Subject Name: **Source Code Management**

Subject Code: **22CS003**

Session: **2024-25**

Department: **DCSE**



**Submitted To:**

**Submitted By:**

Khushal Sardana

2310990630

G23-A, 2nd Sem

Dr. Sharad Chauhan

Professor

Chitkara University PB

**Source Code Management File**

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

Subject Code: **22CS003**

Cluster: **BETA**

**Submitted By:**

**Name:** Khushal Sardana

**Roll no.:** 2310990630

**Group:** G23-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**

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No.** | **Program Title** | **Page No.** | **Signature** |
| 1. | Setting Up of Git Client | 4 |  |
| 2. | Setting Up GitHub Account | 10 |  |
| 3. | Generating Logs | 12 |  |
| 4. | Create and Visualize Branches | 16 |  |
| 5. | Git Lifecycle Description | 18 |  |
|  |  |  |  |
|  |  |  |  |

EXPERIMENT NO. 1:

**Aim: Setting up of Git Client**

***Theory:***

**What is Git?**

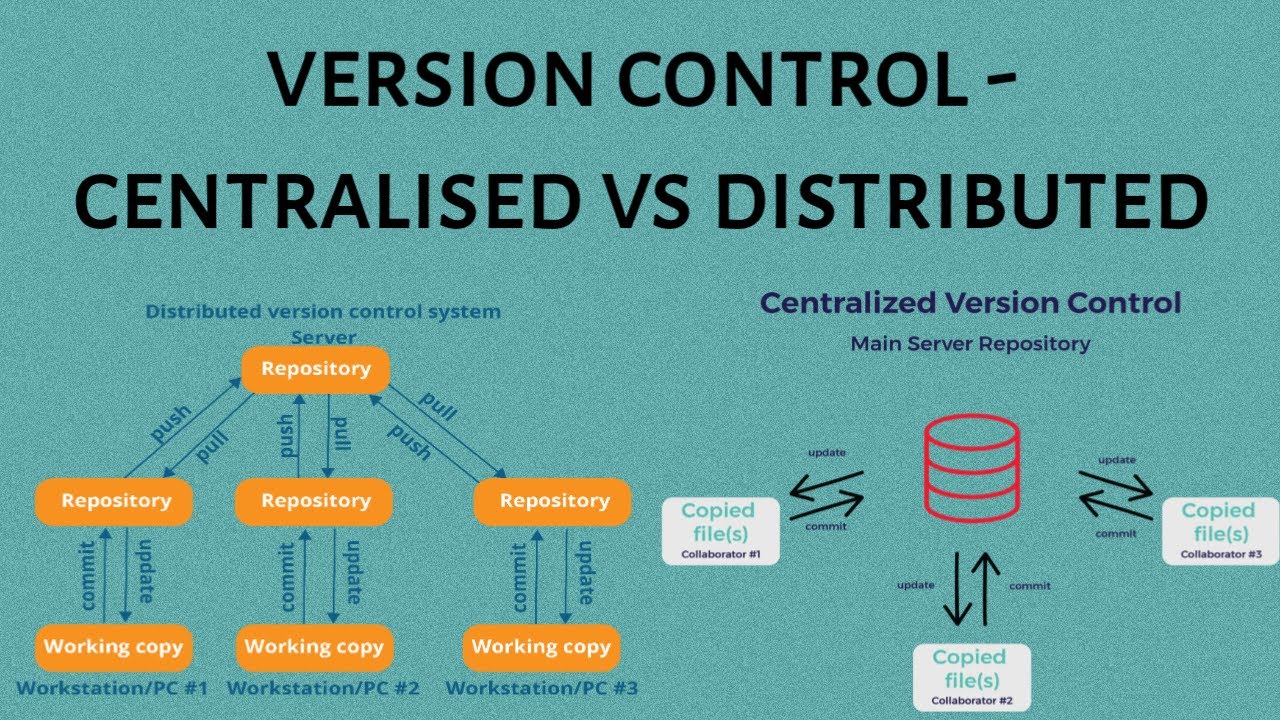
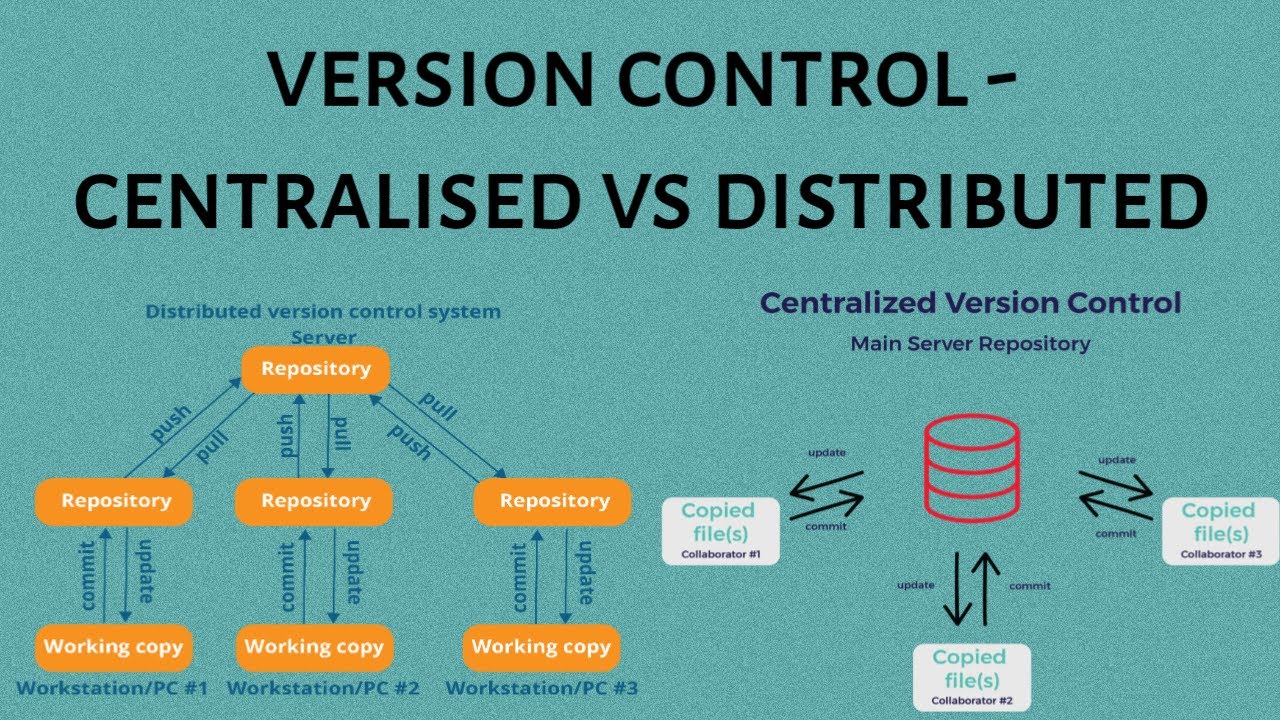
Git is a free and open-source version control system used to handle small to very large projects efficiently. This is also used for tracking changes in any set of files and usually helps in coordinating work among members of a team. Hence, enables multiple developers to work together on non-linear development.

**History of VCS:** The very first Version Control System was created in 1972 at Bell Labs where they also developed UNIX. The first one was called SCCS (Source Code Control System). It was available only for UNIX 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 ever0y developer's computer.

*Figure 1.1 Architecture of VCS*

**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://gitscm.com/download/win and after a few settings the download will start automatically.

1. Begin the installation of Git.

2. Download the Git installer for Windows.

3. Provide necessary information during the installation process.

4. Select the destination location where Git will be installed.

5. Choose the components to be installed (e.g., Git Bash, Git GUI).

6. Select the Start Menu folder where Git shortcuts will be placed.

7. Choose the default text editor to be used by Git .

8. Adjust the name of the initial branch in new repositories if needed.

9. Adjust your PATH environment to include Git if necessary.

10. Choose the SSH executable to use for Git operations.

11. Choose the HTTPS transport backend for Git.

12. Configure the line ending conversion settings based on your needs.

13. Configure the terminal emulator to use with Git Bash.

14. Choose the default behavior of Git pull (e.g., rebase, merge).

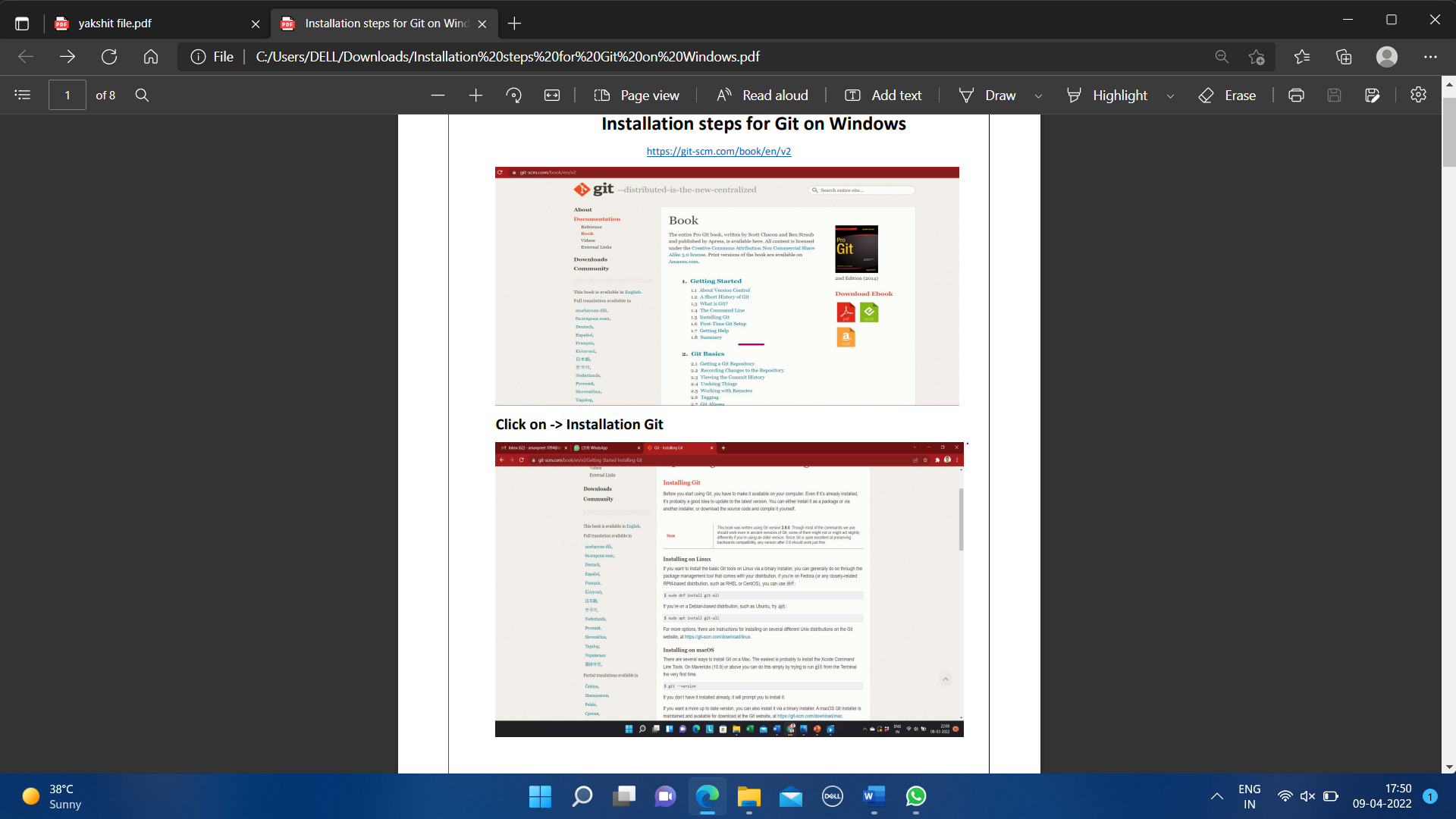
15. Choose a credential helper for Git (e.g., Git Credential Manager).

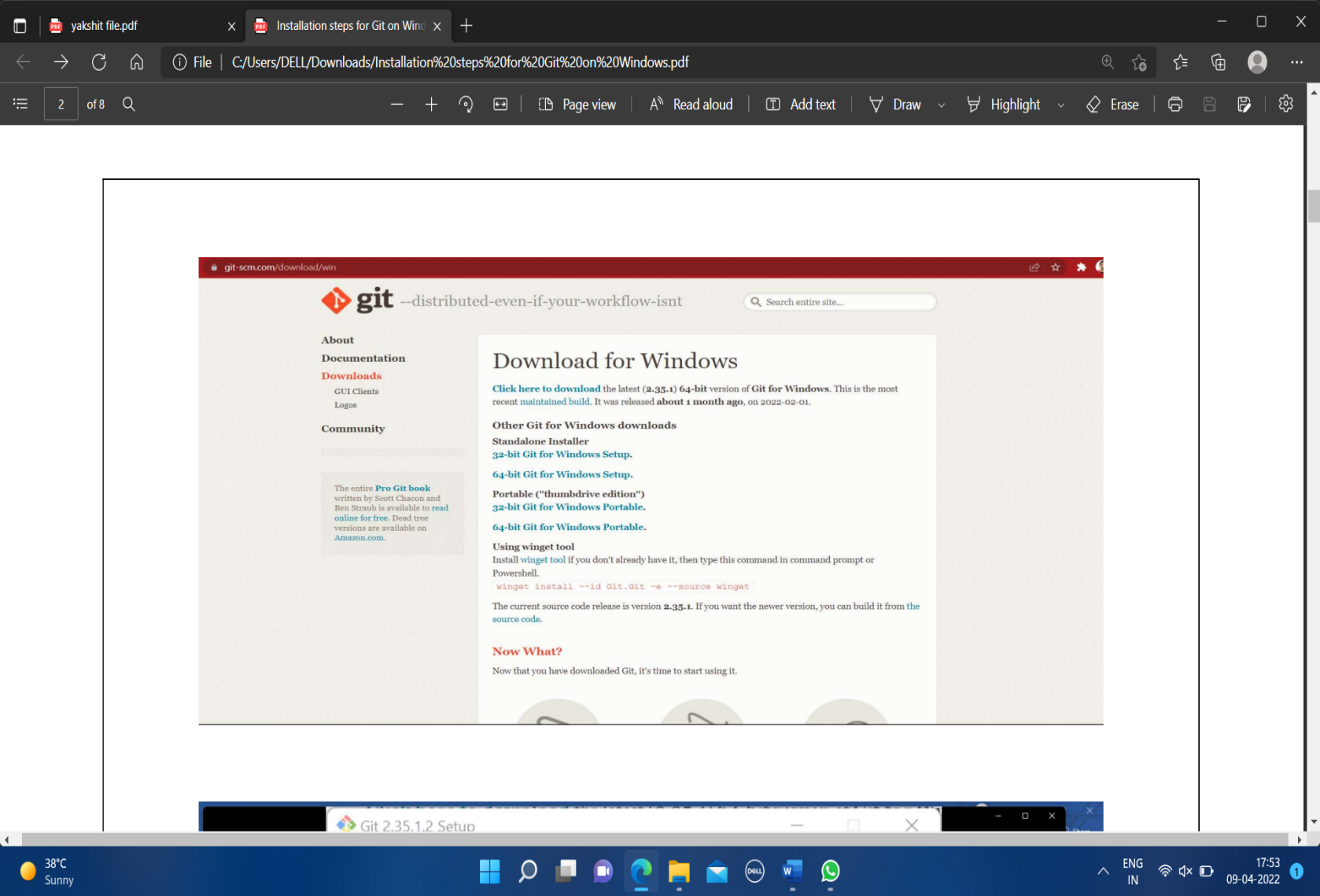
16. Configure any extra options based on your preferences.

17. Configure any experimental options if desired.

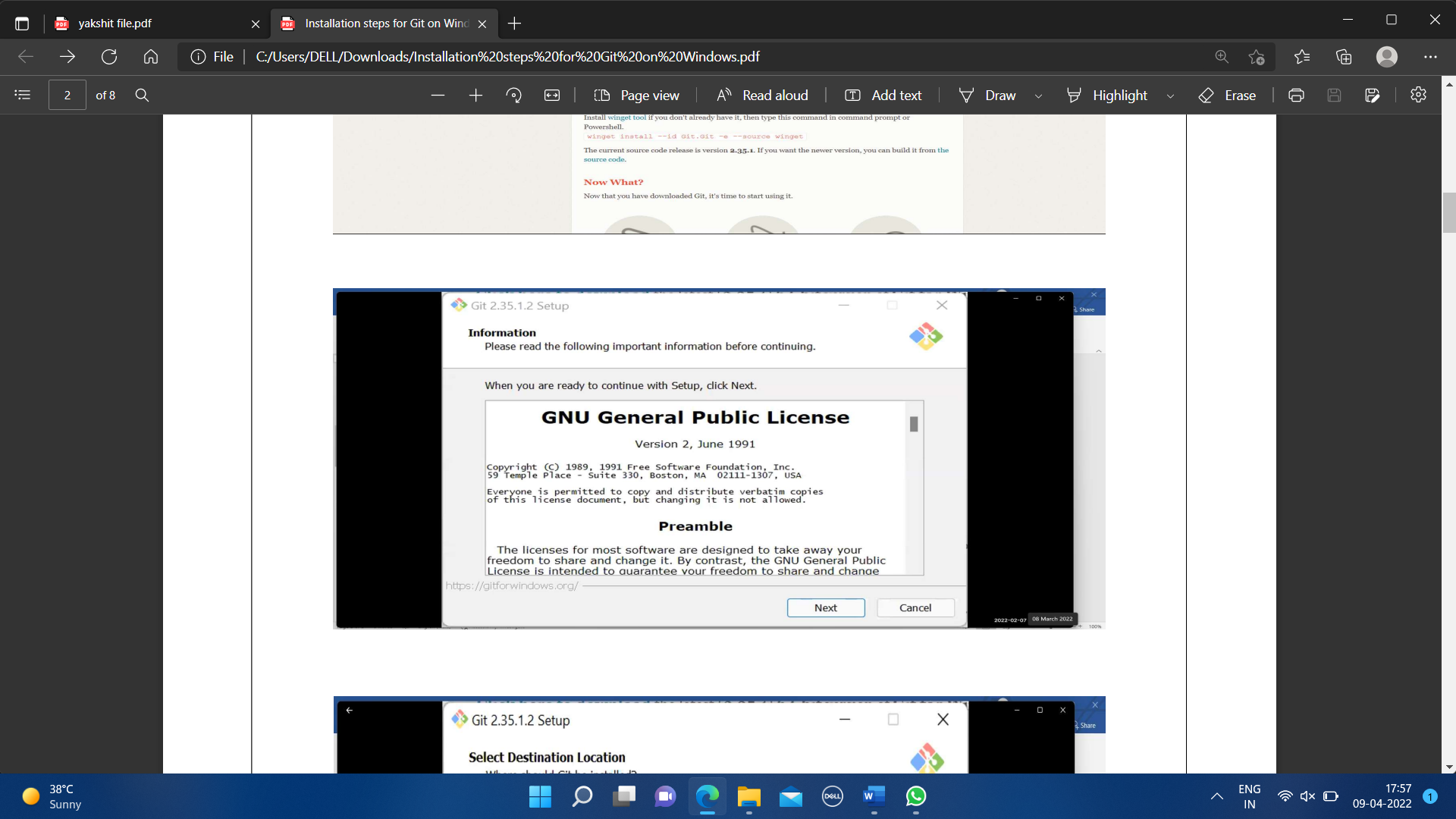
18. Proceed with the installation process.

19. Complete the Git setup wizard.

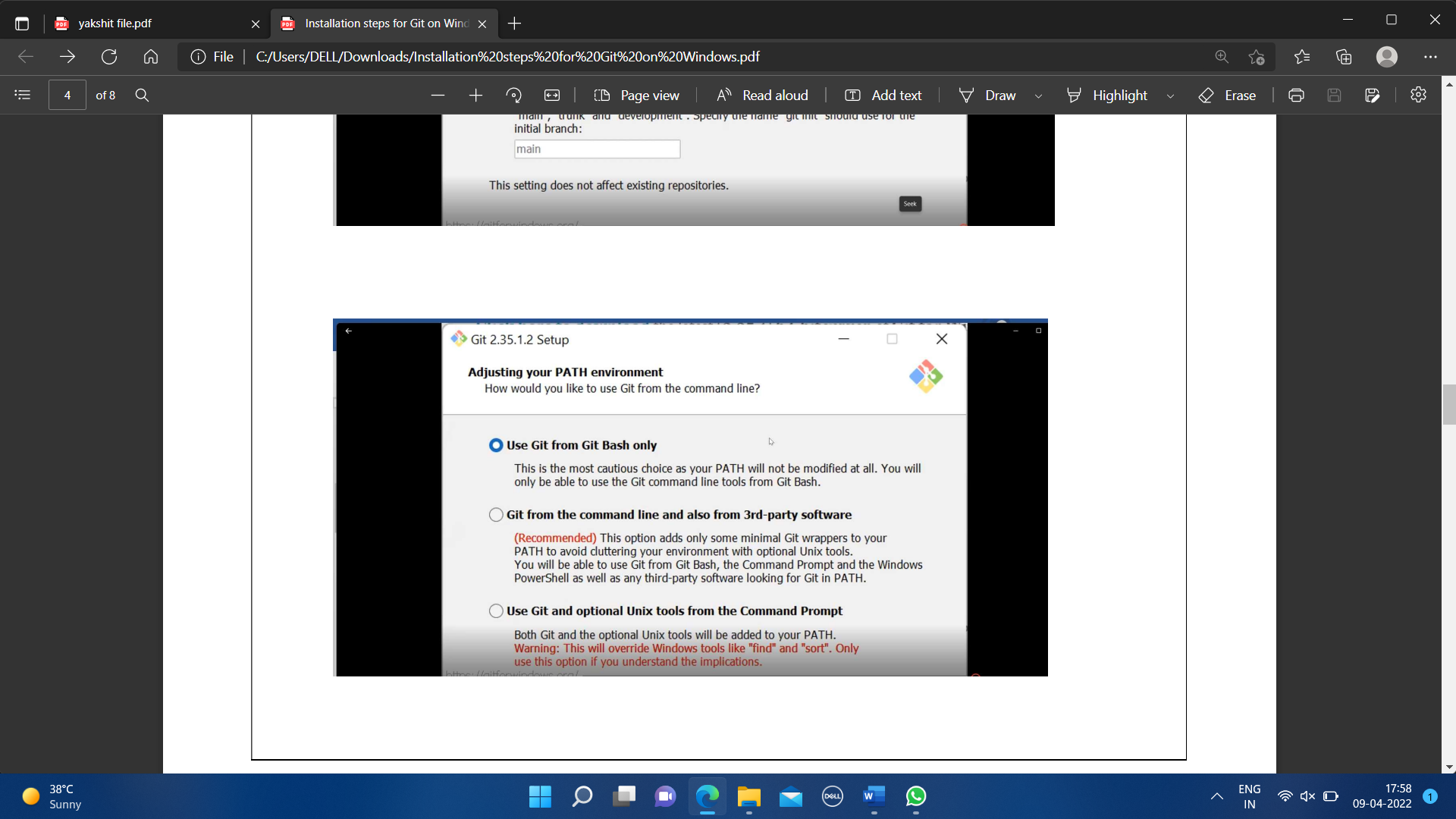
* Visit directly on git book page by <https://git-scm.com/book/en/v2> 
* Then click on Installation Git and click on whatever system you want, available are three- Windows, Apple and Linux.



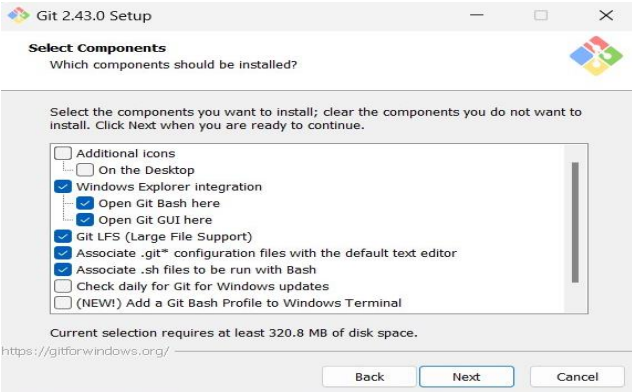
* Start the installation process by selecting the next button after reading all the terms and conditions.



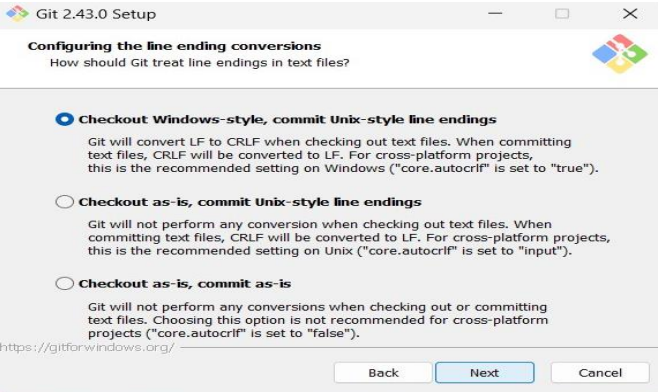
* After some more simple and easy settings and choosing your favourable environment and doing some SSH settings, it finally starts exporting the files in system and completes the Git hub wizard.



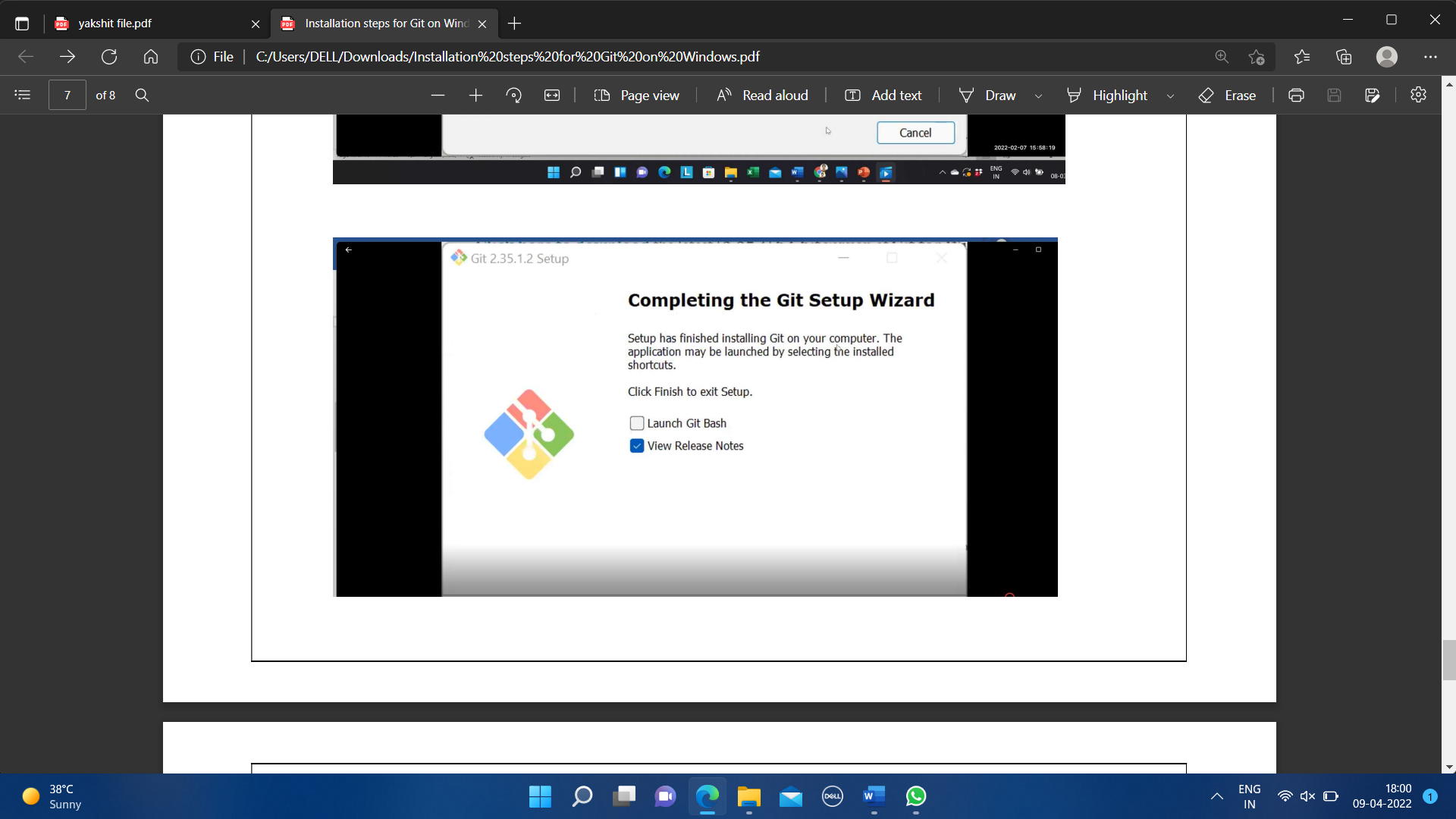
* Choose which components of Git you want to install. This may include features like Git GUI and Git Bash.



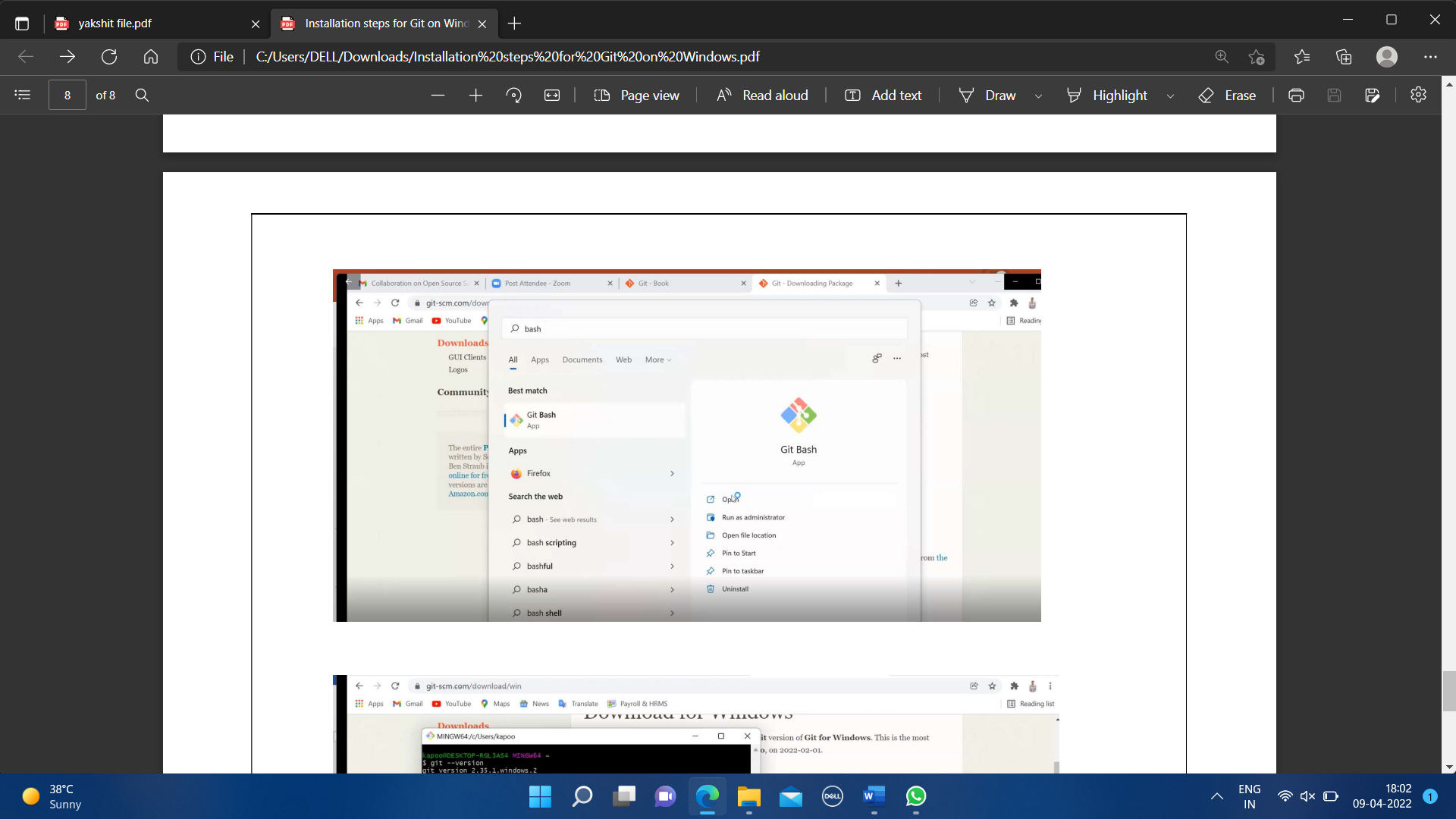
* Decide how line endings should be handled in text files.



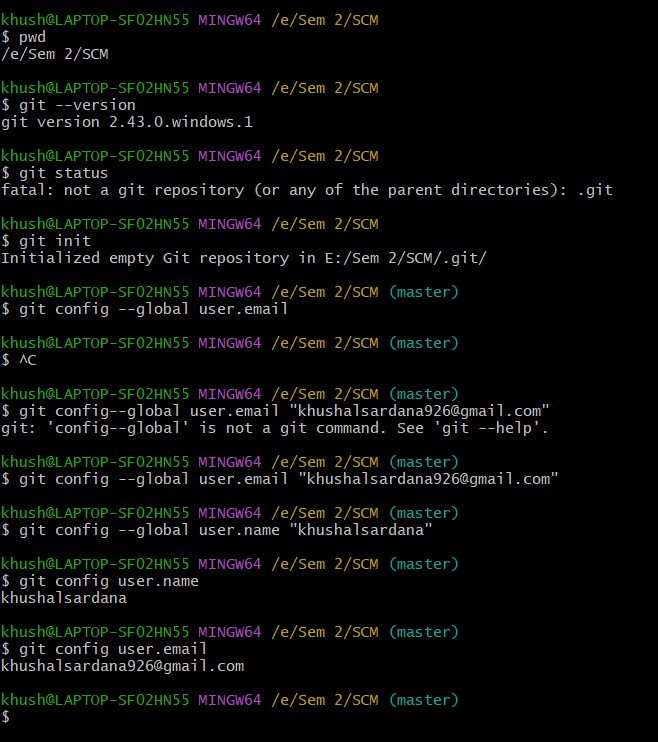
* Complete the setup wizard for the Git .



* 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 using the following command.



*Figure 1.2 Git init Command*

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.

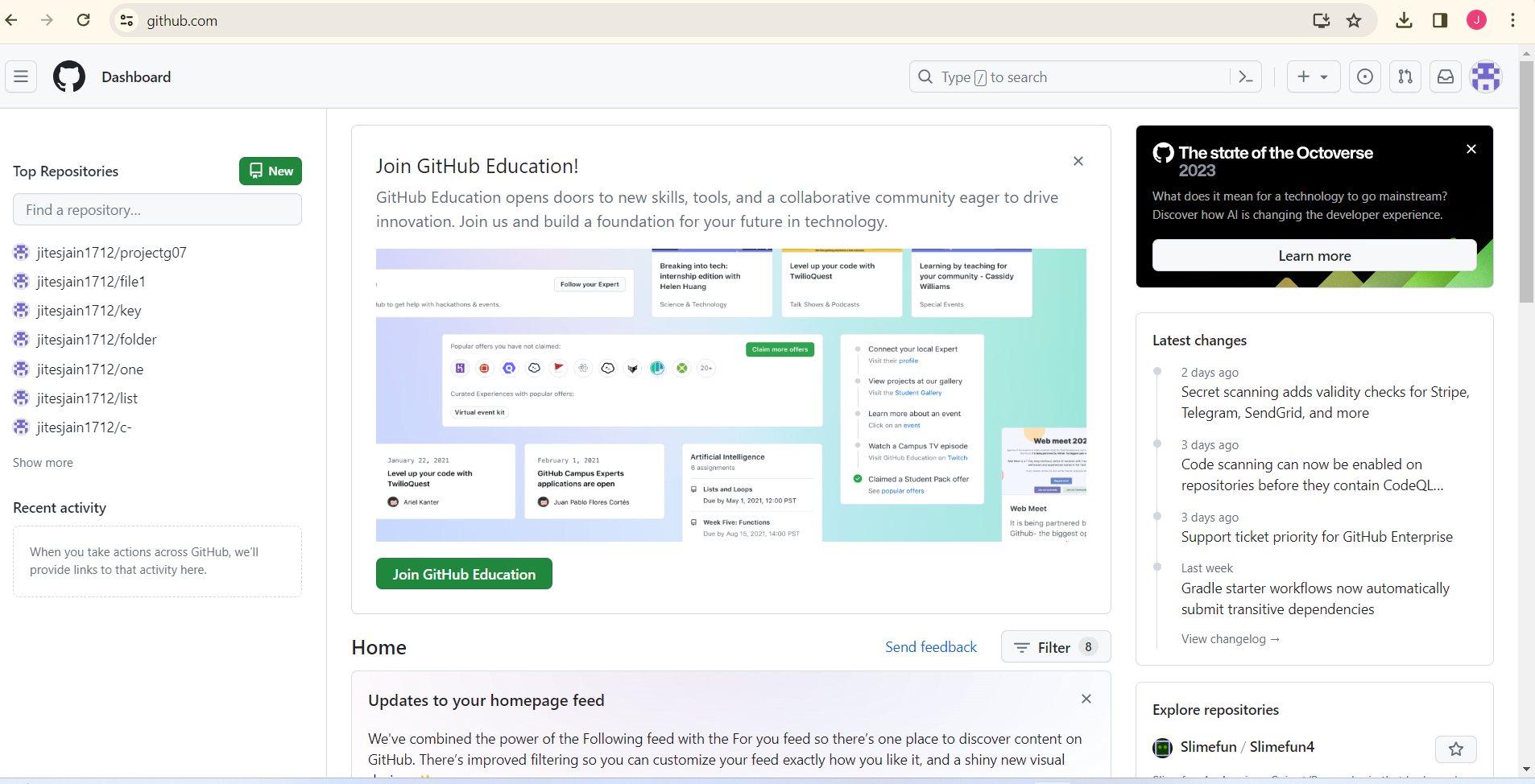
**Procedure:**

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



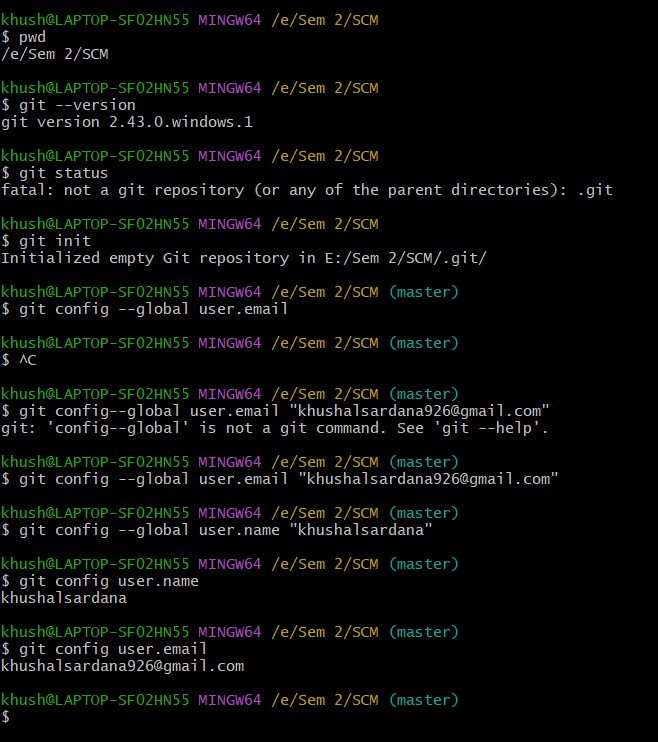
*Figure 2.1 GitHub Landing Page*

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.



*Figure 2.2 Dashboard Of GitHub*

You can login your GitHub Account in Git Bash using the following Commands.



*Figure 2.3 Command Interface for Git Bash*

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.

**Procedure:**

To generate a git log, we first need to make a git repository, the steps to make so are as follows,

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"

git config --global user.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.

Experiment No. 03

7. Check Status Again:

- Execute the command `git status`.

- Verify the changes and the status of the repository after staging the file.

8. Commit the Changes:

- Execute the command `git commit -m "file name"`.

- Commit the staged changes with a descriptive message.

9. 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`.

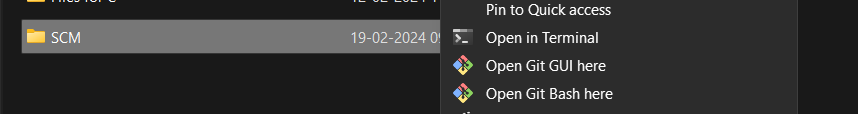
- View the commit history with additional statistics, including the number of

insertions and deletions per file.

12. View Details of a Specific Commit:

- Execute the command `git show commit\_hash`.

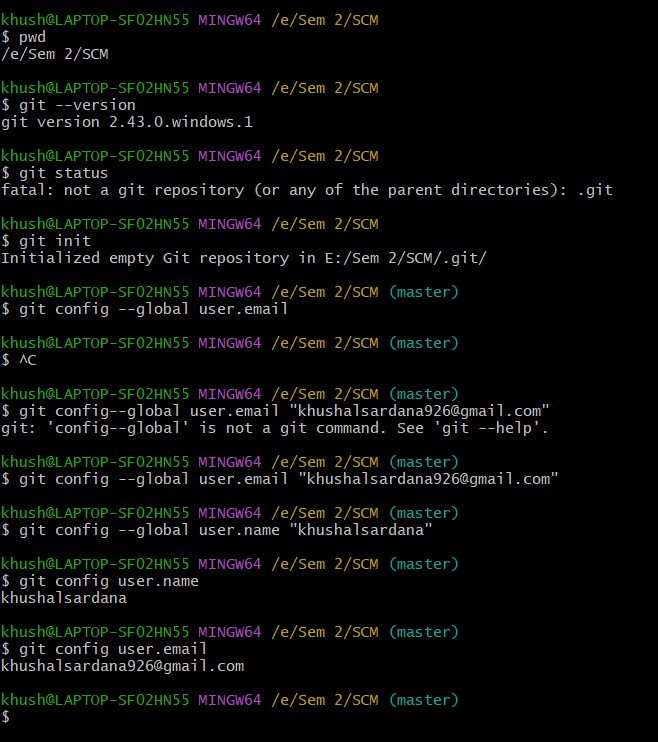
• Create a new folder in your local drive, and right click it, click on “Open Git Bash here”. This will open the git bash terminal.



### Fig 3.1 Opening the source in Git Bash

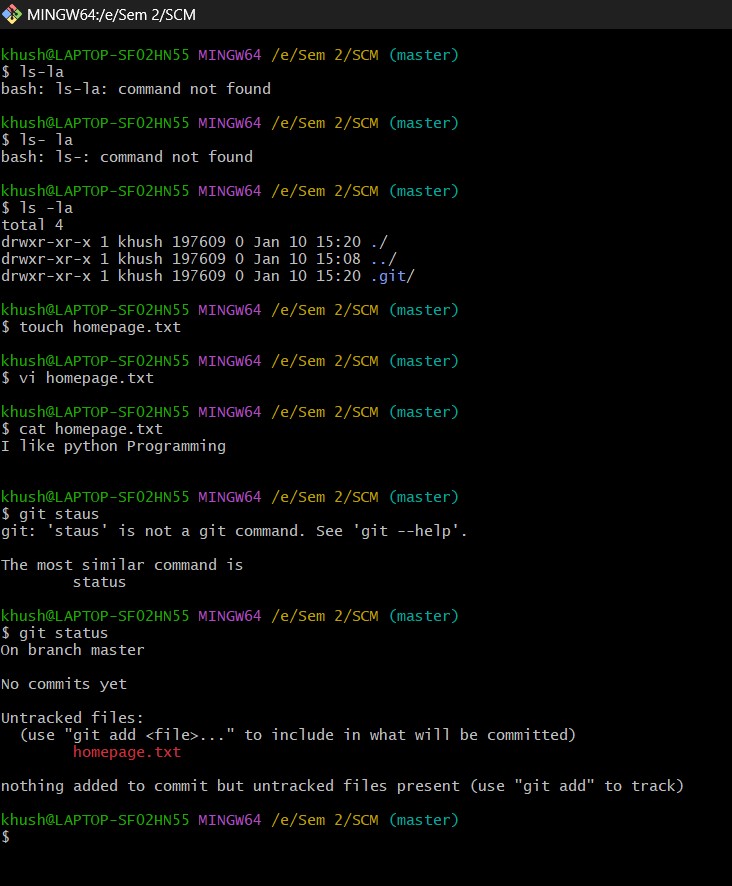
• To create a new local repository, use command, “git init”, this will initialize git i.e.

create a hidden folder, “.git”.



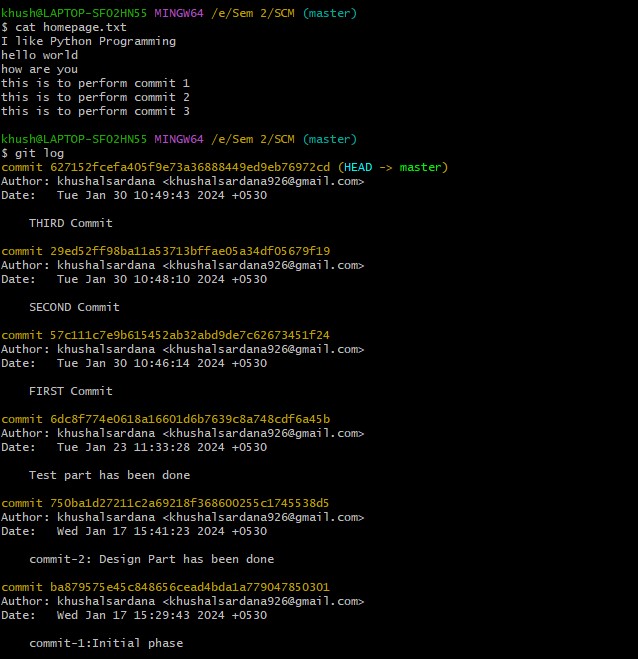
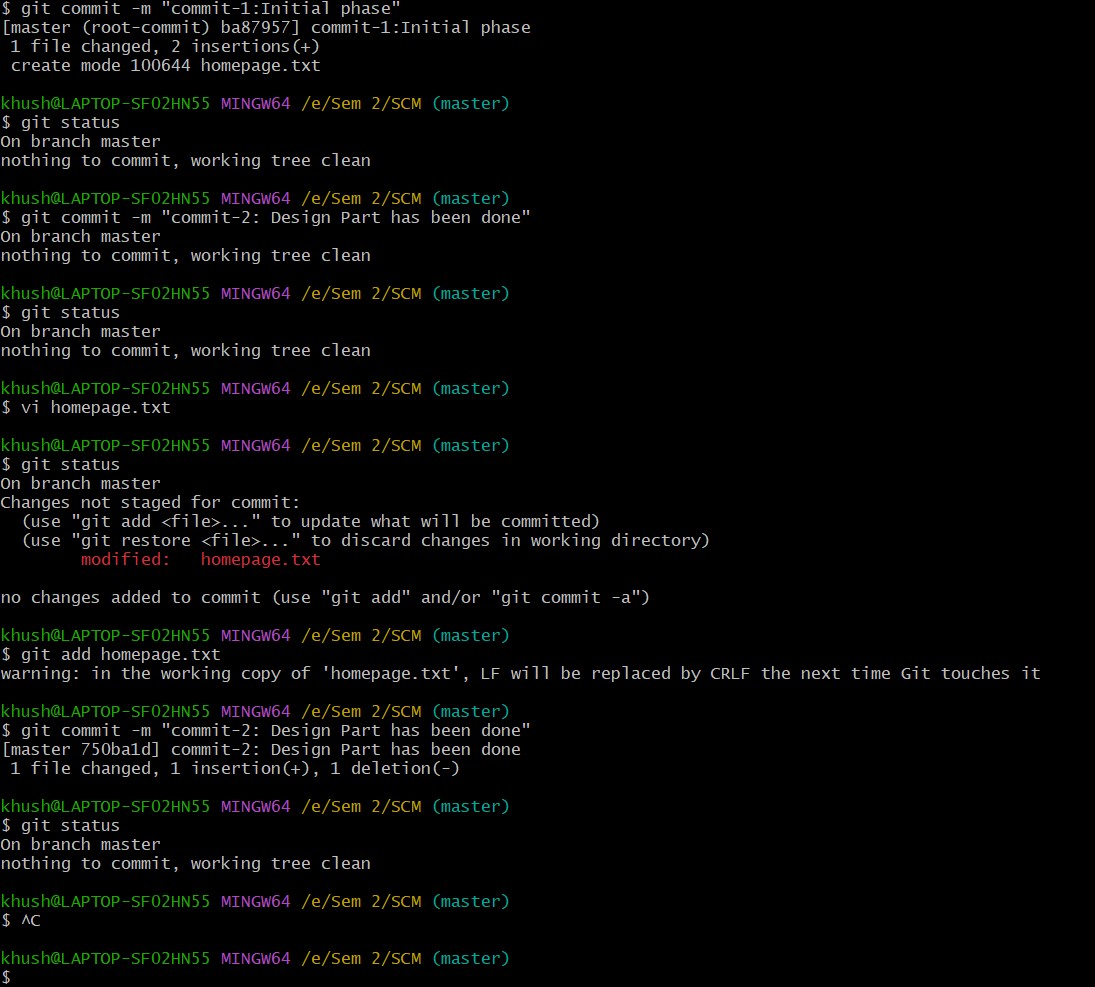
*Fig 3.2 Git init command interface*

* Since, your username and email have already been configured with git, globally i.e. for all the files on the working system, we will not do it again.
* Next, we’ll create a file with the command, “vi <filename>”, the file can be of any type i.e., .c, .txt, .py, etc.
* Then we’ll use the command, “ls”, it will list all the contents of that working directory, i.e. all files and folders present in it.
* After so, we’ll use the command “git status”, to display the state of the working directory and files.



### Fig 3.3 Git Command(s) interface - I

* Now we’ll add our file to staging area for the next commit by using the command, “git add <filename>.
* Next, we’ll commit the changes made to our file by using the command, “git commit -m “message for commit”.
* Now, we’ll finally use the “git log” command to display the commit history of the repository.



### Fig 3.4 Git Command(s) interface - II

Similarly, you can create more files and commit them, after adding them to staging area and finally use the “git log” command to display the newer commit history along with previous one.

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”.



*Fig. 4.1 Git Command For Creating Branch*

1. To check how many branches we have:



*Fig. 4.2 Git Command For Checking Branch*

As you can see here one branch is showing that I created 1stBranch.

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



*Fig. 4.2 Git Command For Switching Branch*

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

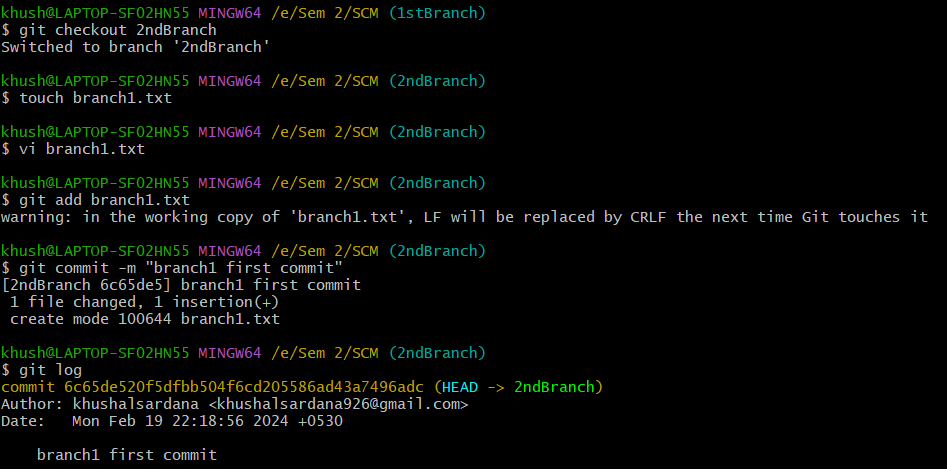
**Visualizing branches:**

For visualizing, we have to create a new file in the branch that we made “2ndBranch” 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 Branch1 that I previously made and after that I check git status. Now I add text in branch1 file (branch1.txt) and use git add “filename”.

Then I use git commit -m “key\_name” command for the changes I made and insertions I do.

At last I check my activities with the help of git log command.



*Fig 4.4 Git Interface*

EXPERIMENT NO. 05

**Aim: Git lifecycle description**

**Algorithm:**

1. Initialize the Git repository:

**git init**

2. Create a new file:

**touch filename**

3. Check the status of the repository:

**git status**

4. Stage all files with the .txt extension:

**git add \*.txt**

5. Check the status again to see staged changes:

**git status**

6. Add further new files.

7. Check the status to see untracked or modified files:

**git status**

8. Stage all changes (including new files):

**git add -A**

9. Check the status to verify staged changes:

**git status**

10. Remove a file from the staging area (but keep it in the working directory):

**git rm --cached <file name>**

11. Commit the staged changes with a comment:

**git commit -m "comment"**

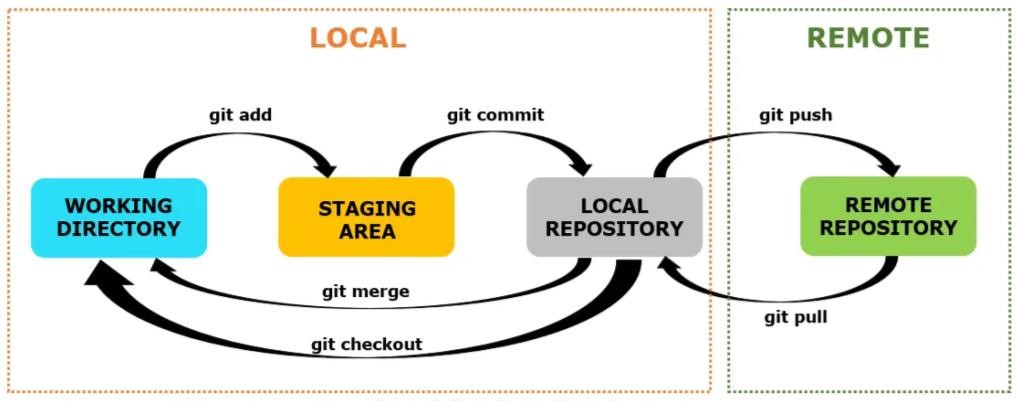
***Theory:***

**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 istracked by Git or not, these phases are still prevalent. However, when a projectis under Git version control system, they are present in three major Git statesin addition to these basic ones. Here are the three Git states:

• Working directory

• Staging area

• Git directory



## *Fig 5.1 Git Lifecycle*

**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.

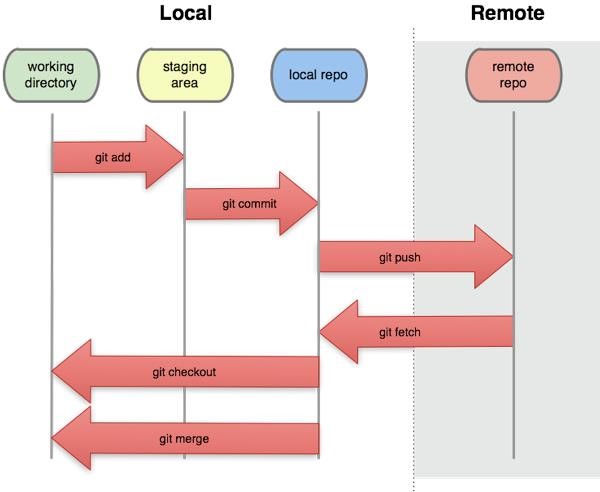
**Staging Area:**

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 the team members.



*Fig 5.2 Major Git Sections*