**Subject Nam**

**e:**

Source Code Management

**Subject Code:**

CS181

**Cluster:**

Beta

**Department:**

CSE



**Department of Computer Science & Engineering**

Chitkara University Institute of Engineering and Technology, Punjab

Jan- June (2021-22)

Institute/School **Chitkara University Institute of Engineering**

Name **and Technology**

**Department of Computer Science &**

Department Name

**Engineering**

**Bachelor of Engineering (B.E.), Computer**

Programme Name

**Science & Engineering**

|  |  |  |  |
| --- | --- | --- | --- |
| Course Name | **Source Code Management** | | Session **2021-22** |
| Course Code | **CS181** | | Semester/Batch **2nd/2021** |
| Group No G01 - | |

**Name: Abhinav Goyal**

**Roll No: 2110990049**

**Date: April 8, 2022**

22

**Table of Content**

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Title** | **Page No.** |
| 1 | Version Control with Git | I |
| 2 | Objective | II |
| 3 | Installation Of Git | 1 - 5 |
| 4 | Setting Up GitHub Account | 6 |
| 5 | Configuration of GIT | 7 |
| 6 | Program to Generate Logs | 8 - 11 |
| 7 | Create and Visualize Branches | 12-15 |
| 8 | GIT Lifecycle Description | 16 - 17 |
| 9 | Uploading Data on GitHub | 18-20 |
| 10 | Final project | 21 |

1. Version Control with GIT

Version control systems are software tools that help software teams manage changes to source code overtime. As development environments have accelerated, version control systems help software teams work faster and smarter.

A version Control system records all the changes made to a file or set of files, so a specific version may be called later if needed.

**How version control helps high performing development and DevOps teams prosper?**

1. Version Control Systems (VCS) have seen great improvements over the past few decades and some are better than others.

1. VCS are sometimes known as SCM (Source Code Management) tools or RCS (Revision Control System).

1. One of the most popular VCS tools in use today is called

Git. Git is a Distributed VCS, a category known as DVCS, more on that later. Like many of the most popular VCS systems available today, Git is free and open source.

2. Objective of GIT

Git is a version control system used for tracking changes in computer files. It is generally used for source code management in software development.

* Git is used to tracking changes in the source code
* The distributed version control tool is used for source code management
* It allows multiple developers to work together
* It supports non-linear development through its thousands of parallel branches

**Features of Git =>**

* Tracks history
* Free and open source
* Supports non-linear development
* Creates backups
* Scalable
* Supports collaboration
* Branching is easier
* Distributed development

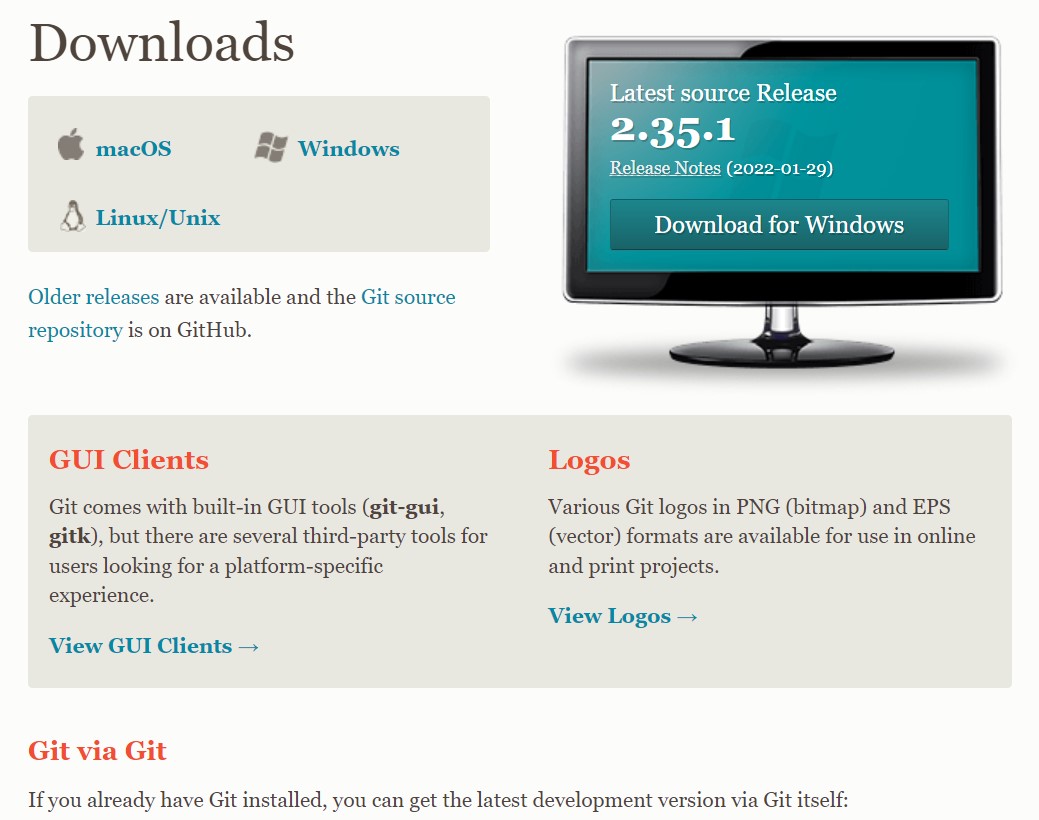
3. Installation of GIT

**Step 1 =>**

To download the Git installer, visit the Git official site and go to the download page.

The link for the download page is <https://git-scm.com/downloads>

The page looks like as: -



Click on the package given on the page as **download 2.23.0 for windows**. The download will start after selecting the package.

Now, the Git installer package has been downloaded.

**Step 2 =>**

Click on the download installer file and then click on next.

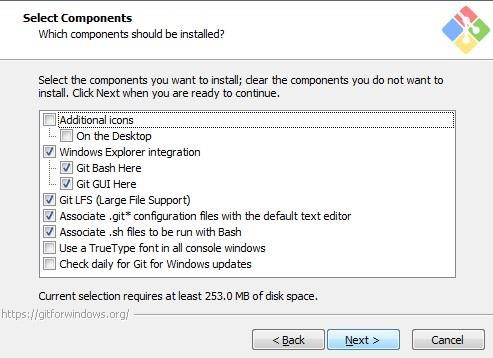
The page looks like as: -



**Step 3 =>**

Simply click on the next button as it automatically selects the required file.

The page looks like as: -



**Step 4 =>**

You can choose your preferred choice. Click next to continue.

The page looks like as: -



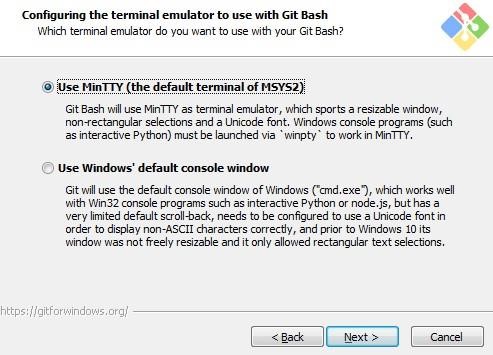
**Step 5 =>**

***Note*: -** Just simply click on next as it automatically selects the

required file.

The page looks like as:

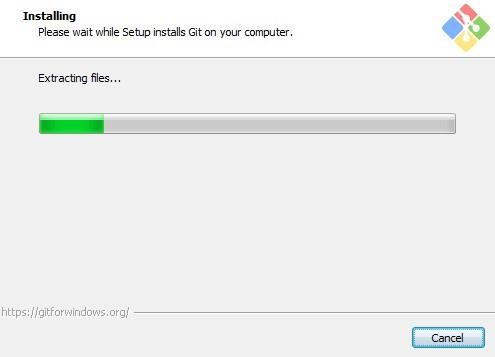
-



**Step 6 =>**

The Git is getting download in your system

The page looks like as: -



**Step 7 =>**

You can check that Git is install by simply type git - -version in

The page looks like as: -

GIT is finally installed on your desktop.

****

4. Setting Up GitHub Account

**Step 1 =>**

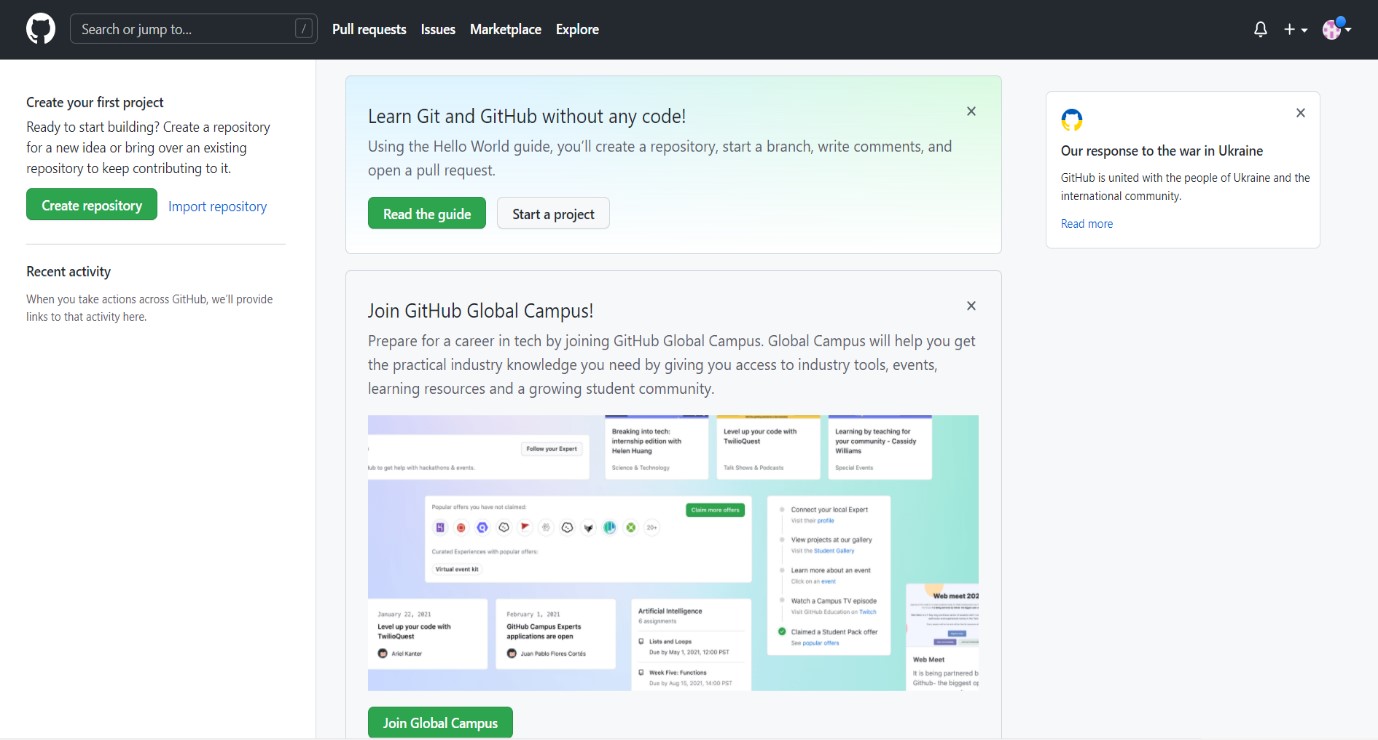
To set up your GitHub account you need to visit <https://github.com/>and click sign-up.

**Step 2 =>**

Enter your email, username and desired password.



Your account is created



5. Configuration Of GIT

**Step 1 =>**

1. config --global user.email "Your Email”
2. Set your username: git config --global user.name "Your Name"

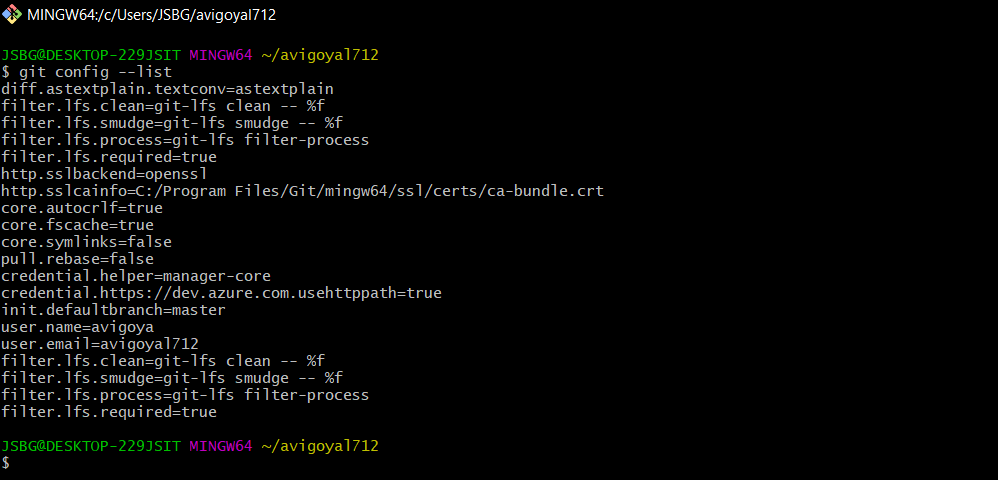


**Step 2 =>**

You can check configuration of Git by typing -

1. git config --list

The page looks like as: -



6. Program to Generate logs

Advantage of version control systems like git is that it can record changes.

‘Git log’ command is used to display all these changes that were made by the user in the repository. Log is a record of all the previous commits.

To understand Logs, we need to get familiar with all the commands that are used in making changes to a repository.

1. Repository: A repository is a directory that contains all the

project-related data.

1. Git init: The git init command is used to create a new blank repository.
2. Git status: We can list all the untracked, modified and deleted files using the git status command.
3. Git add: Adds all the untracked and modified files to the staging area.
4. Git commit: Git commit finalizes the changes in the repository. Every commit is saved in the branch you are working in and can be used to revert back to older versions of the project.

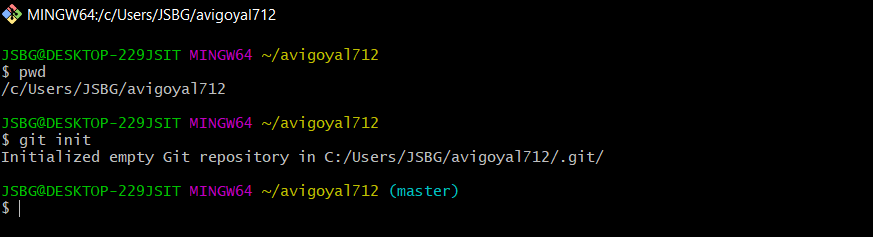
**Making GIT Repository**

**Step1: GIT INIT**

Initializing a new repository, You Can do it by typing: -

1. git init

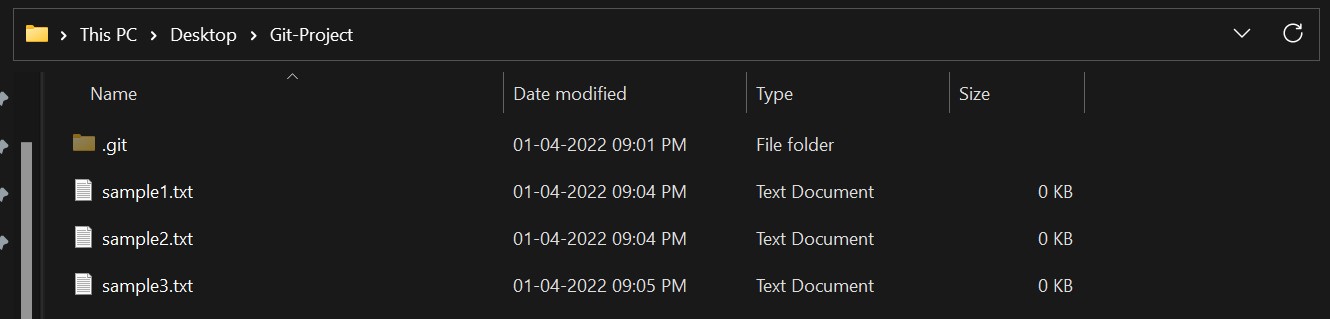
The page looks like as: -



**Step2: ADDING THE FILES TO THE FOLDER**

Just like (Samples...)

The page looks like as: -



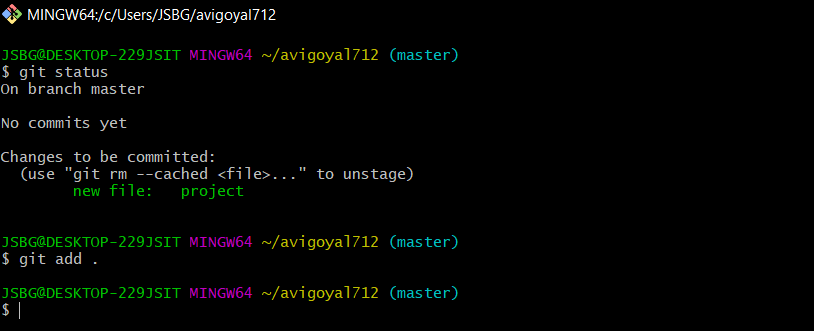
**Step3: GIT ADD**

The git add command adds a change in the working directory to the staging area.

You Can do it by typing: - 1. git add .

2. git add (current file name)

The page looks like as: -

****

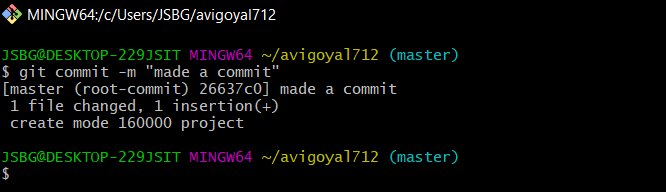
**Step4: GIT COMMIT**

The "commit" command is used to save your changes to the local repository.

You Can do it by typing: -

1. git commit -m”any text”

The page looks like as: -



**Step5: GIT LOG**

Git log will show all the commits made by the author with their time.

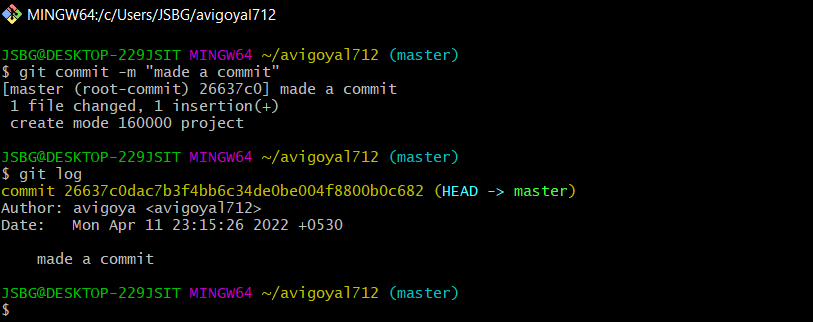
After every commit the checksum value (written In yellow color) of the folder changes.

Checksum is used to verify that the data in that file has not been tampered with or manipulated, possibly by a malicious entity.

You Can do it by typing: -

1. git log

The page looks like as: -

****

7. Create and Visualize Branches

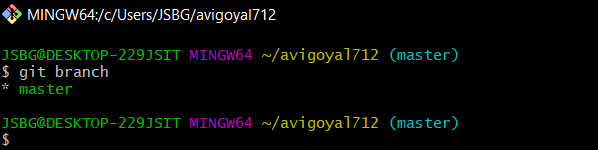
A branch in Git is simply a lightweight movable pointer to one of these commits. The default branch name in Git is master.

**Step1: CHECKING UP THE BRANCHES**

You can check which branch you are working in by using the command

1. ‘git branch’

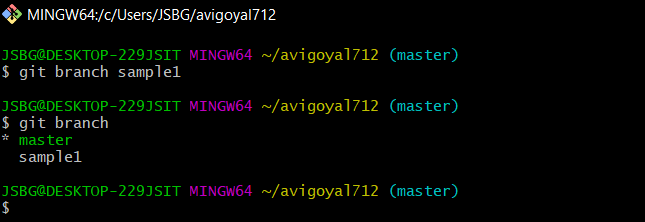
The default branch is always the master branch. The page looks like as: -

****

**Step2: CHECKING MULTIPLE BRANCHES**

You Can do it by typing: -

1. git branch (BRANCH NAME) The page looks like as: -



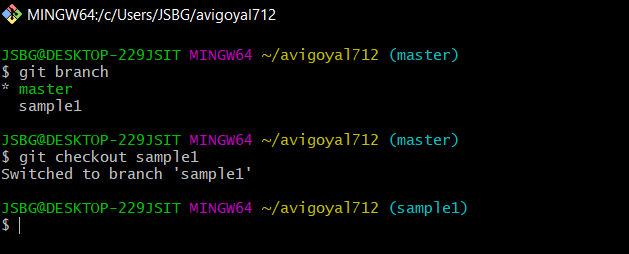
**Step3: CHANGING BRANCHES**

To switch to the other branch

You Can do it by typing: -

1. git checkout (BRANCH NAME)

your page look like as

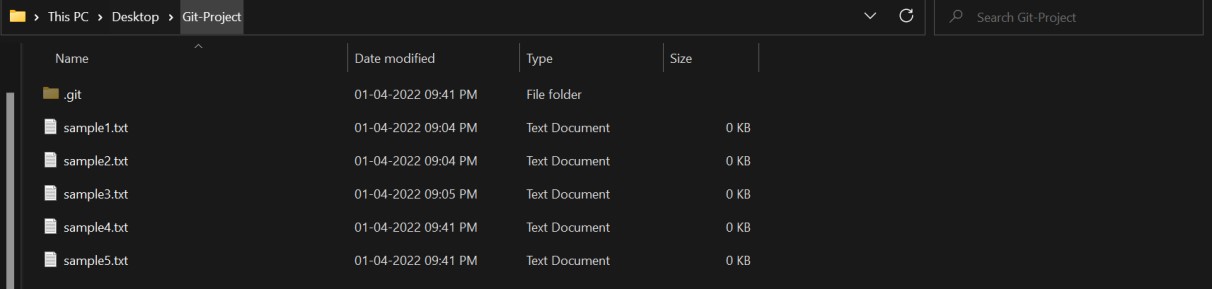


**Step4: NOW ADD FILE TO THE NEW BRANCH AND COMMIT IT**

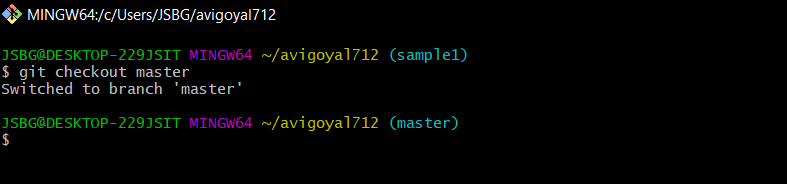
The page looks like

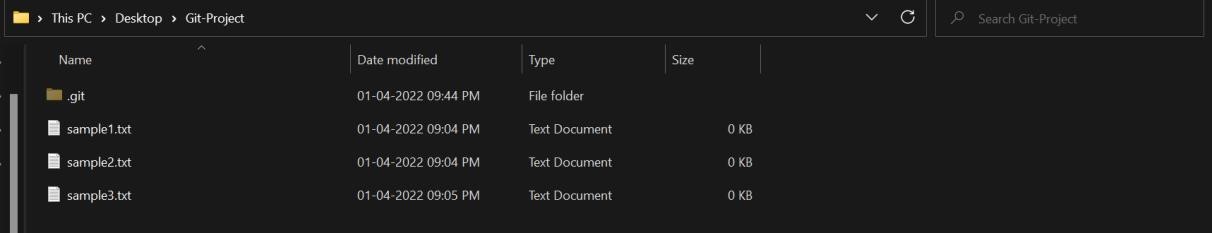
as:

-



**Step5: NOW SWITCH TO BRANCH AND CHECK FILE**





Now you can see that there is no file named HelloWorld.txt in the master branch because we created the file in the sample1 branch. So, it will be exclusive to the feature1 branch.

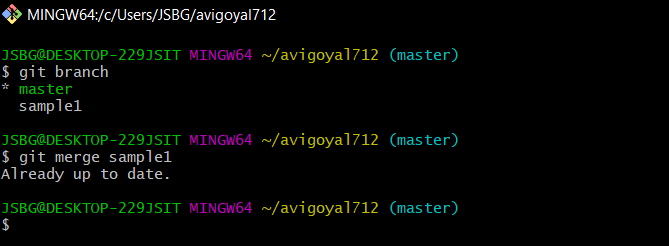
**Step6: GIT MERGING**

Now you can merge two branches by command.

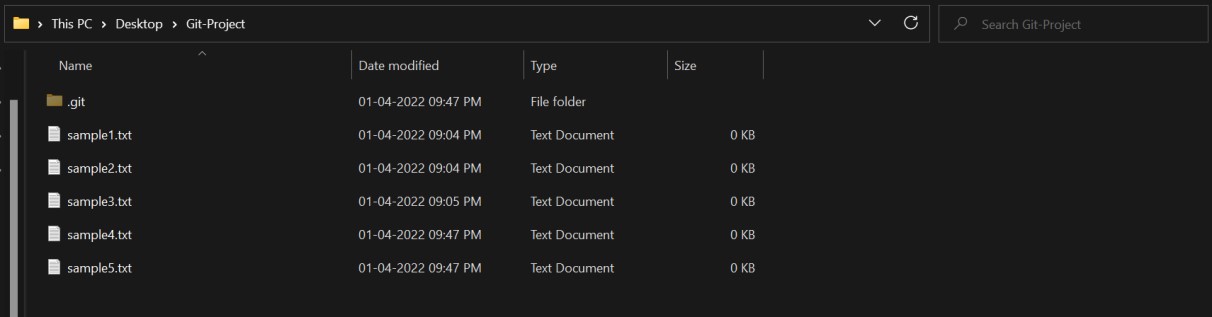
1. git merge (BRANCH NAME)

If you want to merge a new branch in master branch you need to first checkout into the master branch and then run the command.

The page looks like as: -



Now you can check the files in the master branch.

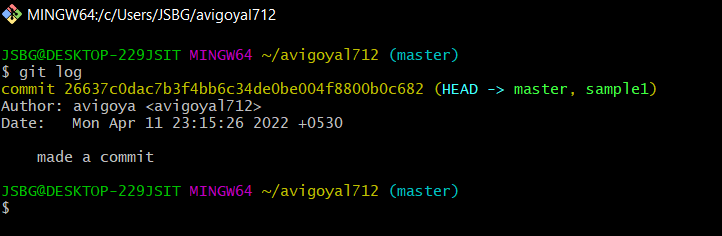


As you can see the sample4.txt and sample5.txt files are added into the master branch.

**Step7: RUNNING GIT LOG**

By running git log command on the master branch you can see all the commits made in master as well as the sample1 branch.

The page looks like as: -



8. Git Lifecycle Description

There are three stages for git lifecycle:

1. Working directory
2. Staging area
3. Git repository

**Working Directory:**

The working directory is the folder in your local computer where the project files and folders are stored.

The local directory is created by the command ‘git init’ which creates a ‘.git’ named folder which is used to track the files in the directory.

‘.git folder’ is generally hidden but can be tracked enabling hidden files.



**Staging area:**

The staging area has those files which are supposed to go to the next commit. Only those files which are needed to go to the next commit stay in the staging area.

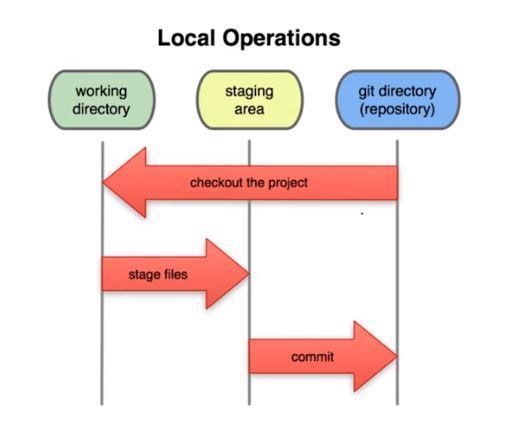
You can shift the files to the git repository by using the command

‘git add --a’.

**Git repository:**

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 commitcommand. 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.

You can commit files by using command ‘git commit -m “message”’



9. Uploading Data on GitHub

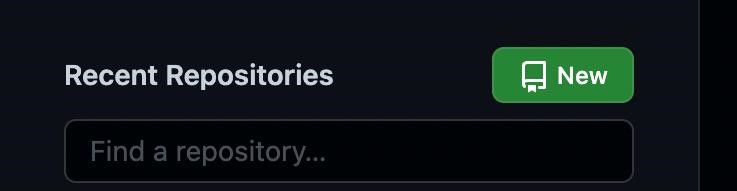
NOTE-

YOU HAVE TO MAKE A REPOSITORY IN GITHUB.

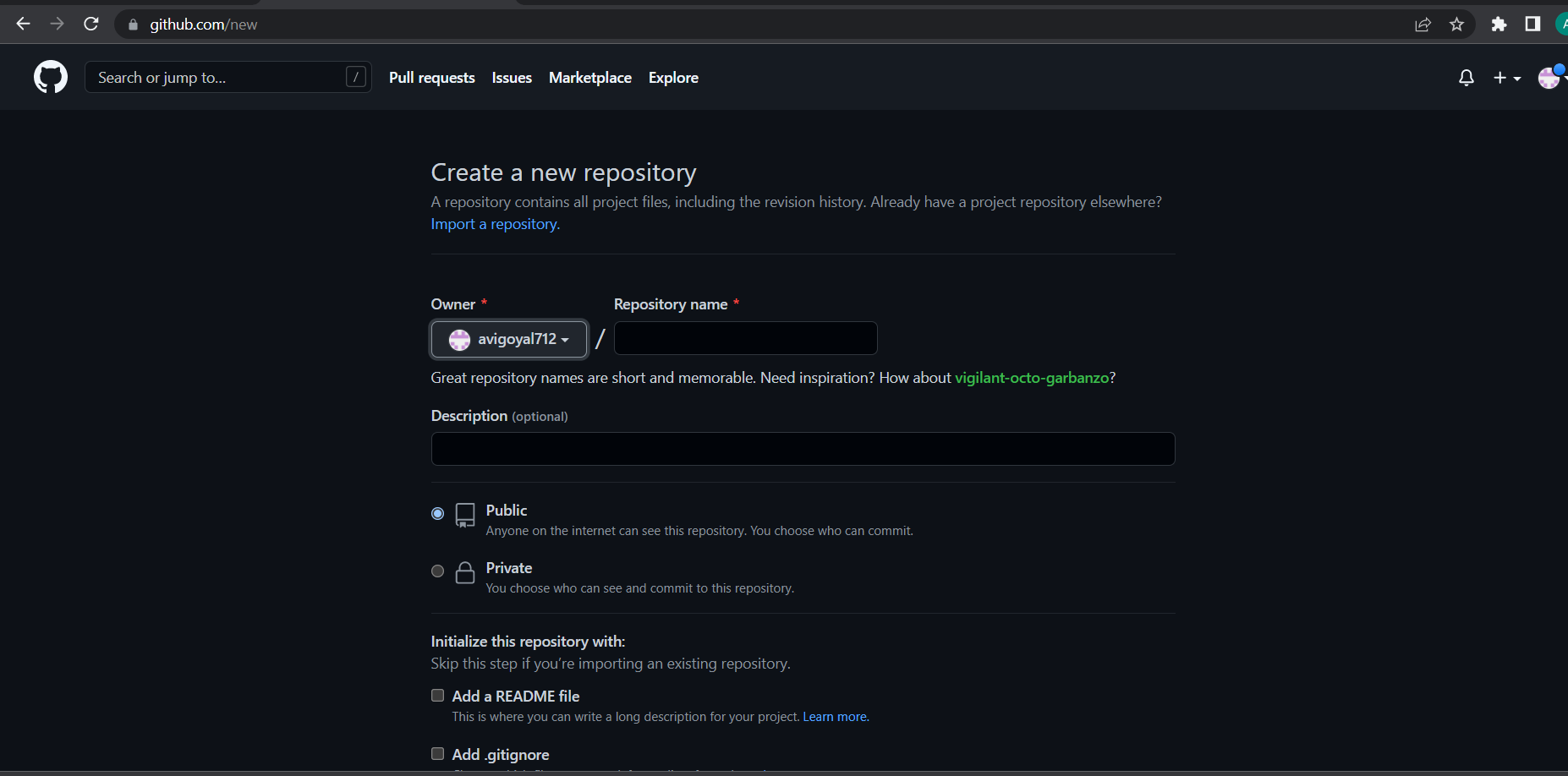
**Step1) CREATING REPOSITORY IN GITHUB**

The page looks like as:

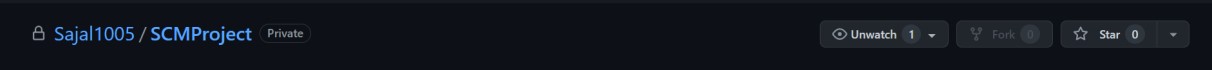
-



By clicking on new you are able to make a new repository.



Write the repository name and click on next.



Your GITHUB Repository has been created.

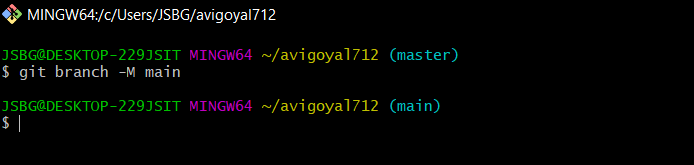
**Step2) GIT ADDING REMOTE BRANCH**

Git stores a branch as a reference to a commit, and a branch represents the tip of a series of commits.

You Can do it by typing: -

1. git branch -M main

The page looks like as: -



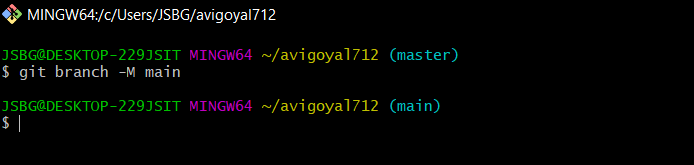
**Step3) GIT ADDING REMOTE ORIGIN**

Is a Git repository that's hosted on the Internet

You Can do it by typing: -

1. git remote add origin (URL)

The page looks like as: -



**Step4) GIT PUSHING**

The git push command is used to upload local repository content to a remote repository.

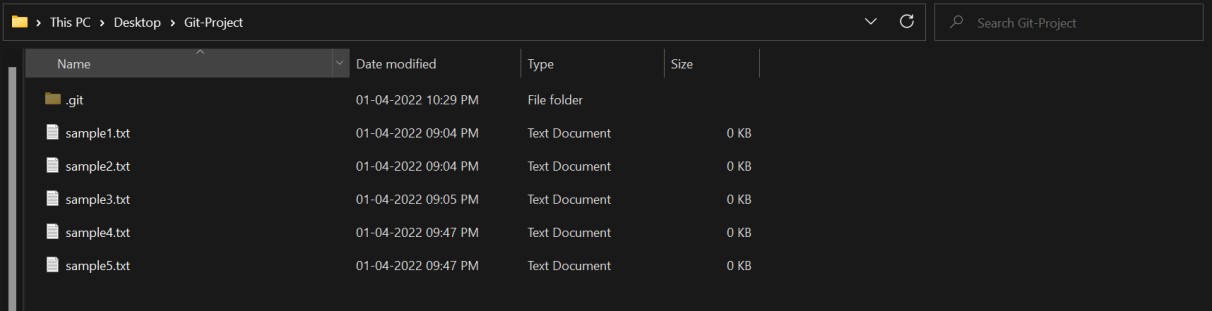
You Can do it by typing: -

1. git push -u origin main

**Final Result**

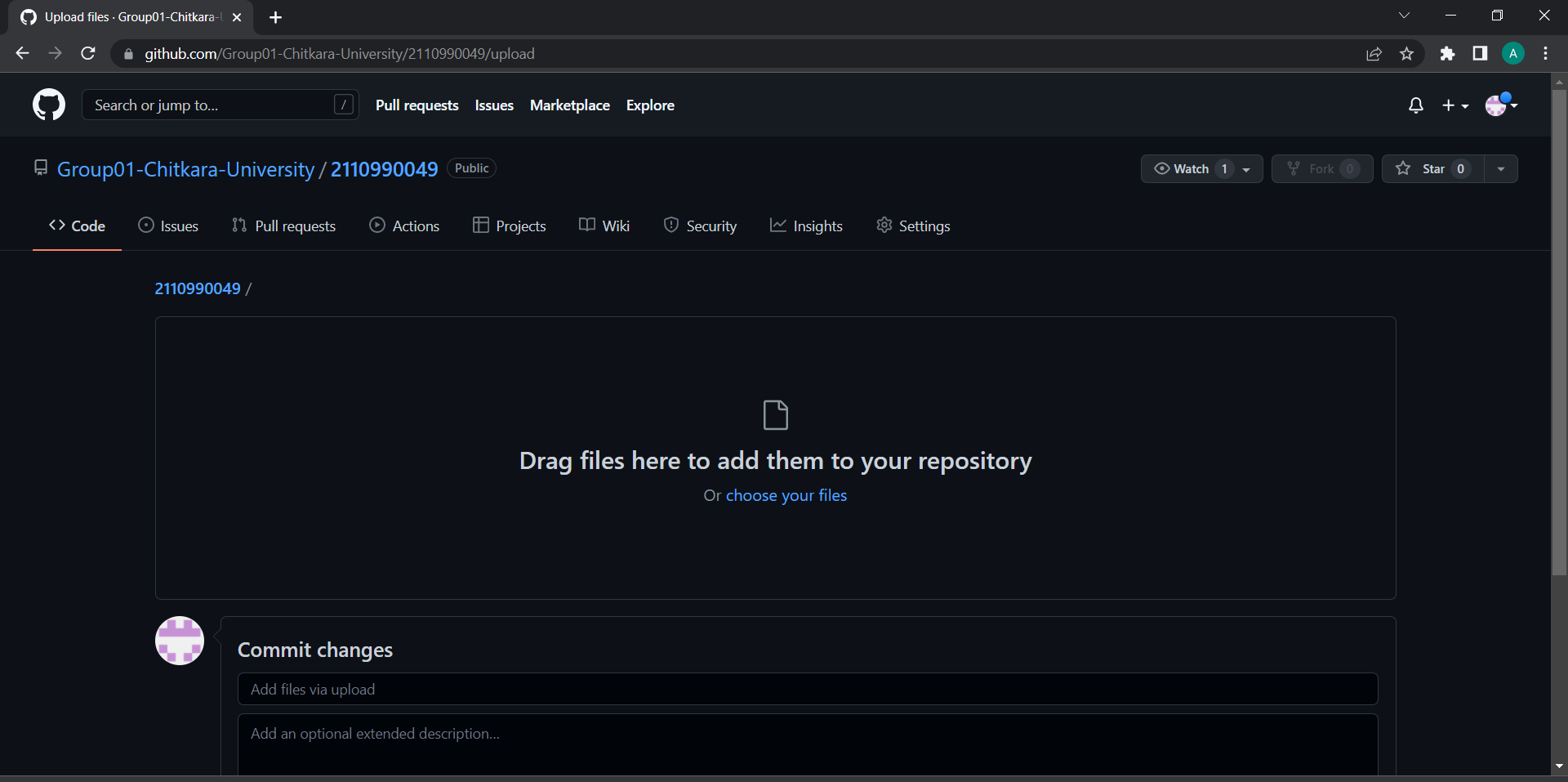
1. Document in your system

The page looks like as: -



1. Document in your GITHUB Repository

The page looks like as: -





Subject Name: **Source Code Management**

Subject Code: **CS181** Cluster: **Beta** Department: **DCSE**



**Submitted By: Submitted To:**

Abhinav Goyal Dr. Monit Kapoor

2110990049

G-1



**INDEX**

|  |  |  |
| --- | --- | --- |
| S No. | Topics | Page No. |
| 1. | Add Collaborators To Git Hub Repository | 3-4 |
| 2. | Fork And Commit | 5-7 |
| 3. | Merge And Resolve Conflicts | 8-11 |
| 4. | Reset And Revert | 12-18 |

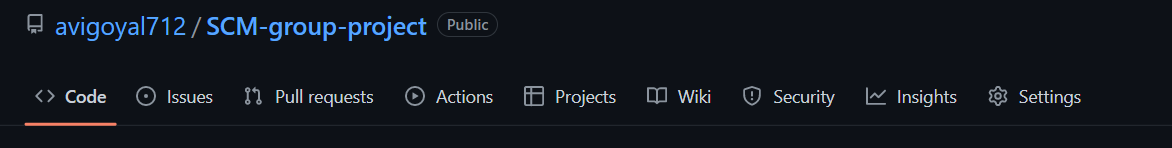


**Experiment No. 01**

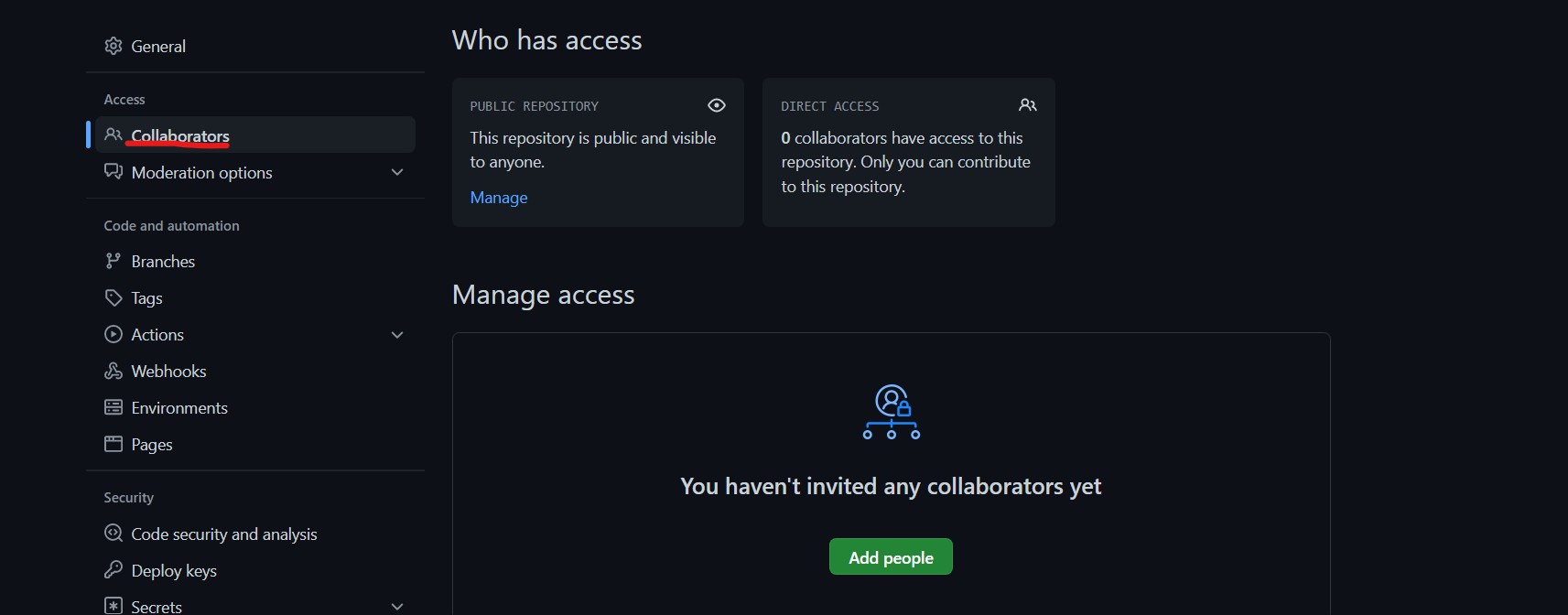
**AIM:** Add collaborators on GitHub Repository

**PROCEDURE:**

1. Click on the **Settings** tab in the right corner of the GitHub page.



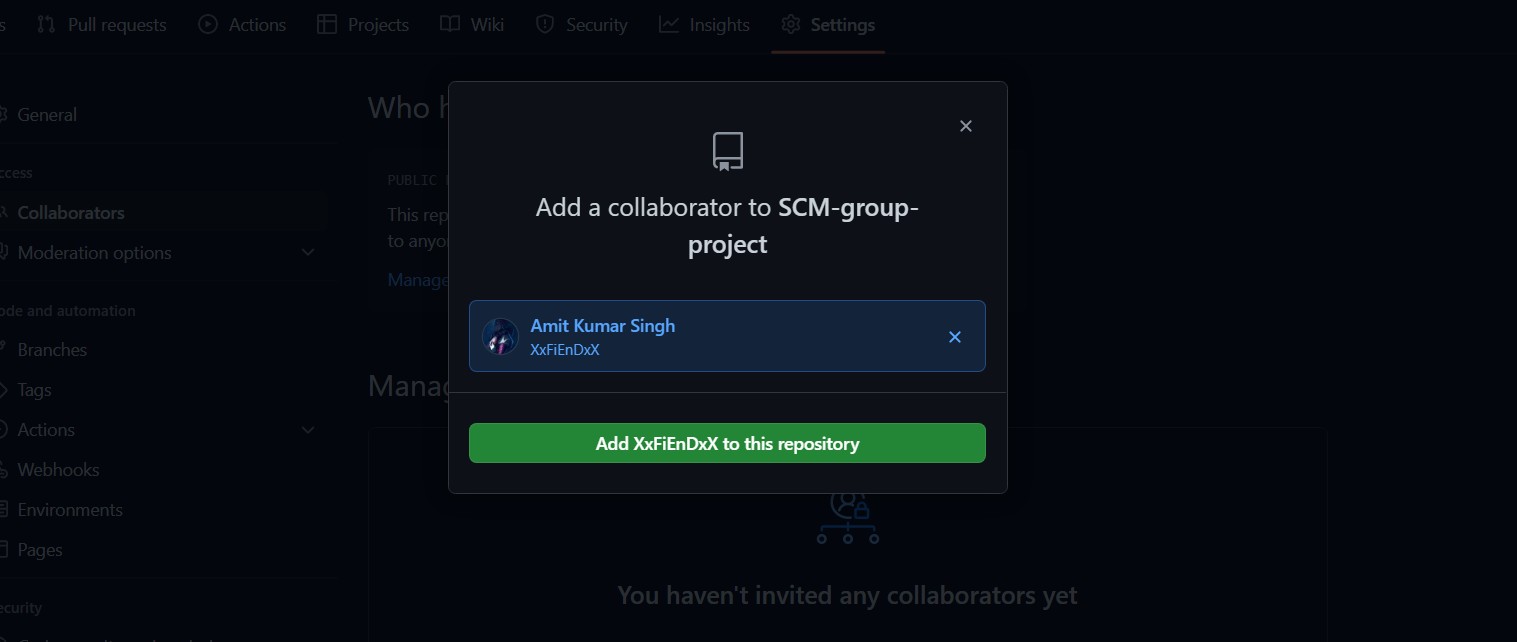
1. Go to **Manage Access** option under the Settings tab. On the Manage Access page, you will see an **Invite collaborator link** as shown in the below diagram.



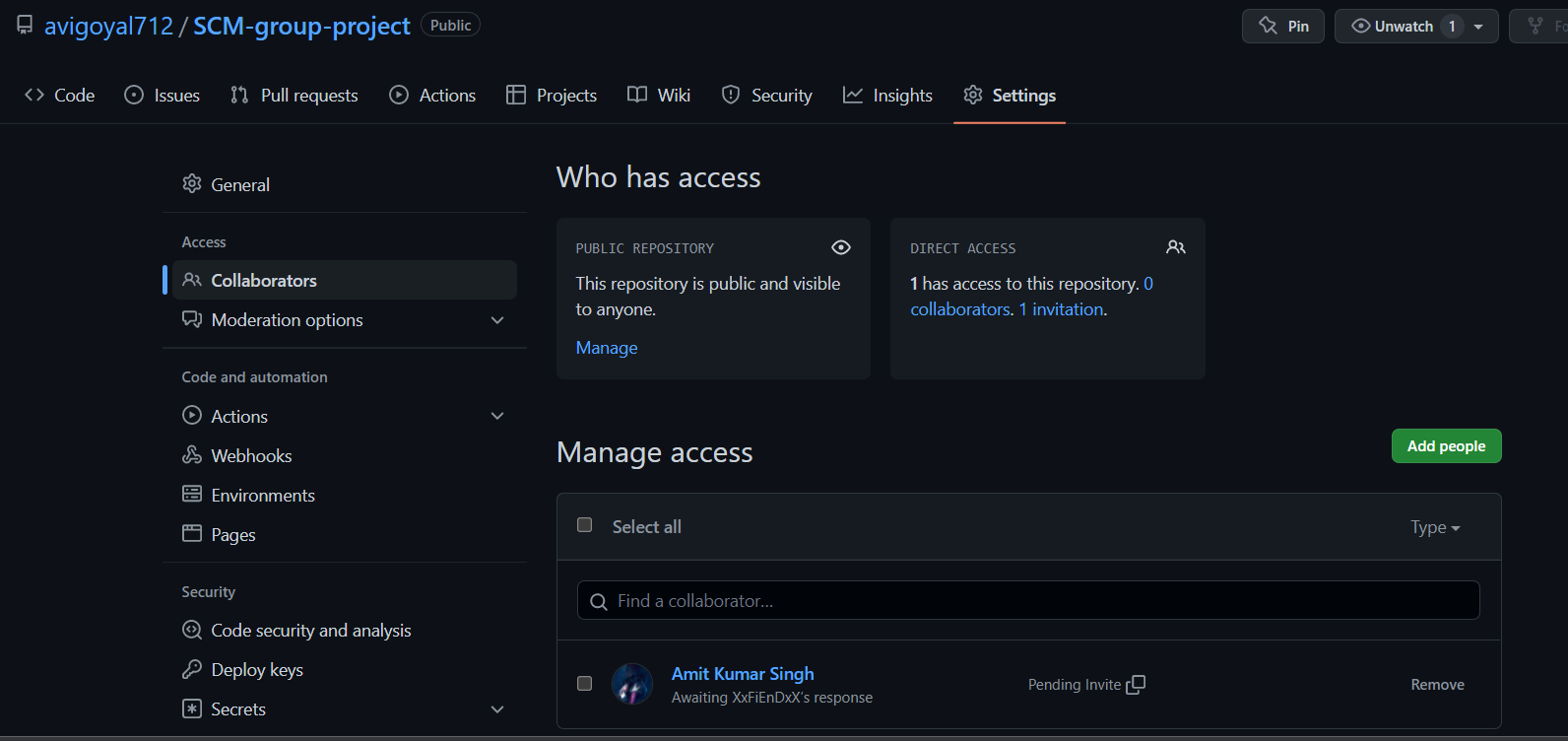
1. You can Invite collaborators by any of the following options −
   * **Username**
   * **Full name**
   * **Email**

After you send the invite, the collaborator receives an email invitation. The collaborator must accept it to get permission to collaborate on thesame project.





1. The **Manage Access** option also allows a repository owner to view the invitations that are pending and not accepted.





**Experiment No. 02**

**AIM:** Fork and Commit

**THEORY:**

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

When a user forks a repository, all the files in the repository are automatically copied to the user's GitHub account. The user is then free to use this repository either for their purpose or experiment with changes in the code. This does not affect the original repository.

Forking a repository on GitHub is done for two main purposes:

* **Improving someone's code/software**: Improving someone's code means fixing the bugs, improving the execution time and adding new features.
* **Reusing the code in a project**: A user can also make use of git fork to fork the repository of another user to use in their own project.



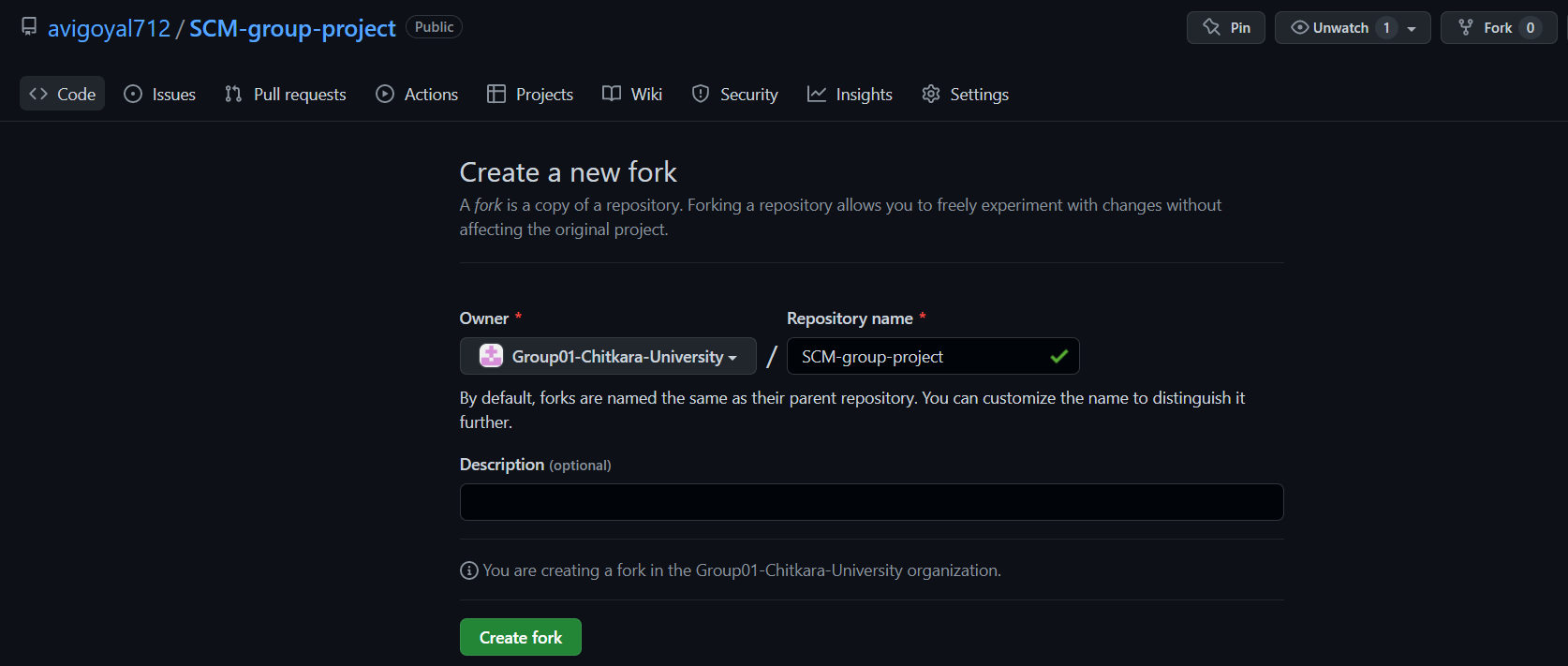
Other people are allowed to use that open-source code to their project, which helps them to save their efforts and time.

**PROCEDURE:**

1. To fork a project, visit the project page and click the **Fork** button at the top-right of the page. After a few seconds, you’ll be taken to your new project page, with your own writeable copy of the code.

**STEPS:**

1. Fork the project.

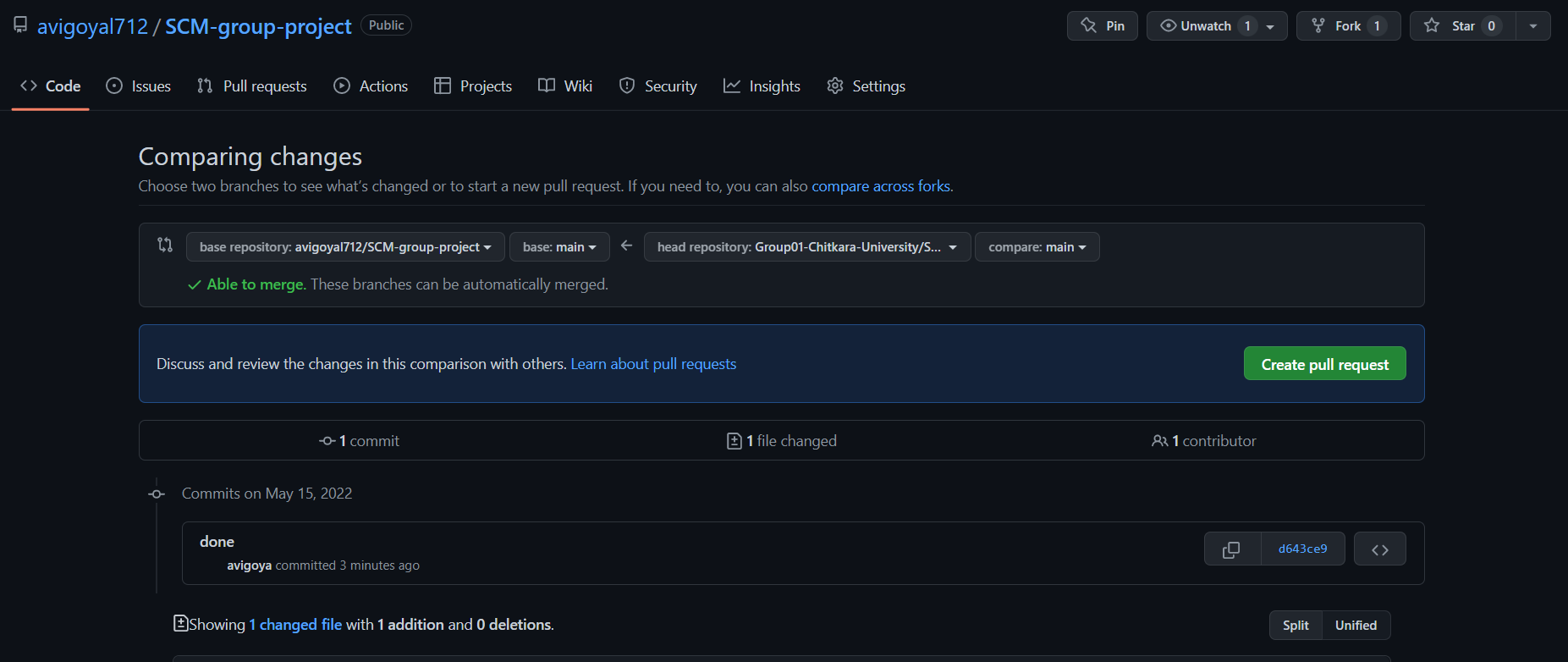


1. Create a topic branch from master.
2. Make some commits to improve the project.
3. Push this branch to your GitHub project.





1. Open a Pull Request on GitHub.



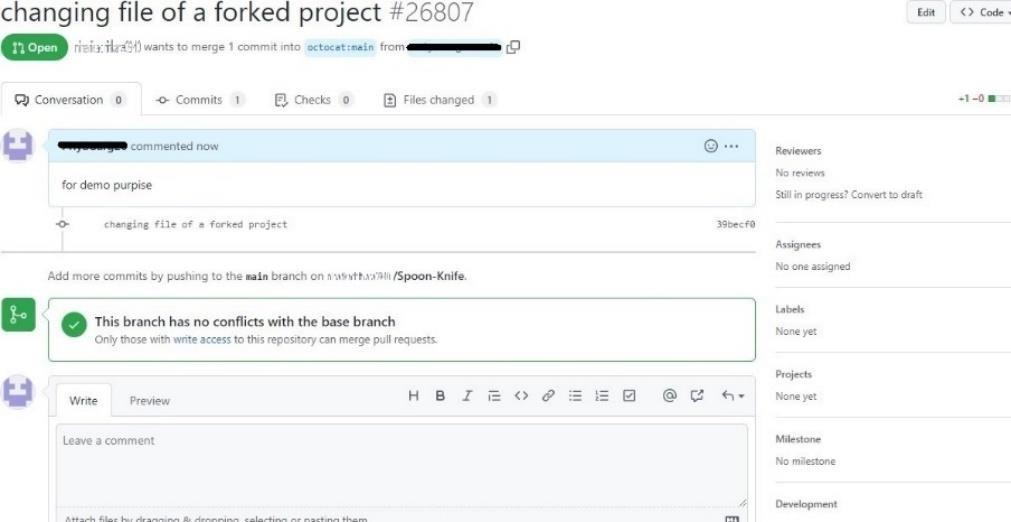


1. Discuss, and optionally continue committing.





1. The project owner merges or closes the Pull Request.



**Experiment No. 0 3 AIM:** Merge and Resolve Conflicts

**THEORY**: A merge conflict is an event that takes place when Git is unable to resolve differences in code between two commits. Git can merge the changes automatically only if the commits are on different lines or branches.

Let’s assume there are two developers: Developer A and B. Both of them pull the same code file from the remote repository and try to make various amendments in that file. After making the changes, Developer A pushes the file back to the remote repository from his local repository. Now, when Developer B tries to push that file after making the changes from his end, he is unable to do so, as the file has already been changed in the remote repository.

To prevent such conflicts, developers work in separate isolated branches. The Git merge command combines separate branches and resolves any conflicting edits.

There are two points when a merge can enter a conflicted state:

1. **Starting the Merge Process**

If there are changes in the working directory’s stage area for the current project, merging won’t start. In this case, conflicts happen due to pending changes that need to be stabilized using different Git commands.

1. **During the Merge Process**

The failure during the merge process indicates that there is a conflict between the local branch and the branch being merged. In this case, Git resolves as much as possible, but there are things that have to be resolved manually in the conflicted files.

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.

The commands listed below are used to resolve conflicts. **git log --merge**

This command helps to produce the list of commits that are causing the conflict **git diff**

It helps to identify the differences between the state's repositories or files **git checkout**

This command is used to undo the changes made to the file, or for changing branches **git reset --mixed**

This command is used to undo changes to the working directory and staging area **git merge --abort**

It helps in exiting the merge process and returning back to the state before the merging began



**git reset**

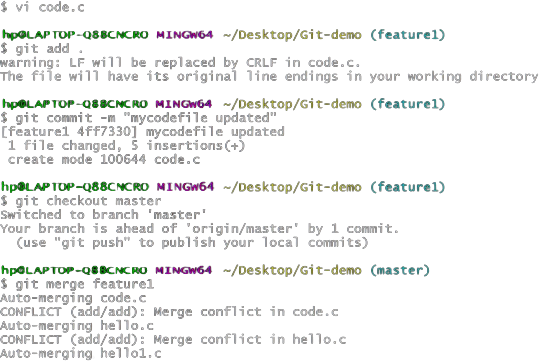
This command is used at the time of merge conflict to reset the conflicted files to their original state.

**PROCEDURE:**

1. Open git bash and switch to master branch.
2. In master branch create a file name “Code.c” by [vi code.c]. It opens an editor and by pressing Insert key write a code there. Then exit it by pressing Esc key. Add this file and commit it.

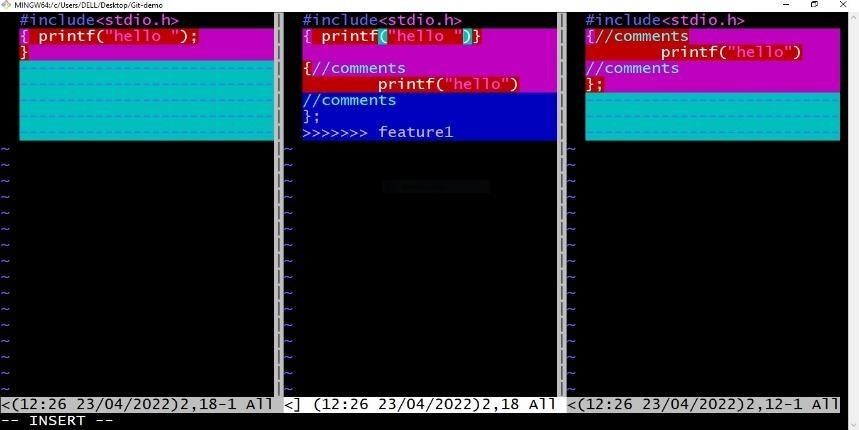


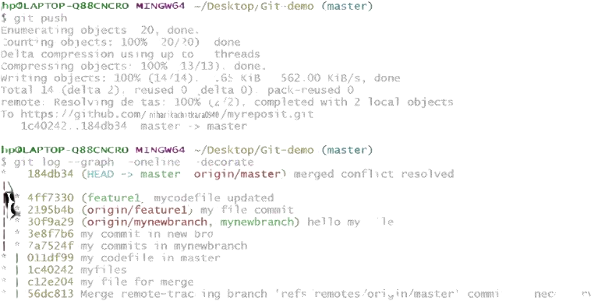
1. Do same steps in other branch but don’t forget to add extra lines of code in that file.
2. Run **git checkout master** and go back to previous branch.
3. Then run **git merge feature1**. It will create a merge conflict.



1. After running **git mergetool** command, all files will be processed. You can see three different files there, and you can see everything that was added or removed.





1. After scrolling, you can verify where exactly the conflict happened.
2. You can remove manually lines in editor.
3. Manual modifications have allowed us to resolve file conflicts. Save the file and close the final file. Commit and add this file. After this use push command to push all files to remote repository.

We can also see the graph of our all commits in Git Client **git log --graph –oneline – decorate.**





We can also see the graph on our GitHub account by visiting insights and then network.



**Experiment No. 04**

**AIM:** Reset and Revert

The term Git Reset stands for undoing changes. This command is used to reset changes. This command has three core forms of invocation which are:

* + Soft
  + Mixed
  + Hard

In terms of Git, Git Reset is a tool that resets the current state of HEAD to a specified state. It is a sophisticated and versatile tool for undoing changes. It acts as a time machine for Git. You can jump up and forth between the various commits. Each of these reset variations affects specific trees that git uses to handle your file in its content.

Additionally, git reset can operate on whole commits objects or at an individual file level. Each of these reset variations affects specific trees that git uses to handle your file and its contents.

Git uses an index (staging area), HEAD, and working directory for creating and reverting commits.

Git Reset Hard will first move the Head and update the index with the contents of the commits. It is the most direct, unsafe, and frequently used option. The --hard option changes the Commit History, and ref pointers are updated to the specified commit. Then, the Staging Index and Working Directory need to reset to match that of the specified commit. Any previously pending commits to the Staging Index and the Working Directory get reset to match Commit Tree. It means any awaiting work will be lost.

I have checked the status of the repository. We can see that the current head position yet not changed because I have not committed the changes. Now, I am going to perform the **reset --hard** option. The git reset hard command will be performed as:



**$ git reset --hard**



Generally, the reset hard mode performs below operations:

* It will move the HEAD pointer.
* It will update the staging Area with the content that the HEAD is pointing.
* It will update the working directory to match the Staging Area.

Git Reset Mixed: It is a default option of the git reset command. If we do not pass any argument, then the git reset command considers **--mixed** as default option. A mixed option updates the ref pointers. The staging area also reset to the state of a specified commit. The undone changes transferred to the working directory. Let's understand it with an example.

In the above output, I have added a **newfile2.txt** to my local repository. Now, we will perform the reset mixed command on this repository. It will operate as:

**$ git reset --mixed**



Or we can use only git reset command instead of this command.

**$ git reset**

The above command will reset the status of the Head, and it will not delete any data from the staging area to match the position of the Head. Consider the below output:



From the above output, we can see that we have reset the position of the Head by performing the git reset -mixed command. Also, we have checked the status of the repository. As we can see that the status of the repository has not been changed by this command. So, it is clear that the mixed-mode does not clear any data from the staging area.

Generally, the reset mixed mode performs the below operations:

* It will move the HEAD pointer
* It will update the Staging Area with the content that the HEAD is pointing to

It will not update the working directory as git hard mode does. It will only reset the index but not the working tree, then it generates the report of the files which have not been updated.

If -N is specified on the command line, then the statements will be considered as intentto- add by Git.

Git Reset Head (Git Reset Soft): The soft option does not touch the index file or working tree at all, but it resets the Head as all options do. When the soft mode runs, the refs' pointers updated, and the resets stop there. It will act as git amend command. It is not an authoritative command. Sometimes developers considered it as a waste of time.



Generally, it is used to change the position of the Head.

**$ git reset--soft <commit-sha>**

The above command will move the HEAD to the particular commit.

I have made changes in my file newfile2.txt and commit it. So, the current position of Head is shifted on the latest commit. To check the status of Head, run the below command:

**$ git log**



From the above output, you can see that the current position of the HEAD is on f05c07719abc2fba352db852f25012286476aef7 commit. But I want to switch it on my older commit 6e56ad4515cdaaa854a4df525bcc956ad539c113. Since the commit-sha number is a unique number that is provided by sha algorithm. To switch the HEAD, run the below command:

**$ git reset --soft 6e56ad4515cdaaa854a4df525bcc956ad539c113**

The above command will shift my HEAD to a particular commit.





Git Revert

In Git, the term revert is used to revert some changes. The git revert command is used to apply revert operation. It is an undo type command. However, it is not a traditional undo alternative. It does not delete any data in this process; instead, it will create a new change with the opposite effect and thereby undo the specified commit. Generally, git revert is a commit. It can be useful for tracking bugs in the project. If you want to remove something from history then git revert is a wrong choice. Moreover, we can say that git revert records some new changes that are just opposite to previously made commits. To undo the changes, run the below command.

**$ git revert**

Git Revert Options:

Git revert allows some additional operations like editing, no editing, cleanup, and more. Let's understand these options briefly:

**< commit>:** The commit option is used to revert a commit. To revert a commit, we need the commit reference id. The git log command can access it.

**$ git revert <commit-ish>**

**<--edit>:** It is used to edit the commit message before reverting the commit. It is a default option in git revert command.

**$ git revert -e <commit-ish>**

**-m parent-number /--mainline parent-number:** it is used to revert the merging. Generally, we cannot revert a merge because we do not know which side of the merge should be considered as the mainline. We can specify the parent number and allows revert to reverse the change relative to the specified parent.

**-n/--no edit:** This option will not open a text editor. It will directly revert the last commit.

**$ git revert -n <commit-ish>**

**--cleanup=<mode>:** The cleanup option determines how to strip spaces and comments from the message.

**-n/--no-commit:** Generally, the revert command commits by default. The no- commit option will not automatically commit. In addition, if this option is used, your index does not have to match the HEAD commit.



The no-commit option is beneficial for reverting more than one commits effect to your index in a row.

Let's understand how to revert the previous commits.

Git Revert to Previous Commit

Suppose you have made a change to a file say **newfile2.txt** of your project. And later, you remind that you have made a wrong commit in the wrong file or wrong branch.

Now, you want to undo the changes you can do so. Git allows you to correct your mistakes. Consider the below image:



As you can see from the above output that I have made changes in newfile2.txt. We can undo it by git revert command. To undo the changes, we will need the commit-ish. To check the commit-ish, run the below command:

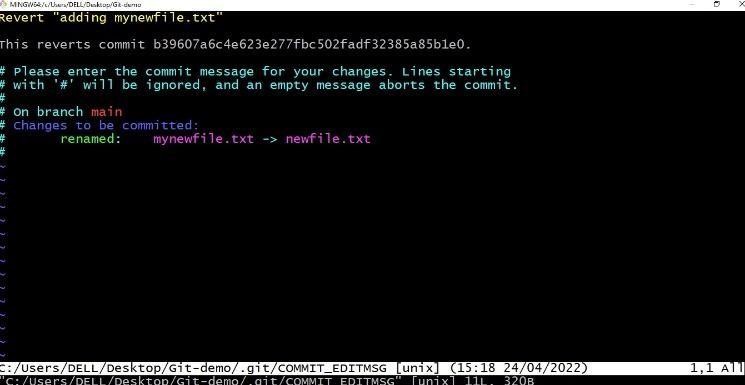
**$ git log**

Consider the below output:

In the above output, I have copied the most recent commit-ish to revert. Now, I will perform the revert operation on this commit. It will operate as: **$ git revert 099a8b4c8d92f4e4f1ecb5d52e09906747420814**

The above command will revert my last commit. Consider the below output





As you can see from the above output, the changes made on the repository have been reverted.

Git Revert Merge: In Git, merging is also a commit that has at least two parents. It connects branches and code to create a complete project. A merge in Git is a commit that has at least two parents. It brings together multiple lines of development. In a work-flow where features are developed in branches and then merged into a mainline, the merge commits would typically have two parents.

How to Revert a Merge:

Usually, reverting a merge considered a complicated process. It can be complex if not done correctly. We are going to undo a merge operation with the help of git revert command. Although some other commands like git reset can do it. Let's understand how to revert a merge. Consider the below example.

I have made some changes to my file newfile.txt on the test and merge it with **test2**. Consider the below output:

To revert a merge, we have to get its reference number. To check commit history, run the below command:



**$ git log**

The above command will display the commit history. Consider the below output:

From the above output, copy your merging commit that you to want to revert and run the below command:

**$ git revert <commit reference> -m 1**

A Project report

on

**“GitHub Web Hosting”**

with

**Source Code Management**

(CS181)

Submitted by

Abhinav Goyal- Roll No.2110990049



**Department of Computer Science & Engineering**

Chitkara University Institute of Engineering and Technology, Punjab

Jan- June   
(2021-22)

|  |  |  |  |
| --- | --- | --- | --- |
| Institute/School Name | **Chitkara University Institute of Engineering and Technology** | | |
| Department Name | **Department of Computer Science & Engineering** | | |
| Programme Name | **Bachelor of Engineering (B.E.), Computer Science & Engineering** | | |
| Course Name | **Source Code Management** | Session | **2021-22** |
| Course Code | **CS181** | Semester/Batch | **2nd/2021** |
| Vertical Name | **Beta** | Group No | G-1 |
| Faculty Name | **Dr. Monit Kapoor** | | |

Submission to Dr. Monit kapoor

Name: Abhinav Goyal

Signature:

Date:

3/6/22

**Table of Content**

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Title** | **Page No.** |
| 1 | Version control with Git | 1-6 |
| 2 | Problem Statement | 7 |
| 3 | Objective | 8 |
| 4 | Concepts and commands | 9-16 |
| 5 | Workflow and Discussion | 17-20 |
| 6 | Reference | 21 |

**1. Version control with Git**

* **What is Version Control System?**

A **Version Control System (VCS)** is a tool that helps software developers keep track of how their software development projects - desktop applications, websites, mobile apps, etc - change over time.

Each snapshot or state of the files and folders in a codebase at a given time can be called a "version." Version control systems were created to allow developers a convenient way to create, manage, and share those versions. It allows them to have *control* over managing the versions of their code as it evolves over time.

Version control systems also enable collaboration within a team of software developers, without losing or overwriting anyone's work. After a developer makes a set of code changes to one or more files, they tell the version control system to save a representation of those changes.

A version control system can also be referred to as a source control system, source code management, version control software, version control tools, or other combinations of these terms.

* **Benefits of the Version Control System 🡪**

The Version Control System is very helpful and beneficial in software development; developing software without using version control is unsafe. It provides backups for uncertainty. Version control systems offer a speedy interface to developers. It also allows software teams to preserve efficiency and agility according to the team scales to include more developers.

Some key benefits of having a version control system are as follows.

* Complete change history of the file
* Simultaneously working
* Branching and merging
* Traceability
* **History of VCS 🡪**

There are 3 types of VCS:

1. **Local Version Control System:**

In a local version control system, files are simply copied into a separate directory locally. Versions of the same file are stored so as to allow the easy retrieval of any particular version at any point in time. This system is commonly used for small personal projects or files as it provides the facility of versioning your project in an easy manner locally. 

**Advantages:**

* Easy to set up
* No internet needed
* Cheap to run

**Disadvantages:**

* Error prone
* Unsafe (stored locally)
* Not suitable for team projects
* As data is stored in local machine. If the local machine crashes, it would not be possible to retrieve the files, and all the information will be lost.

1. **Centralized Version Control System:**

In the Centralized Version Control Systems, there will be a single central server that contains all the files related to the project, and many collaborators checkout files from this single server (you will only have a working copy). The problem with the Centralized Version Control Systems is if the central server crashes, almost everything related to the project will be lost.

**Advantages:**

* Reasonably easy to set up
* Various options (proprietary and open source)
* Allows for file sharing amongst team members
* Project is stored on a more reliable server (possibly cloud)
* Admin can control the use and structure of the repository

**Disadvantages:**

* Single point of failure (if server fails then changes will not be available)
* File conflicts due to updates from different people

1. **Distributed Version Control system:**

**In a distributed version control system,  there will be one or more servers and many collaborators similar to the centralized system. But the difference is, not only do they check out the latest version, but each collaborator will have an exact copy of the main repository on their local machines.**

**Each user has their own repository and a working copy. This is very useful because even if the server crashes we would not lose everything as several copies are residing in several other computers.**

**Advantages:**

* Reliable (everyone has a copy of all versions)
* Allows for file share amongst team members
* Various Options available

**Disadvantages:**

* More complex to use/set up
* Heavy on Local Storage
* **What is Git?**

**Git** is an **open-source distributed version control system**. It is designed to handle minor to major projects with high speed and efficiency. It is developed to co-ordinate the work among the developers. The version control allows us to track and work together with our team members at the same workspace.

Git was created by **Linus Torvalds** in **2005** to develop Linux Kernel. It is also used as an important distributed version-control tool for **the DevOps**.

* **Features of Git**

Some remarkable features of Git are as follows:

* **Open Source**  
  Git is an open-source tool. It is released under the GPL (General Public License) license.
* **Scalable**  
  Git is scalable, which means when the number of users increases, the Git can easily handle such situations.
* **Distributed**  
  One of Git's great features is that it is distributed. Distributed means that instead of switching the project to another machine, we can create a "clone" of the entire repository. Also, instead of just having one central repository that you send changes to, every user has their own repository that contains the entire commit history of the project. We do not need to connect to the remote repository; the change is just stored on our local repository. If necessary, we can push these changes to a remote repository.
* **Security**  
  Git is secure. It uses theSHA1 (Secure Hash Function) to name and identify objects within its repository. Files and commits are checked and retrieved by its checksum at the time of checkout. It stores its history in such a way that the ID of particular commits depends upon the complete development history leading up to that commit. Once it is published, one cannot make changes to its old version.
* **Speed**  
  Git is very fast, so it can complete all the tasks in a while. Most of the git operations are done on the local repository, so it provides a hugespeed. Also, a centralized version control system continually communicates with a server somewhere.  
  Performance tests conducted by Mozilla showed that it was extremelyfast compared to other VCSs. Fetching version history from a locally stored repository is much faster than fetching it from the remote server. The core part of Gitis written in **C**, which ignores runtime overheads associated with other high-level languages.  
  Git was developed to work on the Linux kernel; therefore, it is capable enough to handle largerepositories effectively. From the beginning, speed and performance have been Git's primary goals.
* **Supports non-linear development**  
  Git supports seamless branching and merging, which helps in visualizing and navigating a non-linear development. A branch in Git represents a single commit. We can construct the full branch structure with the help of its parental commit.
* **Branching and Merging**  
  Branching and merging are the great feature**s** of Git, which makes it different from the other SCM tools. Git allows the creation of multiple branches without affecting each other. We can perform tasks like creation**,**deletion**,** and merging on branches, and these tasks take a few seconds only. Below are some features that can be achieved by branching:
  + We can create a separate branch for a new module of the project, commit and delete it whenever we want.
  + We can have a production branch, which always has what goes into production and can be merged for testing in the test branch.
  + We can create a demo branch for the experiment and check if it is working. We can also remove it if needed.
  + The core benefit of branching is if we want to push something to a remote repository, we do not have to push all of our branches. We can select a few of our branches, or all of them together.
* **Data Assurance**  
  The Git data model ensures the cryptographic integrity of every unit of our project. It provides a unique commit ID to every commit through a SHA algorithm. We can retrieve and update the commit by commit ID. Most of the centralized version control systems do not provide such integrity by default.
* **Staging Area**  
  The Staging area is also a unique functionality of Git. It can be considered as a preview of our next commit, moreover, an intermediate area where commits can be formatted and reviewed before completion. When you make a commit, Git takes changes that are in the staging area and make them as a new commit. We are allowed to add and remove changes from the staging area. The staging area can be considered as a place where Git stores the changes.  
  Although, Git doesn't have a dedicated staging directory where it can store some objects representing file changes (blobs). Instead of this, it uses a file called index.

Another feature of Git that makes it apart from other SCM tools is that itis possible to quickly stage some of our files and commit them without committing other modified files in our working directory.

* **Maintain the clean history**  
  Git facilitates with Git Rebase; It is one of the most helpful features of Git. It fetches the latest commits from the master branch and puts our code on top of that. Thus, it maintains a clean history of the project.

2. Problem Statement

We are going to host our Chitkara University Website over Git-hub So that we can control & have an access to track the latest versions of Website.

* Following points are the Objectives of Git:

3. Objective

* + - 1. **To Track Histories:** Git is used to tracking changes in the source code.
      2. **To do collaborations:** Git keeps track of changes to files and allows multiple users to coordinate updates to those files.
      3. **To do branching:** Git allow you to develop features, fix bugs, or safely experiment with new ideas in a contained area of your repository.
      4. **To do distributed development:** Git enables the developers to manage the changes offline and allows you to branch and merge whenever required, giving them full control over the local code base.

**4. Concepts and commands**

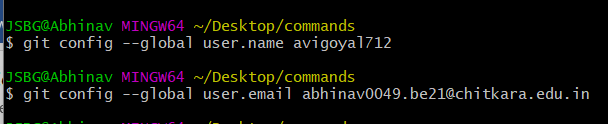
* **Here are some commands which is used in GIT.**

**git config --global user.name “name”** **:**

This command sets the author’s name to be used with your commits.

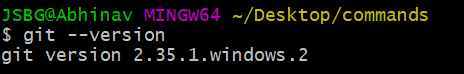
**git config --global user.email “email”** **:**

This command sets the author’s email address to be used with your commits.



**git --version :**

It is used to display the version of git bash.



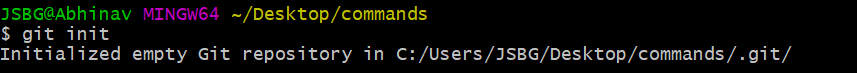
**pwd:**

Gives the full pathname of the current working directory to the standard output

****

**git init :**

Used to initialize the repository.



**touch “filename”:**

It creates a new file.

**git add --a** **:**

Moves changes from the working directory to the staging area.

**git commit -m “info of commit”** **:**

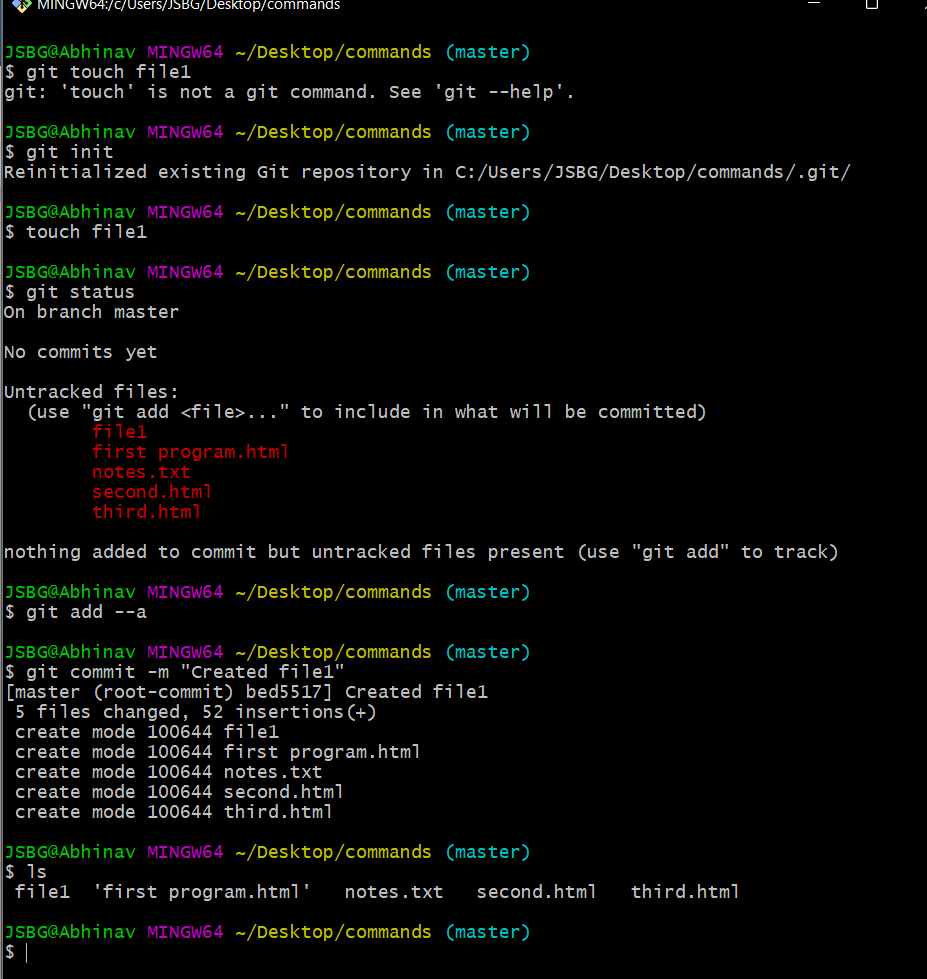
This command records or snapshots the file permanently in the version history.

**ls :**

It gives the name of file present in the current folder.

**git status:**

This command lists all the files that have to be committed.

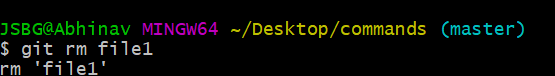
****

**rm -rf .git:**

This command is used delete the .git hidden folder.

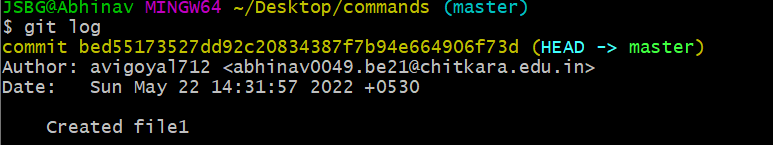
**git rm [file] :**

This command deletes the file from your working directory and stages the deletion.



**git log :**

This command is used to list the version history for the current branch.



**git branch:**

This command lists all the local branches in the current repository.

**git branch [branch name] :**

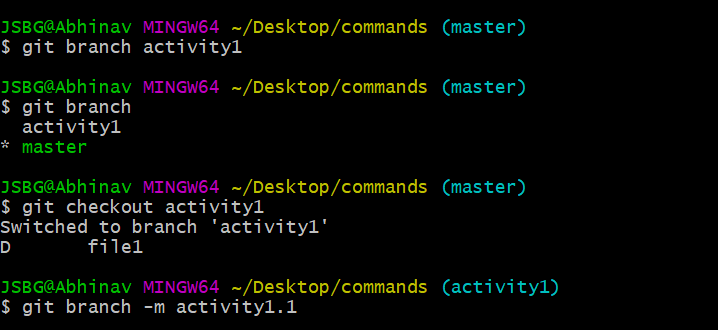
This command creates a new branch.

**git checkout [branch name] :**

This command is used to switch from one branch to another.

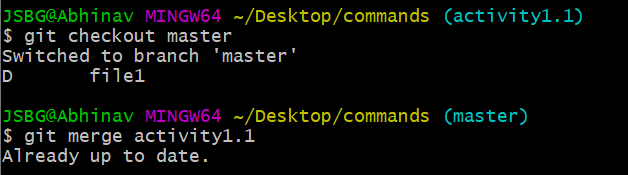
**git branch -m [branch name] :**

This command is used to rename the branch.

****

**git merge [branch name] :**

This command is used to merge a command.

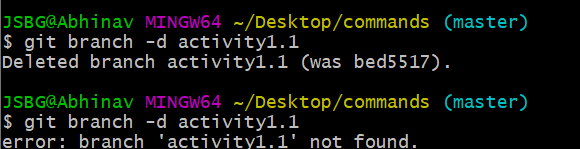


**git branch -d [branch name] :**

This command is used to soft delete a branch.

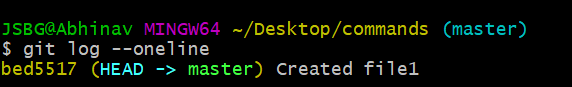
**git branch -D [branch name] :**

When you want to delete a branch without merging it. So, we use this command.



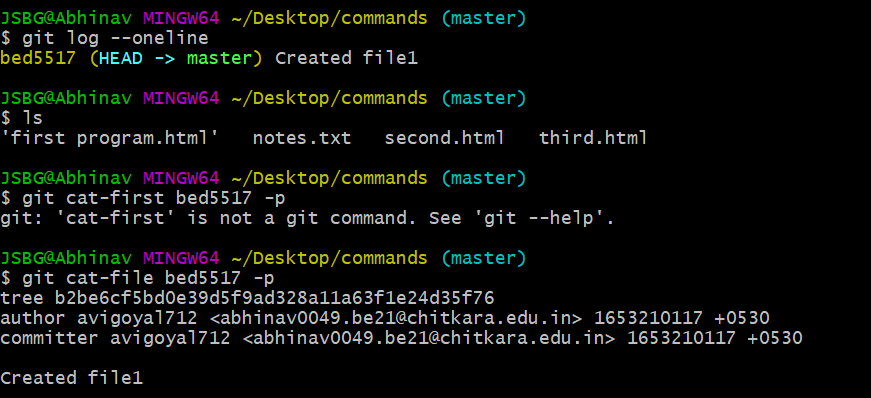
**git log --oneline :**

It gives the checkout information in one line.



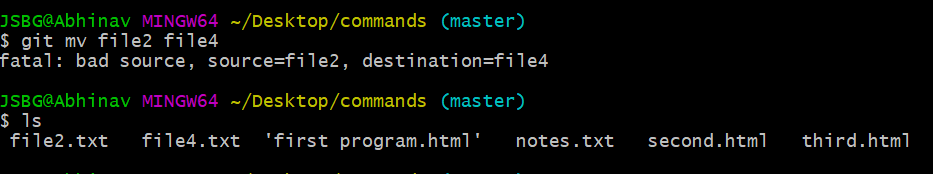
**git cat-file [checksum] -p :**

This command is used to see the commits and content using checksum.



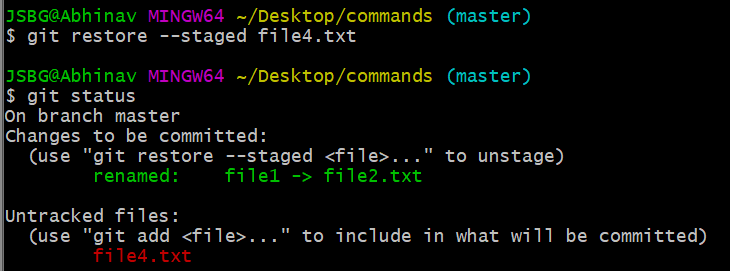
**git mv [file name] [file name] :**

This command is used to rename a file.



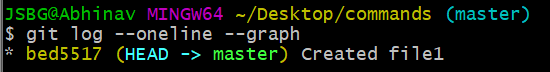
**git restore --staged [file name] :**

This command is used to remove from staging area.



**git log --oneline --graph :**

It gives the checkout in the form of network graph.



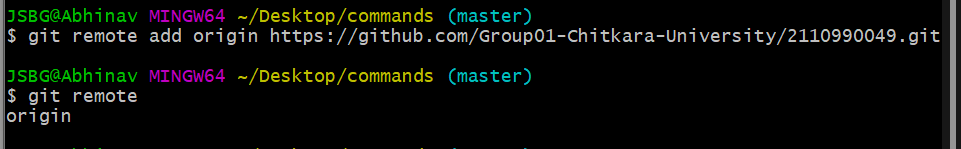
**git clone [URL] :**

This command is used to obtain a repository from an existing URL.



**git remote:**

This command is used to connect your local repository to the remote server.

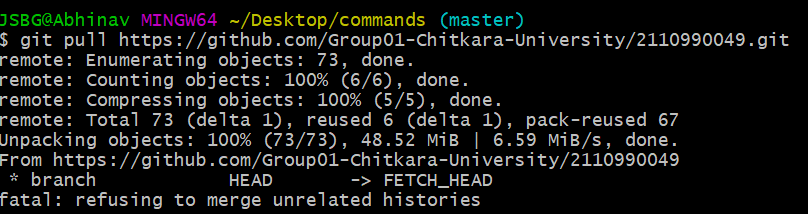
****

**git push :**

This command is used to push our repository.

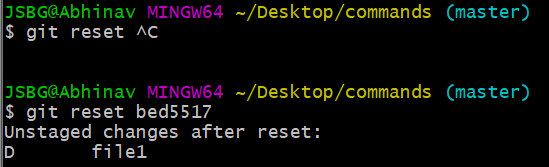
**git pull :**

This command is used to pull a repository.



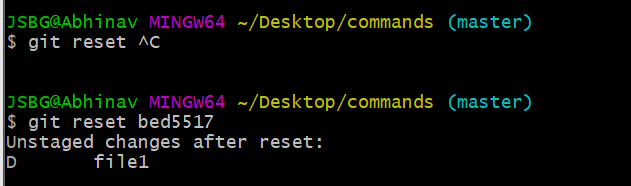
**git remote remove [remote name] :**

This command is to remove the remote.



**git reset [checksum] :**

This command is used to restore the commit upto the checksum.

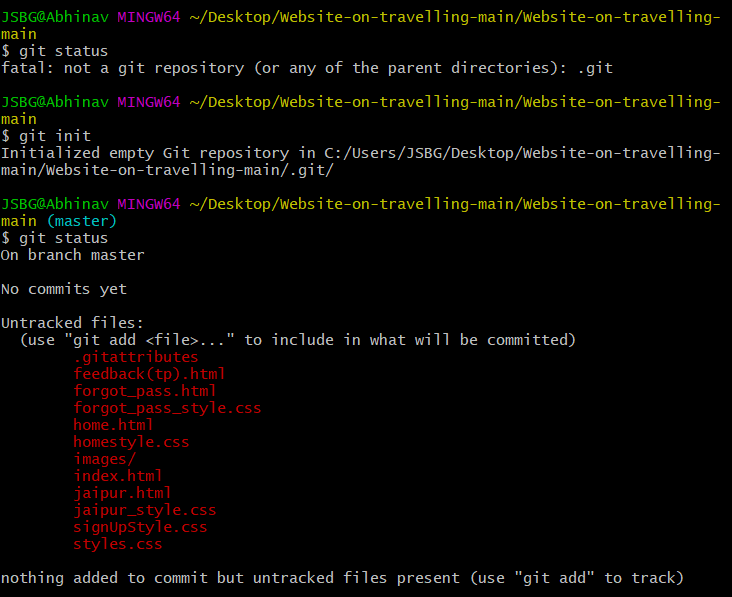


**5. Workflow and Discussion**

1. Firstly, I have to upload our site on github. For that

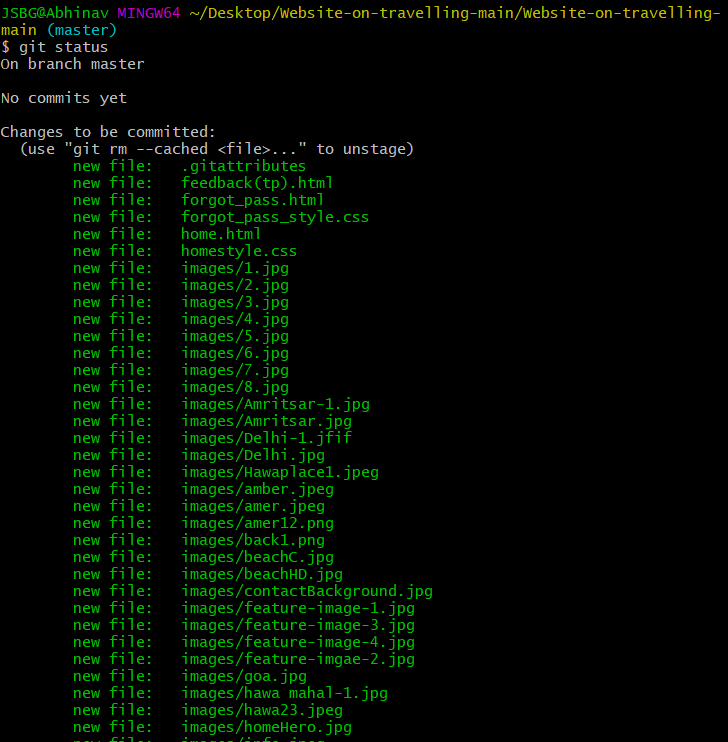
I started GitBash in the folder with files of our website.

**$ git init ->**



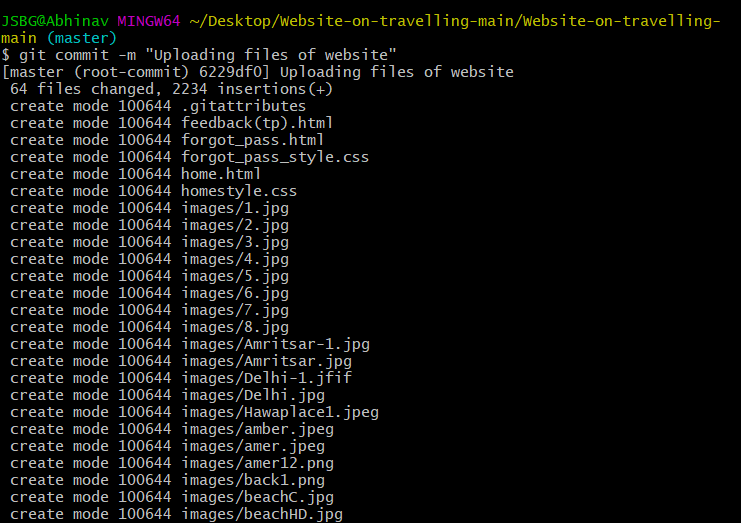
1. Adding files before committing

$ git add –a ->



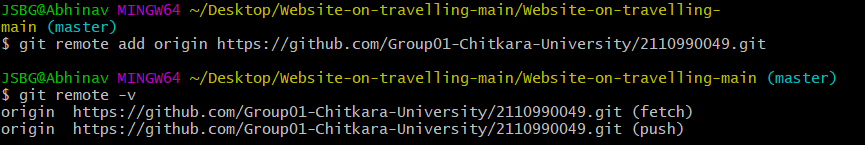
1. Committing files

$git commit -m “” ->



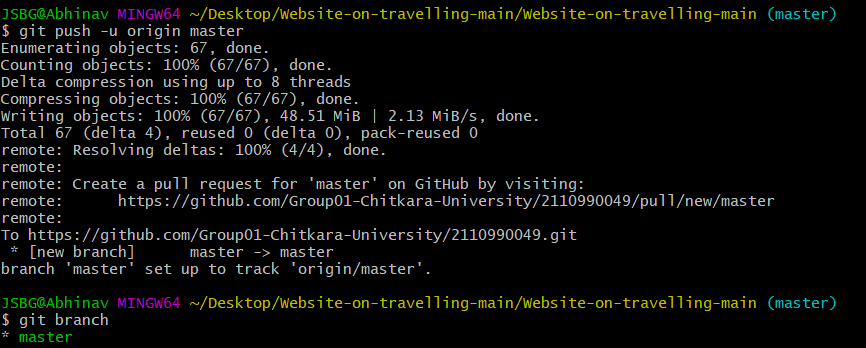
1. For pushing our website folder ,we need to set a remote server to github account.

$ git remote add origin <https://github.com/Group01-Chitkara-University/2110990049.git>



1. **Pushing the changes on the server.**

**$git push -u origin master ->**

****

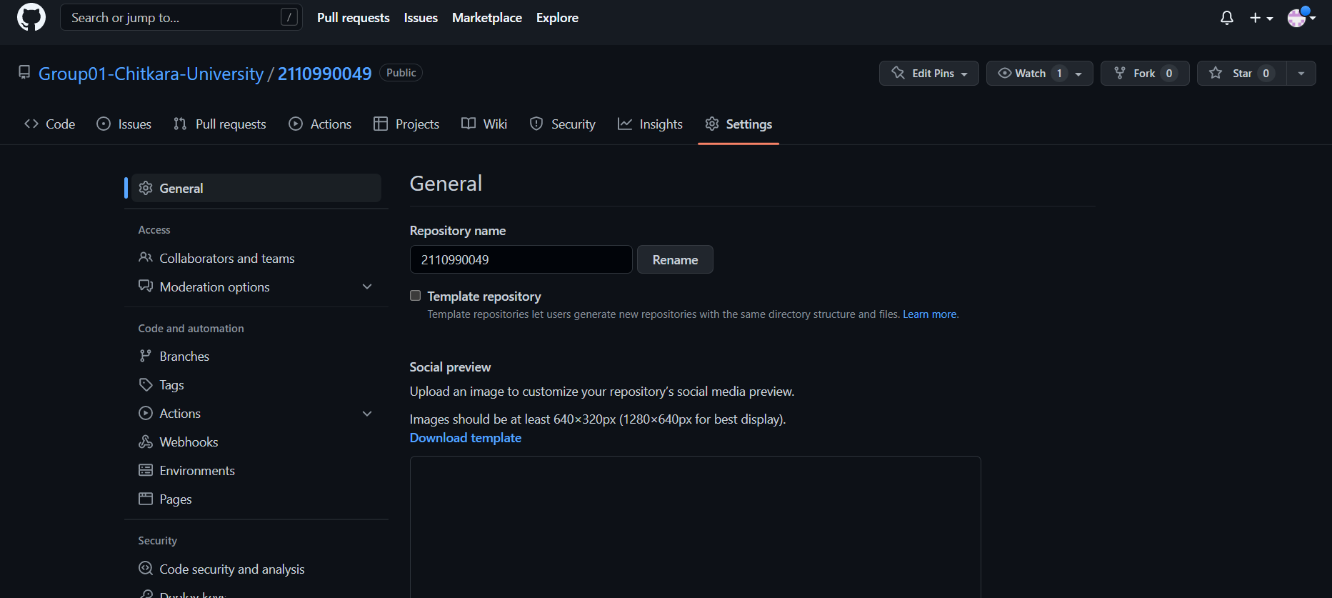
1. **After pushing the local repository on the github.**

****

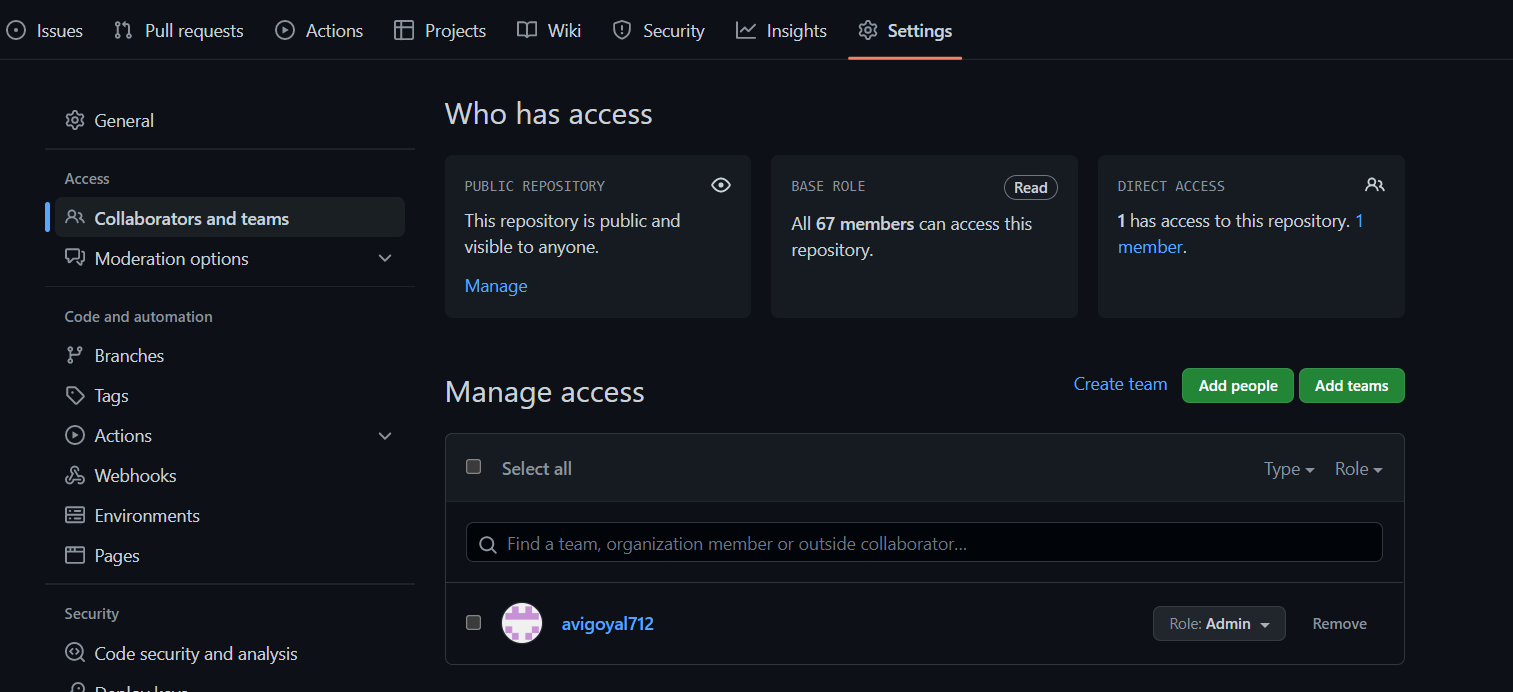
**Now I have to add my team member on my repository as a collaborator so that they can commit their changes on the website and can improve the site.**

**ADDING COLLABORATOR**

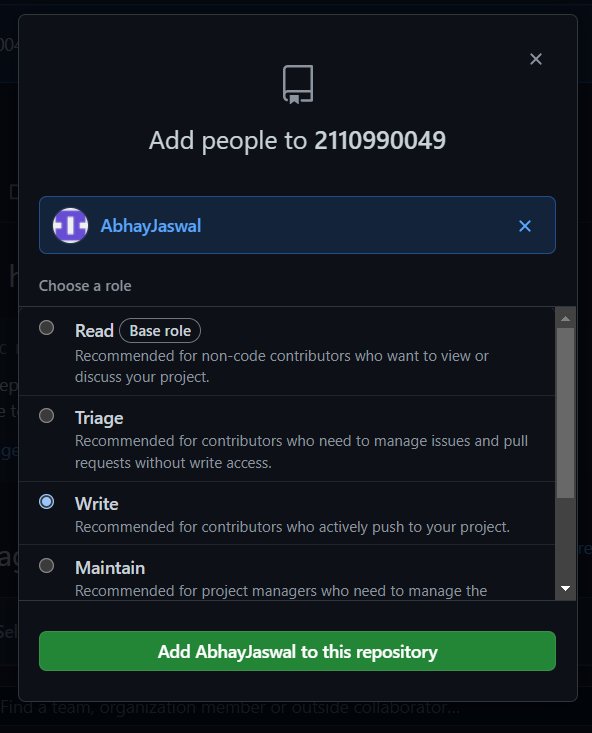
* 1. **After opening the repository . Click on Settings**

****

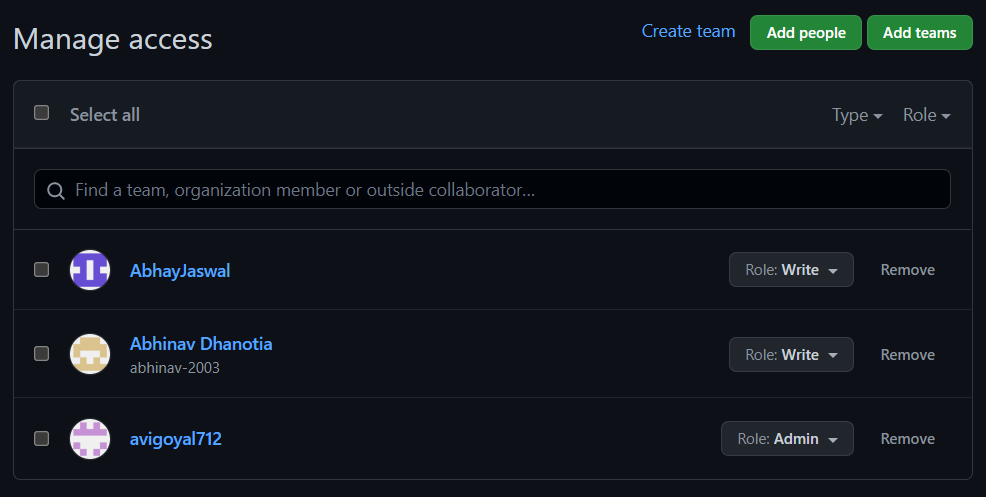
* 1. **Click on “Collaborators and teams” under the access.**

****

* 1. **Now click on “Add people” and type the username of the collaborator you want to add.**

****

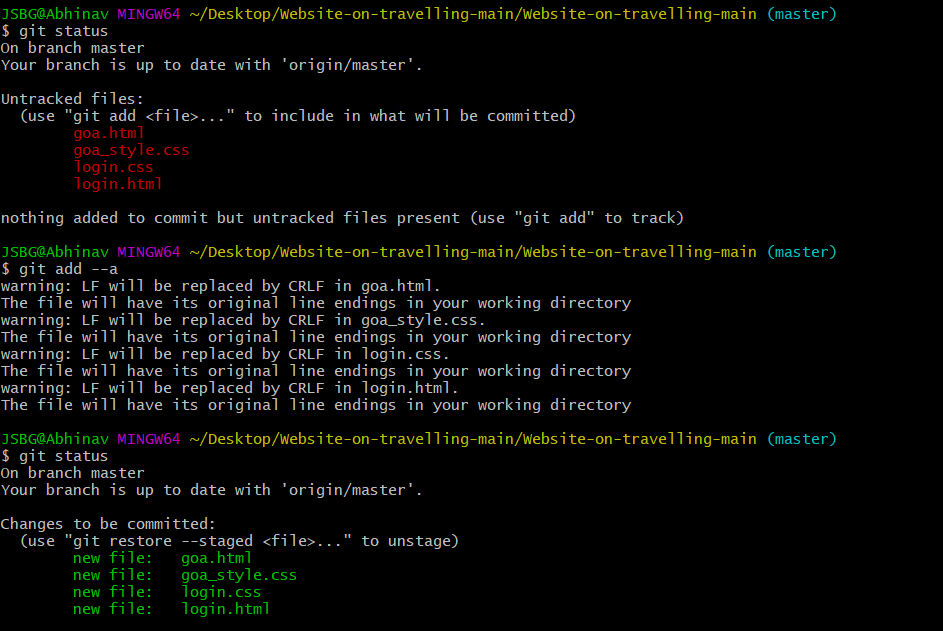
* 1. **After adding all the members.**

****

**Now I have to upload my changes on Github. For improving our website.**

* + - 1. **Open GitBash and run the commands**

**$git add –a 🡪**

****

* + - 1. **Committing the changes after adding all the files**

**$git commit – m “” ->**

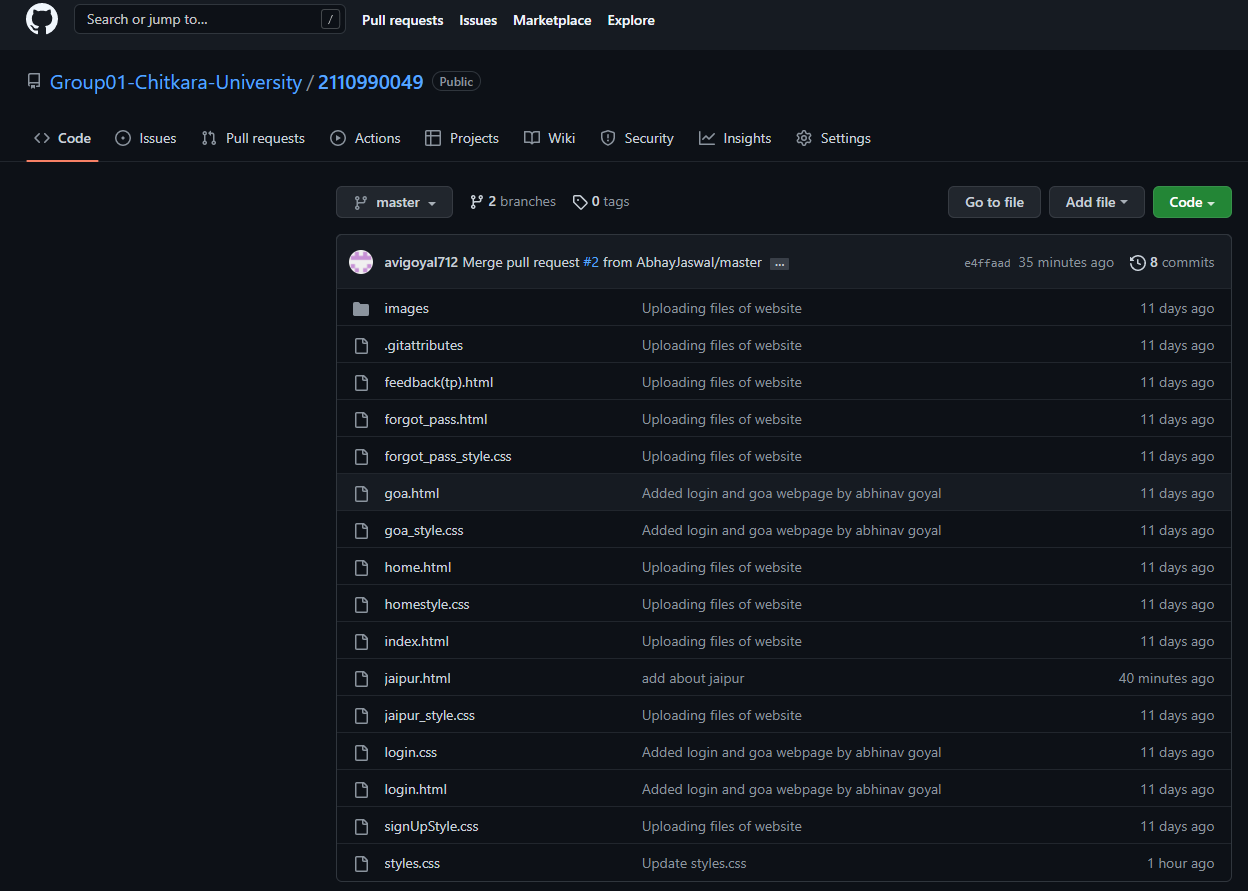
****

1. **Pushing the change on the remote server**

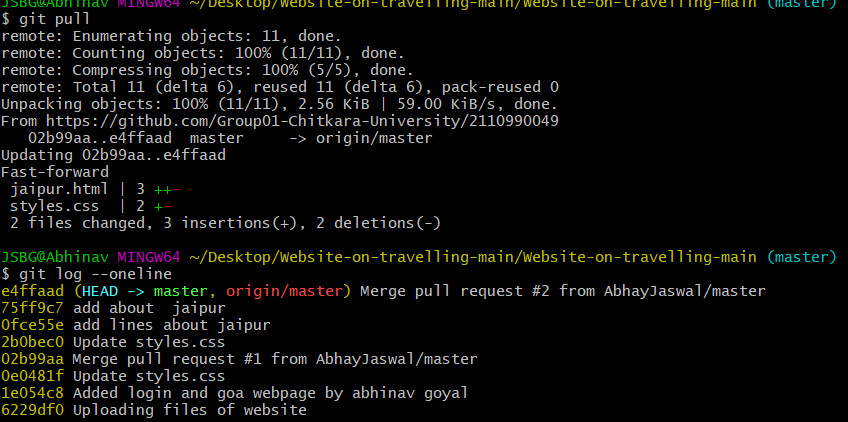
**$git push ->**

****

**5. After pushing all the change on the GitHub repository.**

****

**Git Log**

****



**6. Reference**

|  |  |
| --- | --- |
| S no. | Links |
| 1 | https://stackoverflow.com/questions/3528245/whats-the-difference-between-git-reset-mixed-soft-and-hard |
| 2 | https://initialcommit.com/blog/Technical-Guide-VCS-Internals |
| 3 | https://www.geeksforgeeks.org/version-control-systems/ |
| 4 | https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control |
| 5 | https://www.toolsqa.com/git/what-is-git/ |