

Subject Name: **Source Code Management**

Subject Code: **CS181**

Cluster: **BETA**

Department: **CSE**

CHITKARA
UNIVERSITY



Submitted By:

Aakash Rohilla

2110990006

G01

Submitted To:

Monit Kapoor

<u>EXPERIMENT</u>	<u>TOPIC</u>	<u>PAGE No.</u>
<u>EXPERIMENT -1</u>	<u>Setting up of Git Client</u>	
<u>EXPERIMENT-2</u>	<u>Setting up Git Hub Account</u>	
<u>EXPERIMENT-3</u>	<u>How to Use Git Log</u>	

<u>EXPERIMENT-4</u>	<u>Create and visualize Branch</u>	
<u>EXPERIMENT-5</u>	<u>Life Cycle of the Git</u>	

Experiment-1

AIM :- Setting up of Git Client

Steps For Installing Git for Windows

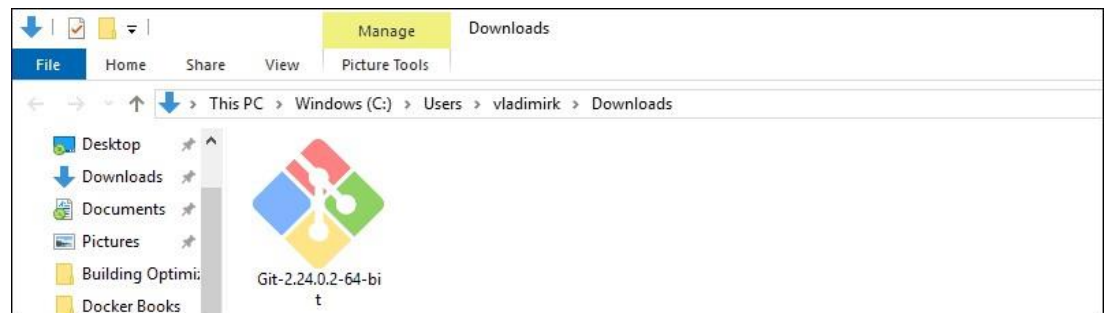
Download Git for Windows

1. Browse to the official Git Website: <https://git-scm.com/downloads>
2. click the download link for Windows and allow the download to complete.

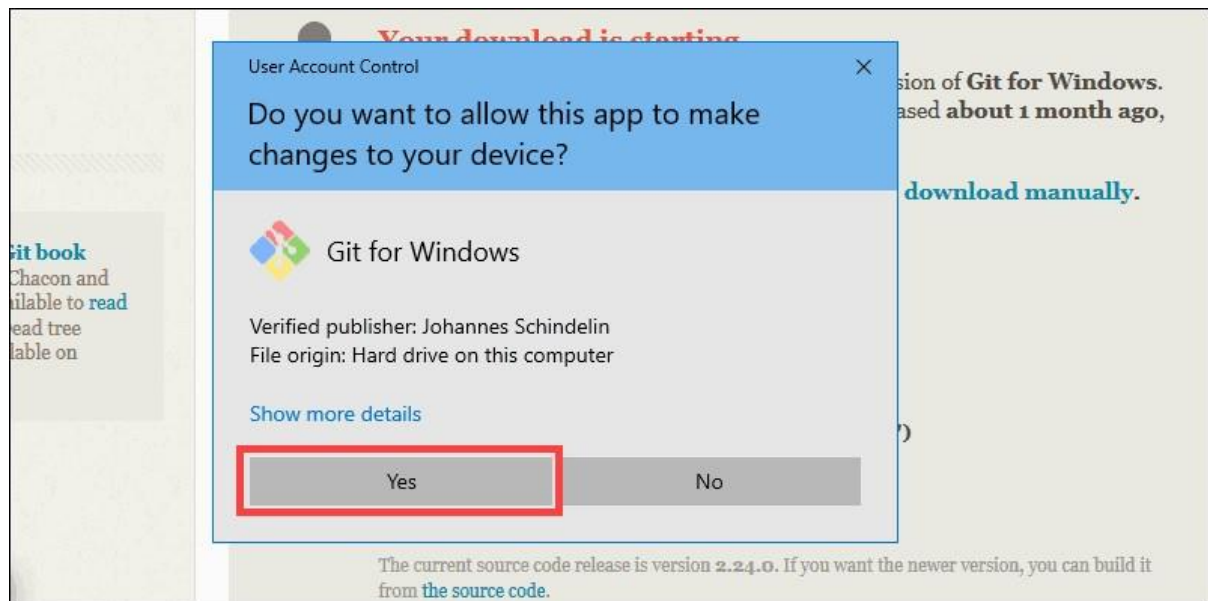


Extract and Launch Git Installer

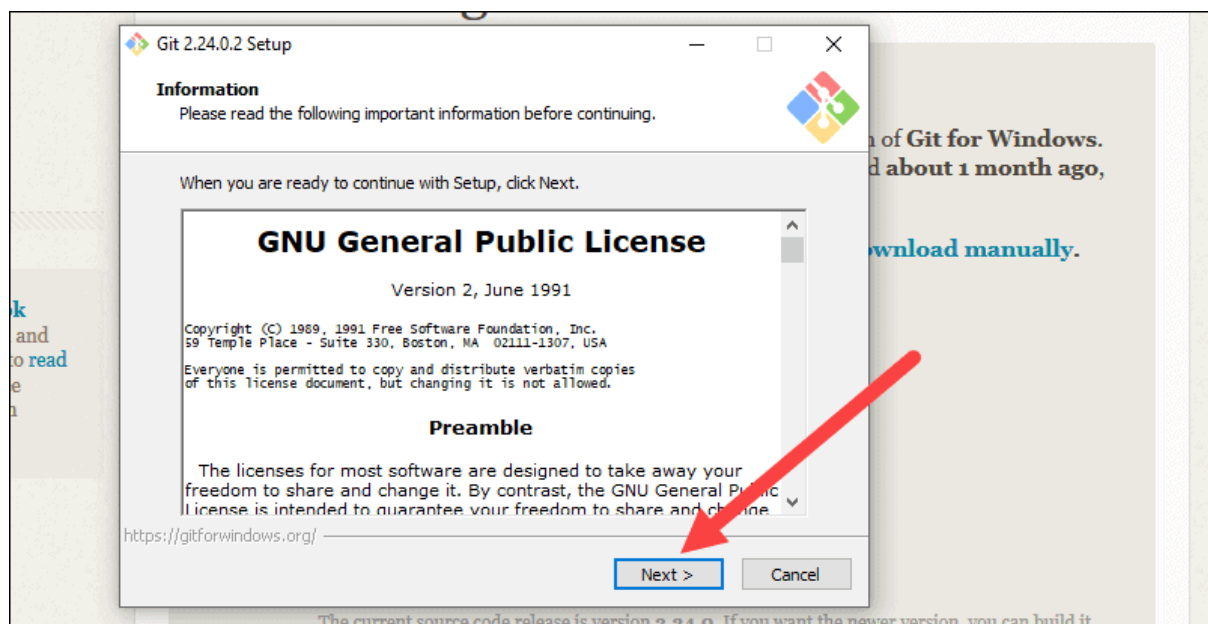
3. Browse to the download location (or use the download shortcut in your browser). Doubleclick the file to extract and launch the installer.



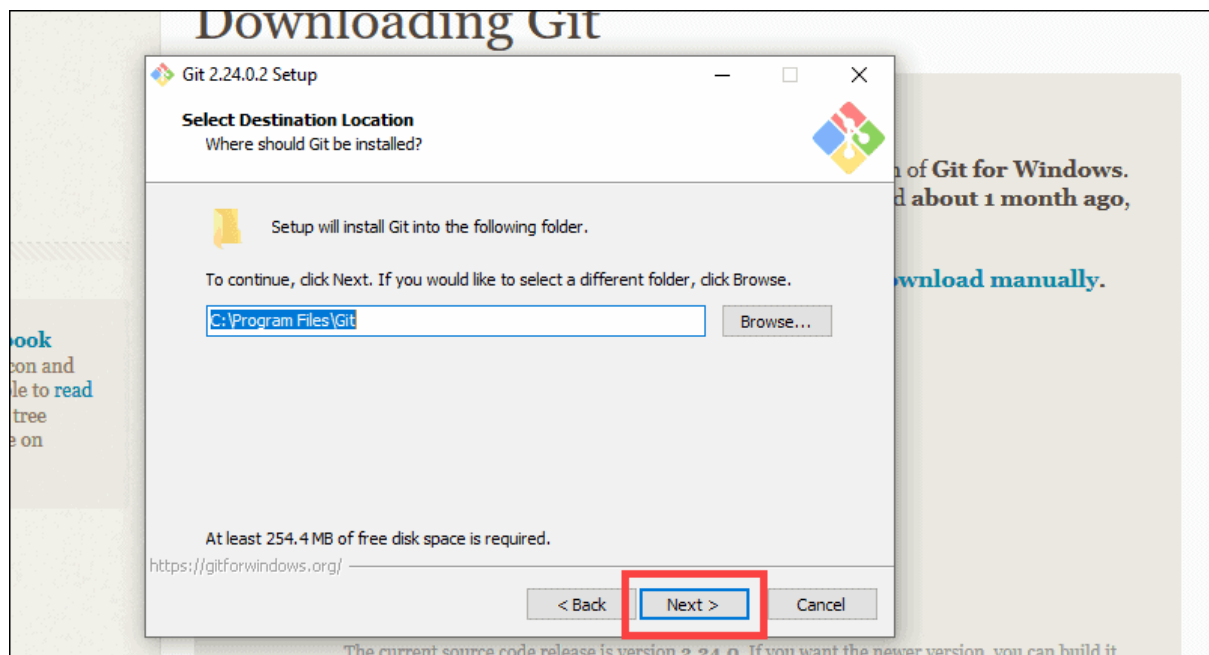
4. Allow the app to make changes to your device by clicking **Yes** on the User Account Control dialog that opens.



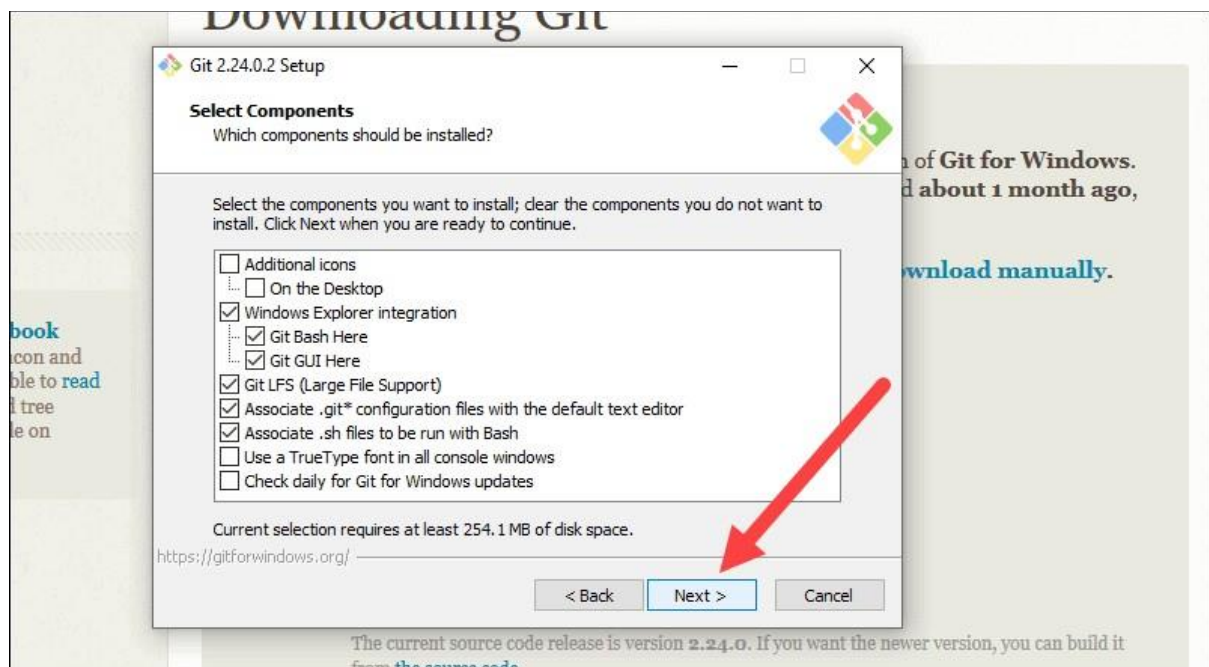
5. Review the GNU General Public License, and when you're ready to install, click **Next**.



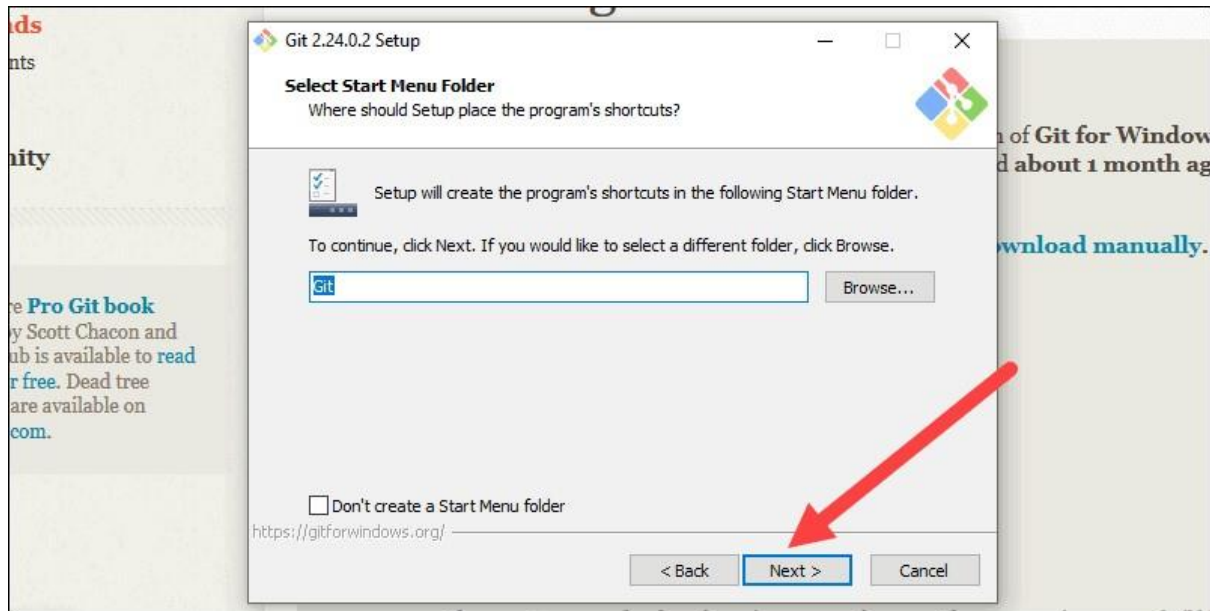
- The installer will ask you for an installation location. Leave the default, unless you have reason to change it, and click **Next**.



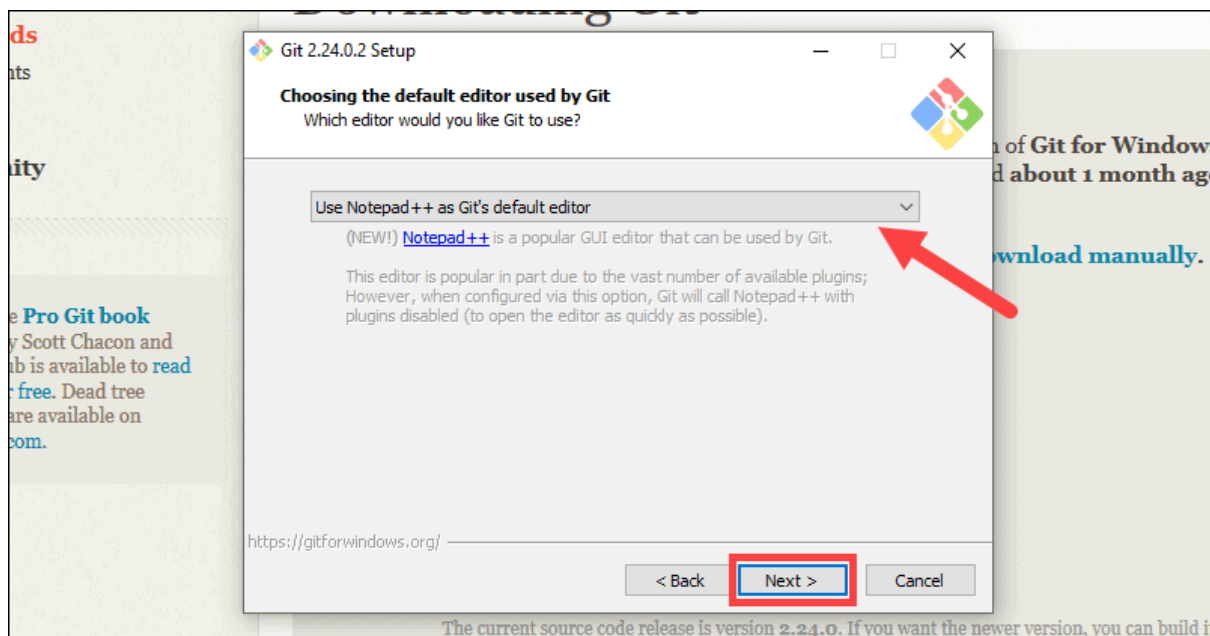
- A component selection screen will appear. Leave the defaults unless you have a specific need to change them and click **Next**.



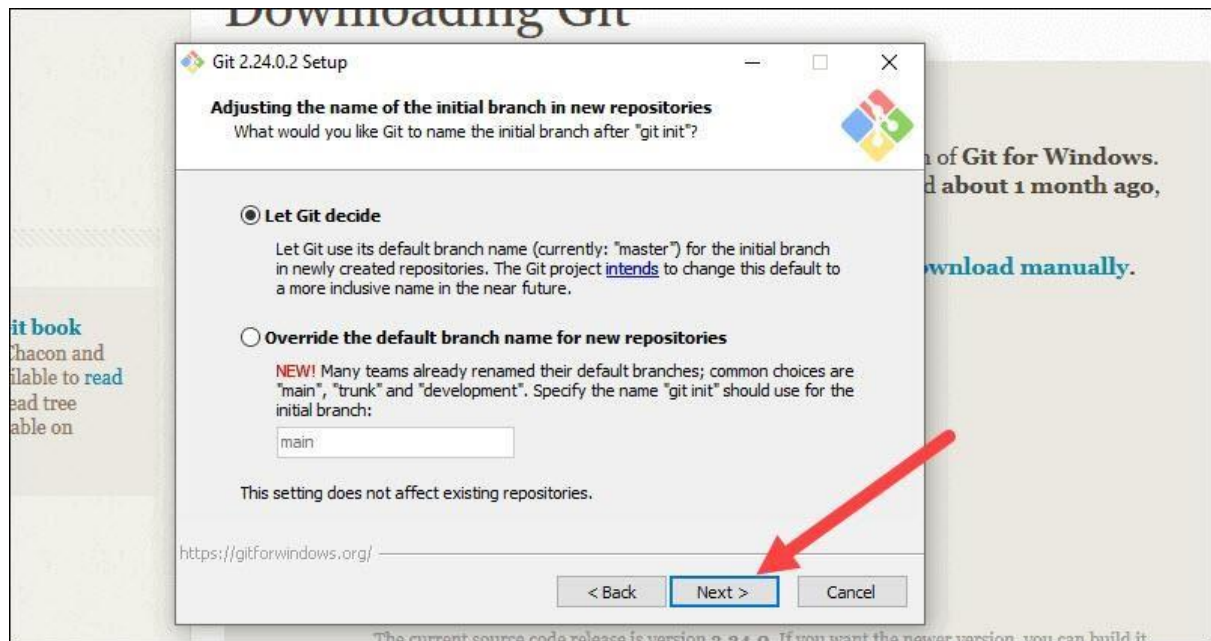
8. The installer will offer to create a start menu folder. Simply click **Next**.



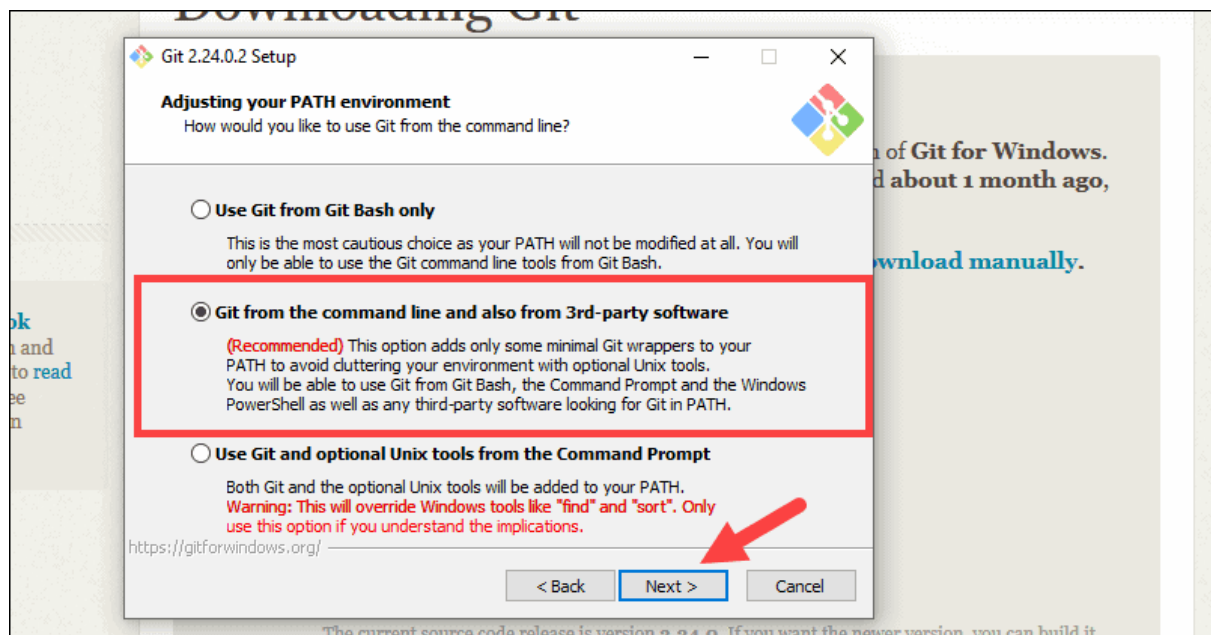
9. Select a text editor you'd like to use with Git. Use the drop-down menu to select Notepad++ (or whichever text editor you prefer) and click **Next**.



10. 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**.

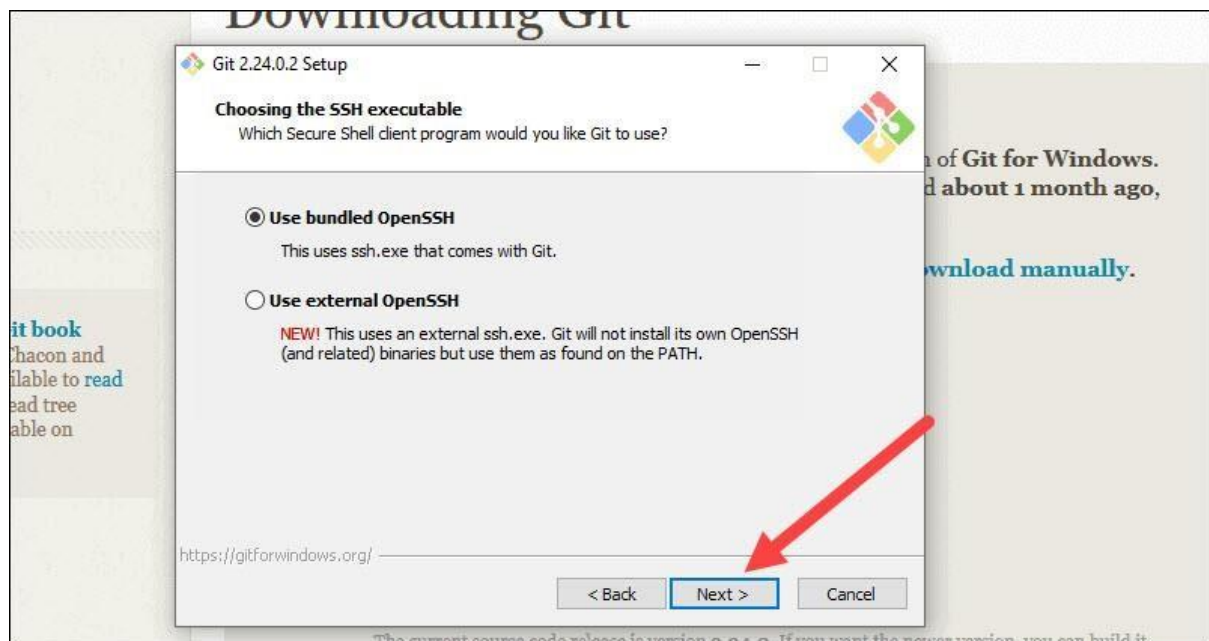


11. This installation step allows you to change the **PATH environment**. The **PATH** is the default set of directories included when you run a command from the command line. Leave this on the middle (recommended) selection and click **Next**.

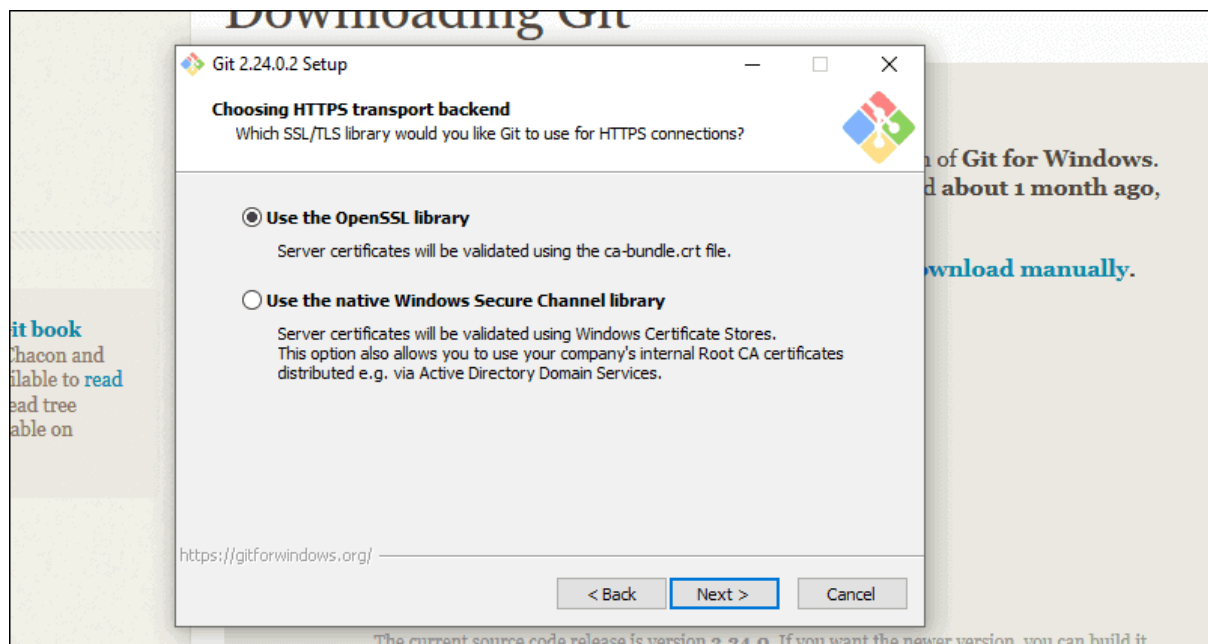


Server Certificates, Line Endings and Terminal Emulators

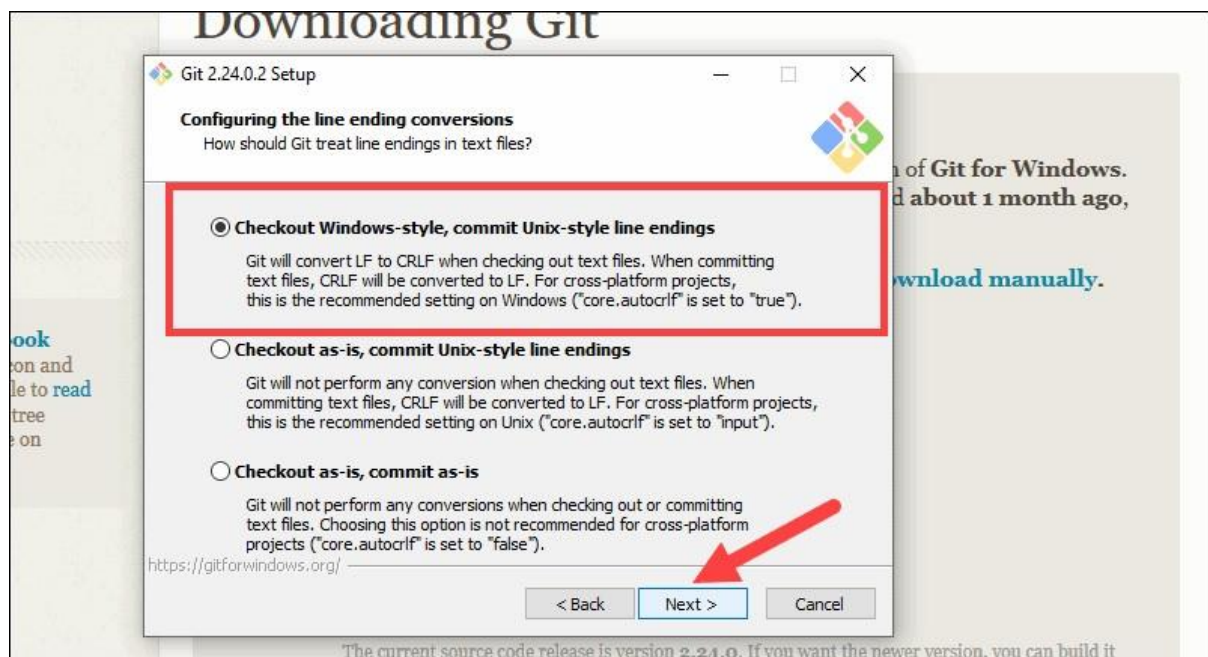
12. The installer now asks which SSH client you want Git to use. Git already comes with its own SSH client, so if you don't need a specific one, leave the default option and click **Next**.



13. 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**.



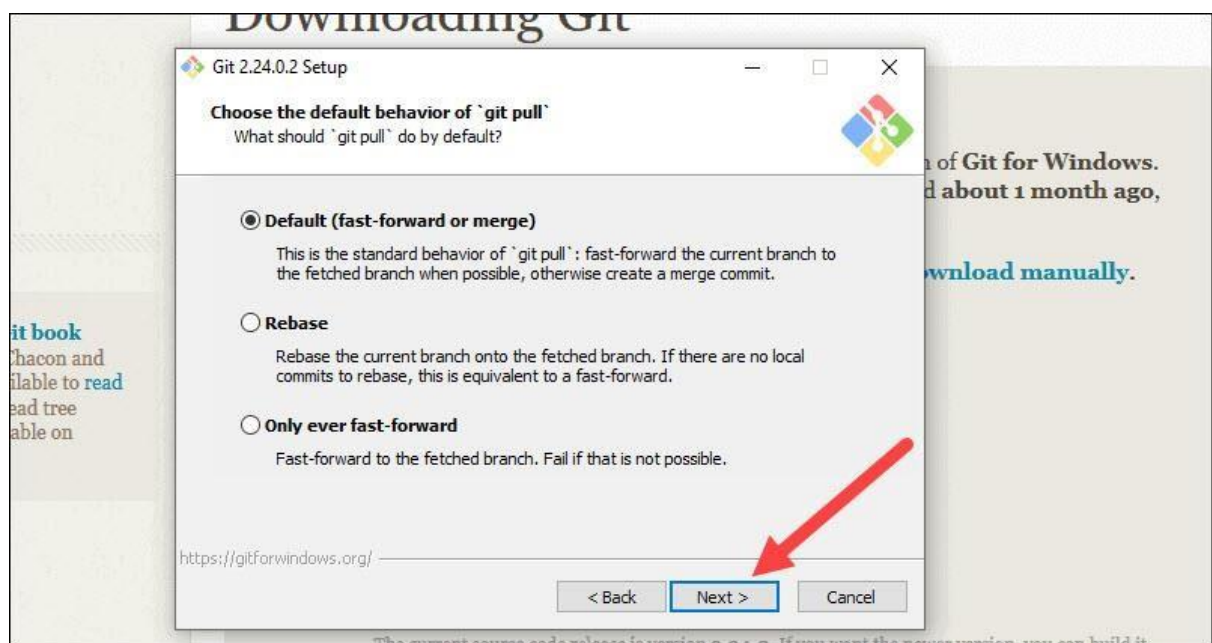
14. The next selection converts line endings. It is recommended that you leave the default selection. This relates to the way data is formatted and changing this option may cause problems. Click **Next**.



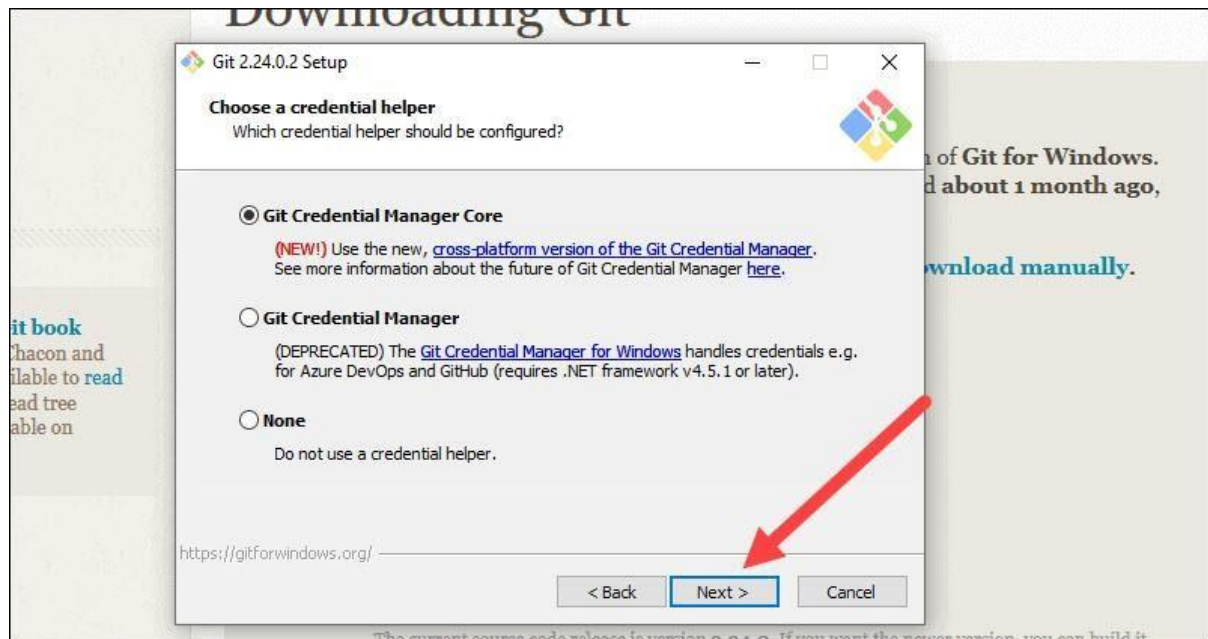
15. Choose the terminal emulator you want to use. The default MinTTY is recommended, for its features. Click **Next**.



16. 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.

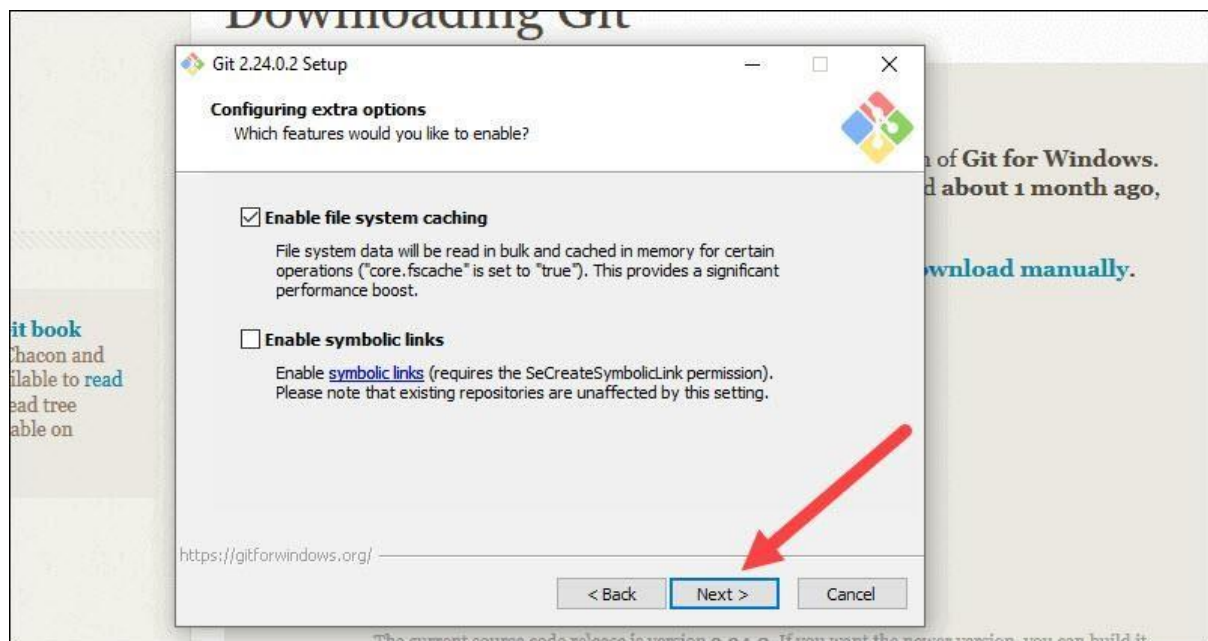


17. Next you should choose which credential helper to use. Git uses credential helpers to fetch or save credentials. Leave the default option as it is the most stable one, and click **Next**.

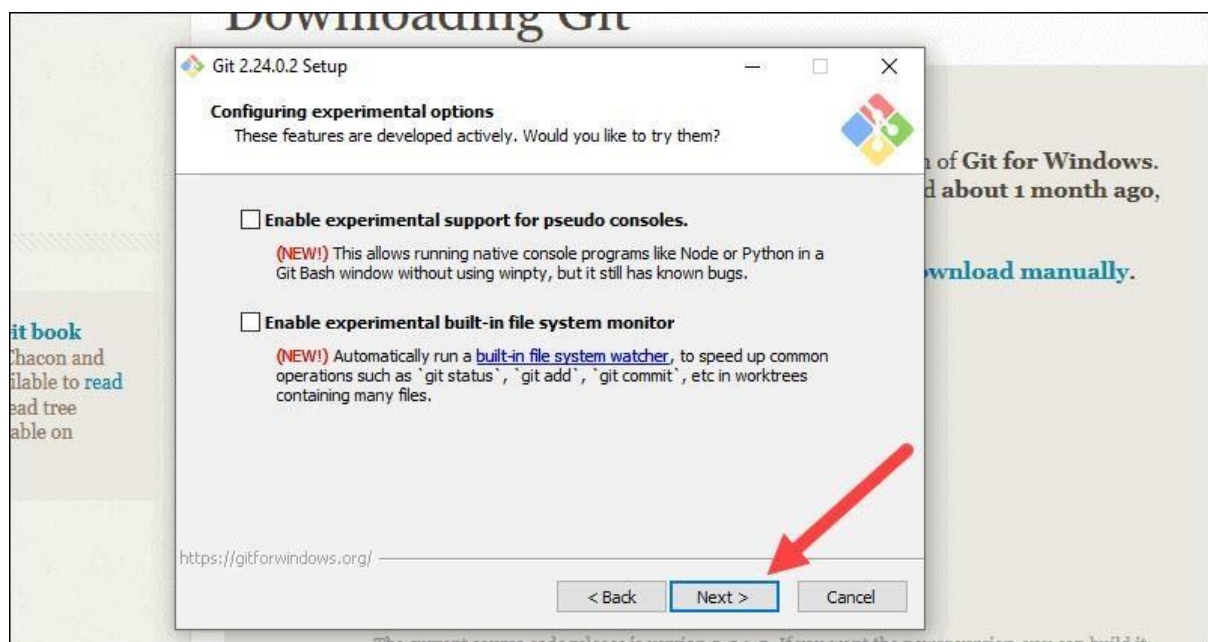


Additional Customization Options

18. 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**.

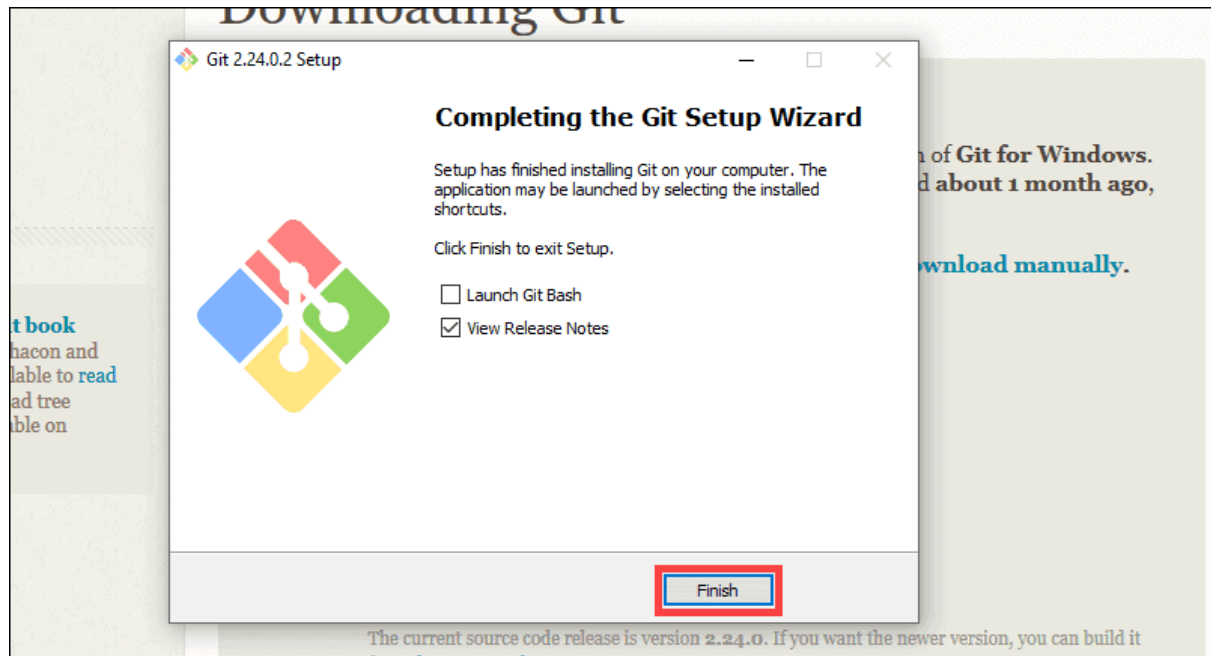


19. Depending on the version of Git you're installing, it may offer to install experimental features. At the time this article was written, the options to include support for pseudo controls and a built-in file system monitor were offered. Unless you are feeling adventurous, leave them unchecked and click **Install**.



Complete Git Installation Process

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



Experiment-2 AIM:-Setting Up GitHub Account

Steps For Making an Account in Git Hub

1

Go to <https://github.com/join> in a web browser. You can use any web browser on your computer, phone, or tablet to join.

- Some ad blockers, including uBlock Origin, prevent GitHub's verification CAPTCHA puzzle from appearing. For best results, disable your web browser's ad blocker when signing up for GitHub.



2

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.^[1]

- Carefully review the Terms of Service at <https://help.github.com/en/articles/github-terms-of-service> and the Privacy Statement at <https://help.github.com/en/articles/github-privacystatement> before you continue. Continuing past the next step confirms that you agree to both documents.

3

Click the green **Create an account** button. It's below the form.



Welcome to GitHub!
Let's begin the adventure

Enter your email
✓ aakashduttrohill@gmail.com

Create a password
✓ akashgoku21

Enter a username
✓ aakashrohill0

Would you like to receive product updates and announcements via email?
Type "y" for yes or "n" for no
→ n

Continue

4 After that u will receive an otp on your mail enter the otp and submit then your github account will be created

You're almost done!
We sent a launch code to aakashduttrohill@gmail.com

→ Enter code

--	--	--	--	--	--	--	--

Experiment-3

AIM:- how to use Git Log The git log Command

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 useful for displaying the history of a repository.

Whereas the git status command is focused on the current working directory, git log allows you to see the history of your repository.



```

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git log
commit 0d3835a746b82a4dc7ca97bcfbdbd4e39b26a680 (HEAD -> master)
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Fri Nov 8 15:49:51 2019 +0530

    newfile2 Re-added

commit 56afce0ea387ab840819686ec9682bb07d72add6 (tag: -d, tag: --delete, tag: --d, tag: projectv1.1, origin/master, testing)
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Wed Oct 9 12:27:43 2019 +0530

    Added an empty newfile2

commit 0d5191fe05e4377abef613d2758ee0dbab7e8d95
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Sun Oct 6 17:37:09 2019 +0530

    added a new image to project

commit 828b9628a873091ee26ba53c0fcfc0f2a943c544 (tag: olderversion)
Author: ImDwivedi1 <52317024+ImDwivedi1@users.noreply.github.com>
Date:   Thu Oct 3 11:17:25 2019 +0530

    Update design2.css

commit 0a1a475d0b15ecec744567c910eb0d8731ae1af3 (test)
Author: ImDwivedi1 <52317024+ImDwivedi1@users.noreply.github.com>
Date:   Tue Oct 1 12:30:40 2019 +0530

    CSS file

    See the proposed CSS file.

commit f1ddc7c9e765bd688e2c5503b2c88cb1dc835891
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Sat Sep 28 12:31:30 2019 +0530
  
```

Experiment-4

AIM:- Create and Visualize branch

How it works

A branch represents an independent line of development. Branches serve as an abstraction for the edit/stage/commit process. You can think of them as a way to request a brand new working directory, staging area, and project history. New commits are recorded in the history for the current branch, which results in a fork in the history of the project.

The `git branch` command lets you create, list, rename, and delete branches. It doesn't let you switch between branches or put a forked history back together again. For this reason, `git branch` is tightly integrated with the [git checkout](#) and [git merge](#) commands.

Common Options

```
git branch
```

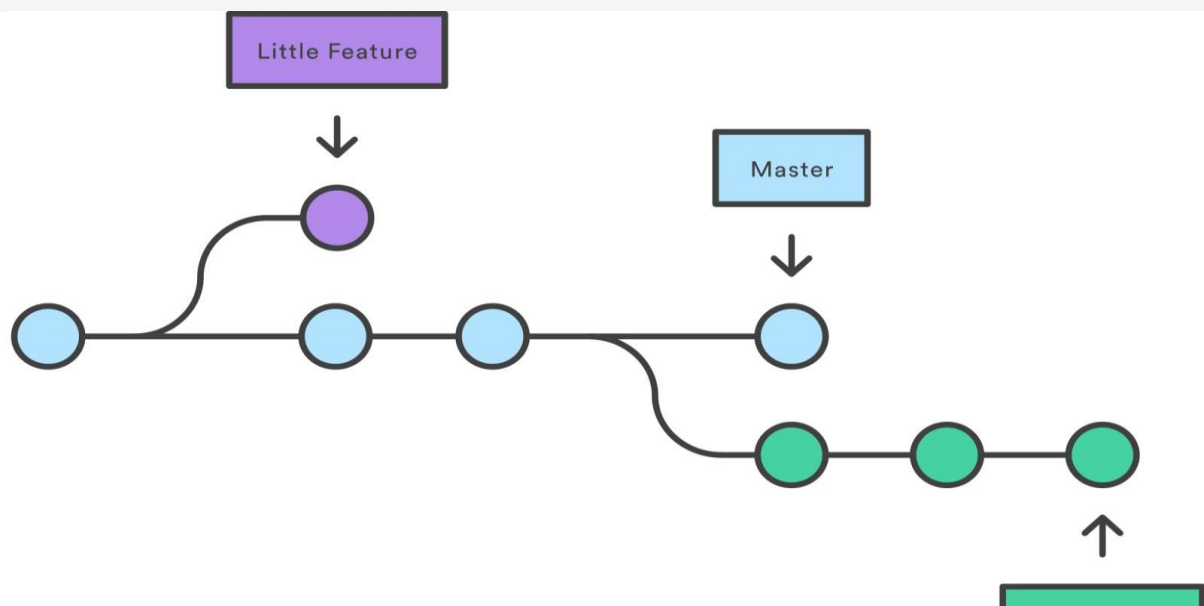
List all of the branches in your repository. This is synonymous with `git branch --list`.

```
git branch <branch>
```

Create a new branch called `<branch>`. This does *not* check out the new branch.

Creating Branches

It's important to understand that branches are just pointers to commits. When you create a branch, all Git needs to do is create a new pointer, it doesn't change the repository in any other way. If you start with a repository that looks like this:

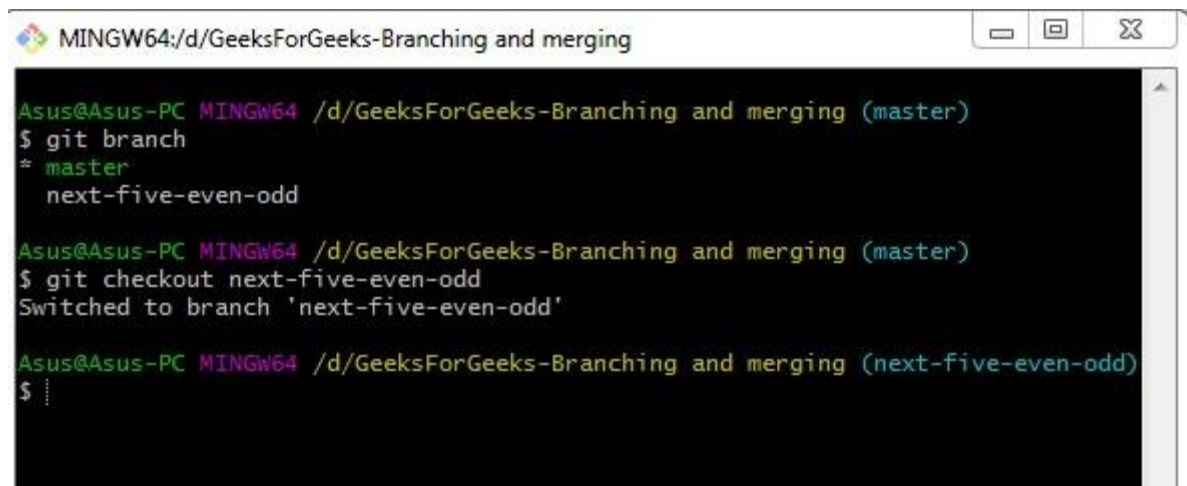


Then, you create a branch using the following command:

```
git branch crazy-experiment
```

The repository history remains unchanged. All you get is a new pointer to the current commit:

Note that this only *creates* the new branch. To start adding commits to it, you need to select it with `git checkout`, and then use the standard `git add` and `git commit` commands.



```

MINGW64:/d/GeeksForGeeks-Branching and merging
Asus@Asus-PC MINGW64 /d/GeeksForGeeks-Branching and merging (master)
$ git branch
* master
  next-five-even-odd

Asus@Asus-PC MINGW64 /d/GeeksForGeeks-Branching and merging (master)
$ git checkout next-five-even-odd
Switched to branch 'next-five-even-odd'

Asus@Asus-PC MINGW64 /d/GeeksForGeeks-Branching and merging (next-five-even-odd)
$
  
```

Experiment-5

AIM:- Git Life Cycle Description

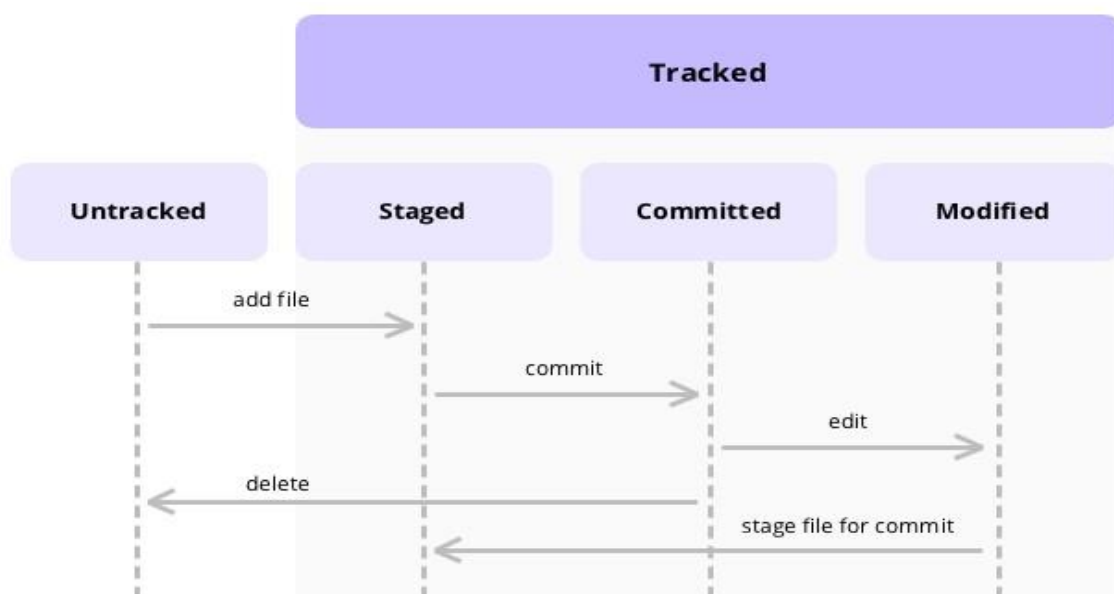
Introduction to Git Life Cycle

Git is one of the premier distributed version control systems available for programmers and corporates. In this article, we will see details about how a project that is being tracked by git proceeds with workflow i.e Git Life Cycle. As the name suggests is regarding different stages involved after cloning the file from the repository. It covers the git central commands or main commands that are required for this particular version control system

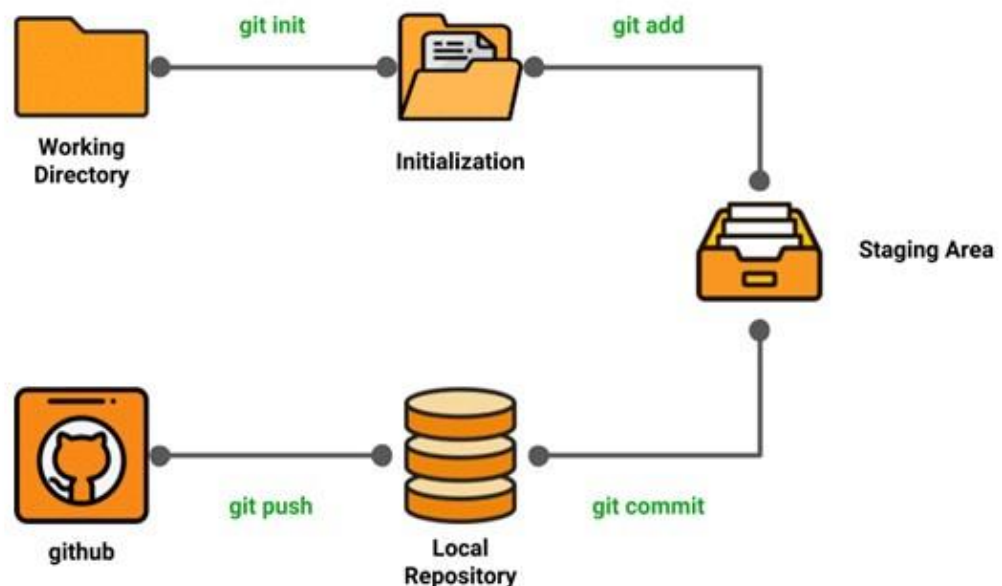
Workflow of Git Life Cycle

The workflow of the Git as follows:

- We will create a branch on which we can work on and later we will merge it with master
- Clone: First, when we have code present in the remote repository, we clone to local to form something called a local repository.
- Modifications/Adding Files: we perform several developments on the existing files or may as well add new files. Git will monitor all these activities and will log them.



- We need to move the content that we require to transform to the master to the staging area by using git commands and the snapshot of staged files will be saved in the git staging area.
- We need to perform commits on the files that are staged and the recorded snapshot from the above steps will be permanently saved on the local repo and this particular is recorded by commit message for future referrals.



□

Once we commit the code is available on the local repo but to send it to the master repo we need to perform PUSH operation

- If someone else is working on the same branch then there will be a possibility that he might have added his changes to the master by push. So we need to perform PULL operation before the PUSH operation if multiple people are working on the same branch and this workflow as shown below.
- Once the target branch is updated we need to get all the required approvals so that merge operation with the master is allowed.

This is the basic workflow of git was lots of intermediate commands like git add, git status, git commit, git push origin, git rebase, git merge, git diff, etc will be used depending upon the requirement of the user.



Stages of Git Life Cycle

So we have seen the workflow of the git life cycle above. But we need to know that we have a project linked with git then that project can reside in there of the following areas. Below mentioned areas are ingredients to the recipe of Git and having an idea of them will help you a lot to track the files that you are working on.

□

The stages are as discussed:

Working Directory

- Staging Area
- Git Directory

These Three Stages are explained below:

1. Working Directory

- If you have your project residing on to your local machines then basically it is called even though it is linked to git or not. In either case, it will be called as the working directory. But when the available project is linked with git then basically there will be .git folder hidden in the so-called working directory. So the presence of the .git folder is enough to say that the folder is working copy on the machine and it is tracked by the git.
- At this stage, git knows what are the files and folders that it's tracking that's it. No other info will be available regarding this. To make sure that the newly added files get tracked in the working copy we need to make sure that those files are staged and this is our second residence for the files.

2. Staging Area

- When we make changes to the existing files in the working repo or if we add any folder of files and if we want these changes to need to be tracked and also need to be moved to the local repo for tracking then we need to move these changed files or newly added folder of file to the staging area. Git add

□

is the basic command which will be used to move the modified files to the staged area.

It's ticked that been give to modified files or newly added folder of file to travel to the local repo for further traction. Those files that don't have that ticket will be tracked by the git but they won't be able to move to the target easily. Here index plays a critical role. GIT Index is something that comes in between local repo and working directory and it is the one that decides what needs to be sent to the local repo and in fact, it decides what needs to be sent to the central repo.

3. *GIT Directory*

- When we have done the modifications or addition of files or folder and want them to be part of the repository they first we do is to move them to the staging area and they will commit ready. When we commit then provide the appropriate commit message and files will be committed and get updated in the working directory.
- Now git tracks the commits and commit messages and preserves the snapshot of commit files and this is done in the Git specific directory called Git Directory. Information related to all the files that were committed and their commit messages will be stored in this directory. We can say that this git directory stores the metadata of the files that were committed.