

SOURCE CODE MANAGEMENT

Practical File



Submitted by

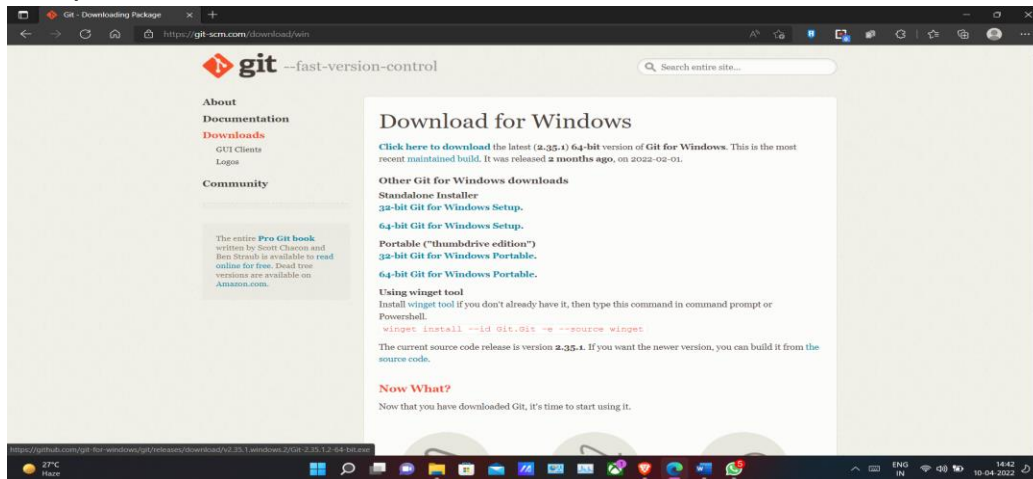
Name: Chayank Das

Roll no.: 2110990388

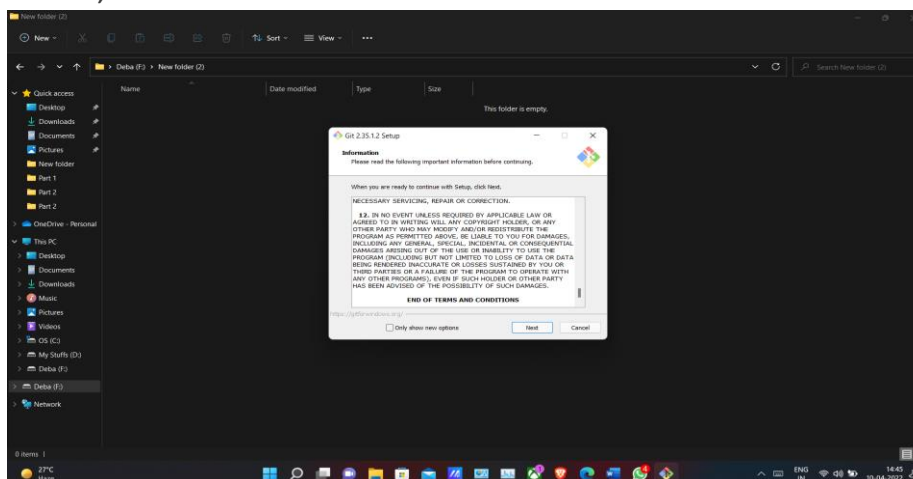
Task 1.1

Installing Git on windows:

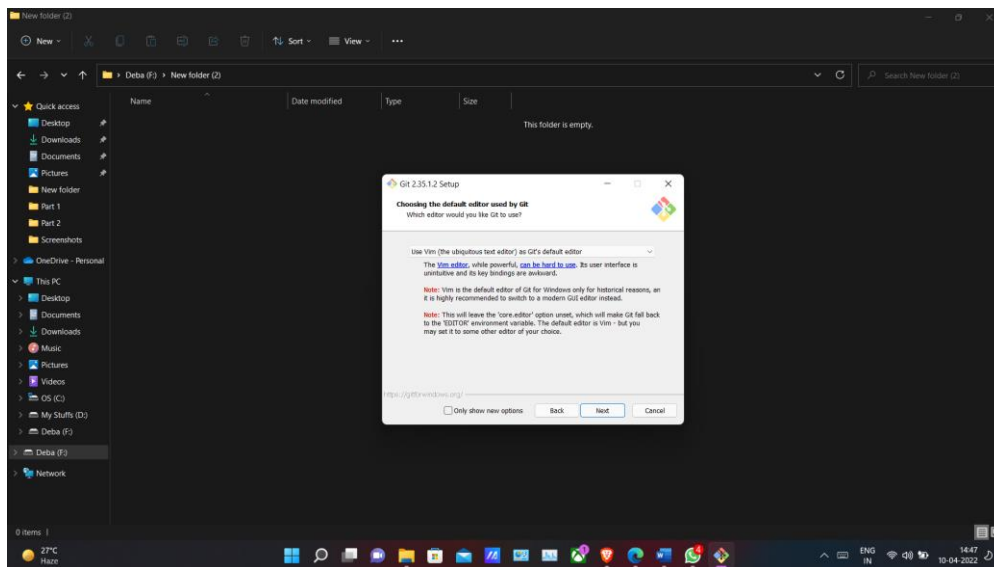
1. Browse to the official Git website: <https://git-scm.com/downloads>
2. Click the download link for Windows and allow the download to complete.



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**.

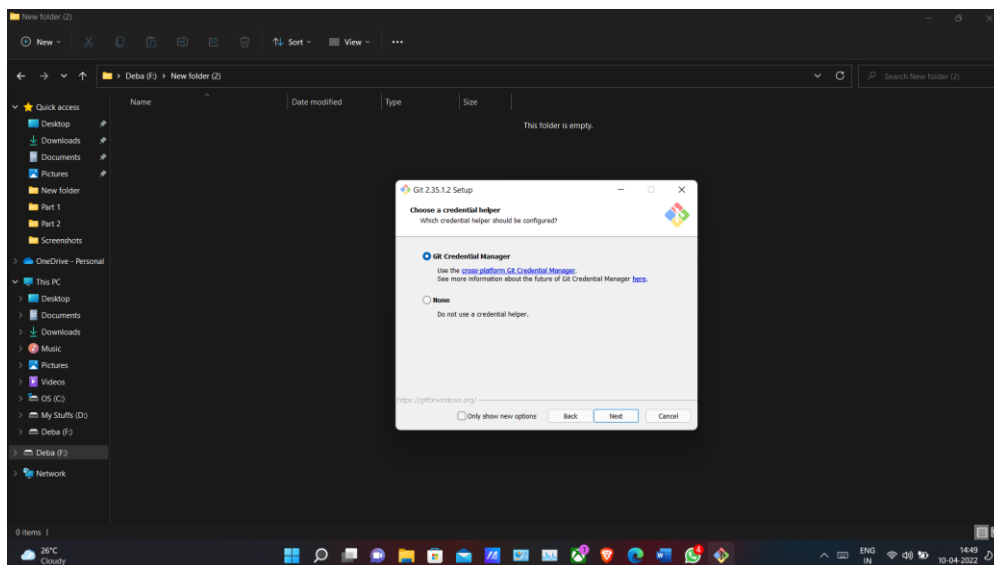


6. The installer will ask you for an installation location. Leave the default, unless you have reason to change it, and click **Next**.
7. A component selection screen will appear. Leave the defaults unless you have a specific need to change them and click **Next**.
8. The installer will offer to create 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**.
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**.

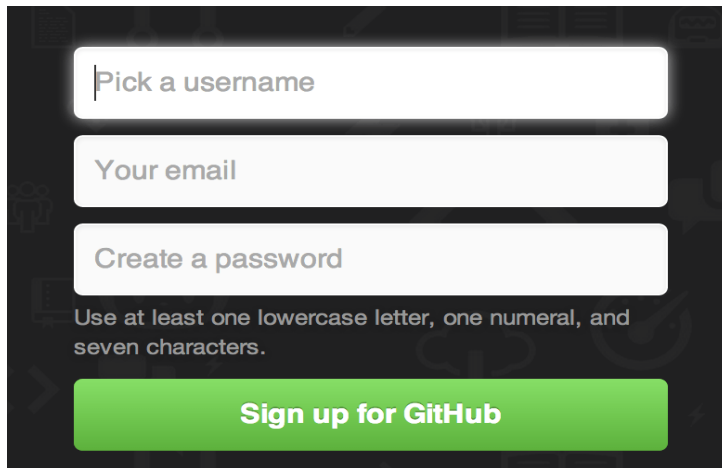
13. The next option relates to server certificates. Most users should use the default. If you're working in an Active Directory environment, you may need to switch to Windows Store certificates. Click **Next**.
14. The next selection converts line endings. It is recommended that you leave the default 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**.



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.



The image shows the GitHub sign-up form. It has a dark background with white text. There are three input fields: "Pick a username", "Your email", and "Create a password". Below the password field, there is a note: "Use at least one lowercase letter, one numeral, and seven characters." At the bottom, there is a large green button that says "Sign up for GitHub".

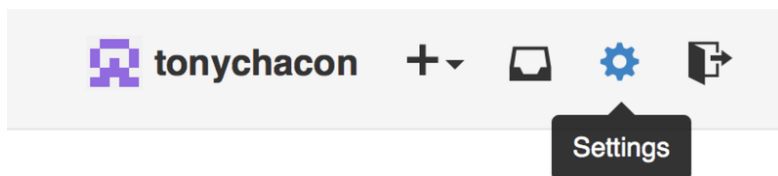
The next thing you’ll see is the pricing page for upgraded plans, but it’s safe to ignore this for now. GitHub will send you an email to verify the address you provided. Go ahead and do this; it’s pretty important (as we’ll see later).

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.

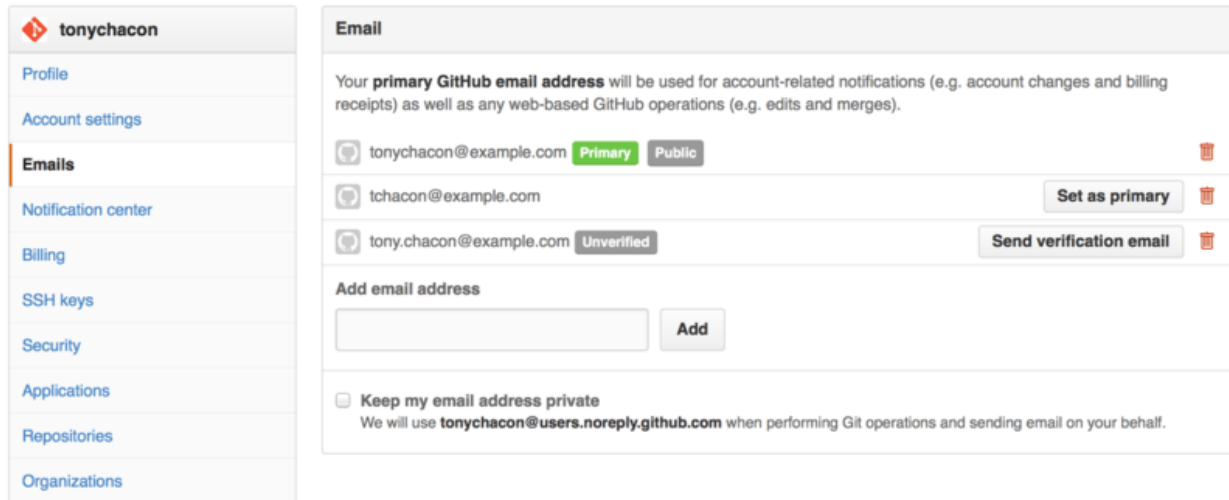


The screenshot shows the GitHub interface for user 'tonychacon'. On the left is a sidebar with navigation links: Profile, Account settings, Emails, Notification center, Billing, SSH keys (highlighted), Security, Applications, Repositories, and Organizations. The main content area has a header with a help link: 'Need help? Check out our guide to [generating SSH keys](#) or troubleshoot [common SSH Problems](#)'. Below this is a section titled 'SSH Keys' with an 'Add SSH key' button. A message states: 'There are no SSH keys with access to your account.' Below this is a form titled 'Add an SSH Key'. The form has two fields: 'Title' (a text input) and 'Key' (a large text area). At the bottom of the form is a green 'Add key' button.

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.



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.



tonychacon

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

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.

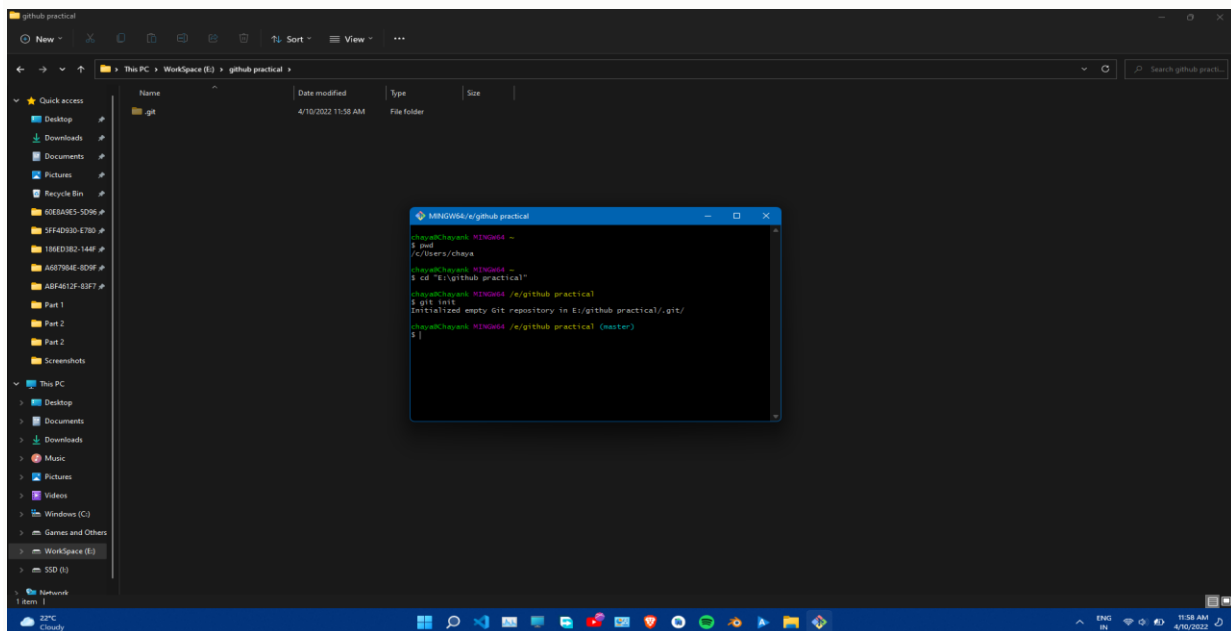


Creating a repository:

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

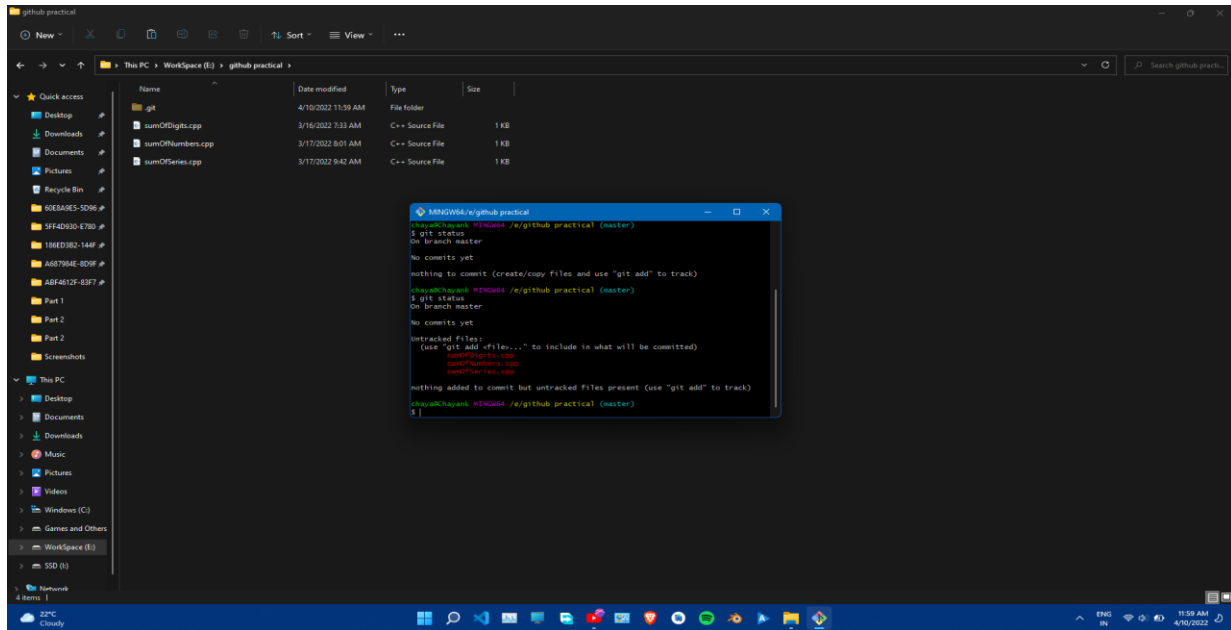
1. You can take a local directory that is currently not under version control, and turn it into a Git repository, or
2. You can clone an existing Git repository from elsewhere.

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.



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)



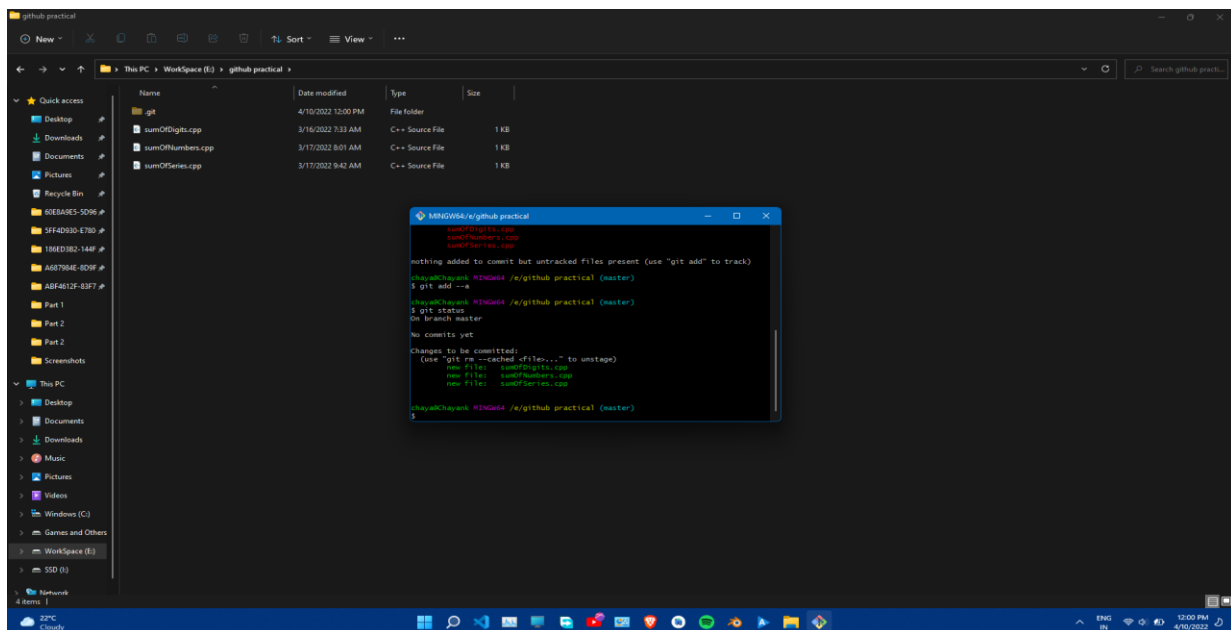
```
github practical
New
This PC > Workspace (E:) > github practical
Name      Date modified      Type      Size
.git      4/19/2022 11:39 AM  File folder
sumOfDigits.cpp  3/18/2022 7:33 AM  C++ Source File  1 KB
sumOfNumbers.cpp  3/17/2022 8:51 AM  C++ Source File  1 KB
sumOfSeries.cpp  3/17/2022 9:42 AM  C++ Source File  1 KB

MINGW64 ~/github practical
$ git status
On branch master
No commits yet
nothing to commit (create/copy files and use "git add" to track)

MINGW64 ~/github practical
$ 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)
MINGW64 ~/github practical
$
```

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.



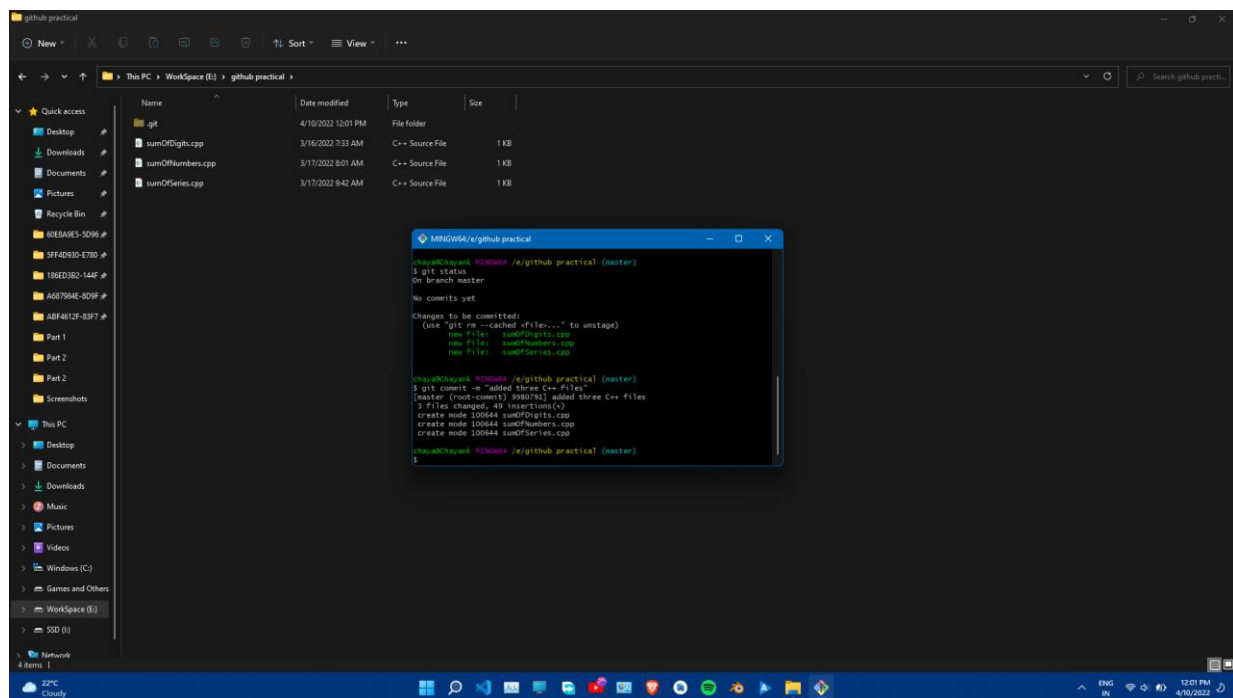
Staging modified files:

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@chaya: ~/github practical (master)
$ git status
On branch master
No commits yet

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

chaya@chaya: ~/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@chaya: ~/github practical (master)
$
```



Basic Git log:

Git log command is one of the most usual commands of git. It is the most useful command of git as 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 can be use as (git log)

```
debar@LAPTOP-VSTFNTTL MINGW64 /d/C++/programming_files (fossil)
$ git log
commit 9511888fc45e37a088b25f94927660f76c89cfa7 (HEAD -> fossil, origin/master, NewOrigin/master, master)
Author: Debarun Das <debarundb14@gmail.com>
Date: Sun Apr 10 18:40:49 2022 +0530

    First commit

commit 7cbe0662e302efbeb0818eff5130579105982cc5
Author: Debarun Das <debarundb14@gmail.com>
Date: Tue Mar 22 12:56:42 2022 +0530

    Second commit/new comment added

commit 9d0fe87f8b450c194341d1738ec550e3a2736029
Author: Debarun Das <debarundb14@gmail.com>
Date: Tue Mar 22 12:52:35 2022 +0530

    second repo

commit 2f150e7ed19f6f24fef352159accf999fb847c52 (origin1/master)
Author: Debarun Das <debarundb14@gmail.com>
Date: Tue Mar 22 12:42:47 2022 +0530

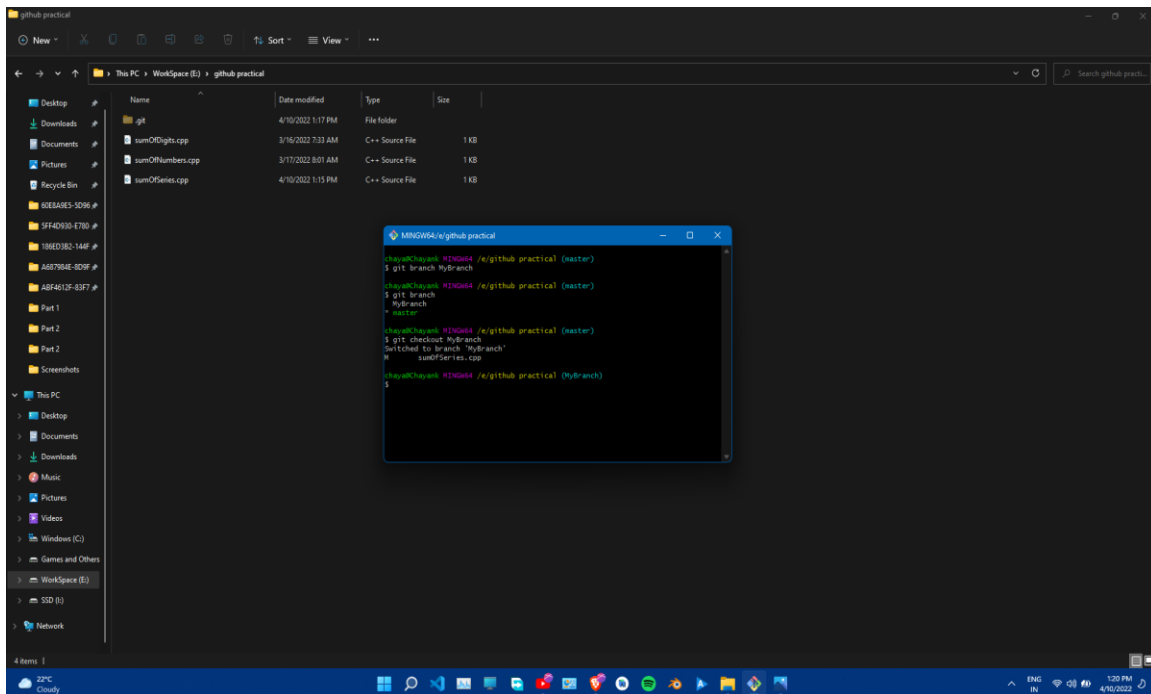
    first repo
debar@LAPTOP-VSTFNTTL MINGW64 /d/C++/programming_files (fossil)
$
```

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”



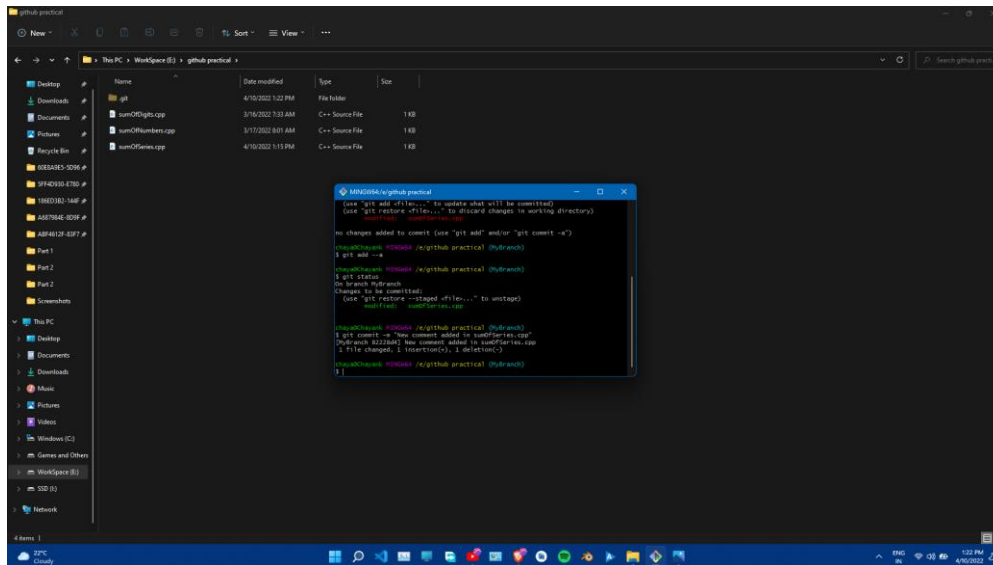
Visualizing Branches:

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 tagging area and commit in the repo. Finally we can rollback to any previously saved version of this file.

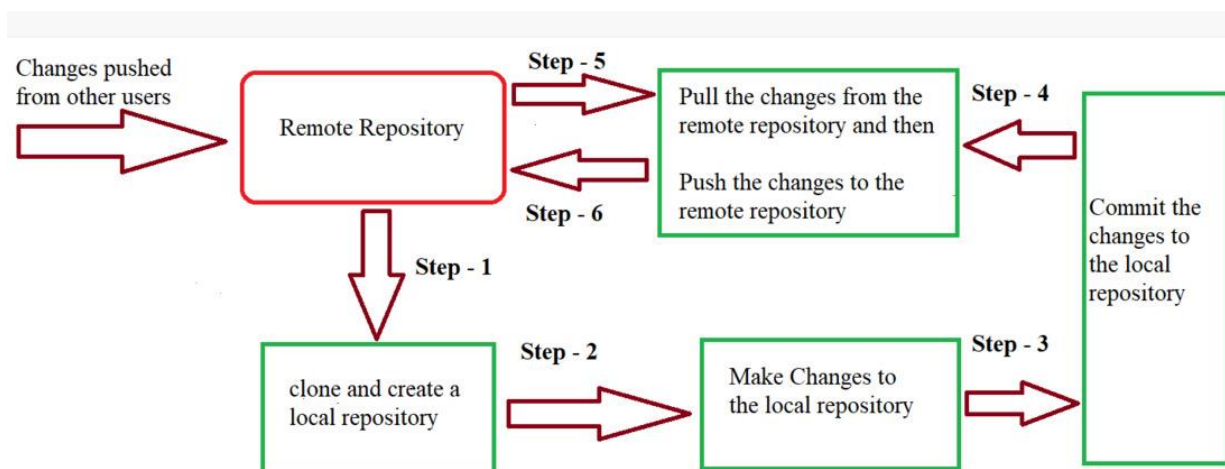
Now, we will change the branch from MyBranch to master, but when we switch back to master branch the file we created i.e “hello” will not be there. Hence the new file will not be shown in the master branch. In this way we can create and change different branches. We can also merge the branches by using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



Git lifecycle:

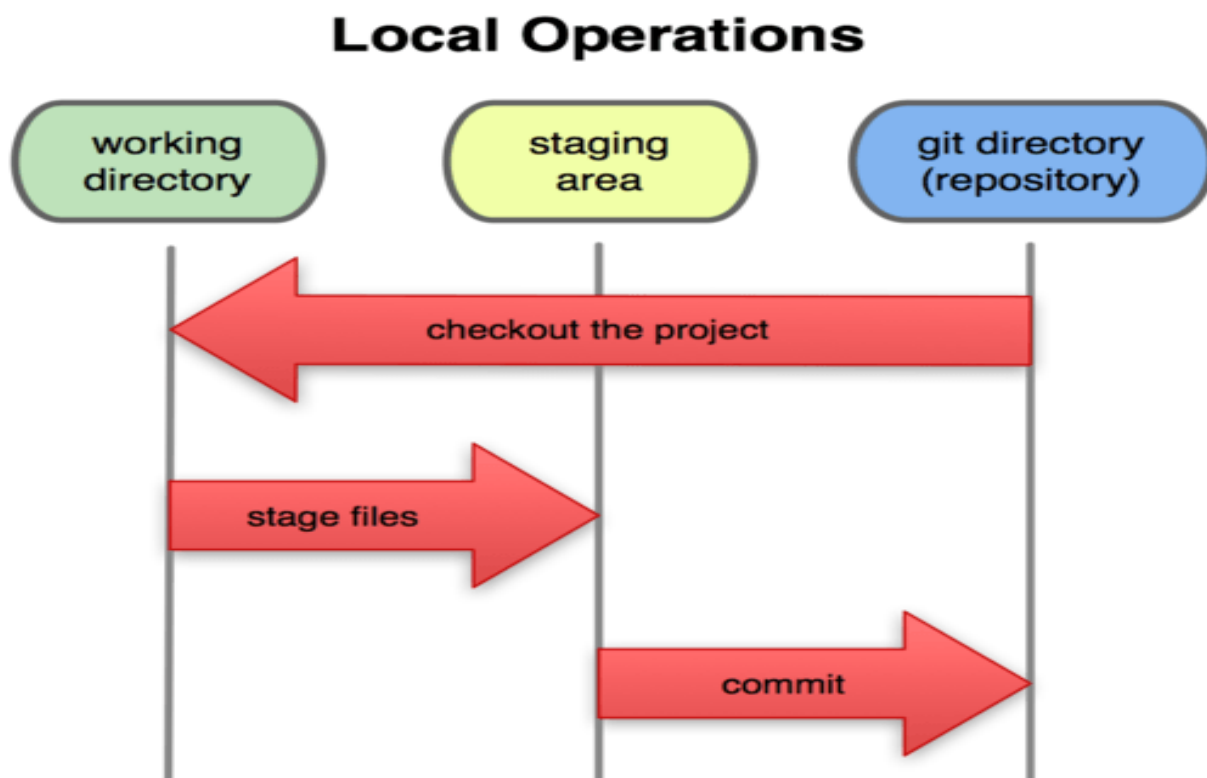
Git is used in our day-to-day work, we use Git for keeping a track of our files, working in a collaboration with our team, to go back to our previous code versions if we face some error. Git helps us in many ways. Let us look at the Lifecycle description that git has and understand more about its life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



- Step 1- We first clone any of the code residing in the remote repository to make our own 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 our 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`

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 our 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`