Subject Name: **Source Code Management**

Subject Code: **CS181**

Cluster: **Beta**

Department: **DCSE**

Submitted By:
ANSHU KUMAR
2110990220 G08

Submitted To:
Dr.MONIT
KAPOOR

AIM :-Setting up of Git Client

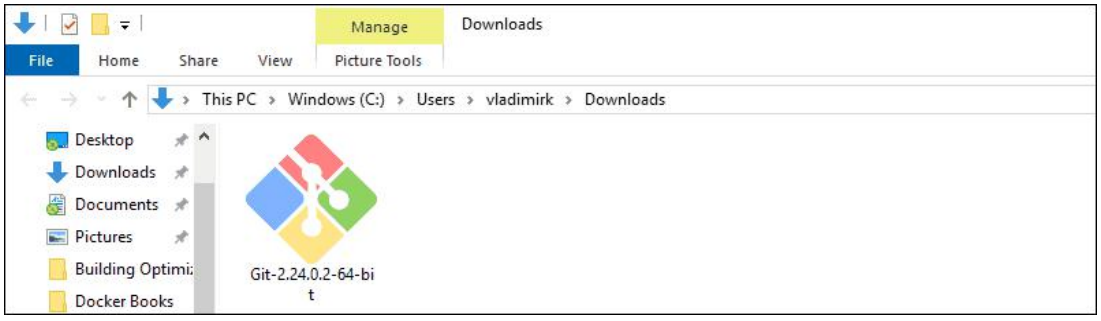
Steps For Installing Git for Windows

Download Git for Windows

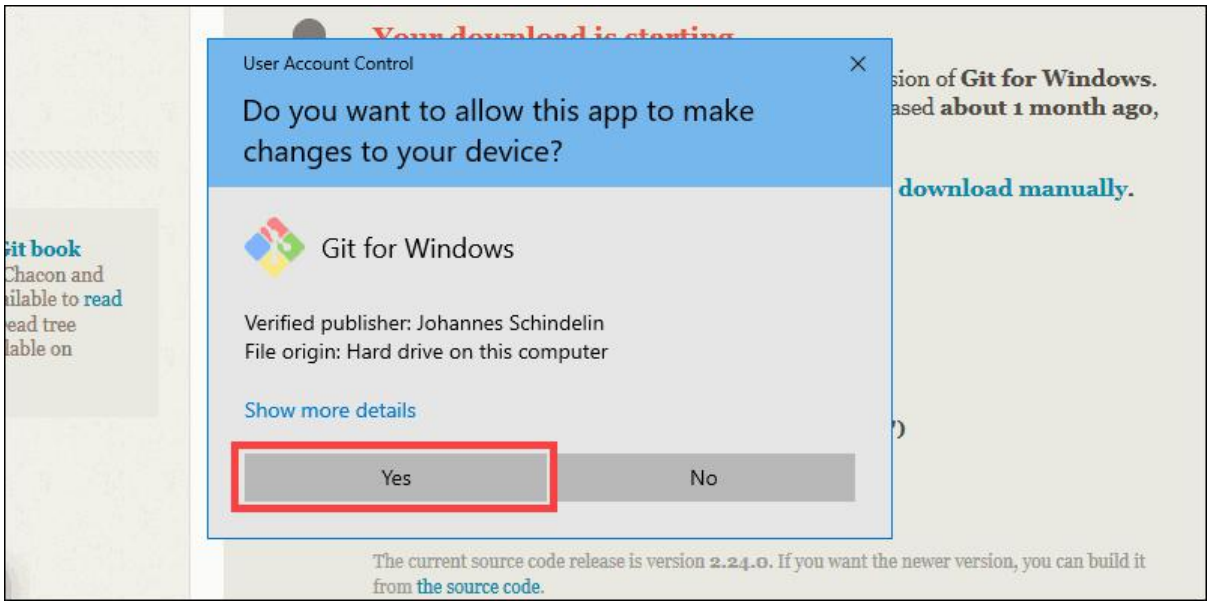
1. Browse to the official Git Website: https://git-scm.com/downloads
2. click the download link for Windows and allow the download to complete.

Extract and Launch Git Installer

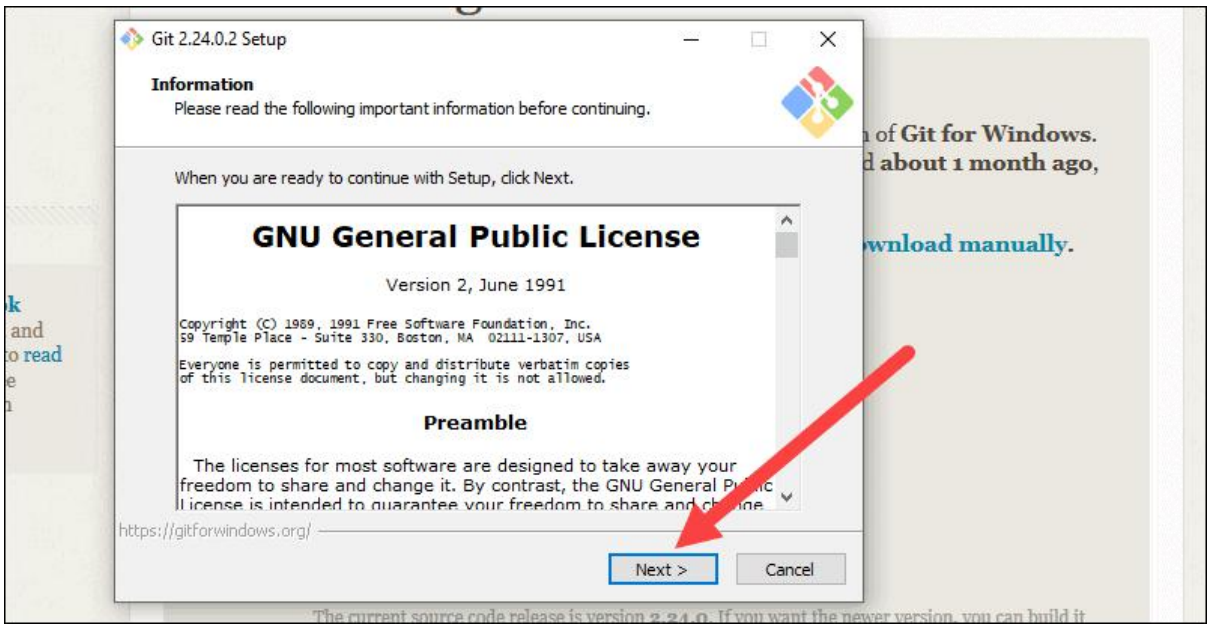
3. Browse to the download location (or use the download shortcut in your browser). Double-click the file to extract and launch the installer.



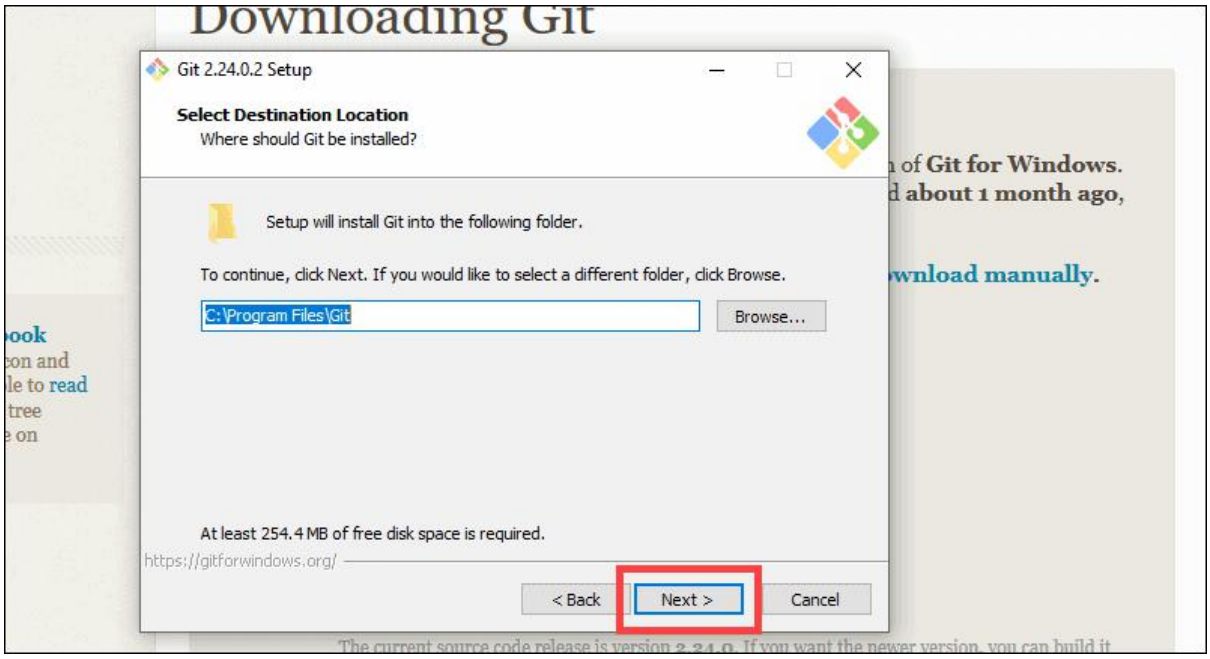
4.Allow the app to make changes to your device by clicking **Yes** on the User Account Control dialog that opens.



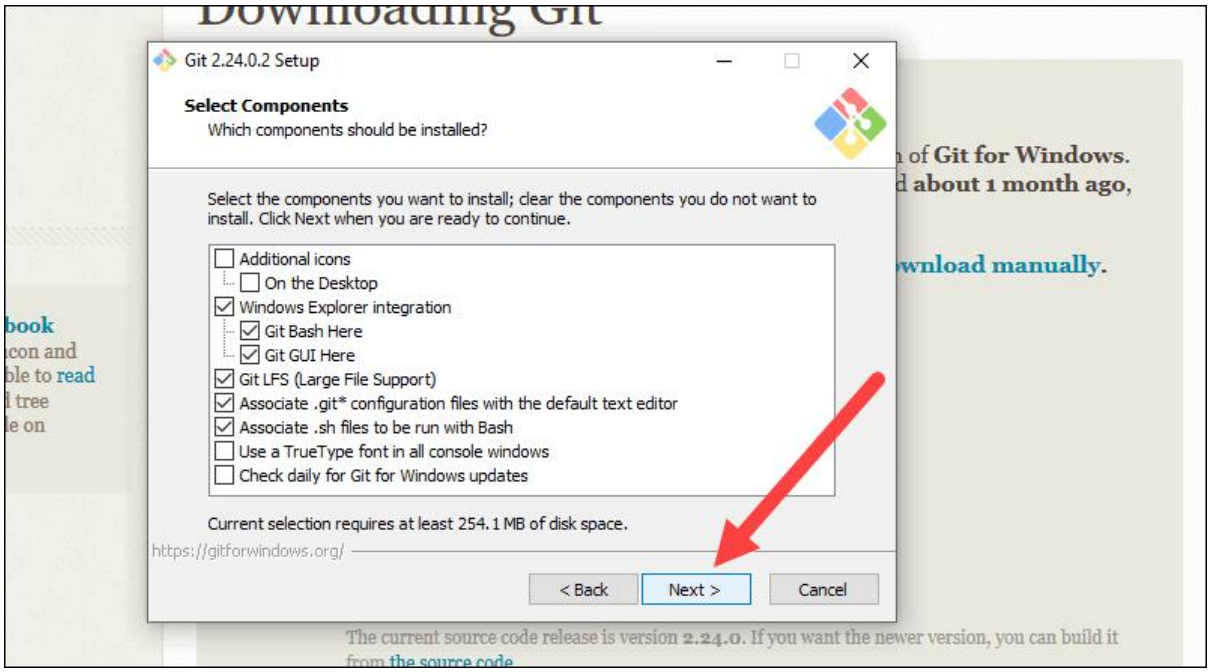
5. Review the GNU General Public License, and when you're ready to install, click **Next**.



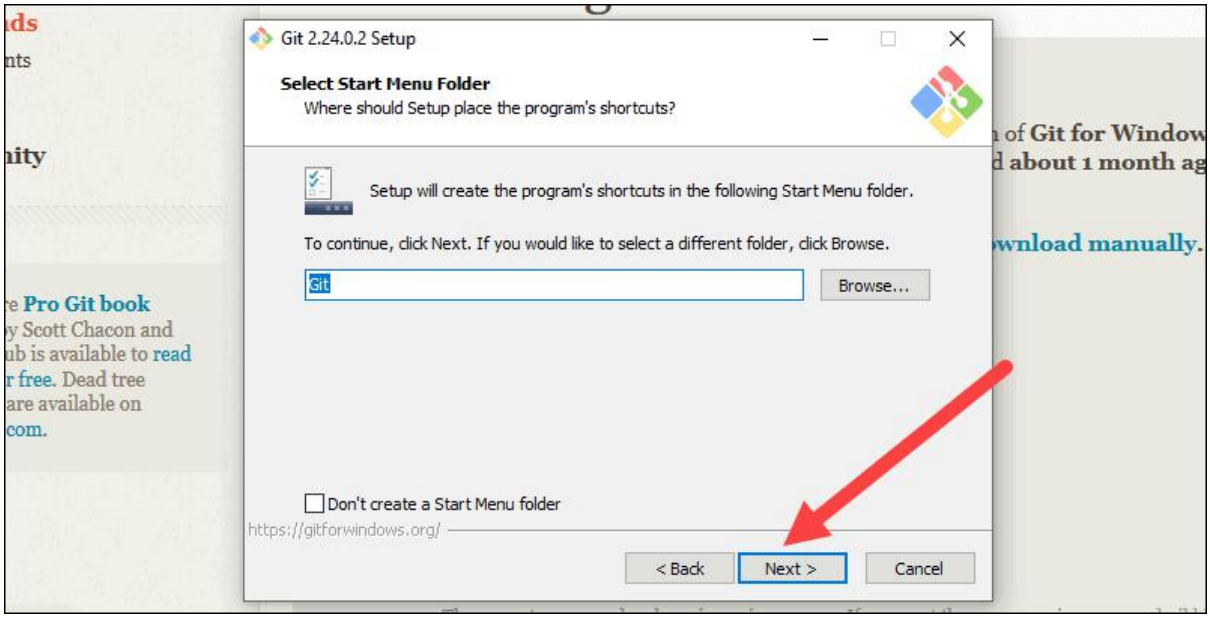
6. The installer will ask you for an installation location. Leave the default, unless you have reason to change it, and click **Next**.



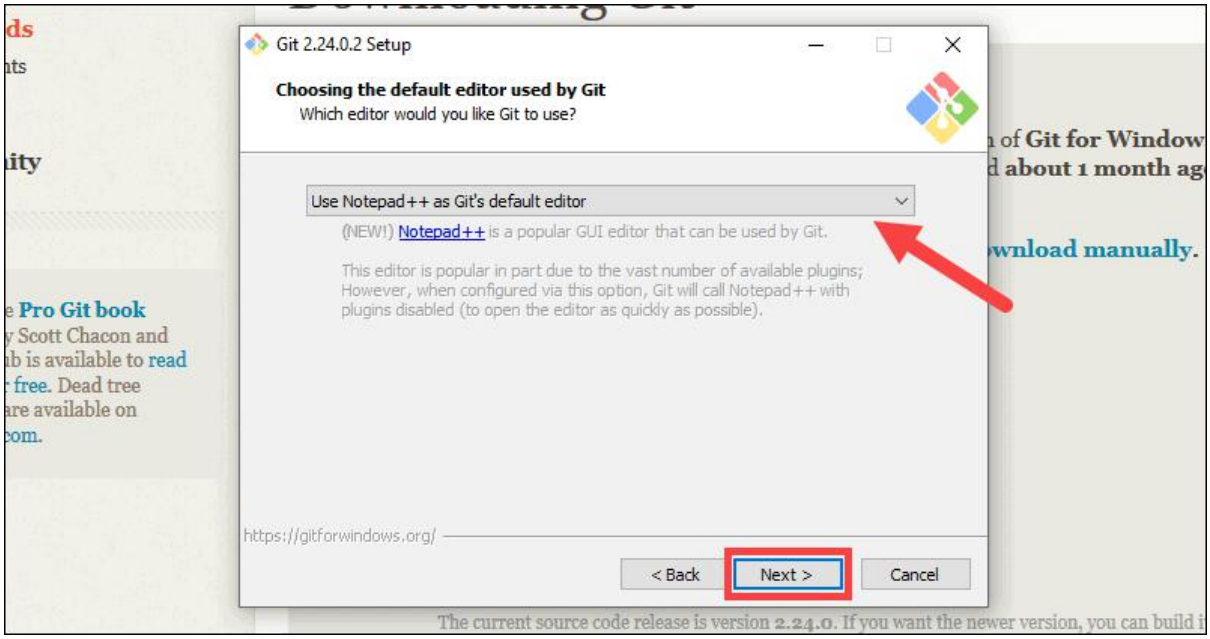
7. A component selection screen will appear. Leave the defaults unless you have a specific need to change them and click **Next**.



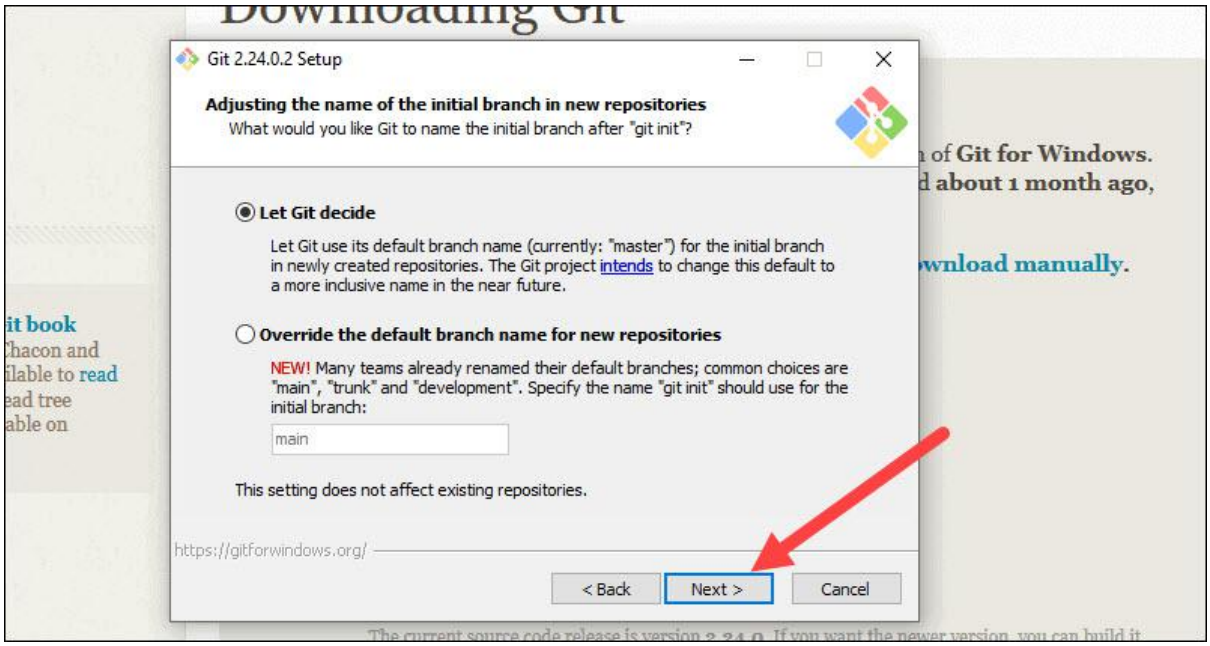
8. The installer will offer to create a start menu folder. Simply click **Next**.



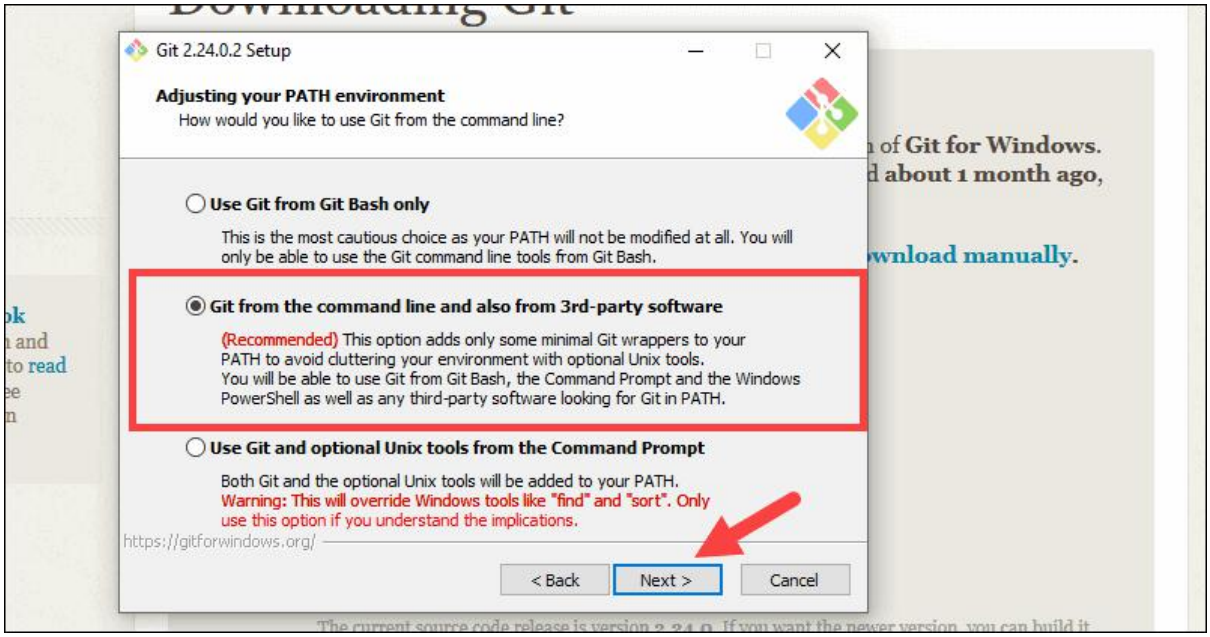
9. Select a text editor you'd like to use with Git. Use the drop-down menu to select Notepad++ (or whichever text editor you prefer) and click **Next**.



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.**

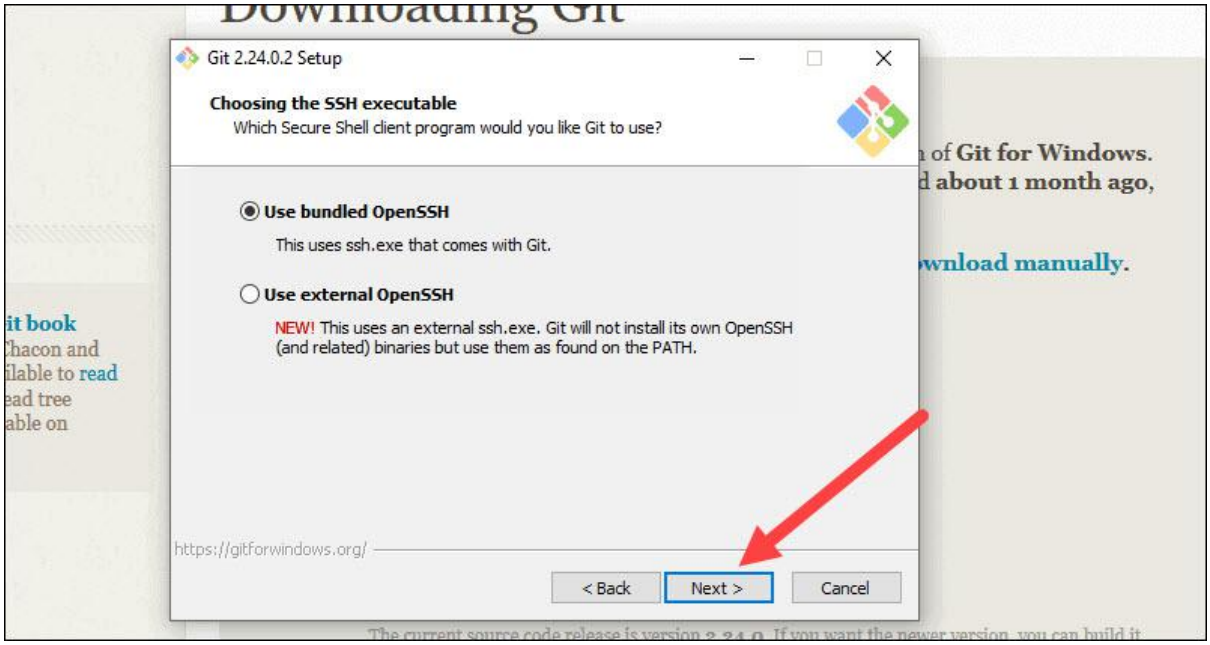


11. This installation step allows you to change the **PATH environment**.
The **PATH** is the default set of directories included when you run a command from the command line. Leave this on the middle (recommended) selection and click **Next**.

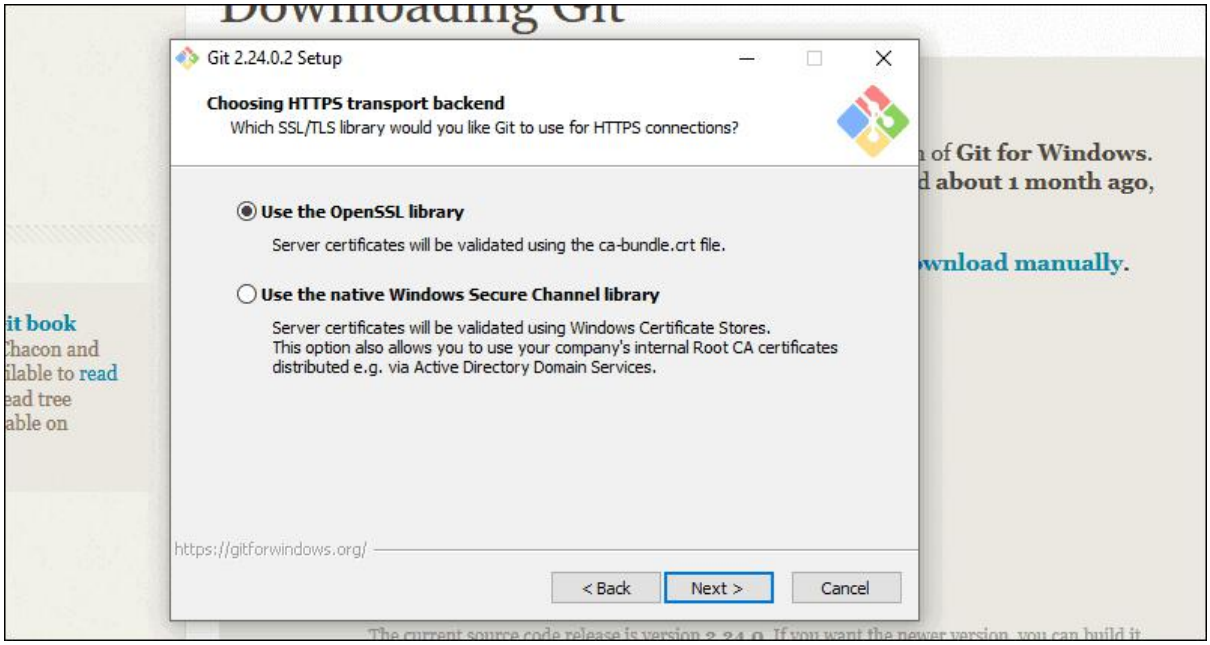


Server Certificates, Line Endings and Terminal Emulators

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.**



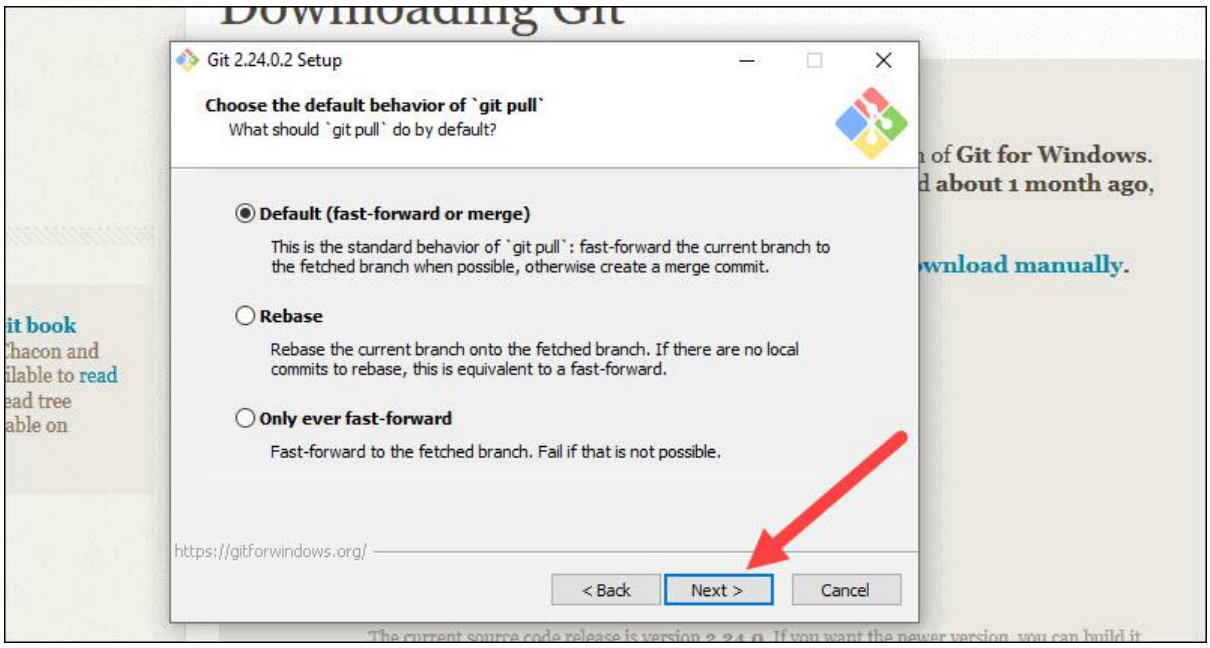
13. The next option relates to server certificates. Most users should use the default. If you're working in an Active Directory environment, you may need to switch to Windows Store certificates. Click **Next**.



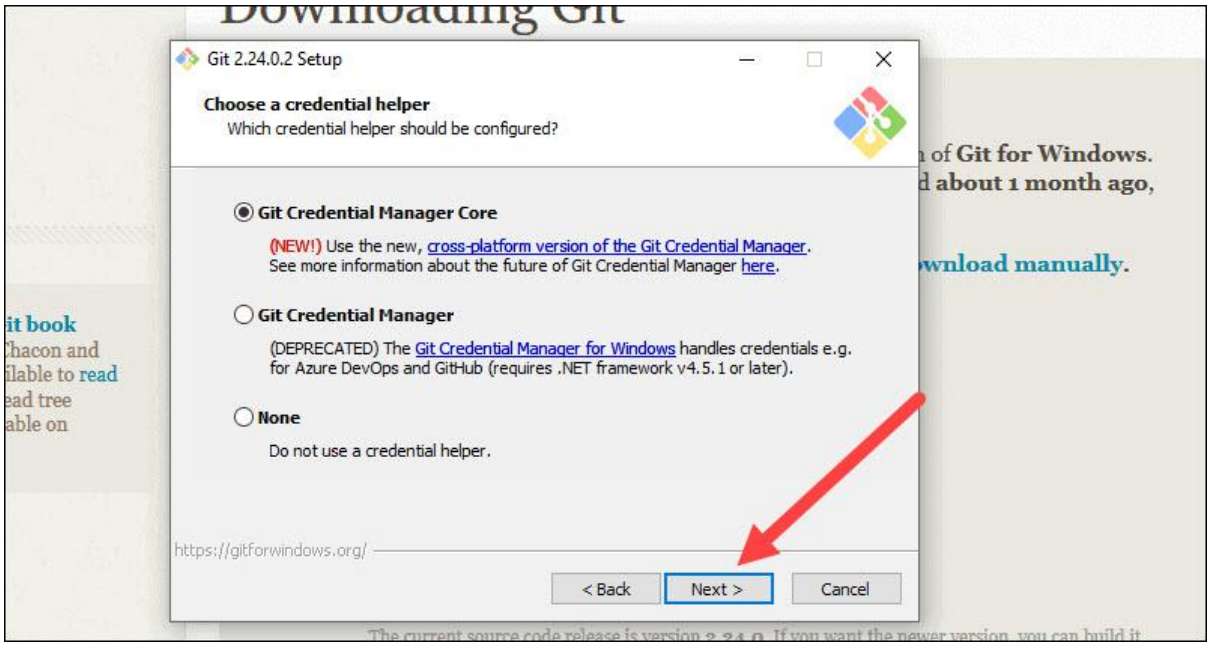
14. The next selection converts line endings. It is recommended that you leave the default selection. This relates to the way data is formatted and changing this option may cause problems. Click **Next**.

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.

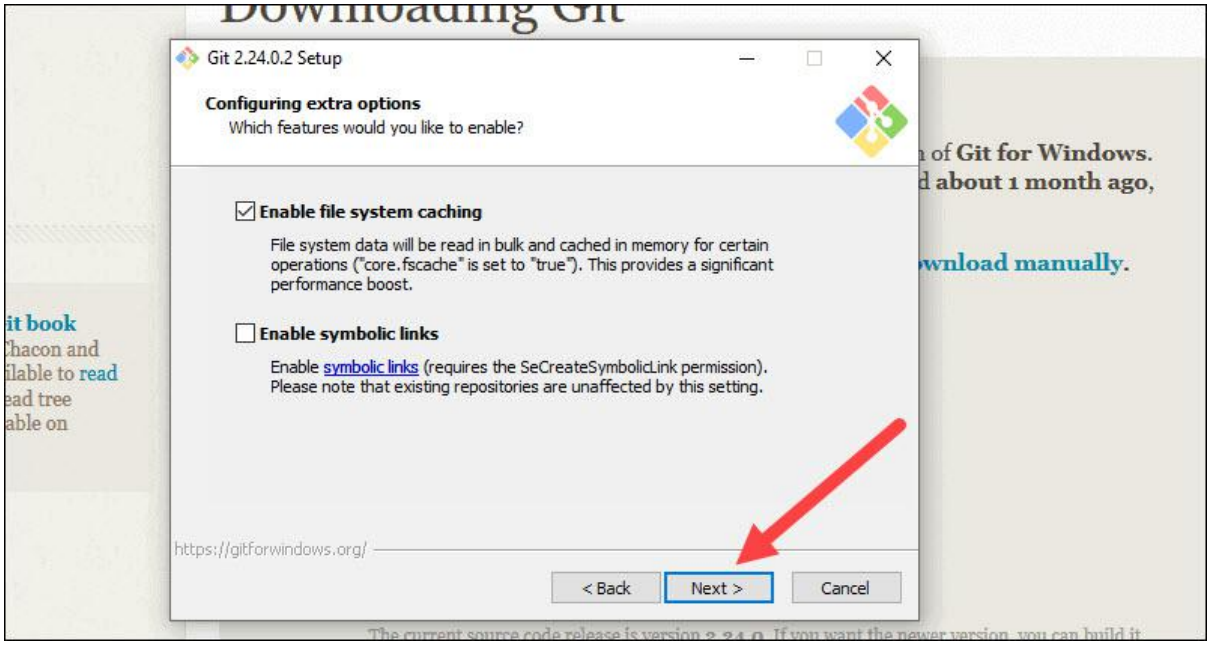


17. Next you should choose which credential helper to use. Git uses credential helpers to fetch or save credentials. Leave the default option as it is the most stable one, and click **Next**.

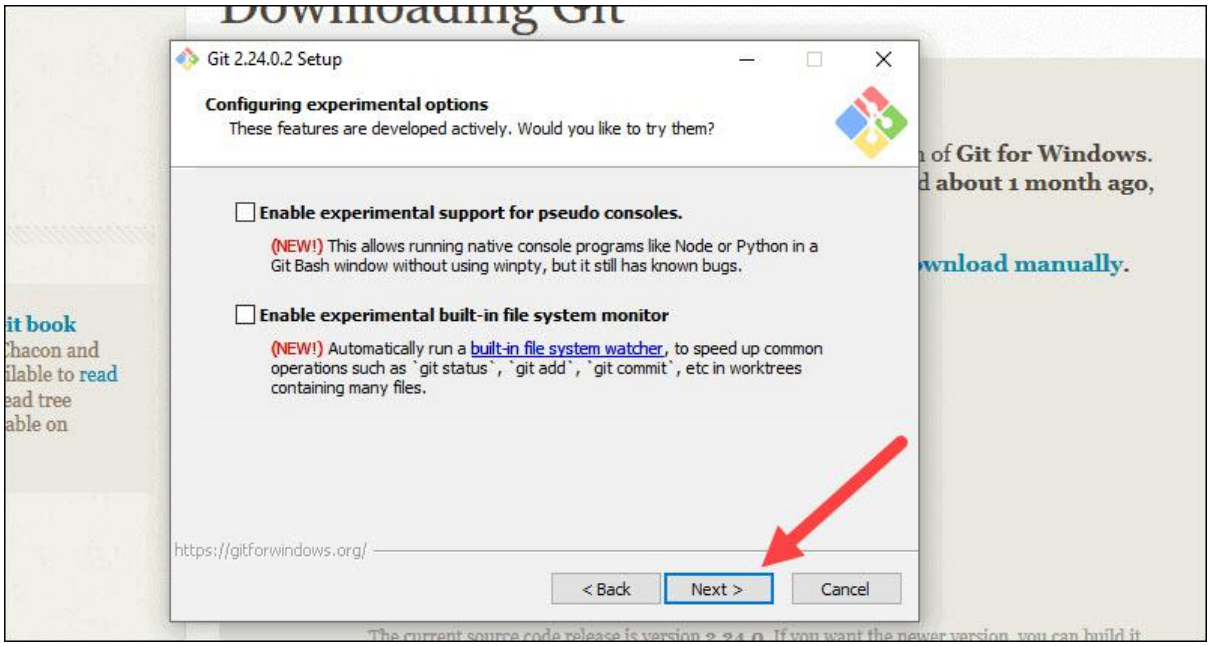


Additional Customization Options

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

Steps For Making an Account in Git Hub

1

God 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 u Block Origin, prevent GitHub's verification CAPTCHA puzzle from appearing. For best results, disable your web browser's ad blocker when signing up for GitHub.

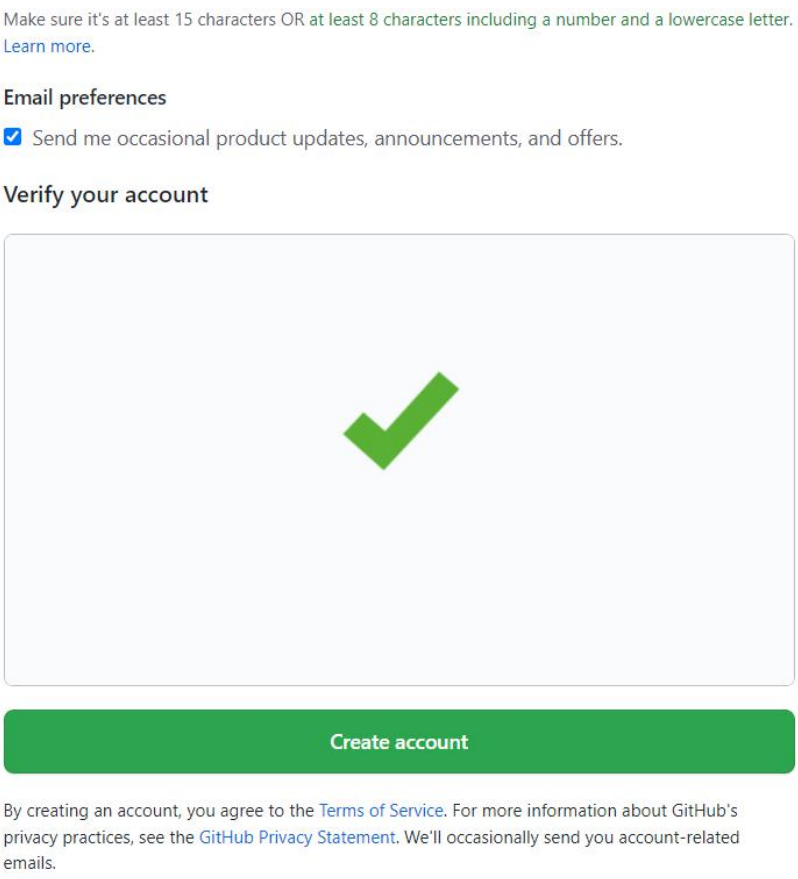
2

Enters 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.

- 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-privacy-statement before you continue. Continuing past the next step confirms that you agree to both documents.

3

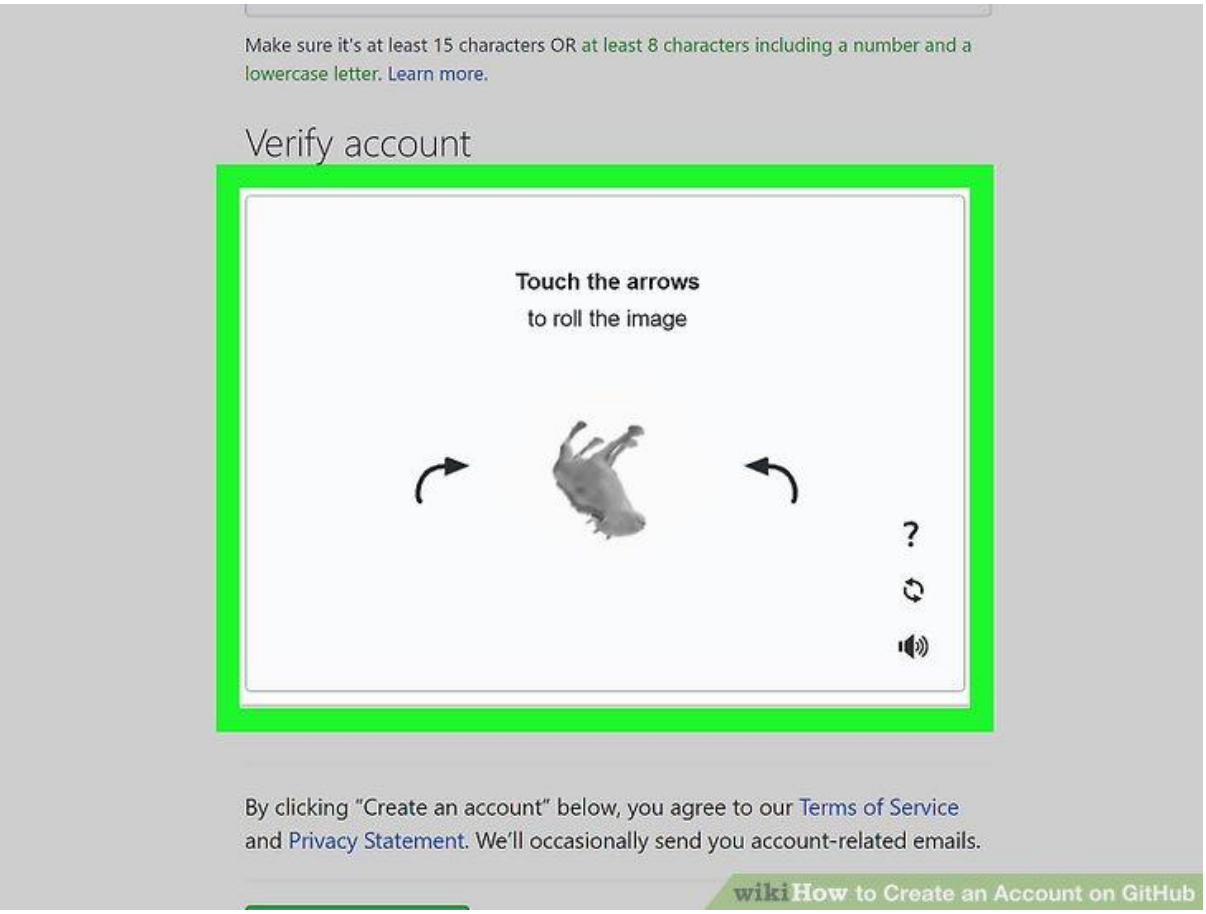
Click the green Create an account **button.** It's below the form.



4

Complete the CAPTCHA puzzle. The instructions vary by puzzle, so just follow the on-screen instructions to confirm that you are a human.

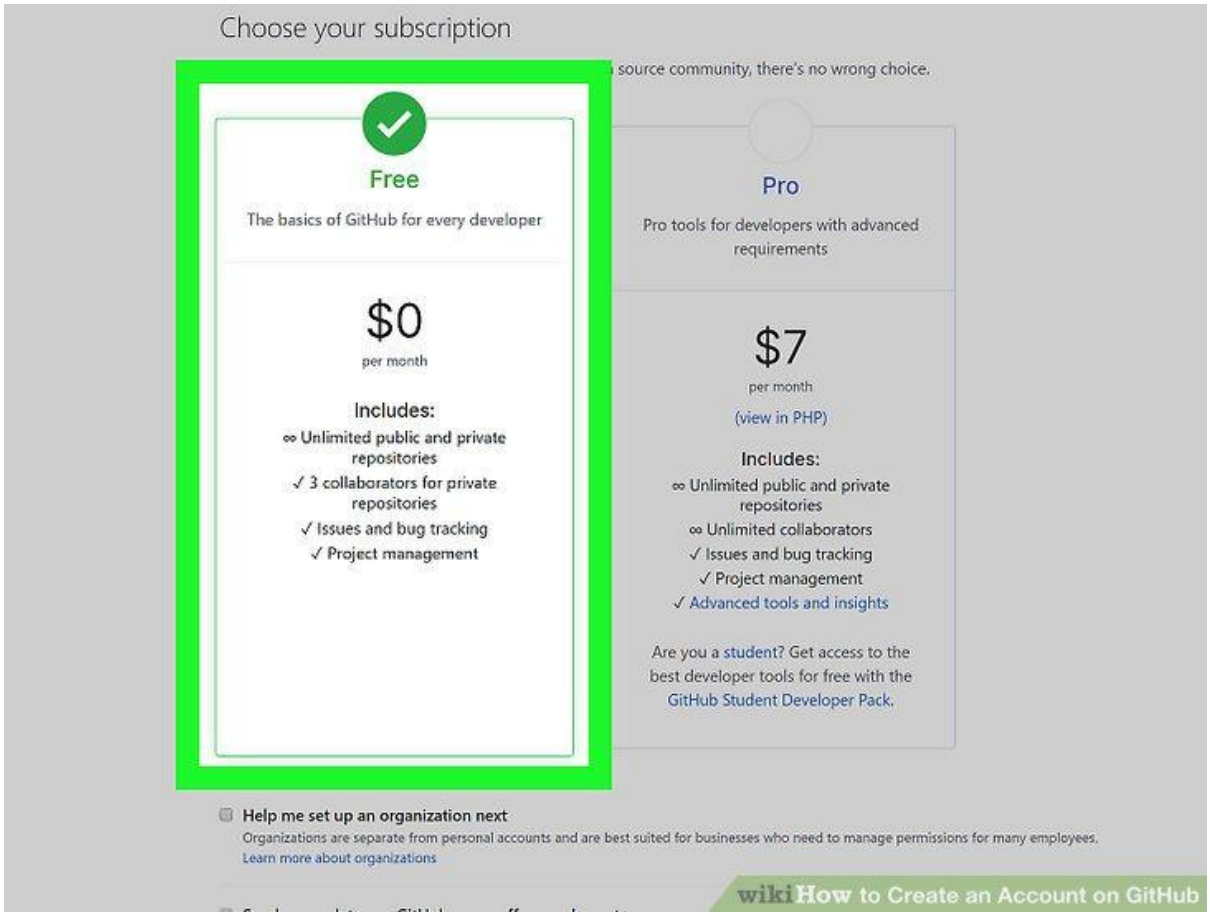
- If you see an error that says "Unable to verify your captcha response," it's because your web browser's ad blocking extension prevented the CAPTCHA puzzle from appearing. Disable all ad-blocking extensions, refresh the page, and then click **VERIFY** to start the CAPTCHA.



5

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:[2]

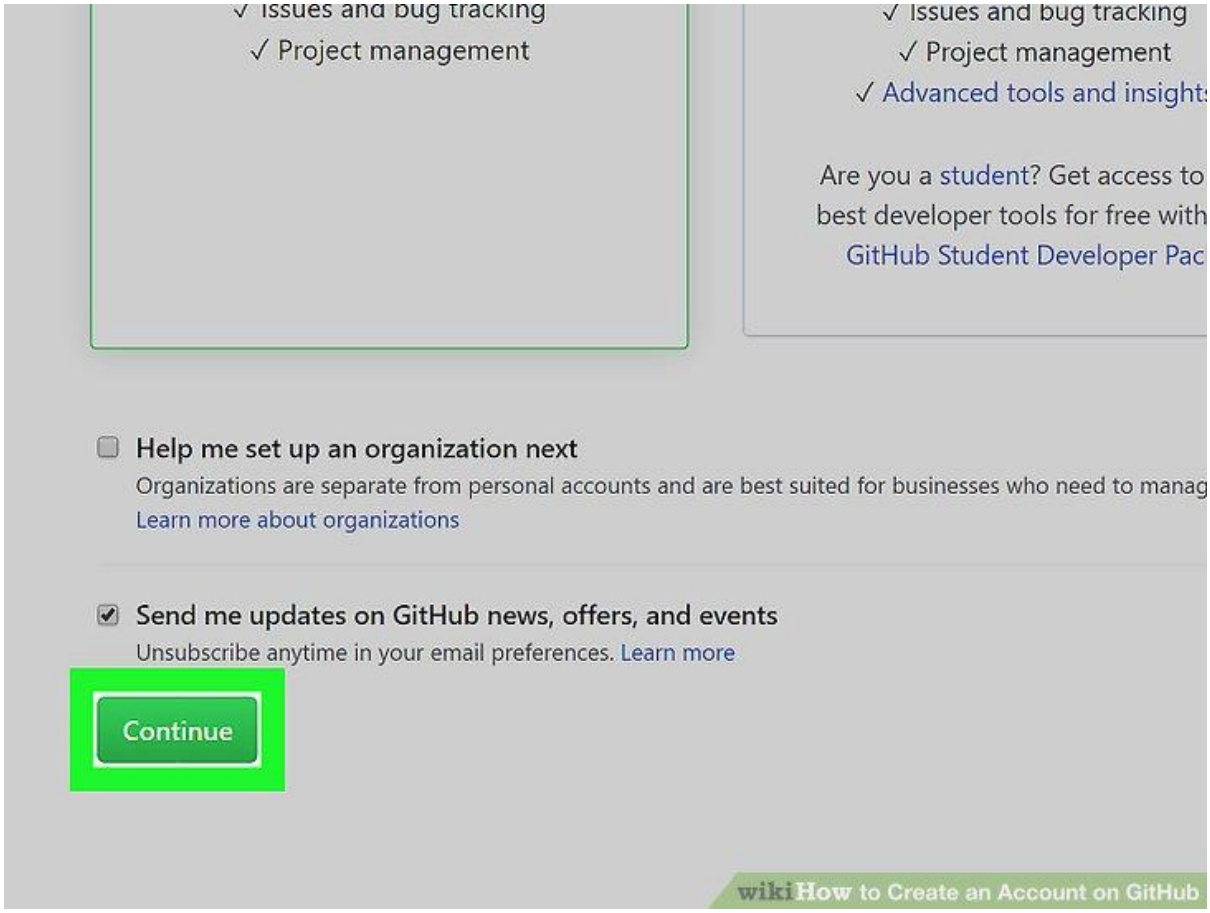
- **Free:** Unlimited public and private repositories, up to 3 collaborators, issues and bug tracking, and project management tools.
- **Pro:** Unlimited access to all repositories, unlimited collaborators, issue & bug tracking, and advanced insight tools.
- **Team:** All of the aforementioned features, 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.



6

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 chose a paid plan, you'll have to enter your payment information as requested before you can continue.



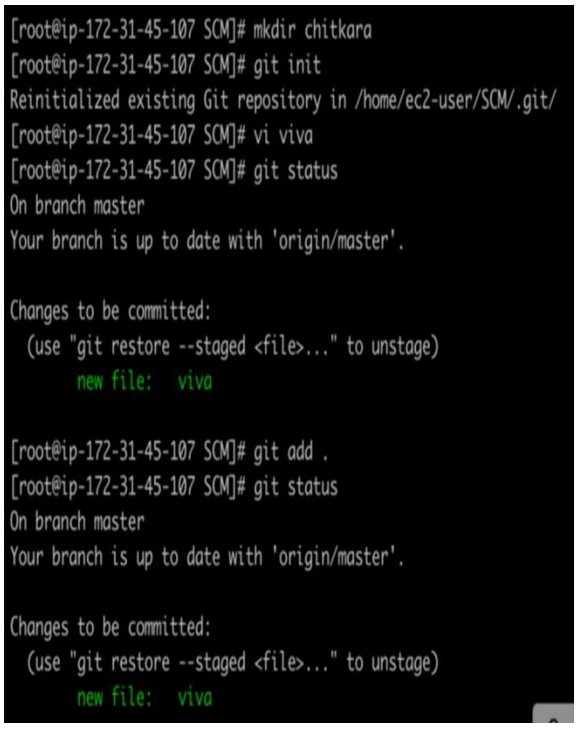
7

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

AIM:-how to use Git Log

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. For this, you need to use `git log`.



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.

In conjunction with these commands, you'll also need git status to view the state of the working directory and the staging area.

```
[root@ip-172-31-45-107 SCM]# git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
[root@ip-172-31-45-107 SCM]#
```

Git Commit

Since we have finished our work, we are ready move from stage to commit for our repo.

Adding commits keep track of our progress and changes as we work. Git considers each commit change point or "save point". It is a point in the project you can go back to if you find a bug, or want to make a change.

When we commit, we should **always** include a **message**.

By adding clear messages to each commit, it is easy for yourself (and others) to see what has changed and when.

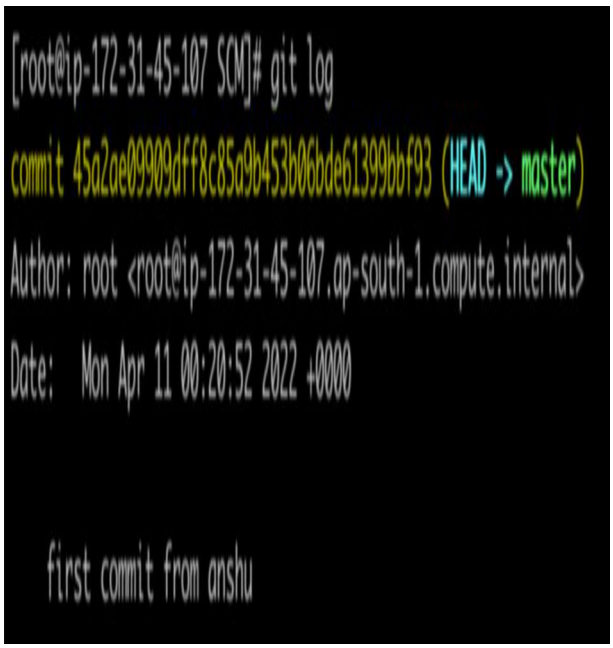
```
git commit –m "commit message"
```

```
[root@ip-172-31-45-107 SCM]# git commit -m "first commit from anshu"
[master 45a2ae0] first commit from anshu
 Committer: root <root@ip-172-31-45-107.ap-south-1.compute.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 1 file changed, 5 insertions(+)
 create mode 100644 viva
[root@ip-172-31-45-107 SCM]# git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
[root@ip-172-31-45-107 SCM]#
```

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.



Experiment No. 04

CHITKARA UNIVERSITY

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
```

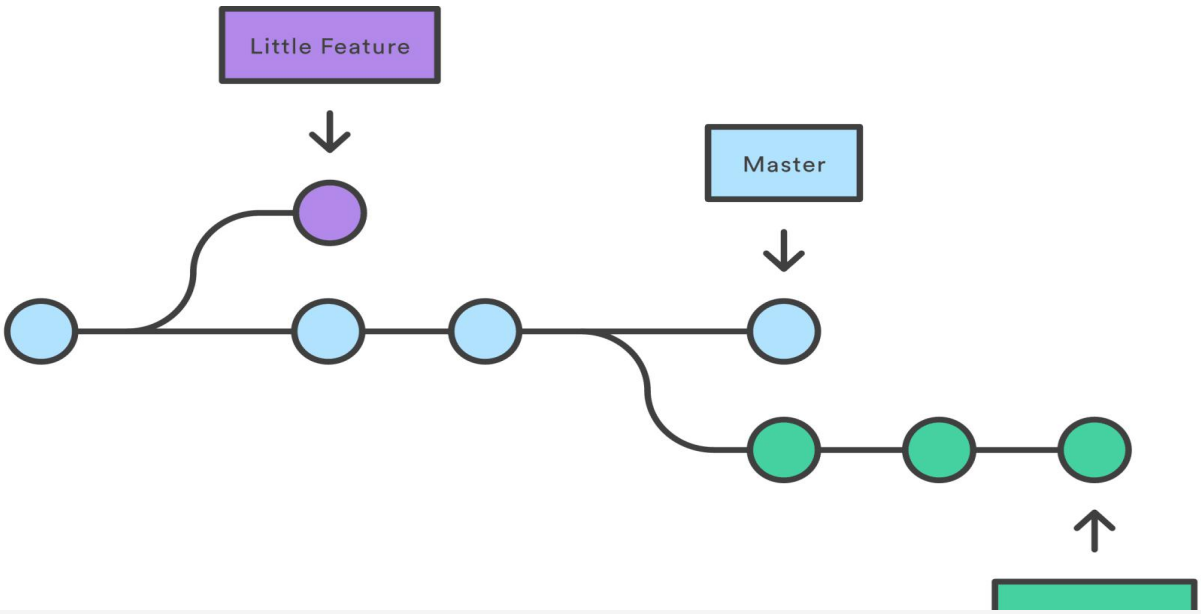
List all of the branches in your repository. This is synonymous with `git branch --list`.

```
git branch <branch>
```

Create a new branch called `<branch>`. Thisdoes *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:

```
gitbranch 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.

```
[root@ip-172-31-45-107 SCM]# git branch
  branch1
* master
[root@ip-172-31-45-107 SCM]# git branch branch2
[root@ip-172-31-45-107 SCM]# git checkout branch2
Switched to branch 'branch2'
[root@ip-172-31-45-107 SCM]#
```

Merging into Master — Fun Part

This is the only step that is different from the steps followed in the rebase article.

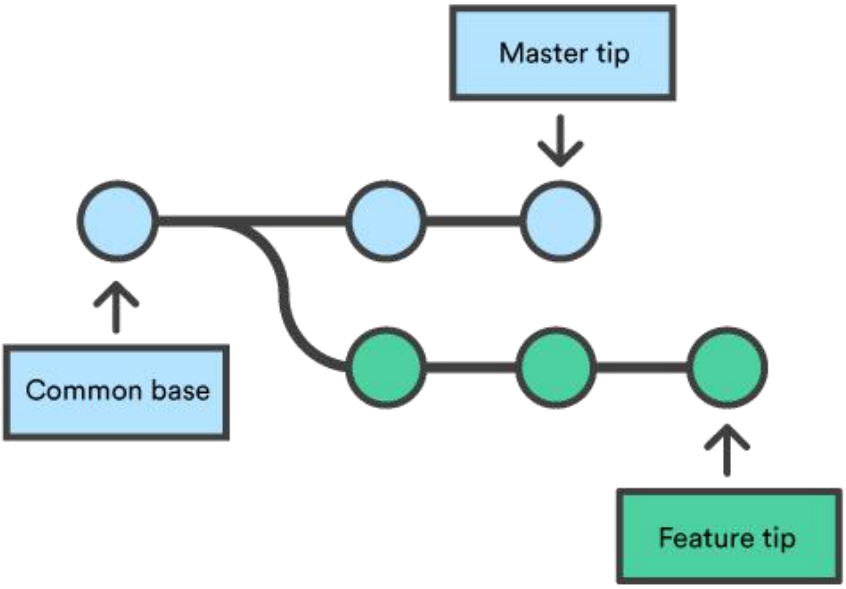
Note that the changes are added to the current branch from the selected branch when doing a merge. Therefore we should first move to the **master** branch to start the merge.

```
git checkout master
```

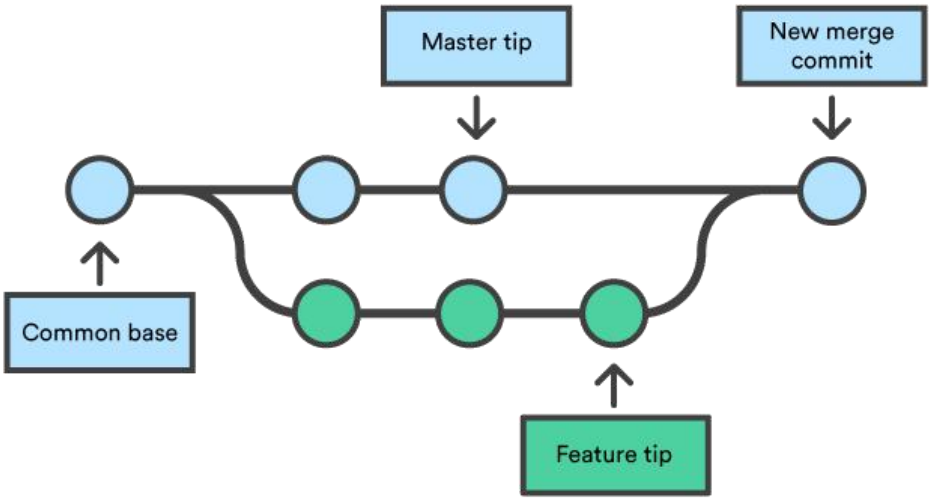
Now you can use the following command to move the changes from **dev** to **master**.

```
git merge <Branch name>
```

When we use this command, Git will find a common commit between the 2 branches and create a new *merge commit* on the **master** combining the queued changes in the **dev** branch.



Before Merge(Image from Git Documentation)




```
[root@ip-172-31-45-107 SCM]# git branch
  branch1
* master
[root@ip-172-31-45-107 SCM]# git branch branch2
[root@ip-172-31-45-107 SCM]# git checkout branch2
Switched to branch 'branch2'
[root@ip-172-31-45-107 SCM]# git merge branch2
Already up to date.
[root@ip-172-31-45-107 SCM]#
```

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

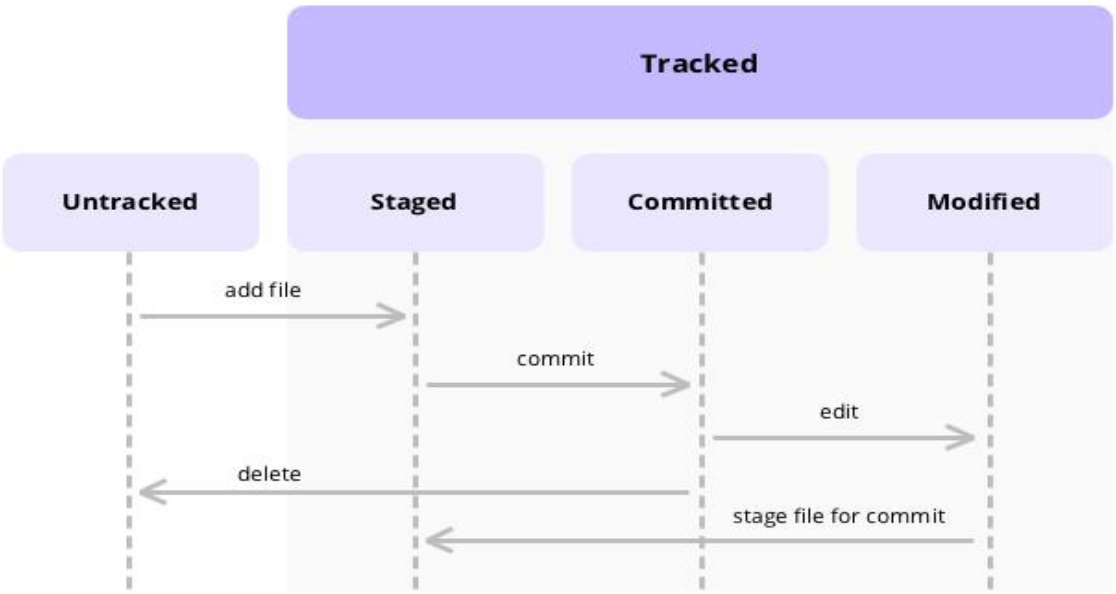
Workflow of Git Life Cycle

The workflow of the Git as follows:

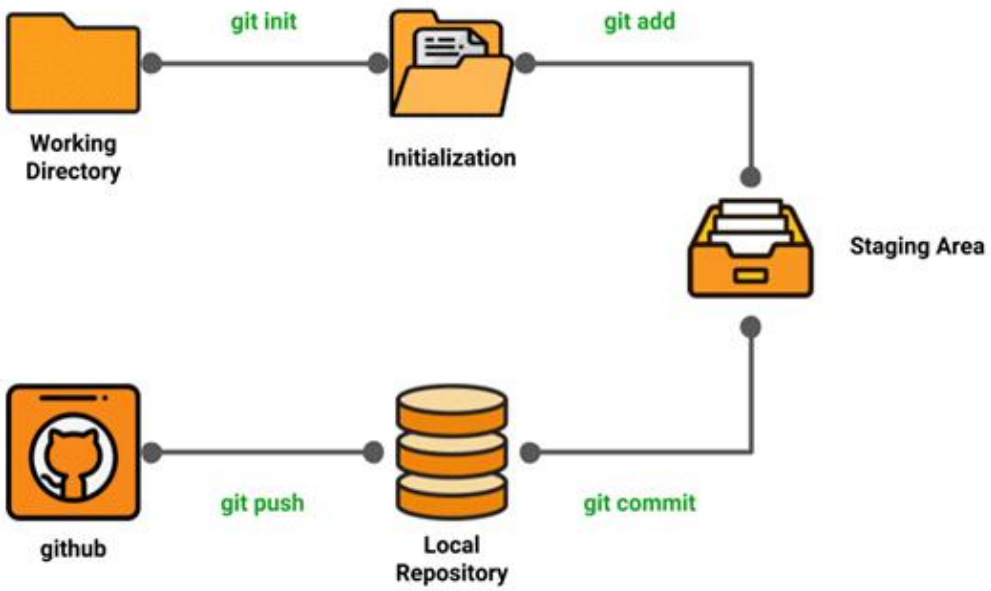
- We will create a branch on which we can work on and later we will merge it with master

- 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.

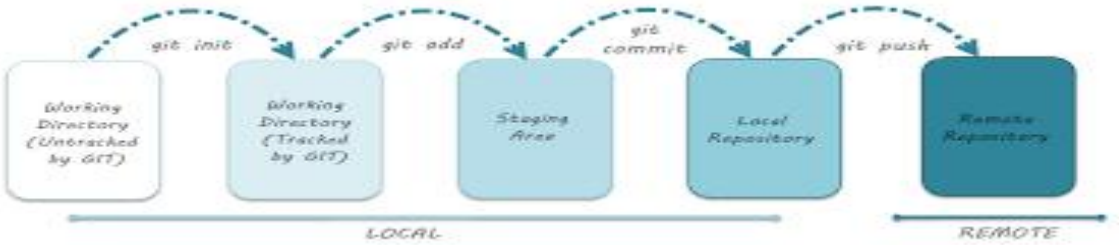


- We need to move the content that we require to transform to the master to the staging area by using git commands and the snapshot of staged files will be saved in the git staging area.
- We need to perform commits on the files that are staged and the recorded snapshot from the above steps will be permanently 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.

Subject Name: **Source Code Management**

Subject Code: **CS181**

Cluster: **Alpha**

Department: **DCSE**

Submitted By:ANSHU KUMAR
2110990220
G8

Submi:Dr. M o n i t
K a p o o r

 # List of Programs #

Experiment No. 06

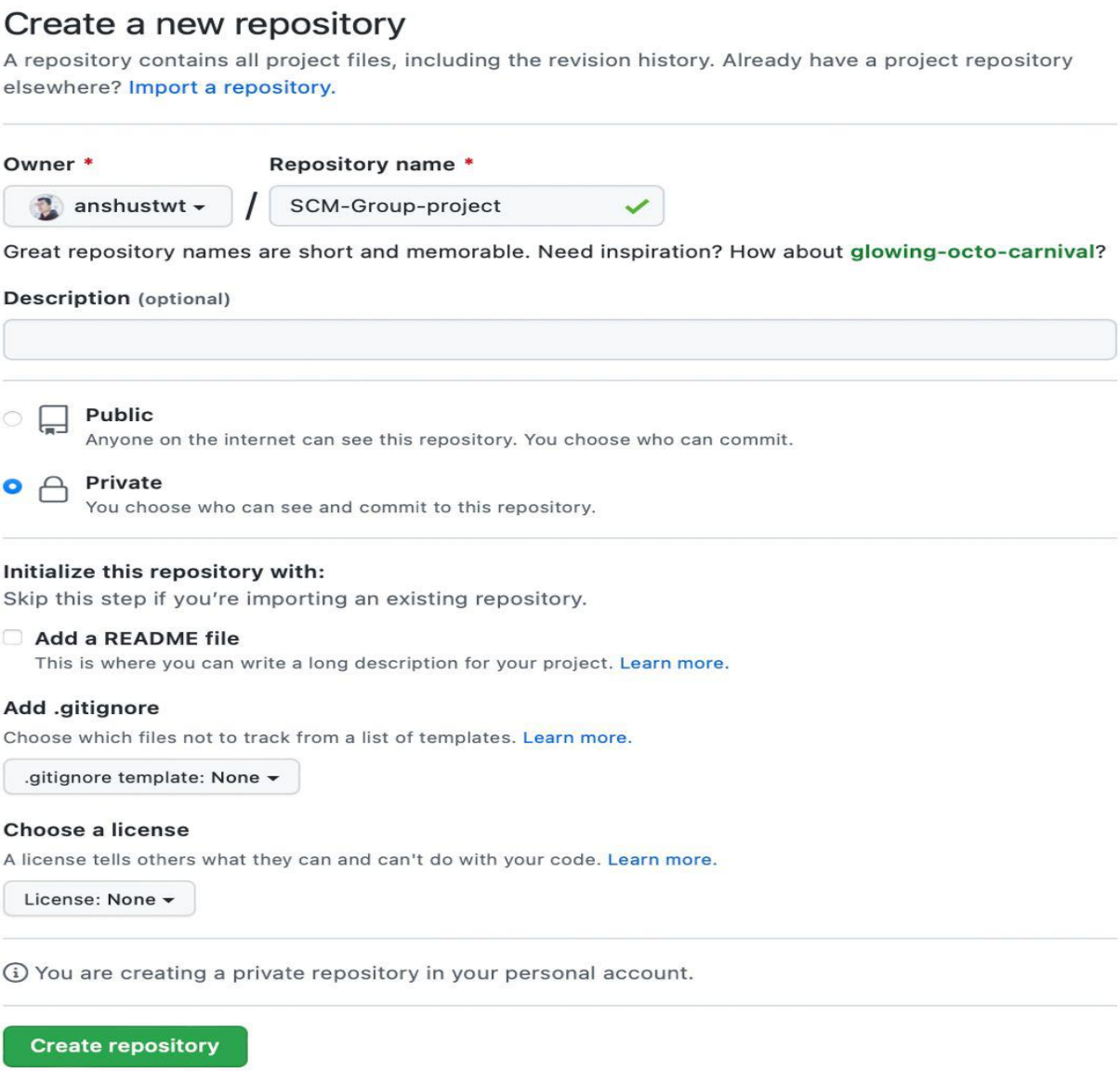
Aim: Add collaborators on Github Repo

Theory:

1. Create a New Repository.

   A repository is like a folder for your project. You can have any number of public and private repositories in your user account. Repositories can contain folders and files, images, videos, spreadsheets, and data sets, as well as the revision history for all files in the repository. For more information, see "About repositories."

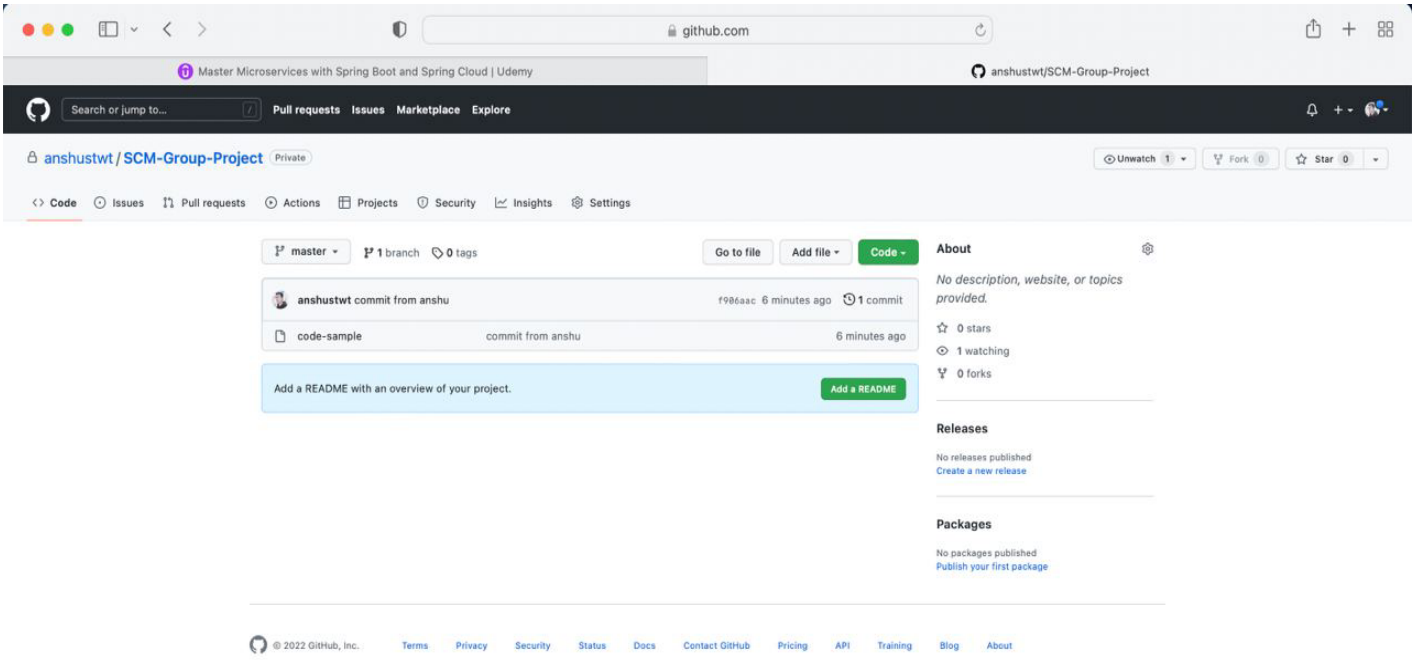
   When you create a new repository, you should initialize the repository with a README file to let people know about your project. For more information, see "Creating a new repository."



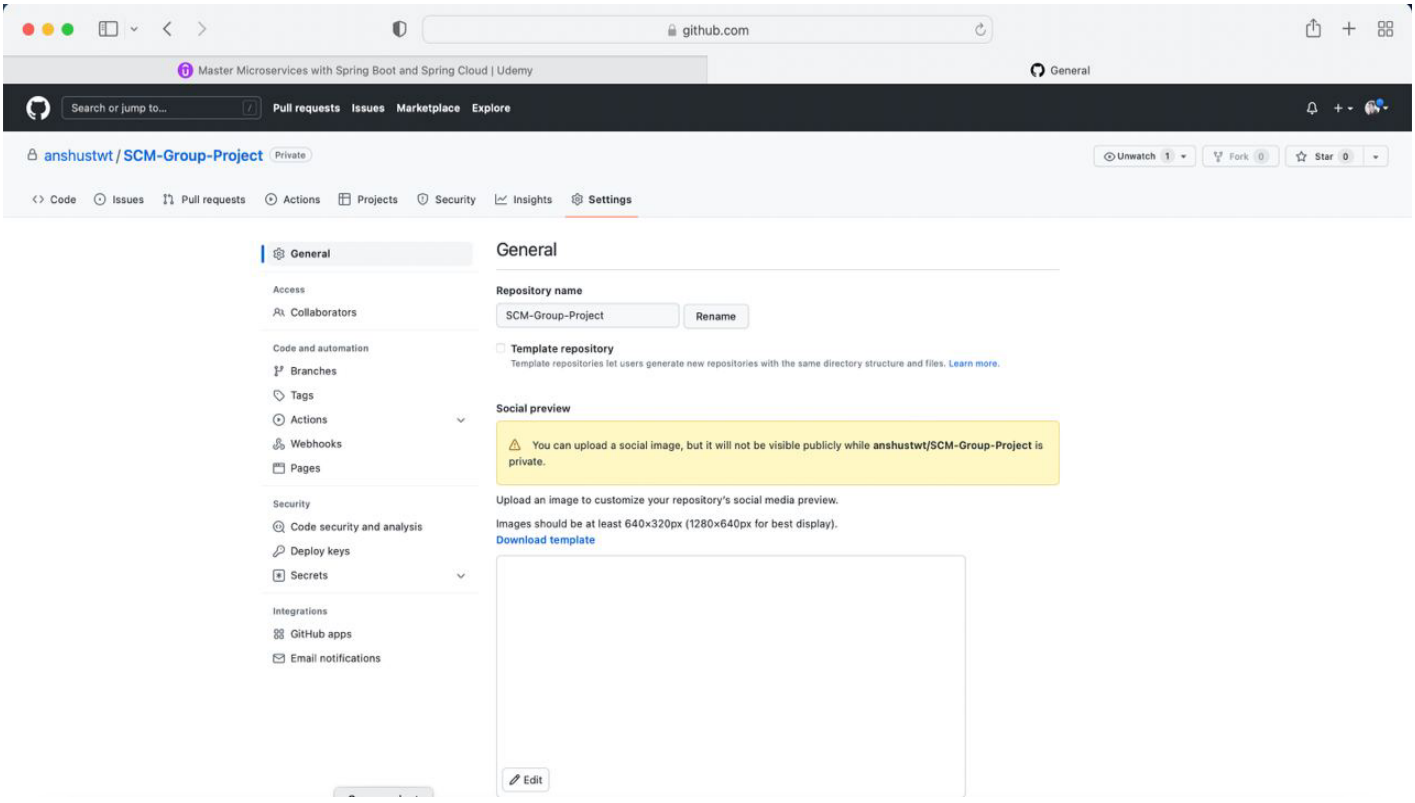
2. Now Copy the HTTP link of your repo and paste it it on your ‘Git CLI’, and merge the local repo in remote repo

```
🚀 workspace % mkdir SCM-Group-Project
🚀 workspace % cd SCM-Group-Project
🚀 SCM-Group-Project % git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /Users/anshukumar/workspace/SCM-Group-Project/.git/
🚀 SCM-Group-Project % vi code-sample
🚀 SCM-Group-Project % git add .
🚀 SCM-Group-Project % git commit -m "commit from anshu"
[master (root-commit) f906aac] commit from anshu
 1 file changed, 32 insertions(+)
 create mode 100644 code-sample
🚀 SCM-Group-Project % git remote add origin https://github.com/anshustwt/SCM-Group-Project.git
🚀 SCM-Group-Project % git push -u origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 744 bytes | 744.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/anshustwt/SCM-Group-Project.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
🚀 SCM-Group-Project %
```

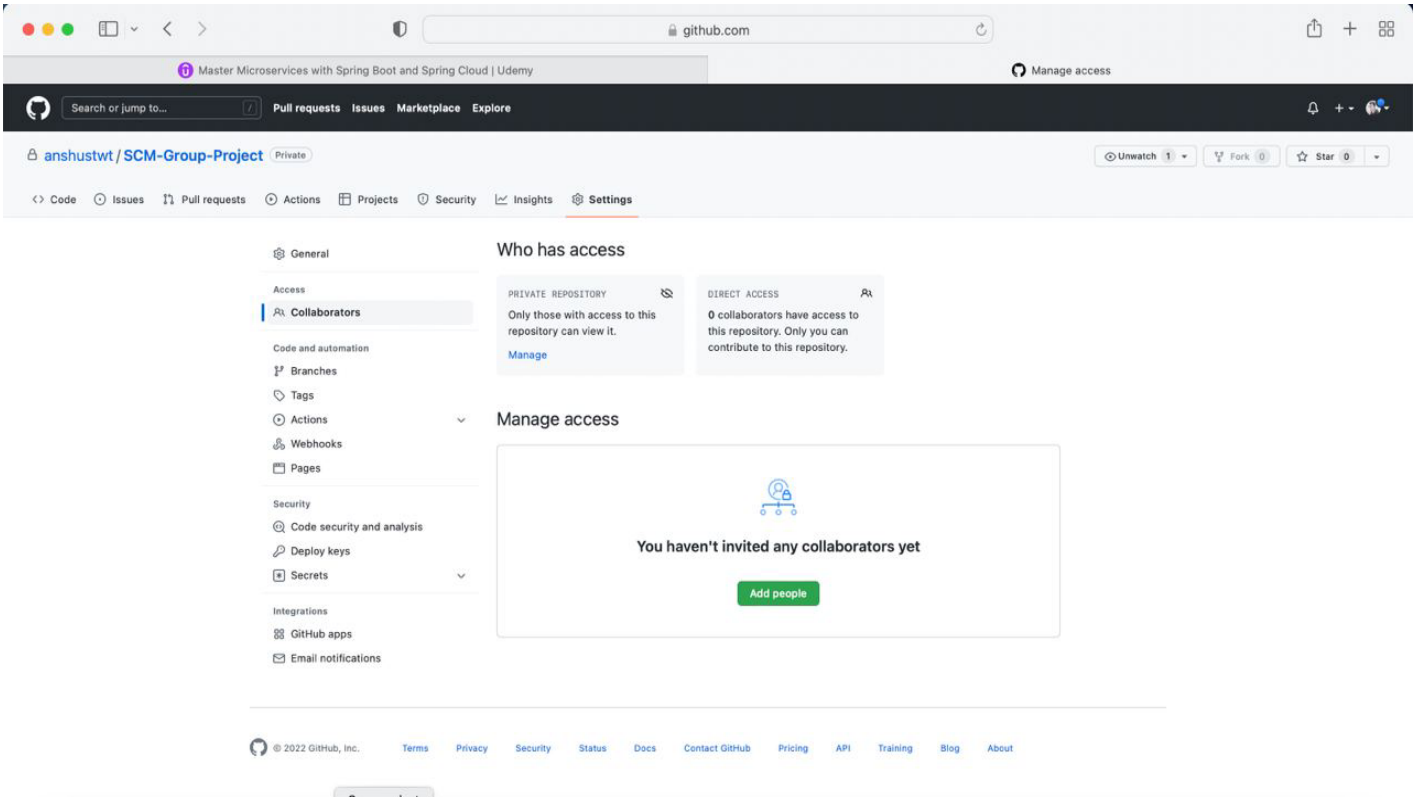
3. Go to Collaborators in Repo Setting, Add the username or email of Collaborator you want to add in your Repo.
    - To add members to your repository, open your repository and select settings option in the navigation bar.



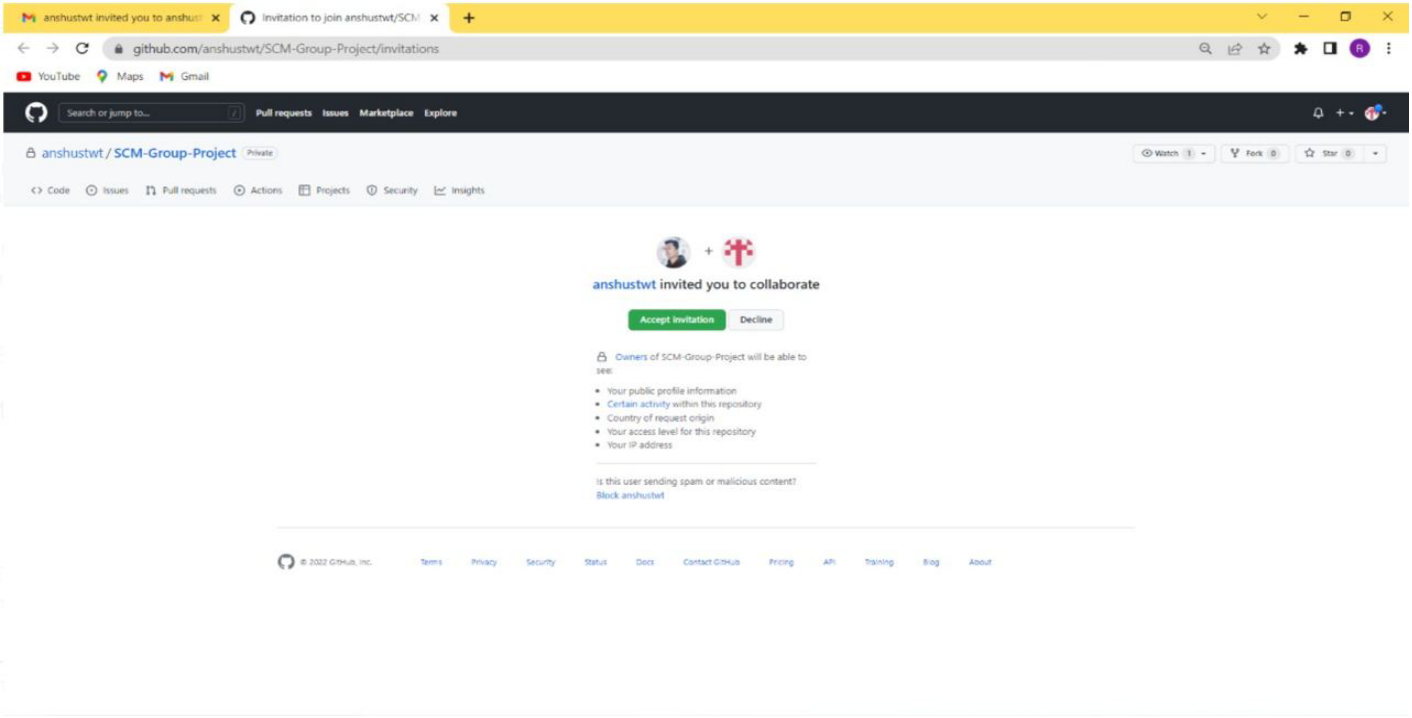
- Click on Collaborators option under the access tab.



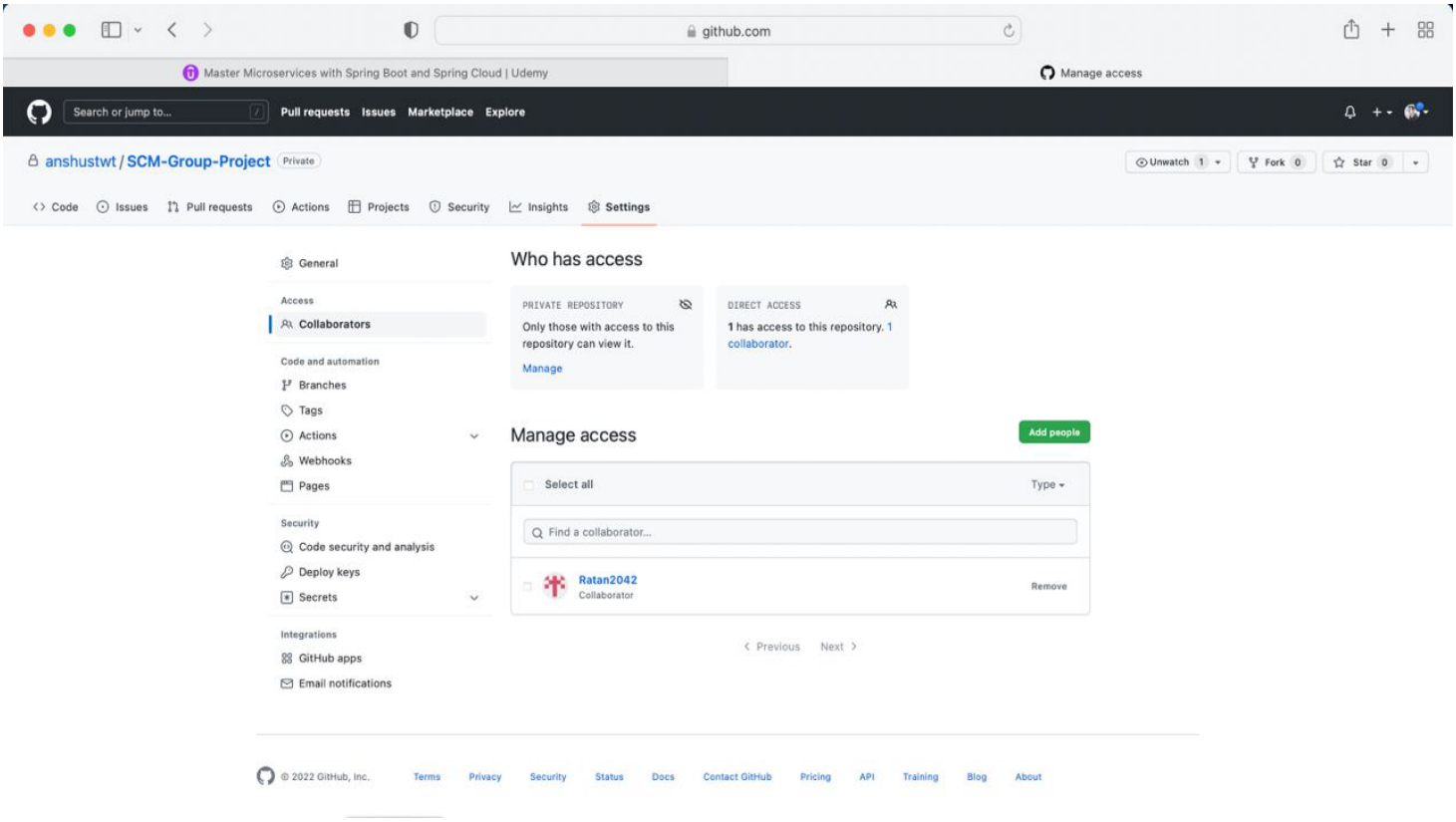
- After clicking on collaborators GitHub asks you to enter your password to confirm the access to the repository.
- After entering the password you can manage access and add/remove team members to your project.
- To add members click on the add people option and search the id of your respective team member.



- Invitation Mail is sent to the Collaborator, the collaborator has to accept this Invitation.



- Now Collaborator has now access to the this Repo.


github.com
Master Microservices with Spring Boot and Spring Cloud | Udemy
Manage access
Search or jump to...
Pull requests
Issues
Marketplace
Explore
anshustwt / SCM-Group-Project
Private
Unwatch 1
Fork 0
Star 0
Code
Issues
Pull requests
Actions
Projects
Security
Insights
Settings
General
Access
Collaborators
Code and automation
Branches
Tags
Actions
Webhooks
Pages
Security
Code security and analysis
Deploy keys
Secrets
Integrations
GitHub apps
Email notifications
Who has access
PRIVATE REPOSITORY
Only those with access to this repository can view it.
Manage
DIRECT ACCESS
1 has access to this repository. 1 collaborator.
Manage access
Add people
Select all
Type
Find a collaborator...
Ratan2042
Collaborator
Remove
Previous
Next
© 2022 GitHub, Inc.
Terms
Privacy
Security
Status
Docs
Contact GitHub
Pricing
API
Training
Blog
About

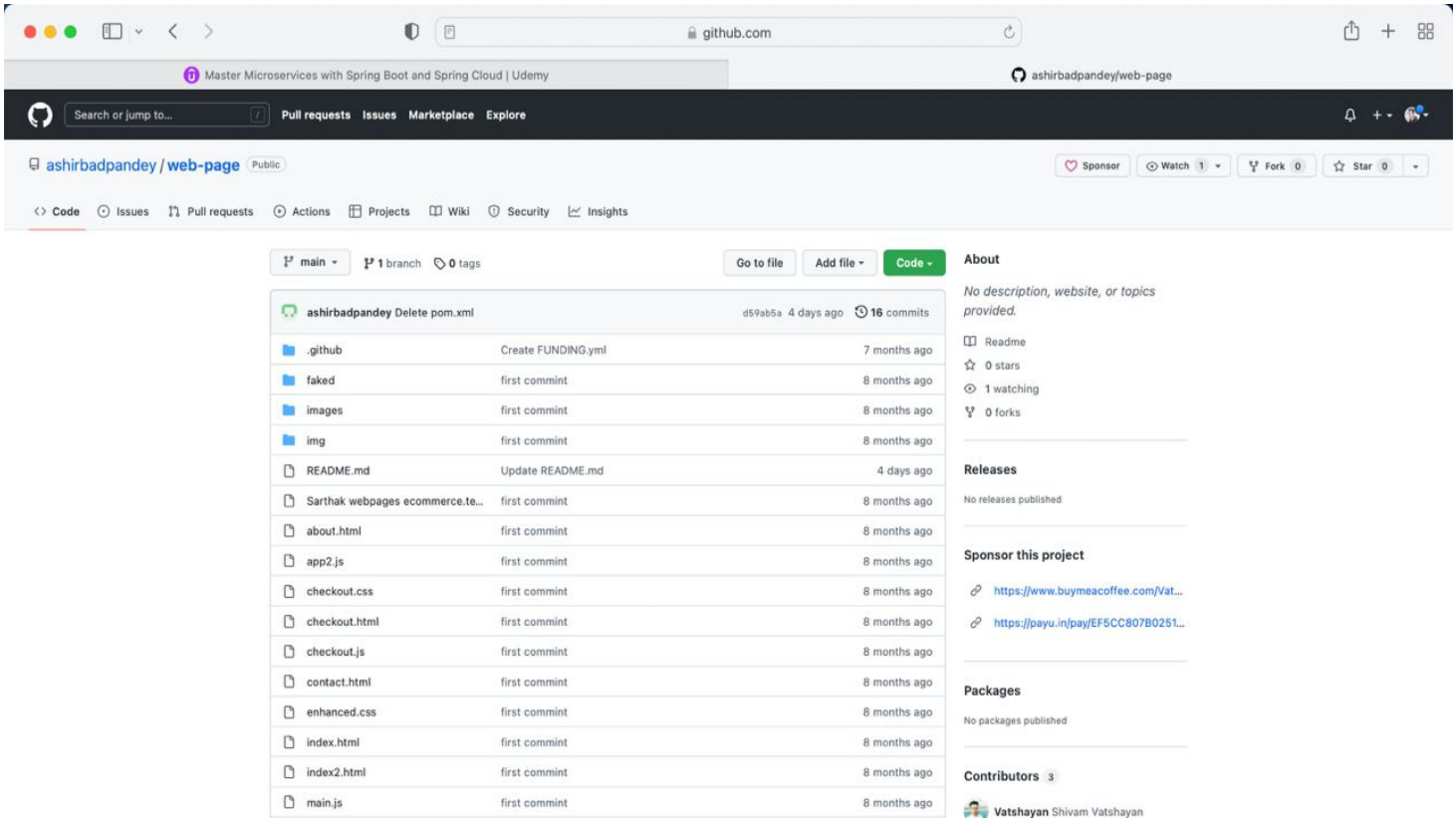
Experiment No. 07

Aim: Fork and Commit

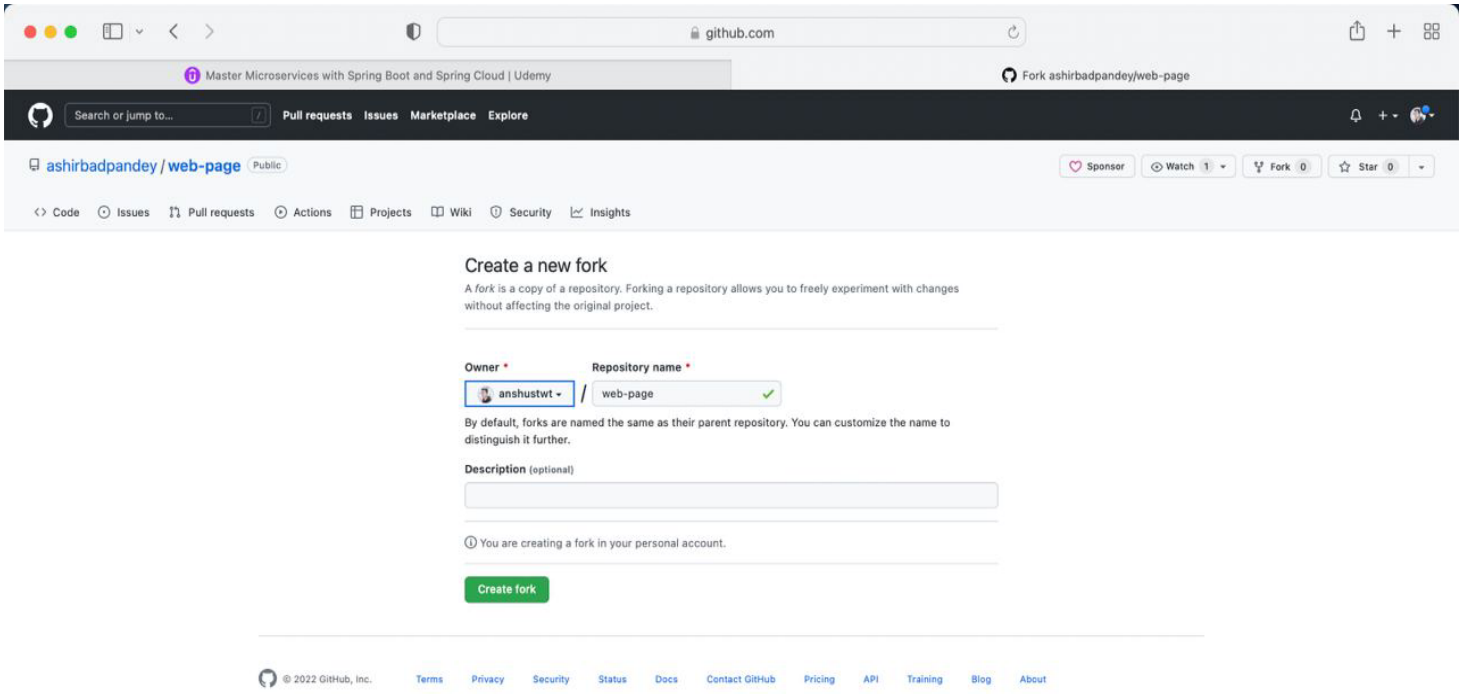
Theory:

Fork :- A Fork is copy of a repository . Forking a repository allows you to freely experiment with changes without affecting the original Project.

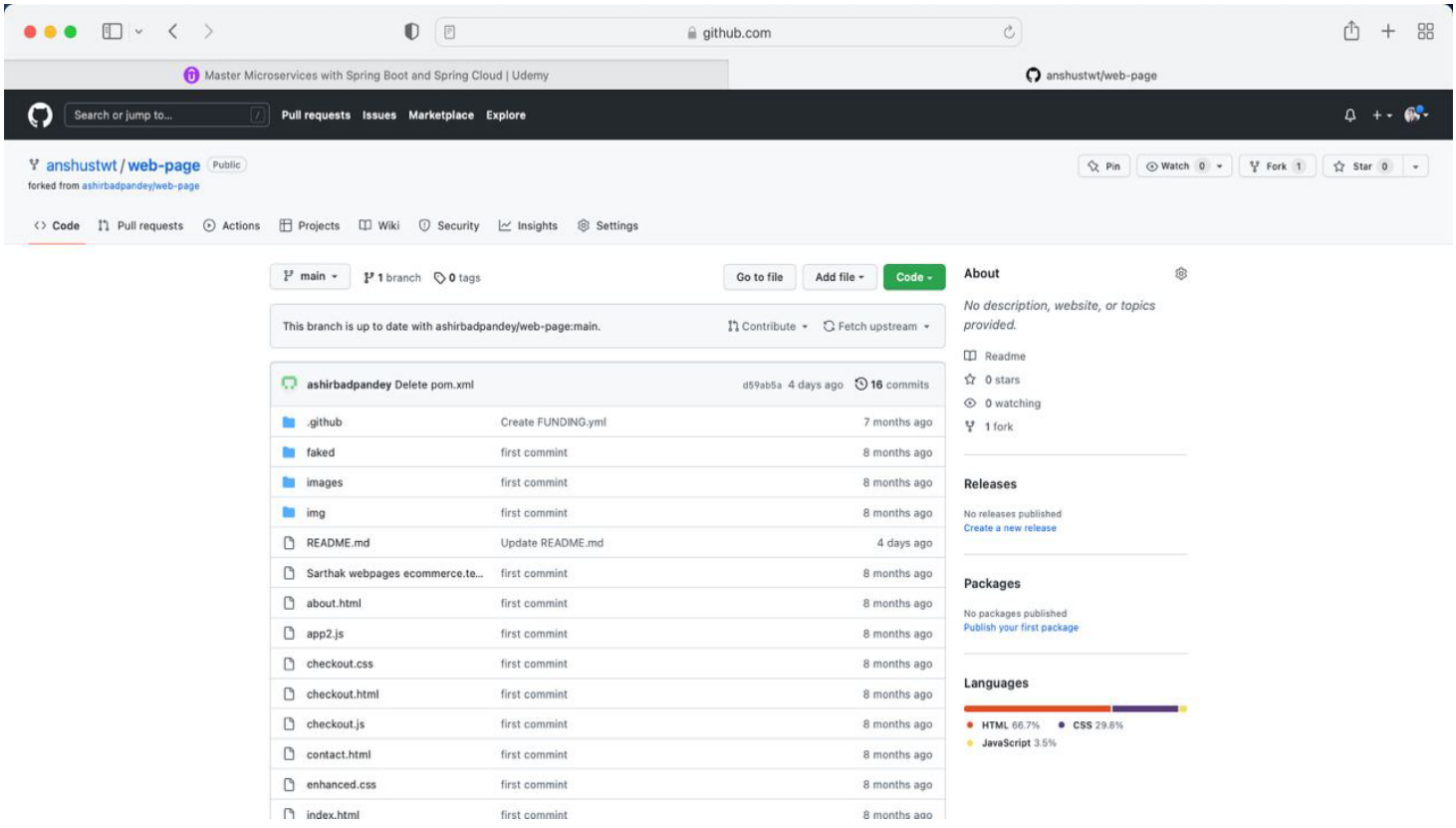
1. To fork a repository first thing you need to do is, sign in to your GitHub account and then you came to the repository you want to fork, so here for demo purpose am using panda repository.



2. Click on the **Fork** button on right upside corner. Then it will ask to create a new fork, add description if you want and then click on create fork.



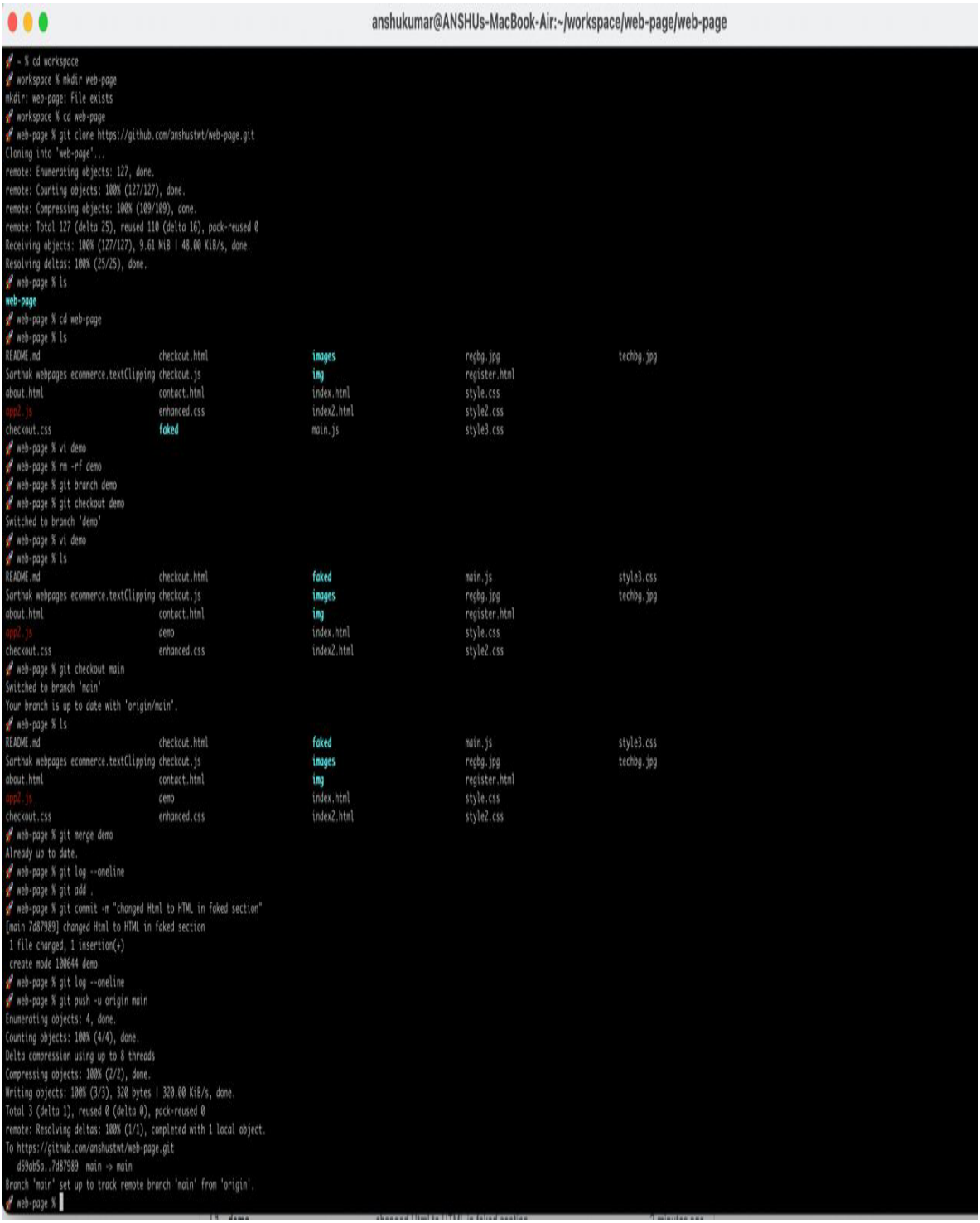
3. Now you will have a copy of the repo you have forked from other user. Now you can do any modification you want without making changes to main source code.



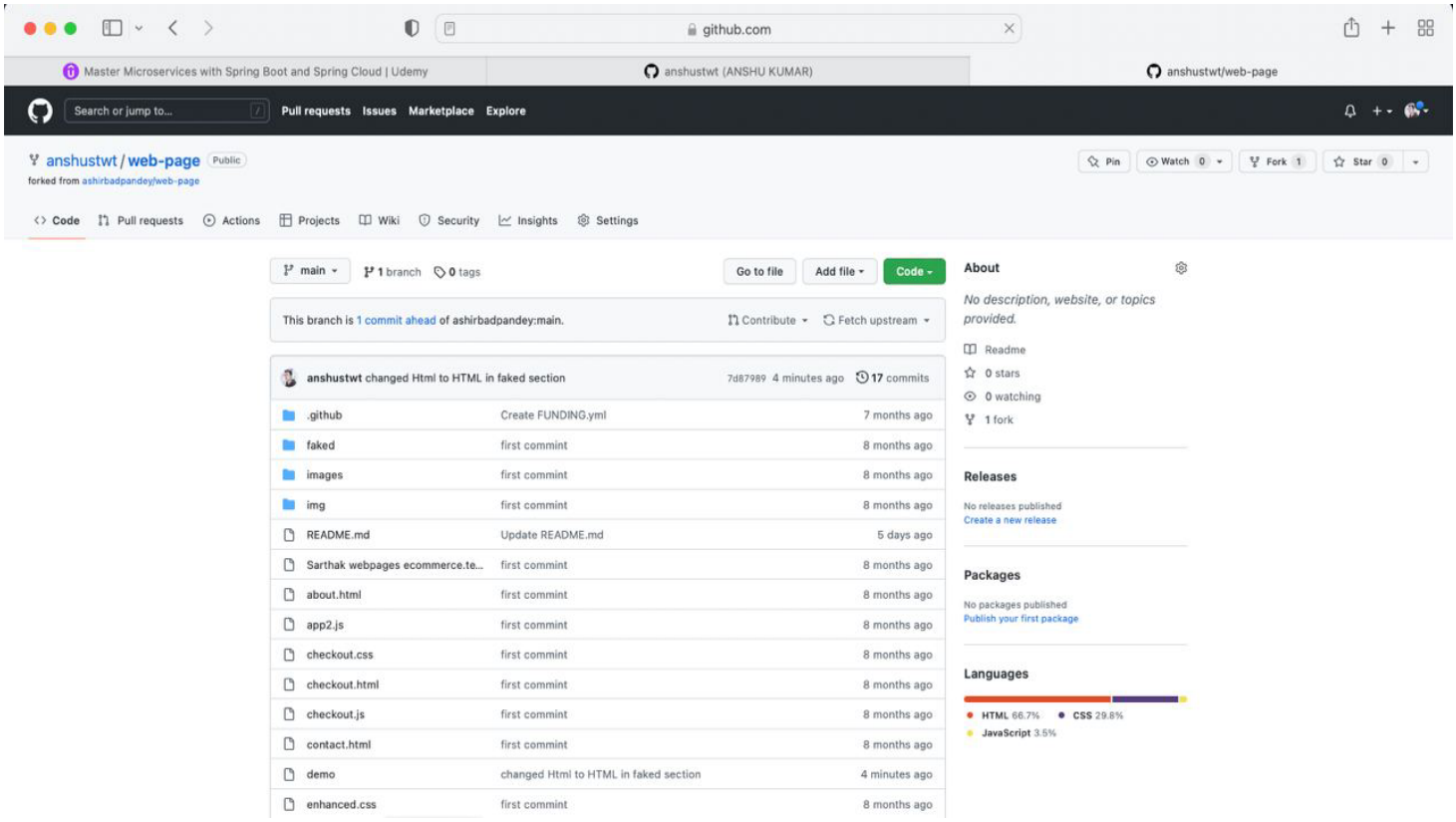
4. Now type https://github.com/VinaySingla1235/pandas-demo-fork.git on CLI.

   Git pull <url> --> This command is used to fetch the remote repo or to clone the repo.

5. Now Open the file make changes in it and commit it and push it to remote.



6. Now you can see your commit in the github the commit is only reflected to your forked repository.


github.com
Master Microservices with Spring Boot and Spring Cloud | Udemy
anshustwt (ANSHU KUMAR)
anshustwt/web-page
Search or jump to...
Pull requests
Issues
Marketplace
Explore
anshustwt / web-page
Public
forked from ashirbadpandey/web-page
Pin
Watch 0
Fork 1
Star 0
Code
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
main
1 branch
0 tags
Go to file
Add file
Code
This branch is 1 commit ahead of ashirbadpandey:main.
Contribute
Fetch upstream
anshustwt changed Html to HTML in faked section
7d87989 4 minutes ago
17 commits
.github Create FUNDING.yml 7 months ago
faked first commint 8 months ago
images first commint 8 months ago
img first commint 8 months ago
README.md Update README.md 5 days ago
Sarthak webpages ecommerce.te... first commint 8 months ago
about.html first commint 8 months ago
app2.js first commint 8 months ago
checkout.css first commint 8 months ago
checkout.html first commint 8 months ago
checkout.js first commint 8 months ago
contact.html first commint 8 months ago
demo changed Html to HTML in faked section 4 minutes ago
enhanced.css first commint 8 months ago
About
No description, website, or topics provided.
Readme
0 stars
0 watching
1 fork
Releases
No releases published
Create a new release
Packages
No packages published
Publish your first package
Languages
HTML 66.7%
CSS 29.8%
JavaScript 3.5%

Experiment No. 08

Aim: Merge and Resolve Conflicts

Theory:

1. Do changes in master branch and commit those change. And checkout to different branch and again do changes and commit it. Now checkout to master branch and merge that branch in master.

COMMIT IN MASTER BRANCH

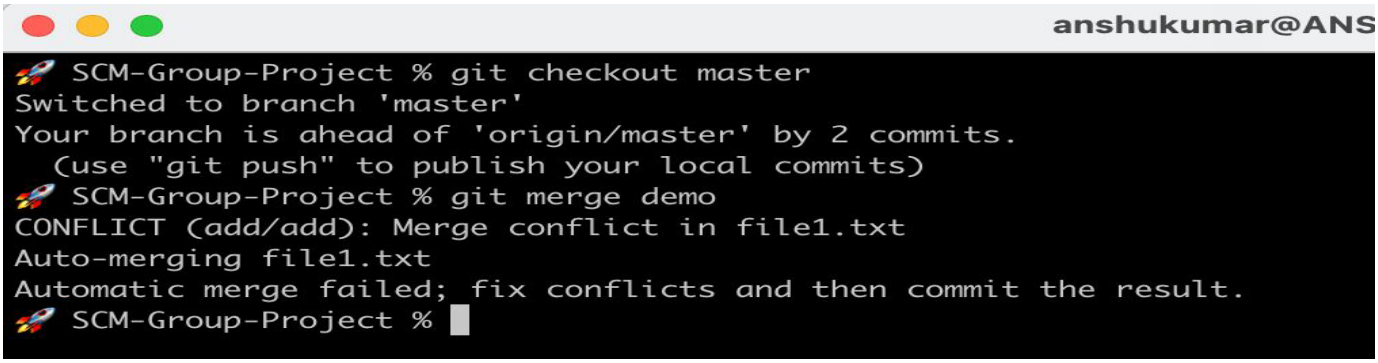
```
🚀 SCM-Group-Project %
🚀 SCM-Group-Project % git branch demo
🚀 SCM-Group-Project % git branch
🚀 SCM-Group-Project % cat >file1.txt
hello, have a good day
🚀 SCM-Group-Project % git add .
🚀 SCM-Group-Project % git commit -m "commit in master branch after creating demo branch"
[master 7dd8994] commit in master branch after creating demo branch
 3 files changed, 1 insertion(+), 1 deletion(-)
 create mode 100644 .DS_Store
 delete mode 100644 demo-file
 create mode 100644 file1.txt
🚀 SCM-Group-Project %
```

COMMIT IN DEMO-BRANCH BRANCH

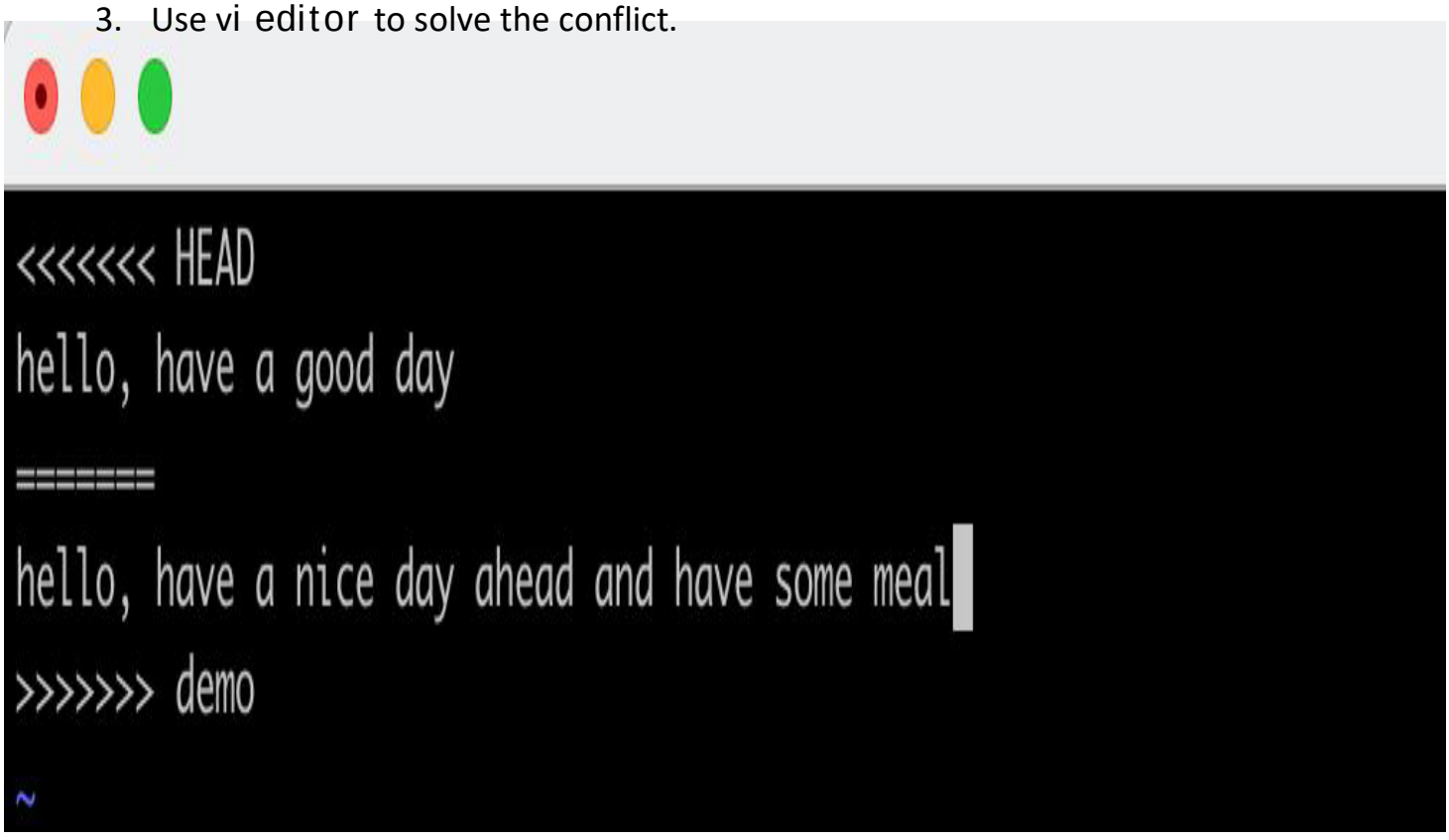
anshukumar@ANSHU

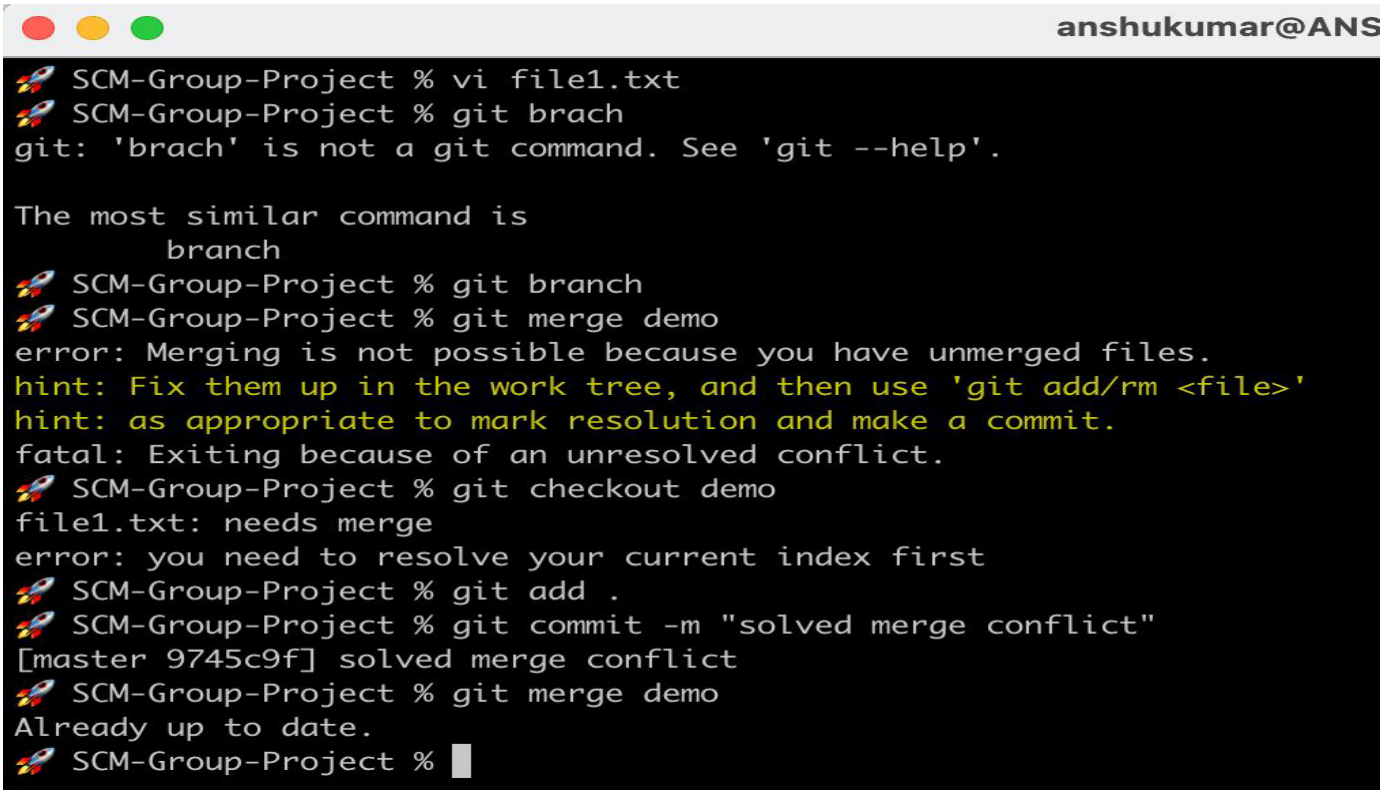
```
🚀 SCM-Group-Project % git checkout demo
Switched to branch 'demo'
🚀 SCM-Group-Project % ls
code-sample demo-file
🚀 SCM-Group-Project % cat >file1.txt
hello, have a nice day ahead
🚀 SCM-Group-Project % rm -rf demo-file
🚀 SCM-Group-Project % ls
code-sample file1.txt
🚀 SCM-Group-Project % git add .
🚀 SCM-Group-Project % git commit -m "commited in demo branch"
[demo 43f4c41] commited in demo branch
 2 files changed, 1 insertion(+), 1 deletion(-)
 delete mode 100644 demo-file
 create mode 100644 file1.txt
🚀 SCM-Group-Project % git status
On branch demo
nothing to commit, working tree clean
🚀 SCM-Group-Project %
```

2. Now try to merge it will give Conflicts Error.



3. Use vi editor to solve the conflict.




anshukumar@ANS
SCM-Group-Project % vi file1.txt
SCM-Group-Project % git brach
git: 'brach' is not a git command. See 'git --help'.
The most similar command is
branch
SCM-Group-Project % git branch
SCM-Group-Project % git merge demo
error: Merging is not possible because you have unmerged files.
hint: Fix them up in the work tree, and then use 'git add/rm <file>'
hint: as appropriate to mark resolution and make a commit.
fatal: Exiting because of an unresolved conflict.
SCM-Group-Project % git checkout demo
file1.txt: needs merge
error: you need to resolve your current index first
SCM-Group-Project % git add .
SCM-Group-Project % git commit -m "solved merge conflict"
[master 9745c9f] solved merge conflict
SCM-Group-Project % git merge demo
Already up to date.
SCM-Group-Project %

Experiment No. 09

Aim: Reset and Revert

Theory:

Git-revert – Revert some existing commits.

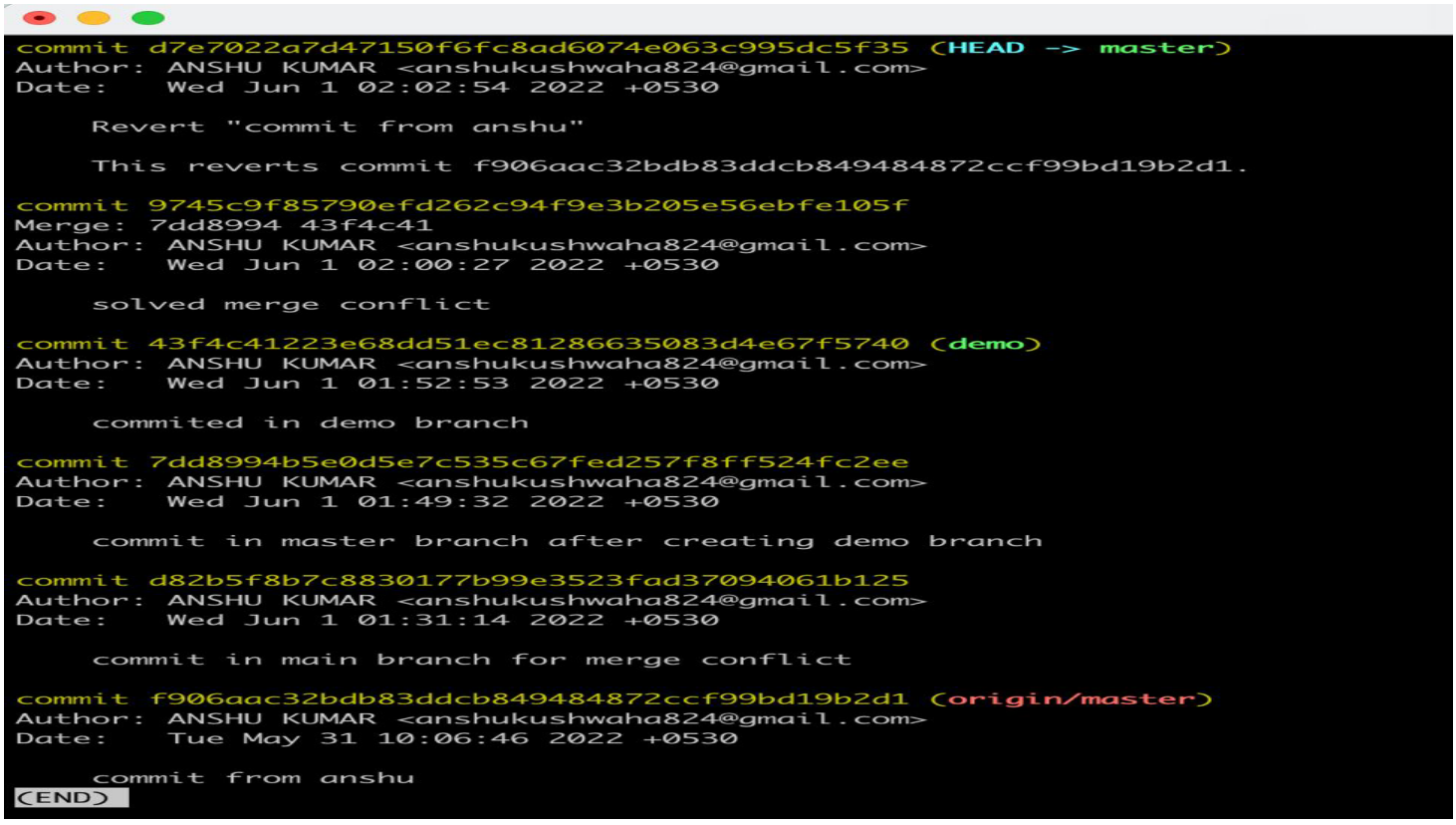
1. On GitBash CLI, Type command “git revert <Commit id>”. It revert the changes that done before Commit.
    - **Git log before revert**

```
commit 9745c9f85790efd262c94f9e3b205e56ebfe105f (HEAD -> master)
Merge: 7dd8994 43f4c41
Author: ANSHU KUMAR <anshukushwaha824@gmail.com>
Date:   Wed Jun 1 02:00:27 2022 +0530

    solved merge conflict

commit 43f4c41223e68dd51ec81286635083d4e67f5740 (demo)
Author: ANSHU KUMAR <anshukushwaha824@gmail.com>
Date:   Wed Jun 1 01:52:53 2022 +0530

    commited in demo branch

commit 7dd8994b5e0d5e7c535c67fed257f8ff524fc2ee
Author: ANSHU KUMAR <anshukushwaha824@gmail.com>
Date:   Wed Jun 1 01:49:32 2022 +0530

    commit in master branch after creating demo branch

commit d82b5f8b7c8830177b99e3523fad37094061b125
Author: ANSHU KUMAR <anshukushwaha824@gmail.com>
Date:   Wed Jun 1 01:31:14 2022 +0530

    commit in main branch for merge conflict

commit f906aac32bdb83ddcb849484872ccf99bd19b2d1 (origin/master)
Author: ANSHU KUMAR <anshukushwaha824@gmail.com>
Date:   Tue May 31 10:06:46 2022 +0530

    commit from anshu
(END)
```

- **Revert the changes specified by that commit in HEAD and create a new commit with the reverted changes**

```
anshukumar@ANSHUs
🚀 SCM-Group-Project % git revert f906aac32bdb83ddcb849484872ccf99bd19b2d1
Removing code-sample
[master d7e7022] Revert "commit from anshu"
 1 file changed, 32 deletions(-)
 delete mode 100644 code-sample
🚀 SCM-Group-Project %
```

- **Git log after Revert**

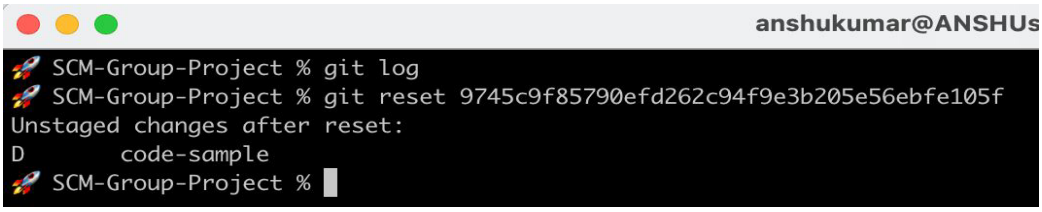


Git-reset - Reset current HEAD to the specified state

2. At a surface level, git reset is similar in behavior to git checkout. Where git checkout solely operates on the HEAD ref pointer, git reset will move the HEAD ref pointer and the current branch ref pointer. To better demonstrate this behaviour consider the following example. This example demonstrates a sequence of commits on the main branch. The HEAD ref and main branch ref currently point to commit d. Now let us execute and compare, both git checkout b and git reset b.

GIT RESET:

1. reset is the command we use when we want to move the repository back to a previous commit, discarding any changes made after that commit.



- **Git log after reset**