



NAME: ANIMESH SHARMA
ROLL NO.: 2110990187

SCM FILE-1

Experiment No. 01

Aim: Setting up the git client.

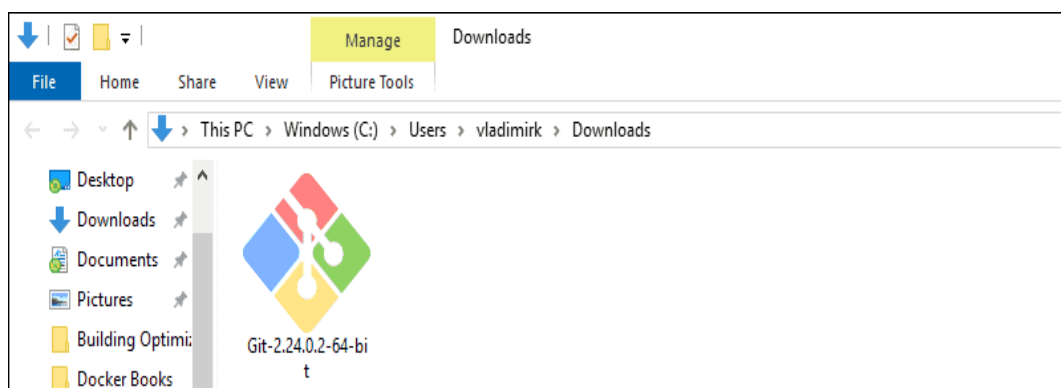
Download Git for Windows

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.

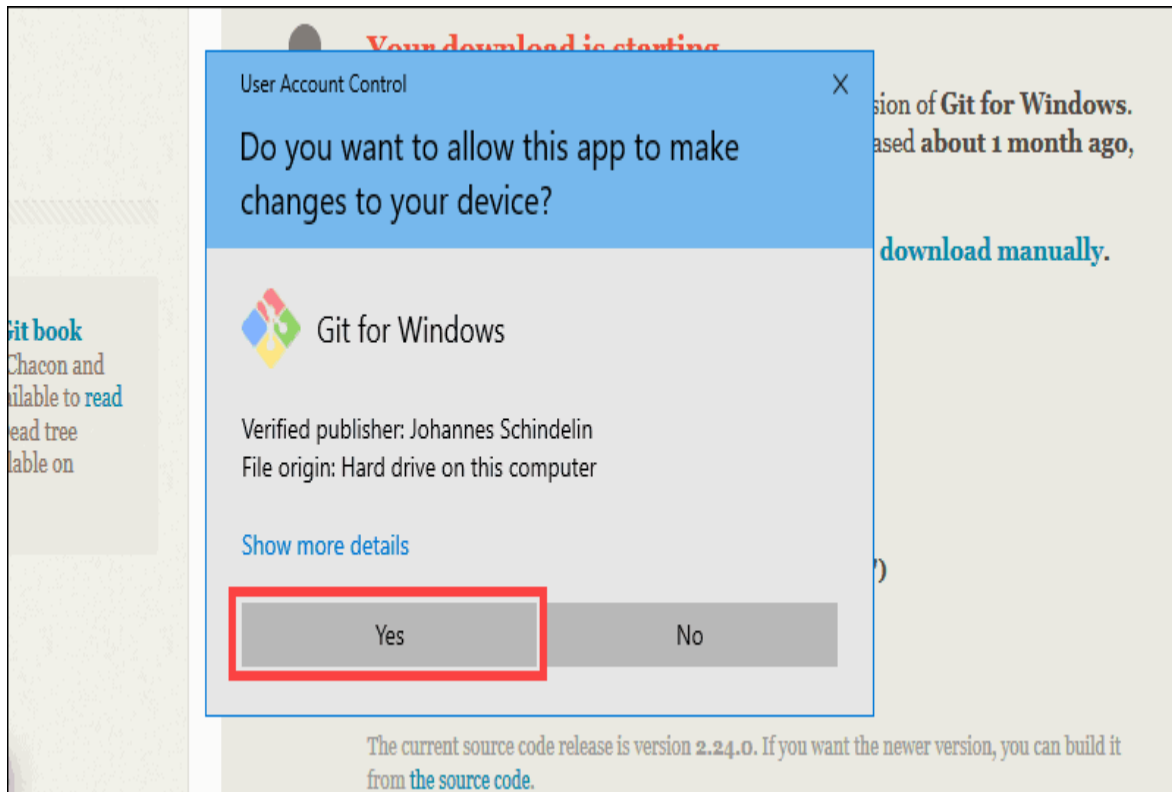


Extract and Launch Git Installer

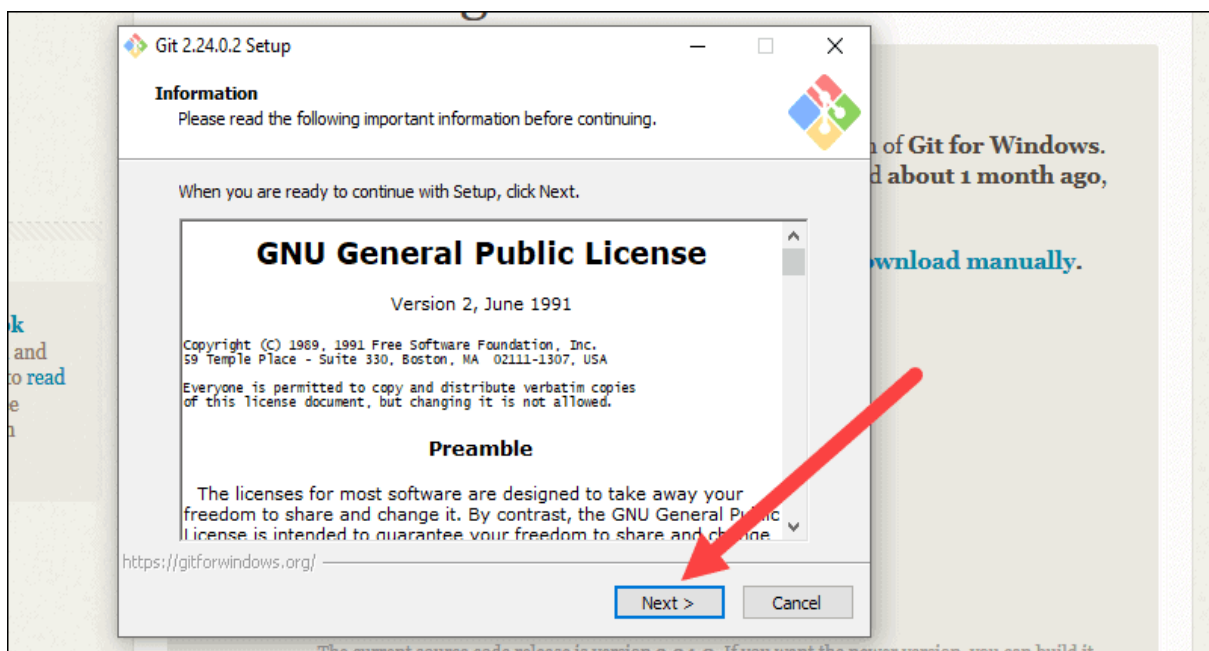
3. Browse to the download location (or use the download shortcut in your browser). Double-click the file to extract and launch the installer



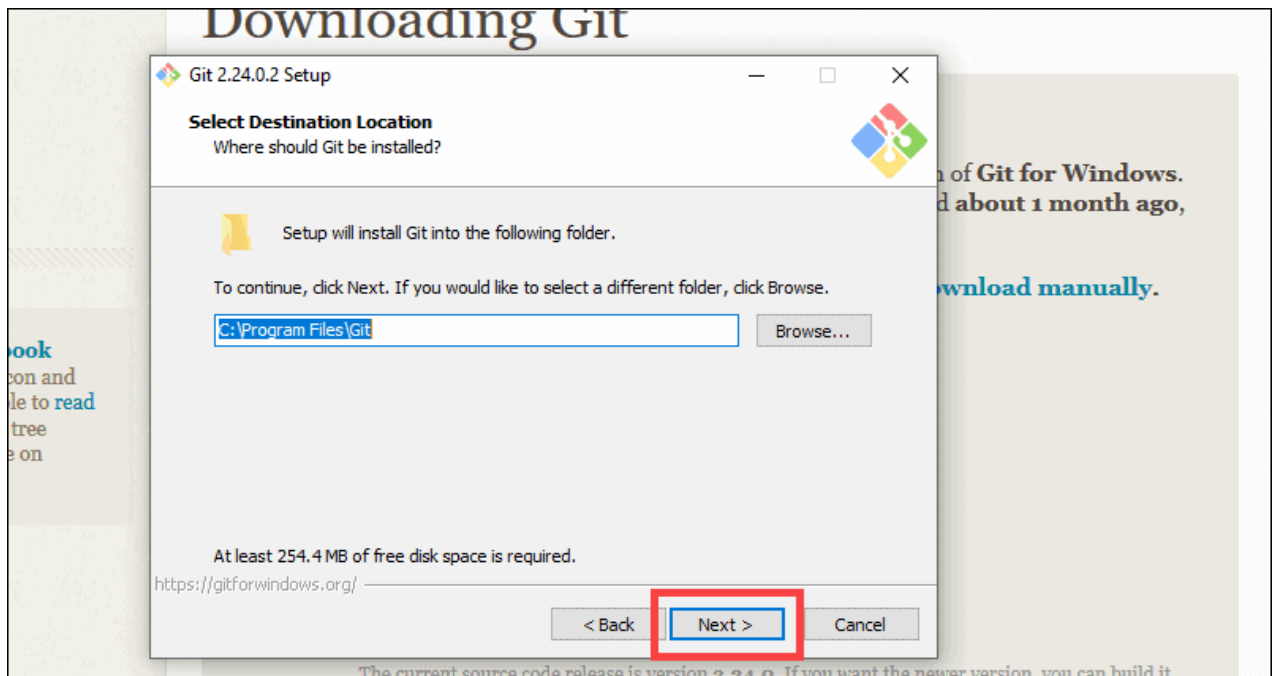
4. Allow the app to make changes to your device by clicking **Yes** on the User Account Control dialog that opens



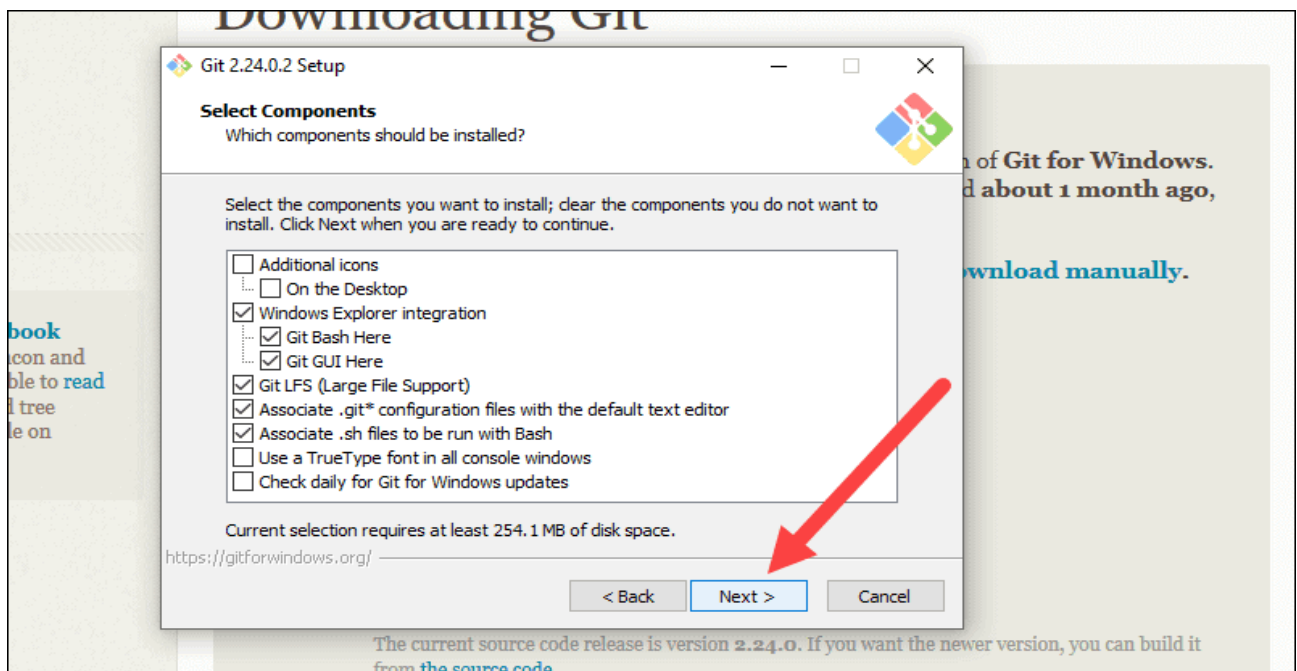
5. Review the GNU General Public License, and when you're ready to install, click **Next**.



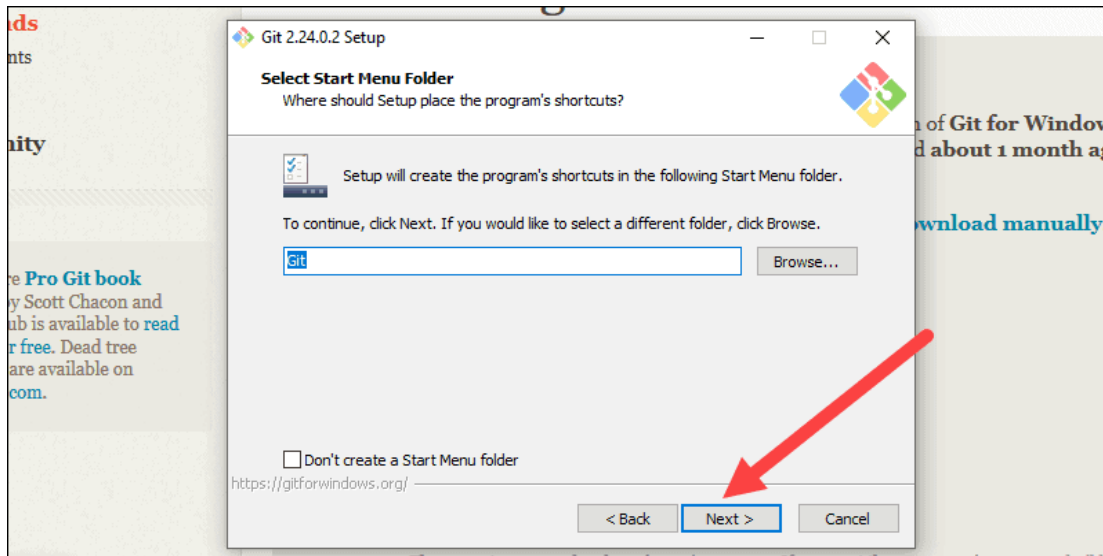
6. The installer will ask you for an installation location. Leave the default, unless you have reason to change it, and click **Next**.



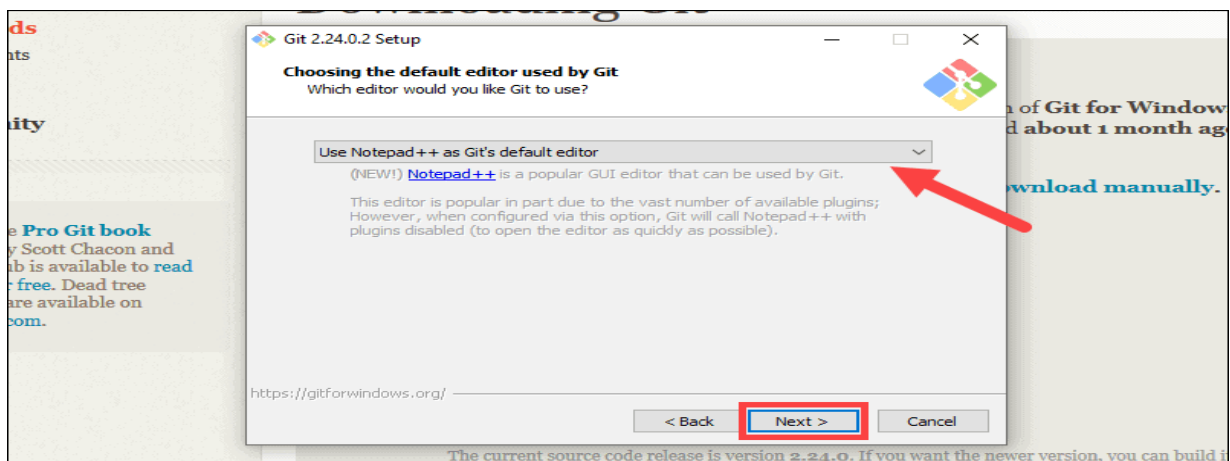
7. A component selection screen will appear. Leave the defaults unless you have a specific need to change them and click **Next**.



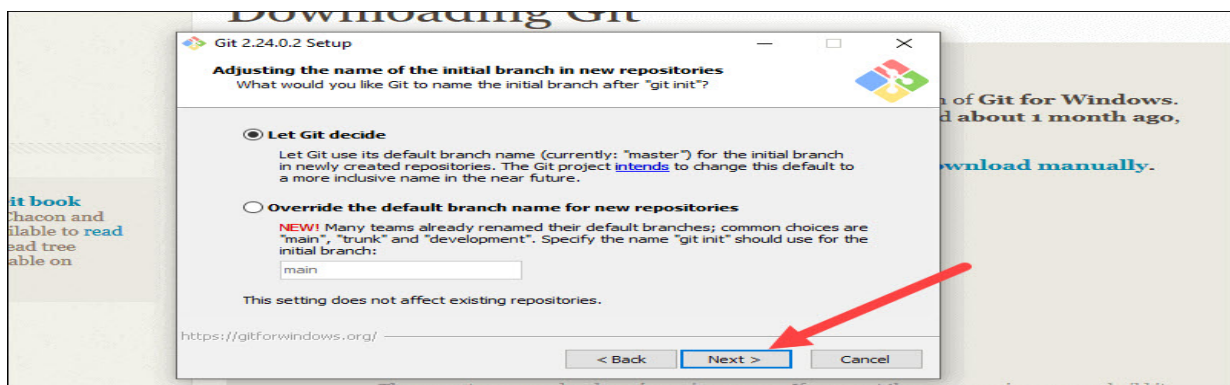
8. The installer will offer to create a start menu folder. Simply click **Next**



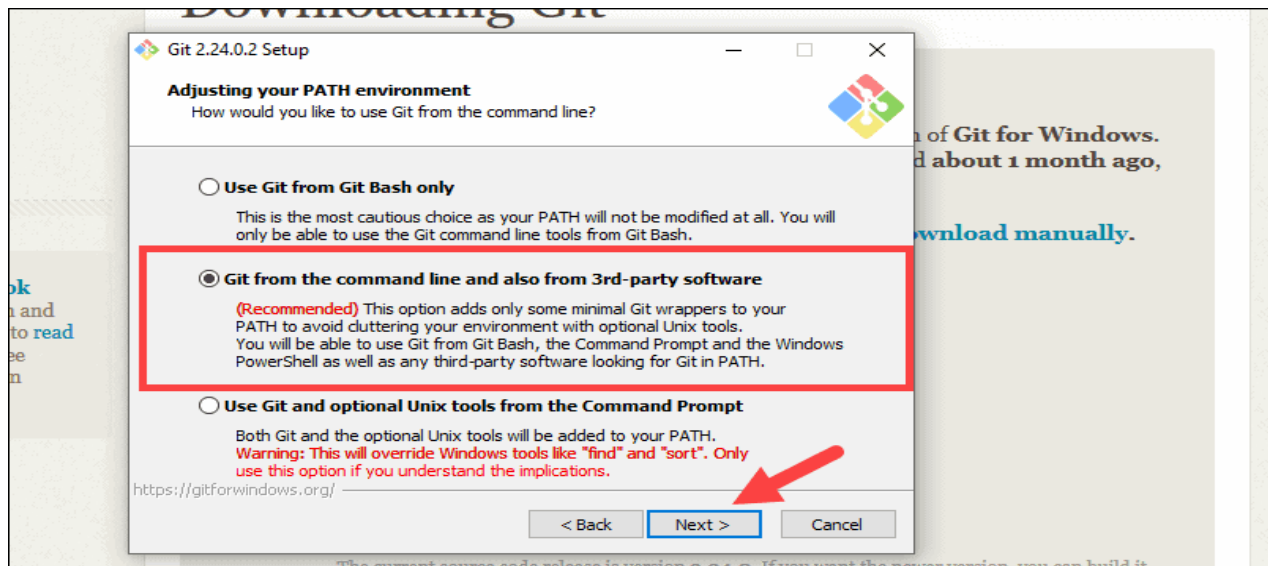
9. Select a text editor you'd like to use with Git. Use the drop-down menu to select Notepad++ (or whichever text editor you prefer) and click **Next**.



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**.

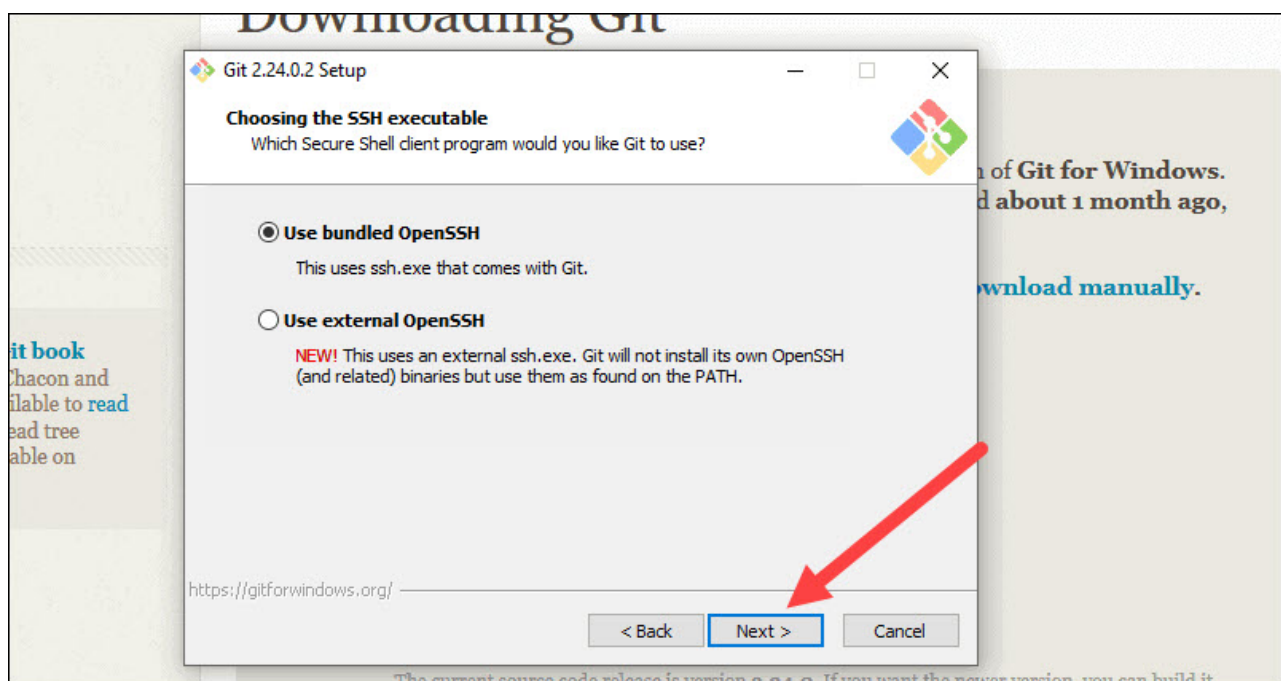


11. This installation step allows you to change the **PATH environment**. The **PATH** is the default set of directories included when you run a command from the command line. Leave this on the middle (recommended) selection and click **Next**.

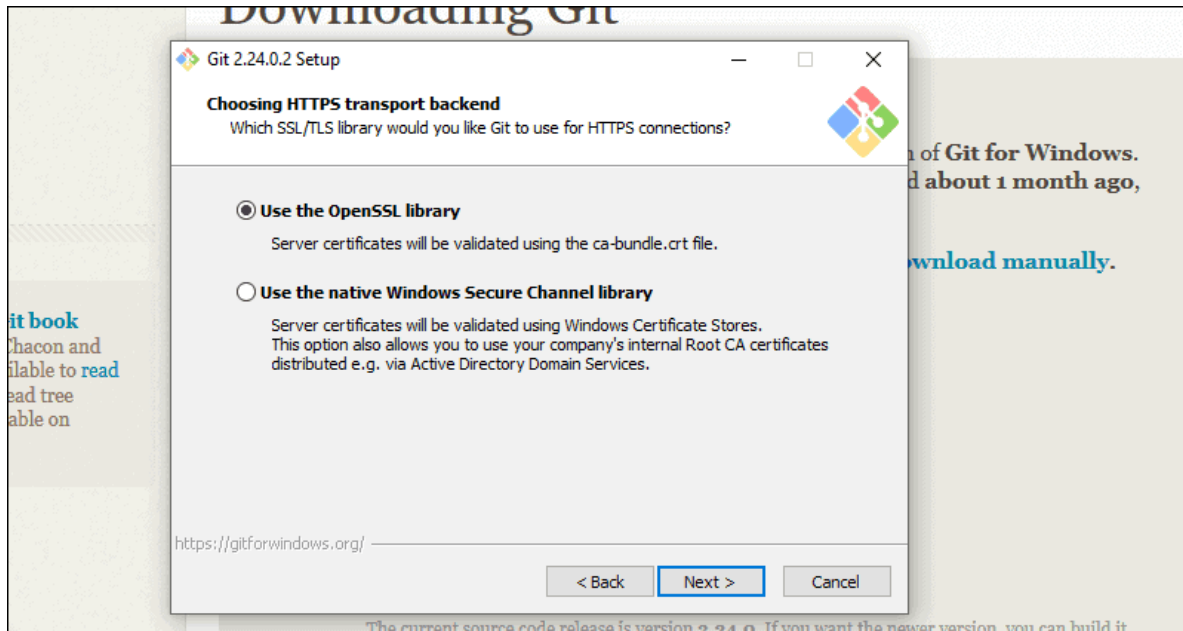


Server Certificates, Line Endings and Terminal Emulators

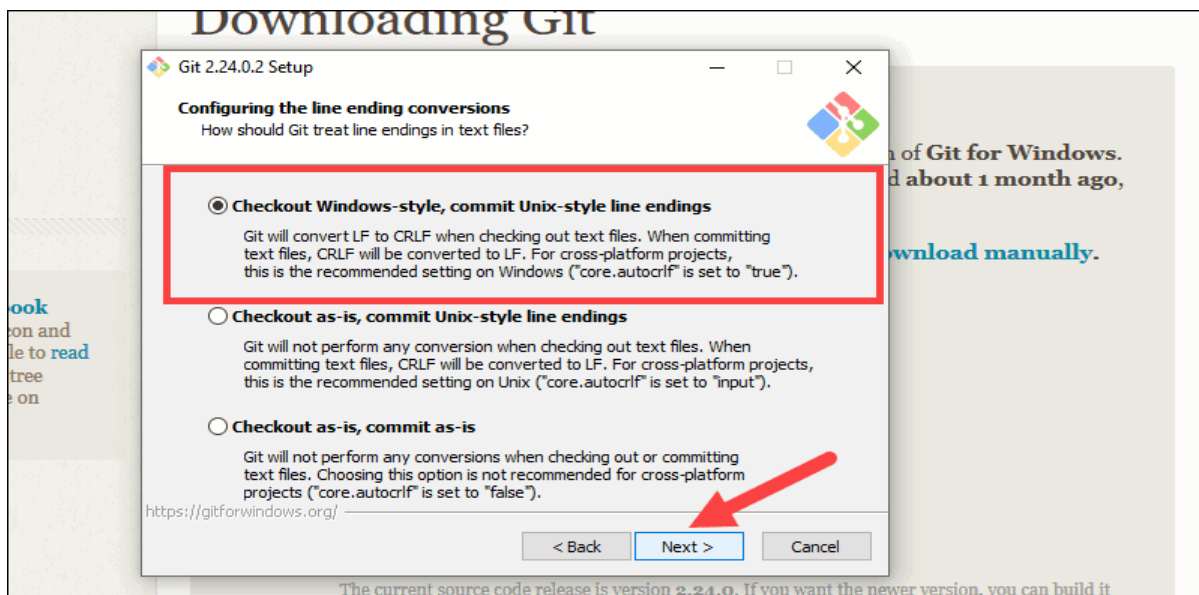
12. The installer now asks which SSH client you want Git to use. Git already comes with its own SSH client, so if you don't need a specific one, leave the default option and click **Next**.



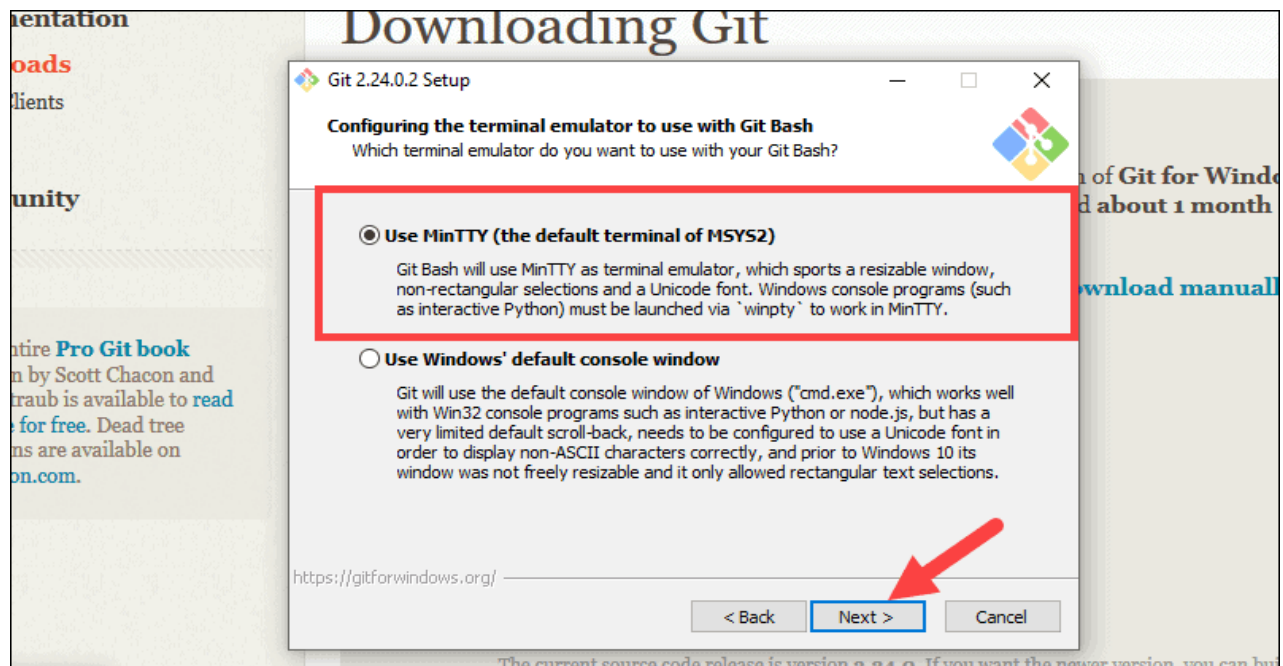
13. The next option relates to server certificates. Most users should use the default. If you're working in an Active Directory environment, you may need to switch to Windows Store certificates. Click **Next**.



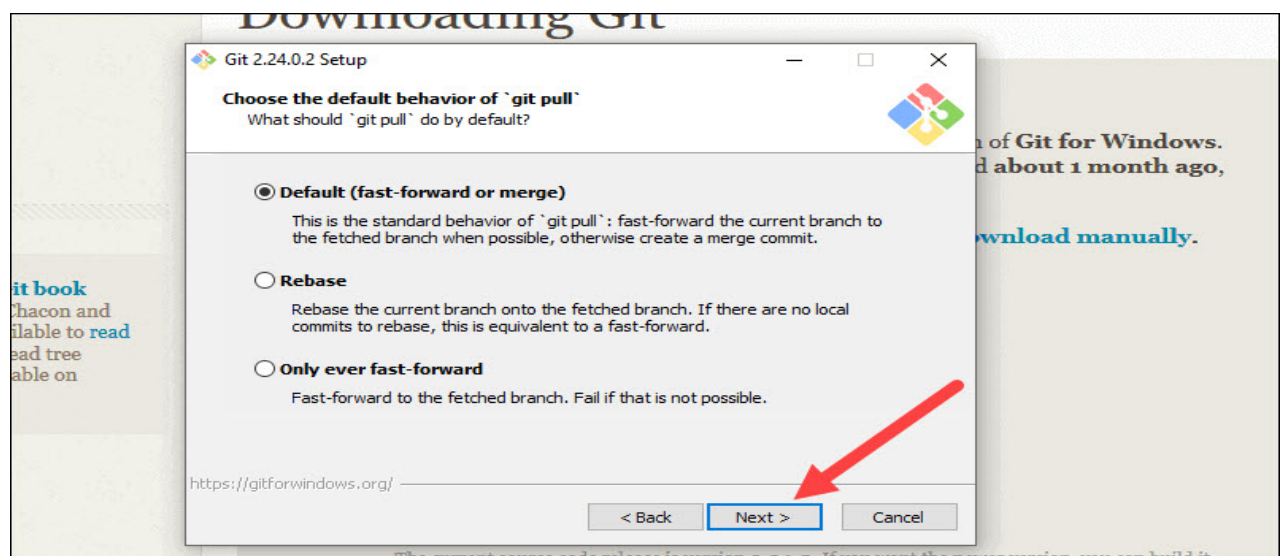
14. The next selection converts line endings. It is recommended that you leave the default selection. This relates to the way data is formatted and changing this option may cause problems. Click **Next**.



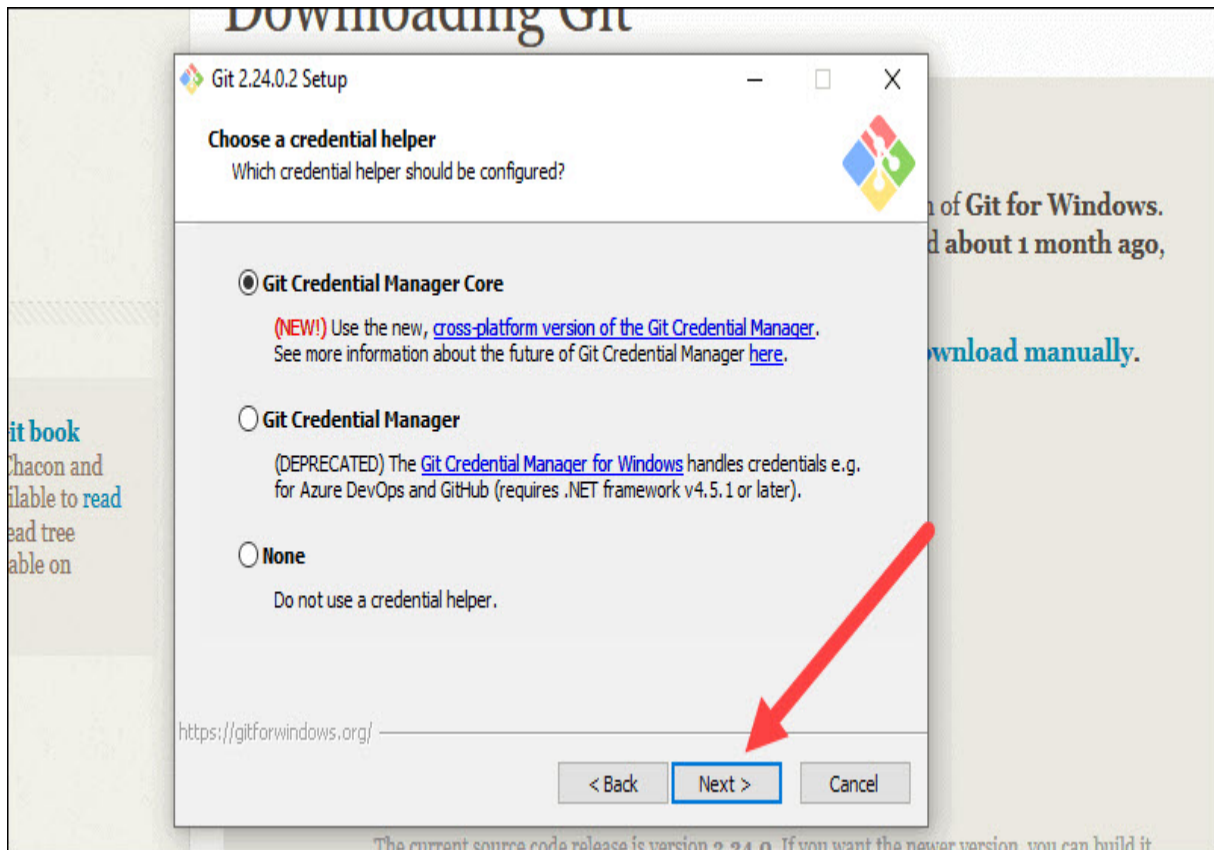
15. Choose the terminal emulator you want to use. The default MinTTY is recommended, for its features. Click **Next**.



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.

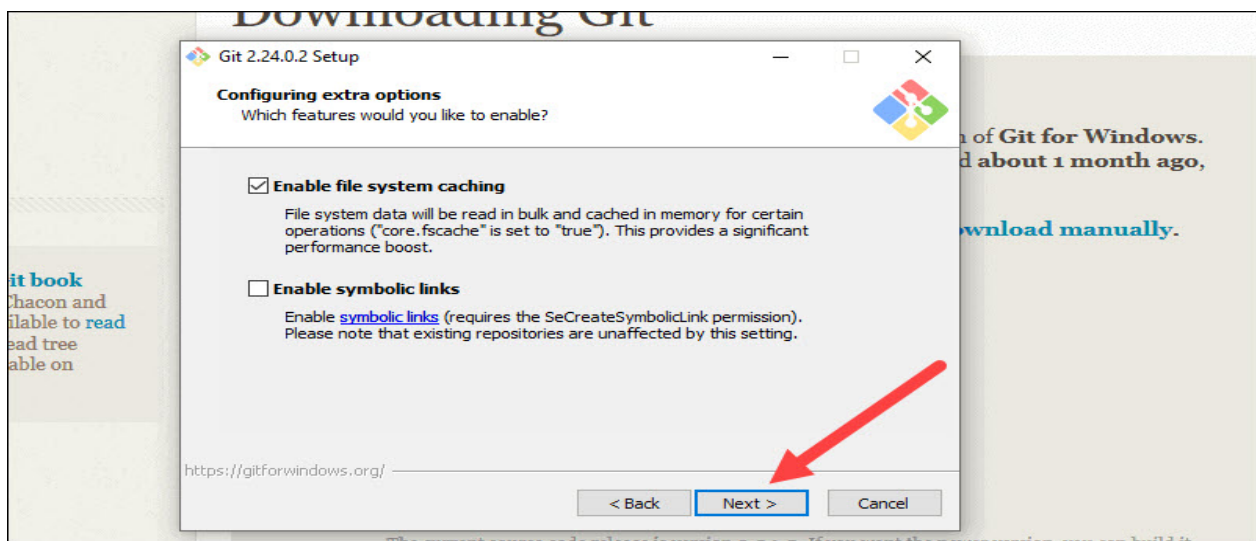


17. Next you should choose which credential helper to use. Git uses credential helpers to fetch or save credentials. Leave the default option as it is the most stable one, and click **Next**.

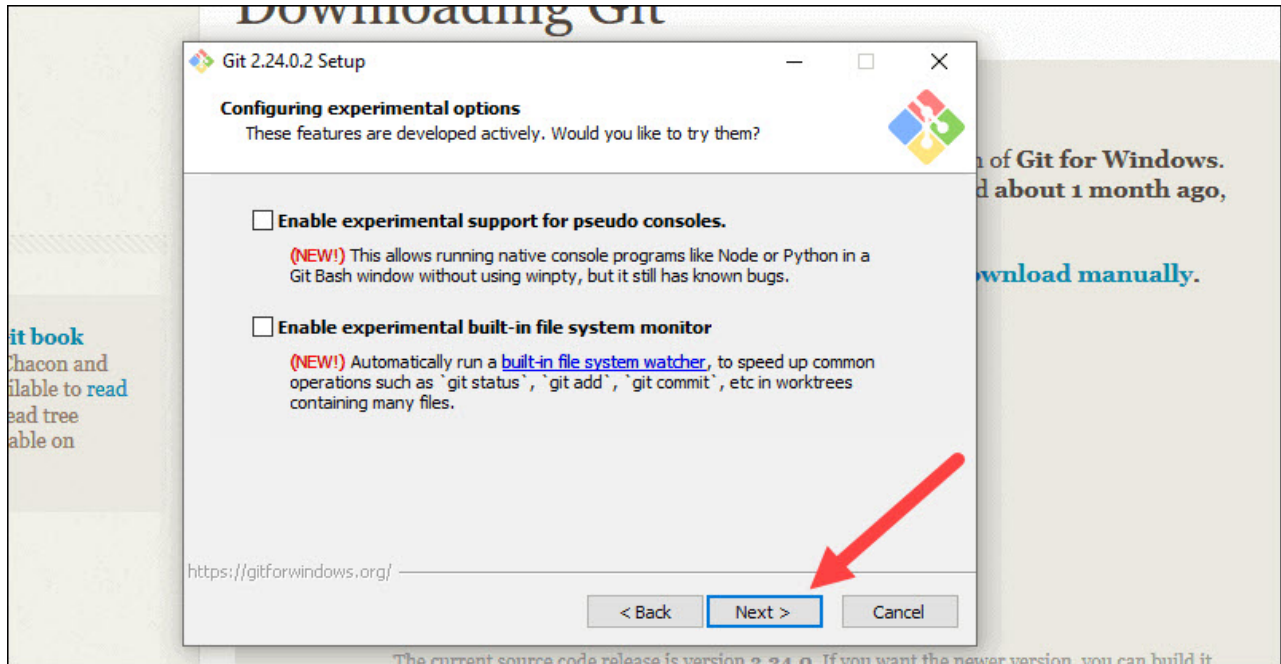


Additional Customization Options

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**

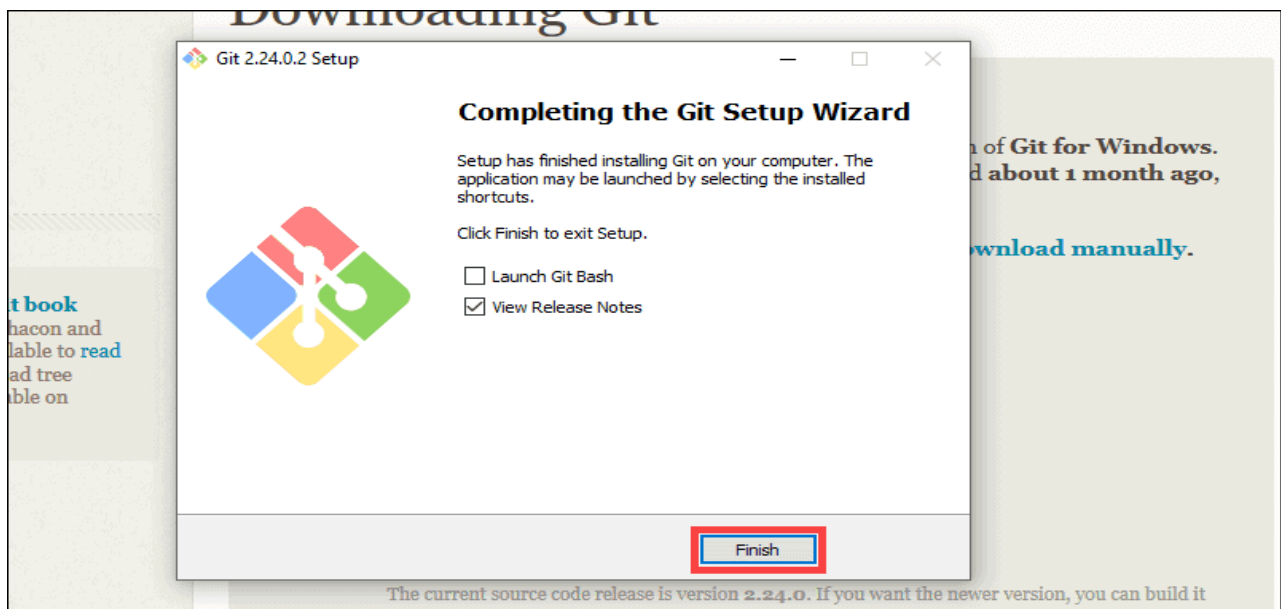


19. Depending on the version of Git you're installing, it may offer to install experimental features. At the time this article was written, the options to include support for pseudo controls and a built-in file system monitor were offered. Unless you are feeling adventurous, leave them unchecked and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



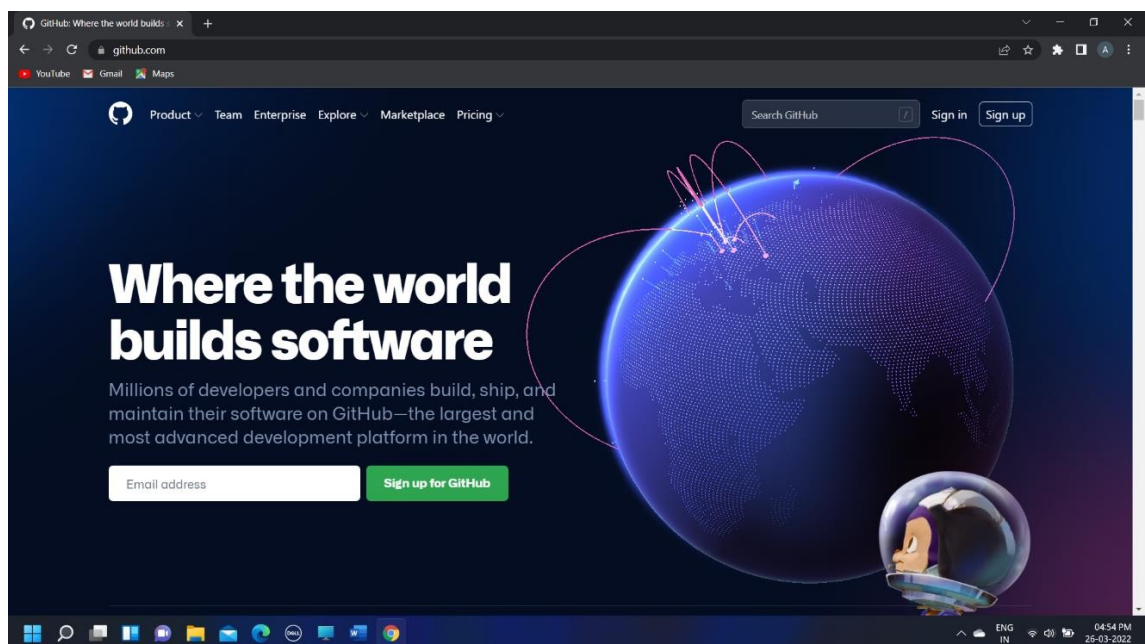
Experiment No. 02

Aim

Setting up GitHub Account

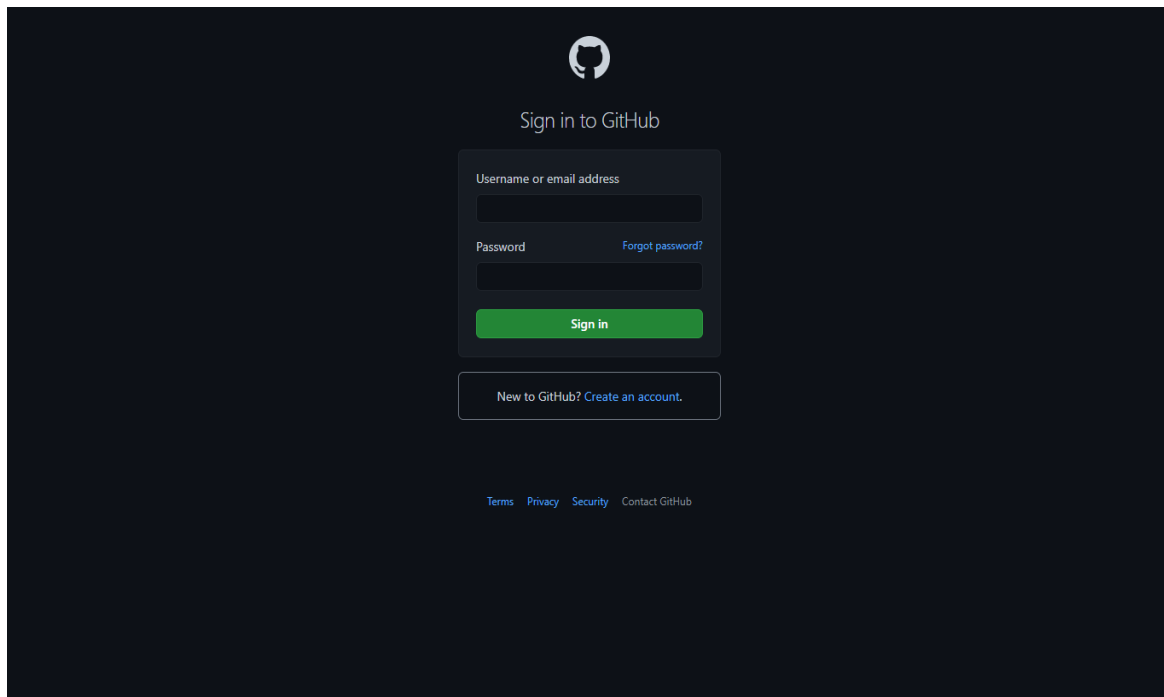
Creating your GitHub account

Go to github.com.

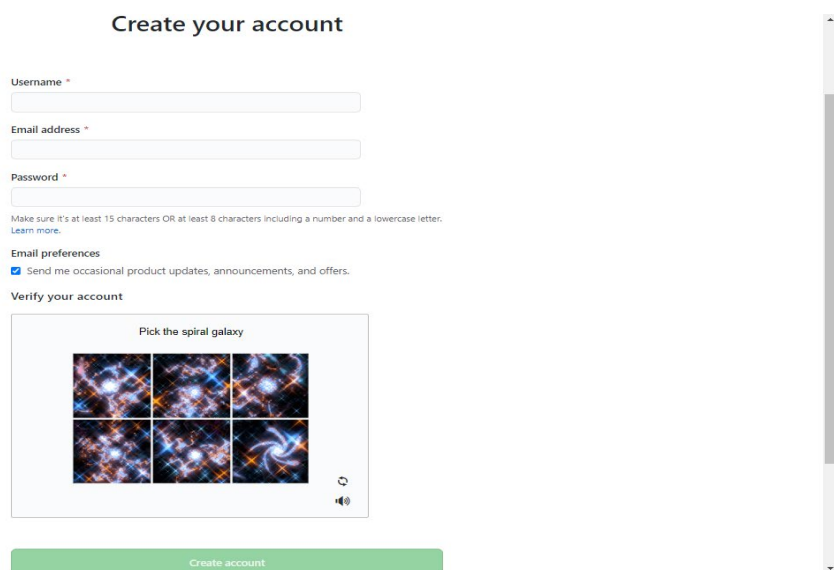


This is the landing page of GitHub. The globe that you see is interactive and it shows recent activities in some public repos. Click on it, move it around, and explore.

If you already have an account, click on the “Sign in” on the top right corner and enter your credentials on the page that opens up, like the one below.

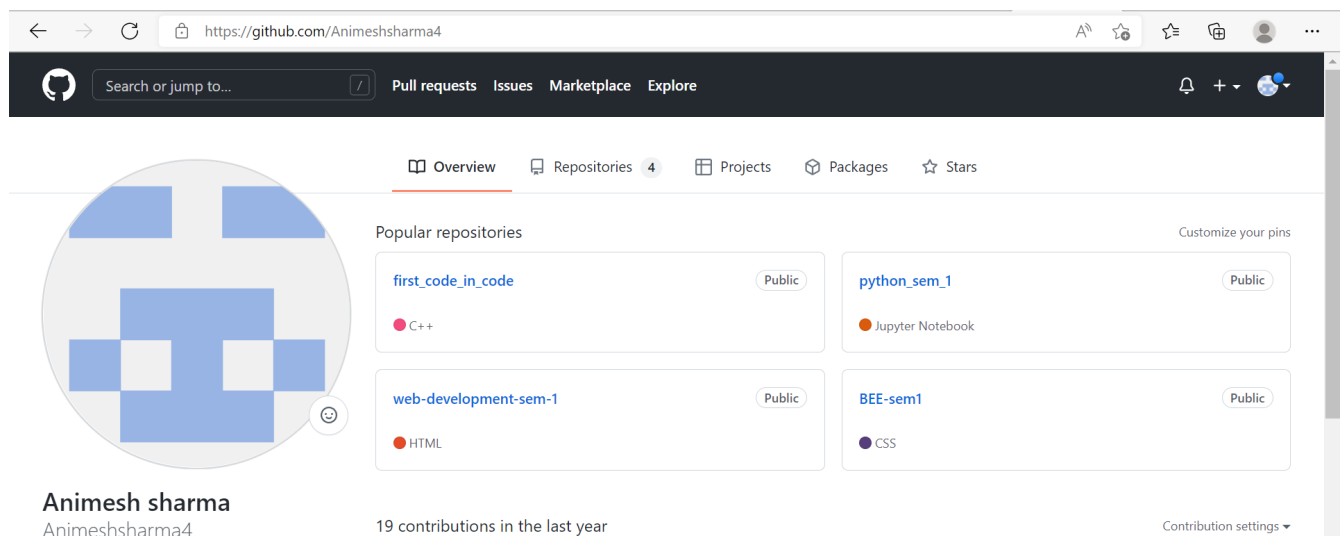


If you don't already have an account, click on the “Sign up” on the landing page or “Create an account.” on the login page.

The image shows the "Create your account" form. At the top, the title "Create your account" is centered. Below it are three input fields: "Username *", "Email address *", and "Password *". Below the password field is a note: "Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter." with a link "Learn more." Below this is the "Email preferences" section, which has a checked checkbox and the text "Send me occasional product updates, announcements, and offers." Below that is the "Verify your account" section, which contains a box titled "Pick the spiral galaxy" with a 2x3 grid of six different galaxy images. To the right of the grid is a speaker icon. At the bottom of the form is a green button labeled "Create account".

Fill in your details in the above form when you reach the Sign-up page. Choose a username that you like. This username will be your representative and will appear in all public places in GitHub. So choose creatively. To verify the account, do whatever the puzzle requires. Then click on the “Create account”. That's it. You will then be taken to your GitHub profile.

GitHub Dashboard



This is how the GitHub account looks like. Since I have already been working for some time, I have a few repos created.

You can create as many repos as you want and also choose to keep them private or public. A public repo can be viewed by anyone on GitHub and cloned.

Experiment No. 03



Aim: Program to generate logs

Basic Git init

Git init command creates a new Git repository. It can be used to convert an existing, undersigned project to a Git repository or initialize a new, empty repository. Most other Git commands are not available outside of an initialize repository, so this is usually the first command you'll run in a new project.

Basic Git status

The git status command displays the state of the working directory and the staging area. It lets you see which changes have been staged, which haven't, and which files aren't being tracked by Git. Status output does not show you any information regarding the committed project history.

Basic 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

Basic Git add command

The git add command adds a change in the working directory to the staging area. It tells Git that you want to include updates to a particular file in the next commit. However, git add doesn't really affect the repository in any significant way—changes are not actually recorded until you run git commit

Basic Git log

Git log command is one of the most usual commands of git. It is the most useful command for Git. Every time you need to check the history, you have to use the git log command. The basic git log command will display the most recent commits and the status of the head. It will use as:



Experiment No. 04



Aim: Create and visualize branches in Git

Pushing Local Files to the Remote Repository

Once you've done some work on the project, you may want to submit those changes to the remote project on GitHub.

1. For example, create a new text file by entering the following into your PowerShell window:
2. Confirmation that the new file is created

```
PS C:\Users\CCBill\git_test> new-item text.txt

Directory: C:\Users\CCBill\git_test

Mode                LastWriteTime         Length Name
----                -
-a-----         12/10/2019 12:58 PM              0 text.txt

PS C:\Users\CCBill\git_test> █
```

3. Now check the status of your [new Git branch](#) and untracked files:
4. Add your new file to the local project:
5. Run **git status** again to make sure the text.txt file has been added. Next, commit the changes to the local project:
6. Finally, push the changes to the remote GitHub repository:

Experiment No. 05

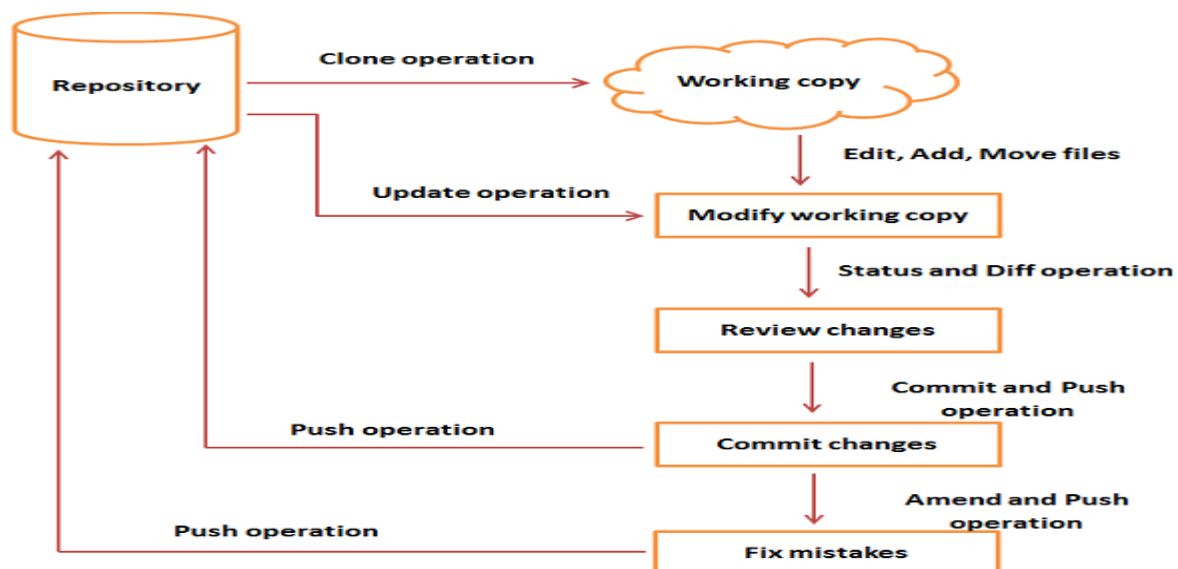
Aim: Git lifecycle description

Git is used in our day-to-day work, we use Git for keeping a track of our files, working in a collaboration with our team, to go back to our previous code versions if we face some error. Git helps us in many ways. Let us look at the Lifecycle description that git has and understand more about its life cycle. Let us see some of the basic steps that we have to follow while working with Git-

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.

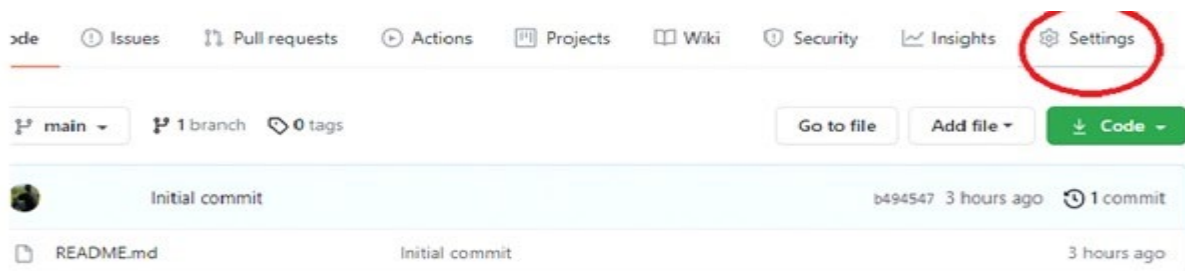
Shown below is the pictorial representation of the work-flow.



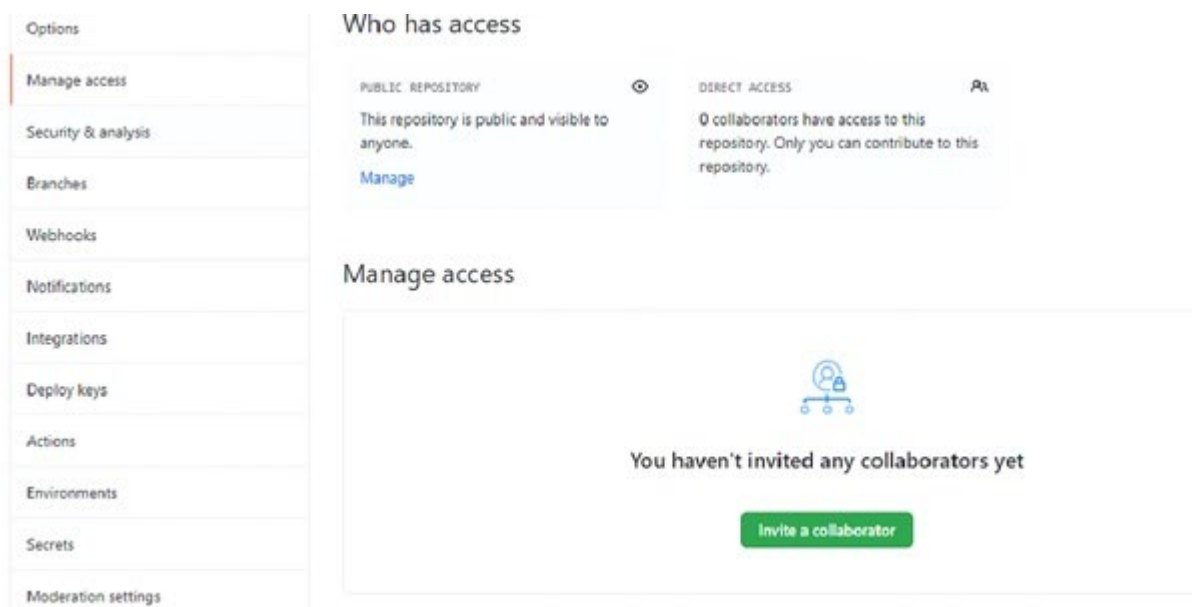
Experiment No.06

Aim: Add collaborators on GitHub Repo

Step 1 – Click on the **Settings** tab in the right corner of the GitHub page.



Step 2 – Go to **Manage Access** option under the Settings tab. On the Manage Access page, you will see an **Invite collaborator** link as shown in the below diagram



Step 3 – You can Invite collaborators by any of the following options –

- Username
- Full name
- Email

After you send the invite, the collaborator receives an email invitation. The collaborator has to accept it in order to get permission to collaborate on the same project.

Experiment No. 07



Aim: Fork and Commit

A fork is a rough copy of a repository. Forking a repository allows you to freely test and debug with changes without affecting the original project. One of the excessive use of forking is to propose changes for bug fixing. To resolve an issue for a bug that you found, you can:

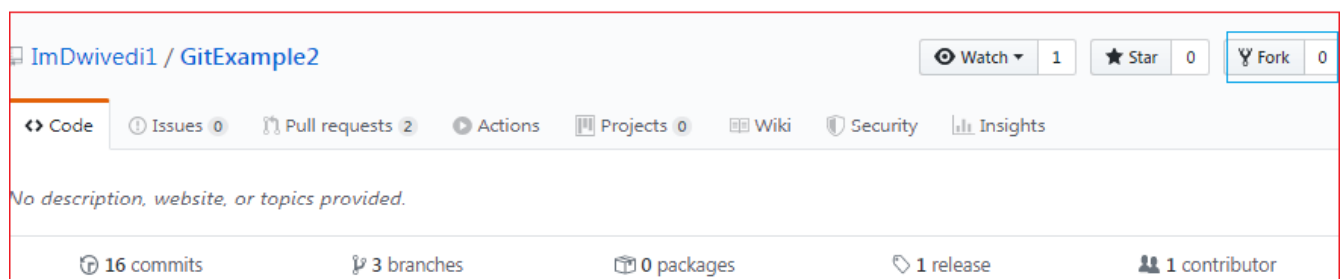
- Fork the repository.
- Make the fix.
- Forward a pull request to the project owner.

The forking and branching are excellent ways to contribute to an open-source project. These two features of Git allows the enhanced collaboration on the projects.

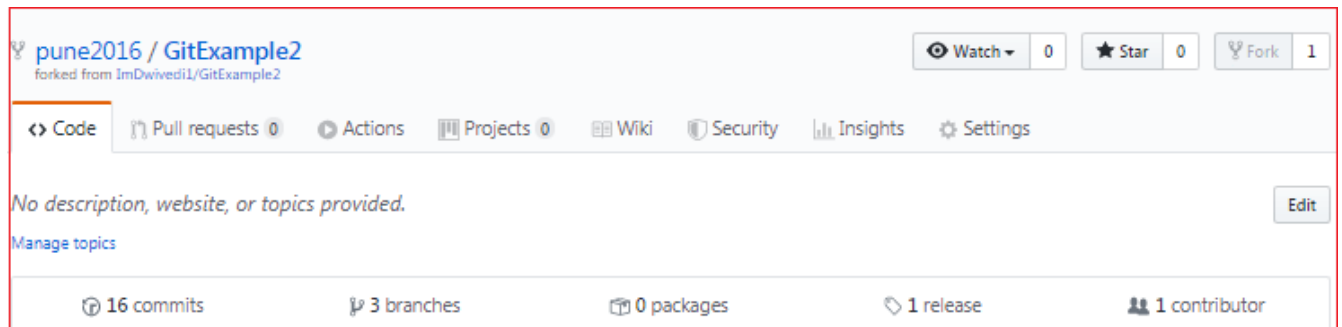
Forking is a safe way to contribute. It allows us to make a rough copy of the project. We can freely experiment on the project. After the final version of the project, we can create a pull request for merging.

It is a straight-forward process. Steps for forking the repository are as follows:

- Login to the GitHub account.
- Find the GitHub repository which you want to fork.
- Click the Fork button on the upper right side of the repository's page.



The above image shows the user interface of my repository from other contributors. We can see the fork option at the top right corner of the repository page. By clicking on that, the forking process will start. It will take a while to make a copy of the project for other users. After the forking completed, a copy of the repository will be copied to your GitHub account. It will not affect the original repository. We can freely make changes and then create a pull request for the main project. The owner of the project will see your suggestion and decide whether he wants to merge the changes or not. The fork copy will look like as follows:



As you can see, the forked repository looks like **pune2016/GitExample2**. At the bottom of the repository name, we can see a description of the repository. At the top right corner, the option fork is increased by 1 number.

Hence one can fork the repository from GitHub.

Experiment No. 08



Aim: Merge and Resolve conflicts created due to own activity and collaborators.

git checkout master

Before you run this command, make sure that you have pushed the latest code in your current branch (**feature_merge** in this example) otherwise Git won't let you checkout the **master** branch.

All this line of code does is to get the code from the **master** branch into your computer


```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: cmd

C:\Users\User\Google Drive\EasySoftwareDeveloper\git2\git_merge_example>git branch
* feature_merge
  master

C:\Users\User\Google Drive\EasySoftwareDeveloper\git2\git_merge_example>git add .

C:\Users\User\Google Drive\EasySoftwareDeveloper\git2\git_merge_example>git commit -m "Added new
[feature_merge 41aea1a] Added new subtitle
1 file changed, 1 insertion(+)

C:\Users\User\Google Drive\EasySoftwareDeveloper\git2\git_merge_example>git push origin feature_me
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 320 bytes | 320.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/gsauzande/git_merge_example.git
1ecbb23..41aea1a feature_merge -> feature_merge

C:\Users\User\Google Drive\EasySoftwareDeveloper\git2\git_merge_example>git checkout master

```

git pull

This command will make sure that you have the latest version of the repository in your computer, so all the new code in `master` and reference to any new branches that might have been created recently.

```

C:\Users\User\Google Drive\EasySoftwareDeveloper\git2\git_merge_example>git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

C:\Users\User\Google Drive\EasySoftwareDeveloper\git2\git_merge_example>git pull
Already up to date.

C:\Users\User\Google Drive\EasySoftwareDeveloper\git2\git_merge_example>

```

git checkout feature_merge

we can get back to the branch that we want to merge into master by running the command above.

```

C:\Users\User\Google Drive\EasySoftwareDeveloper\git2\git_merge_example>git pull
Already up to date.


C:\Users\User\Google Drive\EasySoftwareDeveloper\git2\git_merge_example>git checkout feature_merge
Switched to branch 'feature_merge'
Your branch is up to date with 'origin/feature_merge'.

C:\Users\User\Google Drive\EasySoftwareDeveloper\git2\git_merge_example>

```

Now our computer knows about the latest master version and the latest feature_merge version, which means that we can finally run the merge command

git merge master

This command tells git to merge whichever branch we are currently in (feature_merge) with master. As we can see, git tried to auto-merge but it wasn't able to do so which made it alert us of a merge conflict, which means that now we get to use the secret tool... , Visual Studio Code's conflict resolution tool.

If you have been using VSCode for this, you can probably see something like the picture below being displayed.

```

9   <body>
    Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
10  <<<<<< HEAD (Current Change)
11  <p>
12  |   Changes
13  </p>
14  <div>Yes, more changes</div>
15  =====
16  <h1>New master</h1>
17  >>>>>> master (Incoming Change)
18  </body>
19  </html>

```

You can now click on whichever option you prefer, the tool shows whether you'd like to accept the changes from master and ignore yours, keep yours and ignore master's or keep both.

I decided to keep both and it'll look like this after I pick Accept Both Changes .

```

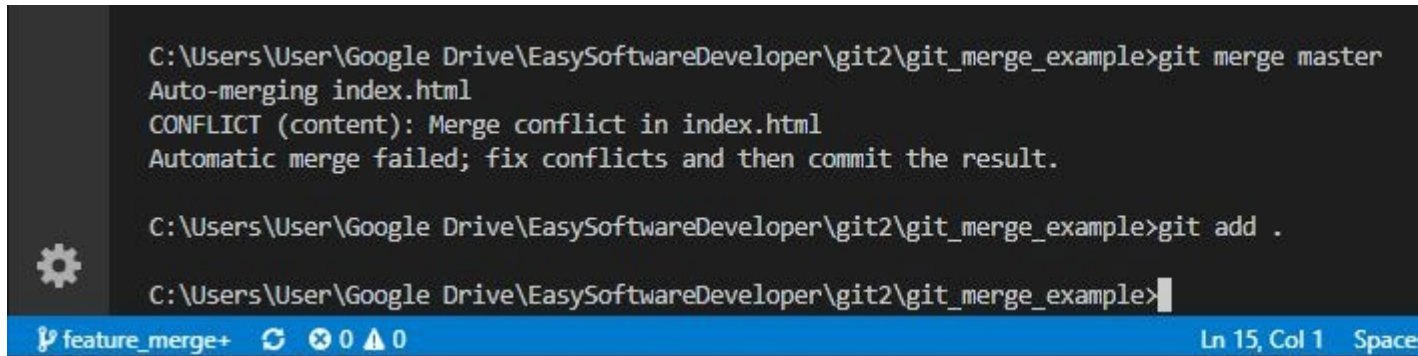
8   </head>
9   <body>
10  <p>
11  |   Changes
12  </p>
13  <div>Yes, more changes</div>
14  <h1>New master</h1>
15  </body>
16  </html>

```

git add .

After saving the file we have just fixed, it's time to glide over to a wonderful merge.

So now we run `git add .` to let git know which files we want to commit next.



```

C:\Users\User\Google Drive\EasySoftwareDeveloper\git2\git_merge_example>git merge master
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.

C:\Users\User\Google Drive\EasySoftwareDeveloper\git2\git_merge_example>git add .

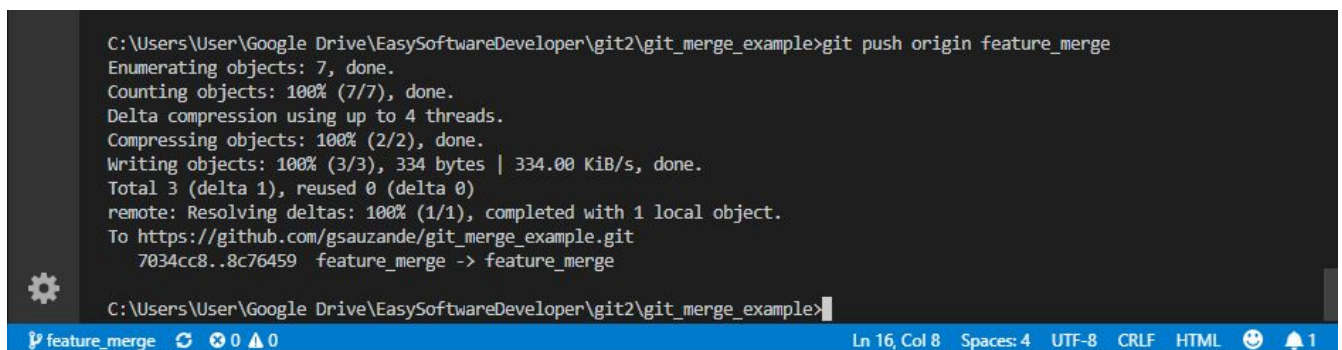
C:\Users\User\Google Drive\EasySoftwareDeveloper\git2\git_merge_example>
  
```

The screenshot shows a Windows command prompt window with a dark background. The title bar indicates the file 'feature_merge+'. The command prompt shows the execution of 'git merge master', which results in a conflict in 'index.html'. The message 'Automatic merge failed; fix conflicts and then commit the result.' is displayed. The user then runs 'git add .' to stage the changes. The status bar at the bottom shows 'Ln 15, Col 1' and 'Space'.

`git commit -m "My first smooth merge"`

This is really all for this step, we have done all the hard work already and as you can see, these steps make things much much better and easier to deal with. Only one last command to run and we are done!

`git push origin feature_merge`



```

C:\Users\User\Google Drive\EasySoftwareDeveloper\git2\git_merge_example>git push origin feature_merge
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 334 bytes | 334.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/gsauzande/git_merge_example.git
  7034cc8..8c76459  feature_merge -> feature_merge

C:\Users\User\Google Drive\EasySoftwareDeveloper\git2\git_merge_example>
  
```

The screenshot shows a Windows command prompt window with a dark background. The title bar indicates the file 'feature_merge'. The command prompt shows the execution of 'git push origin feature_merge'. The output indicates that the push was successful, with objects enumerated, counted, compressed, and written. The status bar at the bottom shows 'Ln 16, Col 8', 'Spaces: 4', 'UTF-8', 'CRLF', 'HTML', and a notification icon.

There we go, our code was safely and easily merged and pushed.



Aim: Reset and Revert

Git Reset

The `git reset` command allows you to RESET your current head to a specified state. You can reset the state of specific files as well as an entire branch. This is useful if you haven't pushed your commit up to GitHub or another remote repository yet.

Git Revert

Both the `git revert` and `git reset` commands undo previous commits. But if you've already pushed your commit to a remote repository, it is recommended that you do not use `git reset` since it rewrites the history of commits. This can make working on a repository with other developers and maintaining a consistent history of commits very difficult. Instead, it is better to use `git revert`, which undoes the changes made by a previous commit by creating an entirely new commit, all without altering the history of commits.

