

EXPERIMENT NO. 1:

Aim: Setting up of Git Client

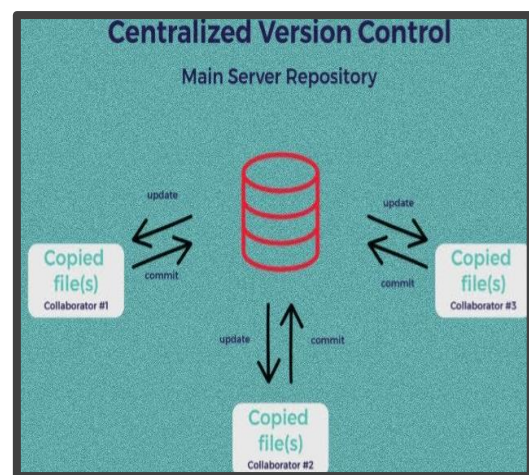
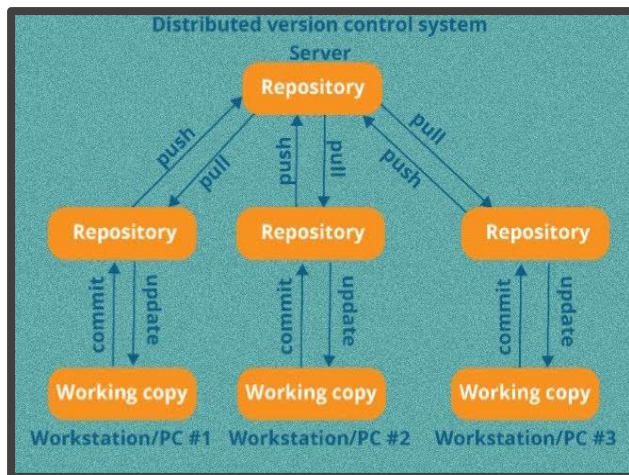
Theory:

What is Git?

Git is a free and open-source version control system used to handle small to very large projects efficiently. This is also used for tracking changes in any set of files and usually helps in coordinating work among members of a team. Hence, enables multiple developers to work together on non-linear development.

History of VCS: The very first Version Control System was created in 1972 at Bell Labs where they also developed LUNIX. The first one was called SCCS (Source Code Control System). It was available only for LUNIX and only worked with Source Code files. Some types of Version Control Systems are:

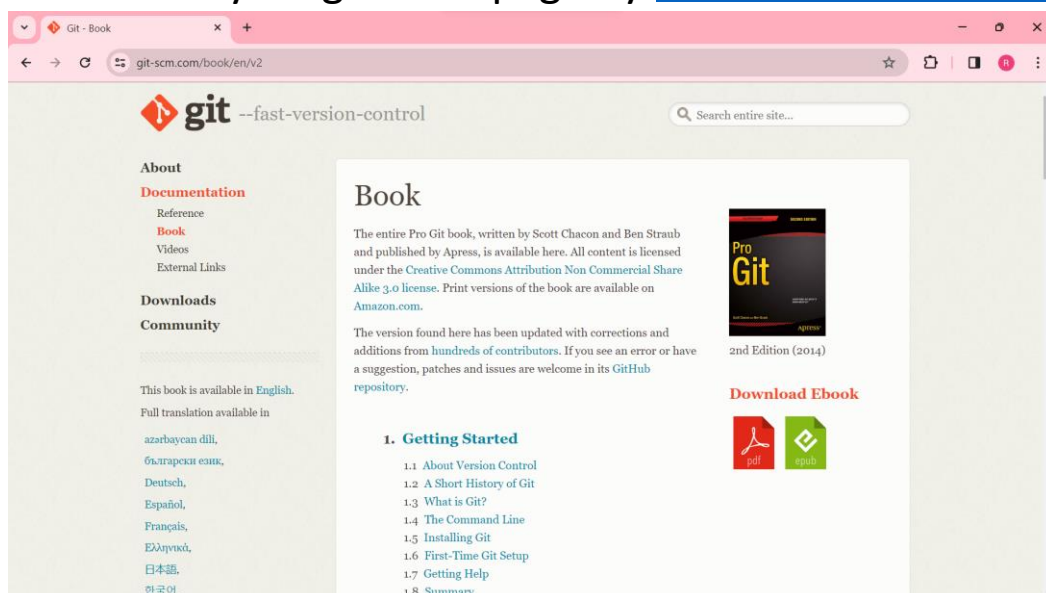
- **Local VCS:** No internet is needed because it uses a database to keep and track of files.
- **Centralized VCS:** Centralized version control systems are based on the idea that there is a single “central” copy of your project somewhere (probably on a server), and programmers will “commit” their changes to this central copy. This simply means recording the change in the central system (OS).
- **Distributed VCS:** A type of version control where the complete codebase including its full version history is mirrored on every developer's computer.



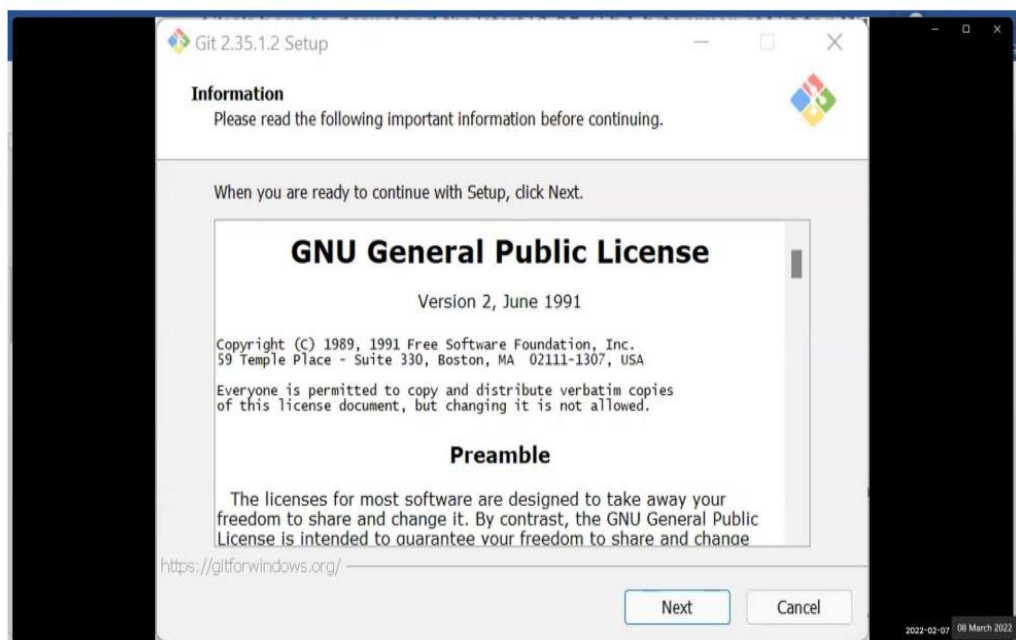
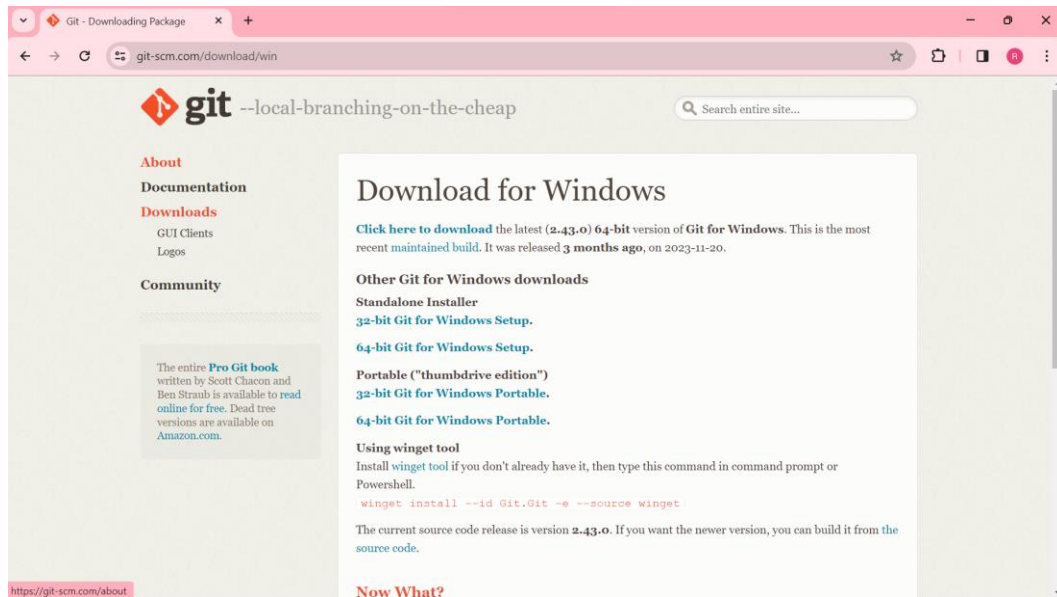
How to install GIT on Windows?

There are many ways to install Git on Windows. The most official build is available for download on the Git website. Go to <https://git-scm.com/download/win> and after a few settings the download will start automatically.

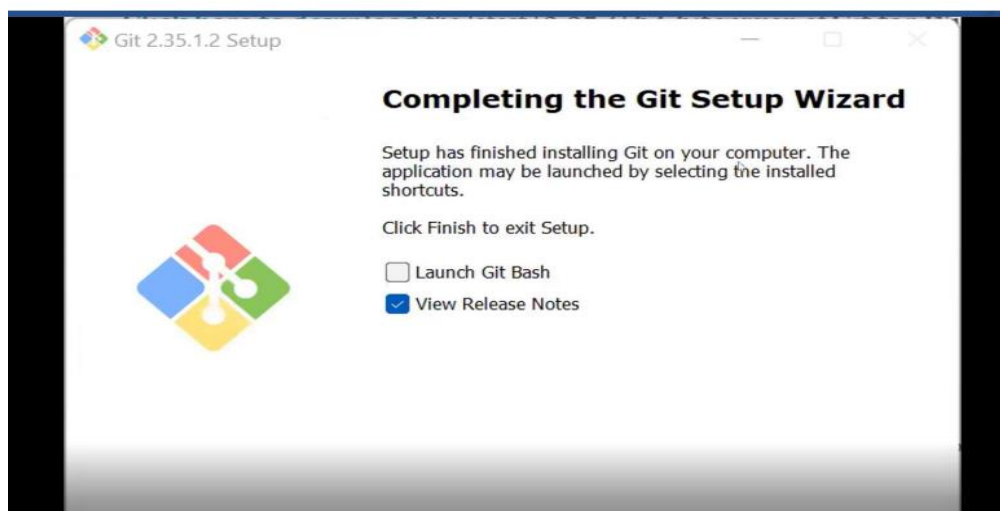
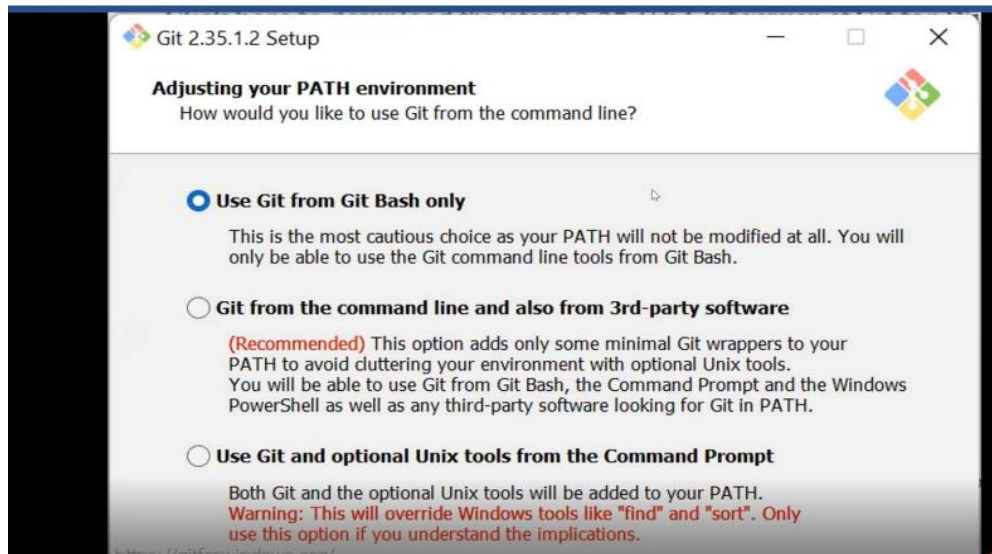
- Visit directly on git book page by <https://git-scm.com/book/en/v2>



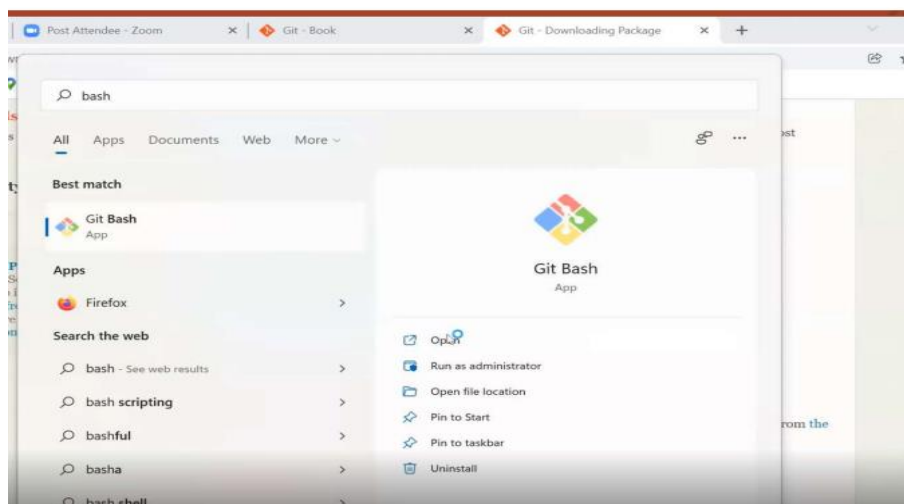
- Then click on Installation Git and click on whatever system you want, available are three- Windows, Apple and Linux.

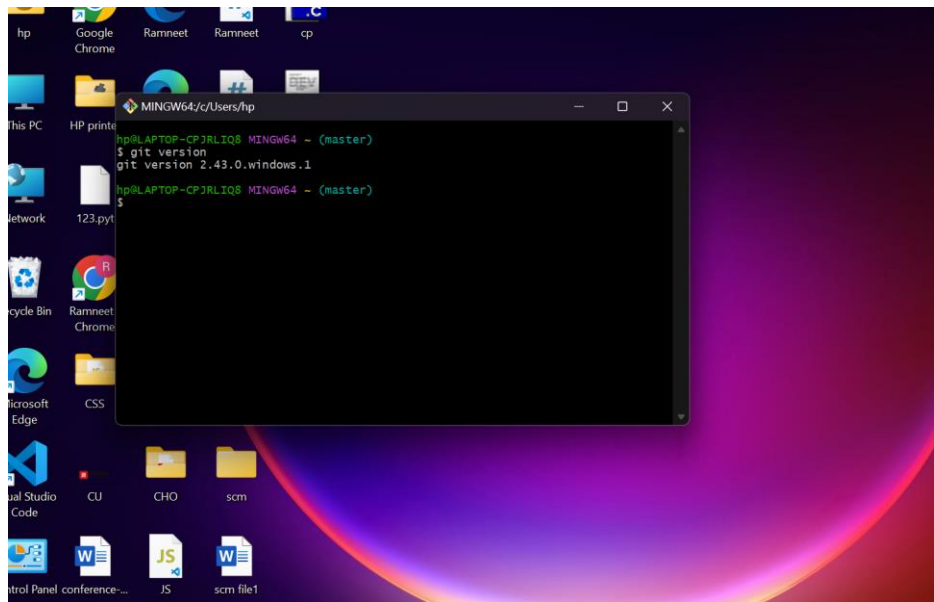


- After some more simple and easy settings and choosing your favourable environment and doing some SSH settings, it finally starts exporting the files in system and completes the Git hub wizard.



- Git bash got installed in system and seemed and opened on clicking seems of like:





You can also check the version of installed software by checking git version.

EXPERIMENT NO. 2:

Aim: Setting up GitHub Account

Theory:

What is GitHub?

GitHub is a code hosting platform for version control and collaboration. GitHub is a development platform inspired by the way you work. From open source to business, we can host and review code, manage projects, and build software alongside 36 million developers.

Advantages:

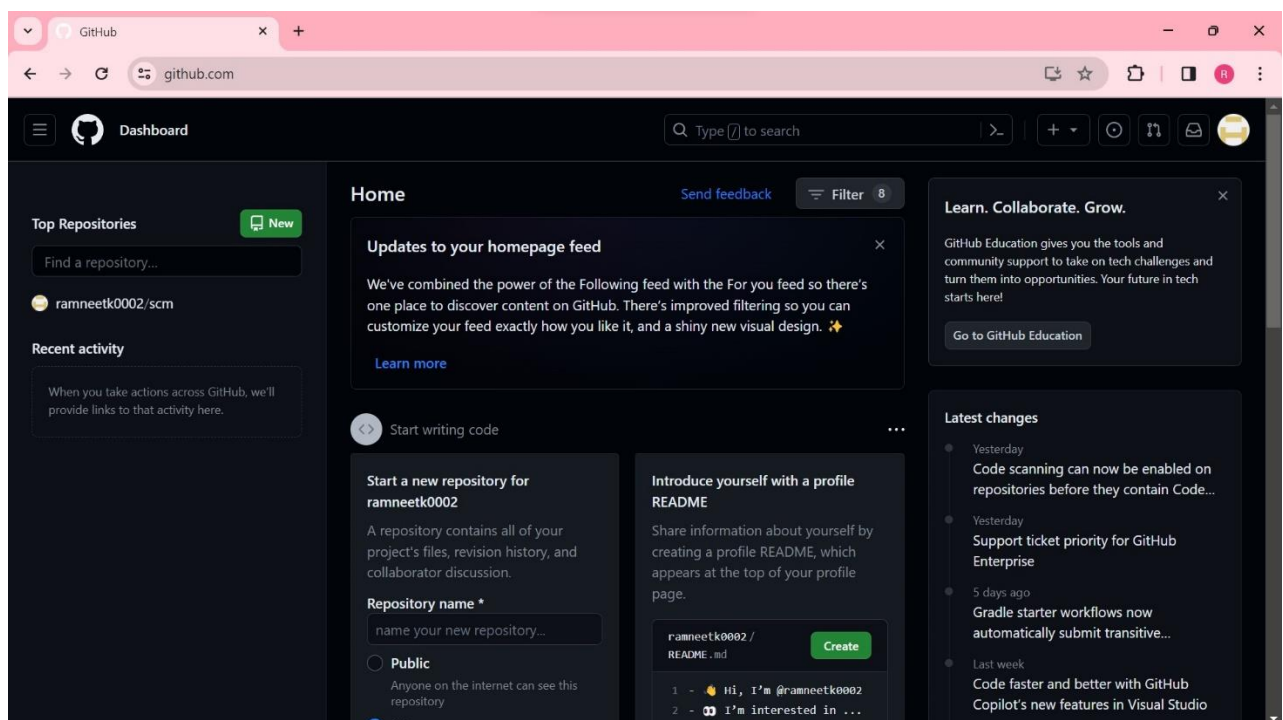
- Documentation.
- Showcase your work.
- Markdown.
- GitHub is a repository.
- Track changes in your code across versions.
- Integration options.

Procedure:

Search about GitHub: <https://github.com/signup>



By signing up for git you must remember your email and pass phases or password. For a new user, you must add your email and click on Sign up for GitHub. Otherwise click on Sign In at the top right corner.



For linking Git Hub with Git Bash:

Username-

`git config -- user.name "username in github"`

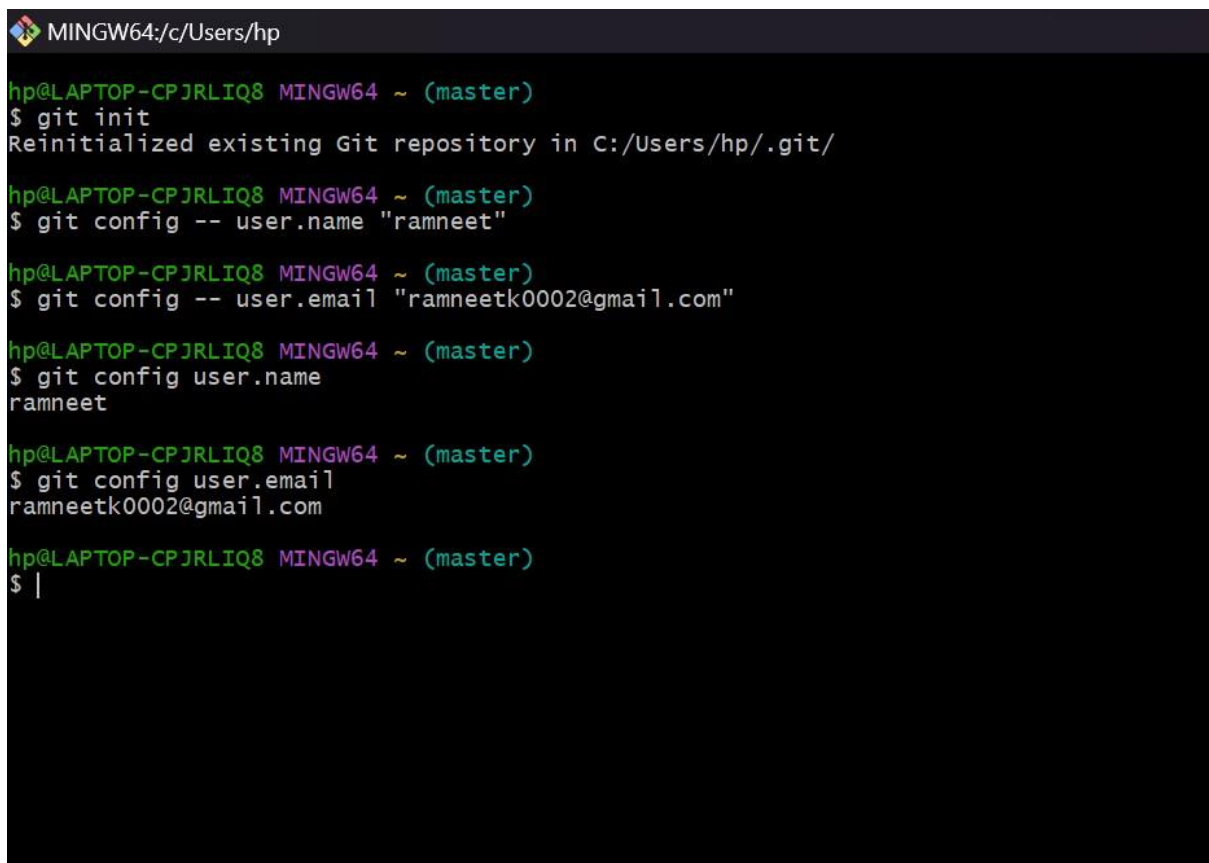
Email-

`git config -- user.email "your email in github"`

Check Username & Email:

`git config user.name`

`git config user.email`



```
MINGW64:/c/Users/hp
hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ git init
Reinitialized existing Git repository in C:/Users/hp/.git/
hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ git config -- user.name "ramneet"
hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ git config -- user.email "ramneetk0002@gmail.com"
hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ git config user.name
ramneet
hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ git config user.email
ramneetk0002@gmail.com
hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ |
```


EXPERIMENT NO. 3:

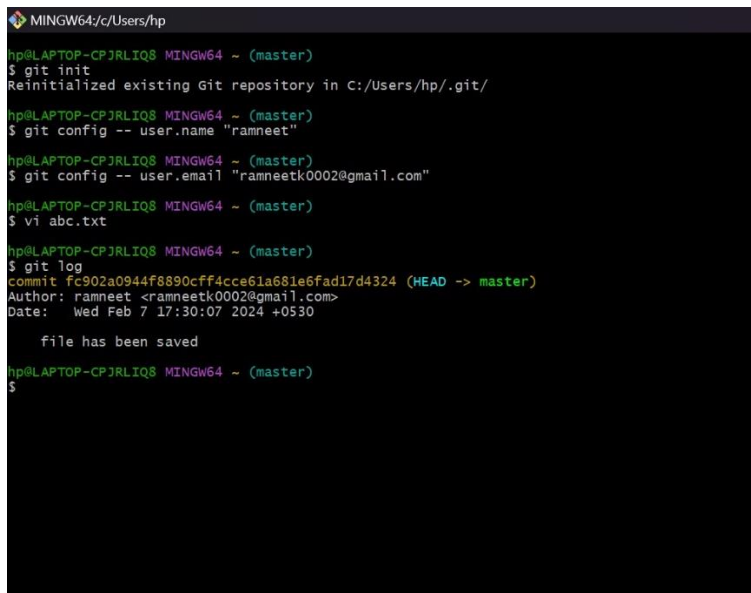
Aim: Generate Logs on Git Hub

Theory:

Git Logs: The git log command shows a list of all the commits made to a repository. You can see the hash of each Git commit, the message associated with each commit, and more metadata. This command is basically used for displaying the history of a repository.

Why do we need logs?

Git log is a utility tool to review and read a history of everything that happens to a repository. Anything we change at what time, by which log, everything is getting recorded in git logs.

A screenshot of a terminal window with a dark background. The window title is 'MINGW64/c/Users/hp'. The terminal shows the following commands and output:

```
hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ git init
Reinitialized existing Git repository in C:/Users/hp/.git/

hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ git config -- user.name "ramneet"

hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ git config -- user.email "ramneetk0002@gmail.com"

hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ vi abc.txt
file has been saved

hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ git log
commit fc902a0944f8890cff4cce61a681e6fad17d4324 (HEAD -> master)
Author: ramneet <ramneetk0002@gmail.com>
Date:   Wed Feb 7 17:30:07 2024 +0530
```

You can use command **git log** to access logs(every change you make with time and date).

EXPERIMENT NO. 04

Aim: Creating and Visualizing the Branches On Git Client

Theory:

How to create branches?

The main branch in which we are working is master branch. you can use the “git branch” command with the branch name and the commit SHA for the new branch.

1. For creating a new branch: git branch “name of the branch”.

```
file has been saved
hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ git branch b1
hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ git branch b2
hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$
```

2. To check how many branches we have:

```
hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ git branch b1
hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ git branch b2
hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ git branch
  b1
  b2
* master
hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$
```

As you can see here two branches are showing that I create- b1 and b2.

3. To change the present working branch: git checkout “name of the branch” and command to go back to the master directory:

```
hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ git branch b1

hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ git branch b2

hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ git branch
  b1
  b2
* master

hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ git checkout b2
Switched to branch 'b2'
M   abc.txt

hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (b2)
$ git checkout master
Switched to branch 'master'
M   abc.txt

hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ |
```

Here, you see by using checkout command we can switch branches and from branches to even master branch too.

Visualizing branches:

for visualizing, we have to create a new file in the branch that we made “b2” instead of the master branch. After this we have to do three step architecture that is working directory, staging area and git repository.

Firstly I've changed the branch from master to b2 that I previously made and after that I check git status. Now I add text in abc file (abc.txt) and use git add "file_name".

```
MINGW64/C:/Users/hp
hp@LAPTOP-CP3RLIQ8 MINGW64 - (master)
$ git init
Reinitialized existing Git repository in C:/Users/hp/.git/
hp@LAPTOP-CP3RLIQ8 MINGW64 - (master)
$ git config -- user.name "ramneet"
hp@LAPTOP-CP3RLIQ8 MINGW64 - (master)
$ git config -- user.email "ramneetk0002@gmail.com"
hp@LAPTOP-CP3RLIQ8 MINGW64 - (master)
$ vi abc.txt
hp@LAPTOP-CP3RLIQ8 MINGW64 - (master)
$ git log
commit fc902a0944f8890cff4cce61a681e6fad17d4324 (HEAD -> master)
Author: ramneet <ramneetk0002@gmail.com>
Date:   Wed Feb 7 17:30:07 2024 +0530

    file has been saved
hp@LAPTOP-CP3RLIQ8 MINGW64 - (master)
$ git branch b1
hp@LAPTOP-CP3RLIQ8 MINGW64 - (master)
$ git branch b2
hp@LAPTOP-CP3RLIQ8 MINGW64 - (master)
$ git branch
  b1
  b2
* master
hp@LAPTOP-CP3RLIQ8 MINGW64 - (master)
$ git checkout b2
Switched to branch 'b2'
M   abc.txt
hp@LAPTOP-CP3RLIQ8 MINGW64 - (b2)
$ git checkout master
Switched to branch 'master'
M   abc.txt
hp@LAPTOP-CP3RLIQ8 MINGW64 - (master)
$
```

EXPERIMENT NO. 05

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:

When a project is residing in our local system we don't know whether the project is tracked by Git or not. In any of the case, this project directory is called our Working directory.

Staging Area:

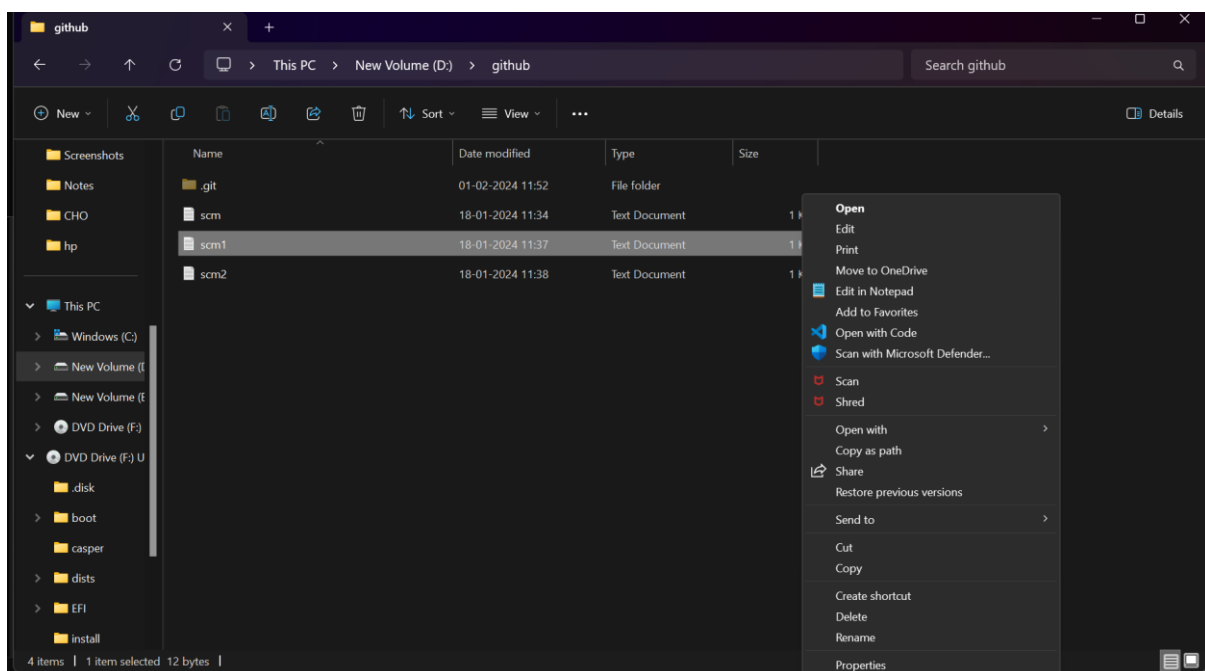
The staging area is like a rough draft space, it's where you can git add the version of a file or multiple files that you want to save in your next commit (in other words in the next version of your project)

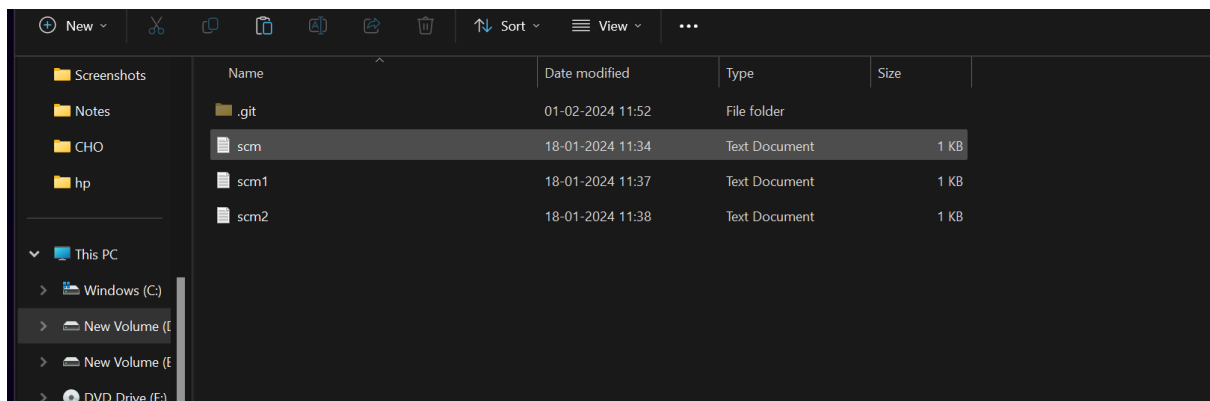
Git Directory:

The .git folder contains all information that is necessary for the project and all information relating commits, remote repository address, etc. It also contains a log that stores the commit history. This log can help you to roll back to the desired version of the code

Remote Repository: Remote repositories are hosted on a server that is accessible for all team members - most likely on the internet or on a local network. Assessable and reachable by all.

Screenshot:





Commands in github

- Git init

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

```
hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ git init
Reinitialized existing Git repository in C:/Users/hp/.git/
hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$
```

- Git config – user.name “ ”

You can change the name that is associated with your Git commits using the `git config` command. The new name you set will be visible in any future commits you push to GitHub from the command line. If you'd like to keep your real name private, you can use any text as your Git username.

```
hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ git config -- user.name "ramneet"

hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ git config -- user.name
ramneet
```

- Git config – user.email “ “

GitHub uses your commit email address to associate commits with your account on GitHub.com. You can choose the email address that will be associated with the commits you push from the command line as well as web-based Git operations you make.

```
hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ git config -- user.email "ramneetk0002@gmail.com"

hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ git config -- user.email
ramneetk0002@gmail.com

hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ |
```

- Vi

The default editor that comes with the UNIX operating system is called **vi** (visual editor). Using vi editor, we can edit an existing file or create a new file from scratch. we can also use this editor to just read a text file.

```
hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ vi abc.txt

hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ |
```

- Git add

The `git add` command adds a change in the working directory to the staging area. It tells Git that you want to include updates to a particular file in the next commit.

```
hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ git add abc.txt
hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
```

- `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.

```
hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ git status abc.txt
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   abc.txt
```

- [Git commit](#)

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.

```
hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ git config user.name
ramneet

hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ git config user.email
ramneetk0002@gmail.com

hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ |
```

- [Git log](#)

The git log command displays committed snapshots. It lets you list the project history, filter it, and search for specific changes.

- [Git branch](#)

Branches allow you to work on different parts of a project without impacting the main branch. When the work is complete, a branch can be merged with the main project. You can even switch between branches and work on different projects without them interfering with each other.

```
hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ git branch b1

hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ git branch b2

hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ git branch
b1
b2
* master
```

- Git checkout

The git checkout command lets you navigate between the branches created by git branch. Checking out a branch updates the files in the working directory to match the version stored in that branch, and it tells Git to record all new commits on that branch.

```
hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ git branch b1
hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ git branch b2
hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ git branch
  b1
  b2
* master
hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$ git checkout b2
Switched to branch 'b2'
M   abc.txt
hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (b2)
$ git checkout master
Switched to branch 'master'
M   abc.txt
hp@LAPTOP-CPJRLIQ8 MINGW64 ~ (master)
$
```