SOURCE CODE MANAGEMENT

Practical File

Submitted by

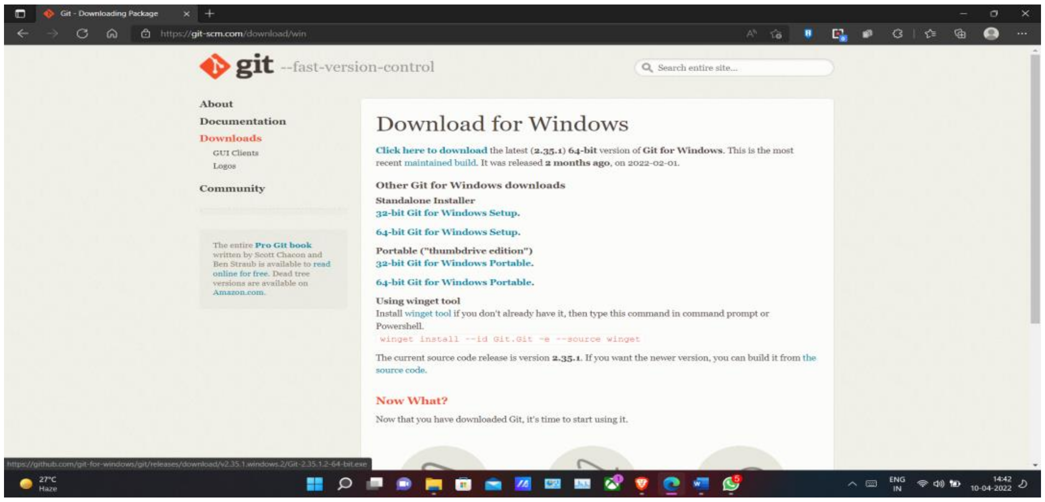
Name: Chayank Das

Roll no.: 2110990388

Task 1.1

Installing Git on windows:

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



The entire Pro Git book written by Scott Chacon and Ben Straub is available to read online for free. Dead tree versions are available on Amazon.com.

Download for Windows

Click here to download the latest (2.35.1) 64-bit version of Git for Windows. This is the most recent maintained build. It was released 2 months ago, on 2022-02-01.

Other Git for Windows downloads

Standalone Installer
32-bit Git for Windows Setup.

64-bit Git for Windows Setup.

Portable ("thumbdrive edition")
32-bit Git for Windows Portable.

64-bit Git for Windows Portable.

Using winget tool
Install winget tool if you don't already have it, then type this command in command prompt or Powershell.

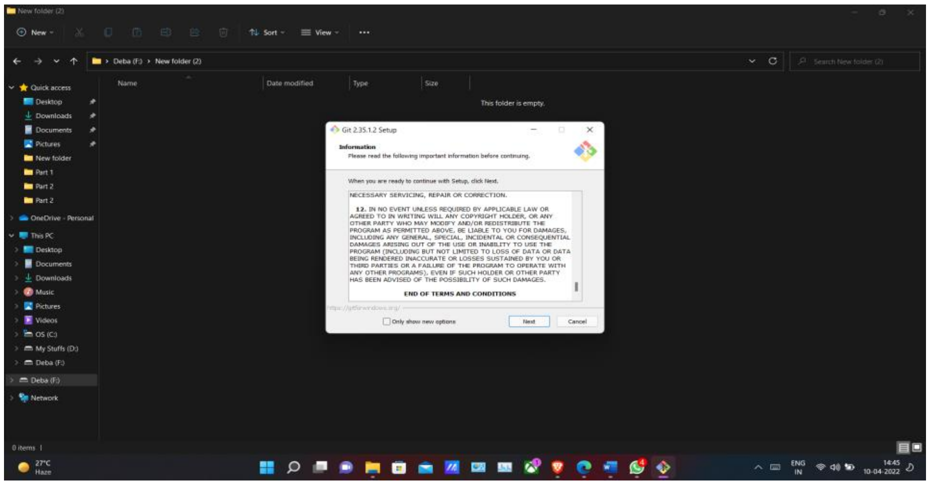
```
winget install --id Git.Git -e --source winget
```

The current source code release is version 2.35.1. If you want the newer version, you can build it from the source code.

Now What?

Now that you have downloaded Git, it's time to start using it.

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**.



Please read the following important information before continuing.

When you are ready to continue with Setup, click Next.

NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

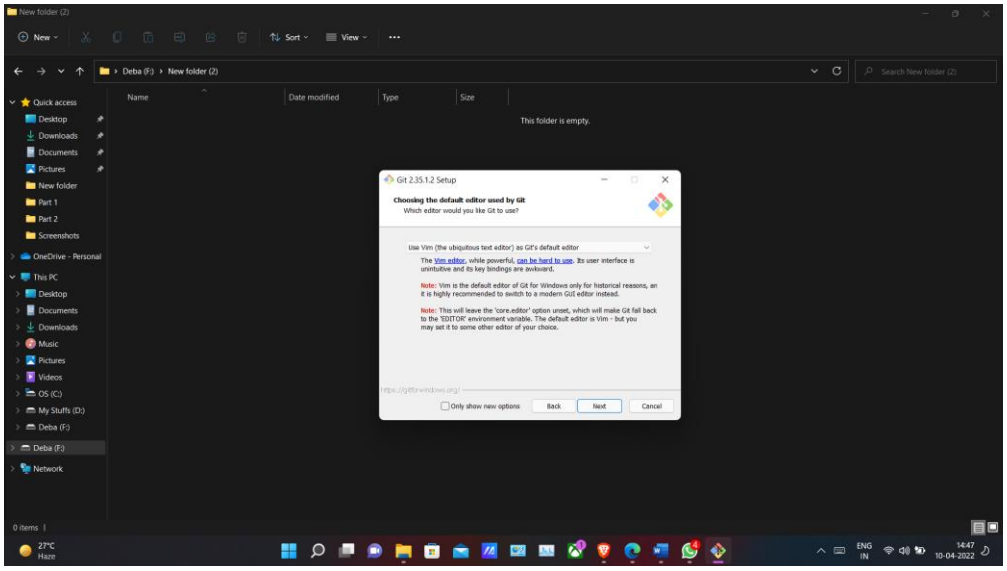
END OF TERMS AND CONDITIONS

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**.



Choosing the default editor used by Git

Which editor would you like Git to use?

Use Vim (the ubiquitous text editor) as Git's default editor

The Vim editor, while powerful, can be hard to use. Its user interface is unintuitive and its key bindings are awkward.

Note: Vim is the default editor of Git for Windows only for historical reasons, an it is highly recommended to switch to a modern GUI editor instead.

Note: This will leave the 'core.editor' option unset, which will make Git fall back to the 'EDITOR' environment variable. The default editor is Vim - but you may set it to some other editor of your choice.

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**.

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.**

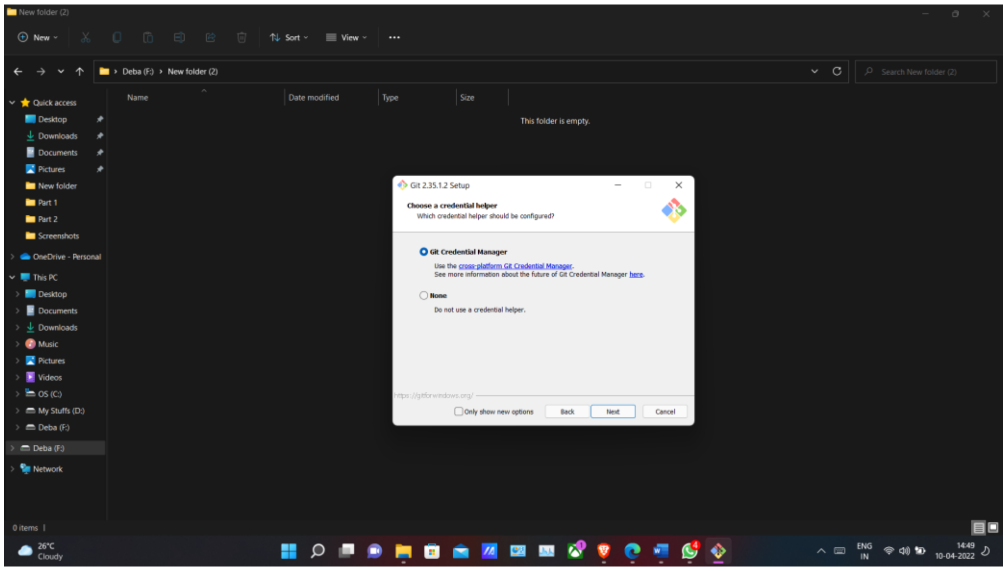
13. The next option relates to server certificates. Most users should use the default. If you're working in an Active Directory environment, you may need to switch to Windows Store certificates. Click **Next**.

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 behaviour. 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**.



Git 2.35.1.2 Setup

Choose a credential helper
Which credential helper should be configured?

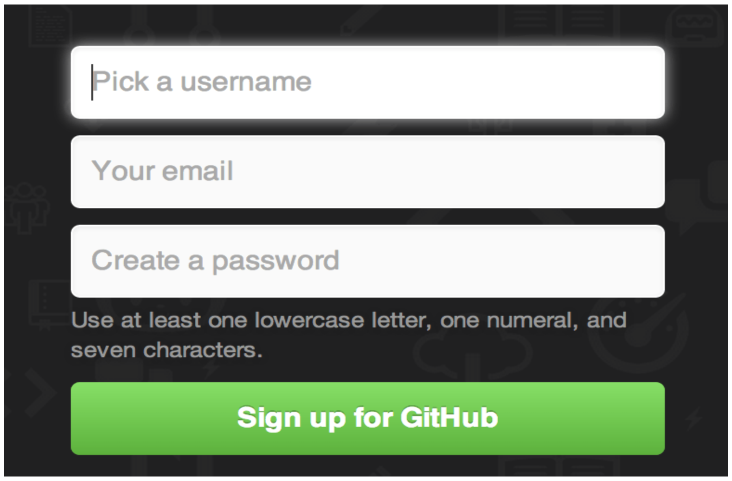
[✓ Git Credential Manager]
Use the cross-platform Git Credential Manager.
See more information about the future of Git Credential Manager here.

None
Do not use a credential helper.

https://gitforwindows.org/

Setting up your github account:

The first thing you need to do is set up a free user account. Simply visit https://github.com, choose a user name that isn't already taken, provide an email address and a password, and click the big green “Sign up for GitHub” button.

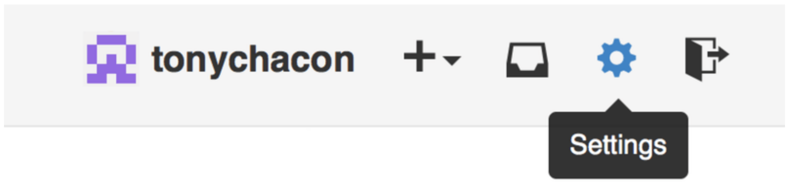

Pick a username
Your email
Create a password
Use at least one lowercase letter, one numeral, and seven characters.
Sign up for GitHub

The next thing you’ll see is the pricing page for upgraded plans, but it’s safe to ignore this for now. GitHub will send you an email to verify the address you provided. Go ahead and do this; it’s pretty important (as we’ll see later).

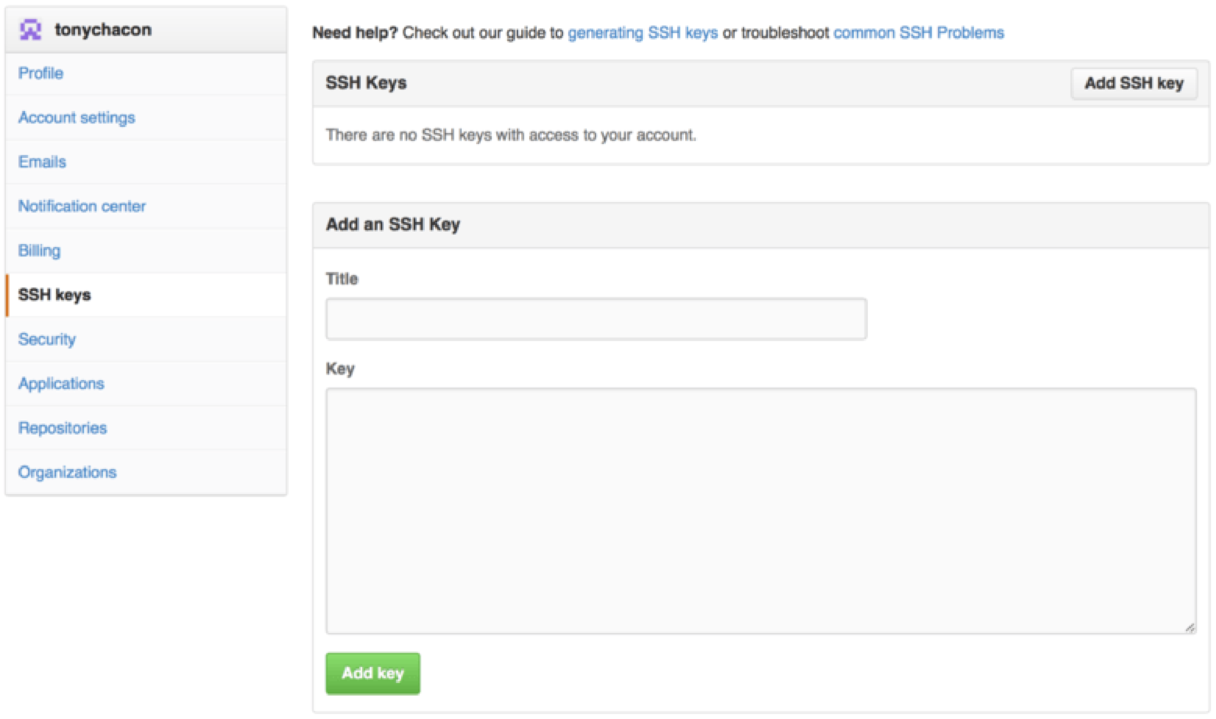
Clicking the Octocat logo at the top-left of the screen will take you to your dashboard page. You’re now ready to use GitHub.

SSH Access

As of right now, you’re fully able to connect with Git repositories using the `https://` protocol, authenticating with the username and password you just set up. However, to simply clone public projects, you don’t even need to sign up - the account we just created comes into play when we fork projects and push to our forks a bit later.
If you’d like to use SSH remotes, you’ll need to configure a public key. If you don’t already have one, see Generating Your SSH Public Key. Open up your account settings using the link at the top-right of the window:



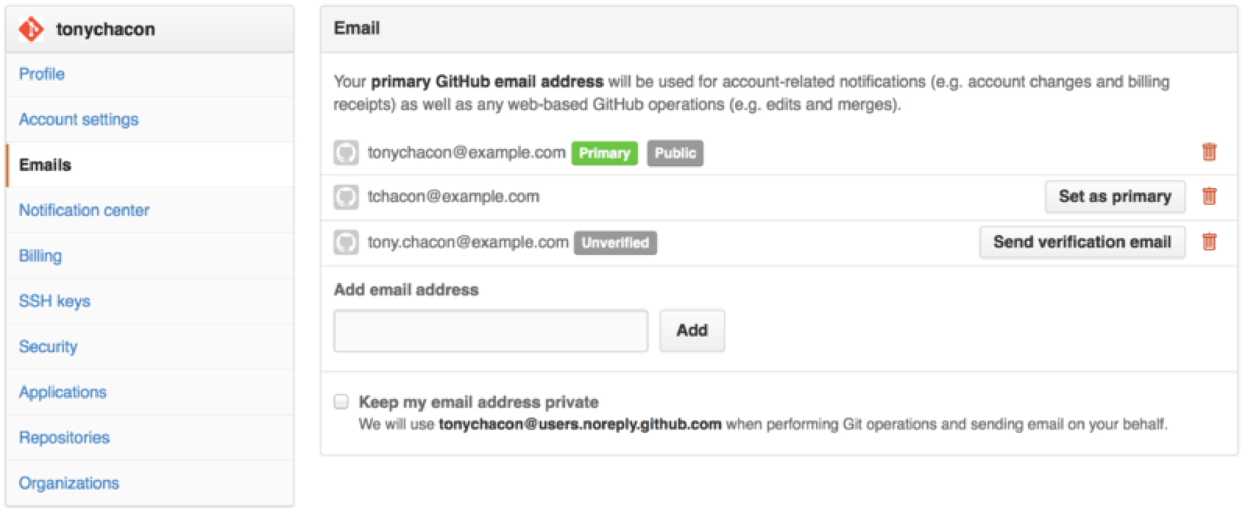
Then select the “SSH keys” section along the left-hand side.



From there, click the “Add an SSH key” button, give your key a name, paste the contents of your `~/.ssh/id_rsa.pub` (or whatever you named it) public-key file into the text area, and click “Add key”.

Your Email Addresses

The way that GitHub maps your Git commits to your user is by email address. If you use multiple email addresses in your commits and you want GitHub to link them up properly, you need to add all the email addresses you have used to the Emails section of the admin section.


tonychacon
Profile
Account settings
Emails
Notification center
Billing
SSH keys
Security
Applications
Repositories
Organizations

Email

Your **primary GitHub email address** will be used for account-related notifications (e.g. account changes and billing receipts) as well as any web-based GitHub operations (e.g. edits and merges).

tonychacon@example.com Primary Public
tchacon@example.com Set as primary
tony.chacon@example.com Unverified Send verification email

Add email address
Add

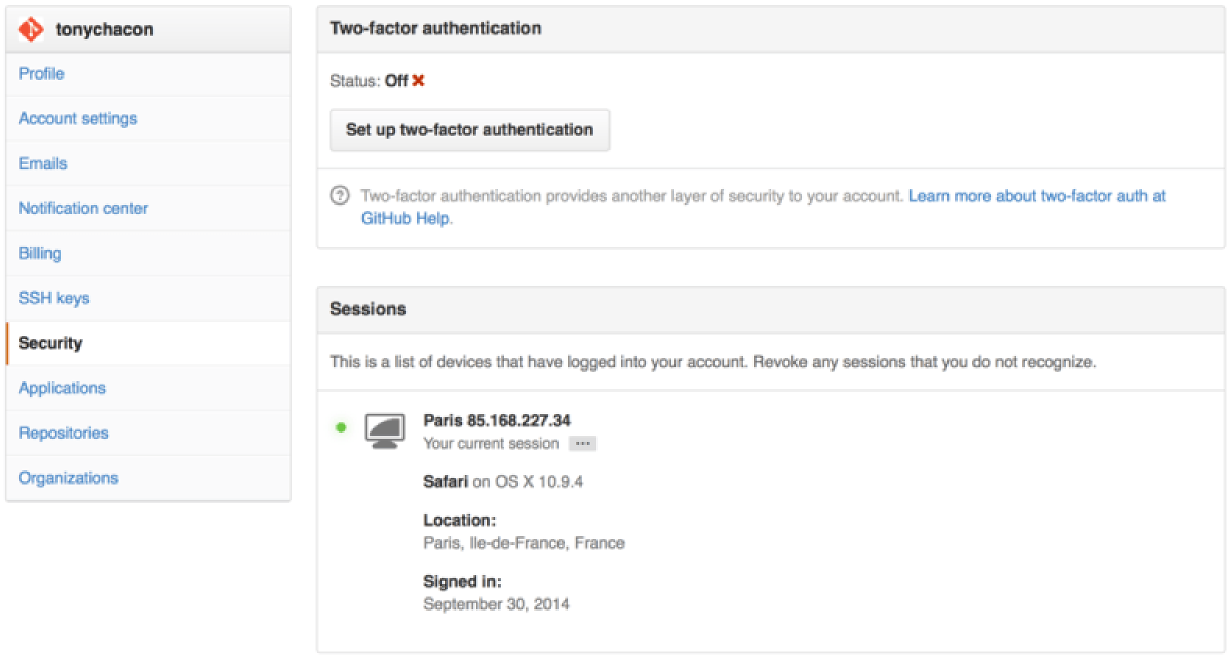
Keep my email address private
We will use **tonychacon@users.noreply.github.com** when performing Git operations and sending email on your behalf.

In Add email addresses we can see some of the different states that are possible. The top address is verified and set as the primary address, meaning that is where you'll get any notifications and receipts. The second address is verified and so can be set as the primary if you wish to switch them. The final address is unverified, meaning that you can't make it your primary address. If GitHub sees any of these in commit messages in any repository on the site, it will be linked to your user now.

Two Factor Authentication

Finally, for extra security, you should definitely set up Two-factor Authentication or "2FA". Two-factor Authentication is an authentication mechanism that is becoming more and more popular recently to mitigate the risk of your account being compromised if your password is stolen somehow. Turning it on will make GitHub ask you for two different methods of authentication, so that if one of them is compromised, an attacker will not be able to access your account.

You can find the Two-factor Authentication setup under the Security tab of your Account settings.



Two-factor authentication

Status: Off ✗

Set up two-factor authentication

Two-factor authentication provides another layer of security to your account. Learn more about two-factor auth at GitHub Help.

Sessions

This is a list of devices that have logged into your account. Revoke any sessions that you do not recognize.

Paris 85.168.227.34
Your current session

Safari on OS X 10.9.4

Location:
Paris, Ile-de-France, France

Signed in:
September 30, 2014

If you click on the “Set up two-factor authentication” button, it will take you to a configuration page where you can choose to use a phone app to generate your secondary code (a “time based one-time password”), or you can have GitHub send you a code via SMS each time you need to log in.

After you choose which method you prefer and follow the instructions for setting up 2FA, your account will then be a little more secure and you will have to provide a code in addition to your password whenever you log into GitHub.

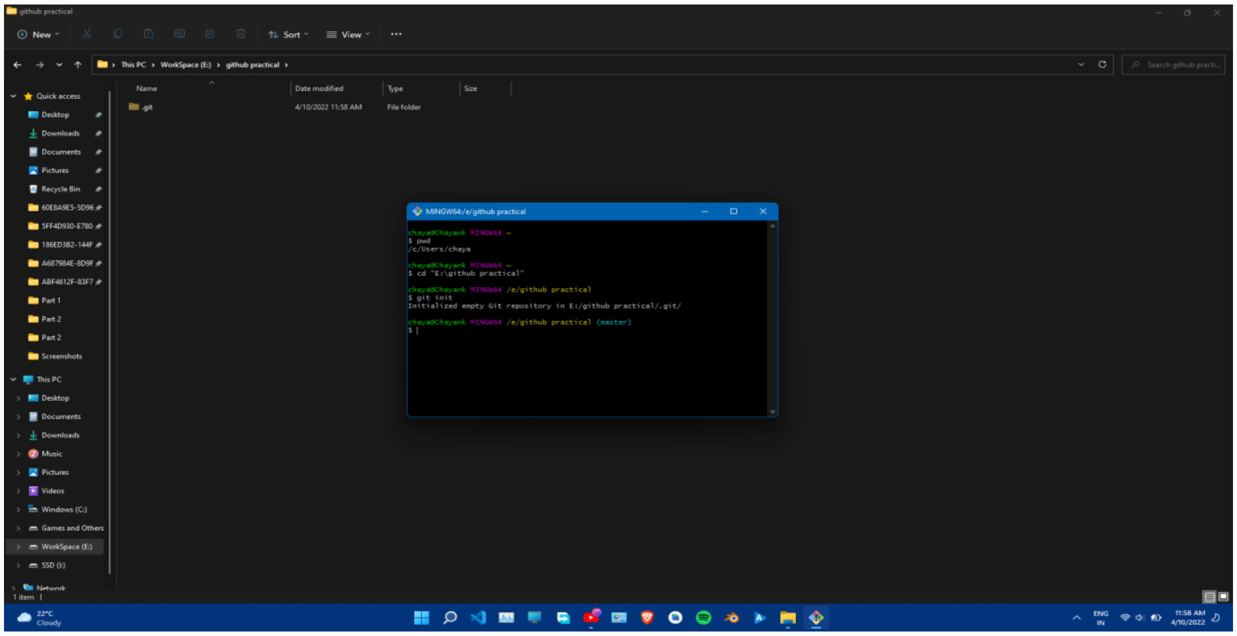
Creating a repository:

You typically obtain a Git repository in one of two ways:

1.You can take a local directory that is currently not under version control, and turn it into a Git repository, or

2.You can clone an existing Git repository from elsewhere.

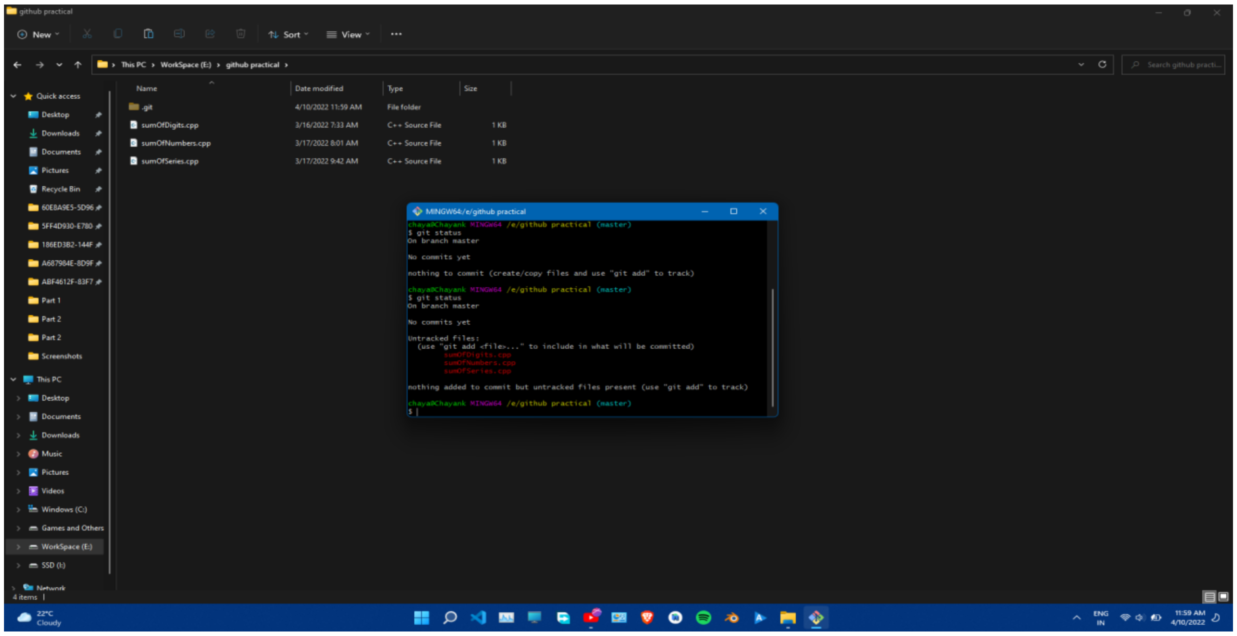
A project directory that is currently not under version control and we want to start controlling it with Git, we first need to go to that project's directory. And we need to execute the command (git init) This creates a new subdirectory named .git that contains all of your necessary repository files — a Git repository skeleton.



```
chaya@Chayank MINGW64 ~
$ pwd
/c/Users/chaya

chaya@Chayank MINGW64 ~
$ cd "E:\github practical"

chaya@Chayank MINGW64 /e/github practical
$ git init
Initialized empty Git repository in E:/github practical/.git/

chaya@Chayank MINGW64 /e/github practical (master)
$ |
```

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. Its command is (git status)



Name	Date modified	Type	Size
.git	4/10/2022 11:59 AM	File folder	
sumOfDigits.cpp	3/16/2022 7:33 AM	C++ Source File	1 KB
sumOfNumbers.cpp	3/17/2022 8:01 AM	C++ Source File	1 KB
sumOfSeries.cpp	3/17/2022 9:42 AM	C++ Source File	1 KB

```
chaya@Chayank MINGW64 /e/github practical (master)
$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)

chaya@Chayank MINGW64 /e/github practical (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        sumOfDigits.cpp
        sumOfNumbers.cpp
        sumOfSeries.cpp

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

chaya@Chayank MINGW64 /e/github practical (master)
$ |
```

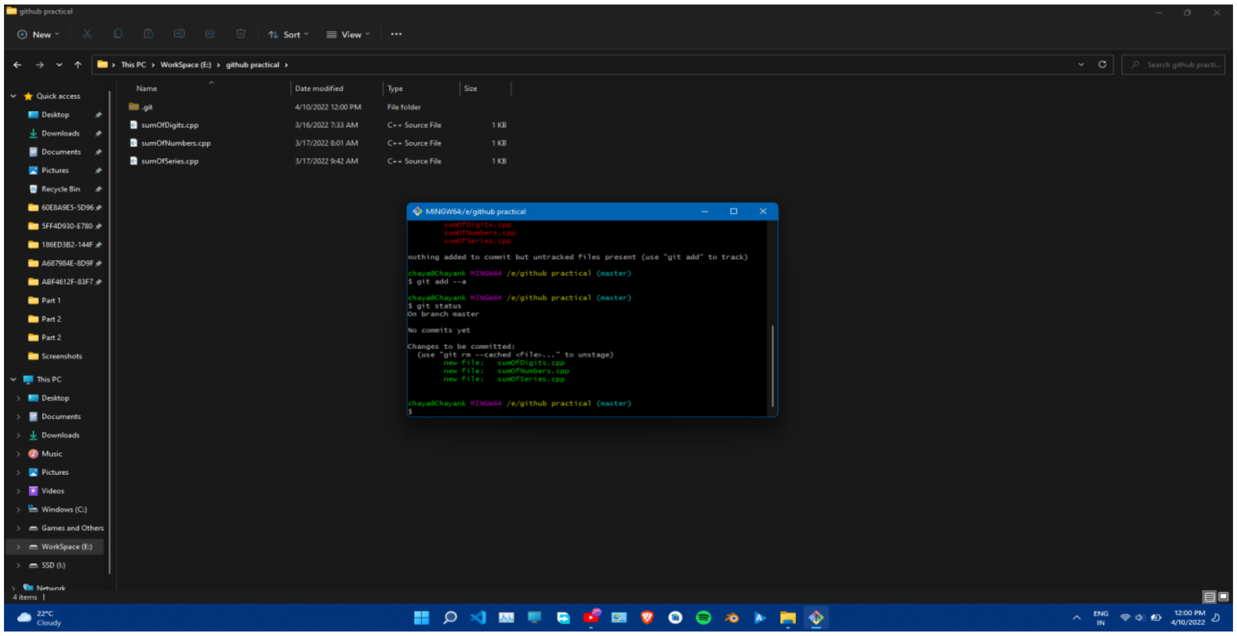
.

Git add:

1.To begin tracking a file, first we need to stage the file, to do so we use the command. (git add “name of the file”)

2.To check if your file is being tracked and staged to be committed, we use the (git status) command.

3.To ensure that the file is being tracked, the file name will appear in green text, and files that are not tracked will appear in red text.



```
        sumOfDigits.cpp
        sumOfNumbers.cpp
        sumOfSeries.cpp

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

chaya@Chayank MINGW64 /e/github practical (master)
$ git add --a

chaya@Chayank MINGW64 /e/github practical (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   sumOfDigits.cpp
        new file:   sumOfNumbers.cpp
        new file:   sumOfSeries.cpp


chaya@Chayank MINGW64 /e/github practical (master)
$
```

Staging modified files:

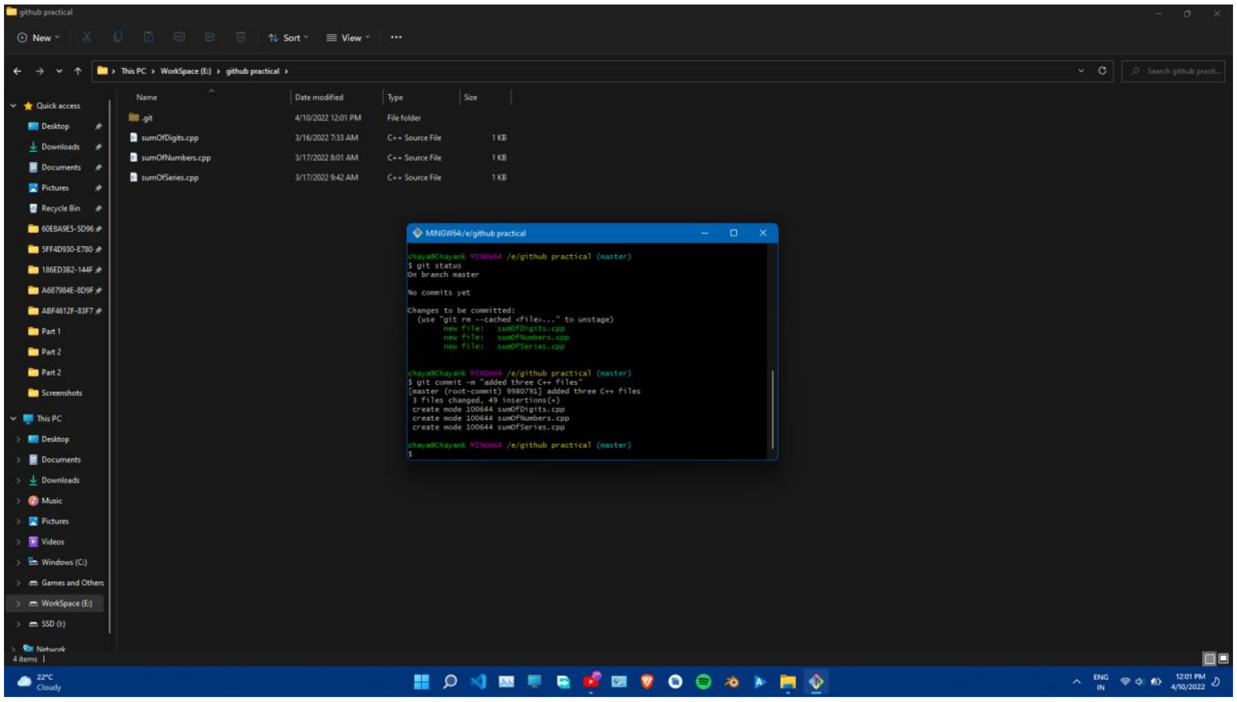
If we modify a file that is already being tracked, and run the (git status) command,

the file will appear under a section named “Changes not staged for commit” which means that the file that is tracked has been modified in the working directory but not yet staged. To stage it, we run the (git add) command.

If we make changes after staging a file, and run git status, the file will again appear in the “Changes not staged for commit” section, even after staging, so to prevent it we need to stage the file after every changes we made to it.

Committing Your Changes:

Now that our staging area is set up the way we want it, we can commit our changes. Remember that anything that is still unstaged — any files we have created or modified that we haven’t run `git add` on since we edited them — won’t go into this commit. They will stay as modified files on our disk. In this case, let’s say that the last time we ran `git status`, we saw that everything was staged, so we are ready to commit our changes. The simplest way to commit is to type `git commit`.

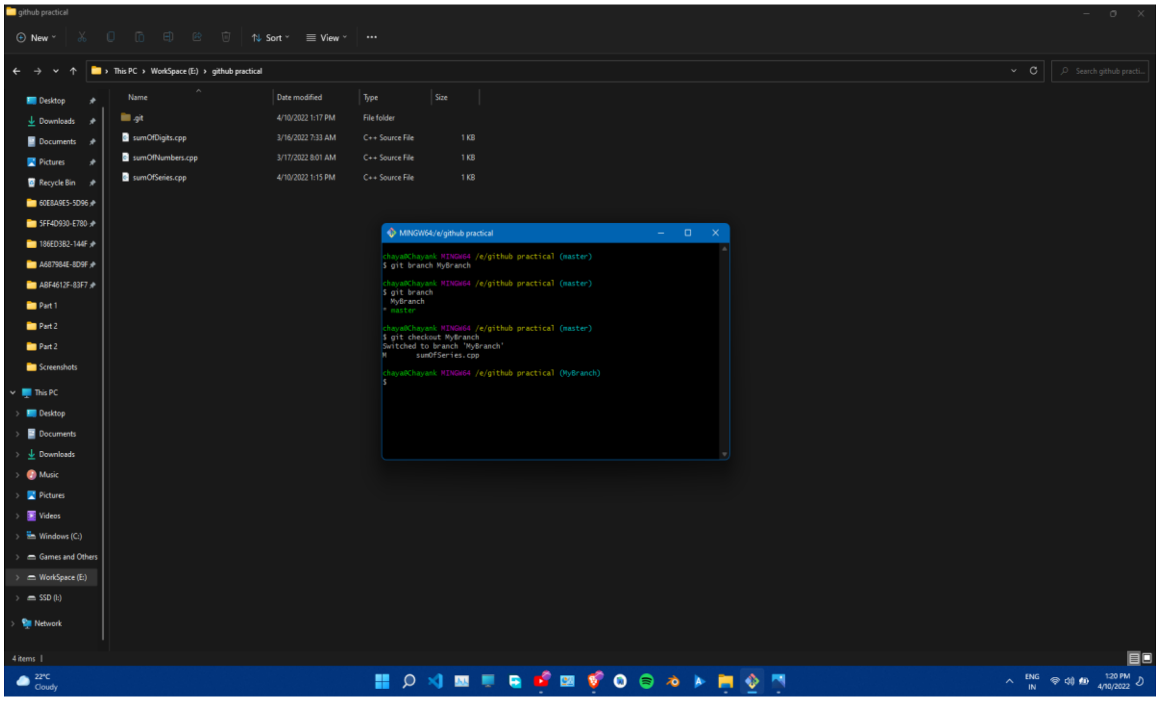


```
chaya@Chayank MINGW64 /e/github practical (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   sumOfDigits.cpp
        new file:   sumOfNumbers.cpp
        new file:   sumOfSeries.cpp


chaya@Chayank MINGW64 /e/github practical (master)
$ git commit -m "added three C++ files"
[master (root-commit) 9980791] added three C++ files
 3 files changed, 49 insertions(+)
 create mode 100644 sumOfDigits.cpp
 create mode 100644 sumOfNumbers.cpp
 create mode 100644 sumOfSeries.cpp

chaya@Chayank MINGW64 /e/github practical (master)
$
```

Create Branches in git:

The main branch in git is called master branch. But we can make branches out of this main master branch. All the files present in master can be shown in branch but the files which are created in branch are not shown in master branch. We also can merge both the parent(master) and child (other branches).

1. For creating a new branch: git branch “name of branch”
2. To check how many branches we have : git branch
3. To change the present working branch: git checkout “name of the branch”



```
chaya@Chayank MINGW64 /e/github practical (master)
$ git branch MyBranch

chaya@Chayank MINGW64 /e/github practical (master)
$ git branch
  MyBranch
* master

chaya@Chayank MINGW64 /e/github practical (master)
$ git checkout MyBranch
Switched to branch 'MyBranch'
M       sumOfSeries.cpp

chaya@Chayank MINGW64 /e/github practical (MyBranch)
$
```

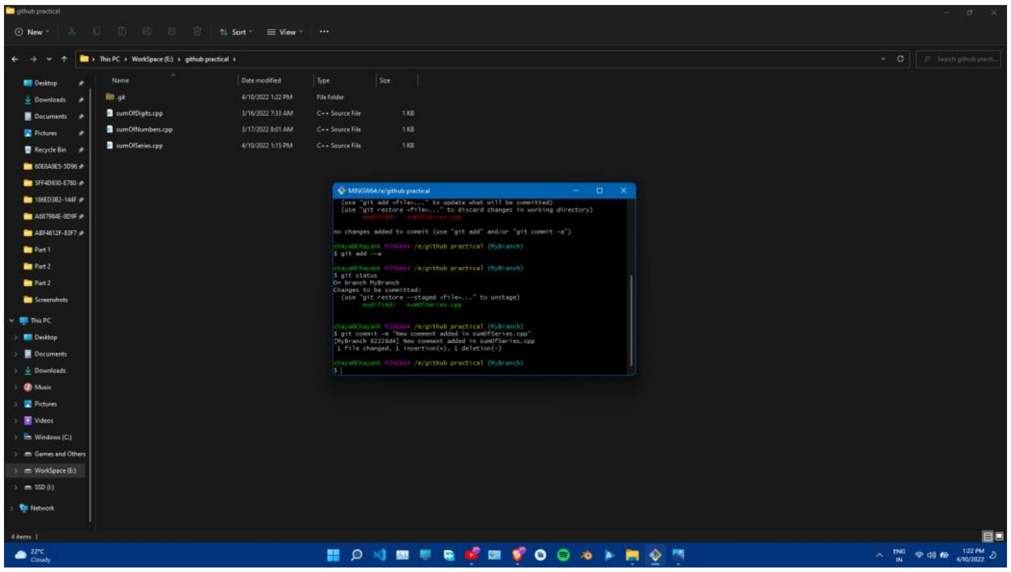
Visualizing Branches:

To visualize, suppose we have to create a new file “hello” in the new branch “MyBranch” instead of the master branch. After this we have to do three step i.e. working directory, staging area and git repository.

After this, we have completed the 3 step process which is tracking the file, send it to stagging area and commit in the repo. Finally we can rollback to any previously saved version of this file.

Now, we will change the branch from MyBranch to master, but when we switch back to master branch the file we created i.e “hello” will not be there. Hence the new file will not be shown in the master branch. In this way we can create and change different branches. We can also merge the branches by using the git merge command to merge the other branch with the master branch. In this way we can create and change different branches.

Committing in the New Branch:

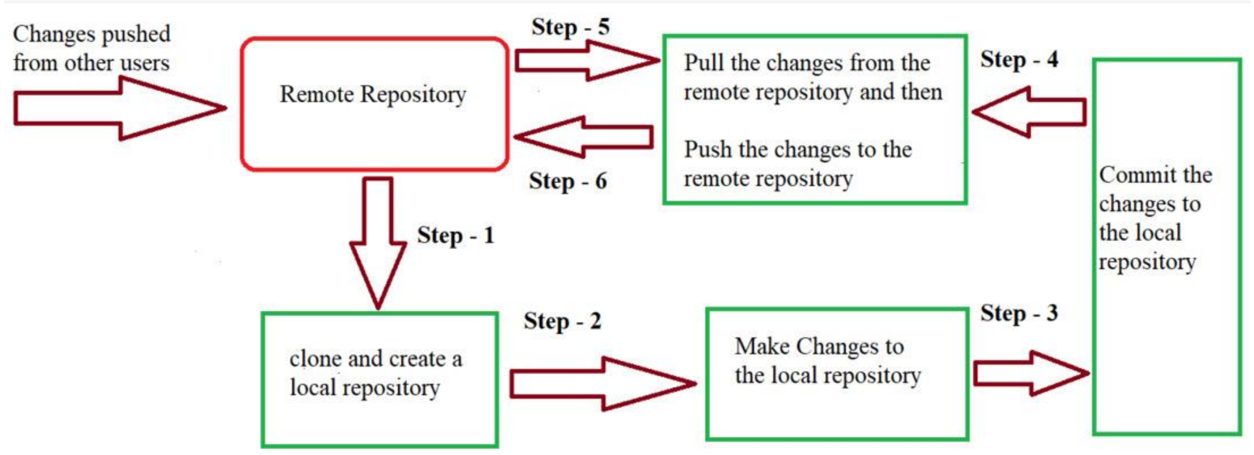


```
Name              Date modified        Type             Size
.git              4/10/2022 1:22 PM    File folder
sumOfDigits.cpp   3/16/2022 7:33 AM    C++ Source File  1 KB
sumOfNumbers.cpp  3/17/2022 8:01 AM    C++ Source File  1 KB
sumOfSeries.cpp   4/10/2022 1:15 PM    C++ Source File  1 KB
```

```
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   sumOfSeries.cpp

no changes added to commit (use "git add" and/or "git commit -a")

chaya@Chayank MINGW64 /e/github practical (MyBranch)
$ git add --a

chaya@Chayank MINGW64 /e/github practical (MyBranch)
$ git status
On branch MyBranch
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   sumOfSeries.cpp


chaya@Chayank MINGW64 /e/github practical (MyBranch)
$ git commit -m "New comment added in sumOfSeries.cpp"
[MyBranch 82228d4] New comment added in sumOfSeries.cpp
 1 file changed, 1 insertion(+), 1 deletion(-)

chaya@Chayank MINGW64 /e/github practical (MyBranch)
$ 
```

Git lifecycle:

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-


Changes pushed from other users
Remote Repository
Step - 5
Pull the changes from the remote repository and then
Push the changes to the remote repository
Step - 6
Step - 4
Commit the changes to the local repository
Step - 1
Step - 2
clone and create a local repository
Make Changes to the local repository
Step - 3

• Step 1- We first clone any of the code residing in the remote repository to make our won local repository.

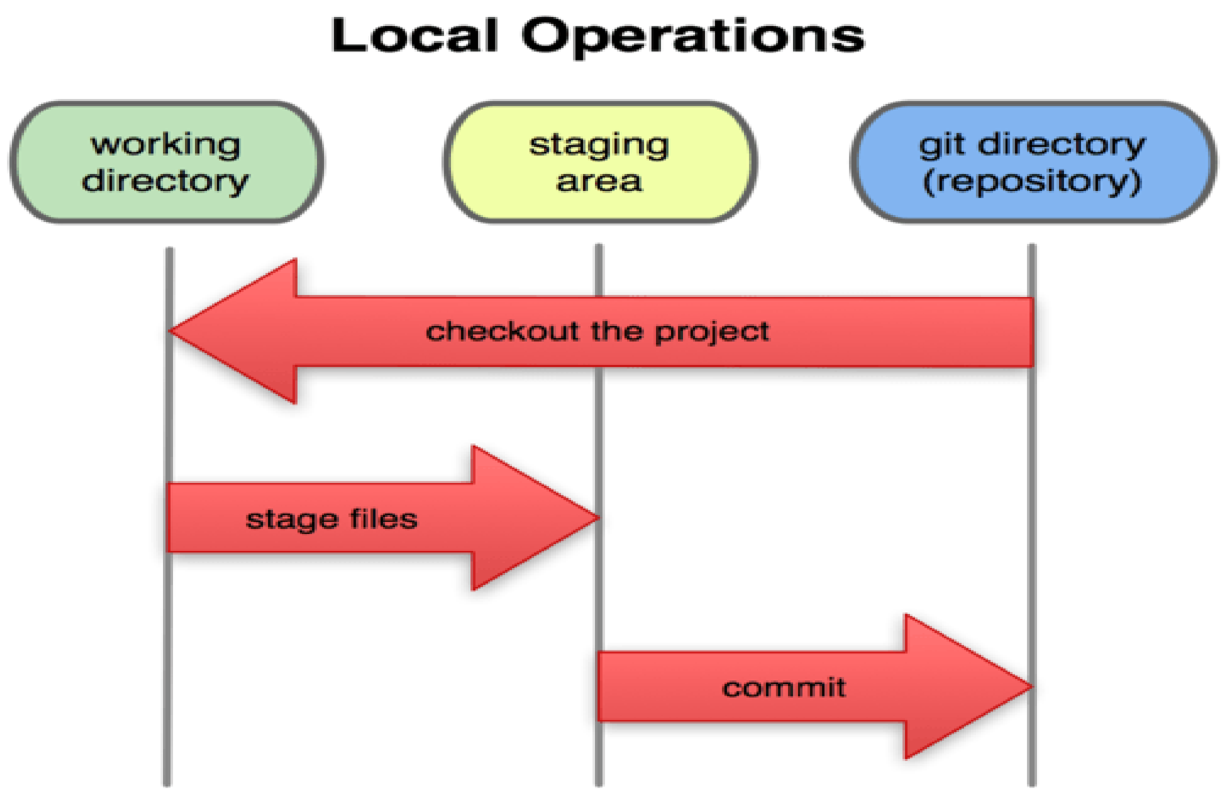
• Step 2- We edit the files that we have cloned in our local repository and make the necessary changes in it.

• Step 3- We commit our changes by first adding them to our staging area and committing them with a commit message.

• Step 4 and Step 5- We first check whether there are any of the changes done in the remote repository by some other users and we first pull that changes.

• Step 6- If there are no changes we push our changes to the remote repository and we are done with our work.

When a directory is made a git repository, there are mainly 3 states which make the essence of Git version Control System. The three states are:



1. Working Directory

Whenever we want to initialize aur local project directory to make a Git repository, we use the git init command. After this command, git becomes aware of the files in the project although it does not track the files yet. The files are further tracked in the staging area.

2. Staging Area

Now, to track files the different versions of our files we use the command git add. We can term a staging area as a place where different versions of our files are stored. git add command copies the version of your file from your working directory to the staging area. We can, however, choose which files we need to add to the staging area because in our working directory there are some files that we don't want to get tracked, examples include node modules, temporary files, etc. Indexing in Git is the one that helps Git in understanding which files need to be added or sent. You can find your staging area in the .git folder inside the index file. git add git add.

3. Git Directory

Now since we have all the files that are to be tracked and are ready in the staging area, we are ready to commit aur files using the git commit command. Commit helps us in keeping the track of the metadata of the files in our staging area. We specify every commit with a message which tells what the commit is about. Git preserves the information or the metadata of the files that were committed in a Git Directory which helps Git in tracking files basically it preserves the photocopy of the committed files. Commit also stores the name of the author who did the commit, files that are committed, and the date at which they are committed along with the commit message. git commit -m