Subject Name:

**Source Code Management**

Subject Code:

**22**

**CS**

**003**

Session:

**2023**

**-**

**2024**

Department:

**DCSE**



**Submitted To: Submitted By:**

**Dr. Sharad Chauhan Komal**

**Assistant Professor 2310992042**

**Chitkara University, Punjab Sem-II (2023-27)**

**Source Code Management File**

Subject Name: **Source Code Management (SCM)**

Subject Code: **22CS003**

Cluster: **Beta**

**Submitted By:**

**Name**: Komal

**Roll No**. 2310992042

**Group**: 23-A

**Task 1.1 Submission** (Week 4)

1. Setting up of Git Client
2. Setting up GitHub Account
3. Generate logs
4. Create and visualize branches
5. Git lifecycle description

# INDEX

|  |  |  |
| --- | --- | --- |
| **Sr.**  **No.** | **Practical Name** | **Teacher Sign** |
| 1 | Setting up of Git Client |  |
| 2 | Setting up GitHub Account |  |
| 3 | Generating logs . |  |
| 4 | Create and visualize branches |  |
| 5 | Git lifecycle description |  |
| 6 |  |  |
| 7 |  |  |
| 8 |  |  |
| 9 |  |  |
| 10 |  |  |
| 11 |  |  |
| 12 |  |  |

## EXPERIMENT 1

**Aim:** Setting up of Git Client

**Theory:**

GIT: It’s a Version Control System (VCS). It is a software or we can say a server by which we are able to track all the previous changes in the code. It is basically used for pushing and pulling of code. We can use git and git-hub parallelly to work with multiple members or individually. We can make, edit, recreate, copy or download any code on git hub using git.

**Procedure:** We can install Git on Windows, using the most official build which is available for download on the GIT’s official website or by just typing (scm git) on any search engine. We can go on <https://git-scm.com/download/win>and can select the platform and bit-version to download. And after clicking on your desired bit-version or ios it will start downloading automatically.

**Snapshots of download:**

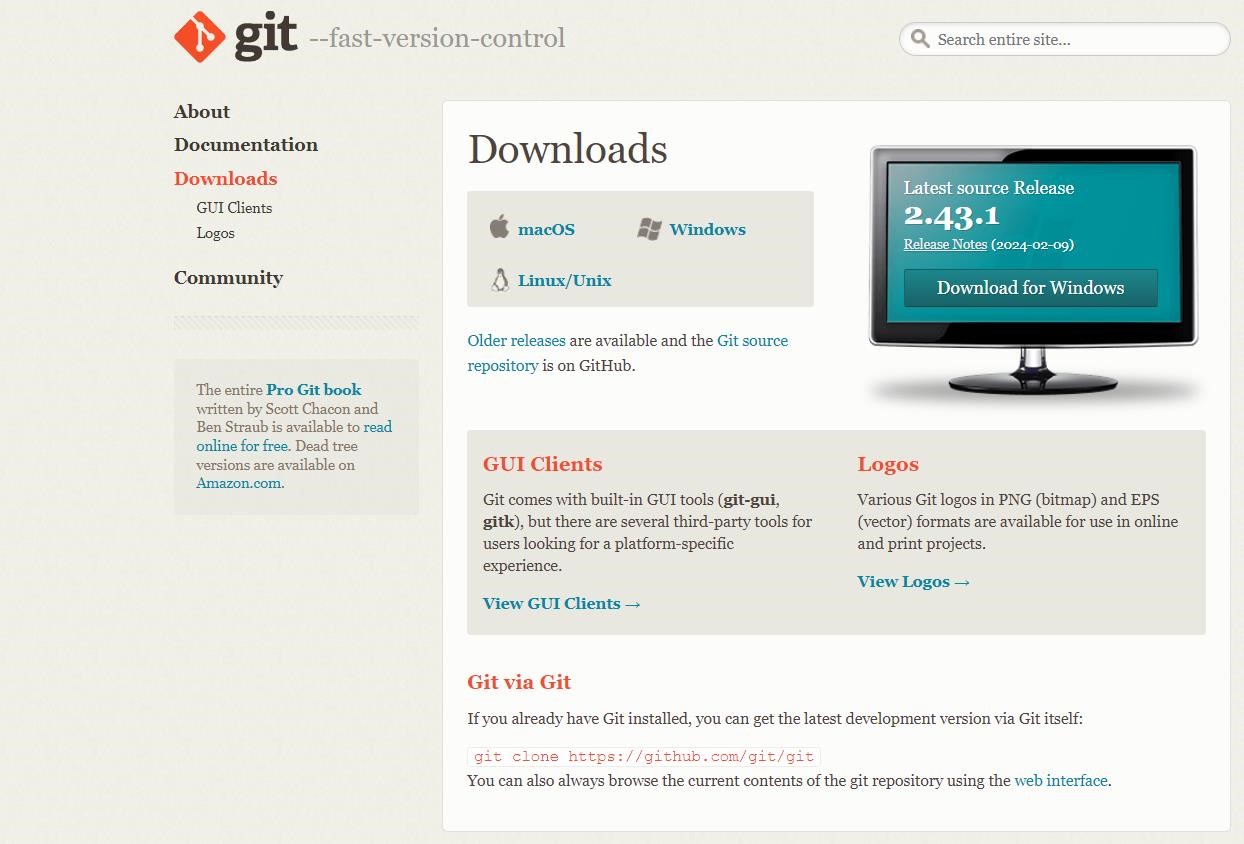


Fig-1.1 Opted for “64-bit Git for Windows Setup”

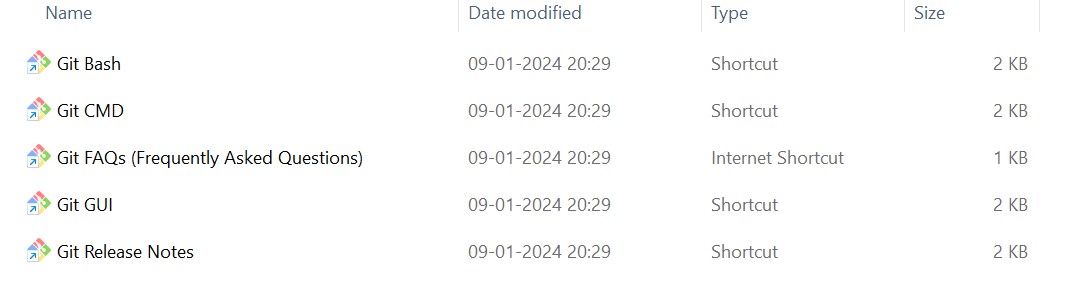


Fig-1.2 Git and its files in downloads

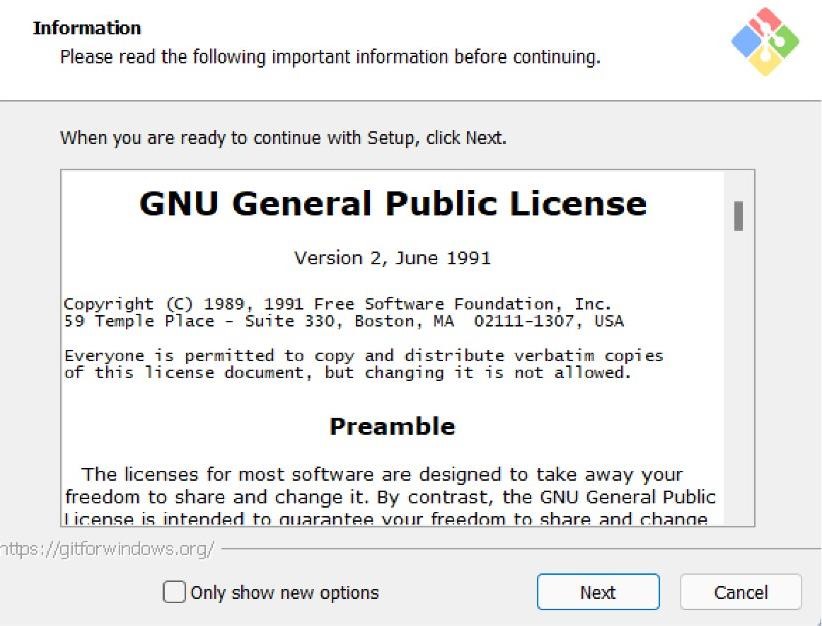
Fig

-

1.3

Git

Setup



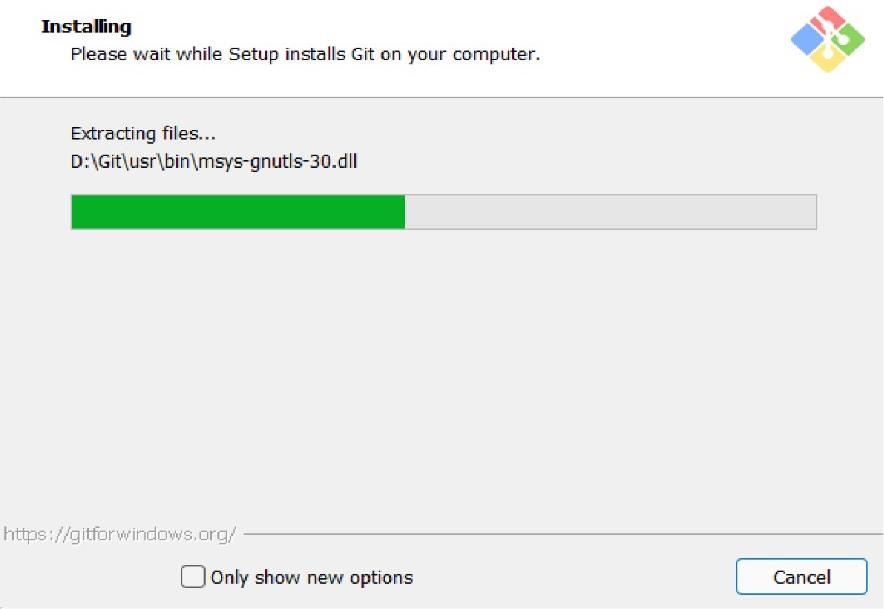
Fig

-

1.4

Git

Installation



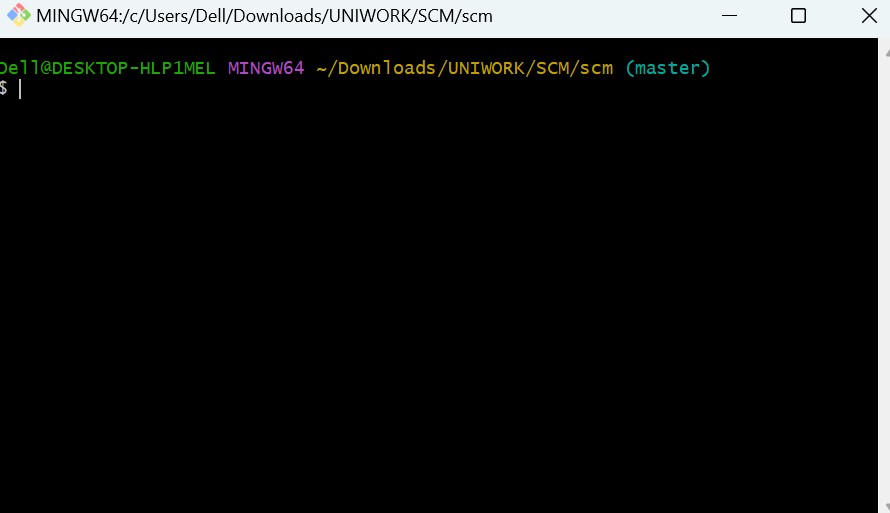


Fig-1.5 Git Bash launched

# EXPERIMENT 2

**Aim:** Setting up GitHub Account

**Theory:**

GitHub: GitHub is a website and cloud-based service (client) that helps an individual or developers to store and manage their code. We can also track as well as control changes to our or public code.

Advantages of GitHub: GitHub has a user-friendly interface and is easy to use.We can connect the git-hub and git but using some commands shown below in figure

001. Without GitHub we cannot use Git because it generally requires a host and if we are working for a project, we need to share it will our team members, which can only be done by making a repository. Additionally, anyone can sign up and host a public code repository for free, which makes GitHub especially popular with open-source projects.

**Procedure:**

To make an account on GitHub, we search for GitHub on our browser or visit [https://github.com/signup.](https://github.com/signup) Then, we will enter our mail ID and create a username and password for a GitHub account.

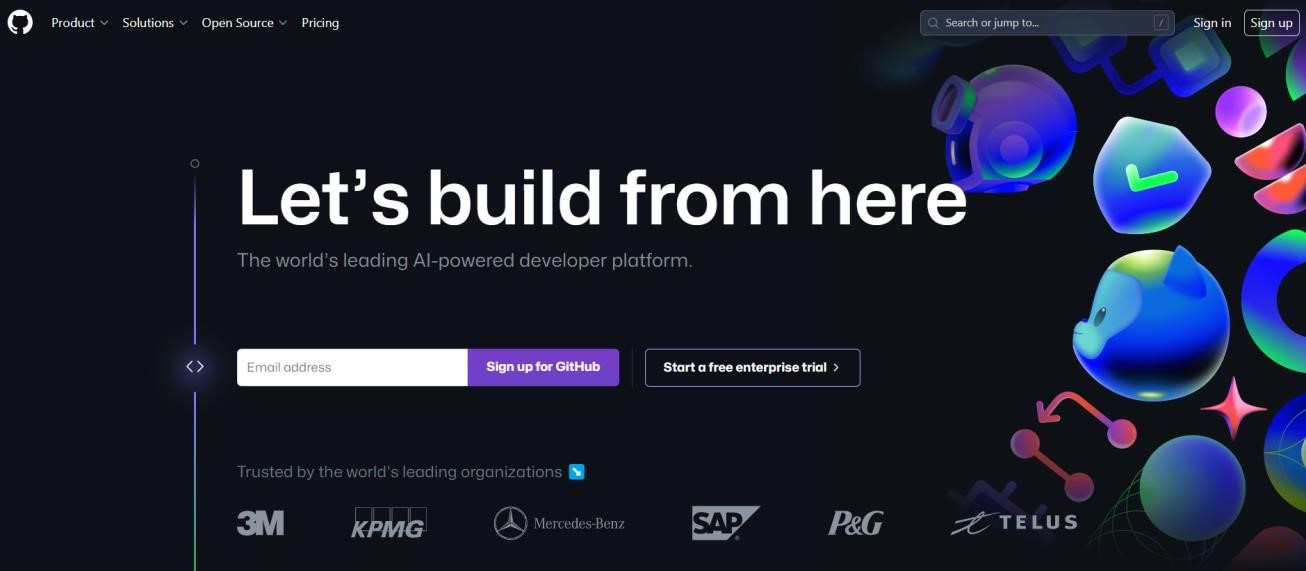
**G**

Fig-2.1 GitHub Sign Up

After visiting the link this type of interface will appear, if you already have an account, you can sign in and if not, you can create.

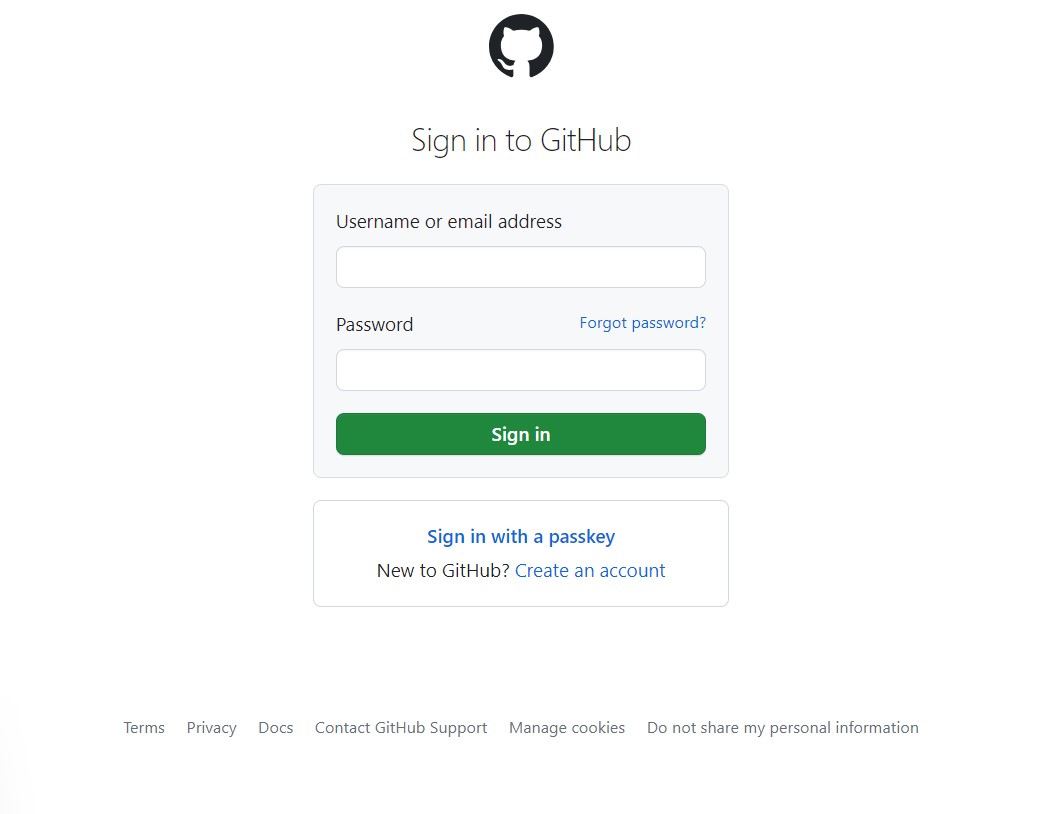


Fig-2.2 GitHub Login

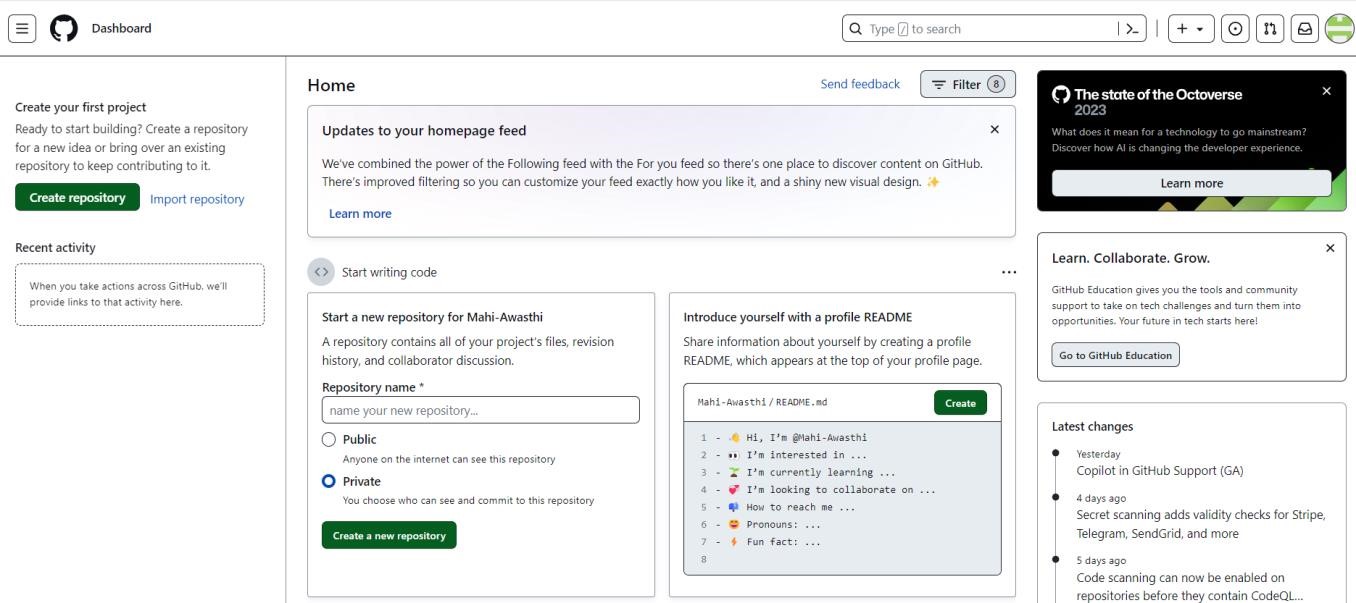


Fig-2.3 GitHub Interface

## EXPERIMENT 3

**Aim:** Program to generate log

**Theory:**

Logs: Logs are nothing but the history which we can see in git by using the code git log. It contains all the past commits, insertions and deletions in it which we can see any time. Logs helps to check that what were the changes in the code or any other file and by whom. It also contains the number of insertions and deletions including at which time it was changed.

**Procedure:**

First of all, create a local repository using Git. For this, you have to make a folder in your device, right click and select “Git Bash Here”. This opens the Git terminal. To create a new local repository, use the command “git init” and it creates a folder “.git”.

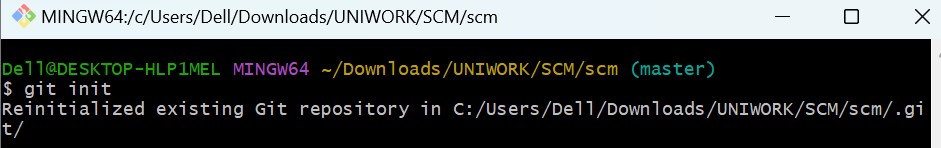


Fig-3.1 Git Init Command

When we use GIT for the first time, we have to give the user name and email so that if I am going to change in project, it will be visible to all.

For this, we use command:

“git config --global user.name Name” “git config --global user.email email”

For verifying the user’s name and email, we use: “git config

--global user.name”

“git config --global user.email”

Some Important Commands:-

1. ls  It gives the file names in the folder.
2. ls -lart  Gives the hidden files also.
3. git status  Displays the state of the working directory and the staged snapshot.
4. touch filename  This command creates a new file in the repository.
5. Clear  It clears the terminal.
6. rm -rf .git  It removes the repository.
7. git log  displays all of the commits in a repository's history
8. git diff  It compares my working tree to staging area.

**Algorithm:**

**1. Check Git Version:**

* Execute the command `**git --version**` in the terminal.
* Retrieve and display the installed Git version.

**2. Configure Username and Email:**

- Execute the commands: **git config --global user.name "Your Name" gitconfig—globaluser.email "your\_email@example.com"** - Set the global username and email for Git.

**3. Initialize a Git Repository:**

* Navigate to the project directory in the terminal.
* Execute the command `**git init**`.
* Initialize a new Git repository in the current directory.

**4. Create a New File:**

- Create a new file with the help of **touch filename**.

**5. Check Status of Git Repository:**

* Execute the command `**git status**`.
* Check the current status of the Git repository, including any untracked or modified files.

**6. Stage the File:**

* Execute the command `**git add filename**`.
* Stage the specified file for the next commit.

**7. Check Status Again:**

* Execute the command `**git status**`.
* Verify the changes and the status of the repository after staging the file.

* 1. **Commit the Changes:**
* Execute the command `**git commit -m "file name"**`.
* Commit the staged changes with a descriptive message.

* 1. **View Commit History:**
* Execute the command `**git log**`.
* View the commit history in the terminal, showing commit hashes, authors, dates, and commit messages.

**10. View Compact Commit History:**

* Execute the command `**git log --oneline**`.
* View a compact version of the commit history, showing abbreviated commit hashes and commit messages.

**11. View Commit History with Stats:**

* Execute the command `**git log --stat**`.
* View the commit history with additional statistics, including the number of insertions and deletions per file.

**12. View Details of a Specific Commit:**

* Execute the command `**git show commit\_hash**`.
* View the detailed information about a specific commit, including changes made and the commit message.

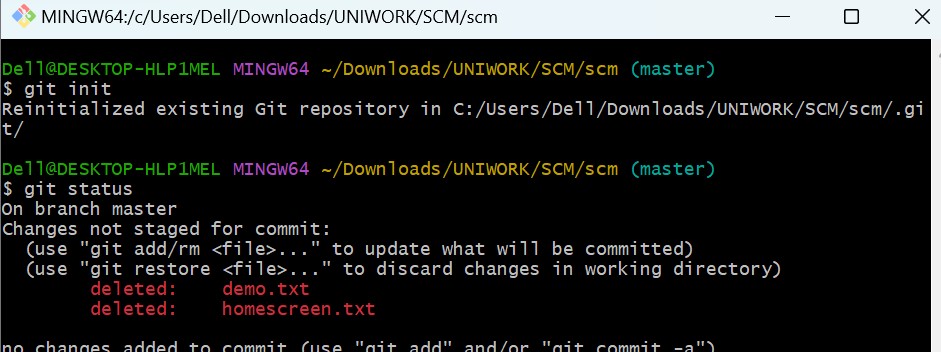


Fig-3.2 Git Status

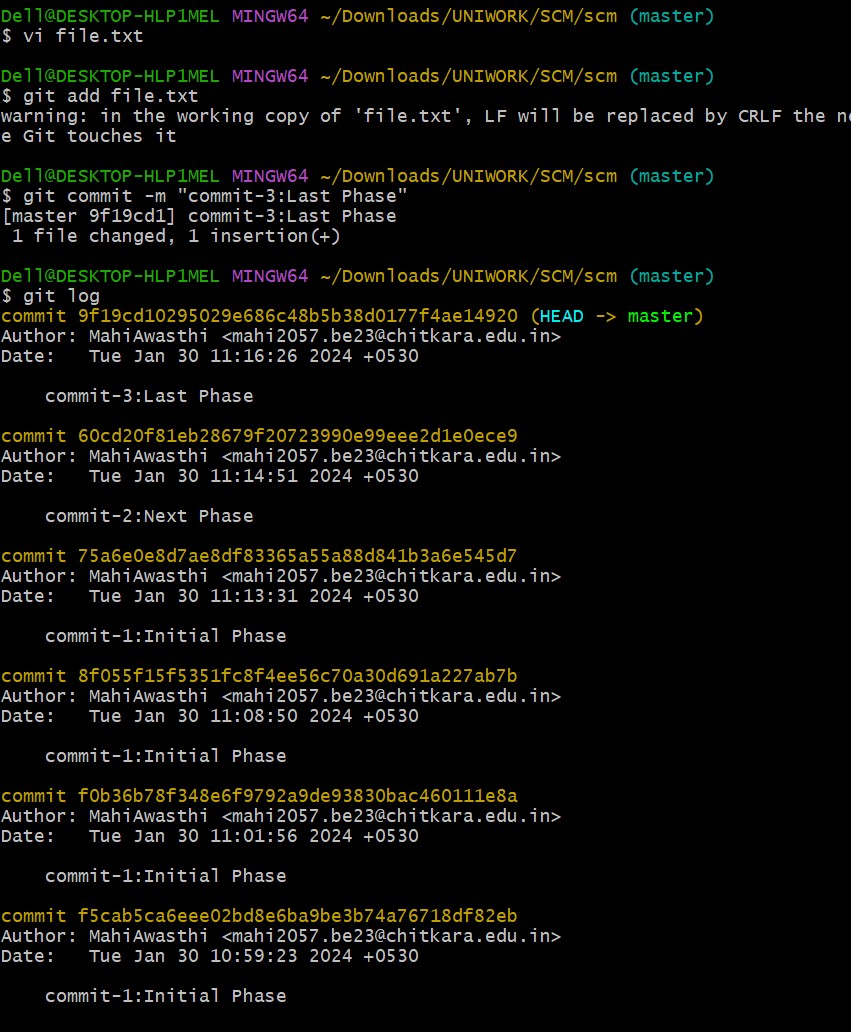


Fig-3.3 Git Log

# EXPERIMENT 4

**Aim:** Create and visualize branches

**Theory:**

Branching: A branch in Git is an independent line of work (a pointer to a specific commit). It allows users to create a branch from the original code (master branch) and isolate their work. Branches allow you to work on different parts of a project without impacting the main branch.

Create branches: The main branch in git is called as 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 file which are created in branch are not shown in master branch. We can also merge both the parent (master) and child (other branches). Syntax:

For creating a new branch, git branch name by default is master branch.

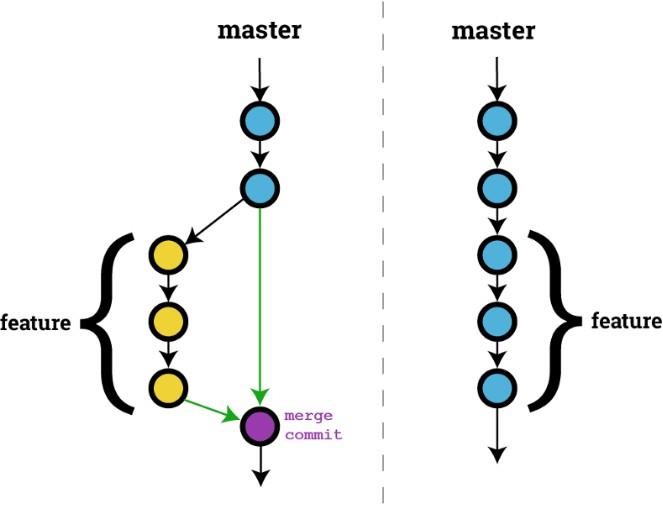


Fig-4.1

**Snapshots –**

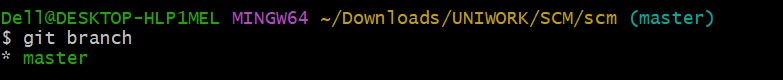


Fig-4.2 Default branch is master branch

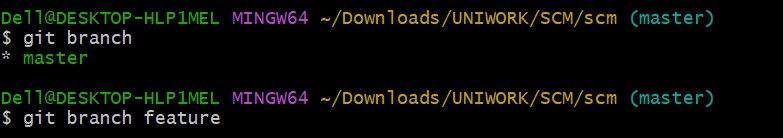


Fig-4.3 Adding a feature branch

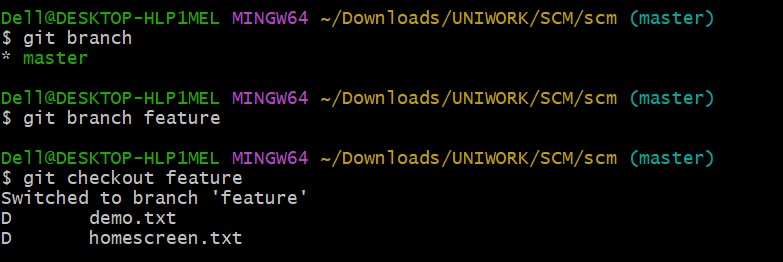


Fig-4.4 Switching to feature branch

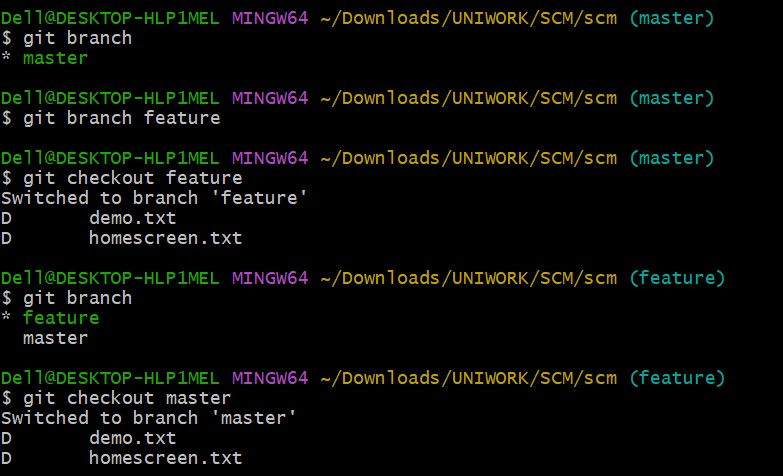


Fig-4.5 Switching to master branch

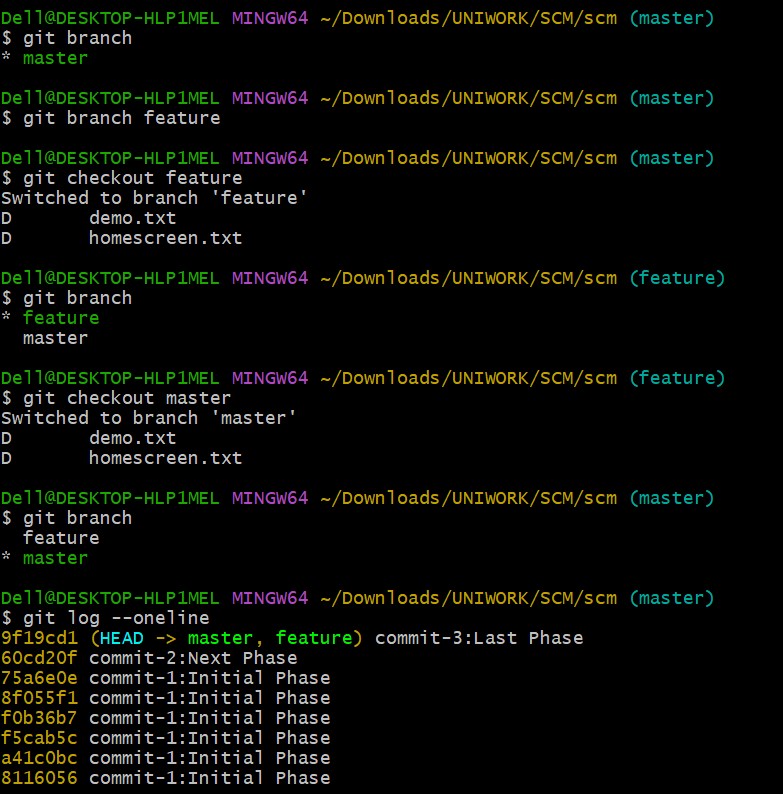


Fig-4.6 Checking commits

## EXPERIMENT 5

**Aim:** Git lifecycle description

**Theory:**

Stages in GIT Life Cycle: Files in a Git project have various stages like Creation, Modification, Refactoring, and Deletion and so on. Irrespective of whether this project is tracked by Git or not, these phases are still prevalent. However, when a project is under Git version control system, they are present in three major Git states in addition to these basic ones. Here are the three Git states:

* Working directory
* Staging area
* Git directory

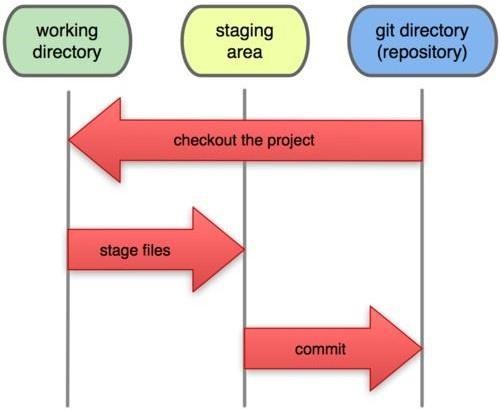


Fig-5.1

**Working Directory:**

Consider a project residing in your local system. This project may or may not be tracked by Git. In either case, this project directory is called your Working directory.

**Staging Area:**

Staging area is the playground where you group, add and organize the files to be committed to Git for tracking their versions.

**Git Directory:**

Now that the files to be committed are grouped and ready in the staging area, we can commit these files. So, we commit this group of files along with a commit message explaining what the commit is about. Apart from commit message, this step also records the author and time of the commit. Now, a snapshot of the files in the commit is recorded by Git. The information related to this commit is stored in the Git directory.

**Remote Repository:** It means mirror or clone of the local Git repository in GitHub. And pushing means uploading the commits from local Git repository to remote repository hosted in GitHub.

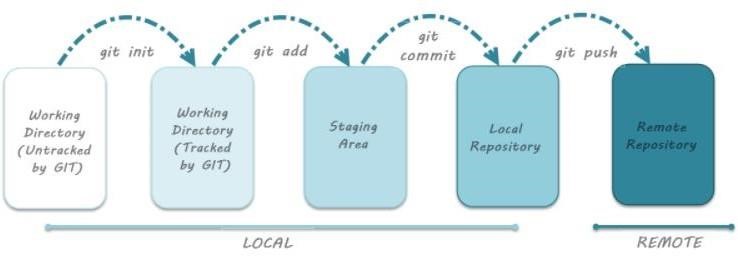


Fig-5.2

**Algorithm:**

1. Initialize the Git repository:
   * **git init**

1. Create a new file:
   * **touch filename**

1. Check the status of the repository:
   * **git status**

1. Stage all files with the .txt extension:
   * **git add \*.txt**

1. Check the status again to see staged changes:
   * **git status**

1. Add further new files.

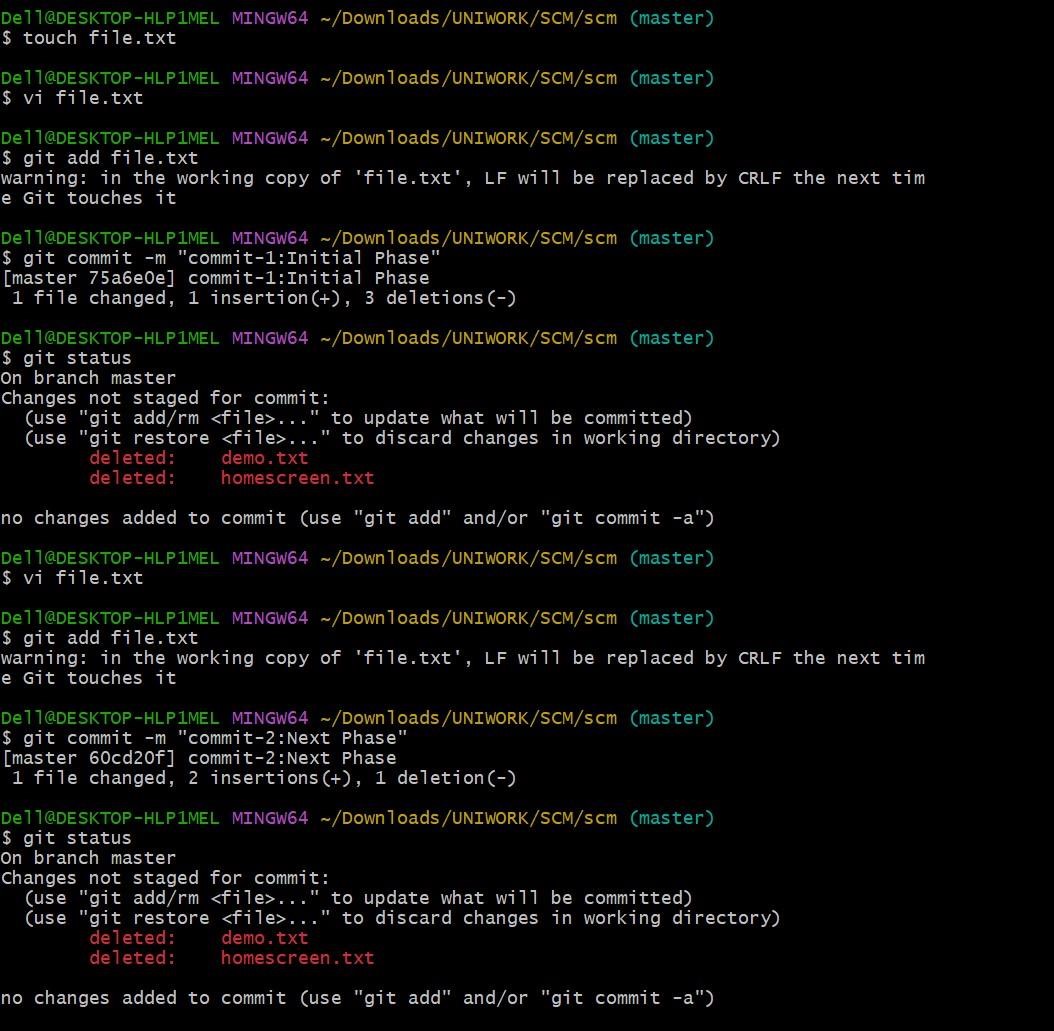
1. Check the status to see untracked or modified files:
   * **git status**

1. Stage all changes (including new files): 
   * **git add -A**

1. Check the status to verify staged changes:
   * **git status**

1. Remove a file from the staging area (but keep it in the working directory): - **git rm --cached <file name>**

1. Commit the staged changes with a comment: -  **git commit -m "comment**"



### Fig-5.3