Subject Name: ***Source Code Management***

Subject Code: ***CS181 Alpha***

Cluster: ***Alpha***

Department: ***DCSE***

Submitted by

Name: ***SRIJAN SAMRIDH***

Roll No: ***2110991384***

Group: ***G-18***

Submitted to

Teacher: Dr. Sihka

Task 1.1

Type of Assessment	Time of Conduction	Total Marks	Description of Tasks for Evaluation
Task 1.1	Week 4	60	1. Setting up of Git Client, 2. Setting up GitHub Account, 3. Generate logs 4. Create and visualize branches 5. Git lifecycle description

INDEX OF PROGRAMS TO DONE IN TASK 1.1

Git and Github

Git and GitHub are generally used hand to hand. The former one is used for version control, and the latter is used to host the code online where each team member can access it.

I want you to assume a scenario where many people are working on some project and suppose they don't use Git, so they have to share their code with each other by some or other means. They can use several ways like transferring it over mail or compressing it and then sending it in the USB drive. So overall, they have to do a lot more work than required. This happens because they are not aware of Git and GitHub.

What is Git?

In layman's terms, Git is a distributed version control system created by Linus Torvalds in 2005.

Version control or source control is a practice of managing and tracking the changes to the program code. So, it's like a tracker that tracks every change in your code. You can commit the changes and can revert to the old changes as well.

What is Github?

- GitHub is a code hosting service that is based on the Git version control system. So, GitHub is a place where developers store their projects and can make network with like-minded people. It's one of the largest communities for devs.
- It acts as a safe house for your application code.

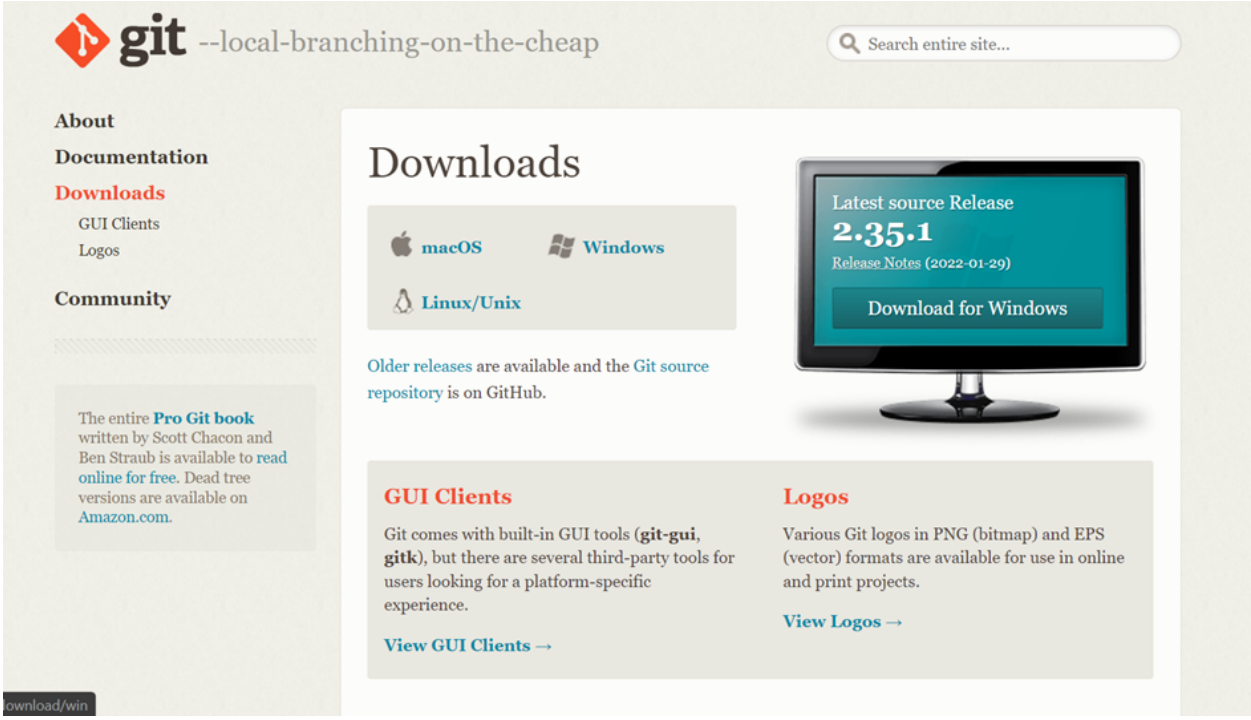
There's a difference between Git and GitHub, but generally we use them together. In a nutshell, we can say, Git is a software that tracks the code, and GitHub is a website where you can put your code.

1. SETTING UP OF GIT CLIENT

1. Downloading and Installing Git

Step-1:

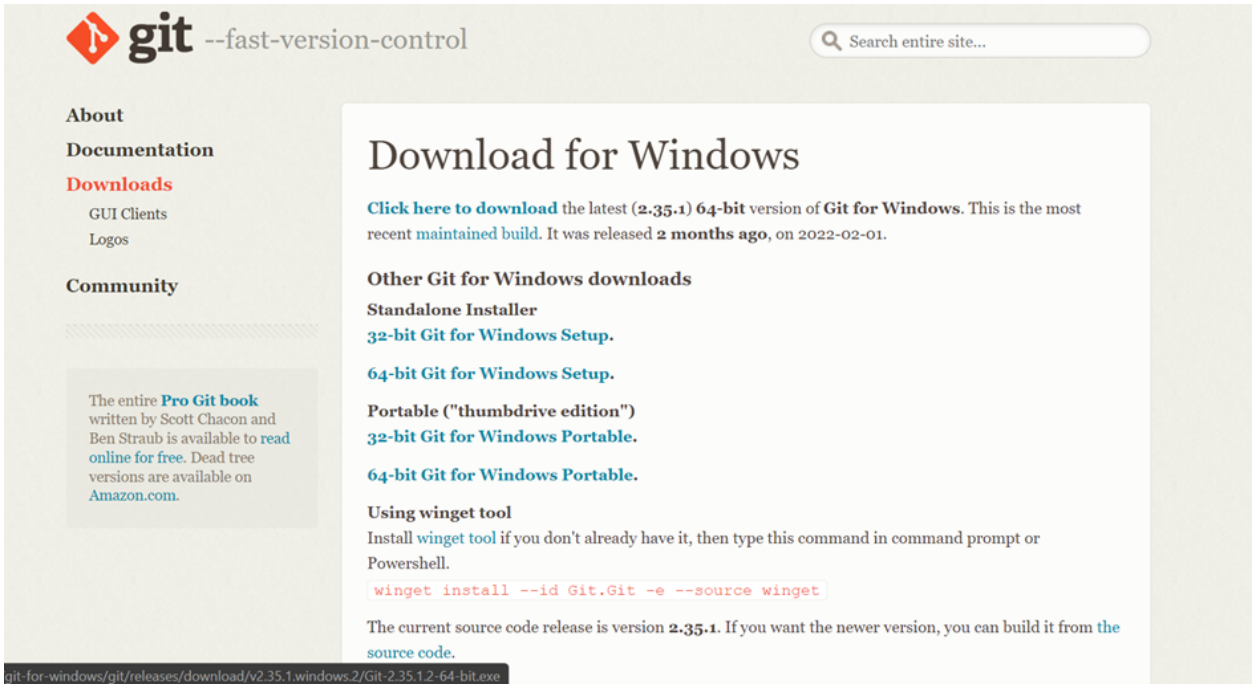
To download git, go to Git - Downloads (git-scm.com) and download git as per your operating system.



It will show like this.

Step-2:

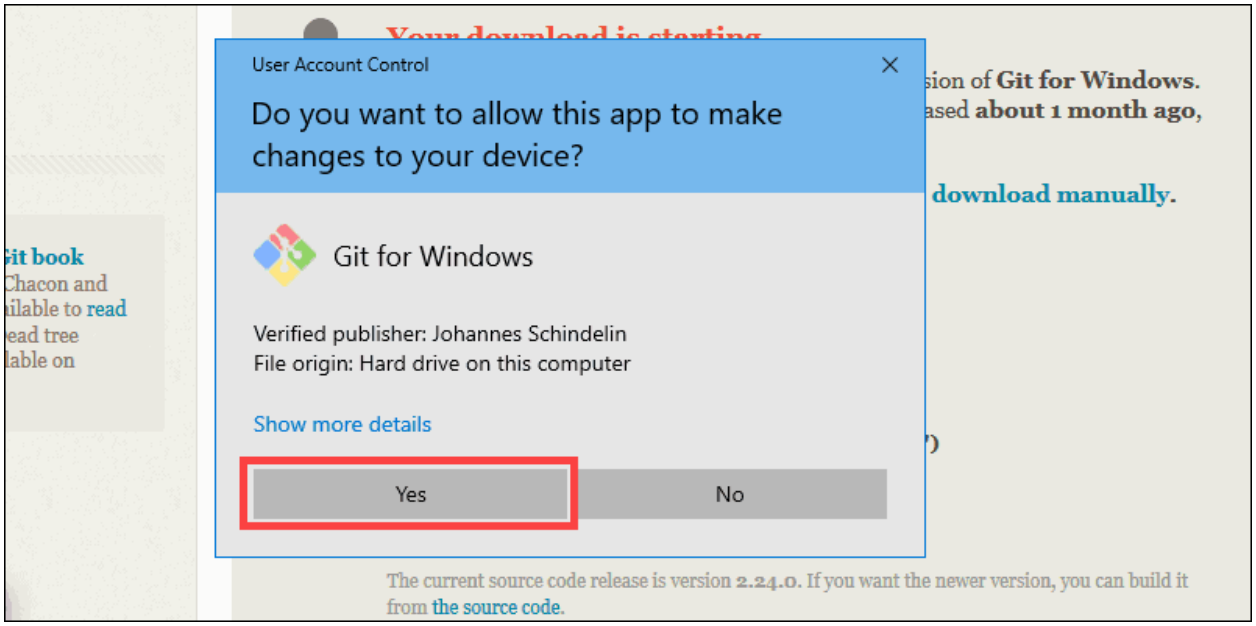
Choose as per your operating system .I chose windows. Once you click on the download button, your download will start. (Its size is around 48-50 Megabyte).



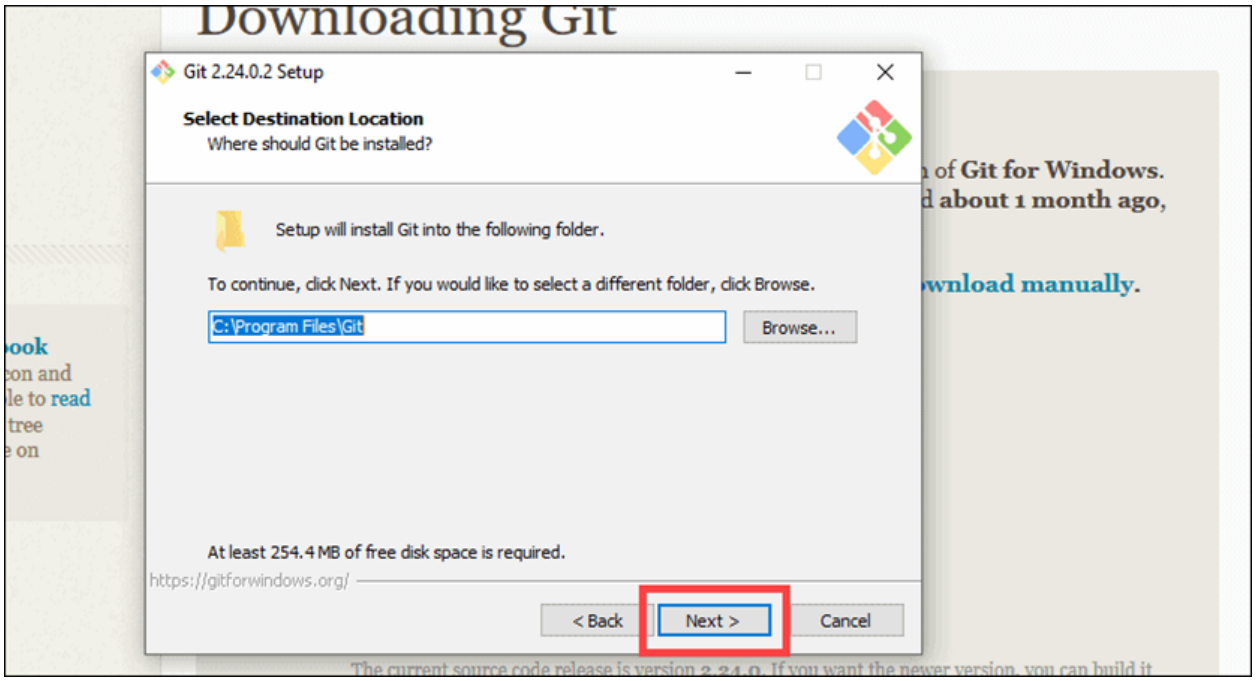
Step-3:

After downloading, double click on it to install.

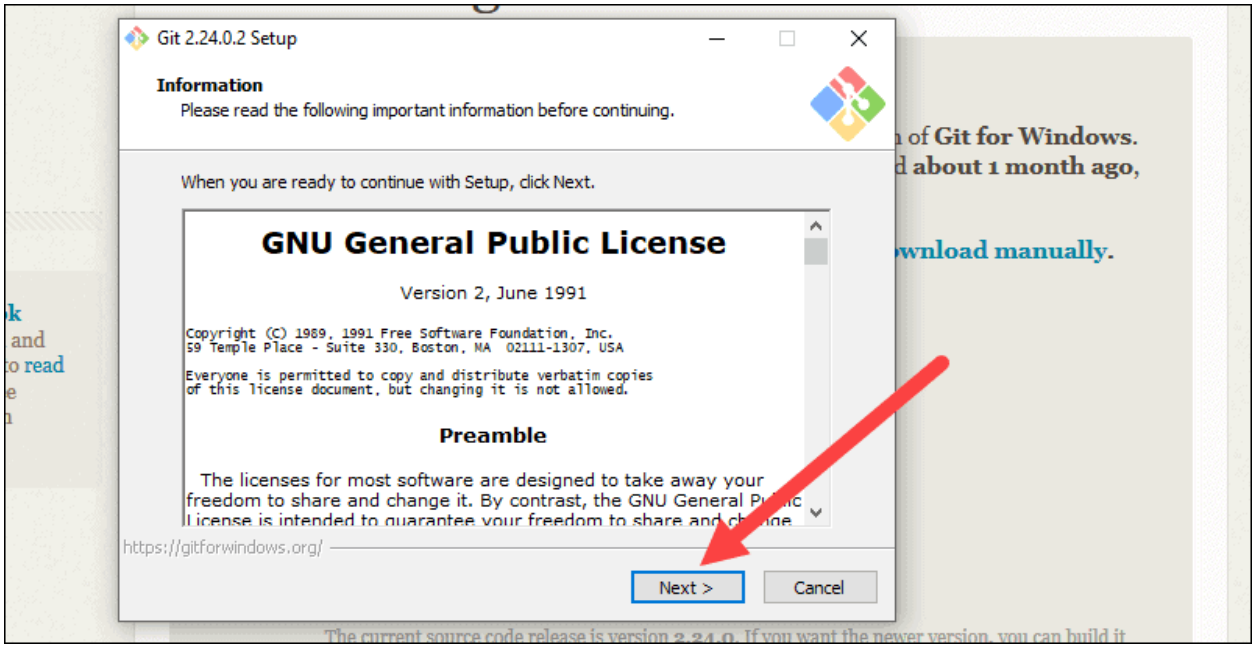
1.You will see a screen as shown below. Simply, click on **Yes**.



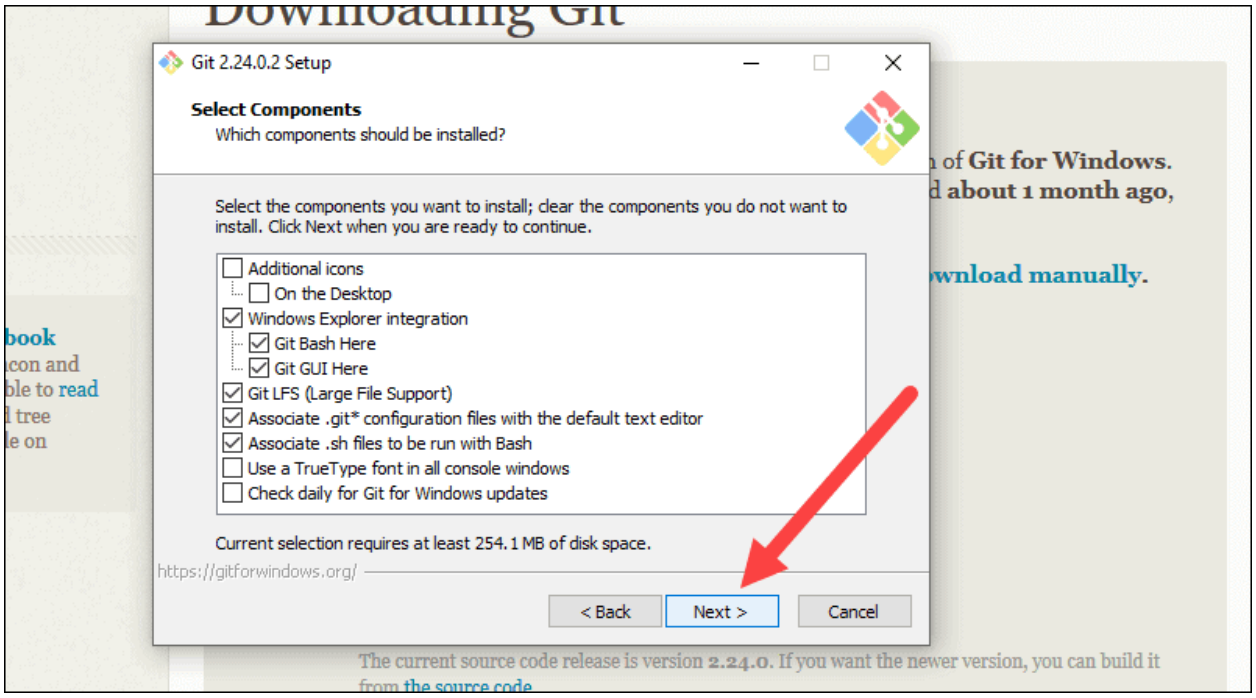
2. The installer will ask you for an installation location. Leave the default, unless you have reason to change it, and click **Next**.



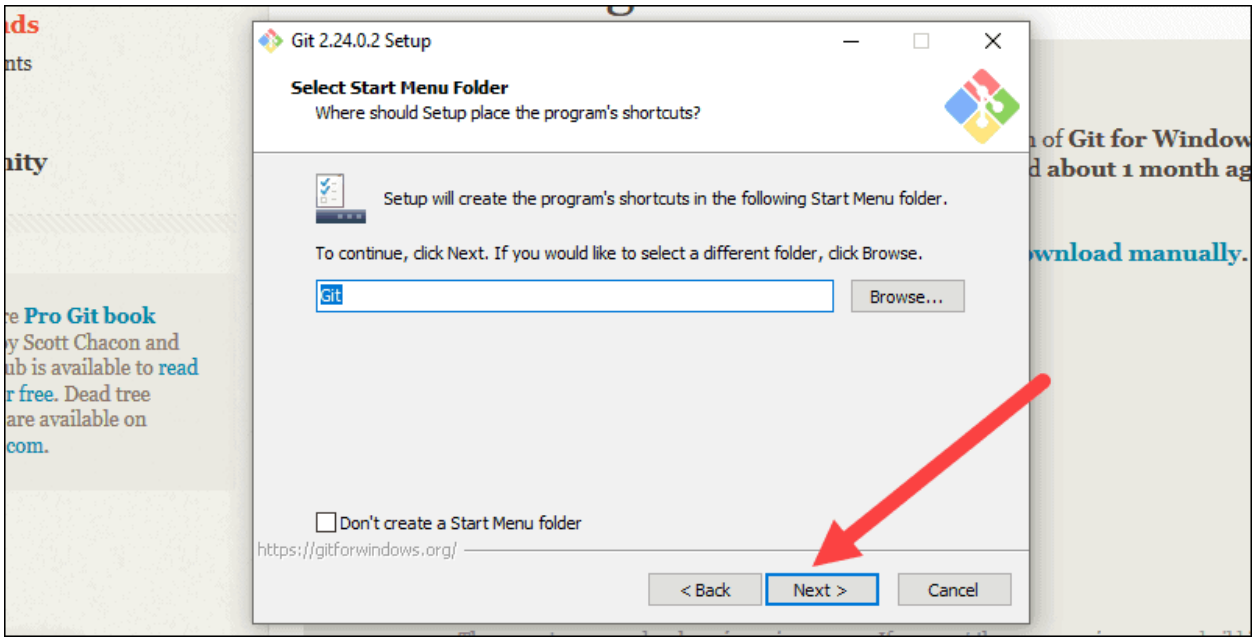
3.Review the GNU General Public License, and when you're ready to install, click **Next**.



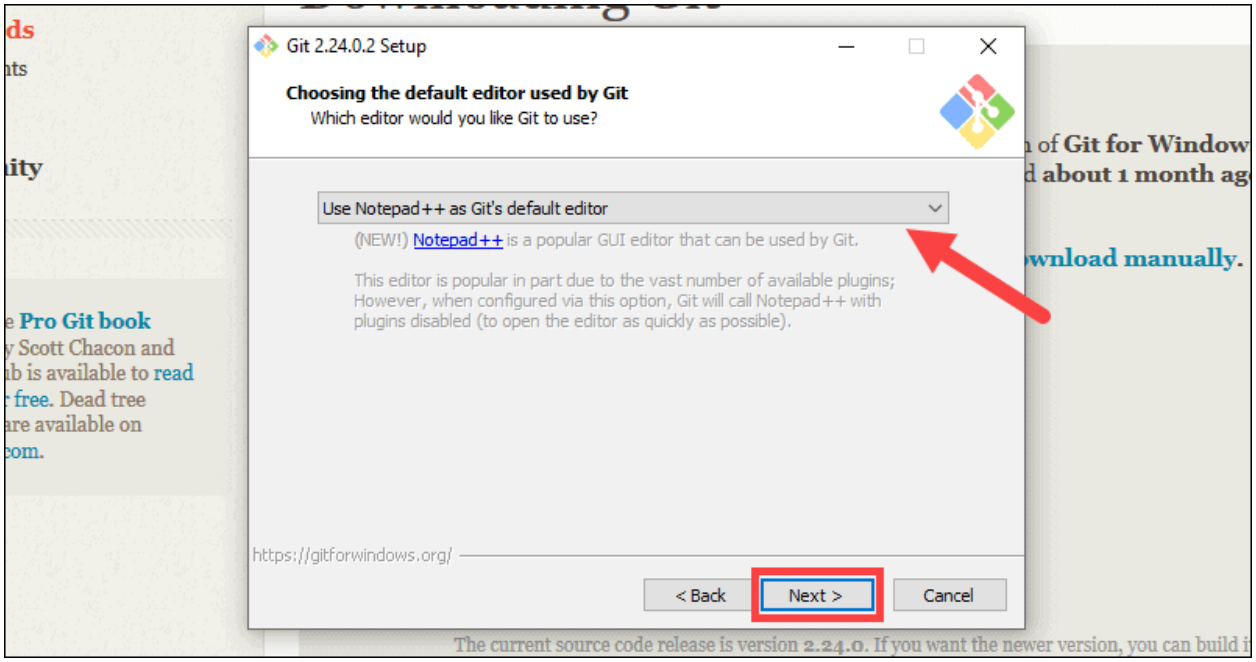
4. A component selection screen will appear. Leave the defaults unless you have a specific need to change them and click **Next**.



5.The installer will offer to create a start menu folder. Simply click **Next**.

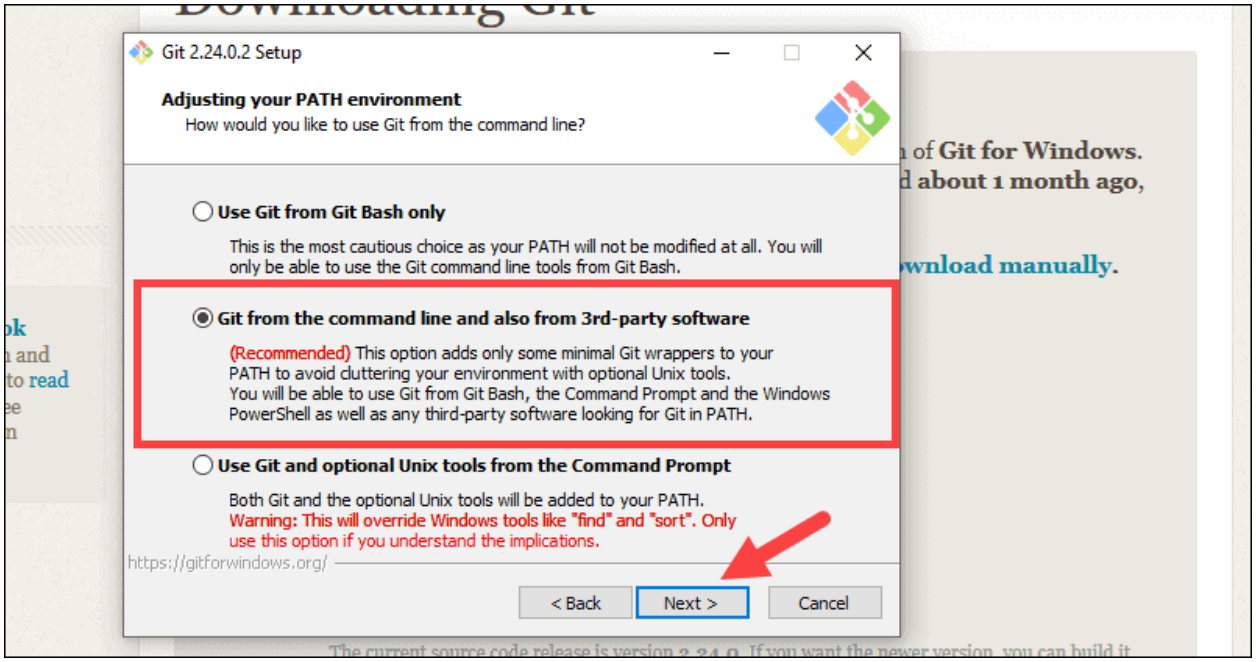


6. Select a text editor you'd like to use with Git. Use the drop-down menu to select Notepad++ (or whichever text editor you prefer) and click **Next**.

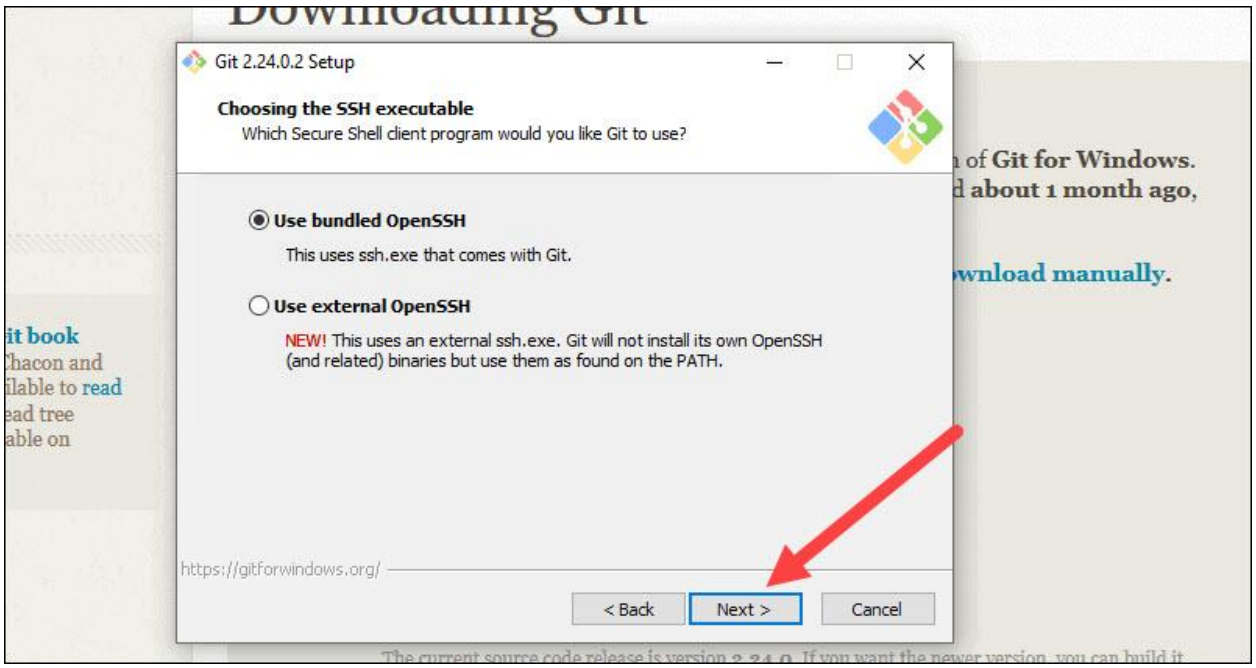


7.The next step allows you to choose a different name for your initial branch. The default is 'master.' Unless you're working in a team that requires a different name, leave the default option and click **Next**.

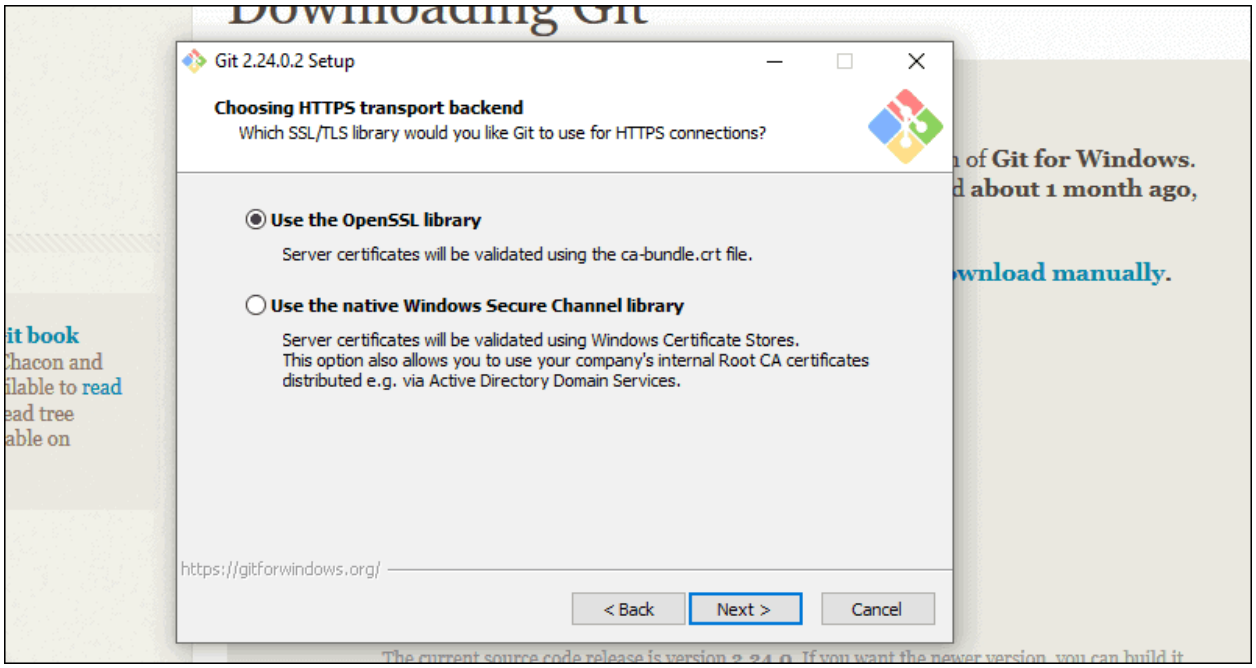
8.This installation step allows you to change the PATH environment. The PATH is the default set of directories included when you run a command from the command line. Leave this on the middle (recommended) selection and click **Next**.



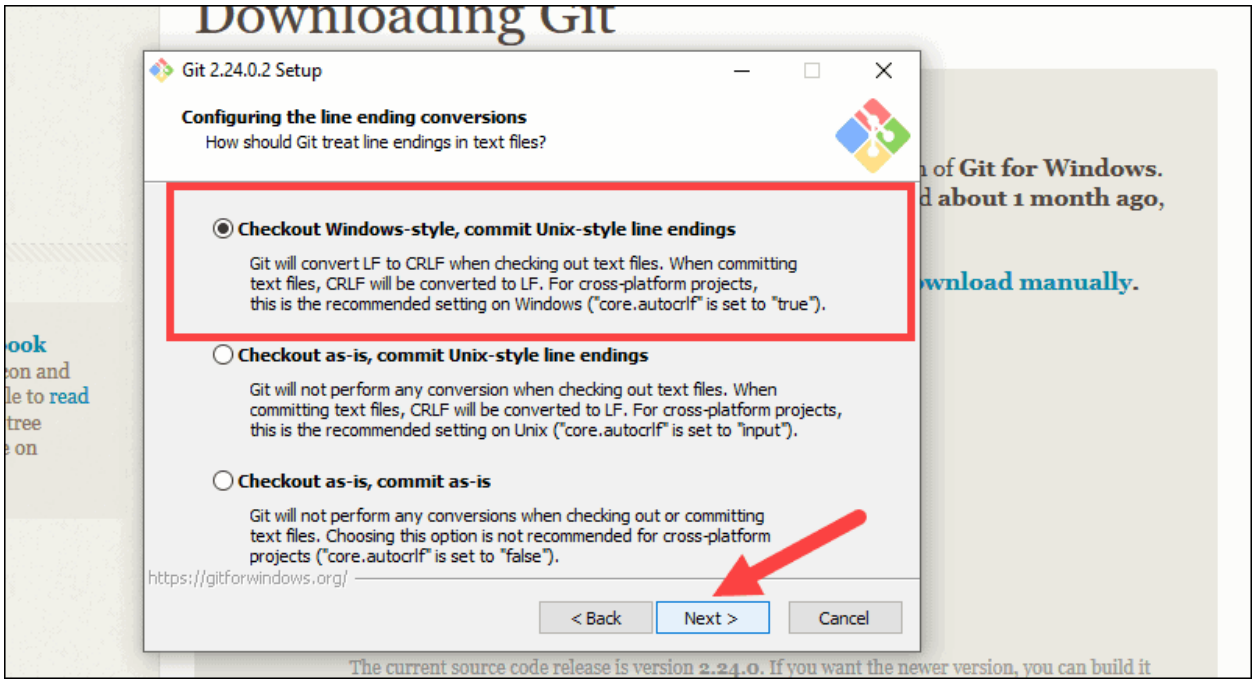
9.The installer now asks which SSH client you want Git to use. Git already comes with its own SSH client, so if you don't need a specific one, leave the default option and click **Next**.



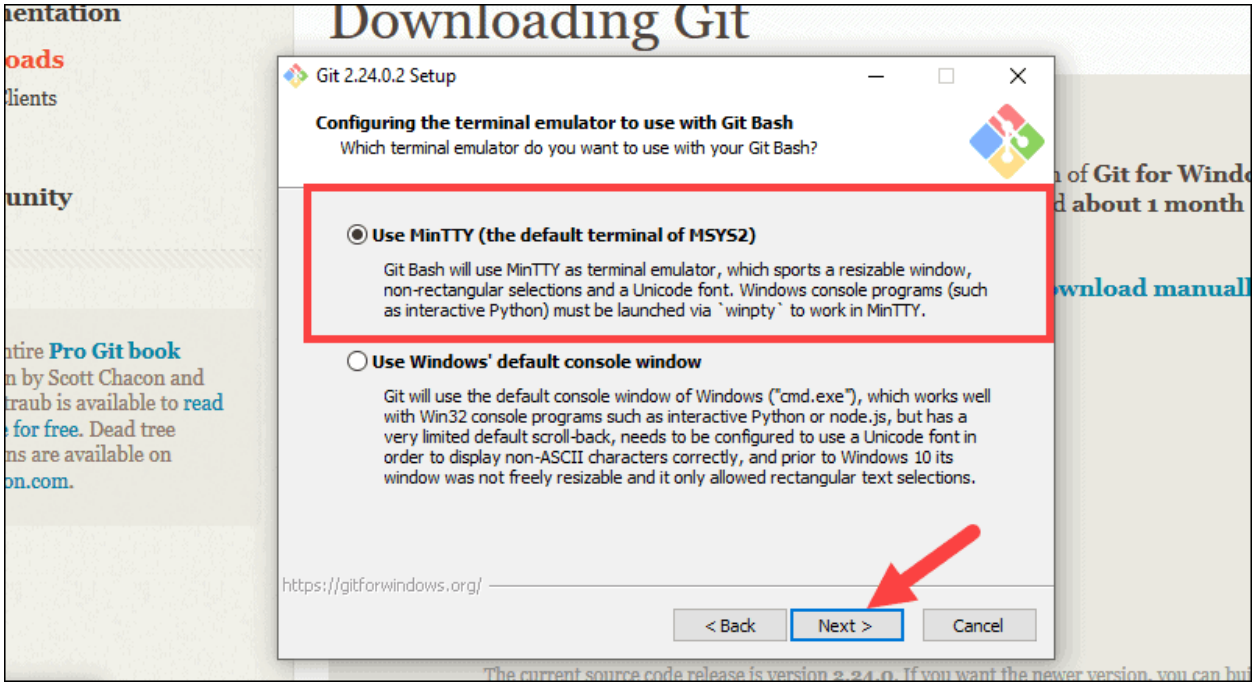
10.The next option relates to server certificates. Most users should use the default. If you're working in an Active Directory environment, you may need to switch to Windows Store certificates. Click **Next**.



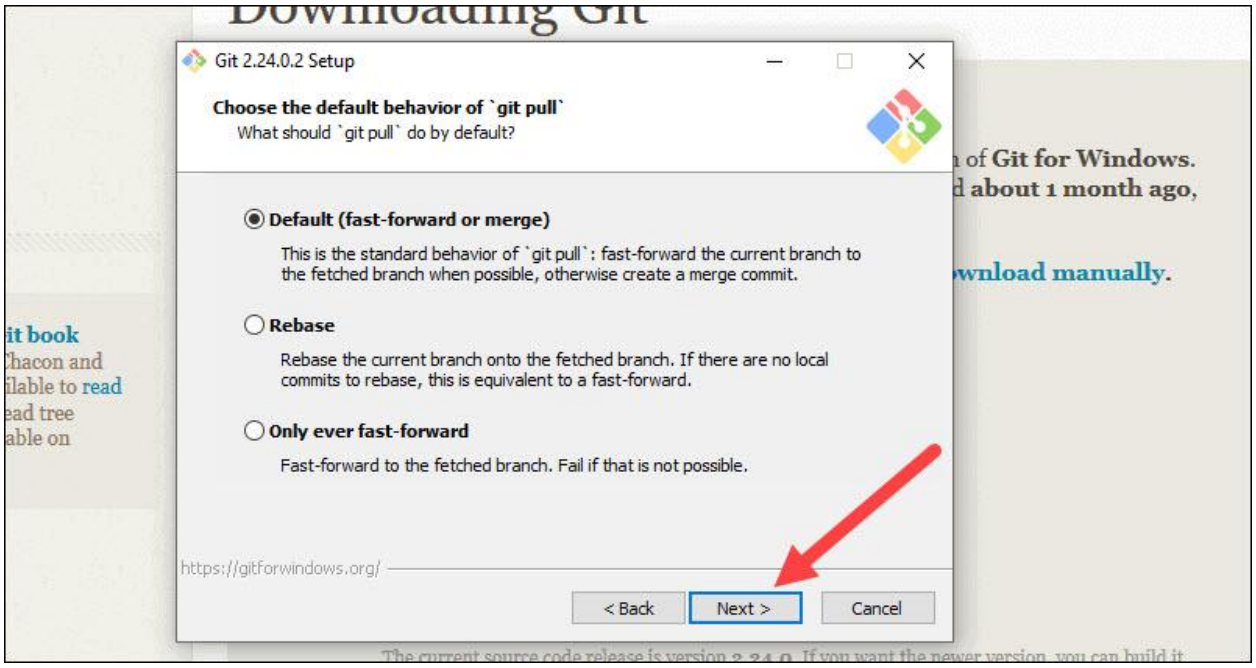
11.The next selection converts line endings. It is recommended that you leave the default selection. This relates to the way data is formatted and changing this option may cause problems. Click **Next**.



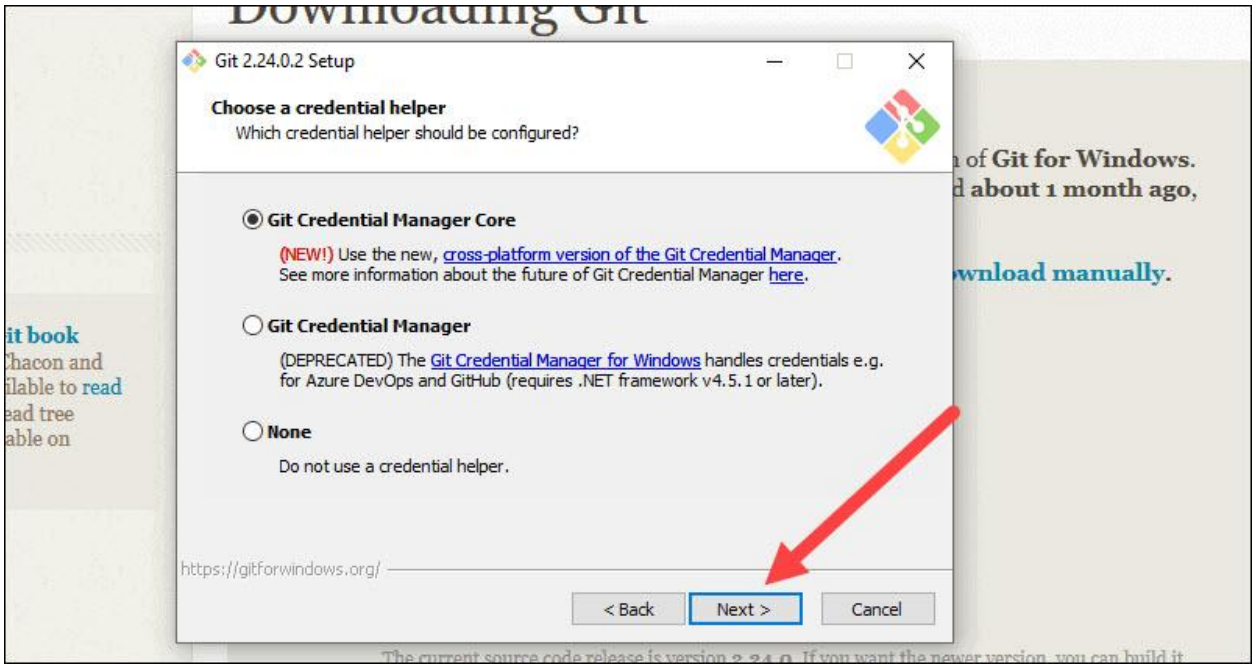
12.Choose the terminal emulator you want to use. The default MinTTY is recommended, for its features. Click **Next**.



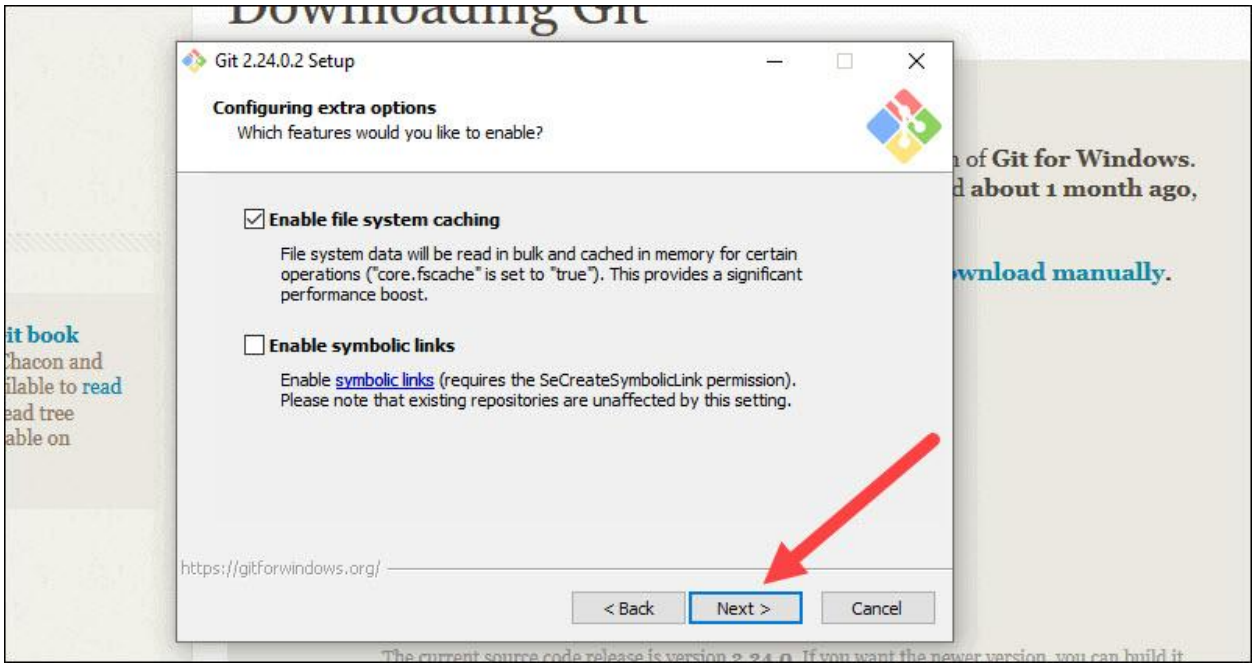
13.The installer now asks what the git pull command should do. The default option is recommended unless you specifically need to change its behavior. Click **Next** to continue with the installation.



14.Next you should choose which credential helper to use. Git uses credential helpers to fetch or save credentials. Leave the default option as it is the most stable one, and click **Next**.

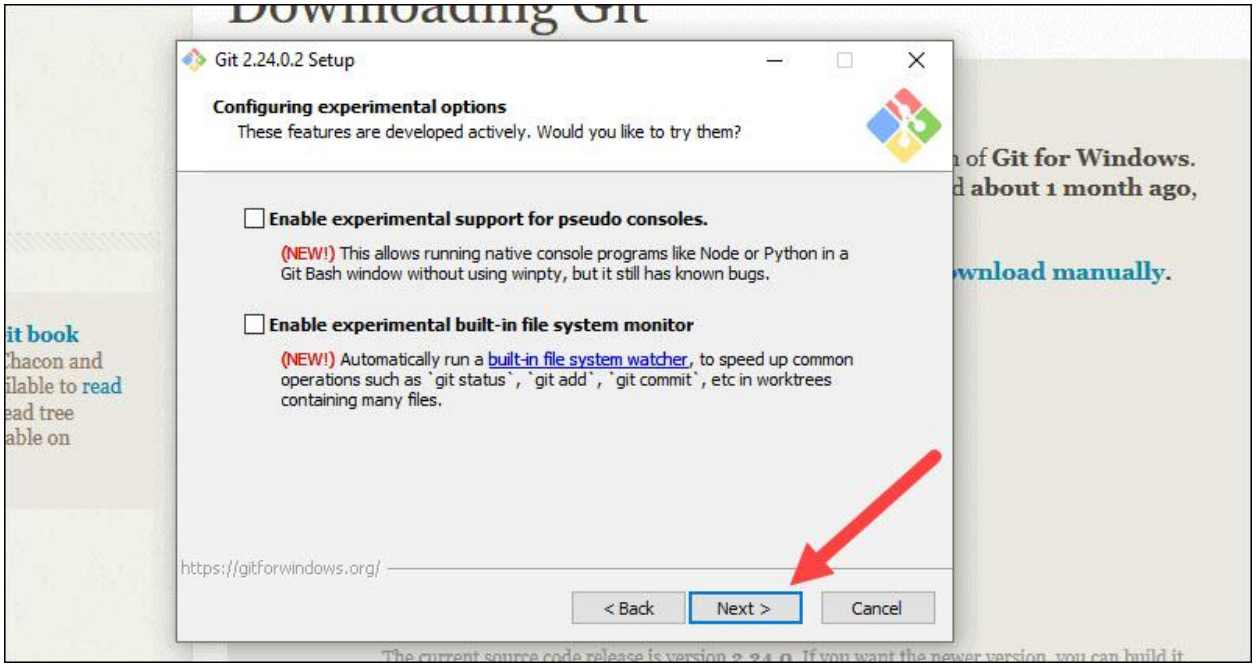


15.The default options are recommended, however this step allows you to decide which extra option you would like to enable. If you use symbolic links, which are like shortcuts for the command line, tick the box. Click **Next**.

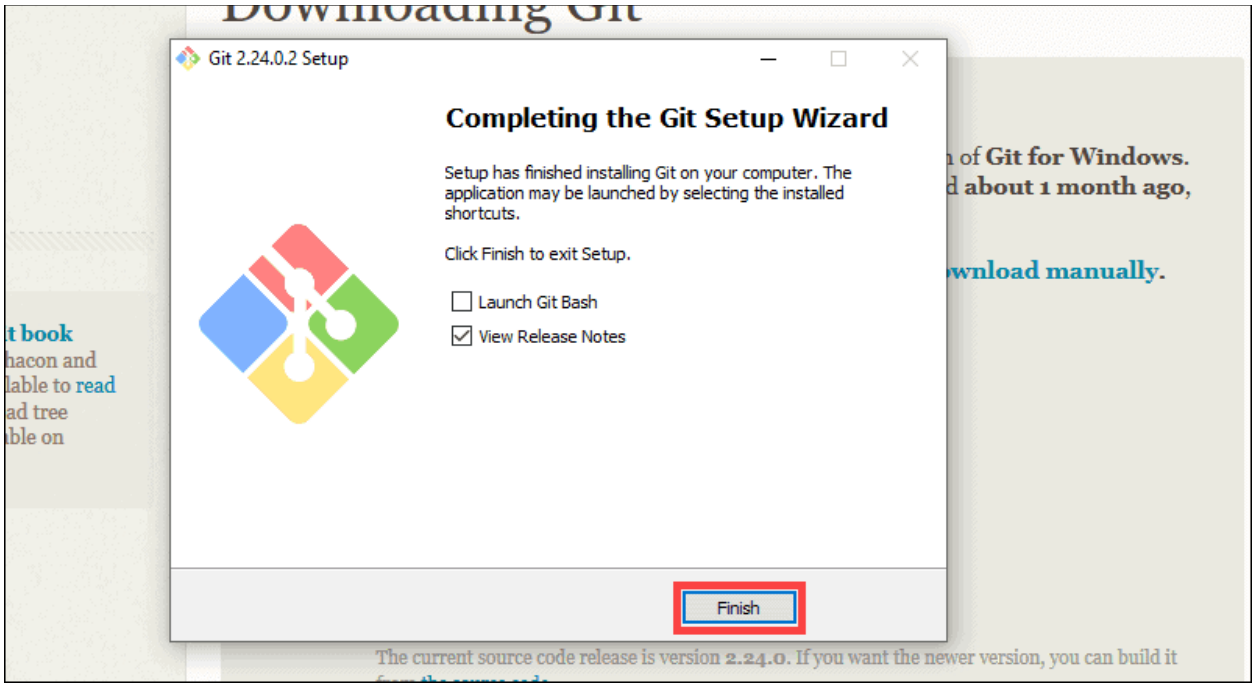


16.Depending on the version of Git you're installing, it may offer to install experimental features. At the time this article was written, the options to include support for pseudo controls and a built-in file system monitor were

offered. Unless you are feeling adventurous, leave them unchecked and click **Install**.



17.Once the installation is complete, tick the boxes to view the Release Notes or Launch Git Bash, then click **Finish**.



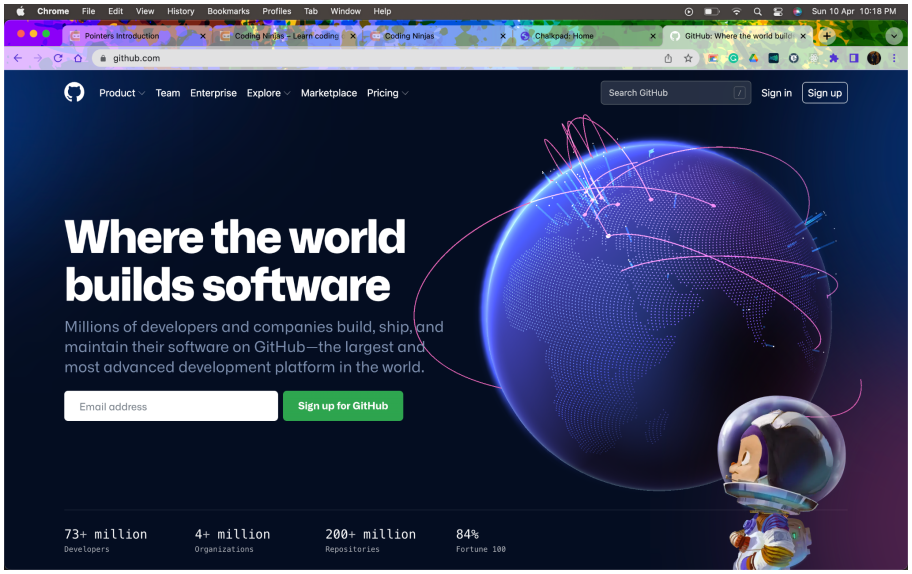
Now launch Git Bash.

Type **“git --version”** and press Enter. It will show the version you just installed.

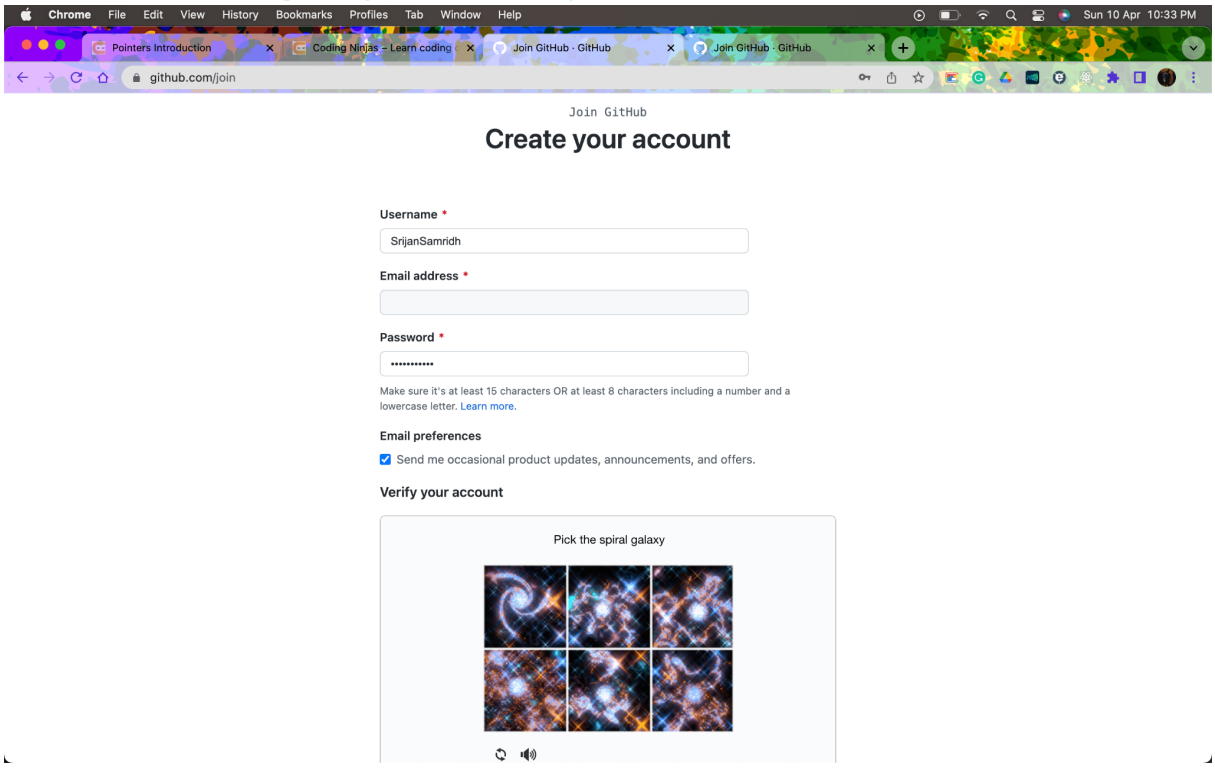
Congrats! You have successfully downloaded and installed git.

2. SETTING UP GITHUB ACCOUNT

- **STEP 1: GITHUB HOME PAGE (https://github.com/)**



- **STEP 2: Go to https://github.com/join in a web**

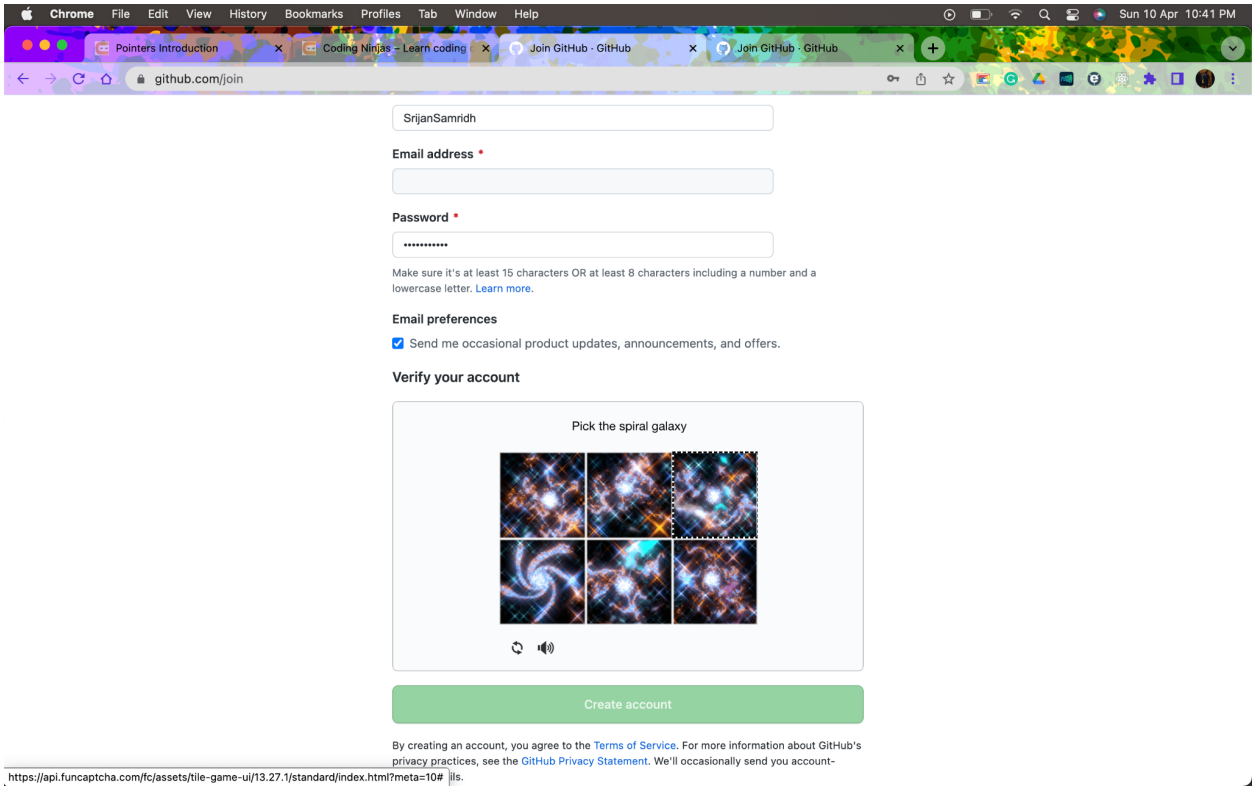


browser.

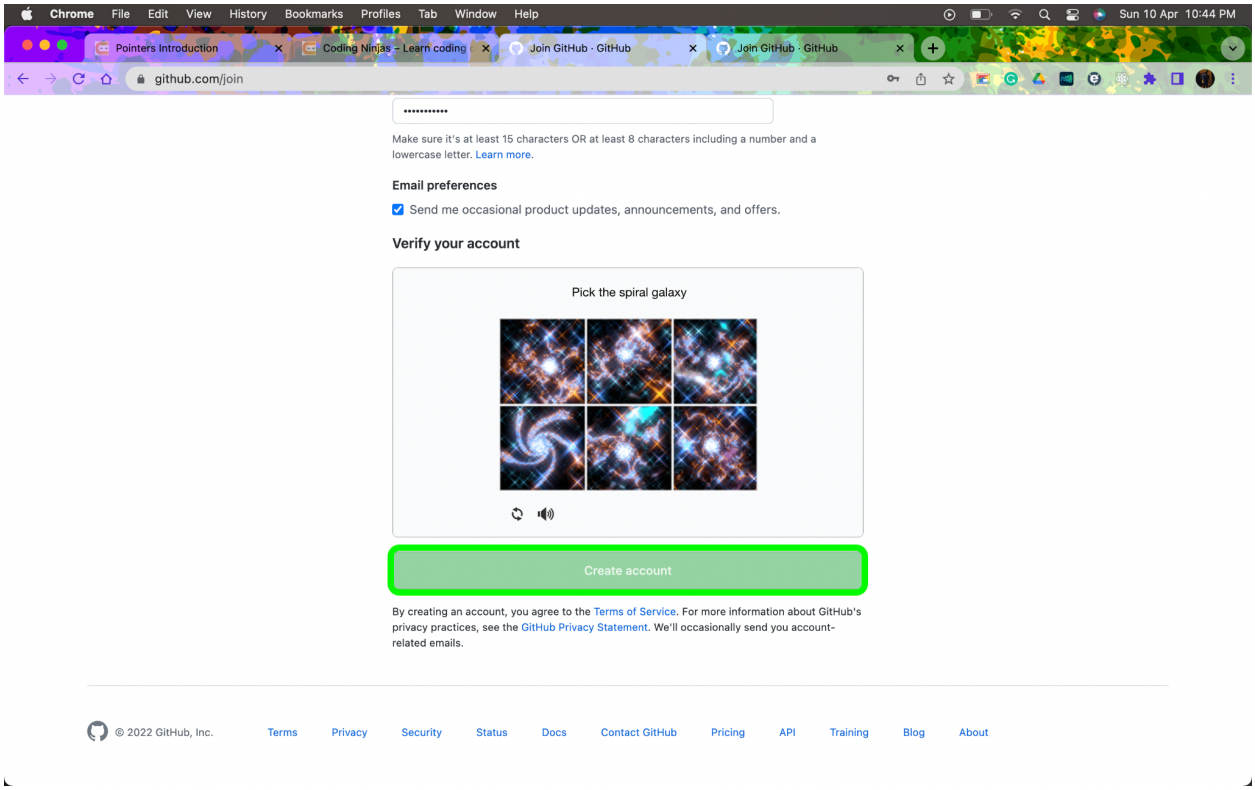
- **STEP 3: Enter your personal details.In addition to creating a username and entering an email address, you'll also have to create a password. Your password must be at**

least 15 characters in length *or* at least 8 characters with at least one number and lowercase letter.

- **STEP 4: Complete the CAPTCHA puzzle.**

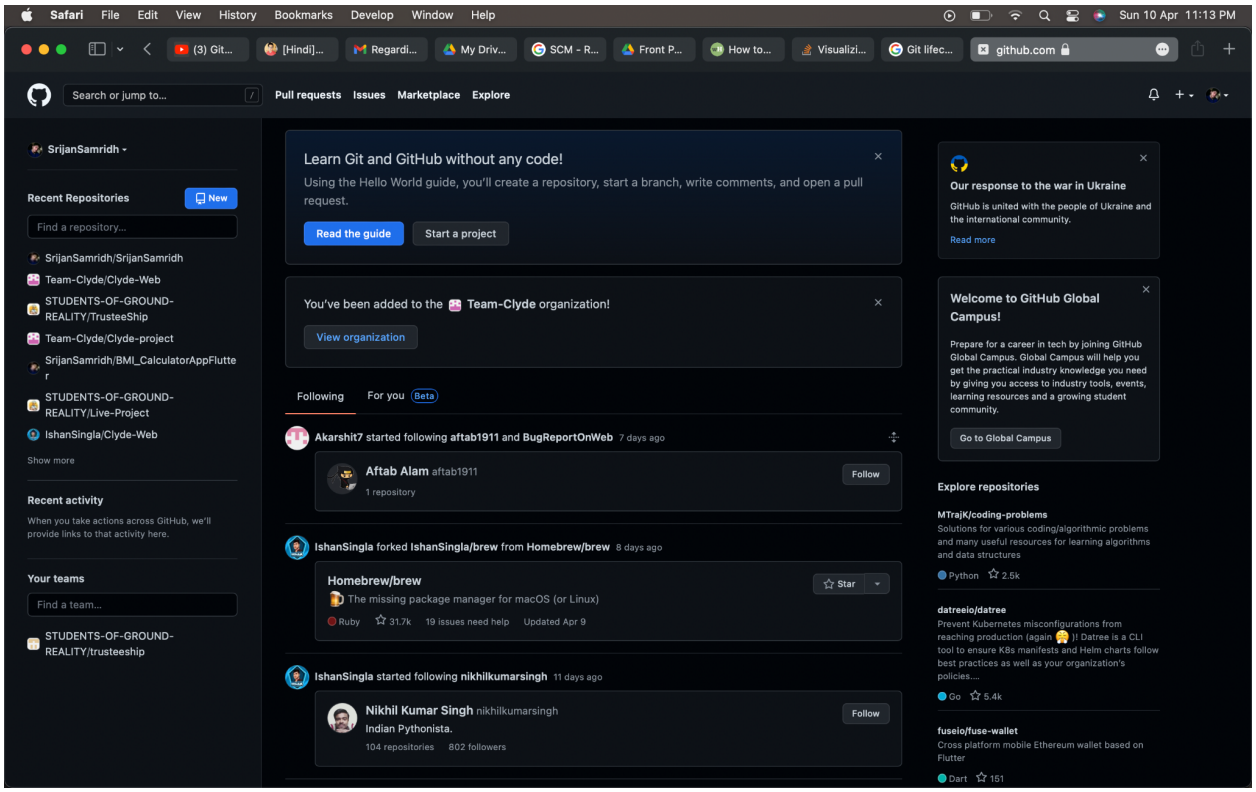


- **STEP 5: Click the green Create Account button.**



- **STEP 6: Click the Choose button for your desired plan.** Once you select a plan, GitHub will send an email confirmation message to the address you entered. The plan options are:
    - **Free:** Unlimited public and private repositories, up to 3 collaborators, issues and bug tracking, and project management tools.
    - **Pro:** Unlimited access to all repositories, unlimited collaborators, issue & bug tracking, and advanced insight tools.
    - **Team:** All of the aforementioned features, plus team access controls and user management.
    - **Enterprise:** All of the features of the Team plan, plus self-hosting or cloud hosting, priority support, single sign-on support, and more.

- **STEP 7: Click the `Verify email address` button in the message from GitHub.** This confirms your email address and returns you to the sign-up process.
- **STEP 8: Review your plan selection and click `Continue`.** You can also choose whether you want to receive updates from GitHub via email by checking or unchecking the "Send me updates" box.
    - If you choose a paid plan, you'll have to enter your payment information as requested before you can continue.
- **STEP 9: Select your preferences and click `Submit`.** GitHub displays a quick survey that can help you tailor your experience to match what you're looking for. Once you make your selection, you'll be taken to a screen that allows you to set up your first repository.
    - If you want to upgrade your Github account in the future, click the menu at the top-right corner, select **Settings**, and choose **Billing** to view your options.



3. GENERATE LOGS

Git log - This command is used to view the history of commits of a repository.

Step 1: First of all, create a folder or choose a folder you want to make as a Git Repository.

Step 2: Open the folder, right-click on the working area, and choose "Git Bash Here."
A terminal window will be shown to you.
Step 3: Type the *"git init"* command on the terminal and hit enter to initialize the directory with the .git/ folder.
After performing the above command, you'll not see any change in the *"Git Tut"* folder, because the .git/ folder is hidden by default.
So, you've perfectly created a repository. Now, I'll explain some commands:

1. **git init -** This command makes the directory a Git repository.
2. **git status -** This command is used to check the status of the repository.
3. **git --version -** This command is used to check the version of the Git installed on our system.
4. **git add . -** This command adds all the files of a repository to the staging area.
5. **git commit -m "message" -** This command is used to commit all the changes. It creates a "save point," i.e., we can revert to the old savepoint when needed.
6. **git log -** This command is used to view the history of commits of a repository.
7. **git config --global user.name "Name" -** This command is used to configure or set the user name.
8. **git config --global user.email "email id" -** This command is used to set email.

Let's have a practical demonstration

Step 1: Create a file or anything that you want in the Git repository. I've created a file named *"index.txt"*. So, after making it, when I perform the git status command, then it shows that there is one untracked file, and you can see "index.txt" in red color.

Step 2: To add *"index.txt"* to the staging area, use "`git add .`" command. Execute the git status command again, and you will see that there is no untracked file, and we're ready to commit.

Step 4: After configuring email and user name, you are all set to commit your changes. Execute the command given below to commit:

```
git commit -m "Your commit message here"
```

Step 5: Now that we've made commit, let's use the "git log" command to check it.

```
AYUSH@DESKTOP-JG2JTD1 MINGW64 ~/OneDrive/Desktop/Git Tut (master)
$ git init
Initialized empty Git repository in C:/Users/AYUSH/OneDrive/Desktop/Git Tut/.git/

AYUSH@DESKTOP-JG2JTD1 MINGW64 ~/OneDrive/Desktop/Git Tut (master)
$ git status
On branch master

No commits yet

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

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

AYUSH@DESKTOP-JG2JTD1 MINGW64 ~/OneDrive/Desktop/Git Tut (master)
$ git add .

AYUSH@DESKTOP-JG2JTD1 MINGW64 ~/OneDrive/Desktop/Git Tut (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   index.txt
```

```
AYUSH@DESKTOP-JG2JTD1 MINGW64 ~/OneDrive/Desktop/Git Tut (master)
$ git commit -m "Commit 1"
[master (root-commit) 30615b3] Commit 1
 1 file changed, 1 insertion(+)
 create mode 100644 index.txt

AYUSH@DESKTOP-JG2JTD1 MINGW64 ~/OneDrive/Desktop/Git Tut (master)
$ git log
commit 30615b36a76958baab6dd4807568d78da40bb3d0 (HEAD -> master)
Author: ayushjawa <ayushjawa02@gmail.com>
Date:   Sun Apr 10 23:22:44 2022 +0530

    Commit 1

AYUSH@DESKTOP-JG2JTD1 MINGW64 ~/OneDrive/Desktop/Git Tut (master)
$
```

4. CREATE AND VISUALIZE BRANCHES

- **BRANCHES**

  Suppose you're working on some project, and you want to check the features of it, but you can't afford to make changes in the actual project because if something wrong happened, it would affect you adversely. So, for that purpose, you can create a separate branch and manipulate the code because whatever you'll do in this new branch will not be reflected in the main or master branch.

Branches in Git are very similar to branches of a tree, i.e., you can put a unique set of code in a branch and can manipulate it, and there will be no effect on the actual or master branch.

We can say that branches can be used for testing purposes by an individual contributor in some projects.

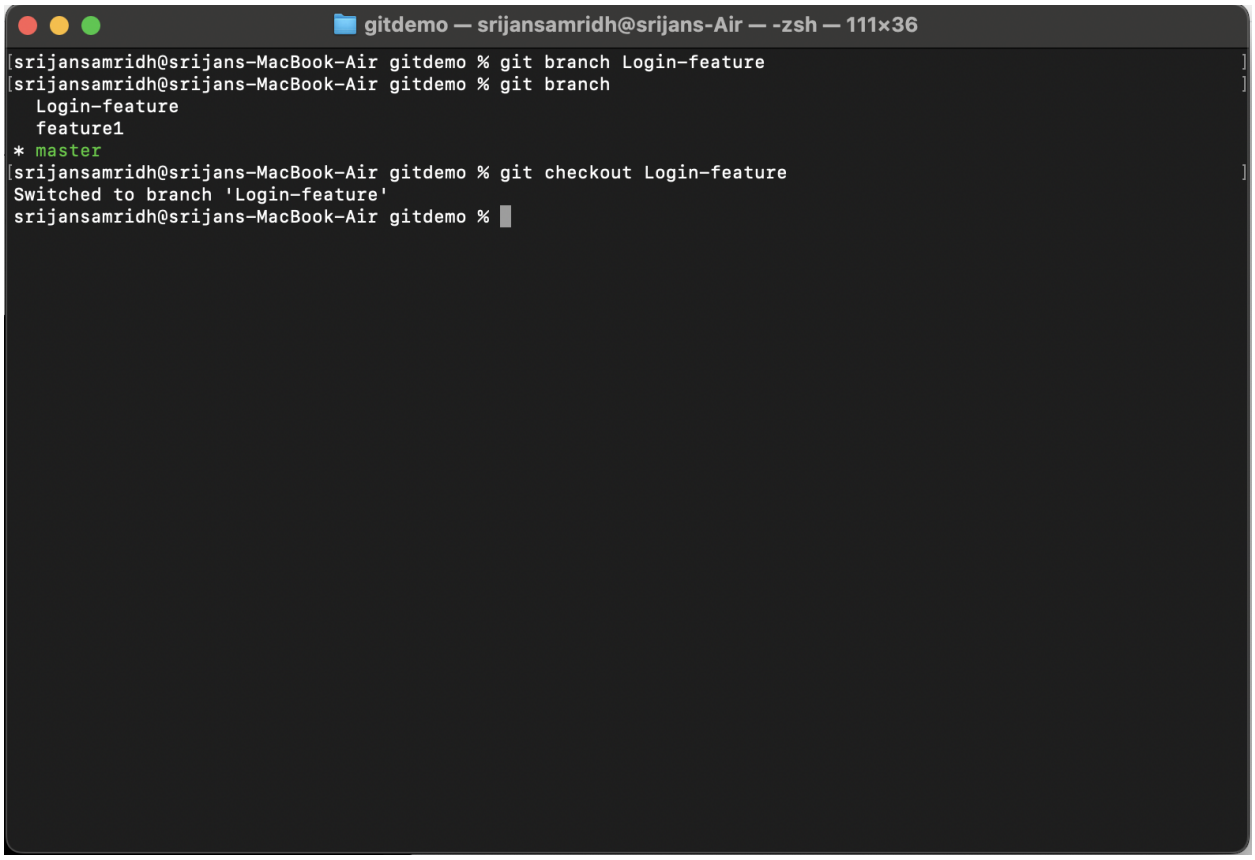
Let's see how we can create a branch in Git.

Step 1: Open the local repository, then open Mac Terminal/Git Bash.
Step 2: Now perform these commands,

- **git branch branch_name -** This command will create a new branch with the specified name.
- **git branch -** This command will show all the available branches present in the repository.

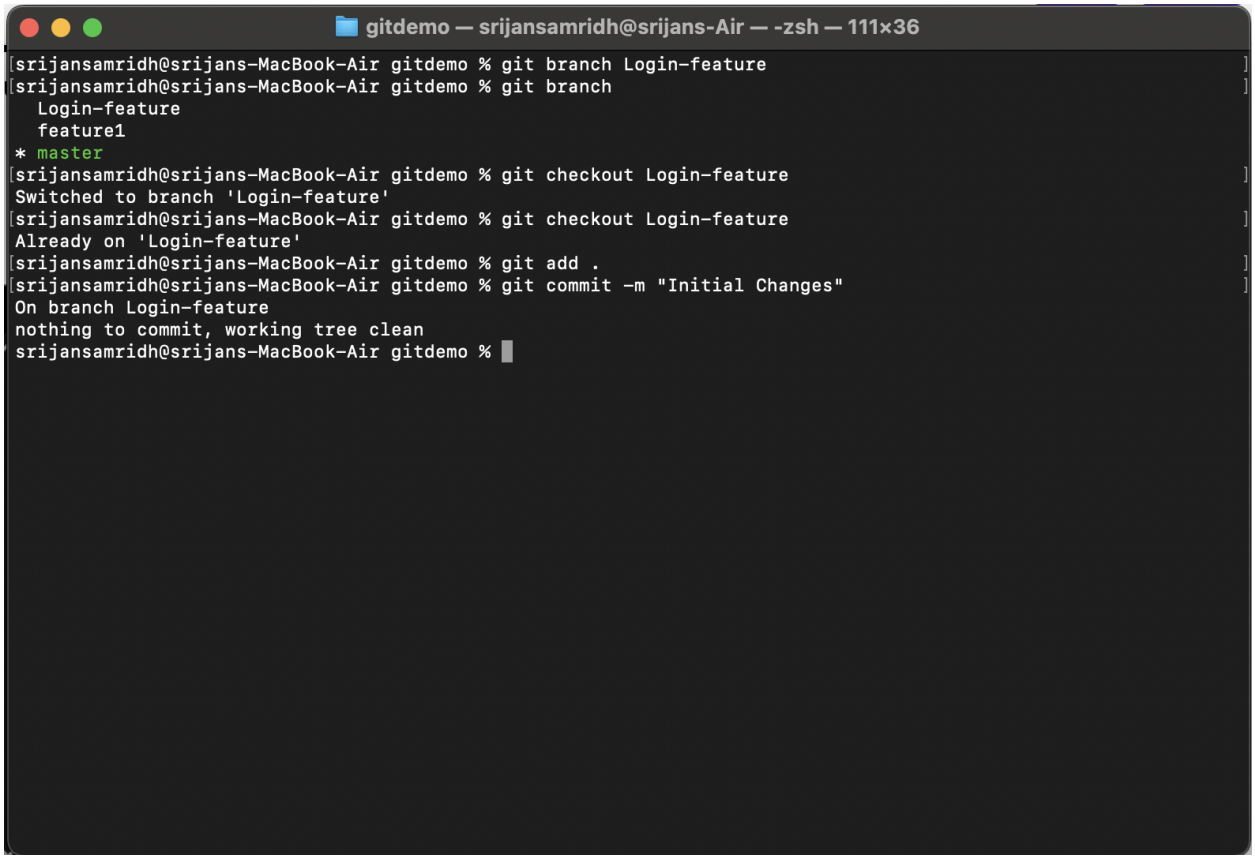
Note: By default the branch is set to "master".

- **git checkout branch_name -** This command will switch the master branch to the specified branch name.



So, I've created a branch with the name 'Login-feature'.
Now, you can make whatever changes or features you want to add to your project because these changes won't be reflected in the master branch. So there's no need to be afraid now, that, if these changes don't work, you might lose the whole project. That's the actual use or importance of branches in Git.

After making all the changes, you've to put the changes into the staging area, and then you can commit them.



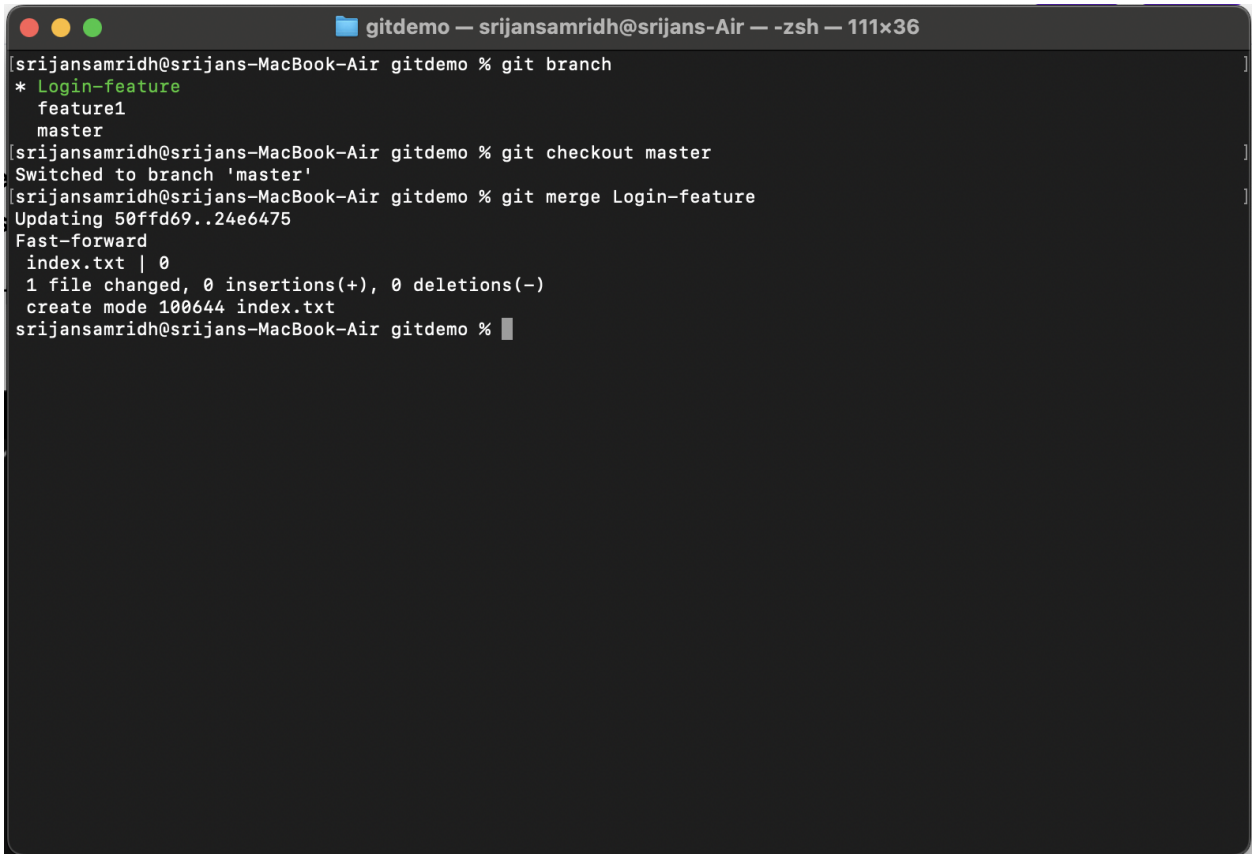
Now you can even push these changes to the remote repository, for that perform this command:

- **git push origin branch_name -** It will push all the changes of the specified branch to the remote repository.

If you see your GitHub repository where you have pushed the changes, you'll find two branches over there, and you'll notice that the changes made in one branch will not be reflected in the other.

Suppose the feature you added was useful, and now you want to merge it with the master branch, then, first of all, you've to switch to master branch, then you've to execute this command,

- **git merge branch_name -** It will merge all the content of the specified branch with the master branch.



Now you can push this to a remote repository.

5. GIT LIFECYCLE DESCRIPTION

In this chapter, we will discuss the life cycle of Git. In later chapters, we will cover the Git commands for each operation.

General workflow is as follows −

- You clone the Git repository as a working copy.
- You modify the working copy by adding/editing files.
- If necessary, you also update the working copy by taking other developer's changes.
- You review the changes before commit.
- You commit changes. If everything is fine, then you push the changes to the repository.
- After committing, if you realize something is wrong, then you correct the last commit and push the changes to the repository.

Shown below is the pictorial representation of the work-flow.