Subject Name: Source Code Management

Subject Code: CS181

Department: CSE

Cluster: Beta

Submitted By:

Ansh Wadhwa

2110990217

G8

Submitted To:

Mr. Monit Kapoor

1. What is GIT and why is it used?

Git is a DevOps tool used for source code management. Git is software for tracking changes in any set of files. It is a free and open-source version control system used to handle small to very large projects efficiently.
Git is used to tracking changes in the source code, enabling multiple developers to work together on non-linear development
Git is an example of a DVCS (hence Distributed Version Control System).

2. What is GITHUB?

GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere.

3. What is Repository?

A repository contains all of your project's files and each file's revision history. You can discuss and manage your project's work within the repository. A Git repository is the .git/ folder inside a project. This repository tracks all changes made to files in your project, building a history over time. Meaning, if you delete the .git/ folder, then you delete your project's history.

4. What is Version Control System (VCS)?

Version Control Systems are the software tools for tracking/managing all the changes made to the source code during the project development. It keeps a record of every single change made to the code. It also allows us to turn back to the previous version of the code if any mistake is made in the current version. Without a VCS in place, it would not be possible to monitor the development of the project.

5. Types of VCS

- Local Visual Control System
- Centralized Version Control System
- Distributed Version Control System

- **Local Visual Control System**

  Local Version Control System is located in your local machine. If the local machine crashes, it would not be possible to retrieve the files, and all the information will be lost. If anything happens to a single version, all the versions made after that will be lost.

- **Centralized Version Control System**

  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.

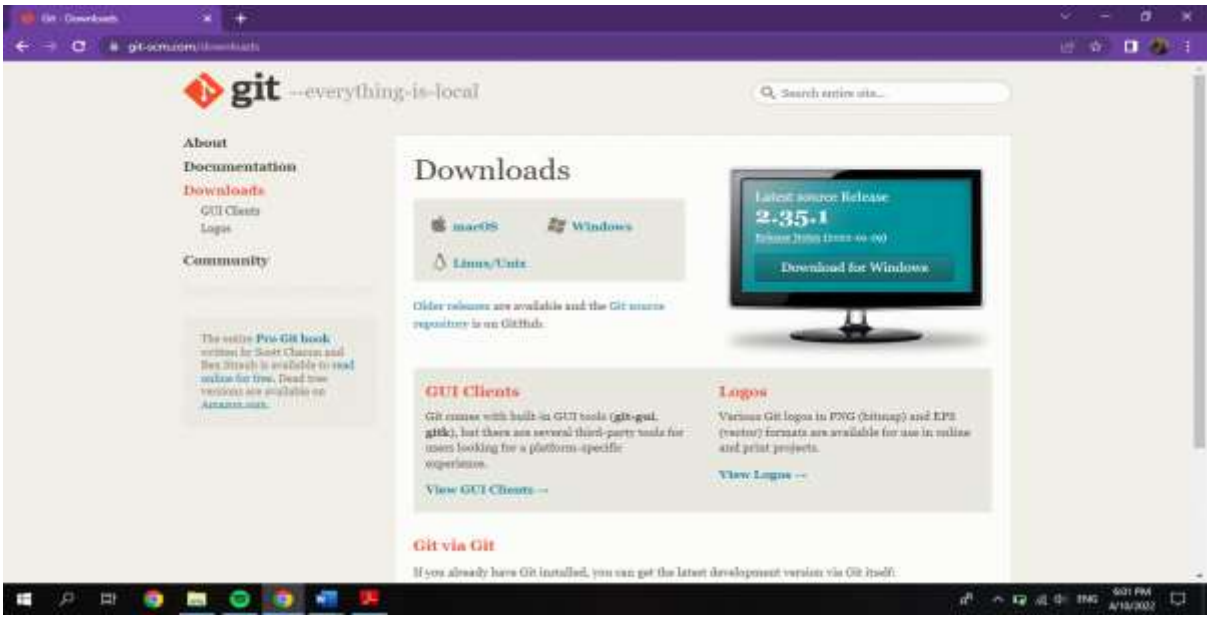
- **Distributed Version Control System**

  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.

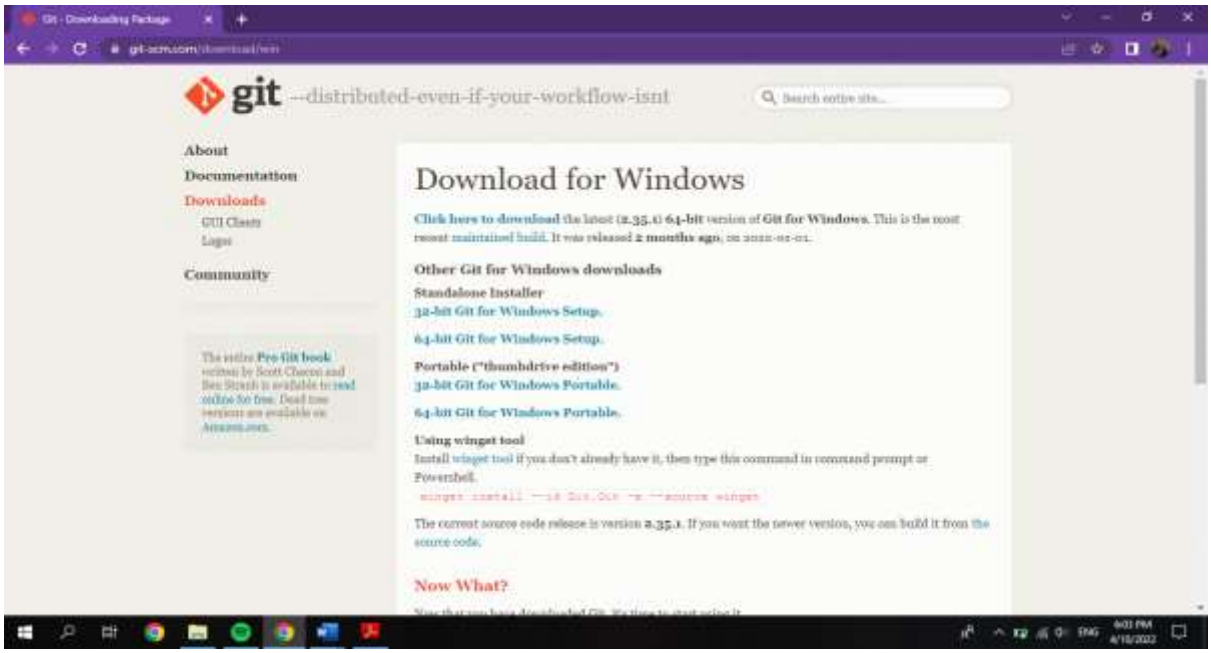
Experiment - 1

Aim: Setting up the git client.

Git Installation: Download the Git installation program (Windows, Mac, or Linux) from Git - Downloads (git-scm.com).



1. After opening this site, you have to select your operating system by clicking on it. Here I will show you the steps for the Windows operating system.



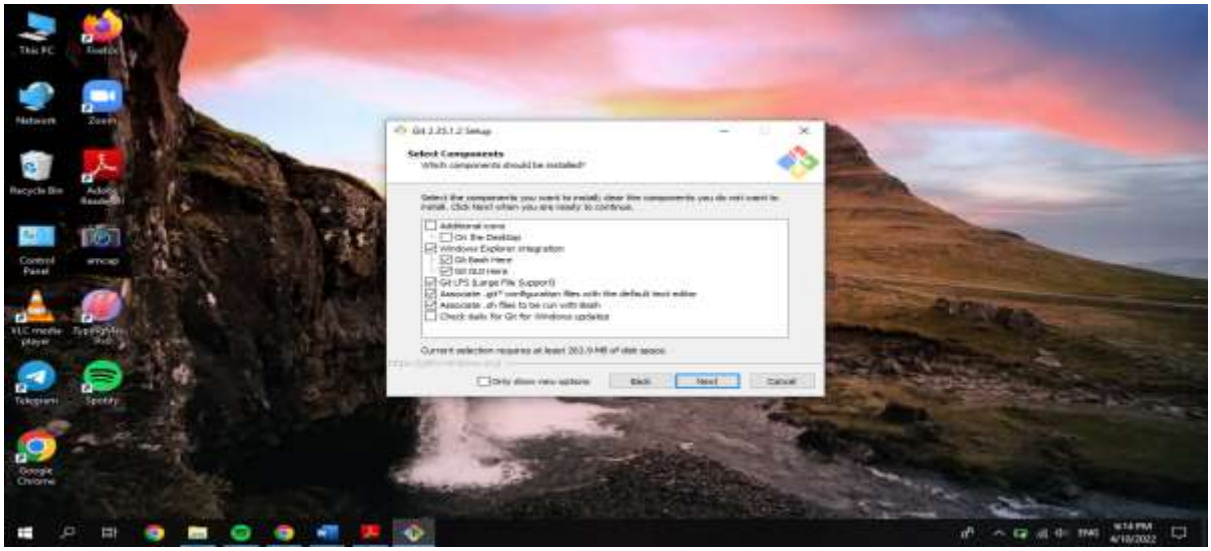
2. Now select the processor of the system you have. (Most of the system are now of 64-bits) After selecting the processor your download will start.

3. Now you have to open this folder.

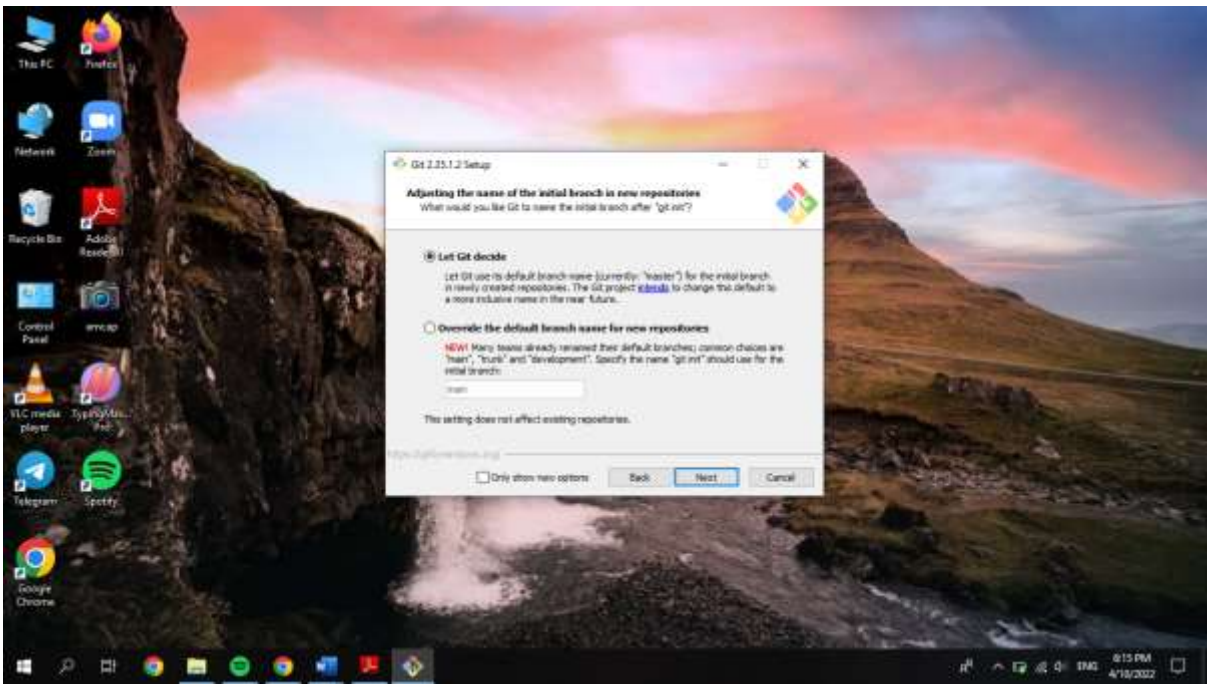
4. After opening you will given a notification “Do you want to allow this app to make changes in your PC”

5. Click Next

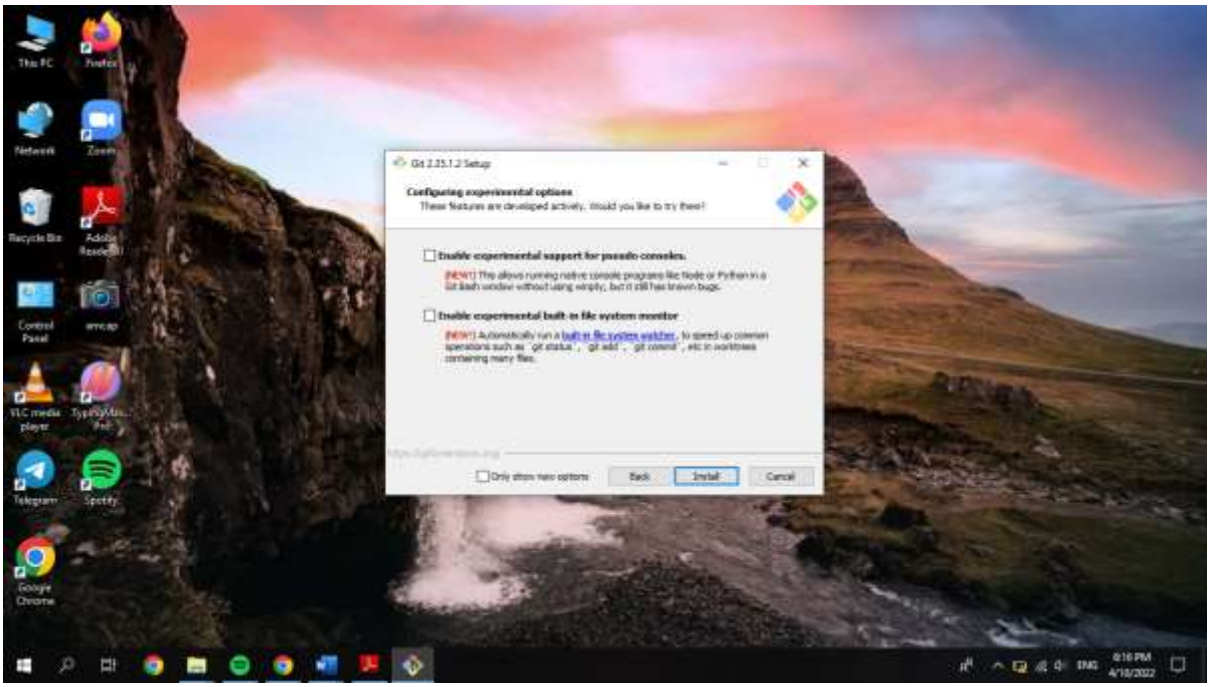
6. Click Next



7. Click Next



8. Click on Install



9. Click on Finish after the installation is finished.
10. The installation of the git is finished and now we have to setup git client and GitHub account.

Experiment - 2

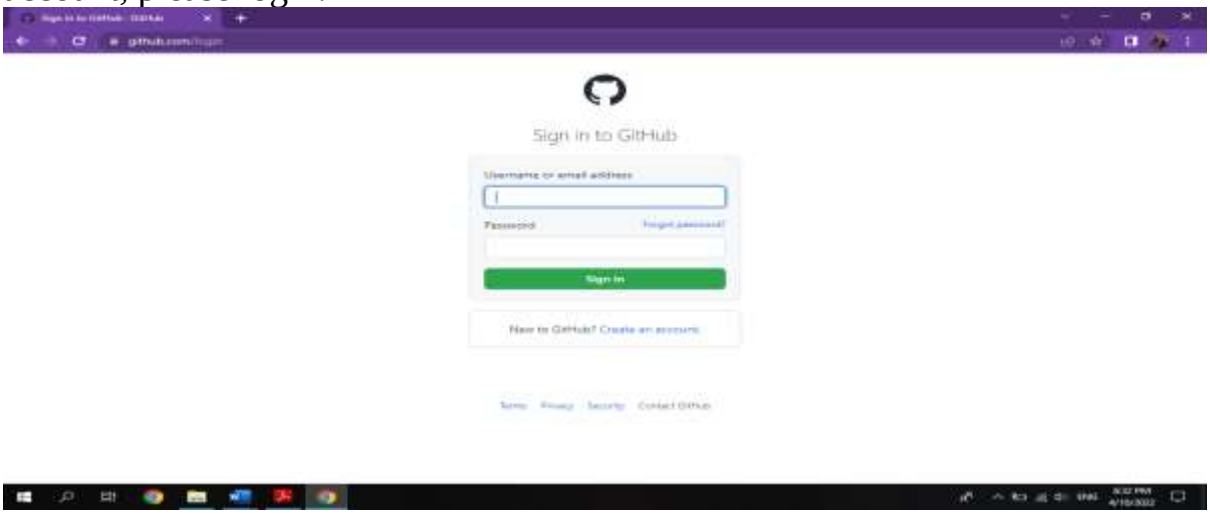
Aim: Setting up GitHub Account

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.

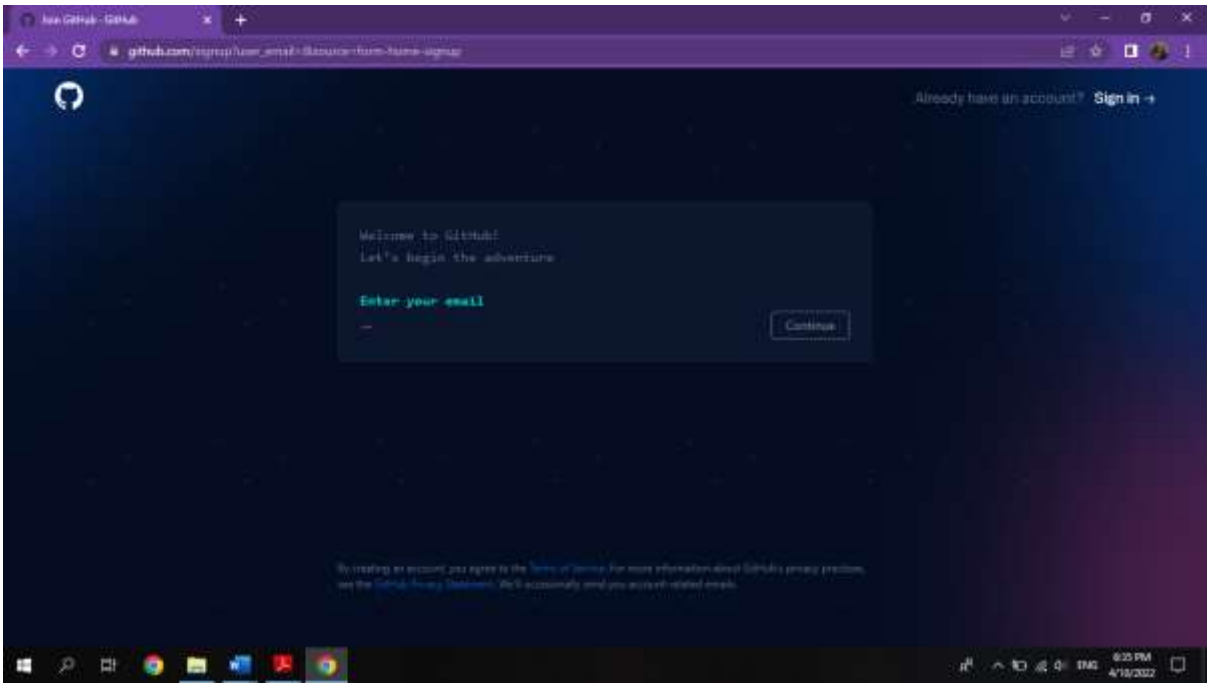
There are several types of accounts on GitHub. Every person who uses GitHub has their own user account, which can be part of multiple organisations and teams. Your user account is your identity on GitHub.com and represents you as an individual.

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

2. Click on Create an account if you are a new user or if you have already an account, please login.



3. After Clicking on create a new account you will be redirected to a new page where you have to enter your email id which you want to use for your account. Now enter your password you want to create for your GitHub account. After that you will be asked to enter your username.



4. Now Click on Create Account.
5. Verify it from your email and you are all set to go.

Experiment - 3

Aim: Program to generate logs

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.

Git Init:

The git init is **one way to start a new project with Git**. To start a repository, use either git init or git clone - not both. To initialize a repository, Git creates a hidden directory called .git . That directory stores all of the objects and refs that Git uses and creates as a part of your project's history.

Git Add:

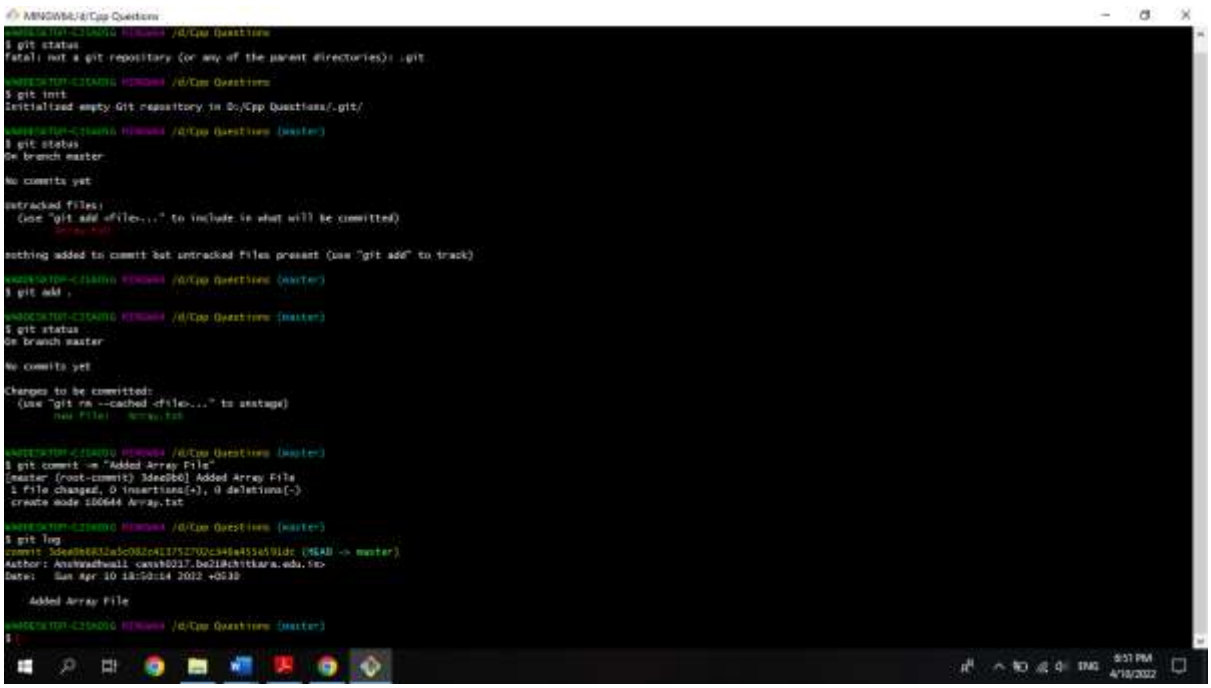
git add [filename] **selects that file, and moves it to the staging area, marking it for inclusion in the next commit**. You can select all files, a directory, specific files, or even specific parts of a file for staging and commit.

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.

Git Log:

Git log is **a utility tool to review and read a history of everything that happens to a repository**. Multiple options can be used with a git log to make history more specific. Generally, the git log is a record of commits.

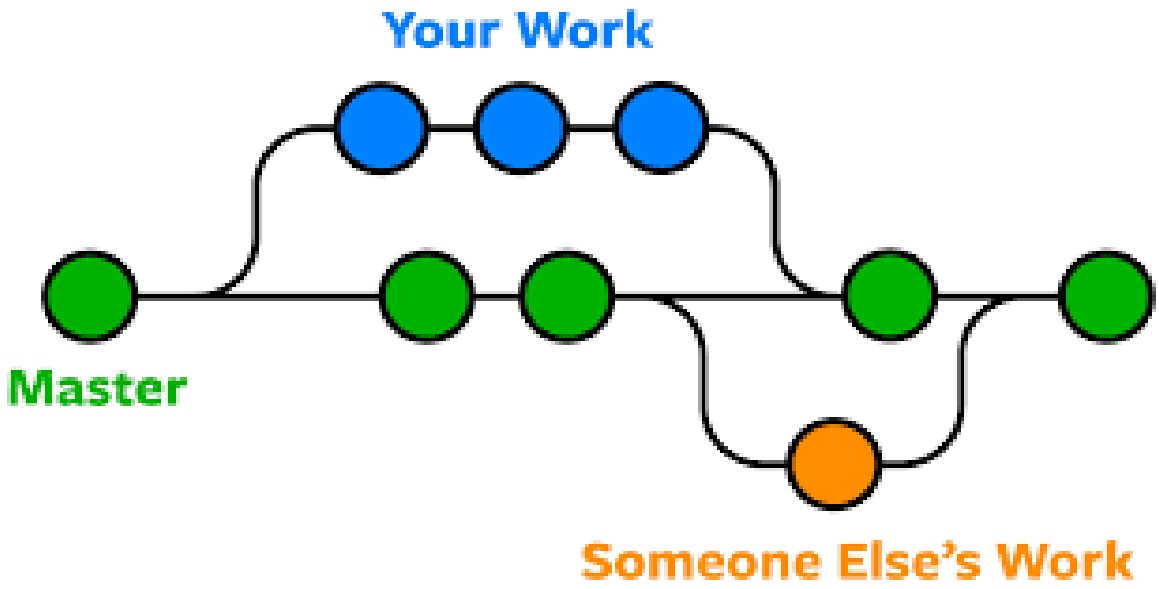

MINGW64:/d/Cpp Questions
$ git status
fatal: not a git repository (or any of the parent directories): .git
$ git init
Initialized empty Git repository in D:/Cpp Questions/.git/
$ git status
On branch master
No commits yet
Untracked files:
(use "git add <file>..." to include in what will be committed)
nothing added to commit but untracked files present (use "git add" to track)
$ git add .
$ git status
On branch master
No commits yet
Changes to be committed:
(use "git rm --cached <file>..." to unstage)
new file: Array.txt
$ git commit -m "Added Array File"
[master (root-commit) 3dee0b0] Added Array File
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 Array.txt
$ git log
commit 3dee0b0832a5c082e41375270dcb49a4554591dc (HEAD -> master)
Author: Anshnadhwani <ansh0217.be21@chitkara.edu.in>
Date: Sun Apr 10 18:50:14 2022 +0530
Added Array File
ENG
8:51 PM
4/10/2022

Experiment – 4

Aim: Create and visualize branches in Git

Git Branches:

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.



Let us see the command of it:
Firstly, add a new branch, let us suppose the branch name is activity1.
For this use command →
git branch name [adding new branch**]**
git branch [use to see the branch's names**]**
git checkout branch name [use to switch to the given branch**]**

MINGW64:/d/Cpp Questions

```
WA@DESKTOP-C2SA05G MINGW64 /d/Cpp Questions (master)
$ git branch
* master

WA@DESKTOP-C2SA05G MINGW64 /d/Cpp Questions (master)
$ git branch main

WA@DESKTOP-C2SA05G MINGW64 /d/Cpp Questions (master)
$ git branch
  main
* master

WA@DESKTOP-C2SA05G MINGW64 /d/Cpp Questions (master)
$ git checkout main
Switched to branch 'main'

WA@DESKTOP-C2SA05G MINGW64 /d/Cpp Questions (main)
$ git branch
* main
  master

WA@DESKTOP-C2SA05G MINGW64 /d/Cpp Questions (main)
$ ls
Array.txt  Loops.txt

WA@DESKTOP-C2SA05G MINGW64 /d/Cpp Questions (main)
$
```

In this you can see that firstly ‘git branch’ shows only one branch in green colour but when we add a new branch using ‘git branch main’, it shows 2 branches but the green colour and star is on master. So, we have to switch to main by using ‘git checkout main’. If we use ‘git branch’, now you can see that the green colour and star is on main. It means you are in main branch and all the data of master branch is also on activity1 branch.

Now add a new file in activity1 branch, do some changes in file and commit the file.

MINGW64:/d/Cpp Questions

```
WA@DESKTOP-C2SA05G MINGW64 /d/Cpp Questions (main)
$ ls
Array.txt  Loops.txt  Patterns.txt

WA@DESKTOP-C2SA05G MINGW64 /d/Cpp Questions (main)
$
```

If we switched to master branch, ‘Patterns.txt’ file is not there. But the file is in main branch.

MINGW64:/d/Cpp Questions

```
WA@DESKTOP-C2SA05G MINGW64 /d/Cpp Questions (main)
$ git checkout master
Switched to branch 'master'

WA@DESKTOP-C2SA05G MINGW64 /d/Cpp Questions (master)
$ git branch
  main
* master

WA@DESKTOP-C2SA05G MINGW64 /d/Cpp Questions (master)
$ ls
Array.txt  Loops.txt

WA@DESKTOP-C2SA05G MINGW64 /d/Cpp Questions (master)
$
```

To add these files in master branch, we have to do merging. For this firstly switch to master branch and then use command:

Git merge branch name [use to merge branch**]**

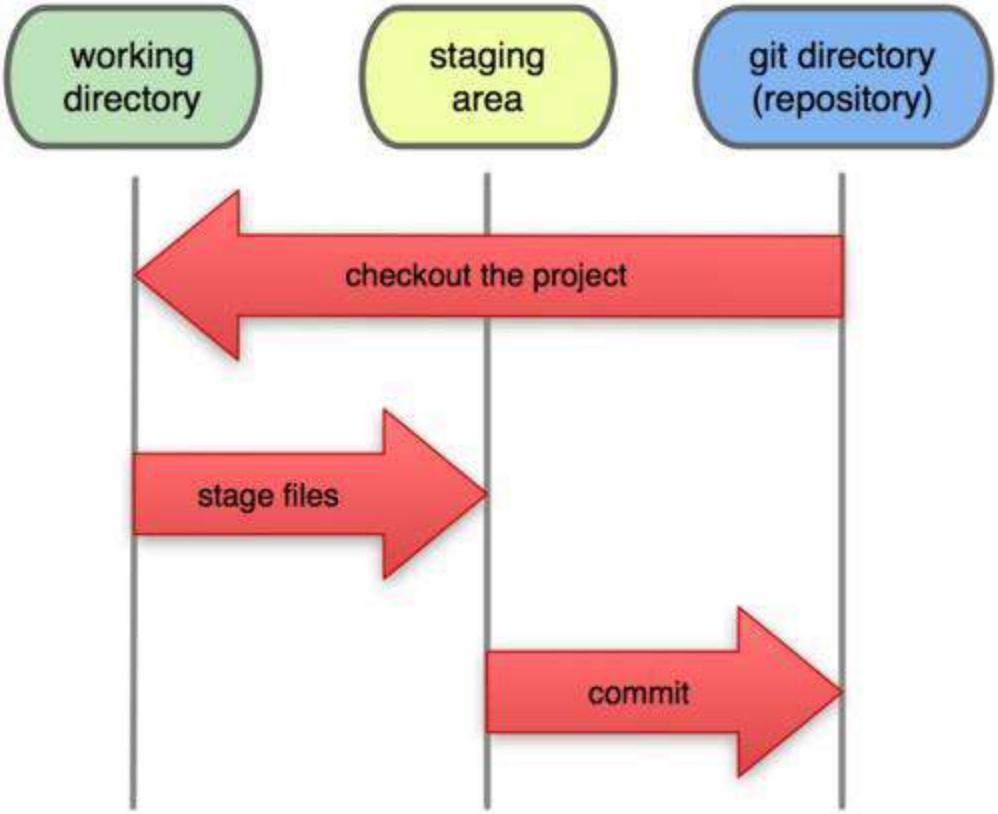
MINGW64:/d/Cpp Questions

```
WA@DESKTOP-C2SA05G MINGW64 /d/Cpp Questions (master)
$ git branch
  main
* master

WA@DESKTOP-C2SA05G MINGW64 /d/Cpp Questions (master)
$ ls
Array.txt  Loops.txt

WA@DESKTOP-C2SA05G MINGW64 /d/Cpp Questions (master)
$ git merge main
Updating 578dd81..4d21dd0
Fast-forward
 Patterns.txt | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 Patterns.txt

WA@DESKTOP-C2SA05G MINGW64 /d/Cpp Questions (master)
$ git branch
  main
* master

WA@DESKTOP-C2SA05G MINGW64 /d/Cpp Questions (master)
$ ls
Array.txt  Loops.txt  Patterns.txt

WA@DESKTOP-C2SA05G MINGW64 /d/Cpp Questions (master)
$
```

Experiment – 5

Aim: Git Life Cycle Description

Now let's understand the three-stage architecture of Git:



Working Directory: This is the directory that we've initialized, and here all the changes are made to commit on GitHub.

Staging Area: This is where we first put out code or files of the working repository. The command that we use to stage code is, "git add --a", "git add File Name" or "git add -A".
In simple terms, staging means telling Git what files we want to commit (new untracked files, modified files, or deleted files).

Git directory(repository): This is where all the commits are stored whenever we make a commit. We can revert to an older version of or project using the "git checkout" command from this directory.

Some Important Commands

- **Git operations and commands:**

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**.

- 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"

- **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.