

**SCM LAB RECORD**

**A logo with a black background

AI-generated content may be incorrect. **

**Name: Dayashree S**

**Branch: B.Tech**

**Section: CSE**

**Course: Source Code Management**

**SEN: A86605223011**

**Faculty: Dr. Monit Kapoor**

**Date of Submission: 20 may 2025**



Amity School of   
Engineering & Technology

**💡 GIT FUNDAMENTALS**

**1. Introduction to Git**

Git is a **distributed version control system (DVCS)** developed by Linus Torvalds in 2005. It is used to manage and track changes in source code during software development, allowing multiple developers to collaborate efficiently.

**2. Features of Git**

* **Distributed**: Every developer has a full copy of the repository.
* **Speed**: Git operations are very fast due to local repositories.
* **Data Integrity**: Every file and commit is checksummed using SHA-1.
* **Support for Branching and Merging**: Git handles branching and merging efficiently.
* **Lightweight**: Branches and tags are lightweight and quick to create.

**3. Basic Git Terminology**

* **Repository (Repo)**: A directory that contains your project work tracked by Git.
* **Commit**: A snapshot of changes made to files in a repository.
* **Branch**: A pointer to a specific commit; used for parallel development.
* **Merge**: The process of combining changes from different branches.
* **Staging Area**: A place where changes are grouped before committing.
* **Working Directory**: The local directory where files are modified by the user.

**4. Branching in Git**

Branching allows developers to work on different features, fixes, or experiments independently from the main codebase. A branch represents a separate line of development.

Common branch types:

* **Main or Master**: Stable and deployable version of the code.
* **Development (dev)**: Integration branch for all features.
* **Feature Branches**: Created for developing individual features.
* **Hotfix Branches**: Used to fix urgent issues in production code.

After work on a branch is complete, it can be merged back into the main or development branch.

**5. Merge Conflicts**

A merge conflict occurs when changes in two branches affect the same part of a file and Git cannot automatically reconcile them. These conflicts must be resolved manually by the developer.

Once resolved, the merge can be completed, ensuring that the code integrates correctly without overwriting important changes.

**6. .gitignore File**

The .gitignore file is used to specify files and directories that Git should ignore and not track. This is useful for excluding temporary files, compiled code, environment configuration files, and other data that shouldn't be versioned.

Examples of ignored files:

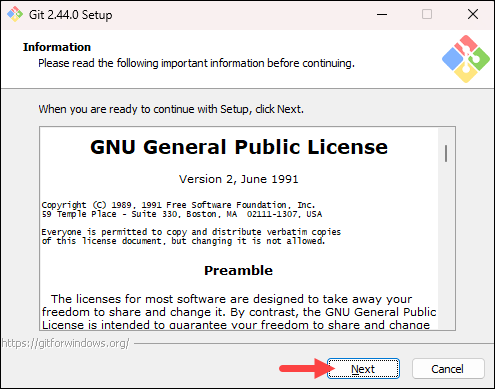
* Log files
* System files like .DS\_Store
* Dependency folders like node\_modules
* Build folders like dist or out

**7. Advantages of Git**

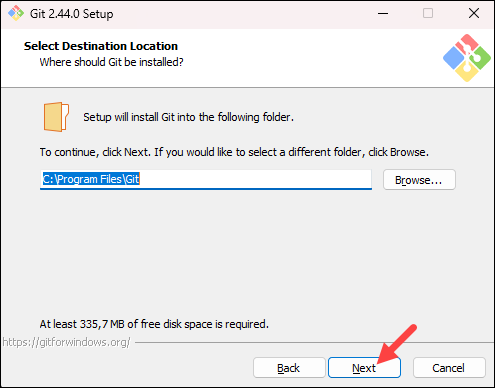
* Facilitates collaboration in teams.
* Maintains a detailed history of code changes.
* Allows easy switching between different versions of code.
* Enhances productivity through branching and merging.
* Supports open-source development and contribution through platforms like GitHub, GitLab, etc.

**GIT INSTALLATION**

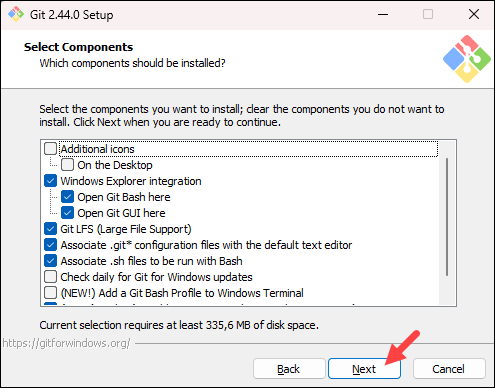
1. Navigate to the [**official Git downloads page**](https://git-scm.com/download/win) and click the download link for the latest Git version for Windows:
2. Double-click the downloaded [**file**](https://phoenixnap.com/glossary/what-is-a-file) to extract and launch the installer
3. Review the [**GNU General Public License**](https://phoenixnap.com/glossary/gnu-general-public-license), and when you are ready to install, click **Next**.



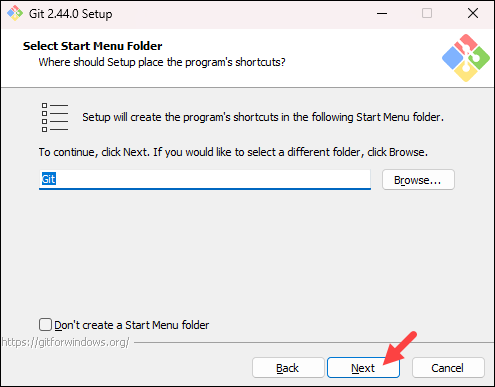
1. The installer prompts you for an installation location. Leave the default one unless you want to change it, and click **Next**.



1. In the component selection screen, leave the defaults unless you need to change them and click **Next**.

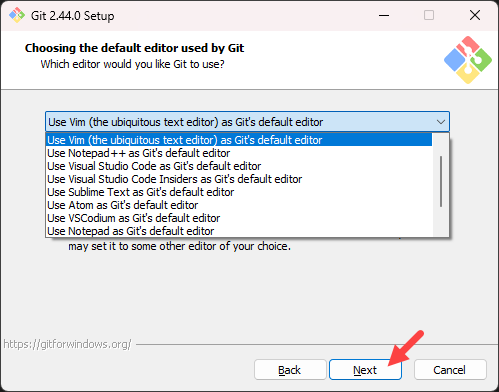


1. The installer offers to create a start menu [**folder**](https://phoenixnap.com/glossary/what-is-a-folder). Click **Next** to accept and proceed to the next step.

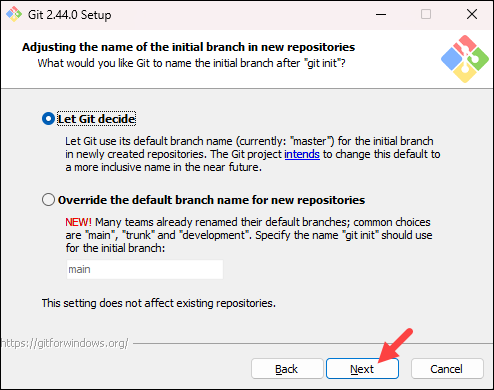


7. Select a text editor you want to use with Git. Use the drop-down menu to select Notepad++ (or whichever text editor you prefer) and click **Next**.

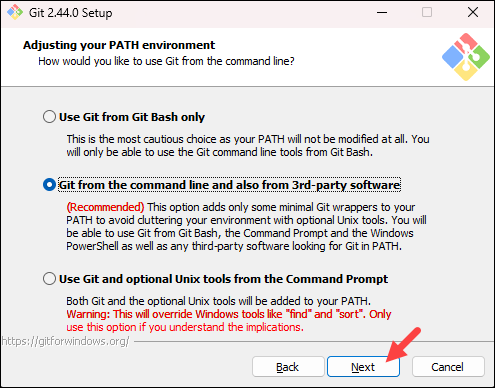
If you prefer to use a CLI text editor in [**Git Bash**](https://phoenixnap.com/kb/what-is-git-bash), select [**nano**](https://phoenixnap.com/kb/use-nano-text-editor-commands-linux) or [**Vim**](https://phoenixnap.com/kb/vim-commands-cheat-sheet) from the list.



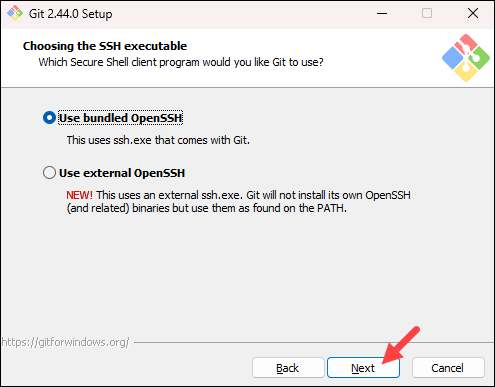
1. The next step allows you to choose a different name for your initial branch. The default is **master**. Unless you are working in a team that requires a different name, leave the default option and click **Next.**



9. The next step allows you to change the **PATH environment**. The **PATH**is the default set of [**directories**](https://phoenixnap.com/glossary/what-is-a-directory) included when you run a command from the command line. Keep the middle (recommended) selection and click **Next**.



10. The installer prompts you to select the SSH client for Git to use. Git already comes with its own SSH client, so if you don't need a specific one, leave the default option and click **Next.**



11. The next option relates to server certificates. The default option is recommended for most users. If you work in an Active Directory environment, you may need to switch to Windows Store certificates. Select your preferred option and click **Next**.

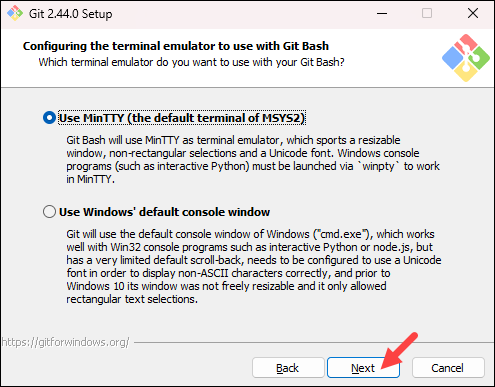
A screenshot of a computer

AI-generated content may be incorrect.

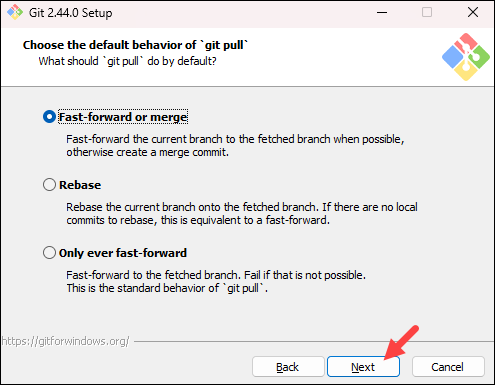
12. The following selection configures line-ending conversion, which relates to the way data is formatted. The default selection is recommended for Windows. Click **Next** to proceed.



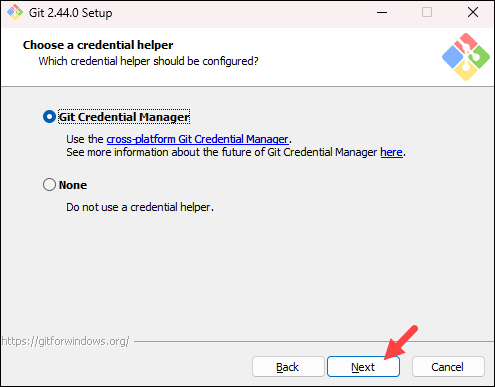
13. Choose the [**terminal emulator**](https://phoenixnap.com/glossary/terminal-emulation) you want to use. The default MinTTY is recommended for its features. Click **Next** to continue.



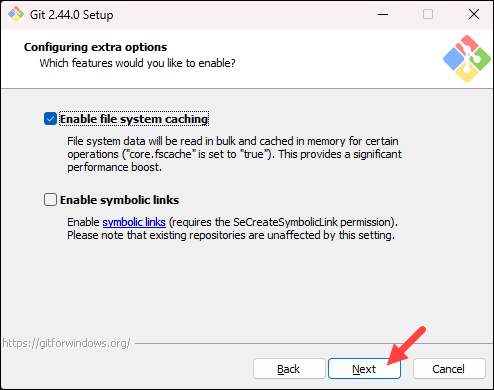
14. The next step allows you to choose what the **git pull** command will do. The default option is recommended unless you specifically need to change its behavior. Click **Next**to continue with the installation.



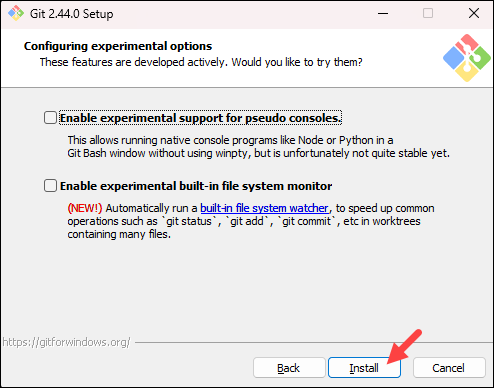
15. The next step is to choose which credential helper to use. Git uses credential helpers to fetch or save credentials. The default option is the most stable one. Select your preferred credential manager and click **Next**.



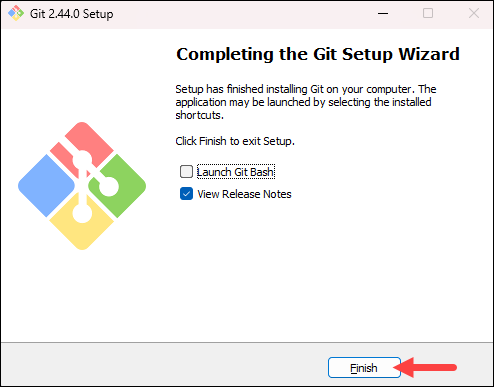
16. The next step lets you decide which extra options to enable. If you use [**symbolic links**](https://phoenixnap.com/kb/symbolic-link-linux), which represent shortcuts for the command line, tick the box. Keep [**file system**](https://phoenixnap.com/glossary/filesystem) caching checked and click **Next**.



17. Depending on which Git version you are installing, it may offer to install experimental features. At the time this article was written, the installer offered options to include support for pseudo controls and a built-in file system monitor. For the most stable operation, do not install experimental features and click **Install**.



18. Once the installation is complete, tick the boxes to view the Release Notes or launch Git Bash if you want to start using Git right away, and click **Finish**.

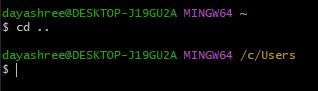


**BASIC COMMANDS IN GIT**

1. **pwd**
   * *Purpose*: Displays the Present Working Directory.
   * *Usage*: Use this to find the directory you're currently working in.



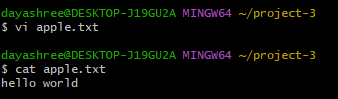
1. **cd**
   * *Purpose*: Change directory.
   * *Usage*: Navigate to a different folder.

v

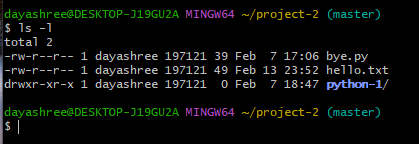
1. **ls**
   * *Purpose*: Lists all folders in the current directory.



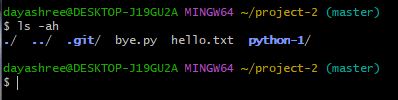
1. **Vi Command**
   * *Purpose*: Open the Vim/Vi editor to create or edit files.
   * *Usage*: vi filename. Press i to insert, esc to exit insert mode, and :wq to save and quit.
2. **cat**
   * *Purpose*: View file contents.
   * *Usage*: cat filename.



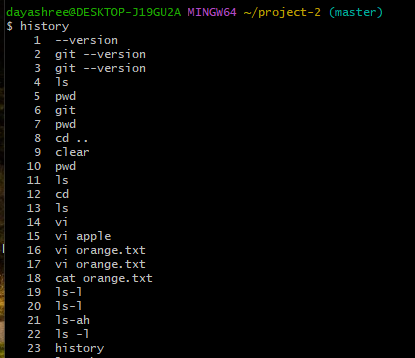
1. **ls -l**
   * *Purpose*: Lists files in long format with additional details (permissions, owners, size).



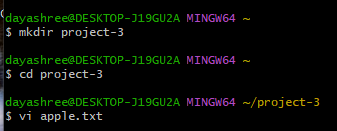
1. **ls -ah**
   * *Purpose*: Lists all files, including hidden ones.



1. **history**
   * *Purpose*: Displays the history of previously run commands.



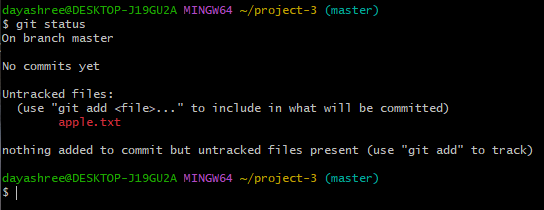
1. **mkdir**
   * *Purpose*: Create a new folder.
   * *Usage*: mkdir foldername.



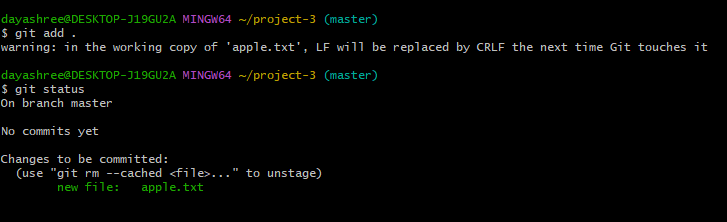
1. **git init**
   * *Purpose*: Initializes an empty Git repository in the current directory.



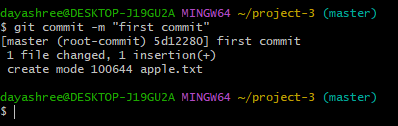
1. **git status**
   * *Purpose*: Displays the current status of the working directory and staging area, showing:
   * Modified files not yet staged.
   * Staged files ready for commit.
   * Untracked files.
   * Current branch and whether it's up-to-date with the remote branch.



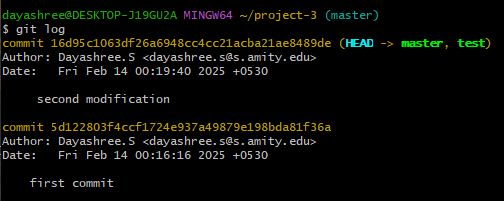
1. **git add**
   * *Purpose*: Stage a file to prepare it for commit.
   * *Usage*: git add filename.



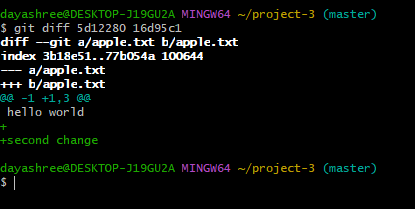
1. **git commit**
   * *Purpose*: Commit changes to the repository with a message.
   * *Usage*: git commit -m "Message".



1. **git config**
   * *Purpose*: Configure Git with username and email.
   * *Usage*:
     + git config --global user.mail "email@example.com"
     + git config --global user.name "Your Namve"
2. **git log**
   * *Purpose*: View commit history.



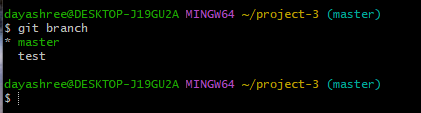
1. **git diff** 
   * *Purpose*: Shows the changes made in the working directory since the last commit.



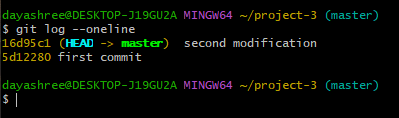
1. **git branch <branch name>**
   * *Purpose*: Creates a new branch.



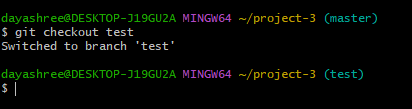
1. **git branch** 
   * *Purpose*: Lists all branches in the repository.



1. **git log –oneline**
   * *Purpose*: View a concise commit history.



1. **git checkout <branch name>**
   * *Purpose*: Switch to the master branch.



20.**git remote add origin " "**

* Purpose: Add a new remote repository named origin with the provided ,it would typically be git remote add origin <repository\_url>.

21.**git remote**

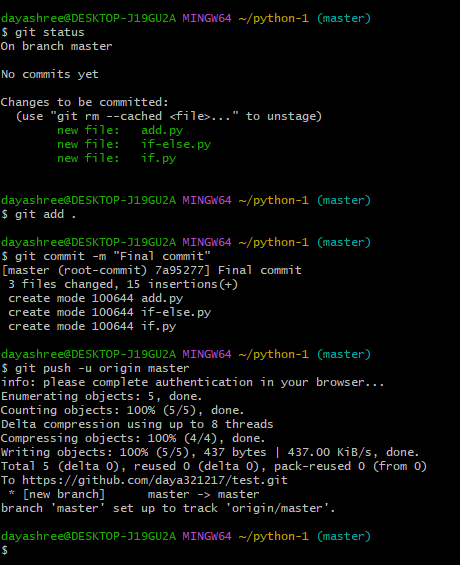
* Purpose: List the names of remote repositories you have configured.

A computer screen shot of code

AI-generated content may be incorrect.

22.**git push -u origin master**

* Purpose: Push the master branch from your local repository to the origin remote repository and set origin as the upstream for the master branch (so subsequent git pull and git push commands can be used without specifying the remote and branch).



**GIT MERGE**

* **Production Code** is the finalized, deployable version of software used in a live environment.
* **Branches** in Git allow multiple versions of code to exist simultaneously, starting from the project's inception.
* **Merging** happens when different branches need to be combined. Git does not automatically decide what to keep—it requires manual resolution.
* **Conflict Resolution**: If conflicts arise, Git will halt the merge. A merge tool helps resolve differences.

A computer screen shot of text

AI-generated content may be incorrect.

A black screen with yellow text

AI-generated content may be incorrect.

A computer screen with text on it

AI-generated content may be incorrect.

A computer code with text on it

AI-generated content may be incorrect.A black background with white text

AI-generated content may be incorrect.

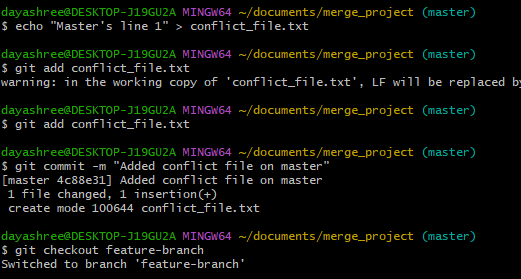
A screenshot of a computer program

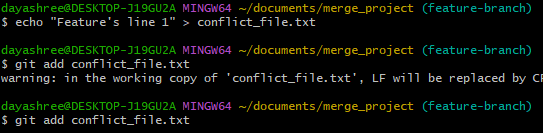
AI-generated content may be incorrect.

A black screen with white text

AI-generated content may be incorrect.

**Handle a simple merge conflict**





A screen shot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.



**Vim editor**

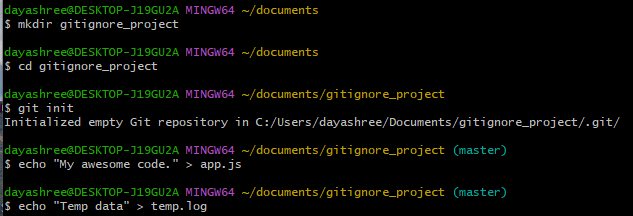
A black screen with white text

AI-generated content may be incorrect.

A screen shot of a computer code

AI-generated content may be incorrect.

**GIT IGNORE**



A screen shot of a computer

AI-generated content may be incorrect.



A screenshot of a computer error

AI-generated content may be incorrect.

These are the files to be ignored

A screenshot of a computer program

AI-generated content may be incorrect.

**GIT CLONE**

Navigate a project that you are interested in and fork it

A screenshot of a computer

AI-generated content may be incorrect.

Fork it

A screenshot of a computer

AI-generated content may be incorrect.

Copy the URL

A screenshot of a chat bot

AI-generated content may be incorrect.

Go to git bash and paste it with the following commands





A black background with yellow text

AI-generated content may be incorrect.

A computer screen shot of a computer code

AI-generated content may be incorrect.

A black screen with yellow text

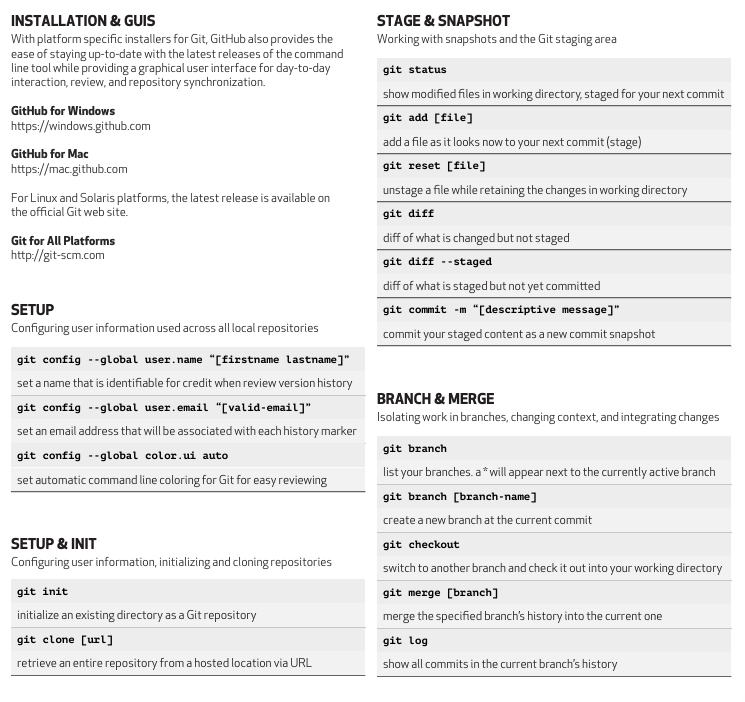
AI-generated content may be incorrect.

A black screen with yellow text

AI-generated content may be incorrect.

Now the file can be accessed

**GIT CHEAT SHEET**



A screenshot of a computer program

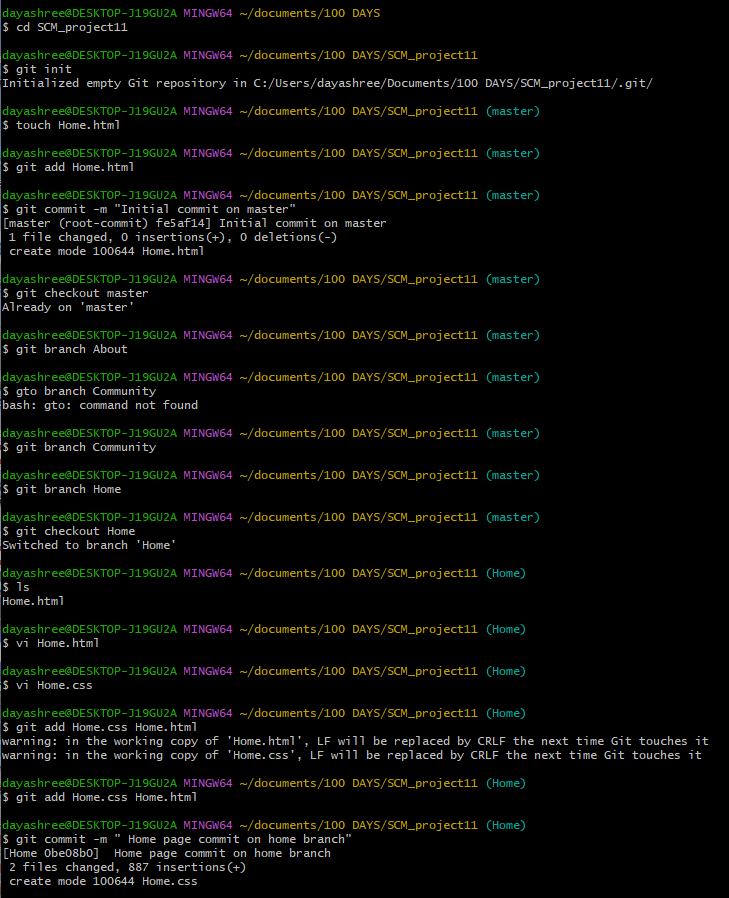
AI-generated content may be incorrect.

**SCM Project**

The project was to make a repository in GitHub, make 3 branches and merge it with the main branch and access all 4 team-mate’s repositories, fork it, clone it, make some changes and merge them.

First, make your own repositories and make 3 branches and add files and merge with the main branch.

Then create a dev branch where the team members will contribute.



A screenshot of a computer program

AI-generated content may be incorrect.

From the Above figure we can see that the three branches are created with their sub files and required commits are made.  
Now we have to merge it into main (here master) branch.

A screen shot of a computer

AI-generated content may be incorrect.  
merge the branches into main one by one and create a dev branch from master which acts like a copy of master as well as a branch for pull requests merge.

A computer screen shot of a program

AI-generated content may be incorrect.

Push it into Github from where the team members shall access the code

FORKING AND PULL REQUESTS

A screenshot of a computer

AI-generated content may be incorrect.

Go to your team member’s repository and fork it to your own GitHub account.  
**Important:** While forking, make sure to **uncheck** the option that says *“Copy the master branch only”* — we need access to all branches, especially the dev branch, for pushing our changes.

A screenshot of a computer

AI-generated content may be incorrect.

Now its forked as one of your repositories

A screenshot of a computer

AI-generated content may be incorrect.

A computer screen shot of a program

AI-generated content may be incorrect.

Clone it into your local machine

And make edits wherever required.

A computer screen shot of a program

AI-generated content may be incorrect.

We've cloned the "SCM\_Shashank-repo." For the edits, a new branch was created from the main branch, and all modifications were made there. These changes were then added, committed, and the specific branch was pushed to GitHub.

A screenshot of a computer

AI-generated content may be incorrect.

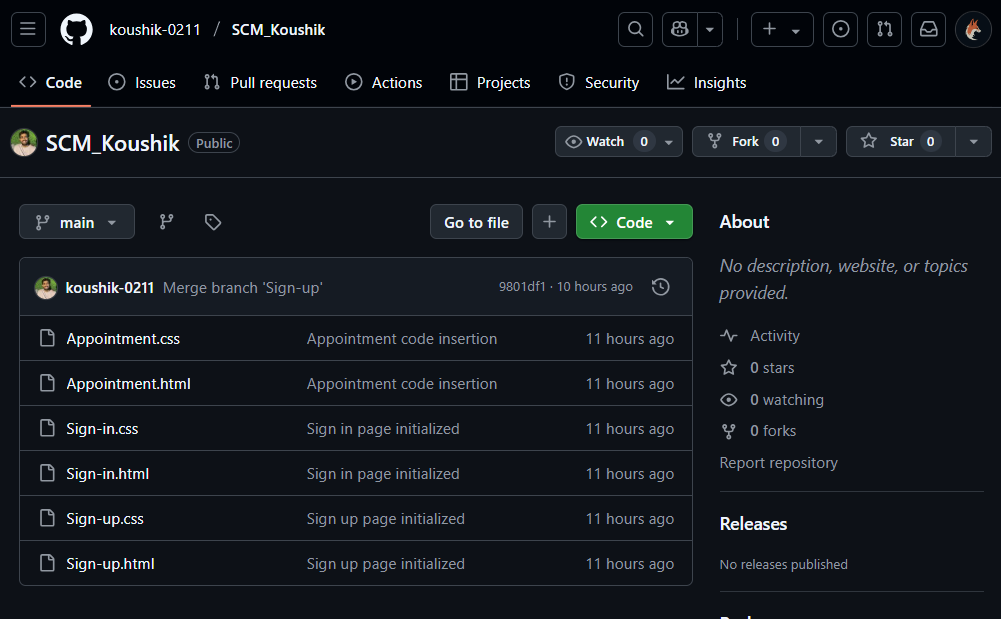
This is an automatic pop up from which we pull requests

A screenshot of a computer

AI-generated content may be incorrect.

Choose which branch to merge (here its dev) choose dev and proceed  
A pull request is generated

2nd Teammate



Navigate the repository in teammates account and fork it

A screenshot of a computer

AI-generated content may be incorrect.

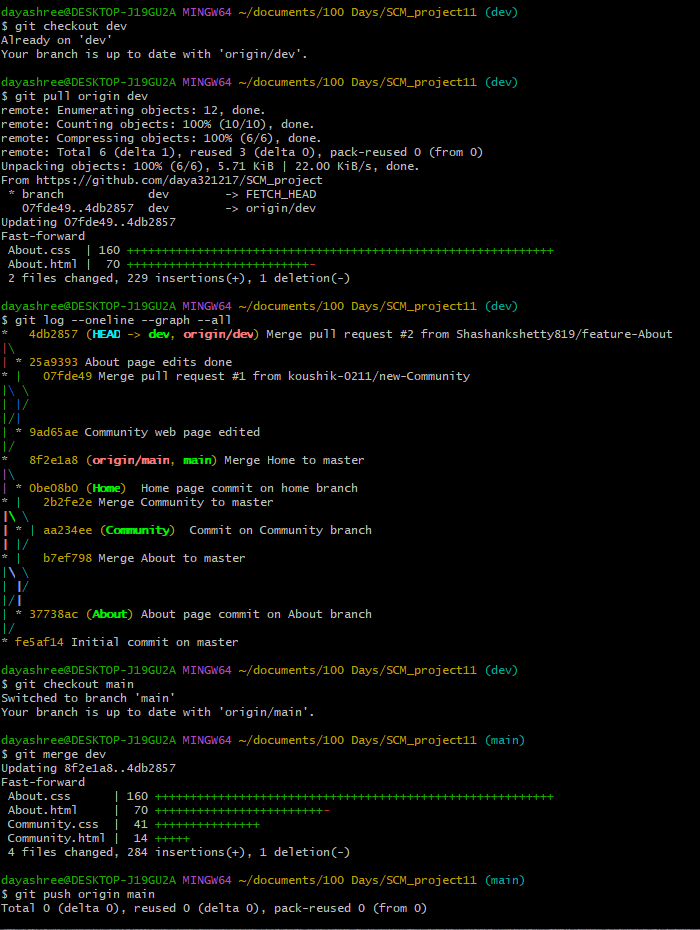


Once done check for pull requests on your own repository where others have contributed on your code

Accept the pull request send the teammates, got to your git bash local repo and checkout to dev branch  
git pull origin dev – will pull all the edited/ contributions from other teammates to your local machine

Once done it shows the amt of files edited   
later checkout to main branch and merge dev into main

And finally push main.

  
Once this is done the project is good to go once the main branch is pushed

This is the Venn diagram of the initial branching system  
A diagram of a community

AI-generated content may be incorrect.