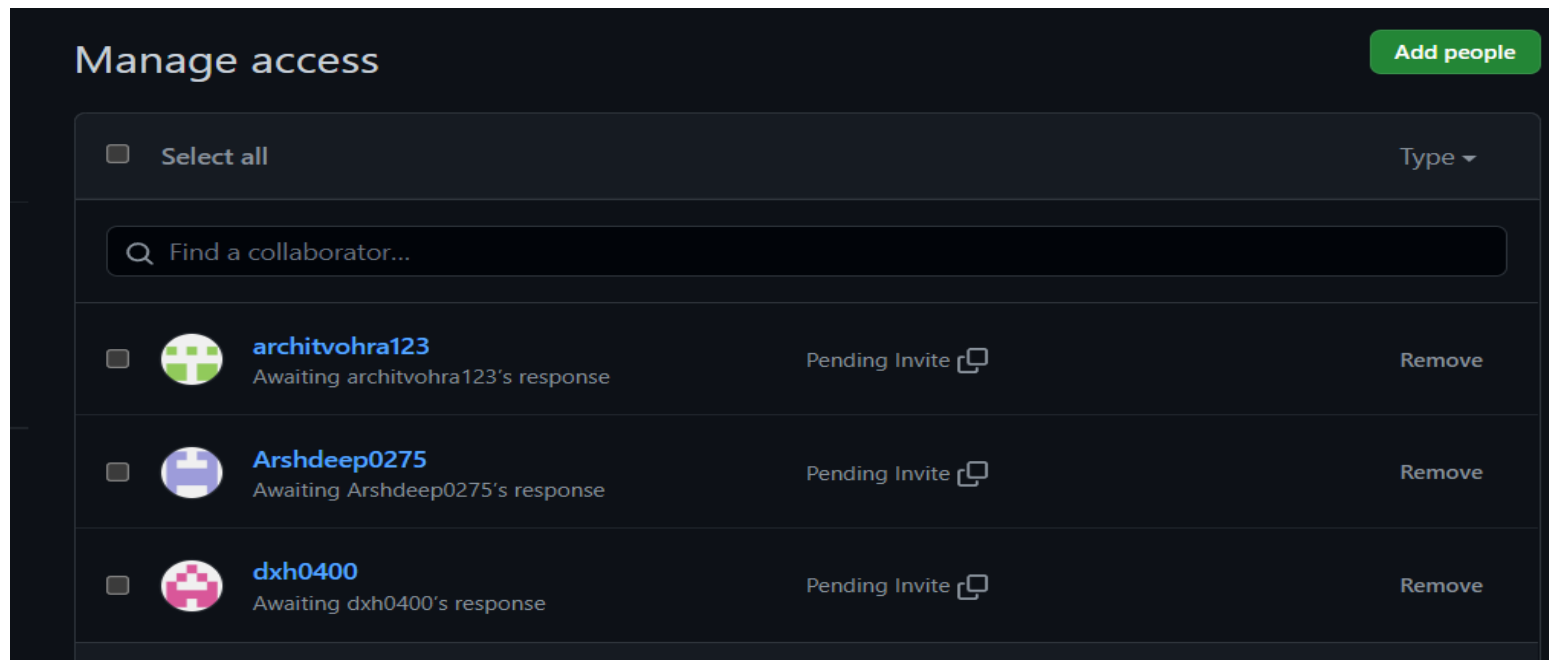
Below this is the 'Manage access' section, which features a large box with the message: 'You haven't invited any collaborators yet' and a green 'Add people' button.

The bottom of the browser window shows a taskbar with a file named 'TASK_FILE_1.pdf' and a 'Show all' button.

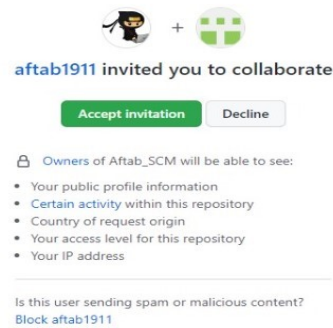
Step 5: Search you team mates with there GitHub ids.



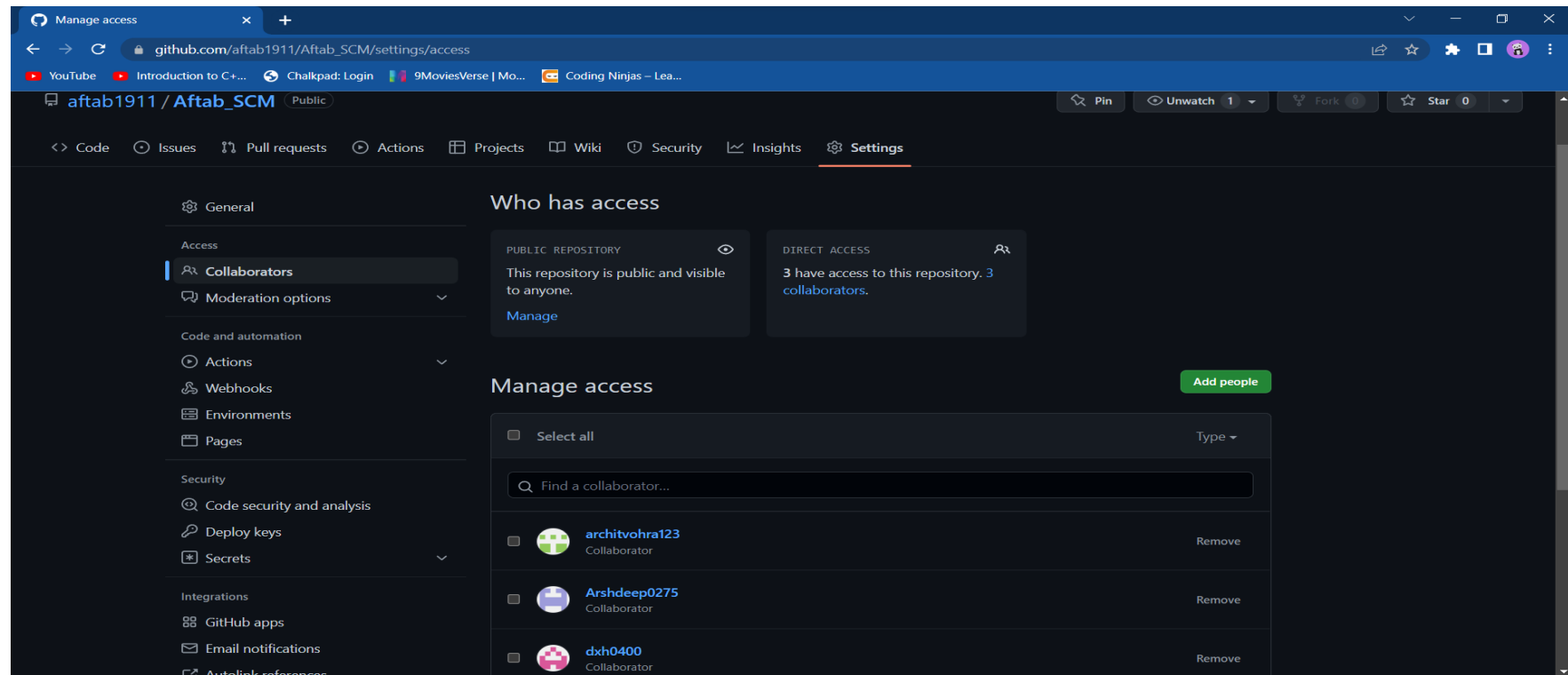
After adding them as a collaborators they will get an email regarding accepting or rejecting the invitation.



Team mates will receive invitation like this :



After accepting the invitation they'll be added as collaborator .

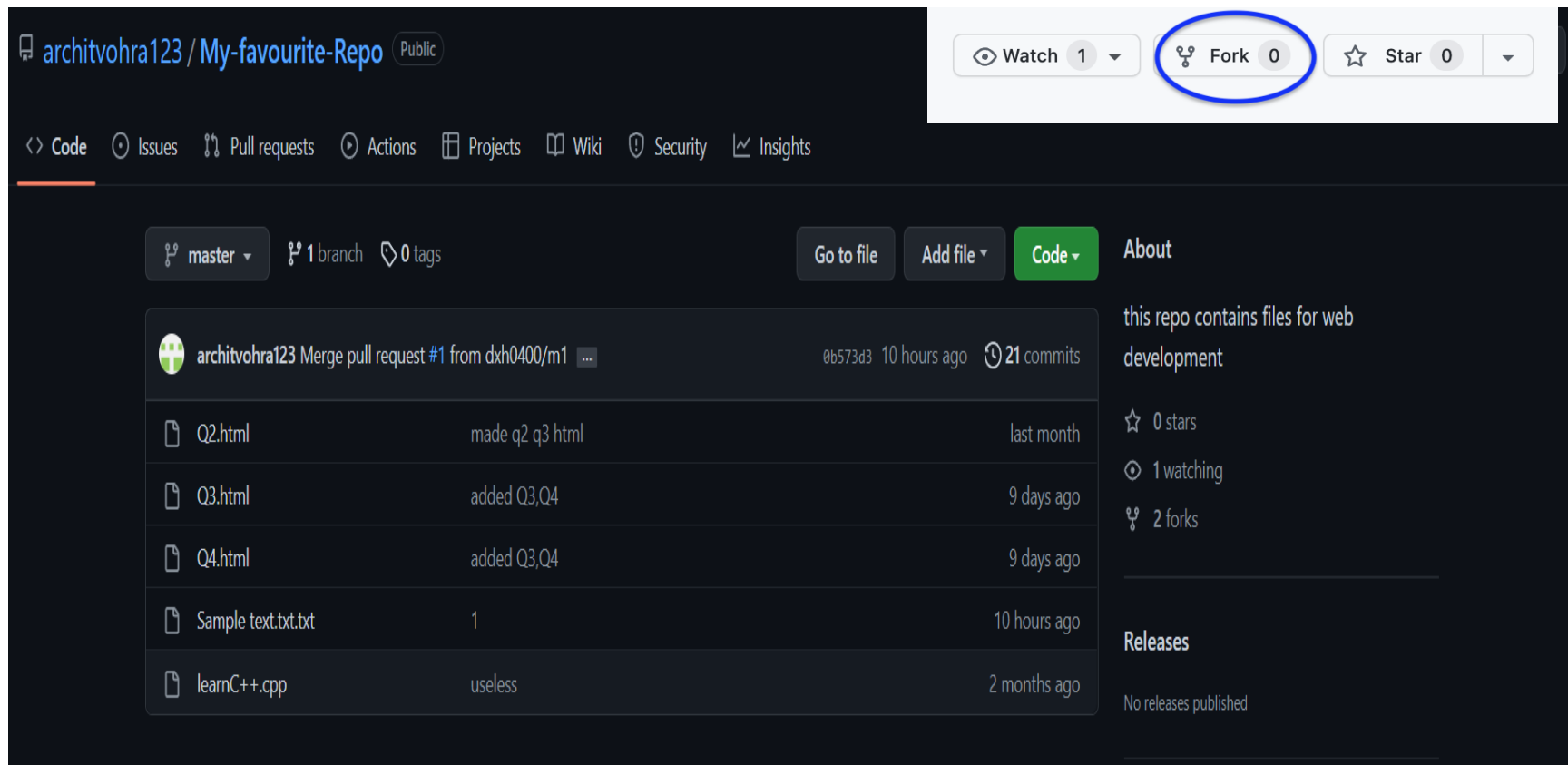


Open and close a pull request :

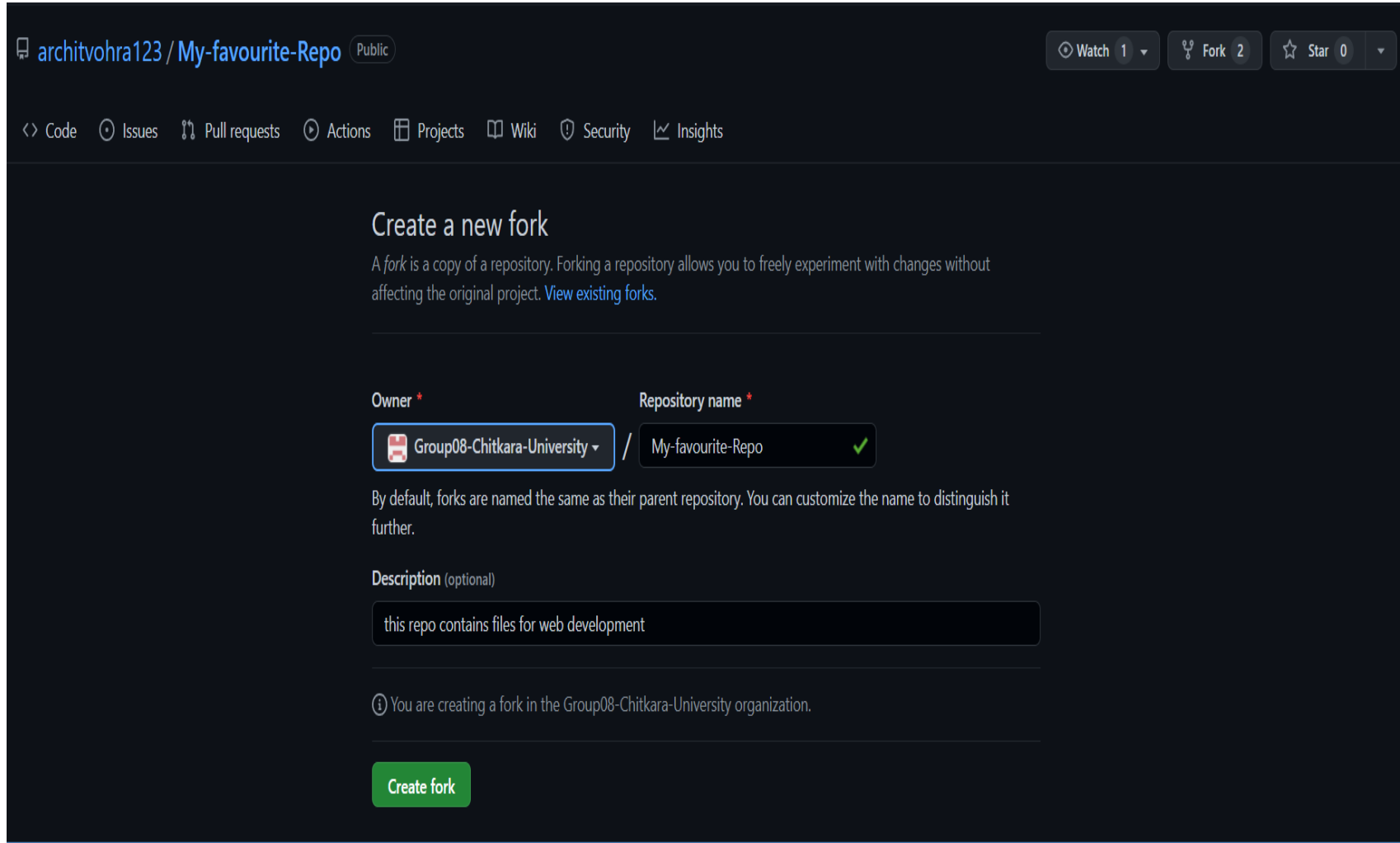
To make a pull request the go through the following steps :

- Fork someones's repository .
- clone that repo to your local repository .
- make some changes .
- make the pull request .

Step 1: Forking the repository .



Step 2: Name the repository and create the fork .



The screenshot shows the GitHub interface for creating a new fork. At the top, the repository path is 'architvohra123 / My-favourite-Repo' with a 'Public' badge. To the right are buttons for 'Watch 1', 'Fork 2', and 'Star 0'. Below this is a navigation bar with links for 'Code', 'Issues', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', and 'Insights'. The main heading is 'Create a new fork', followed by a descriptive paragraph about forking. The form has two required fields: 'Owner' and 'Repository name'. The 'Owner' dropdown is set to 'Group08-Chitkara-University'. The 'Repository name' field contains 'My-favourite-Repo' with a green checkmark. Below these fields is a note about default naming and a 'Description (optional)' text area containing 'this repo contains files for web development'. A message at the bottom states 'You are creating a fork in the Group08-Chitkara-University organization.' and a green 'Create fork' button is at the bottom.

architvohra123 / My-favourite-Repo Public

Watch 1 Fork 2 Star 0

Code Issues Pull requests Actions Projects Wiki Security Insights

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. [View existing forks.](#)

Owner * Repository name *

Group08-Chitkara-University / My-favourite-Repo ✓

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

Description (optional)

this repo contains files for web development

ⓘ You are creating a fork in the Group08-Chitkara-University organization.

Create fork

Y'll be having your forked repo :

aftab1911 / My-favourite-Repo Public

forked from architvohra123/My-favourite-Repo

<> Code Pull requests Actions Projects Wiki Security Insights Settings

master 1 branch 0 tags

Go to file Add file Code

About

this repo contains files for web development

0 stars
0 watching
2 forks

Releases

No releases published
[Create a new release](#)

Packages

No packages published
[Publish your first package](#)

This branch is up to date with architvohra123/My-favourite-Repo:master. [Contribute](#) [Fetch upstream](#)

architvohra123 Merge pull request [architvohra123#1](#) from dxh0400/m1 [...](#) 0b573d3 10 hours ago 21 commits

| | | |
|---------------------|-----------------|--------------|
| Q2.html | made q2 q3 html | last month |
| Q3.html | added Q3,Q4 | 9 days ago |
| Q4.html | added Q3,Q4 | 9 days ago |
| Sample text.txt.txt | 1 | 10 hours ago |
| learnC++.cpp | useless | 2 months ago |

Help people interested in this repository understand your project by adding a README. [Add a README](#)

Step 3: Now clone the forked repo in your local space through the link

Go to file Add file Code

Clone ?

HTTPS SSH GitHub CLI

<https://github.com/aftab1911/My-favourite-Repo> [Copy](#)

Use Git or checkout with SVN using the web URL.

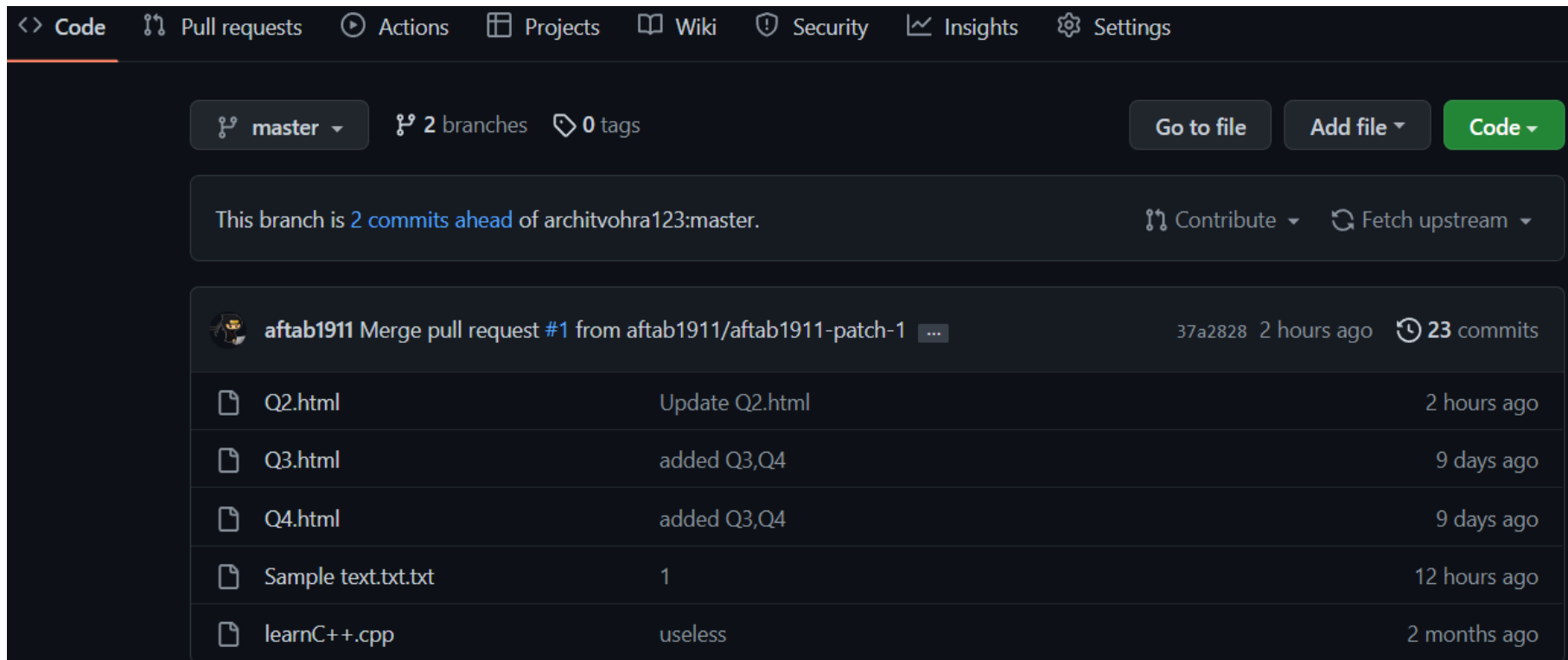
Open with GitHub Desktop

Download ZIP

Step 4: Make some changes in the code files and add them to staging area and then commit.

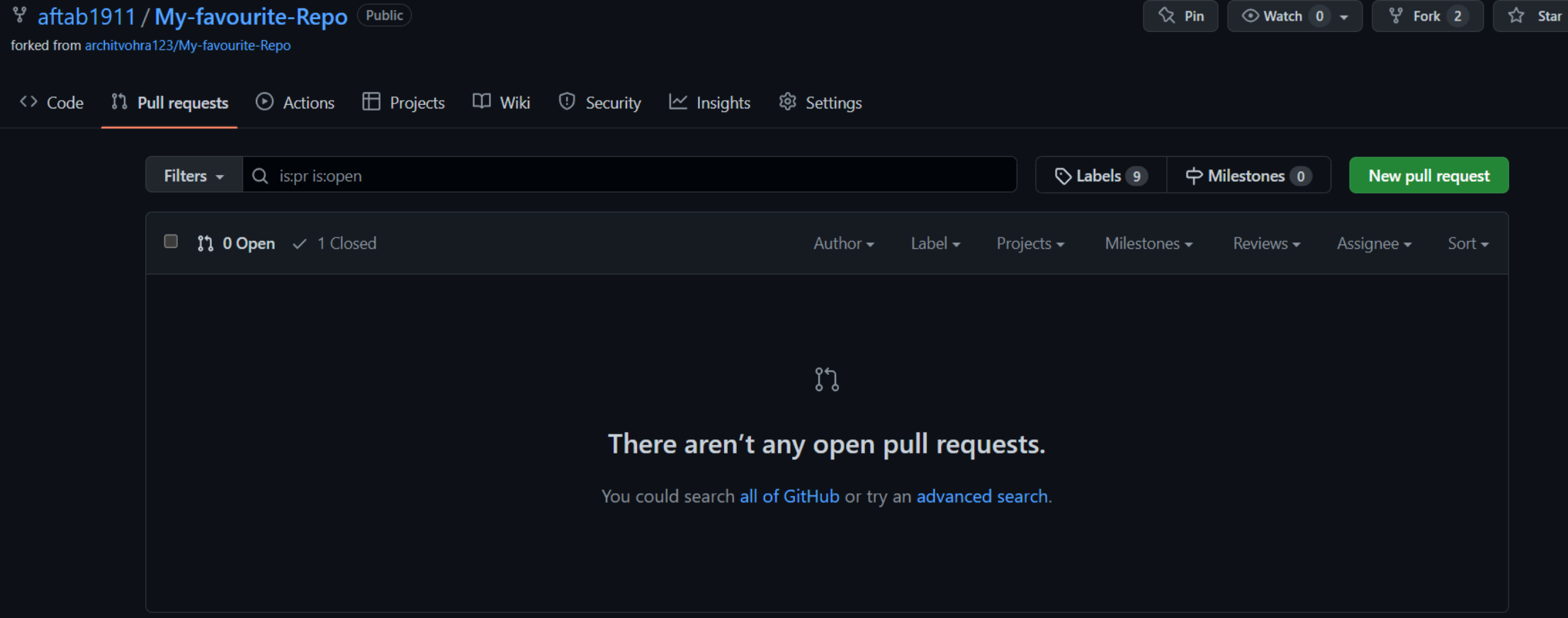
Step 5: Now push the changes to the forked repo .

Step 6: Now make the pull request .

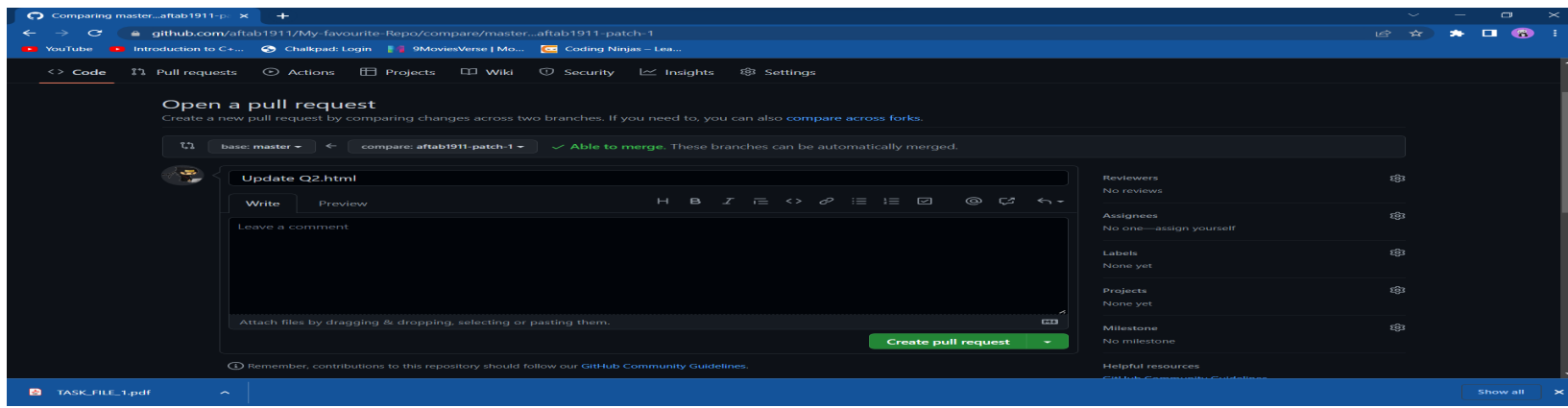


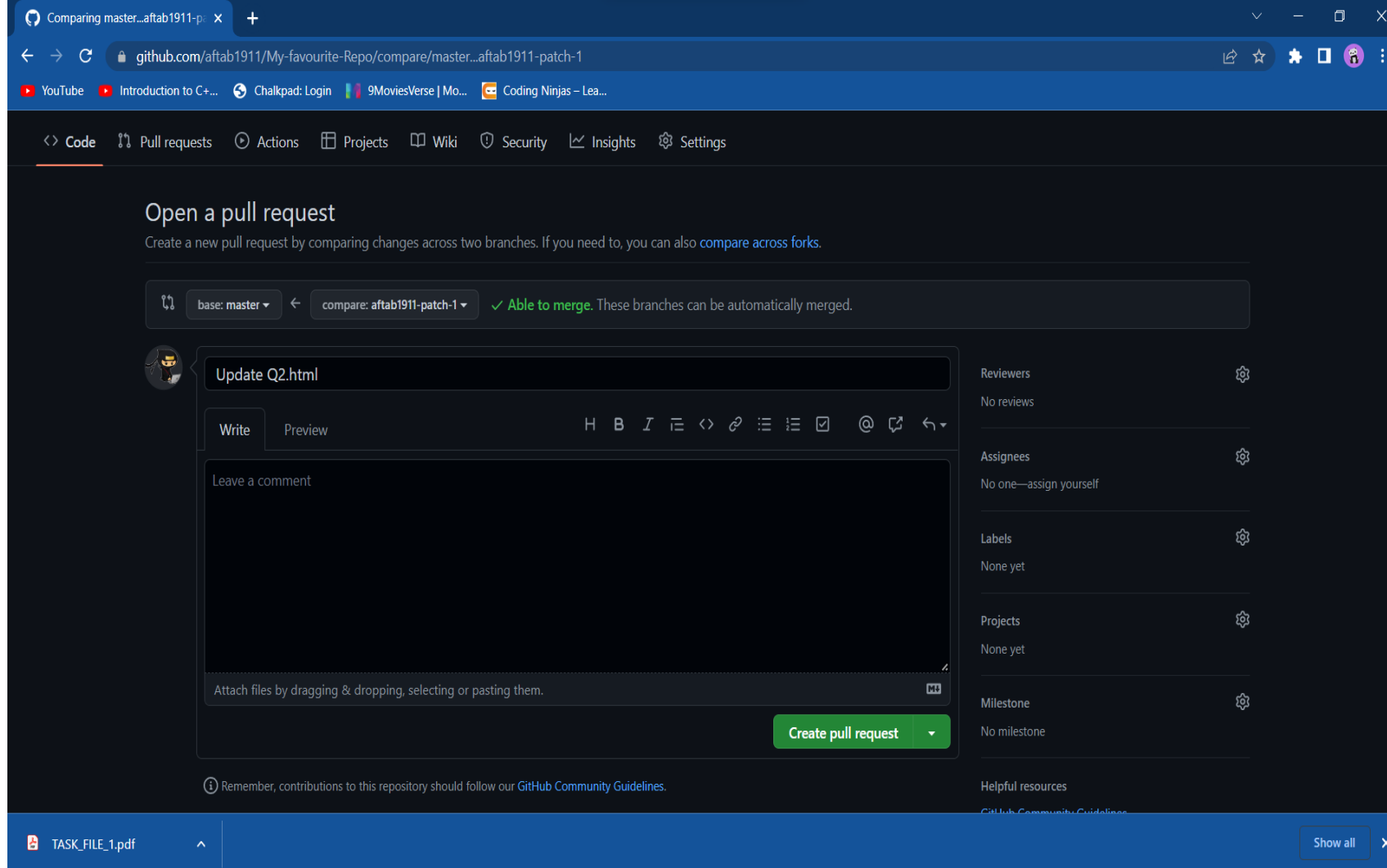
The screenshot displays the GitHub web interface for a repository. At the top, navigation tabs include Code, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. Below these, a branch selector shows 'master' with 2 branches and 0 tags. Action buttons for 'Go to file', 'Add file', and 'Code' are visible. A status bar indicates the current branch is 2 commits ahead of the upstream 'architvohra123:master'. A pull request merge summary for 'aftab1911' is shown, including the commit hash '37a2828' and the time '2 hours ago'. Below the summary is a table of file changes:

| File | Change | Time |
|---------------------|----------------|--------------|
| Q2.html | Update Q2.html | 2 hours ago |
| Q3.html | added Q3,Q4 | 9 days ago |
| Q4.html | added Q3,Q4 | 9 days ago |
| Sample text.txt.txt | 1 | 12 hours ago |
| learnC++.cpp | useless | 2 months ago |

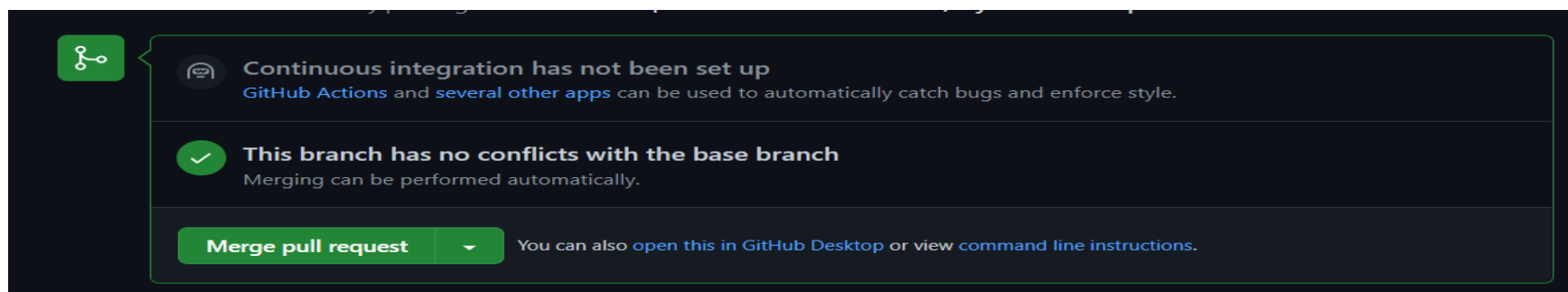


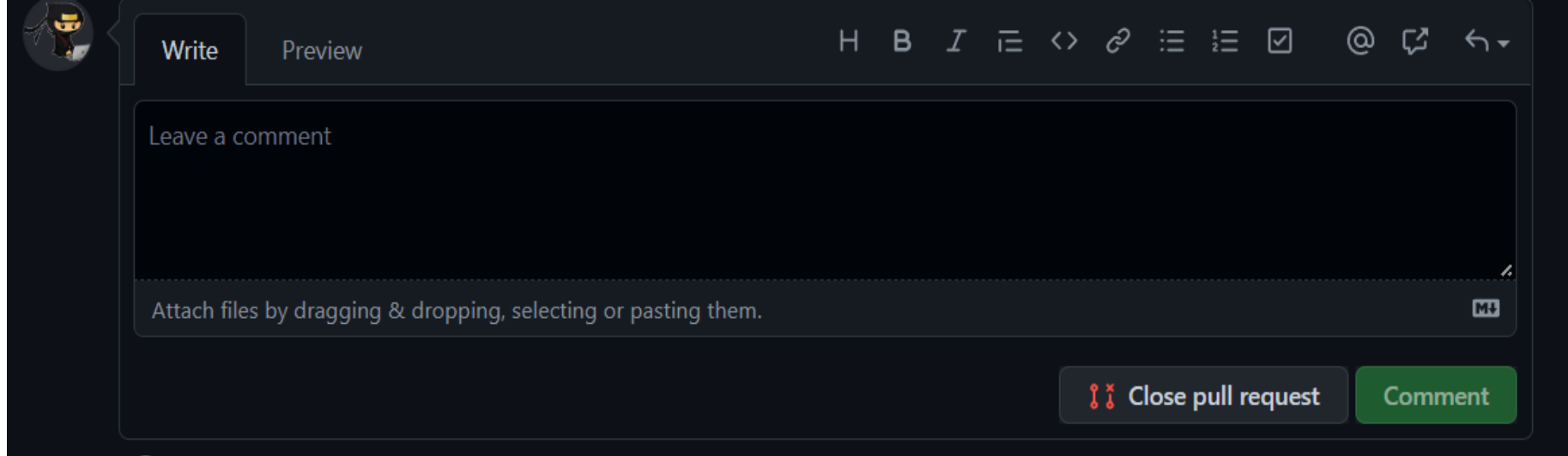
After clicking on “New pull request” you can create the pull request and also can see the changes you made in the files .



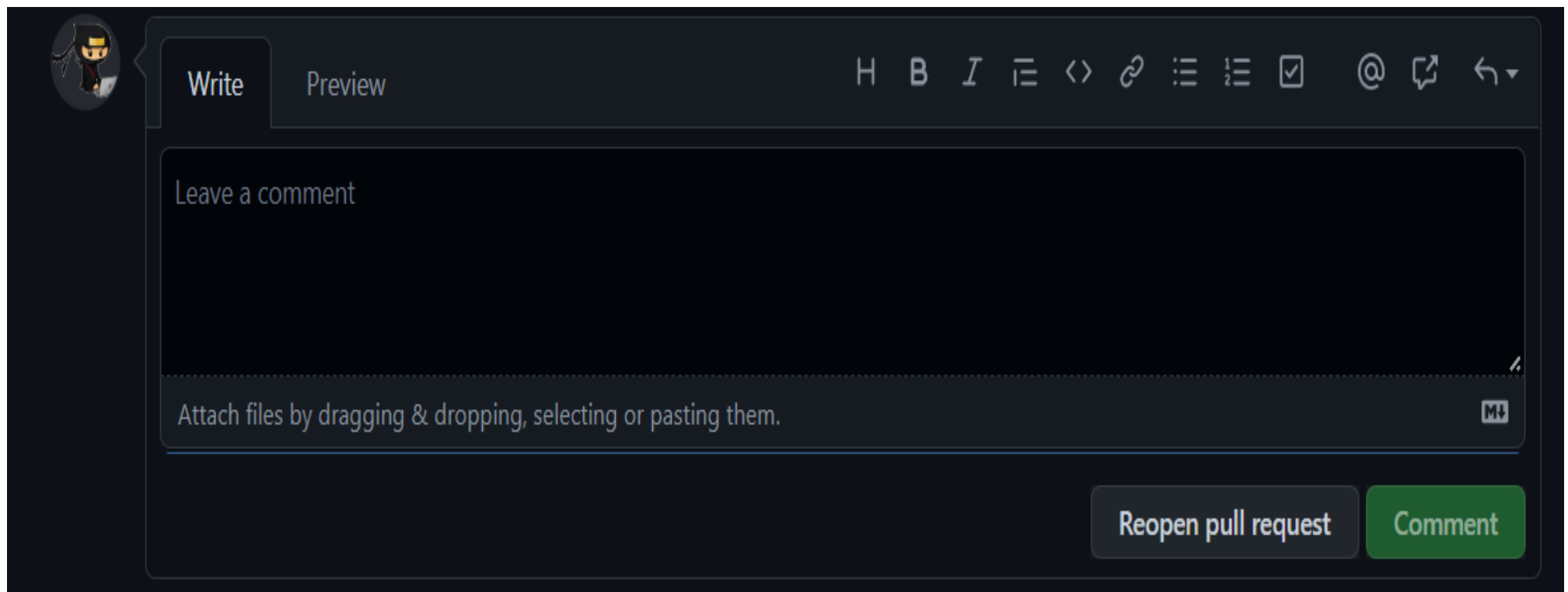


Now you can close the pull request you created or also you can merge that .





After closing the pull request you can also reopen that pull request .

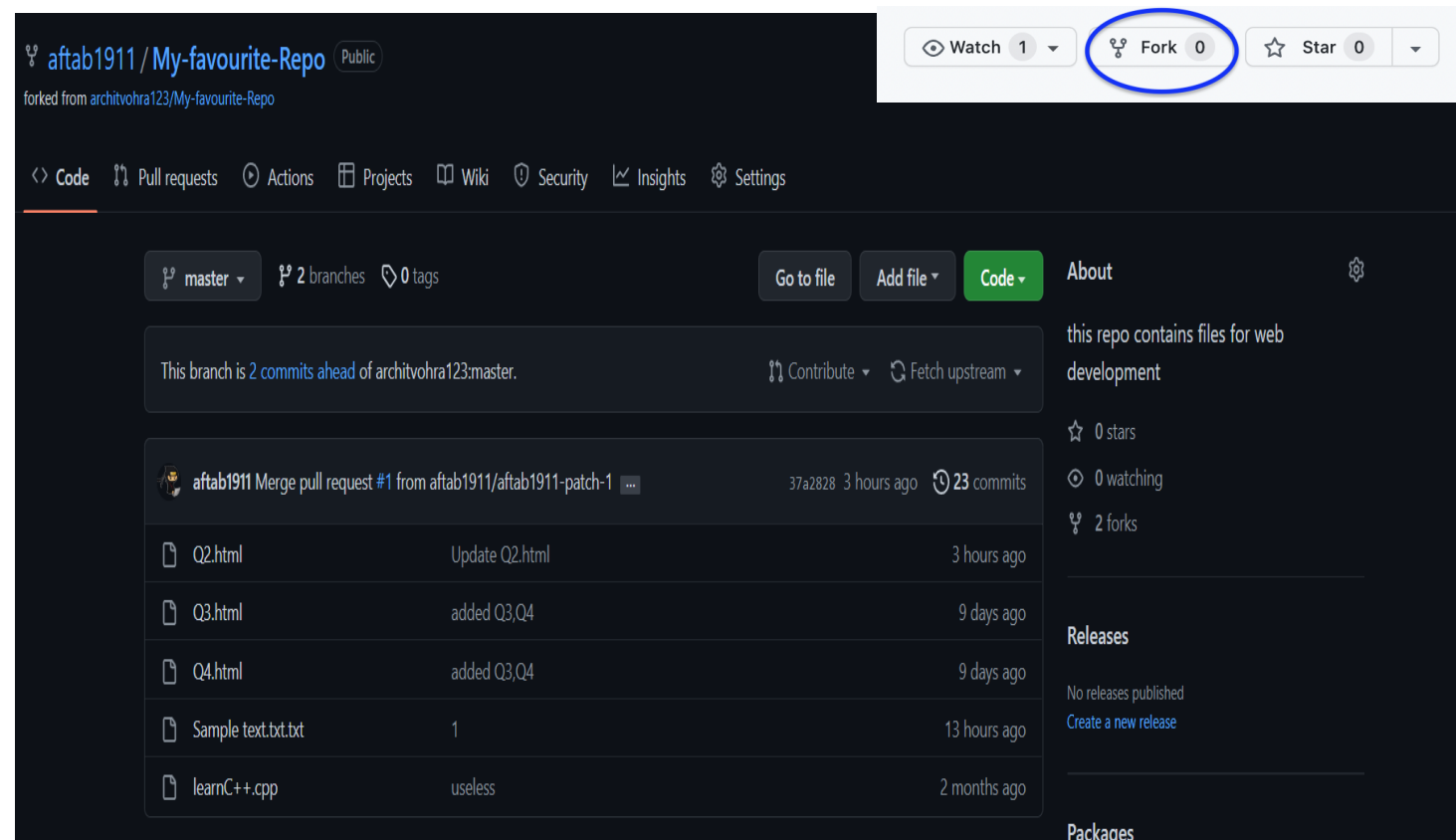


Creating pull requests on team members repo and closing pull request generated by team members :

Team members are : 1. Architvohra123
2. dxh0400
3. Arshdeep0275

Creating pull request on architvohra123 repository :

Step 1. Forking his repo.



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. [View existing forks.](#)

Owner *



Group08-Chitkara-University ▾



Repository name *


My-favourite-Repo



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

Description (optional)

this repo contains files for web development

 You are creating a fork in the Group08-Chitkara-University organization.

Create fork

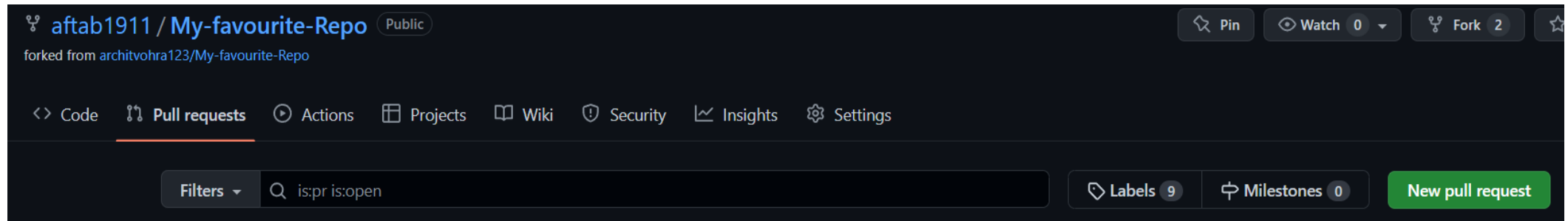
After forking

Step 2: Clone the repo on local space .

Step 3 : Make some changes and commit.

Step 4 : Push the changes to the forked repo.

Step 5: Make the pull request .



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: [architvohra123/My-favourite-R...](#) base: master ← head repository: [aftab1911/My-favourite-Repo](#) compare: master

✓ **Able to merge.** These branches can be automatically merged.

changes in code #2
changes

[View pull request](#)

Create another pull request to discuss and review the changes again. [Learn about pull requests](#)

[Create pull request](#)



change1

Write

Preview

H B I

Add a link, <Ctrl+k>

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.



☒ Allow edits by maintainers

Create pull request



This branch has no conflicts with the base branch

Only those with [write access](#) to this repository can merge pull requests.



Write

Preview

H B I

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.



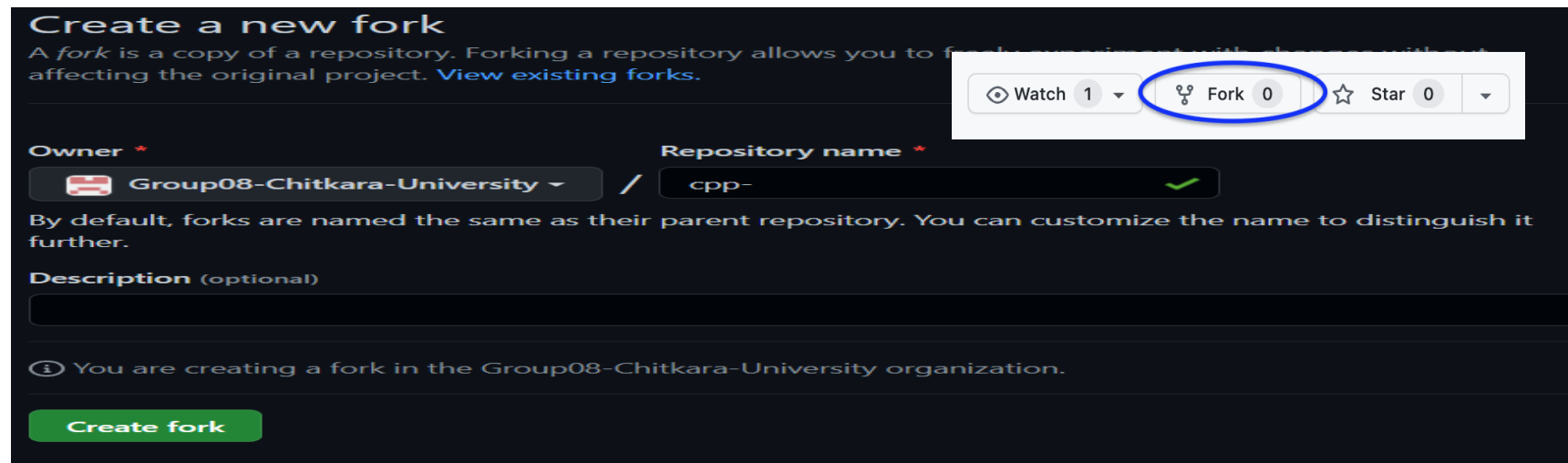
Close pull request

Comment

After making the pull request the owner of the repo will receive an email regarding whether he wants to merge those changes or not.



Creating pull request on dxh0400 repository :

Step 1: Forking the repo




Create a new fork
A fork is a copy of a repository. Forking a repository allows you to experiment with changes without affecting the original project. [View existing forks.](#)

Owner * **Repository name ***

 Group08-Chitkara-University / cpp- 

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

Description (optional)

 You are creating a fork in the Group08-Chitkara-University organization.

[Create fork](#)

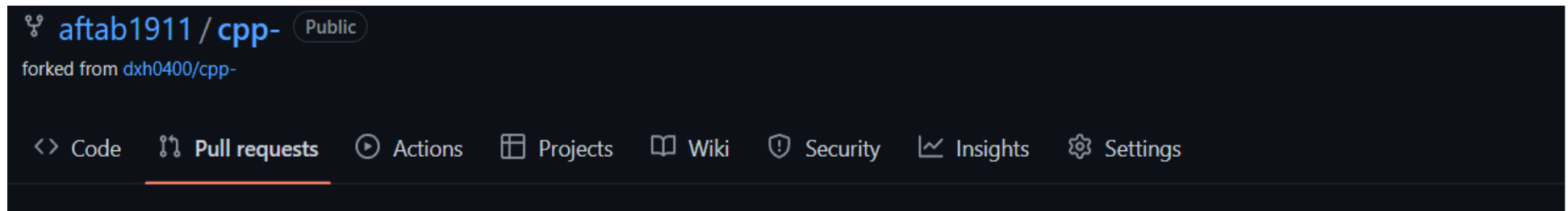
Watch 1 Fork 0 Star 0

Step 2: Clone that repo on local repo.

Step 3 : Make some changes in repo and commit .

Step 4: Push the code to the remote repo .

Step 5 : Make the pull request .



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: dxh0400/cpp-

base: main



head repository: aftab1911/cpp-

compare: main

✓ **Able to merge.** These branches can be automatically merged.



Hello World Program #1

No description available

[View pull request](#)

Create another pull request to discuss and review the changes again. [Learn about pull requests](#)

[Create pull request](#)

1 commit

1 file changed

1 contributor

Open a pull request



Update README.md

Write

Preview

H B I @

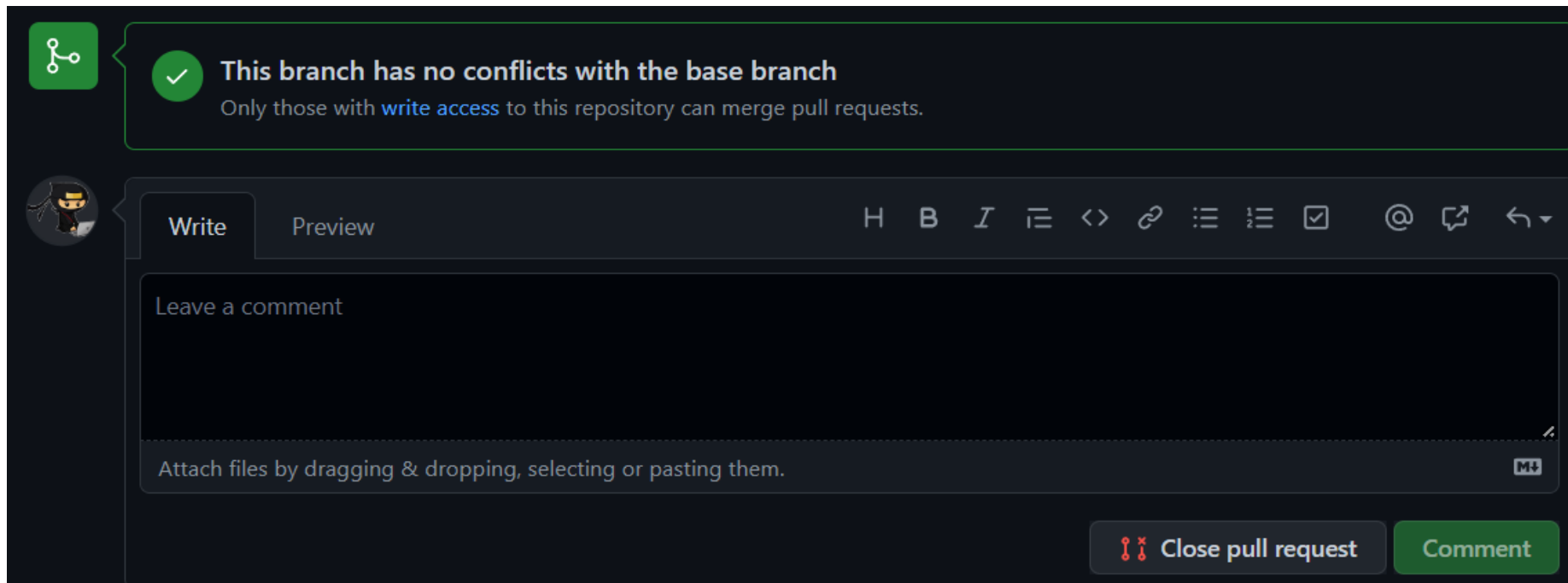
Leave a comment

Attach files by dragging & dropping, selecting or pasting them.



☒ Allow edits by maintainers



[Create pull request](#)









After making the pull request in this manner the team mate will receive the mail regarding he want to merge the changes with his main repo .

Closing the pull request generated by team member (Arshdeep0275):



So my team members forked my repo and made some changes in that repo and made a pull request as a result I received an email that whether I want to have that changes in my own main repo or not .

 Open Arshdeep0275 wants to merge 1 commit into [aftab1911:Code](#) from [Arshdeep0275:a1](#) 


 Conversation 0  Commits 1  Checks 0  Files changed 1



 Arshdeep0275 commented 2 hours ago First-time contributor  ...


No description provided.

  change fcfa927

Add more commits by pushing to the **a1** branch on **Arshdeep0275/Lets-Code**.

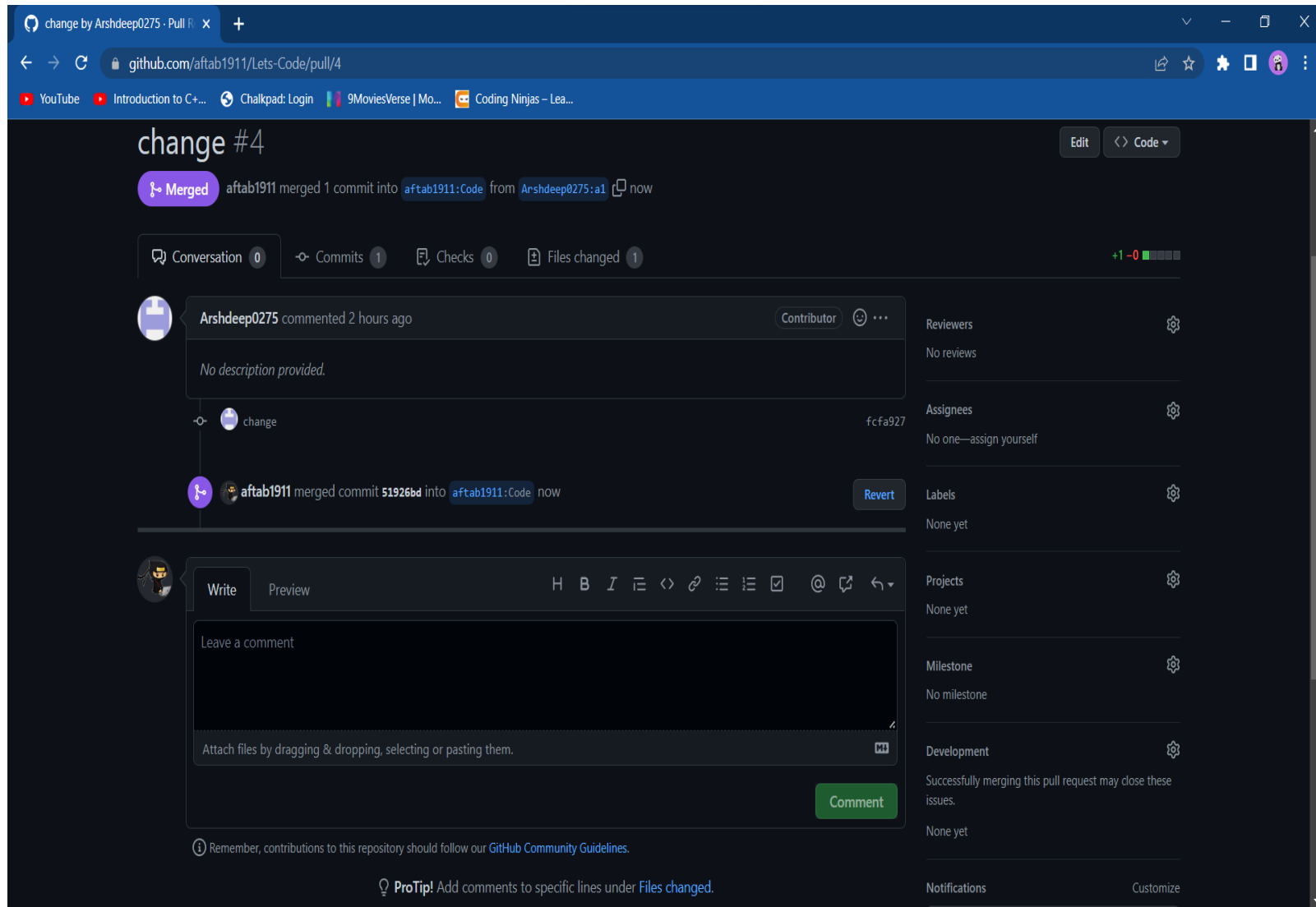


-  **Continuous integration has not been set up**
[GitHub Actions](#) and [several other apps](#) can be used to automatically catch bugs and enforce style.
-  **This branch has no conflicts with the base branch**
Merging can be performed automatically.

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

If you want to merge that change you can merge and if you want to close that pull request made by your team member without merging you can close as well .

So as a maintainer of the repo I am closing the pull request made by my team member



If you want to reopen that pull request you can reopen.

Closing the pull request generated by architvohra123:

Architvohra123 made some changes in repo as a result I received the pull request .

After visiting the link from email you can merge the pull request or also you can close the pull request without merging

The screenshot shows a GitHub pull request titled "changed 2D array #2". At the top, it indicates that user "architvohra123" wants to merge 1 commit into "aftab1911:Code" from "architvohra123:b1". Below this, there are tabs for "Conversation" (0), "Commits" (1), "Checks" (0), and "Files changed" (3). A comment from "architvohra123" is shown, stating "No description provided." Below the comment, a commit is listed: "changed 2D array" by "architvohra123" with commit hash "fa759da". A message below the commit says: "Add more commits by pushing to the **b1** branch on **architvohra123/Aftab-s-Repo**." At the bottom, there are two status boxes. The first box, with a warning icon, states: "Continuous integration has not been set up" and "GitHub Actions and several other apps can be used to automatically catch bugs and enforce style." The second box, with a checkmark icon, states: "This branch has no conflicts with the base branch" and "Merging can be performed automatically." At the very bottom, there is a green button labeled "Merge pull request" and a link that says "You can also open this in GitHub Desktop or view command line instructions."

Aftab1911 closing the pull request as the maintainer of the repository.

If you want to reopen the pull request you can simple clicking on reopen pull request

The screenshot displays a GitHub pull request titled "change1 #3". At the top, a red "Closed" label is visible, followed by the text "aftab1911 wants to merge 2 commits into architvohra123:master from aftab1911:master". Below this, a navigation bar shows "Conversation 0", "Commits 2", "Checks 0", and "Files changed 1".

The main content area shows a comment from "aftab1911" made 1 minute ago, stating "No description provided." Below the comment, a commit history is shown: "aftab1911 added 2 commits 3 hours ago", including "Update Q2.html" (Verified, 4b5b450) and "Merge pull request #1 from aftab1911/aftab1911-patch-1" (Verified, 37a2828). A red "Closed" label with a lock icon indicates the pull request was closed by "aftab1911" just now.

On the right side, there are sections for "Reviewers" (No reviews), "Assignees" (No one assigned), "Labels" (None yet), "Projects" (None yet), "Milestone" (No milestone), "Development" (Successfully merged issues), and "Notifications" (None yet).

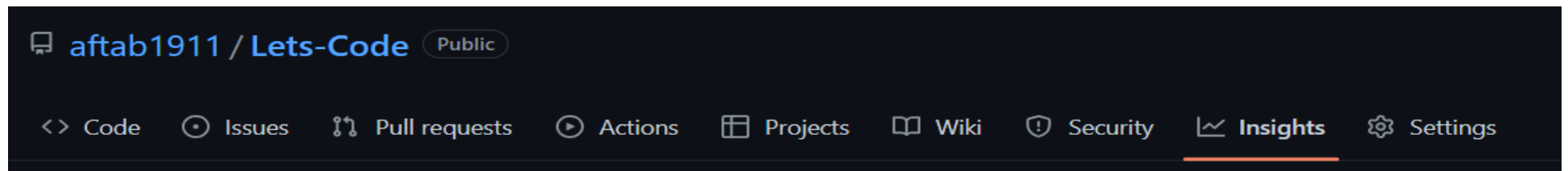
At the bottom, there is a "Write" tab with a "Preview" button and a rich text editor. The editor contains the placeholder text "Leave a comment" and "Attach files by dragging & dropping, selecting or pasting them." Below the editor, there are two buttons: "Reopen pull request" and "Comment".

Publishing and printing network graphs :

The network graph is one of the useful feature for developers on Github. It is used to display the branch history of the entire repository network ,including branches of the root repository and branches of forks that contain unique to network .

Accessing the network graph :

1. On Github.com, navigate to the main page of the repository .
2. Under your repository name, click insights .



3. In the left sidebar , click Networks .

