



# SOURCE CODE MANAGEMENT FILE

**Submitted by:**

Name: Lakshya Ashwini

Roll No. 2310991883

Group 21

Submission of:

Task 1.1

**Submitted To:**

Dr. Renu Popli

Assistant Professor

Department of Computer

Science & Engineering

Chitkara University Institute

of Engineering and

Technology

Rajpura, Punjab

## **Source Code Management File**

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

Subject Code: **22CS003**

**Submitted By:**

Name: **Lakshya Ashwini**

Roll No.: **2310991883**

Group: **G21-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

# Experiment 1

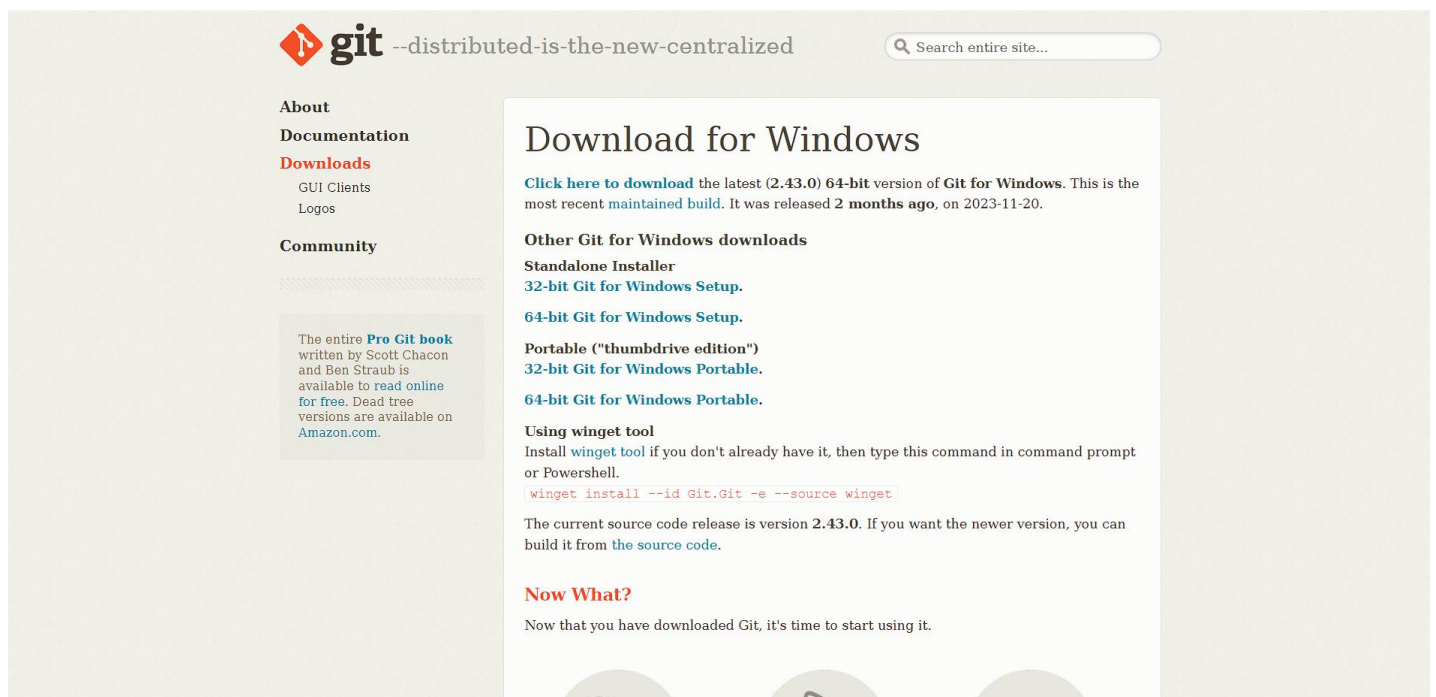
**Aim:** Setting up of Git Client

## 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.

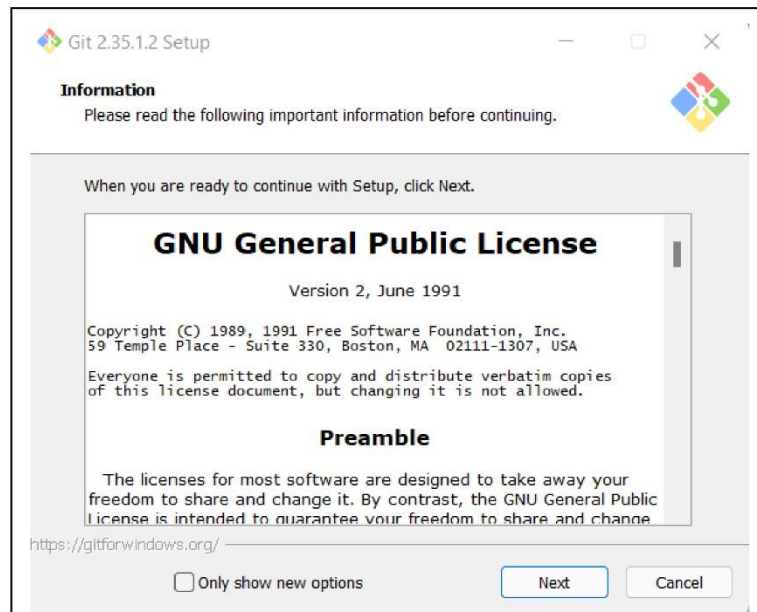
## Snapshots of download:



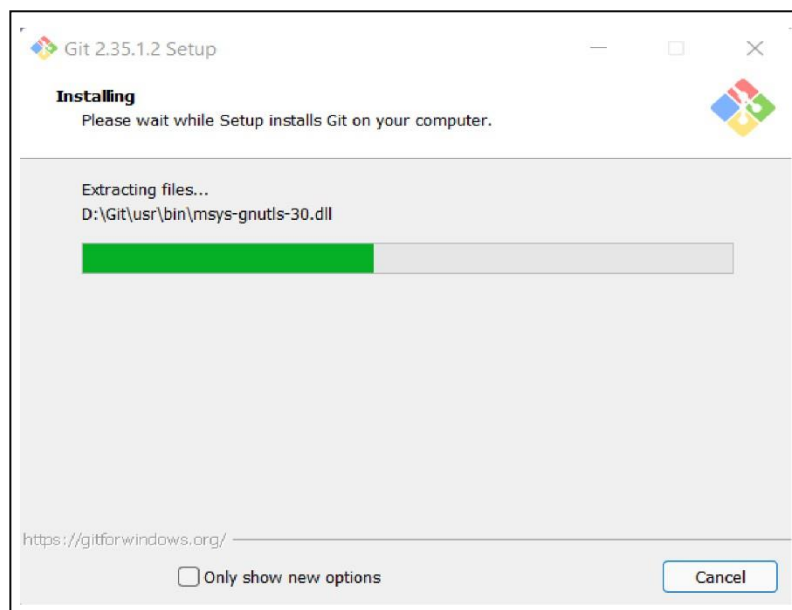
Opted for “64-bit Git for Windows Setup”

Name	Date modified	Type	Size
 Git Bash	16-03-2022 08:51	Shortcut	2 KB
 Git CMD	16-03-2022 08:51	Shortcut	2 KB
 Git FAQs (Frequently Asked Questions)	16-03-2022 08:51	Internet Shortcut	1 KB
 Git GUI	16-03-2022 08:51	Shortcut	2 KB
 Git Release Notes	16-03-2022 08:51	Shortcut	2 KB

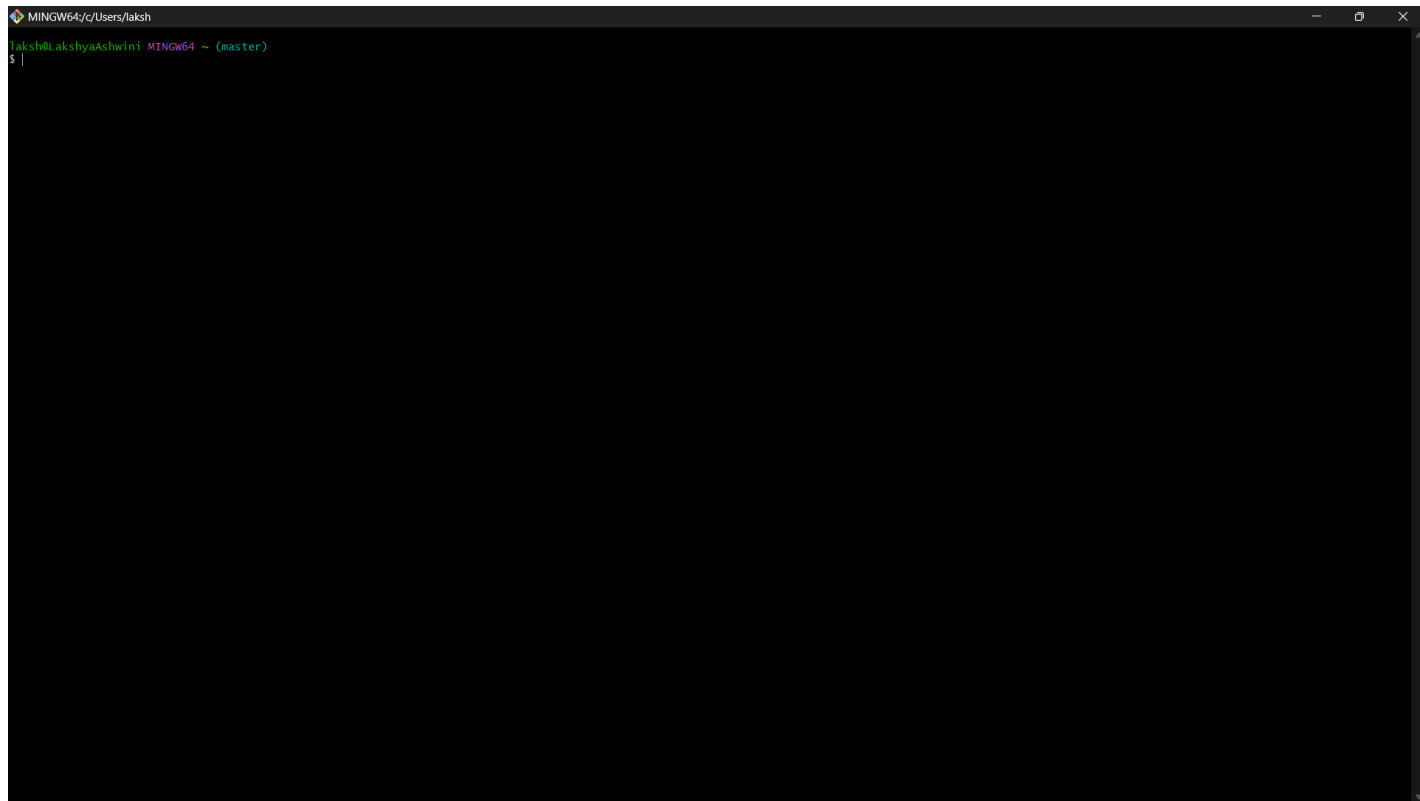
## Git and its files in downloads



## Git Setup



## Git Installation



Git Bash launched



## Experiment 2

**Aim:** Setting up GitHub Account

### Theory:

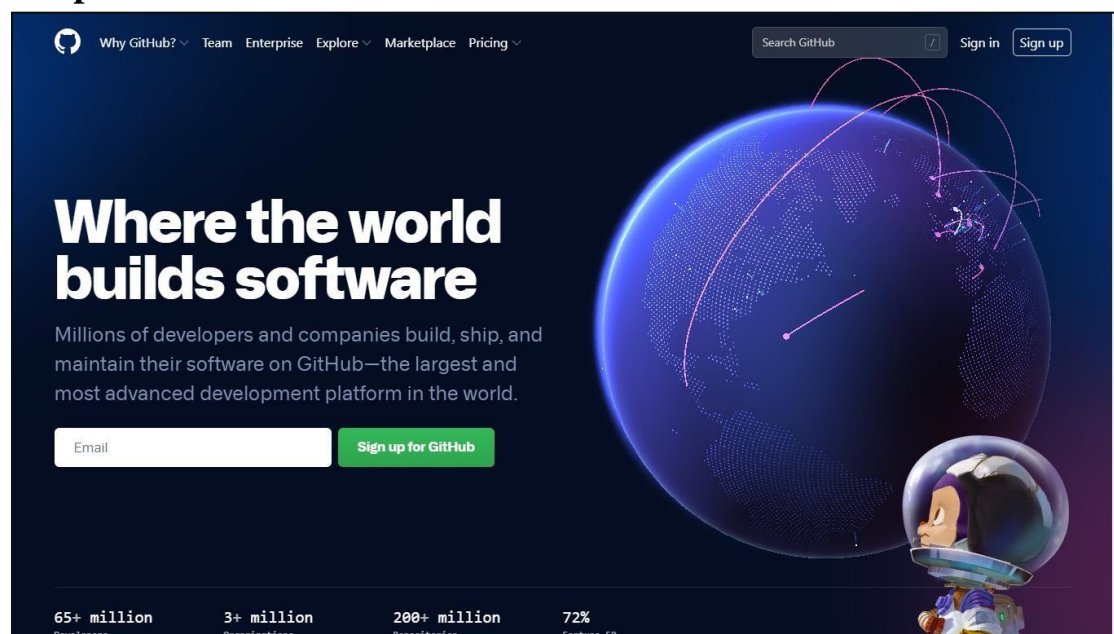
**GitHub:** GitHub is a website and cloud-based service (client) that helps an individual or developers to store and manage their code. We can also track as well as control changes to our or public code.

**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 01. 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 with 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.

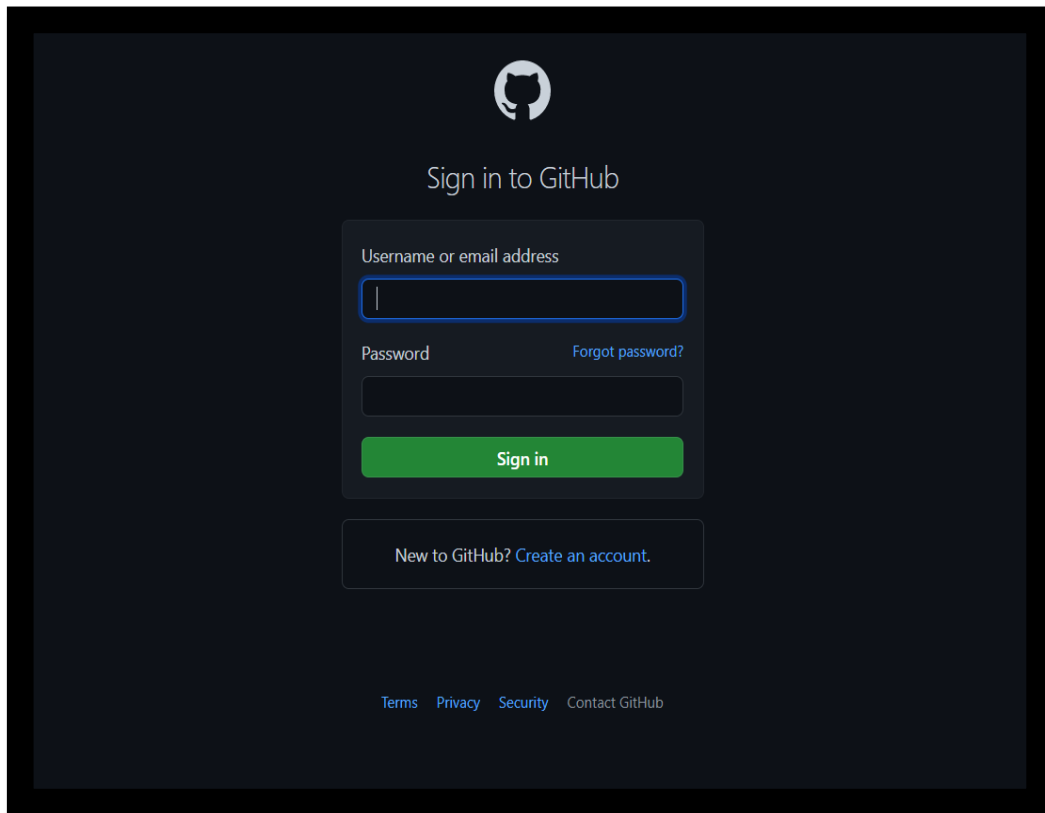
### Procedure:

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

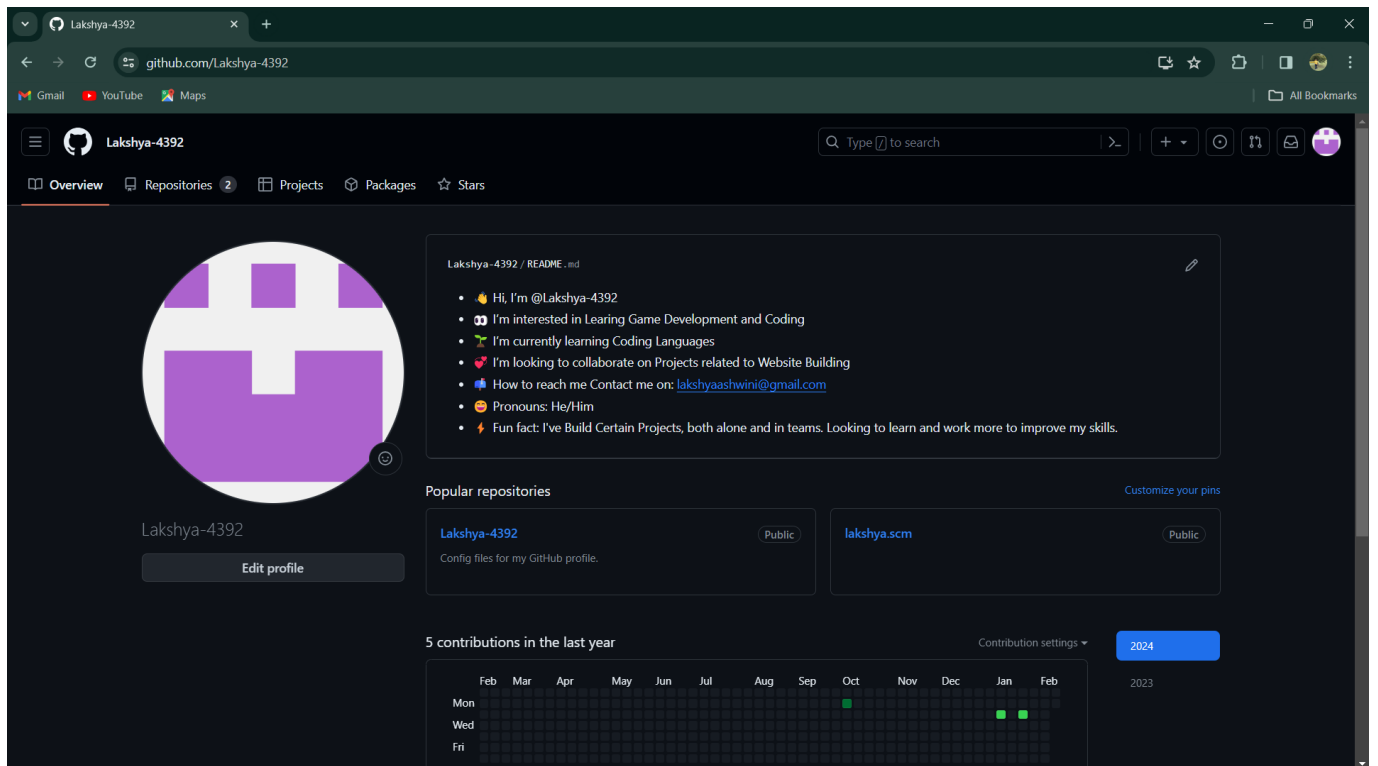
### Snapshots –



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.



## GitHub Login



## GitHub Interface

# Experiment 3

**Aim:** Program to Generate log

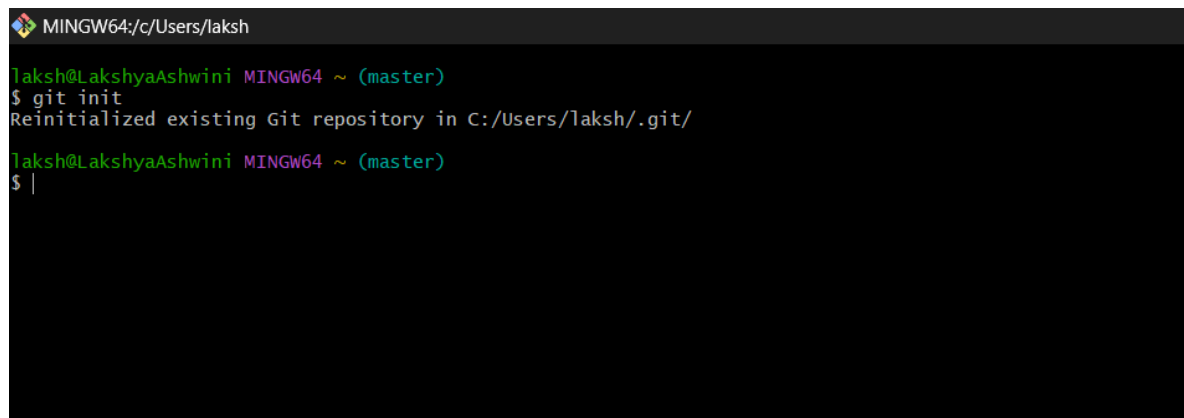
## Theory:

**Logs:** Logs are nothing but the history which we can see in git by using thecode git log.

It contains all the past commits, insertions and deletions in it which we can seeany 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 deletionsincluding 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”.

A screenshot of a terminal window titled 'MINGW64:/c/Users/laksh'. The prompt is 'laksh@LakshyaAshwini MINGW64 ~ (master)'. The user enters '\$ git init'. The terminal output is 'Reinitialized existing Git repository in C:/Users/laksh/.git/'. The prompt returns to 'laksh@LakshyaAshwini MINGW64 ~ (master)' and the user enters '\$ |'.

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

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

Git status:

```
MINGW64:/c:/Users/laksh/myproject_SCM
laksh@LakshyaAshwini MINGW64 ~
$ cd myproject_SCM

laksh@LakshyaAshwini MINGW64 ~/myproject_SCM (master)
$ git init
Reinitialized existing Git repository in C:/Users/laksh/myproject_SCM/.git/

laksh@LakshyaAshwini MINGW64 ~/myproject_SCM (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   myfile.txt

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:    myproject_SCM/myFile.txt

laksh@LakshyaAshwini MINGW64 ~/myproject_SCM (master)
$
```

Git log:

```
MINGW64:/c:/Users/laksh/myproject_SCM
laksh@LakshyaAshwini MINGW64 ~ (master)
$ git init
Reinitialized existing Git repository in C:/Users/laksh/.git/

laksh@LakshyaAshwini MINGW64 ~ (master)
$ cd myproject_SCM

laksh@LakshyaAshwini MINGW64 ~/myproject_SCM (master)
$ git init
Reinitialized existing Git repository in C:/Users/laksh/myproject_SCM/.git/

laksh@LakshyaAshwini MINGW64 ~/myproject_SCM (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   myfile.txt

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:    myproject_SCM/myFile.txt

laksh@LakshyaAshwini MINGW64 ~/myproject_SCM (master)
$ git log
commit e6eb7dc8ae6566d3eab0c9915b94c3645a5181ed (HEAD -> master, feature)
Author: Lakshya <lakshyaashwini@gmail.com>
Date: Thu Feb 8 11:17:56 2024 +0530

    file added

laksh@LakshyaAshwini MINGW64 ~/myproject_SCM (master)
$ |
```

The git log command displays a record of the commits in a Git repository. By default, the git log command displays a commit hash, the commit message and other commit metadata.

## Experiment 4

**Aim:** Create and visualize branches

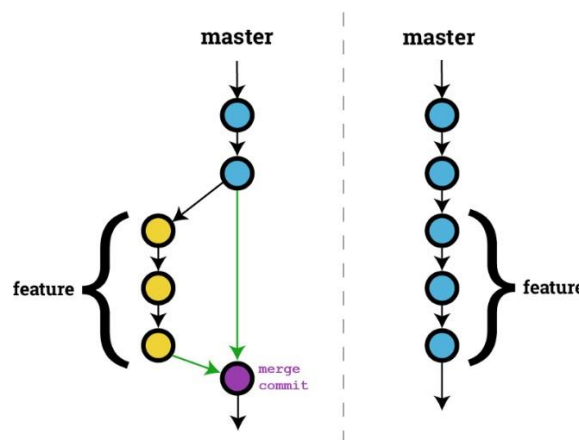
### Theory:

**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.



### Snapshots –

```
laksh@LakshyaAshwini MINGW64 ~/myproject_SCM (master)
$ git branch
* master

laksh@LakshyaAshwini MINGW64 ~/myproject_SCM (master)
$
```

Default branch is master branch

```
laksh@LakshyaAshwini MINGW64 ~/myproject_SCM (master)
$ git branch
* master

laksh@LakshyaAshwini MINGW64 ~/myproject_SCM (master)
$ git branch feature

laksh@LakshyaAshwini MINGW64 ~/myproject_SCM (master)
$ git branch
feature
* master

laksh@LakshyaAshwini MINGW64 ~/myproject_SCM (master)
$
```

Adding a feature branch

```
MINGW64:/c/Users/laksh/myproject_SCM

laksh@LakshyaAshwini MINGW64 ~/myproject_SCM (master)
$ git branch
feature
* master

laksh@LakshyaAshwini MINGW64 ~/myproject_SCM (master)
$ git checkout feature
Switched to branch 'feature'
A    myfile.txt
D    myproject_SCM/myfile.txt

laksh@LakshyaAshwini MINGW64 ~/myproject_SCM (feature)
$ git branch
* feature
  master

laksh@LakshyaAshwini MINGW64 ~/myproject_SCM (feature)
$
```

Switching to feature branch

```

MINGW64:/c/Users/laksh/myproject_SCM

laksh@LakshyaAshwini MINGW64 ~/myproject_SCM (feature)
$ git branch
* feature
  master

laksh@LakshyaAshwini MINGW64 ~/myproject_SCM (feature)
$ git branch branch2

laksh@LakshyaAshwini MINGW64 ~/myproject_SCM (feature)
$ git checkout branch2
Switched to branch 'branch2'
A       myfile.txt
D       myproject_SCM/myfile.txt

laksh@LakshyaAshwini MINGW64 ~/myproject_SCM (branch2)
$ git branch
* branch2
  feature
  master

laksh@LakshyaAshwini MINGW64 ~/myproject_SCM (branch2)
$ git checkout master
Switched to branch 'master'
A       myfile.txt
D       myproject_SCM/myfile.txt

laksh@LakshyaAshwini MINGW64 ~/myproject_SCM (master)
$ git branch
  branch2
  feature
* master

laksh@LakshyaAshwini MINGW64 ~/myproject_SCM (master)
$ |

```

Switching to master branch

```

MINGW64:/c/Users/laksh/myproject_SCM

laksh@LakshyaAshwini MINGW64 ~/myproject_SCM (master)
$ git branch
  branch2
  feature
* master

laksh@LakshyaAshwini MINGW64 ~/myproject_SCM (master)
$ git log --oneline
e6eb7dc (HEAD -> master, feature, branch2) file added

laksh@LakshyaAshwini MINGW64 ~/myproject_SCM (master)
$

```

Checking commits

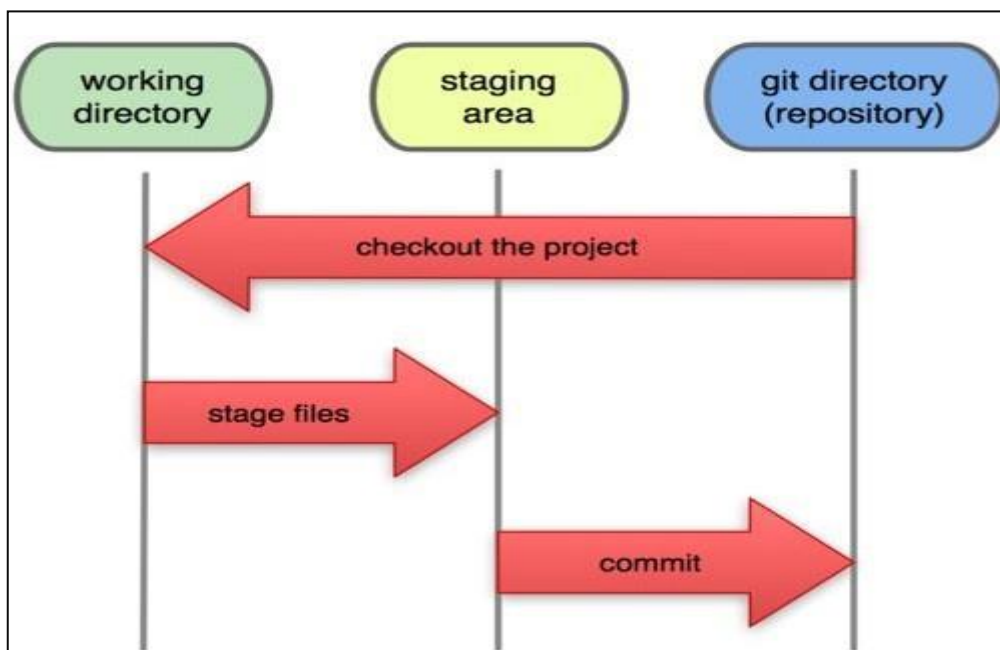
# Experiment 5

**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



## 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 is the commit 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.

