



SCM

TASK 1

BY-

AARCHI SHARMA

ROLL NO. 2110990012

INCLUDES-

1. Setting up of Git Client.
2. Setting up a GitHub account.
3. Generate logs.
4. Branches.
5. Git lifecycle description.

WHAT IS GIT

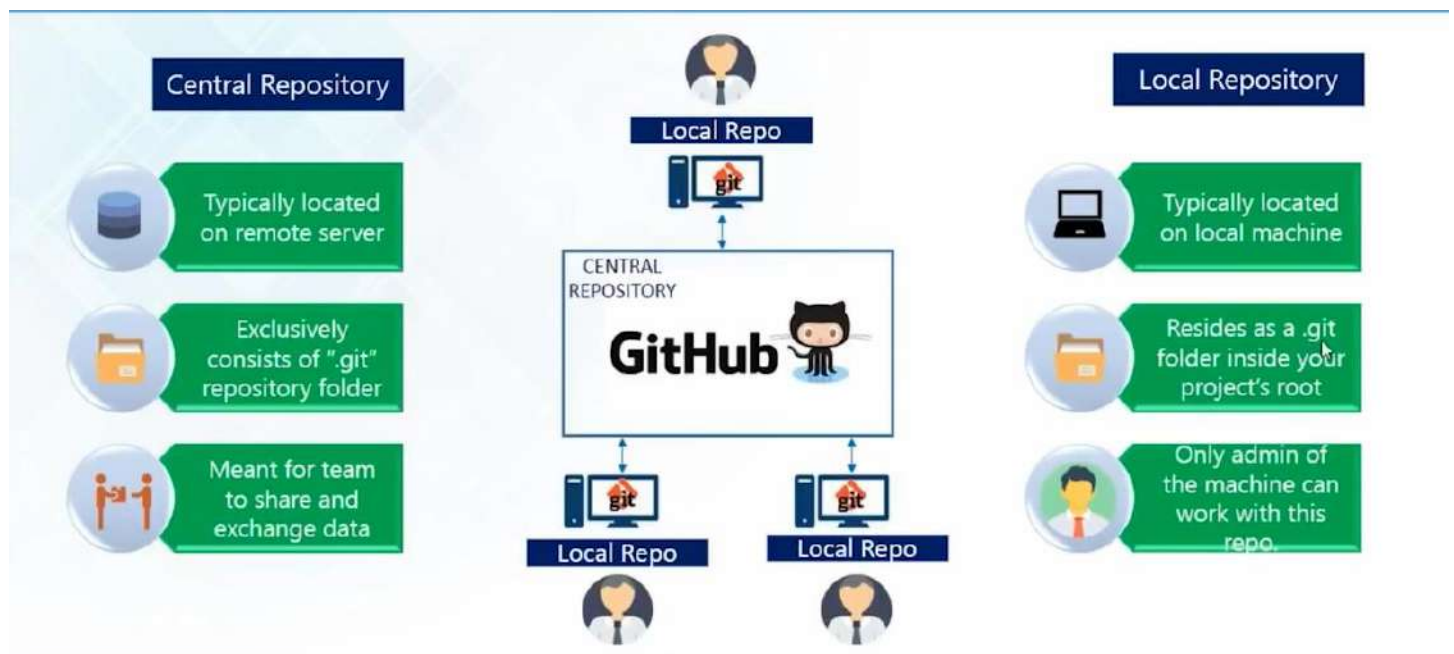
Git is a free and open-source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is easy to learn and has a tiny footprint with lightning-fast performance. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like cheap local branching, convenient staging areas, and multiple workflows.

WHAT IS GITHUB

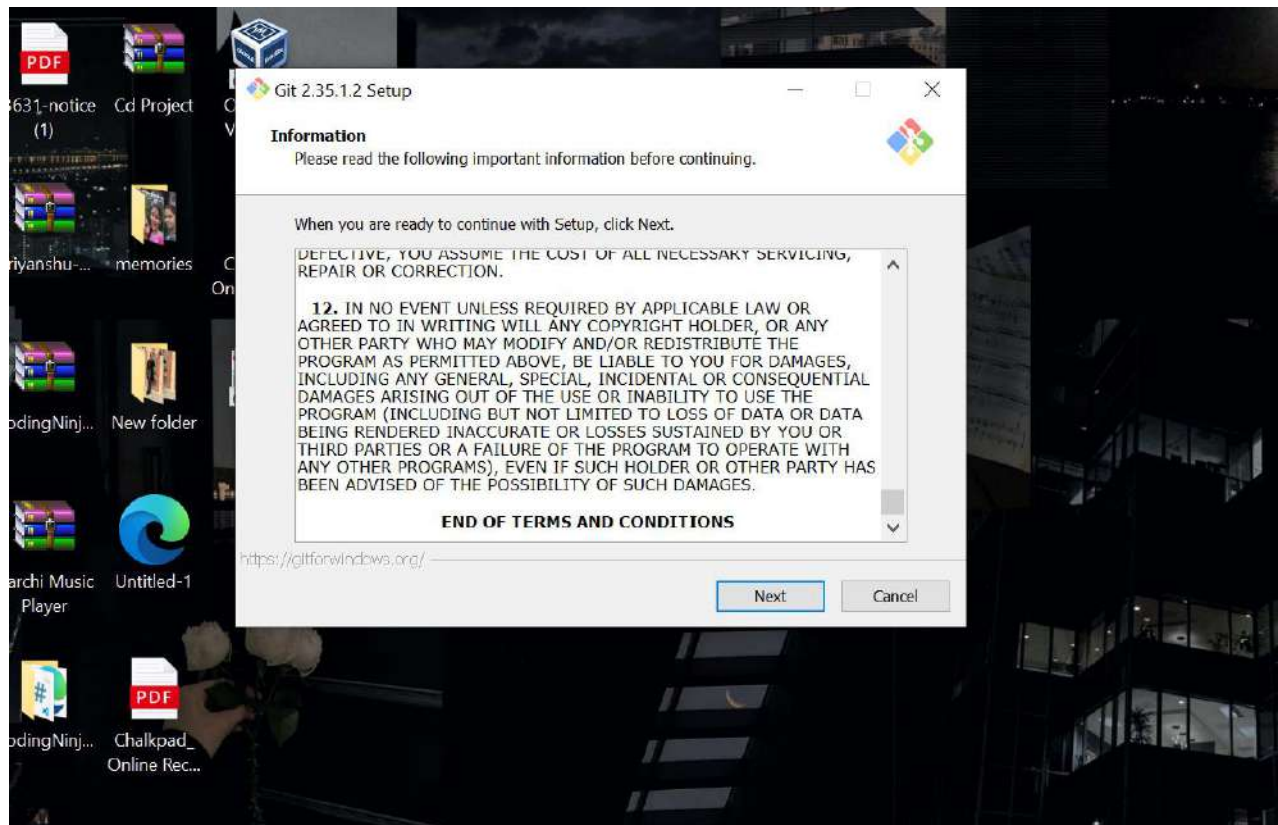
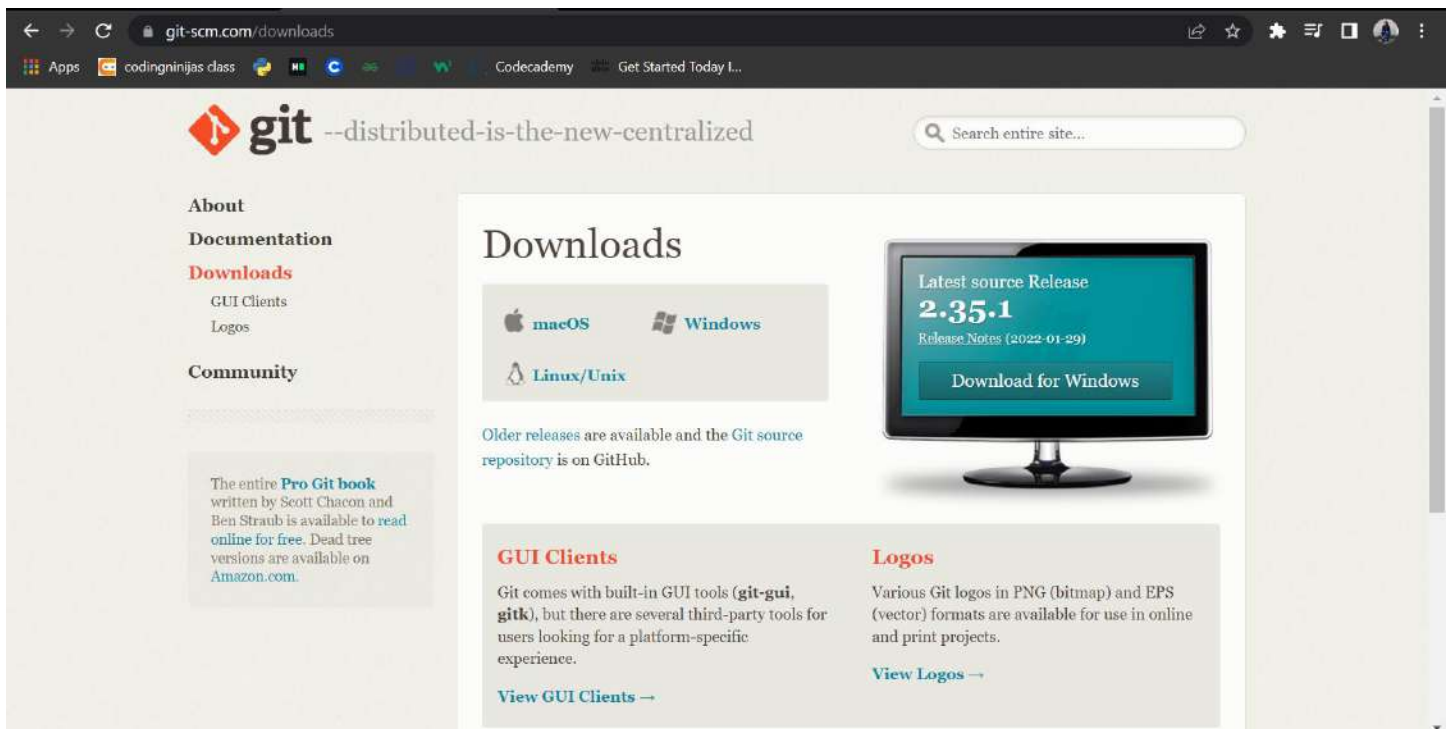
GitHub is a website and cloud-based service that helps developers store and manage their code, as well as track and control changes to their code.

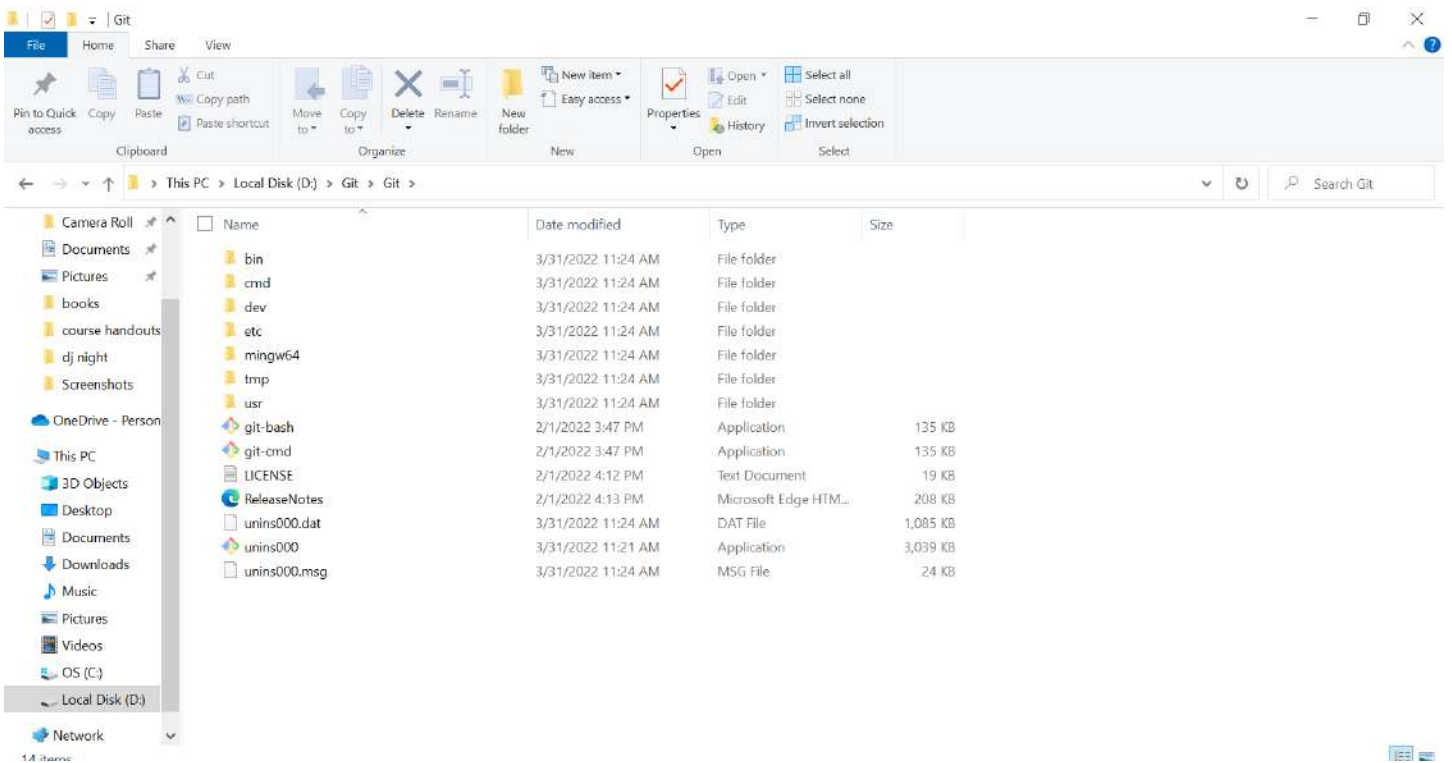
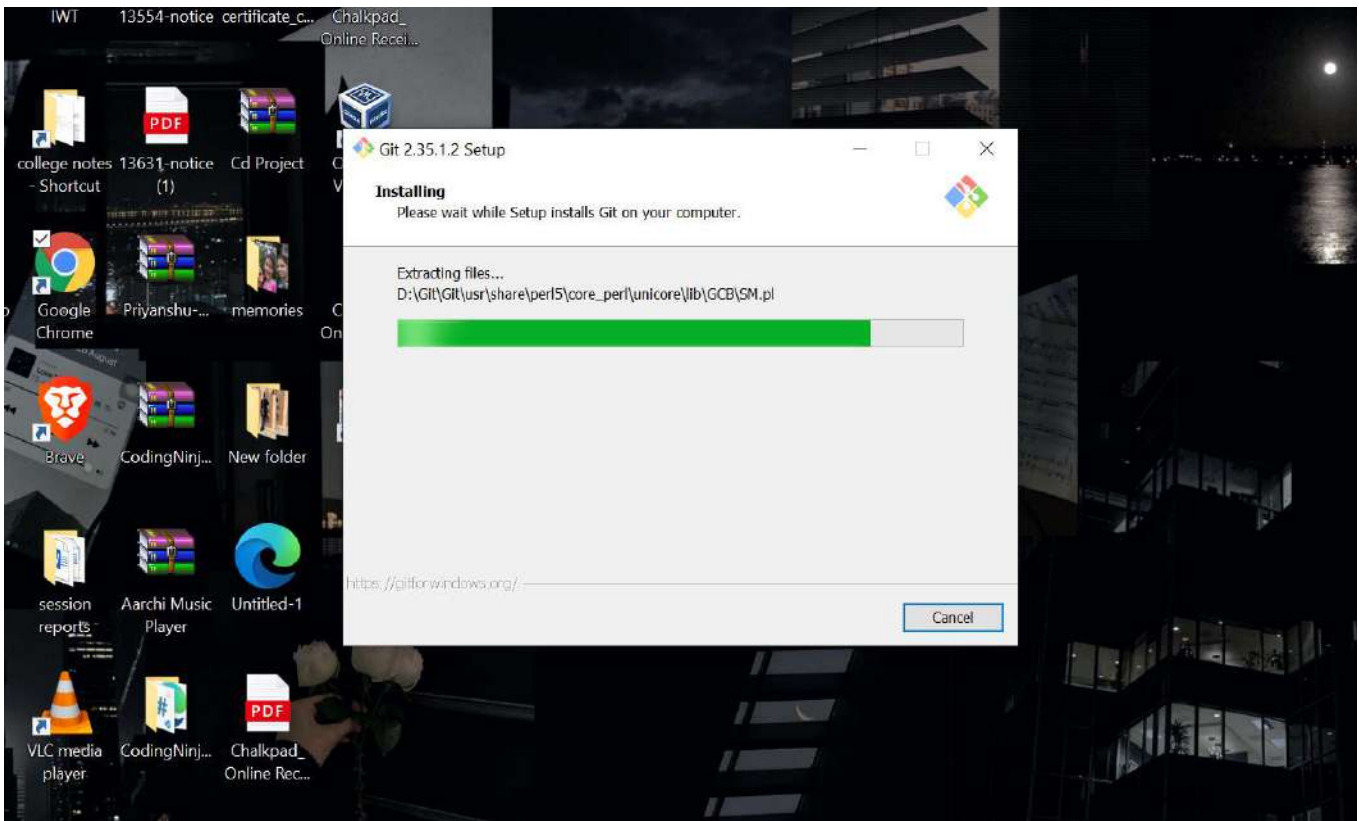
GIT VS GITHUB



INSTALLING GIT

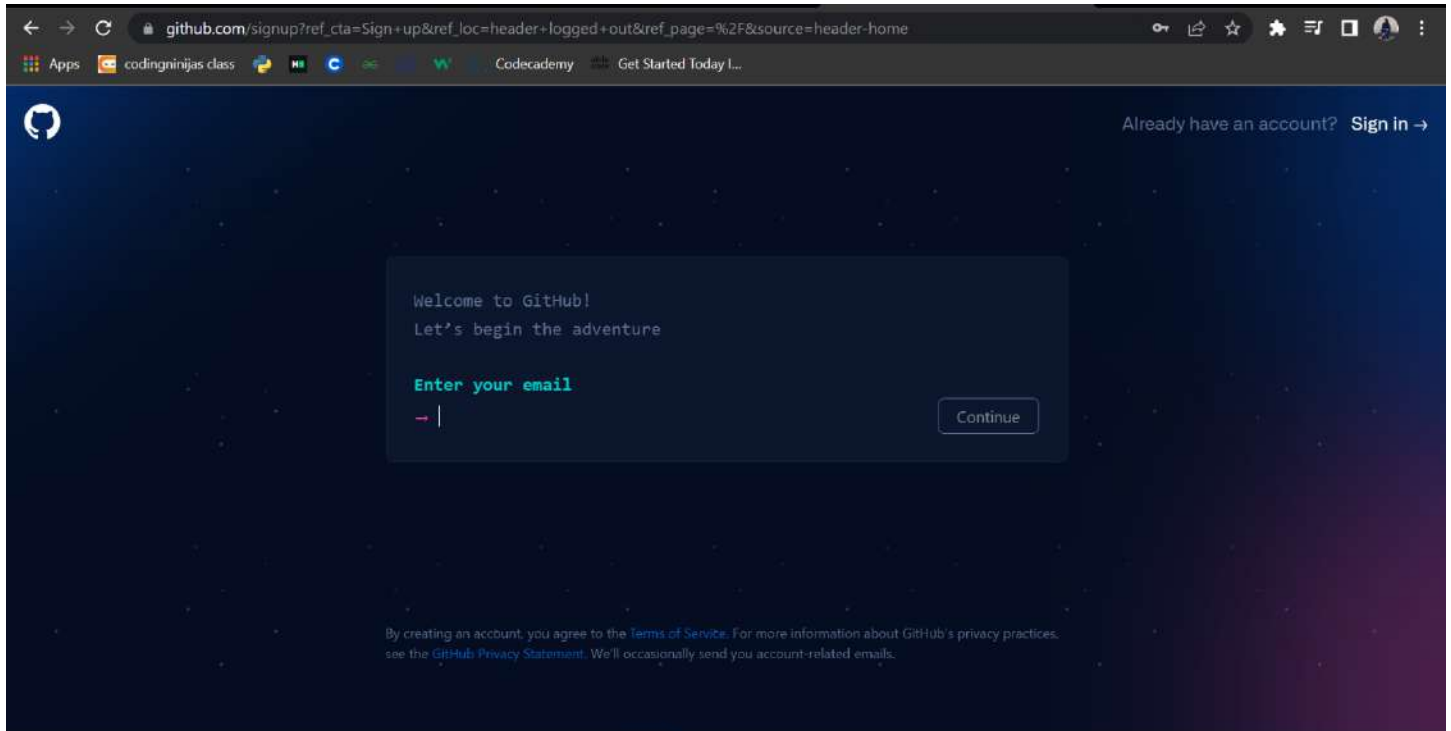
The most official build is available for download on the Git website. Just go to <https://git-scm.com/download/win> and the download will start automatically. Follow the steps listed further to proceed with the installation.





SETTING UP OF GITHUB ACCOUNT

You can set up a GitHub account by going onto the website <https://github.com/> and choosing the ‘Sign Up’ option. Enter the required details and you are done with creating account on GitHub.



The screenshot shows the GitHub sign-up page in a web browser. The address bar displays the URL: `github.com/signup?ref_cta=Sign+up&ref_loc=header+logged+out&ref_page=%2F&source=header-home`. The page features the GitHub logo in the top left and a link to "Sign In" in the top right. The main content area has a dark blue background with a central white box containing the following text:

Welcome to GitHub!
Let's begin the adventure

Enter your email

— |

Continue

At the bottom, a small disclaimer states: "By creating an account, you agree to the [Terms of Service](#). For more information about GitHub's privacy practices, see the [GitHub Privacy Statement](#). We'll occasionally send you account-related emails."

SETTING USER-NAME AND EMAIL

```
MINGW64/c/Users/ASUS
ASUS@Aarchi MINGW64 ~ (master)
$ git config --global user.name "aarchi103"

ASUS@Aarchi MINGW64 ~ (master)
$ git config --global user.email "aarchi0012.be21@chitkara.edu.in"

ASUS@Aarchi MINGW64 ~ (master)
$ git config
usage: git config [<options>]

Config file location
--global          use global config file
--system          use system config file
--local           use repository config file
--worktree        use per-worktree config file
-f, --file <file> use given config file
--blob <blob-id>  read config from given blob object

Action
--get             get value: name [value-pattern]
--get-all        get all values: key [value-pattern]
--get-regexp      get values for regexp: name-regex [value-pattern]
--get-urlmatch    get value specific for the URL: section[.var] URL
--replace-all    replace all matching variables: name value [value-pattern]
--add            add a new variable: name value
--unset          remove a variable: name [value-pattern]
--unset-all      remove all matches: name [value-pattern]
--rename-section  rename section: old-name new-name
--remove-section  remove a section: name
-l, --list        list all
--fixed-value     use string equality when comparing values to 'value-pattern'
-e, --edit        open an editor
--get-color       find the color configured: slot [default]
--get-colorbool   find the color setting: slot [stdout-is-tty]

Type
-t, --type <type> value is given this type
--bool            value is "true" or "false"
--int            value is decimal number
--bool-or-int     value is --bool or --int
--bool-or-str     value is --bool or string
--path           value is a path (file or directory name)
--expiry-date     value is an expiry date

Other
-z, --null        terminate values with NUL byte
--name-only       show variable names only
--includes        respect include directives on lookup
```

MAKING A DIRECTORY AND VARIOUS COMMANDS

```
MINGW64/d/g1/scmproject
ASUS@Aarchi MINGW64 /d/g1
$ git config --global user.name "aarchi103"

ASUS@Aarchi MINGW64 /d/g1
$ git config --global user.email "aarchi0012.be21@chitkara.edu.in"

ASUS@Aarchi MINGW64 /d/g1
$ mkdir scmproject

ASUS@Aarchi MINGW64 /d/g1
$ cd scmproject

ASUS@Aarchi MINGW64 /d/g1/scmproject
$ pwd
/d/g1/scmproject

ASUS@Aarchi MINGW64 /d/g1/scmproject
$ ls

ASUS@Aarchi MINGW64 /d/g1/scmproject
$ vi first.cpp

ASUS@Aarchi MINGW64 /d/g1/scmproject
$ ls
first.cpp

ASUS@Aarchi MINGW64 /d/g1/scmproject
$ cat first.cpp
#include <iostream>
using namespace std;
int main(){
cout<<"hello world"<<endl;
}

ASUS@Aarchi MINGW64 /d/g1/scmproject
$
```


mkdir=> It is used for creating a directory using Git Bash.

cd command=>Both Bash and Windows console host have a cd command. cd is an acronym for 'Change Directory'. cd is invoked with an appended directory name. Executing cd will change the terminal sessions current working directory to the passed directory argument.

pwd=>The Bash command pwd is used to print the 'present working directory'. This is the folder or path that the current Bash session resides in.

ls=>The Bash command ls is used to 'list' contents of the current working directory.

git status=>The main tool you use to determine which files are in which state is the git status command. the command tells you which branch you're on and informs you that it has not diverged from the same branch on the server.

git add =>In order to begin tracking a new file, you use the command git add. git add is a multipurpose command — you use it to begin tracking new files, to stage files, and to do other things like marking merge-conflicted files as resolved. It may be helpful to think of it more as “add precisely this content to the next commit” rather than “add this file to the project”.

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. Prior to the execution of git commit, The git add command is used to promote or 'stage' changes to the project that will be stored in a commit. These two commands git commit and git add are two of the most frequently used.

git push=> When you have your project at a point that you want to share, you have to push it upstream. The command for this is simple: git push . If you want to push your master branch to your origin server (again, cloning generally sets up both of those names for you automatically), then you can run this to push any commits you've done back up to the server: \$
git push origin master


```
MINGW64/d/g1/scmpproject
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    first.cpp

nothing added to commit but untracked files present (use "git add" to track)

ASUS@Aarchi MINGW64 /d/g1/scmpproject (master)
$ git add first.cpp
warning: LF will be replaced by CRLF in first.cpp.
The file will have its original line endings in your working directory

ASUS@Aarchi MINGW64 /d/g1/scmpproject (master)
$ git status
On branch master

No commits yet

changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   first.cpp

ASUS@Aarchi MINGW64 /d/g1/scmpproject (master)
$ git commit -m "first commit | repo demo for class"
[master (root-commit) b91cfad] first commit | repo demo for class
1 file changed, 5 insertions(+)
create mode 100644 first.cpp

ASUS@Aarchi MINGW64 /d/g1/scmpproject (master)
$
```

GENERATING 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 which can be seen anytime.

```
ASUS@Aarchi MINGW64 /d/g1/scmpproject (feature)
$ git log --oneline
040a5c6 (HEAD -> feature) Added a display function
b91cfad (origin/master, master) first commit | repo demo for class
```

GIT 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.

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]

```
ASUS@Aarchi MINGW64 ~ (master)
$ cd D:

ASUS@Aarchi MINGW64 /d
$ cd g1

ASUS@Aarchi MINGW64 /d/g1
$ cd scmpproject

ASUS@Aarchi MINGW64 /d/g1/scmpproject (master)
$ git branch
* master

ASUS@Aarchi MINGW64 /d/g1/scmpproject (master)
$ git log --oneline
b91cfad (HEAD -> master, origin/master) first commit | repo demo for class

ASUS@Aarchi MINGW64 /d/g1/scmpproject (master)
$ ls
first.cpp

ASUS@Aarchi MINGW64 /d/g1/scmpproject (master)
$ git branch feature

ASUS@Aarchi MINGW64 /d/g1/scmpproject (master)
$ git branch
  feature
* master

ASUS@Aarchi MINGW64 /d/g1/scmpproject (master)
$ git checkout feature
Switched to branch 'feature'
M      first.cpp
```

```
$ git branch
* master

ASUS@Aarchi MINGW64 /d/g1/scmpproject (master)
$ git log --oneline
b91cfad (HEAD -> master, origin/master) first commit | repo demo for class

ASUS@Aarchi MINGW64 /d/g1/scmpproject (master)
$ ls
first.cpp

ASUS@Aarchi MINGW64 /d/g1/scmpproject (master)
$ git branch feature

ASUS@Aarchi MINGW64 /d/g1/scmpproject (master)
$ git branch
  feature
* master

ASUS@Aarchi MINGW64 /d/g1/scmpproject (master)
$ git checkout feature
Switched to branch 'feature'
M      first.cpp

ASUS@Aarchi MINGW64 /d/g1/scmpproject (feature)
$ git branch
* feature
  master

ASUS@Aarchi MINGW64 /d/g1/scmpproject (feature)
$ git log --oneline
b91cfad (HEAD -> feature, origin/master, master) first commit | repo demo for class
```

no changes added to commit (use "git add" and/or "git commit -a")

ASUS@Aarchi MINGW64 /d/g1/scmpproject (feature)
\$

ASUS@Aarchi MINGW64 /d/g1/scmpproject (feature)
\$ git add .
warning: LF will be replaced by CRLF in first.cpp.
The file will have its original line endings in your working directory

ASUS@Aarchi MINGW64 /d/g1/scmpproject (feature)
\$ git commit -m "Added a display function"
[feature 040a5c6] Added a display function
1 file changed, 4 insertions(+)

ASUS@Aarchi MINGW64 /d/g1/scmpproject (feature)
\$ git log --oneline
040a5c6 (HEAD -> feature) Added a display function
b91cfad (origin/master, master) first commit | repo demo for class

ASUS@Aarchi MINGW64 /d/g1/scmpproject (feature)
\$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

ASUS@Aarchi MINGW64 /d/g1/scmpproject (master)
\$ git log --oneline
b91cfad (HEAD -> master, origin/master) first commit | repo demo for class

ASUS@Aarchi MINGW64 /d/g1/scmpproject (master)
\$ vi first.cpp

ASUS@Aarchi MINGW64 /d/g1/scmpproject (function)
\$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

ASUS@Aarchi MINGW64 /d/g1/scmpproject (master)
\$ git checkout feature
Switched to branch 'feature'

ASUS@Aarchi MINGW64 /d/g1/scmpproject (feature)
\$ git push -u origin feature
fatal: unable to access 'https://github.com/aarchi103/g1.git/': could not resolve host: github.com

ASUS@Aarchi MINGW64 /d/g1/scmpproject (feature)
\$ git remote
origin

ASUS@Aarchi MINGW64 /d/g1/scmpproject (feature)
\$ git push -u origin feature
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 368 bytes | 368.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'feature' on GitHub by visiting:
remote: <https://github.com/aarchi103/g1/pull/new/feature>
remote:
To <https://github.com/aarchi103/g1.git>
* [new branch] feature -> feature
branch 'feature' set up to track 'origin/feature'.

GIT LIFECYCLE DESCRIPTION

It is important for us to have an abstract idea of the different stages of Git before going into more detailed understanding of Git.

Files in a **Git** project have various stages like **Creation, Modification, Refactoring,** and **Deletion** and so on.

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:

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.

