Subject Name: **Source Code Management**

Subject Code: **CS181** Cluster: **Zeta** Department: **CSE**



**Submitted By:** Aditya Koul 2110992068

G27

Submitted To:

Dr. Anuj Jain

 **Experiment No. 01**

**Aim:** Setting up of Git Client

# Theory:

**What is Git?**

Git is a software used for tracking changes in any set of files, usually used for coordinating work among members of a team.

# History of VCS:

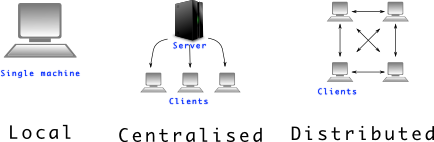
* **Local VCS:** No internet is needed because it uses a database to keep track of files.

Centralized version control systems are based on the

# Centralized VCS:

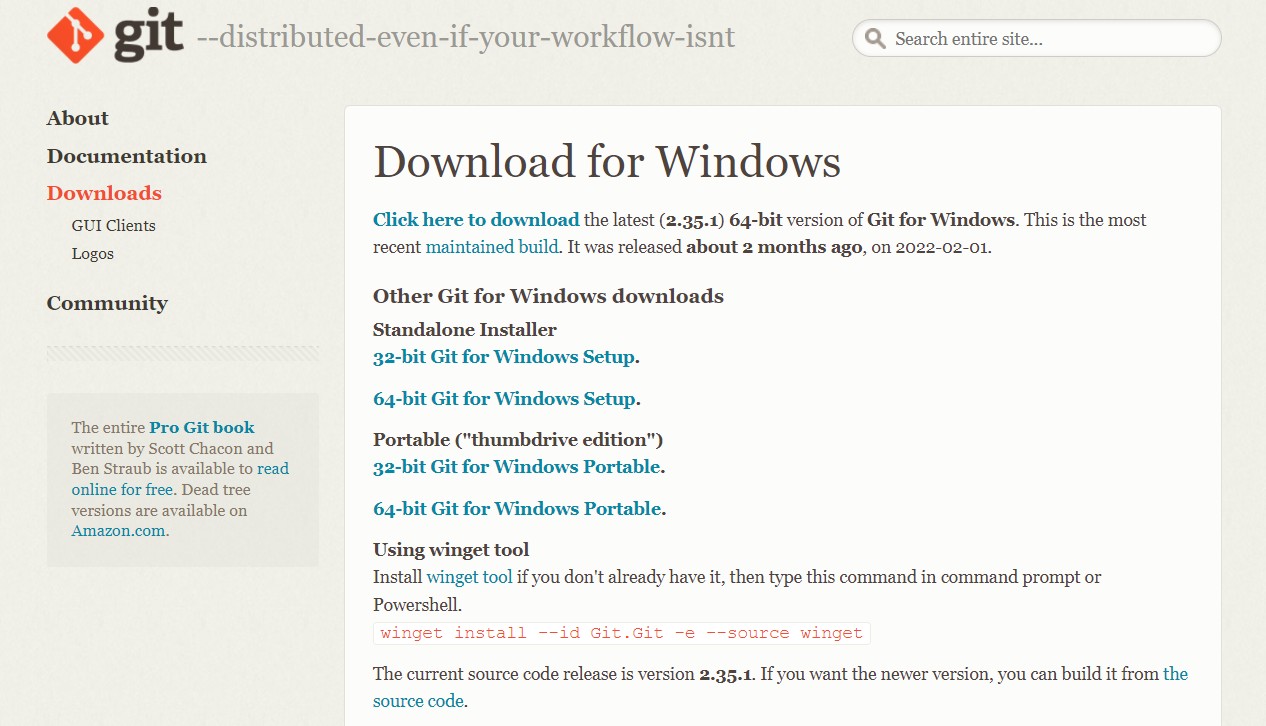
idea that there is a single “central” copy of your project somewhere (probably on a server), and programmers will “commit” their changes to this central copy. “Committing” a change simply means recording the change in the central system.

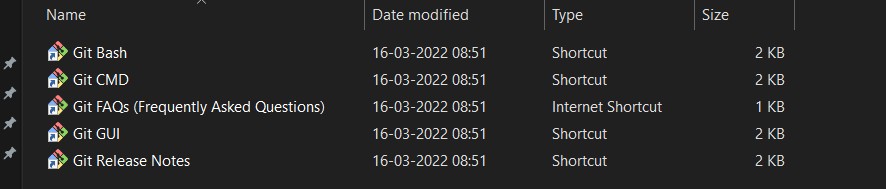
* **Distributed VCS:** A type of version control where the complete codebase including its full version history is mirrored on every developer's computer.



# How to install GIT on Windows?

There are also a few ways to install Git on Windows. The most official build is available for download on the Git website. Just go to [https://git-](https://git-scm.com/download/win) [scm.com/download/win](https://git-scm.com/download/win) and the download will start automatically. Note that this is a project called Git for Windows, which is separate from Git itself; for more information on it, go to [https://gitforwindows.org](https://gitforwindows.org/).





Check version of git by using git –version command.

**Experiment No. 02 **

**Aim:** Setting up GitHub Account

# Theory:

**What is GitHub?**

GitHub is a code hosting platform for version control and collaboration. In other words, it manages repositories.

# Advantages:

* It makes it easy to contribute to Open-Source projects.
* Track changes in your code across versions.

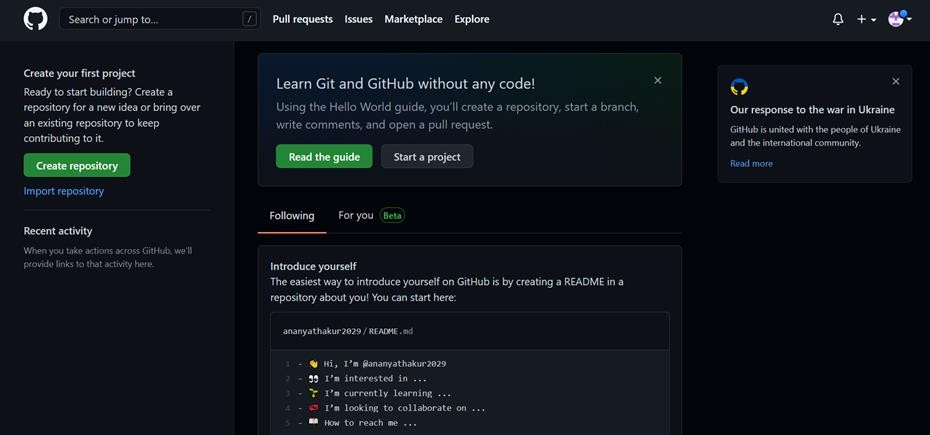
# Procedure:

Search for GitHub in any search engine or <https://github.com/signup>



If you’re a new user add your email and click on **Sign up for GitHub**. Otherwise click on **Sign In** at the top right corner

# Signing into GitHub:



**Linking GitHub account with Git Bash:**

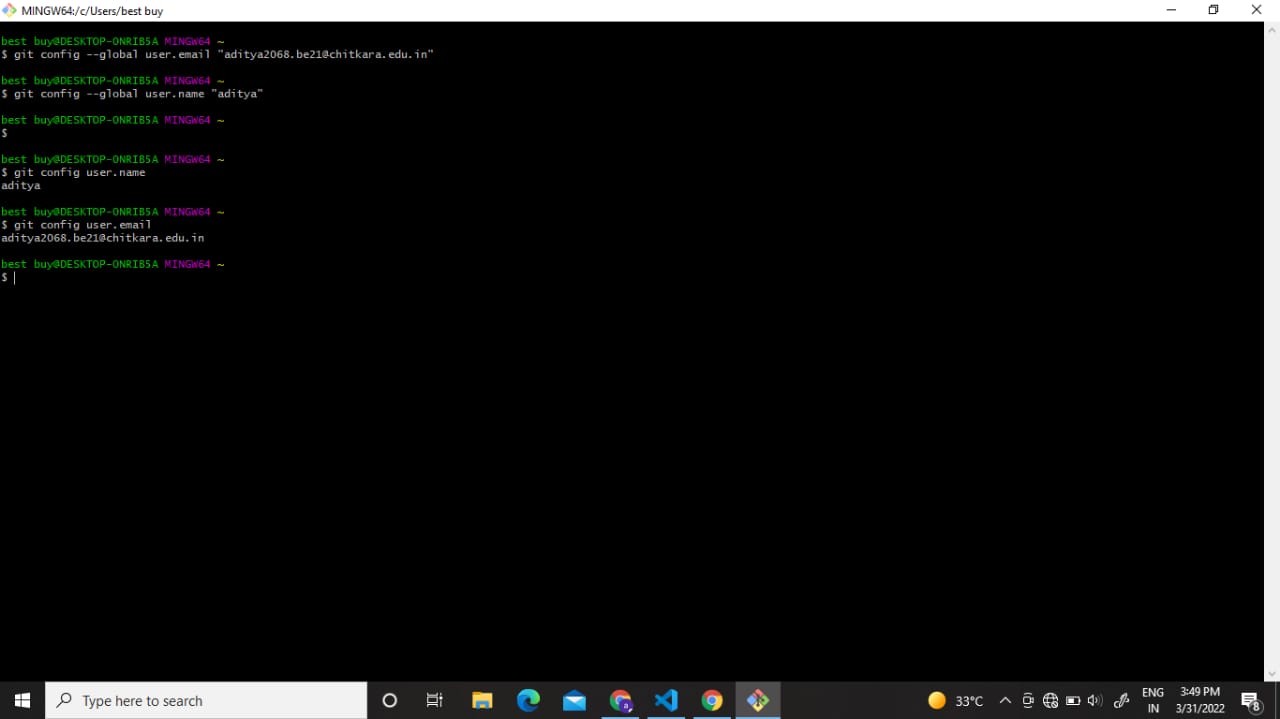
**Username:**

git config --global user.name “username in github”

**Email:**

git config --global user.email “your email in github”

**Check Username & Email:**

git config user.name git config user.email

**Experiment No. 03 **

**Aim:** Program to Generate log

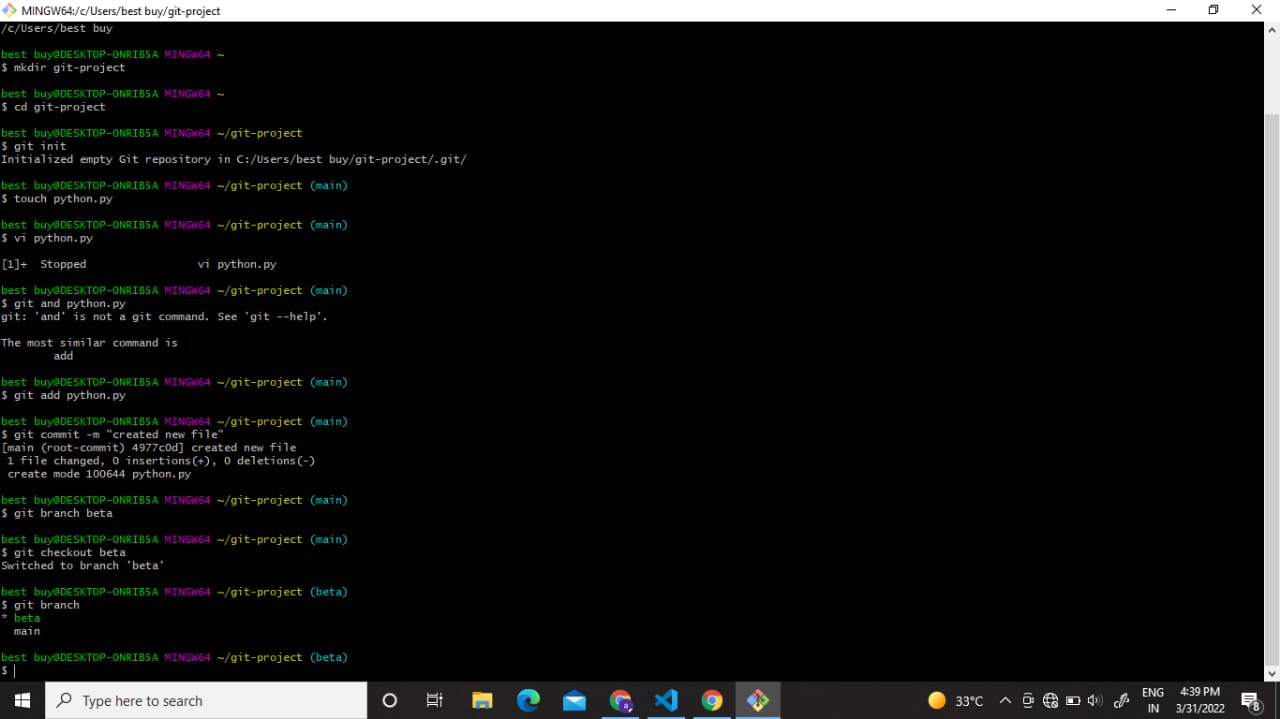
# Theory:

**Git Logs:**

Logs are nothing but the history which we can see in Git by using the code Git log. It contains all the past commits, insertions and deletions which can be seen anytime.

**Why do we need logs?**

Logs help us to check the changes made in code or files and by whom. It also contains the details of insertions and deletions and also the time it was changed at.



* **Use command** git log **to access logs.**

**Experiment No. 04 **

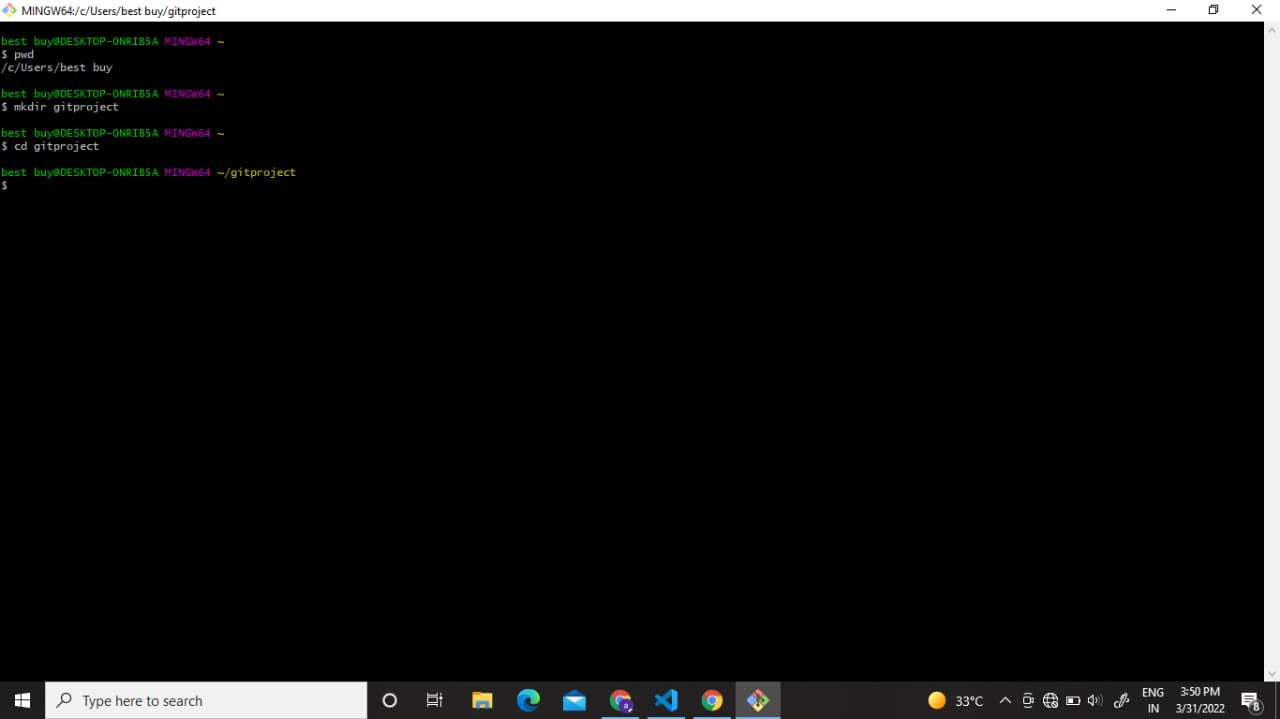
**Aim:** Create and visualize branches

# Theory:

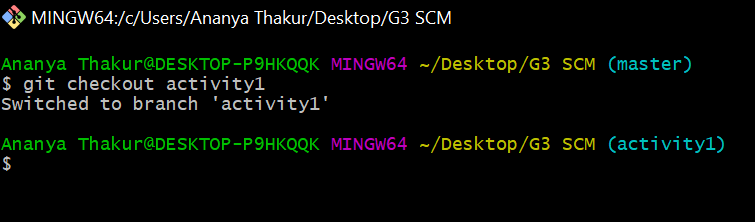
**How to create branches?**

The main branch in git is called the 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 can also merge both the parent (master) and child (other branches).

1. **For creating a new branch:** git branch “name of the branch”
2. **To check how many branches we have:** git branch

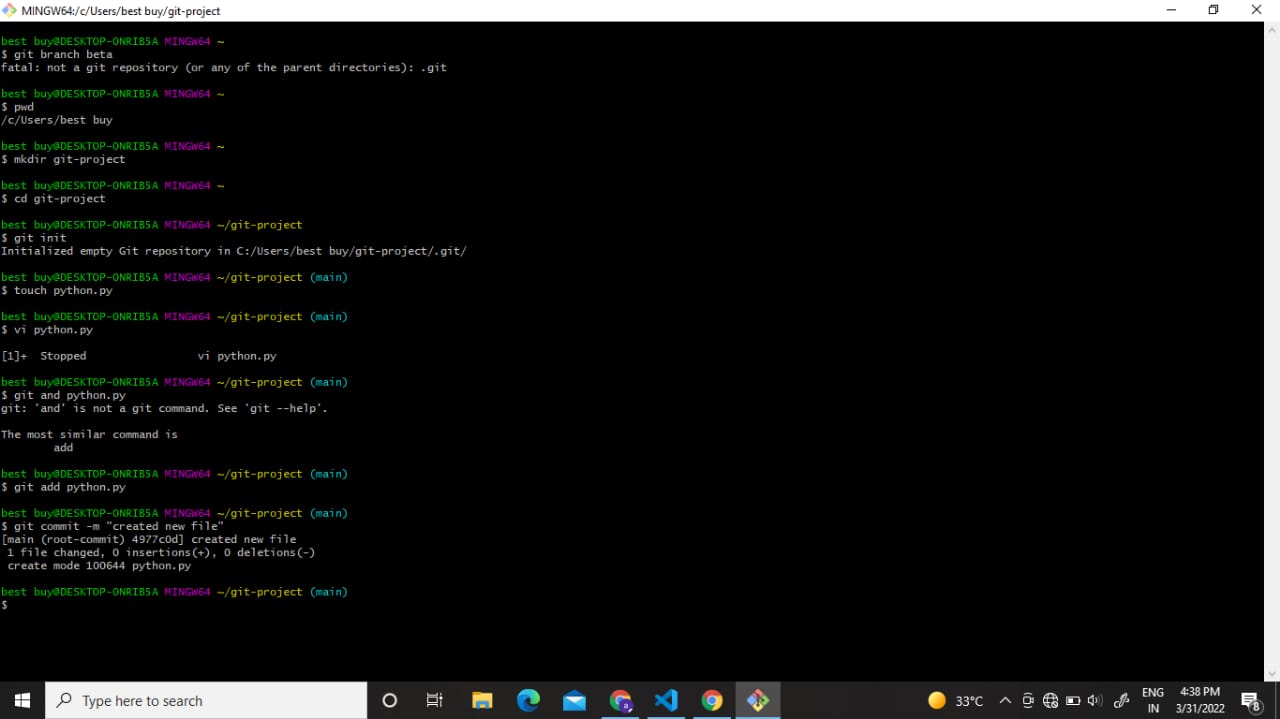


1. **To change the present working branch:** git checkout “name of the branch”



# Visualizing branches:

To visualize, we have to create a new file in the new branch “activity1” instead of the master branch. After this we have to do three step architecture i.e working directory, staging area and git repository.



After this I have done the 3 Step architecture which is tracking the file, send it to stagging

area and finally we can rollback to any previously saved version of this file.

After this we will change the branch from activity1 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.

In this way we can create and change different branches. We can also merge the branches by using git merge command.

**Experiment No. 05 **

**Aim:** Git lifecycle description

# Theory:

**Stages in GIT Life Cycle:**

Files in a Git project have various stages like Creation, Modification, Refactoring, and Deletion and so on. Irrespective of whether this project is tracked by Git or not, these phases are still prevalent. However, when a project is under Git version control system, they are present in three major Git states in addition to these basic ones. Here are the three Git states:

* Working directory
* Staging area
* Git directory

**Working Directory:**

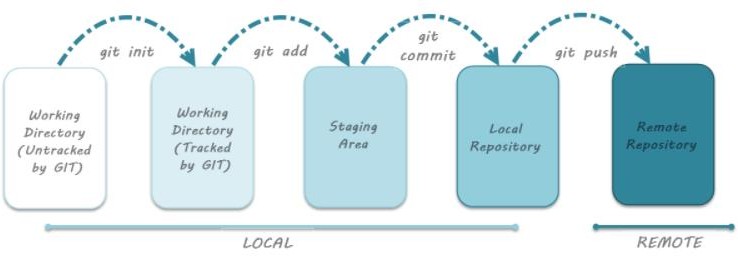
Consider a project residing in your local system. This project may or may not be tracked by Git. In either case, this project directory is called your Working directory.

**Staging Area:**

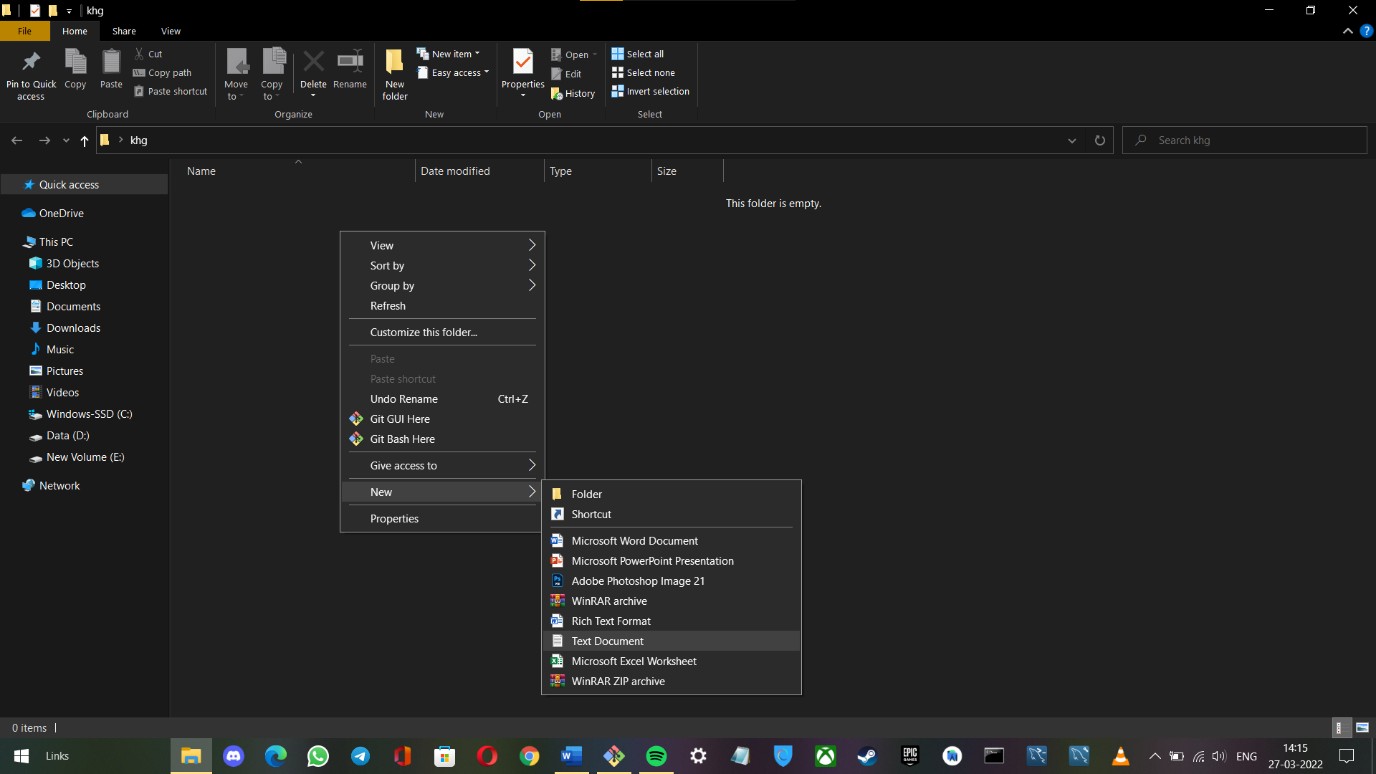
Staging area is the playground where you group, add and organize the files to be committed to Git for tracking their versions.

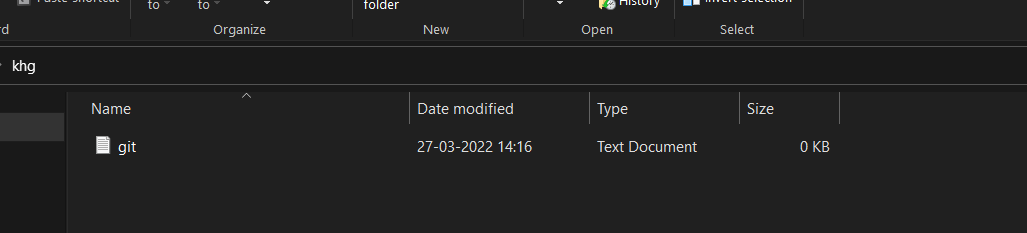
**Git Directory:**

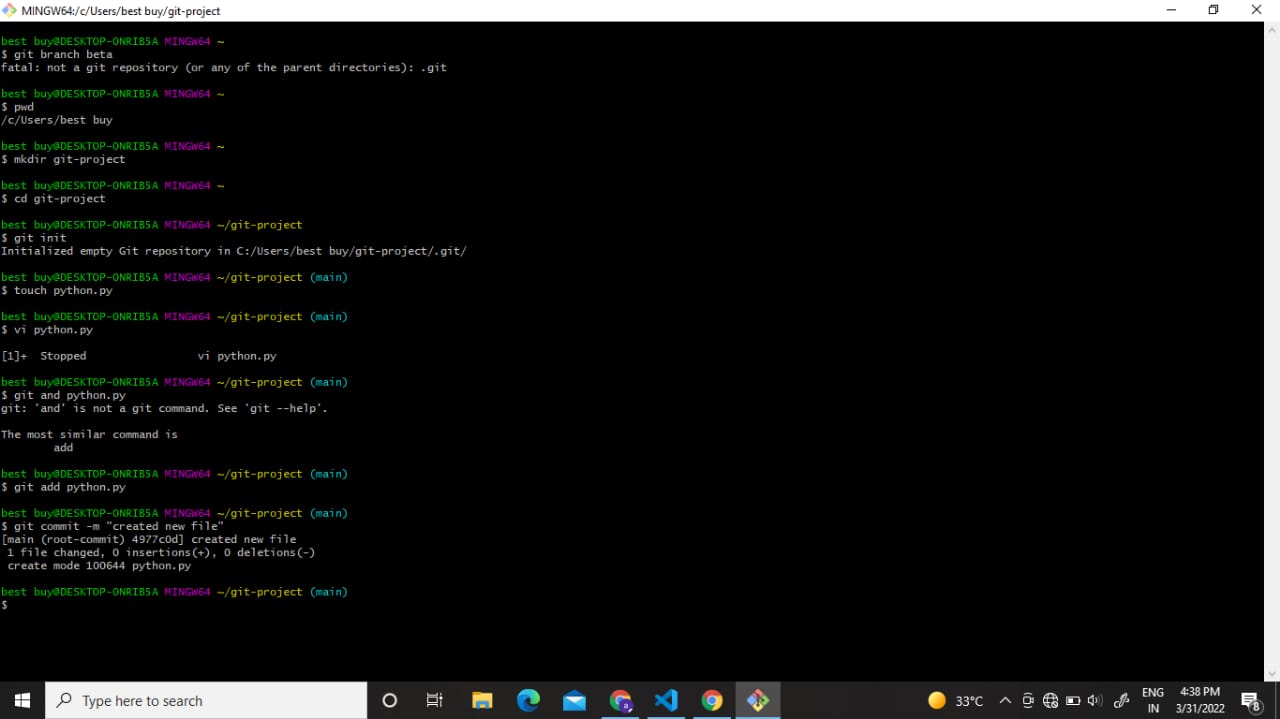
Now that the files to be committed are grouped and ready in the staging area, we can commit these files. So, we commit this group of files along with a commit message explaining what is the commit about. Apart from commit message, this step also records the author and time of the commit. Now, a snapshot of the files in the commit is recorded by Git. The information related to this commit is stored in the Git directory.

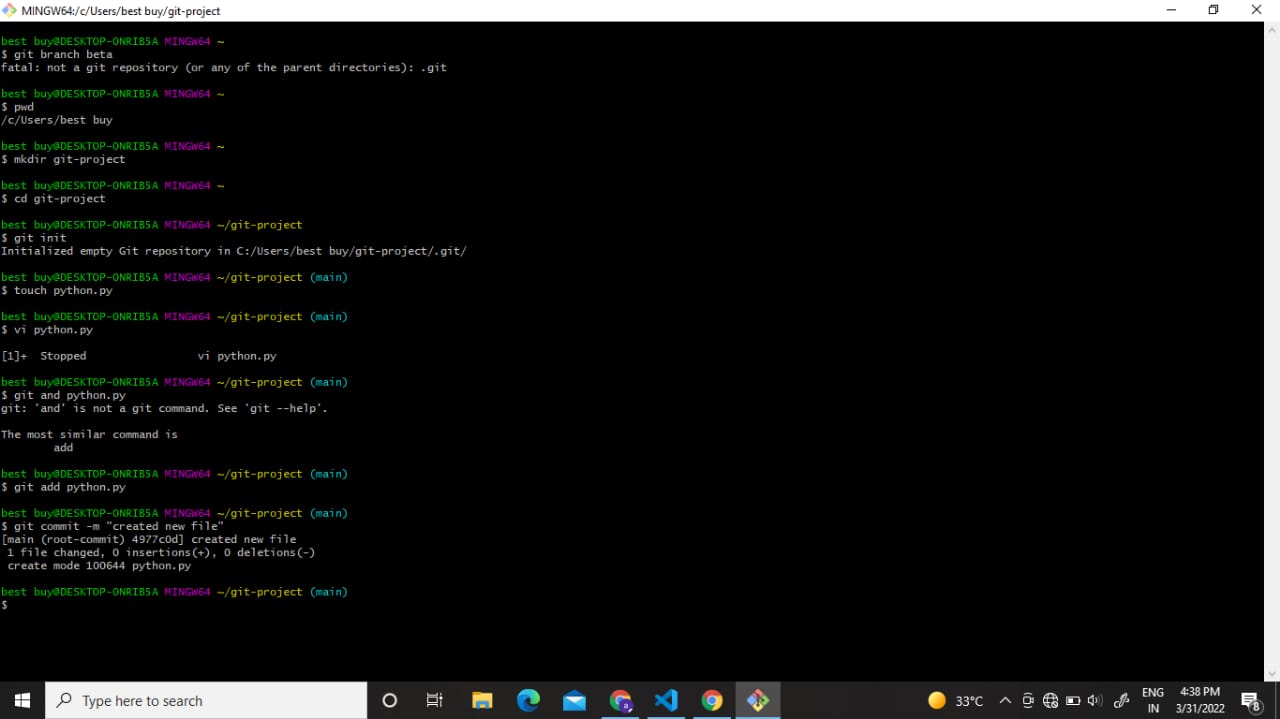
**Remote Repository:** means mirror or clone of the local Git repository in GitHub. And pushing means uploading the commits from local Git repository to remote repository hosted in GitHub.

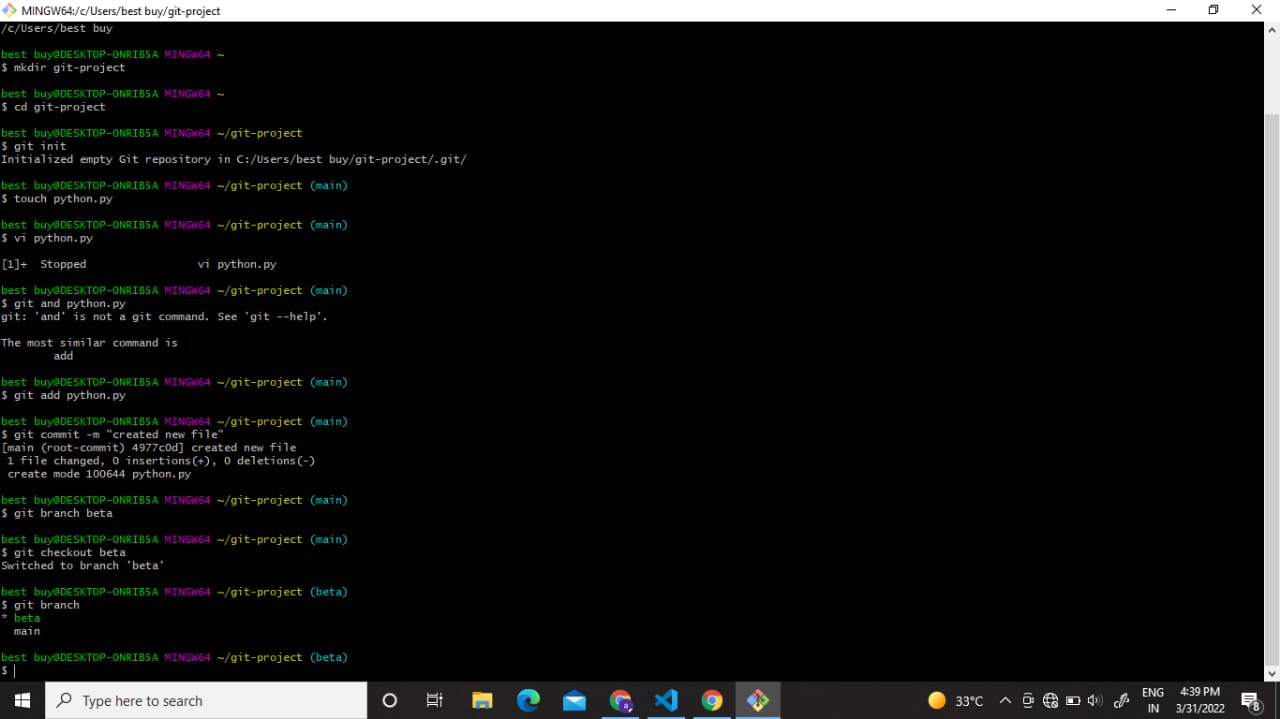
# Snapshots:











Subject Name: **Source Code Management**

Subject Code: **CS181** Cluster: **Zeta** Department: **CSE**



**Submitted By:** Ankit kumar rai 2110992102

G27

Submitted To:

Dr. Anuj Jain

 **Experiment No. 01**

**Aim:** Setting up of Git Client

# Theory:

**What is Git?**

Git is a software used for tracking changes in any set of files, usually used for coordinating work among members of a team.

# History of VCS:

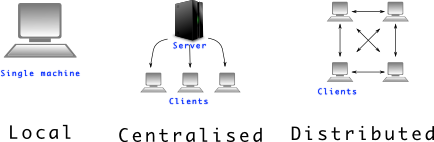
* **Local VCS:** No internet is needed because it uses a database to keep track of files.

Centralized version control systems are based on the

# Centralized VCS:

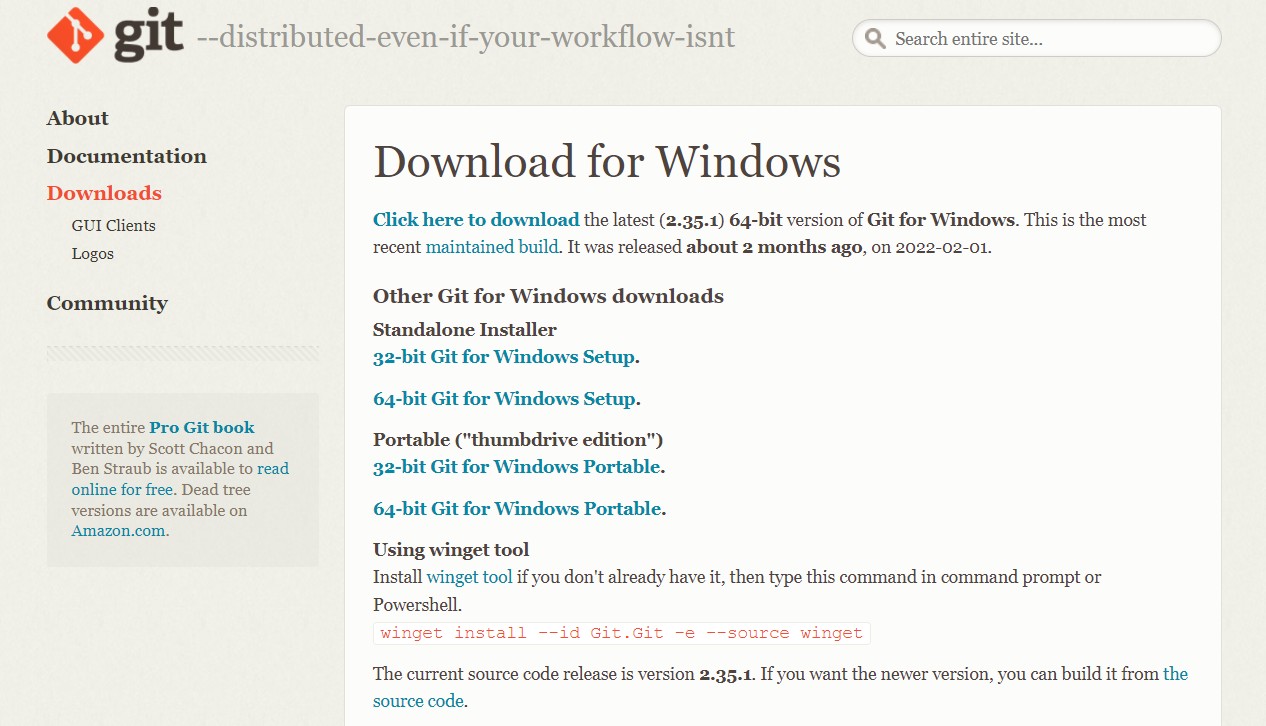
idea that there is a single “central” copy of your project somewhere (probably on a server), and programmers will “commit” their changes to this central copy. “Committing” a change simply means recording the change in the central system.

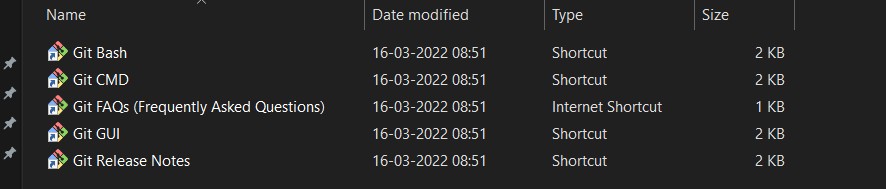
* **Distributed VCS:** A type of version control where the complete codebase including its full version history is mirrored on every developer's computer.



# How to install GIT on Windows?

There are also a few ways to install Git on Windows. The most official build is available for download on the Git website. Just go to [https://git-](https://git-scm.com/download/win) [scm.com/download/win](https://git-scm.com/download/win) and the download will start automatically. Note that this is a project called Git for Windows, which is separate from Git itself; for more information on it, go to [https://gitforwindows.org](https://gitforwindows.org/).





Check version of git by using git –version command.

**Experiment No. 02 **

**Aim:** Setting up GitHub Account

# Theory:

**What is GitHub?**

GitHub is a code hosting platform for version control and collaboration. In other words, it manages repositories.

# Advantages:

* It makes it easy to contribute to Open-Source projects.
* Track changes in your code across versions.

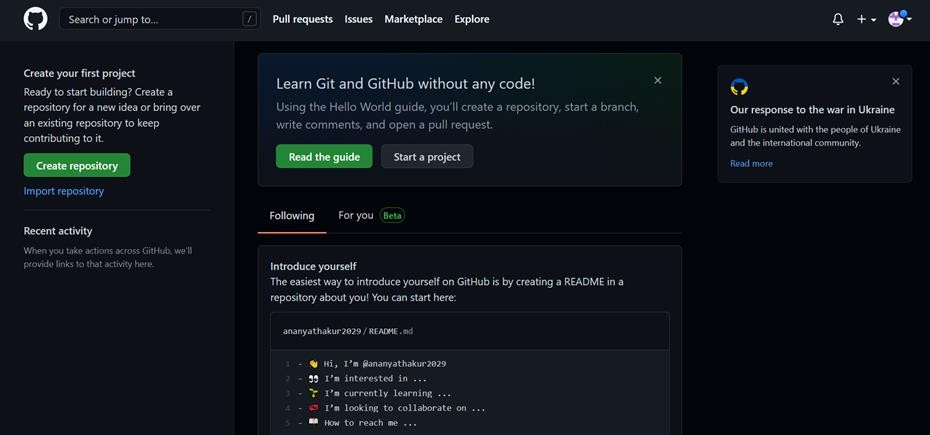
# Procedure:

Search for GitHub in any search engine or <https://github.com/signup>



If you’re a new user add your email and click on **Sign up for GitHub**. Otherwise click on **Sign In** at the top right corner

# Signing into GitHub:



**Linking GitHub account with Git Bash:**

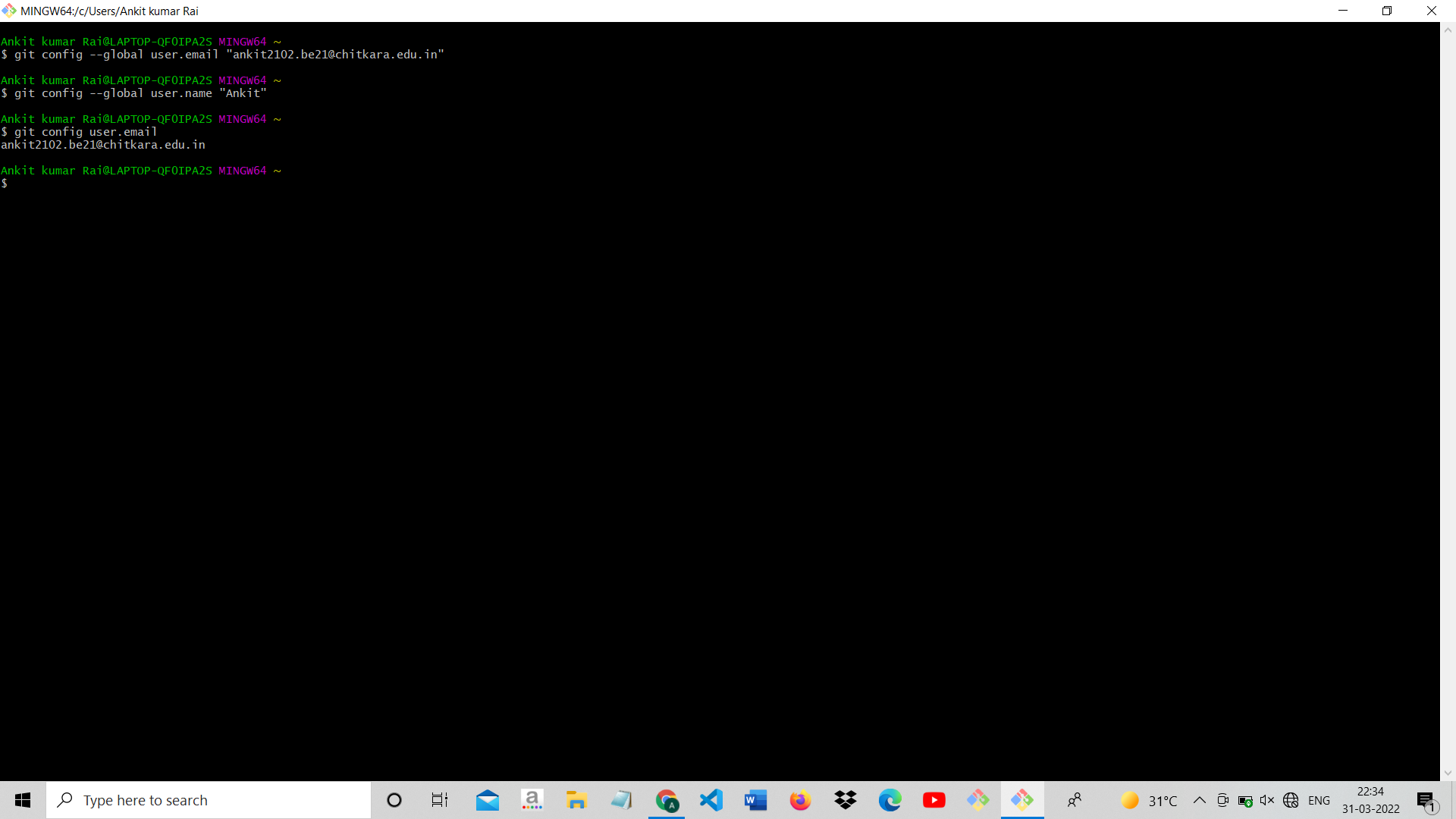
**Username:**

git config --global user.name “username in github”

**Email:**

git config --global user.email “your email in github”

**Check Username & Email:**

git config user.name git config user.email

**Experiment No. 03 **

**Aim:** Program to Generate log

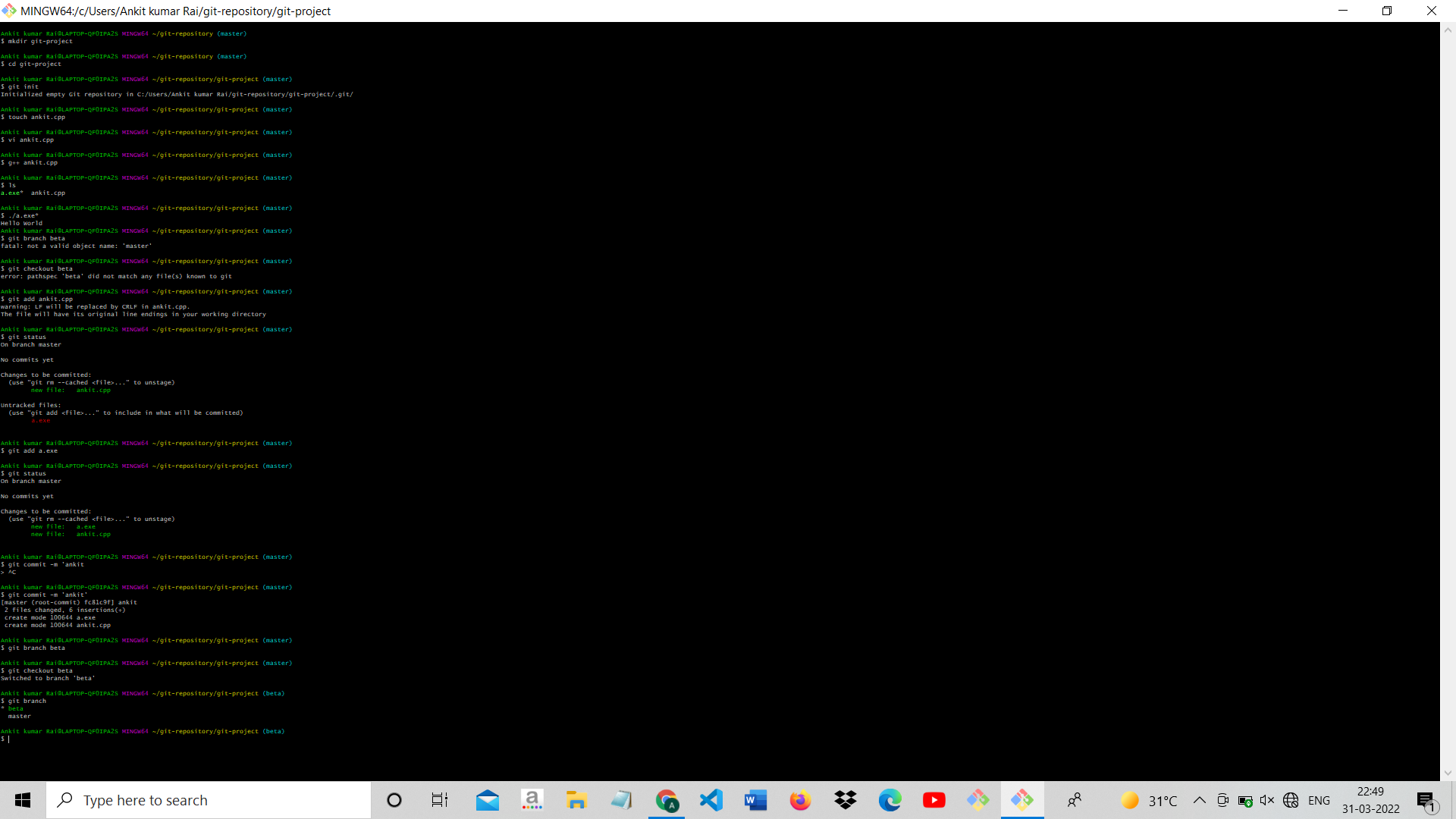
# Theory:

**Git Logs:**

Logs are nothing but the history which we can see in Git by using the code Git log. It contains all the past commits, insertions and deletions which can be seen anytime.

**Why do we need logs?**

Logs help us to check the changes made in code or files and by whom. It also contains the details of insertions and deletions and also the time it was changed at.



* **Use command** git log **to access logs.**

**Experiment No. 04 **

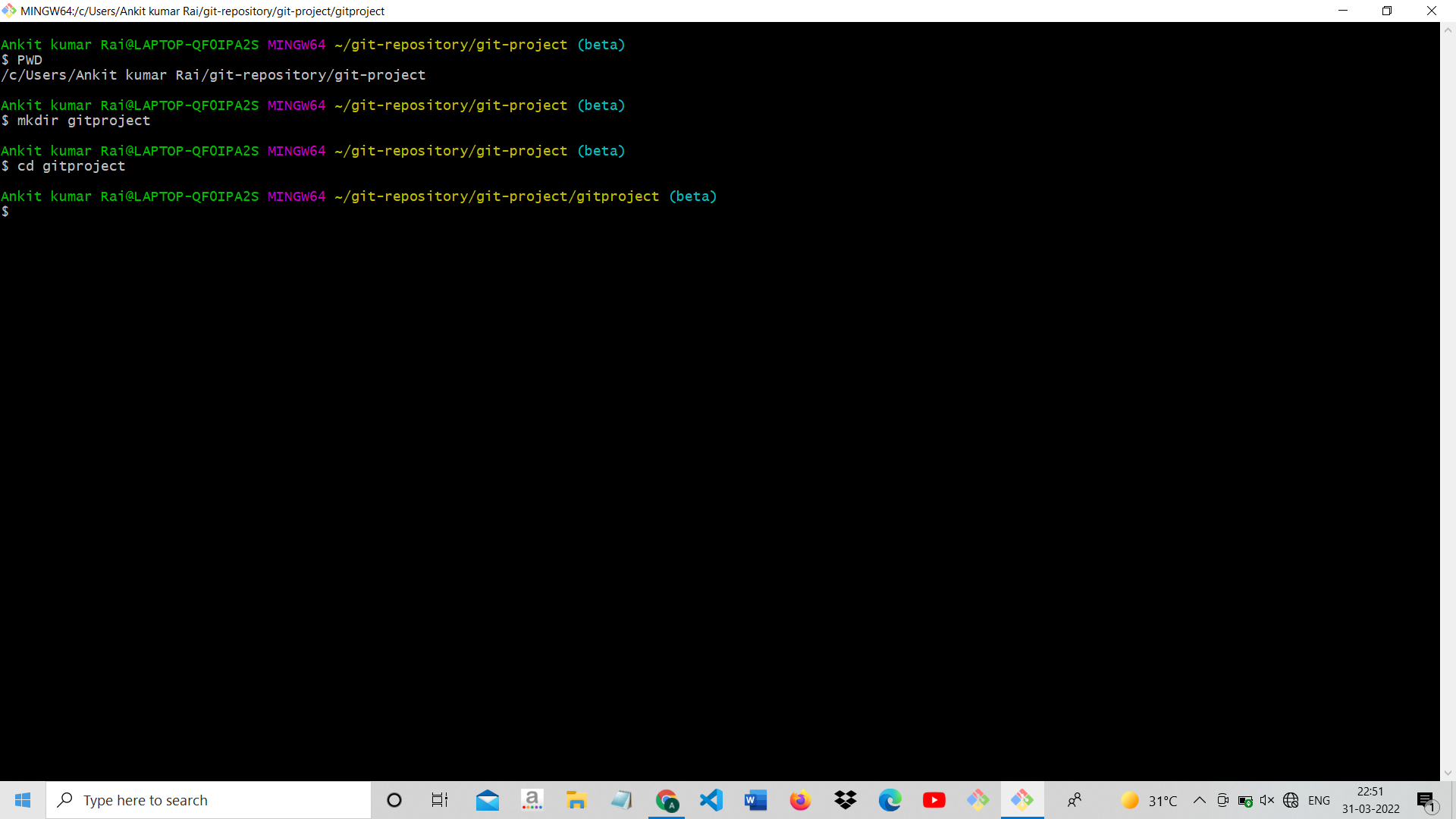
**Aim:** Create and visualize branches

# Theory:

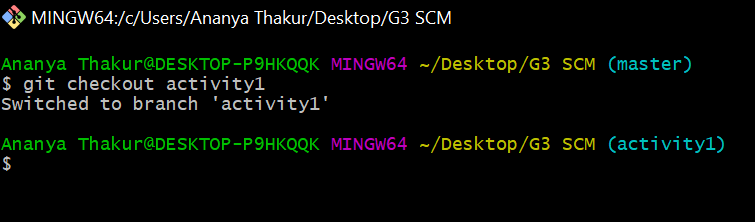
**How to create branches?**

The main branch in git is called the 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 can also merge both the parent (master) and child (other branches).

1. **For creating a new branch:** git branch “name of the branch”
2. **To check how many branches we have:** git branch

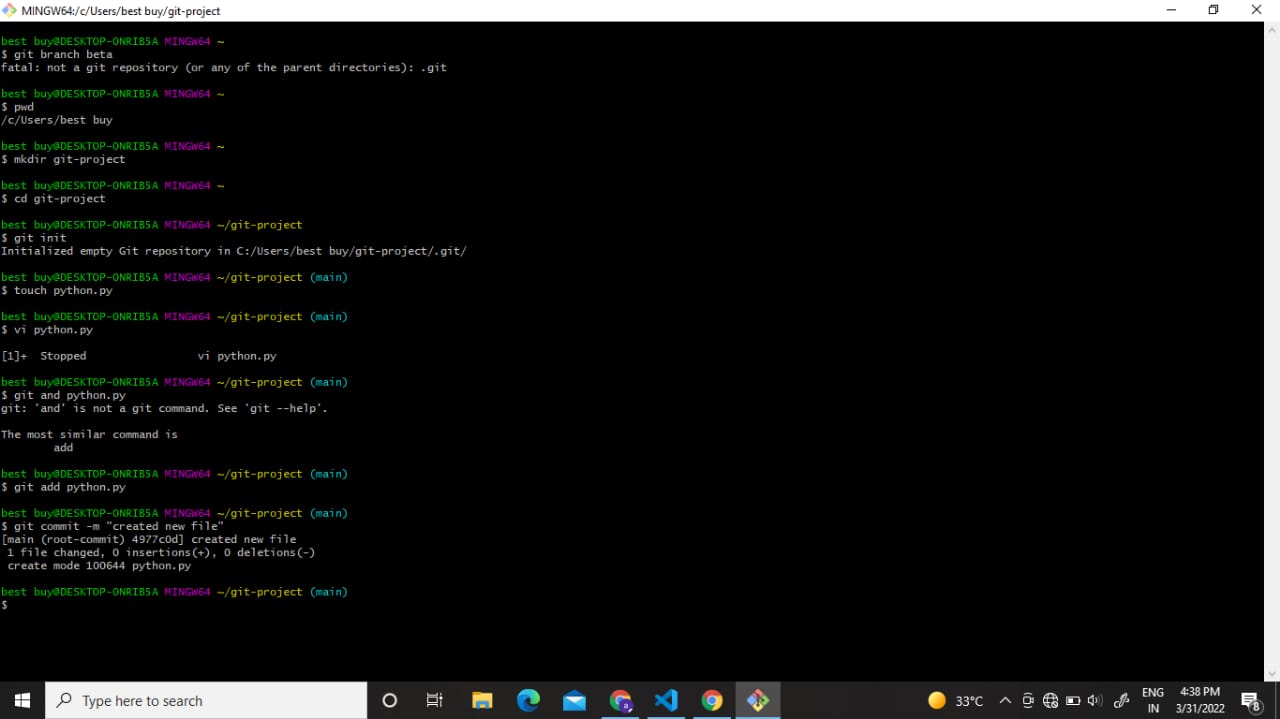


1. **To change the present working branch:** git checkout “name of the branch”



# Visualizing branches:

To visualize, we have to create a new file in the new branch “activity1” instead of the master branch. After this we have to do three step architecture i.e working directory, staging area and git repository.



After this I have done the 3 Step architecture which is tracking the file, send it to stagging

area and finally we can rollback to any previously saved version of this file.

After this we will change the branch from activity1 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.

In this way we can create and change different branches. We can also merge the branches by using git merge command.

**Experiment No. 05 **

**Aim:** Git lifecycle description

# Theory:

**Stages in GIT Life Cycle:**

Files in a Git project have various stages like Creation, Modification, Refactoring, and Deletion and so on. Irrespective of whether this project is tracked by Git or not, these phases are still prevalent. However, when a project is under Git version control system, they are present in three major Git states in addition to these basic ones. Here are the three Git states:

* Working directory
* Staging area
* Git directory

**Working Directory:**

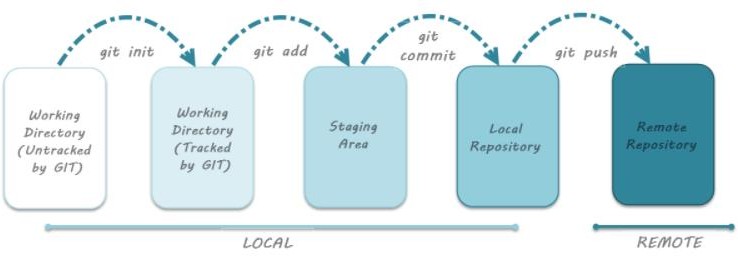
Consider a project residing in your local system. This project may or may not be tracked by Git. In either case, this project directory is called your Working directory.

**Staging Area:**

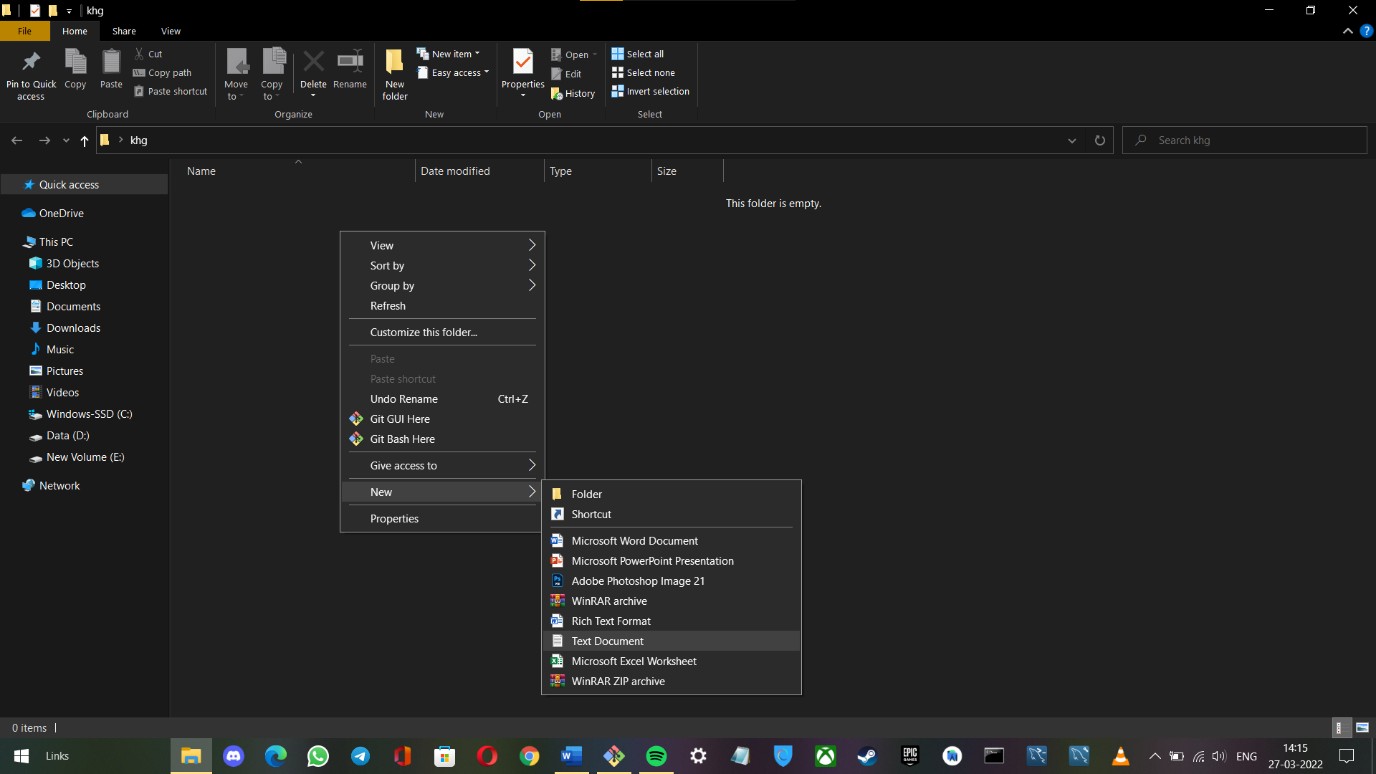
Staging area is the playground where you group, add and organize the files to be committed to Git for tracking their versions.

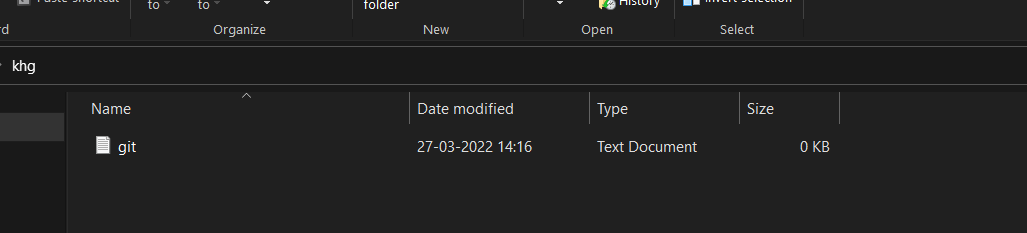
**Git Directory:**

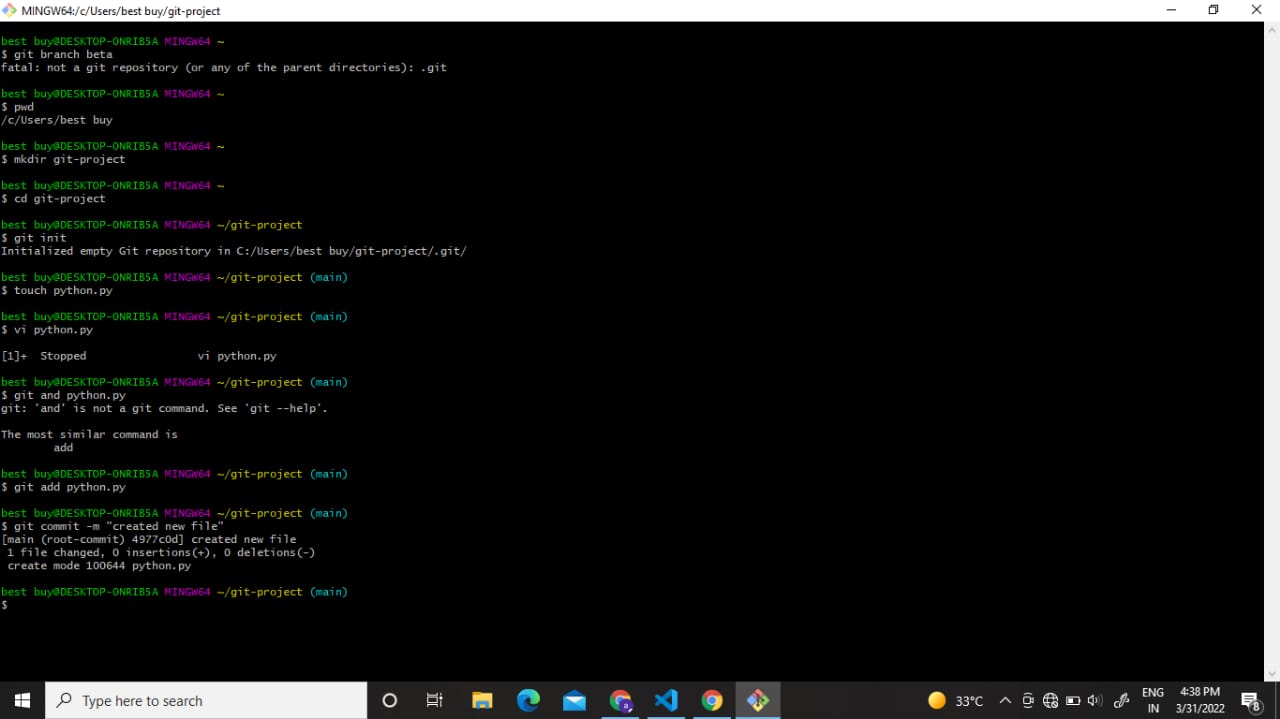
Now that the files to be committed are grouped and ready in the staging area, we can commit these files. So, we commit this group of files along with a commit message explaining what is the commit about. Apart from commit message, this step also records the author and time of the commit. Now, a snapshot of the files in the commit is recorded by Git. The information related to this commit is stored in the Git directory.

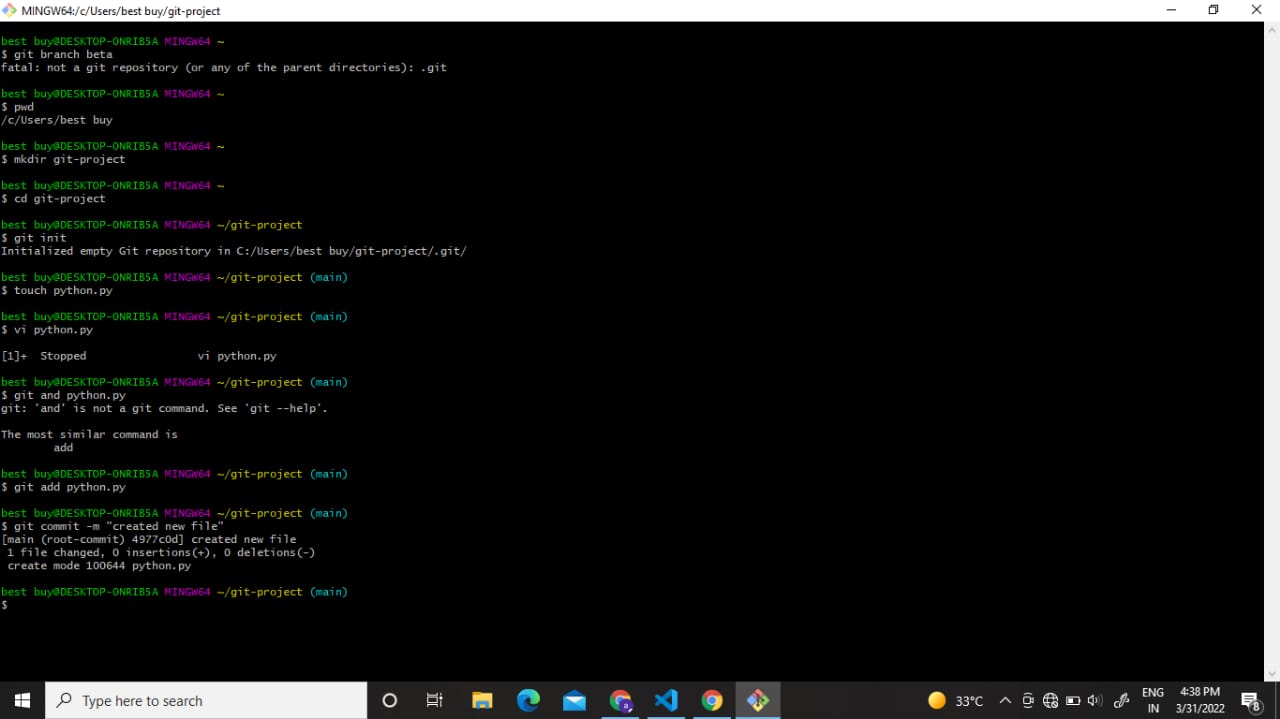
**Remote Repository:** means mirror or clone of the local Git repository in GitHub. And pushing means uploading the commits from local Git repository to remote repository hosted in GitHub.

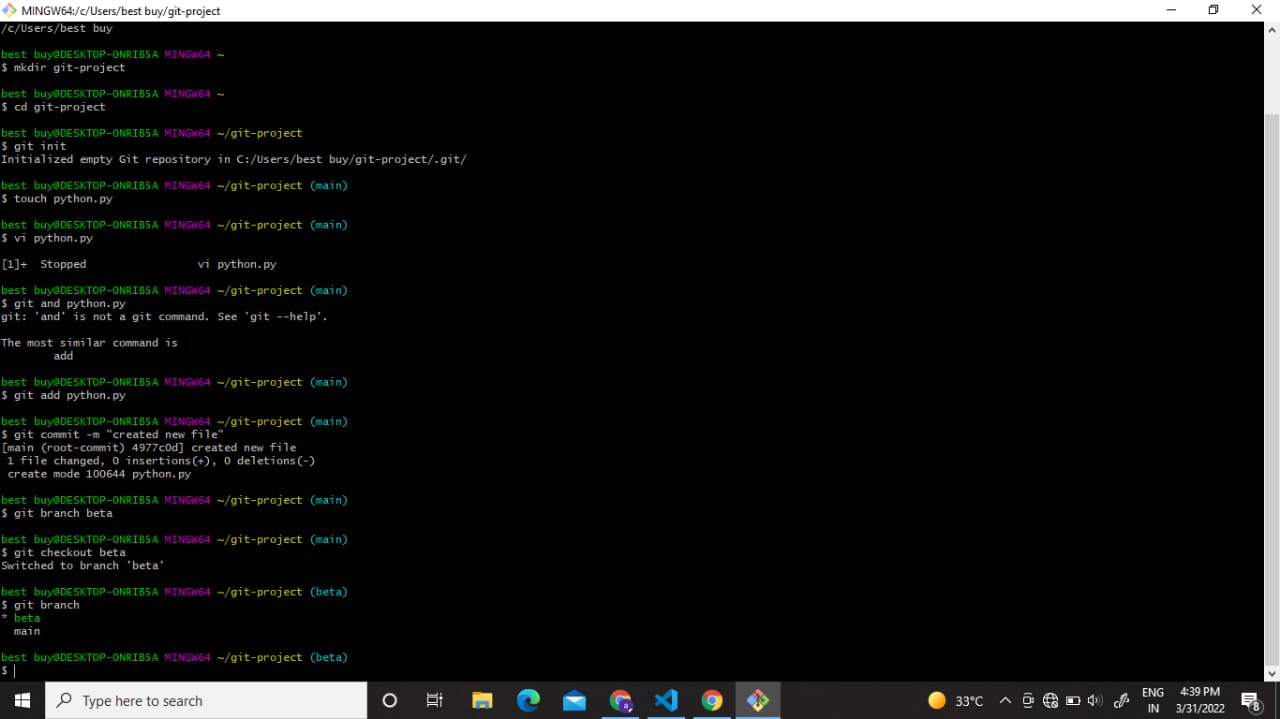
# Snapshots:











02

A Project report

on

**EDUCATION FOR ALL**

with

**Source Code Management**

(CS181)

**Submitted by**

Team Member 1 .ISHIT 2110992060.

Team Member 2 .ADITYA KOUL 2110992068.

Team Member 3 .ANKIT RAI 2110992102.

Team Member 4.SAMARVEER 2110992106.



**Department of Computer Science & Engineering**

Chitkara University Institute of Engineering and Technology, Punjab

Jan- June  
(2021-22)

|  |  |  |  |
| --- | --- | --- | --- |
| Institute/School Name | **Chitkara University Institute of Engineering and Technology** | | |
| Department Name | **Department of Computer Science & Engineering** | | |
| Programme Name | **Bachelor of Engineering (B.E.), Computer Science & Engineering** | | |
| Course Name | **Source Code Management** | Session | **2021-22** |
| Course Code | **CS181** | Semester/Batch | **2nd/2021** |
| Vertical Name | **zeta** | Group No | G-27 |
| Course Coordinator | **Dr. Sachendra Singh** | | |
| Faculty Name | **Dr. Neeraj singhla** | | |

Submission

Name:ADITYA KOUL

Signature:

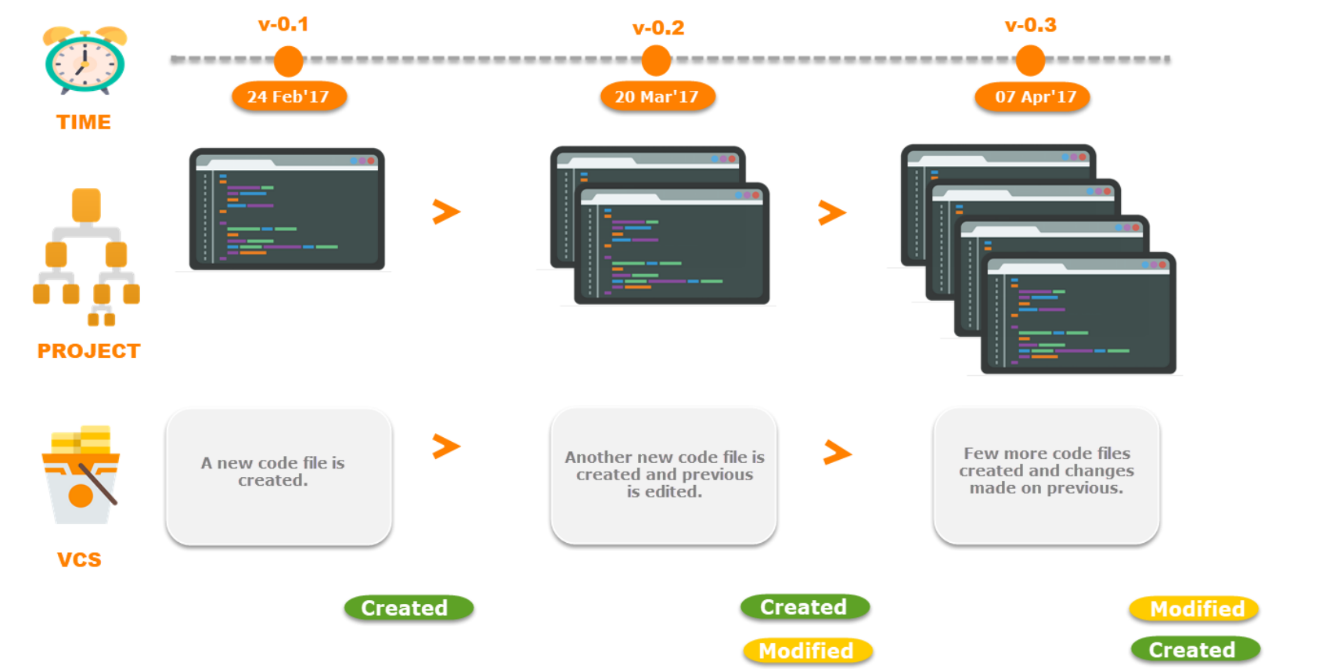
Date:20/5/2022

**Table of Content**

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Title** | **Page No.** |
| 1 | Version control with Git | 1-5 |
| 2 | Problem Statement |  |
| 3 | Objective |  |
| 4 | Concepts and commands |  |
| 5 | Workflow and Discussion |  |
| 6 | Reference |  |

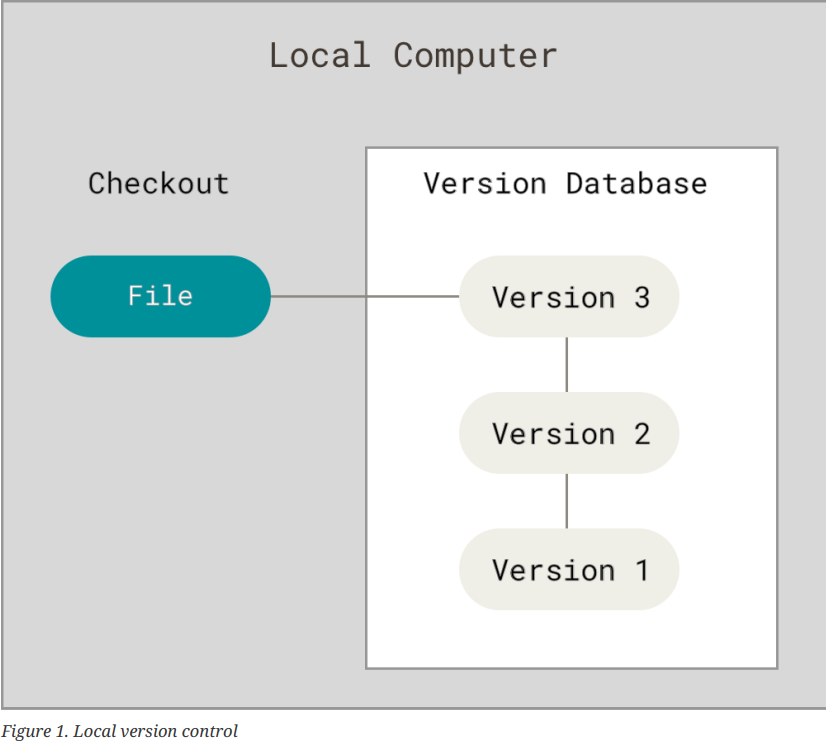
1. Version control with Git

What is “version control”, and why should you care? Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later. For the examples in this book, you will use software source code as the files being version controlled, though in reality you can do this with nearly any type of file on a computer. If you are a graphic or web designer and want to keep every version of an image or layout (which you would most certainly want to), a Version Control System (VCS) is a very wise thing to use. It allows you to revert selected files back to a previous state, revert the entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more. Using a VCS also generally means that if you screw things up or lose files, you can easily recover. In addition, you get all this for very little overhead.



**Local Version Control Systems**

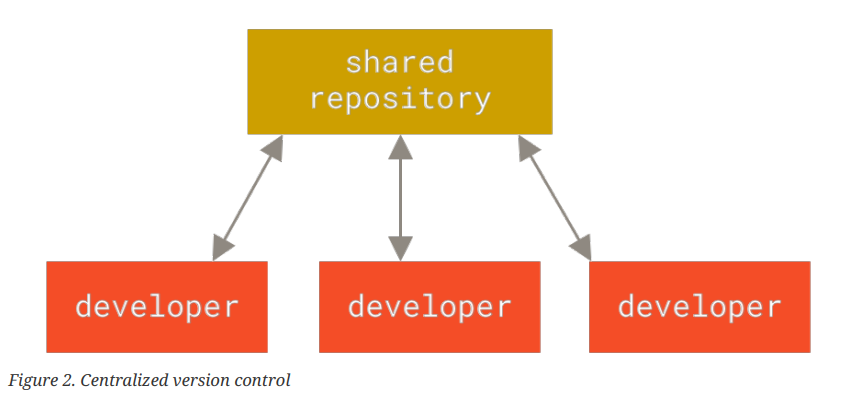
Many people’s version-control method of choice is to copy files into another directory (perhaps a time-stamped directory, if they’re clever). This approach is very common because it is so simple, but it is also incredibly error prone. It is easy to forget which directory you’re in and accidentally write to the wrong file or copy over files you don’t mean to. To deal with this issue, programmers long ago developed local VCSs that had a simple database that kept all the changes to files under revision control.



One of the most popular VCS tools was a system called RCS, which is still distributed with many computers today. RCS works by keeping patch sets (that is, the differences between files) in a special format on disk; it can then re-create what any file looked like at any point in time by adding up all the patches.

**Centralized Version Control Systems**

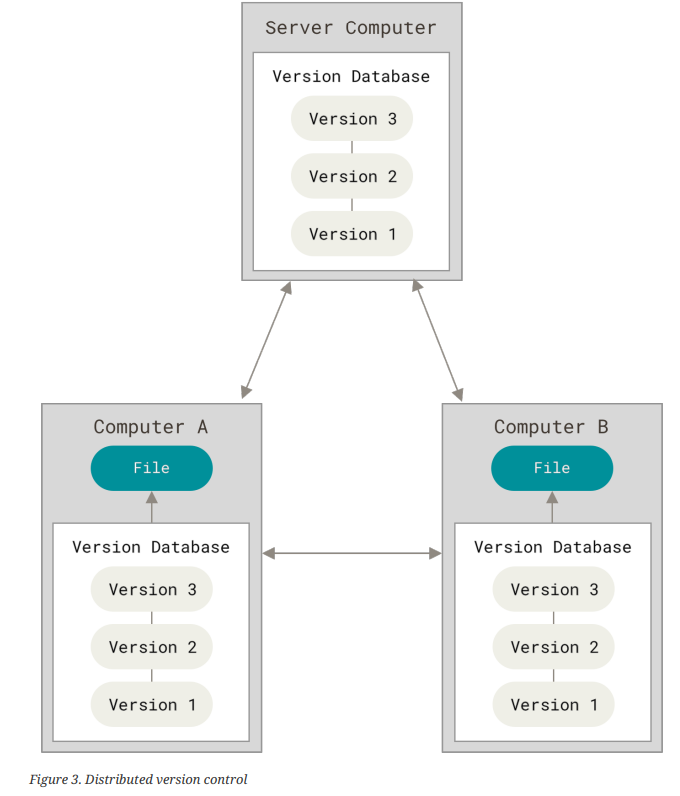
The next major issue that people encounter is that they need to collaborate with developers on other systems. To deal with this problem, Centralized Version Control Systems (CVCSs) were developed. These systems (such as CVS, Subversion, and Perforce) have a single server that contains all the versioned files, and a number of clients that check out files from that central place. For many years, this has been the standard for version control.



This setup offers many advantages, especially over local VCSs. For example, everyone knows to a certain degree what everyone else on the project is doing. Administrators have fine-grained control over who can do what, and it’s far easier to administer a CVCS than it is to deal with local databases on every client. However, this setup also has some serious downsides. The most obvious is the single point of failure that the centralized server represents. If that server goes down for an hour, then during that hour nobody can collaborate at all or save versioned changes to anything they’re working on. If the hard disk the central database is on becomes corrupted, and proper backups haven’t been kept, you lose absolutely everything — the entire history of the project except whatever single snapshots people happen to have on their local machines. Local VCSs suffer from this same problem — whenever you have the entire history of the project in a single place, you risk losing everything.

**Distributed Version Control Systems**

This is where Distributed Version Control Systems (DVCSs) step in. In a DVCS (such as Git, Mercurial, Bazaar or Darcs), clients don’t just check out the latest snapshot of the files; rather, they fully mirror the repository, including its full history. Thus, if any server dies, and these systems were collaborating via that server, any of the client repositories can be copied back up to the server to restore it. Every clone is really a full backup of all the data.



Furthermore, many of these systems deal pretty well with having several remote repositories they can work with, so you can collaborate with different groups of people in different ways simultaneously within the same project. This allows you to set up several types of workflows that aren’t possible in centralized systems, such as hierarchical models.

**2. Problem Statement**

# **Why Do We Need It?**

Complex code development is impossible without a Version Control System (VCS). Version Control is used to track and control changes to source code. It is an essential tool to ensure the integrity of the codebase.

## Why is Version Control Essential?

* Easy Modification of Codebase
* Reverting Error
* Improves visibility.
* Accelerates product delivery.

**3.Objective**

Version control system keeps track on changes made on a particular software and take a snapshot of every modification .Let’s suppose if a team of developer add some new functionalities in a application and the updated version is not working properly so as version control system keeps track of our work so with the help of version control system we can omit the new changes and continue with the previous version .

**Benefits of the version control system:**

•Enhances the project development speed by providing efficient collaboration,

•Leverages the productivity, expedites product delivery, and skills of the employees through better communication and assistance,

•Reduce possibilities of errors and conflicts meanwhile project development through traceability to every small change,

•Employees or contributors of the project can contribute from anywhere irrespective of the different geographical locations through this VCS,

•For each different contributor to the project, a different working copy is maintained and not merged to the main file unless the working copy is validated. The most popular example is Git, Helix core, Microsoft TFS,

•Helps in recovery in case of any disaster or contingent situation,

•Informs us about Who, What, When, Why changes have been made.

4. Concepts and Commands

* **Here are some commands which is used in GIT.**

**git config --global user.name “name”:**

This command sets the author’s name to be used with your commits.

Graphical user interface, text

Description automatically generated

**git config --global user.email “email”:**

This command sets the author’s email address to be used with your commits.

Text

Description automatically generated

**git --version :**

It is used to display the version of git bash.

A screenshot of a computer

Description automatically generated with medium confidence

**pwd:**

Gives the full pathname of the current working directory to the standard output

Shape

Description automatically generated with medium confidence

**git init :**

Used to initialize the repository.

Text

Description automatically generated

**touch “filename”:**

It creates a new file.

Text

Description automatically generated

**Vi “filename”:**

It creates and add content to a file.

Graphical user interface

Description automatically generated

**git add –a:**

Moves changes from the working directory to the staging area.



**git commit -m “info of commit”:**

This command records or snapshots the file permanently in the version history.

Text

Description automatically generated

**ls :**

It gives the name of file present in the current folder.

Text

Description automatically generated with medium confidence

**git status:**

This command lists all the files that have to be committed.

**Text

Description automatically generated**

**rm -rf .git:**

This command is used delete the .git hidden folder.

Text

Description automatically generated with medium confidence

**git rm [file] :**

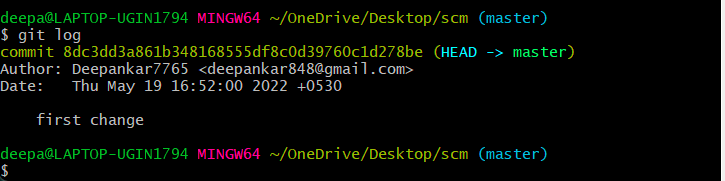
This command deletes the file from your working directory and stages the deletion.

Text

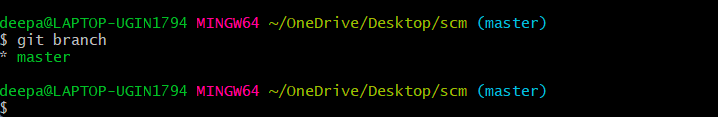
Description automatically generated

**git log :**

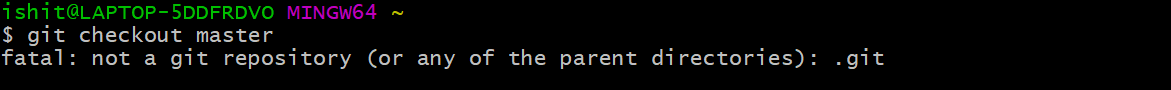
This command is used to list the version history for the current branch.



**git branch:**

This command lists all the local branches in the current repository.

**git branch [branch name] :**

This command creates a new branch.

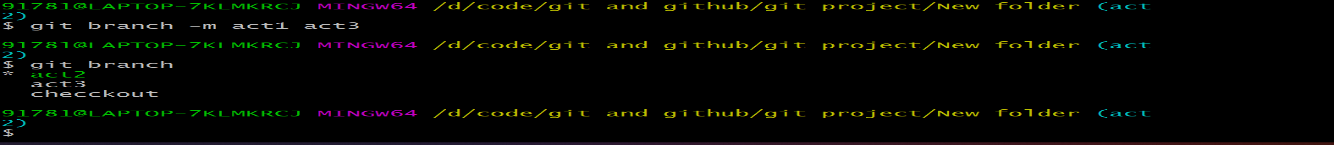
**git checkout [branch name] :**

This command is used to switch from one branch to another.A screenshot of a computer

Description automatically generated with medium confidence

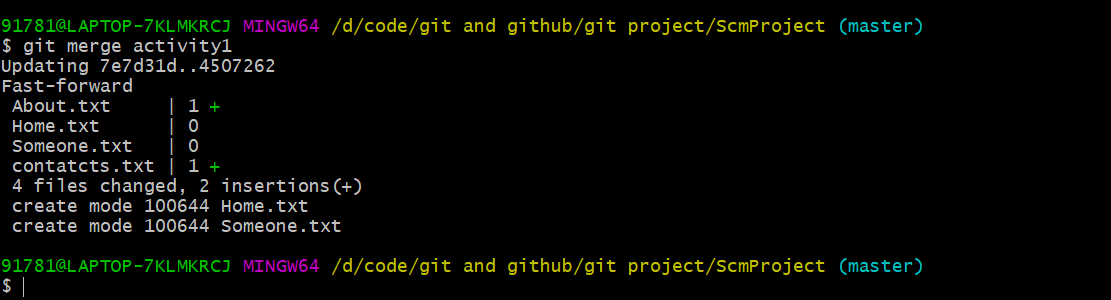
**git branch -m [branch name] :**

This command is used to rename the branch.

****

**git merge [branch name] :**

This command is used to merge a command.



**git branch -d [branch name] :**

This command is used to soft delete a branch.

Text

Description automatically generated

**git branch -D [branch name] :**

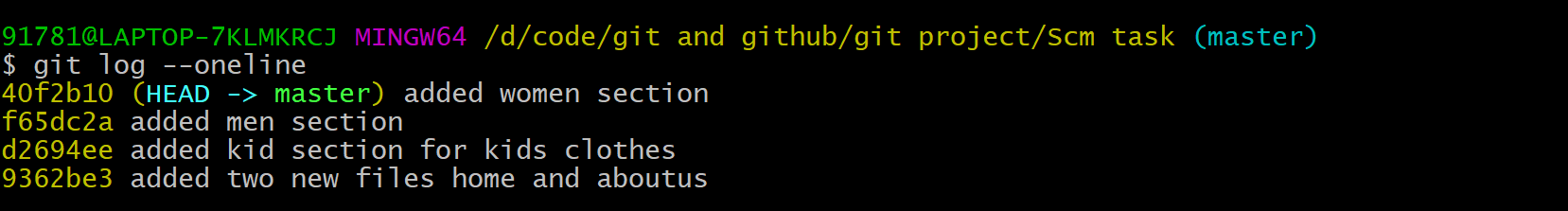
When you want to delete a branch without merging it. So, we use this command.

Text

Description automatically generated

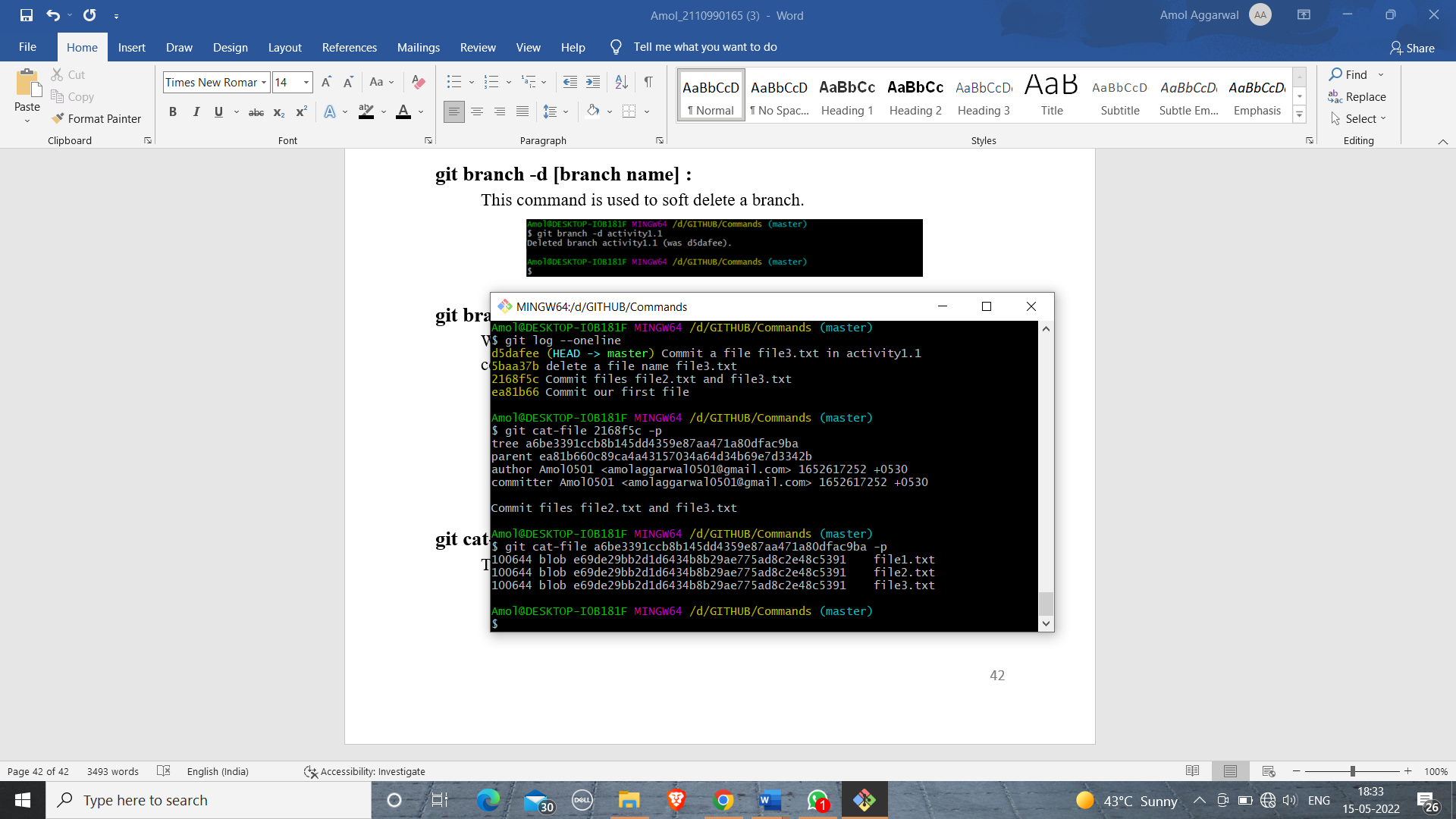
**git log --oneline :**

It gives the checkout information in one line.



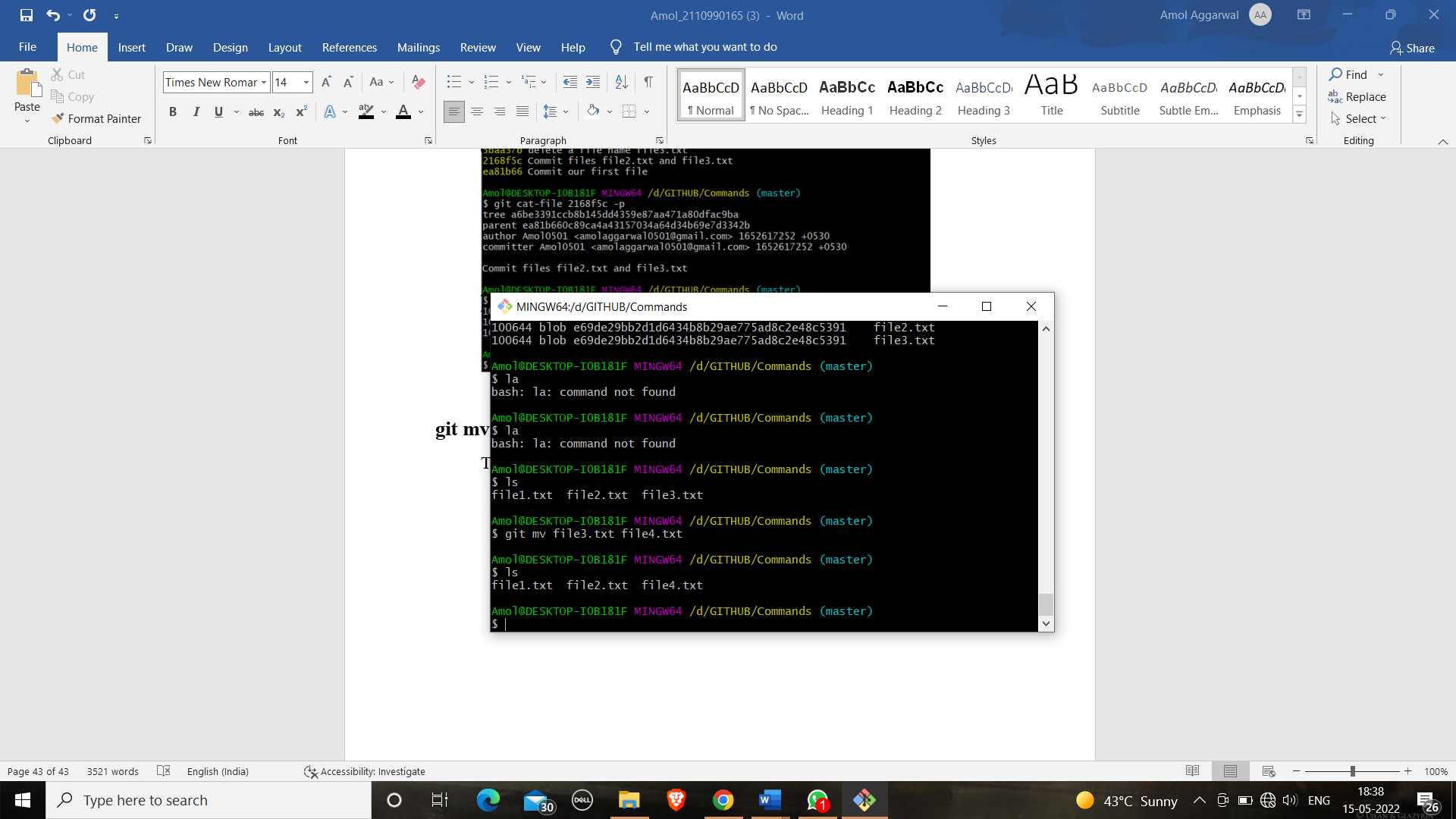
**git cat-file [checksome] -p :**

This command is used to see the commits and content using checksome.



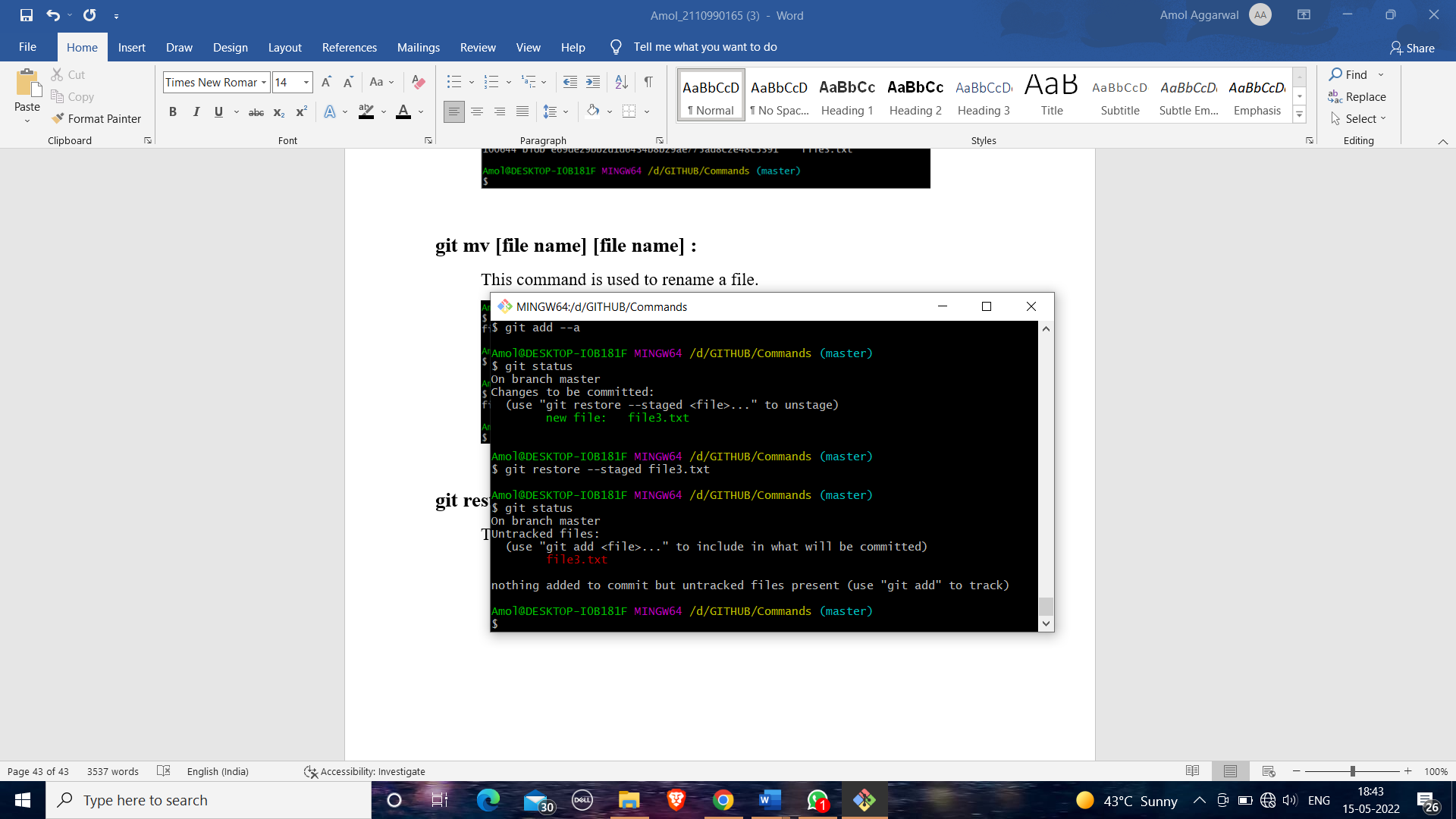
**git mv [file name] [file name] :**

This command is used to rename a file.



**git restore --staged [file name] :**

This command is used to remove from staging area.



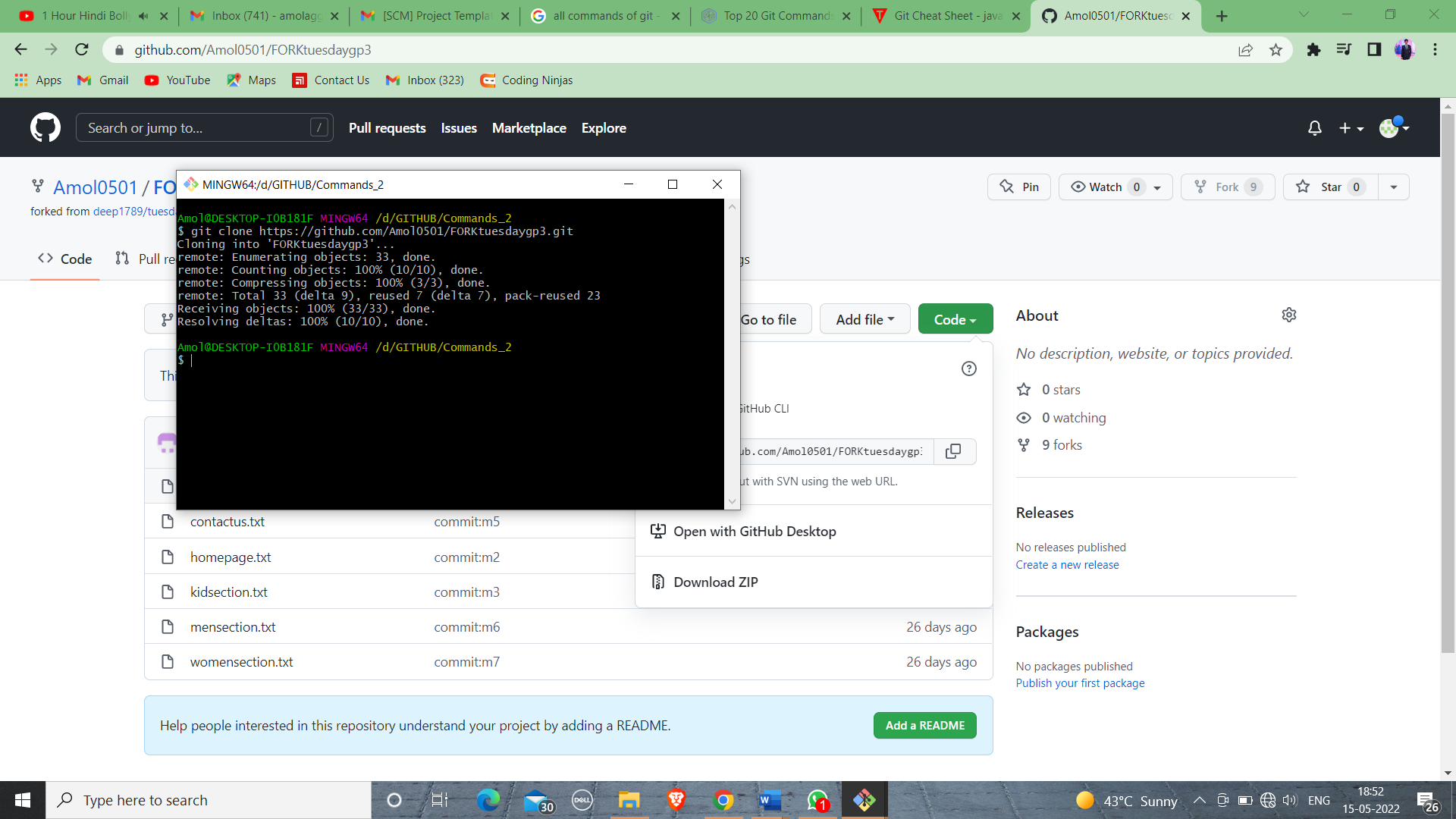
**git log --oneline --graph :**

It gives the checkout in the form of network graph.



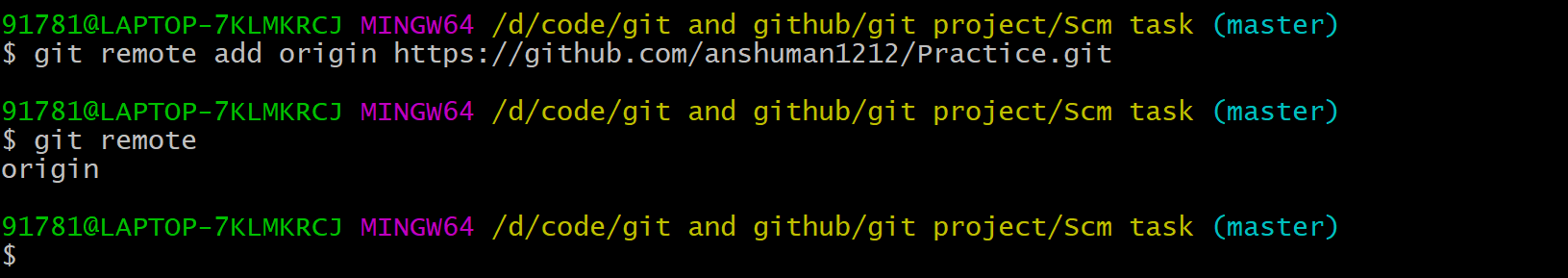
**git clone [URL] :**

This command is used to obtain a repository from an existing URL.



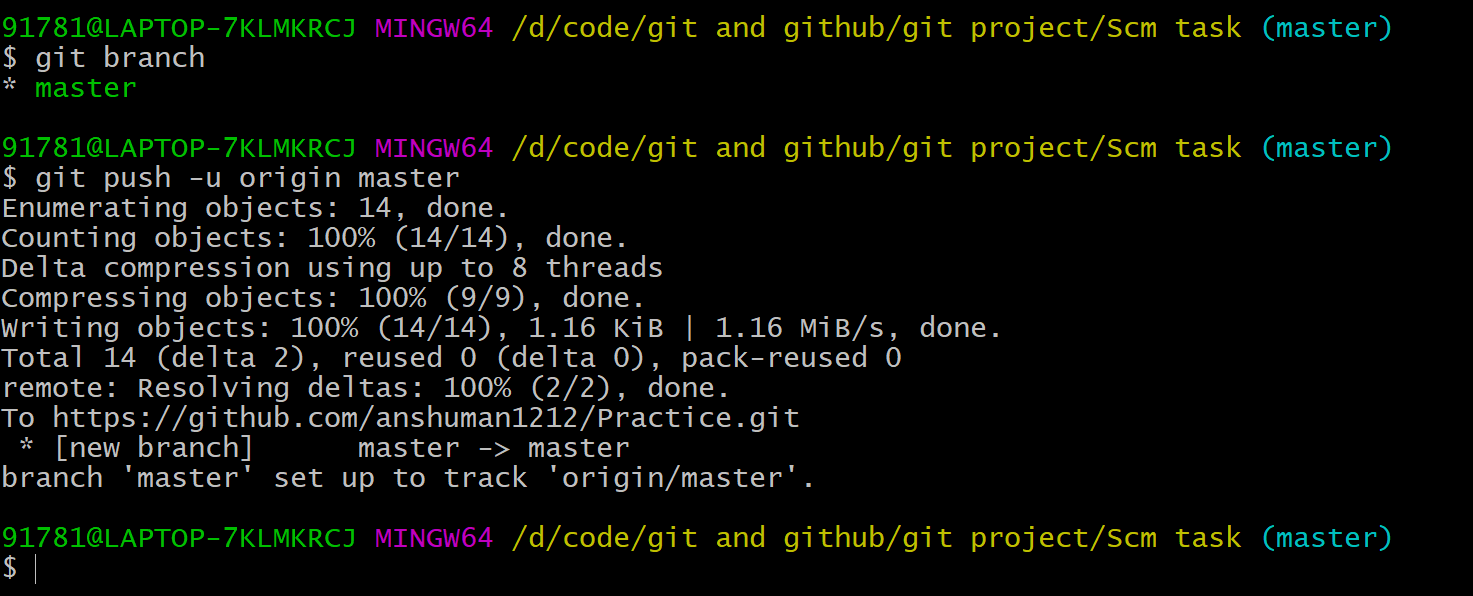
**git remote:**

This command is used to connect your local repository to the remote server.



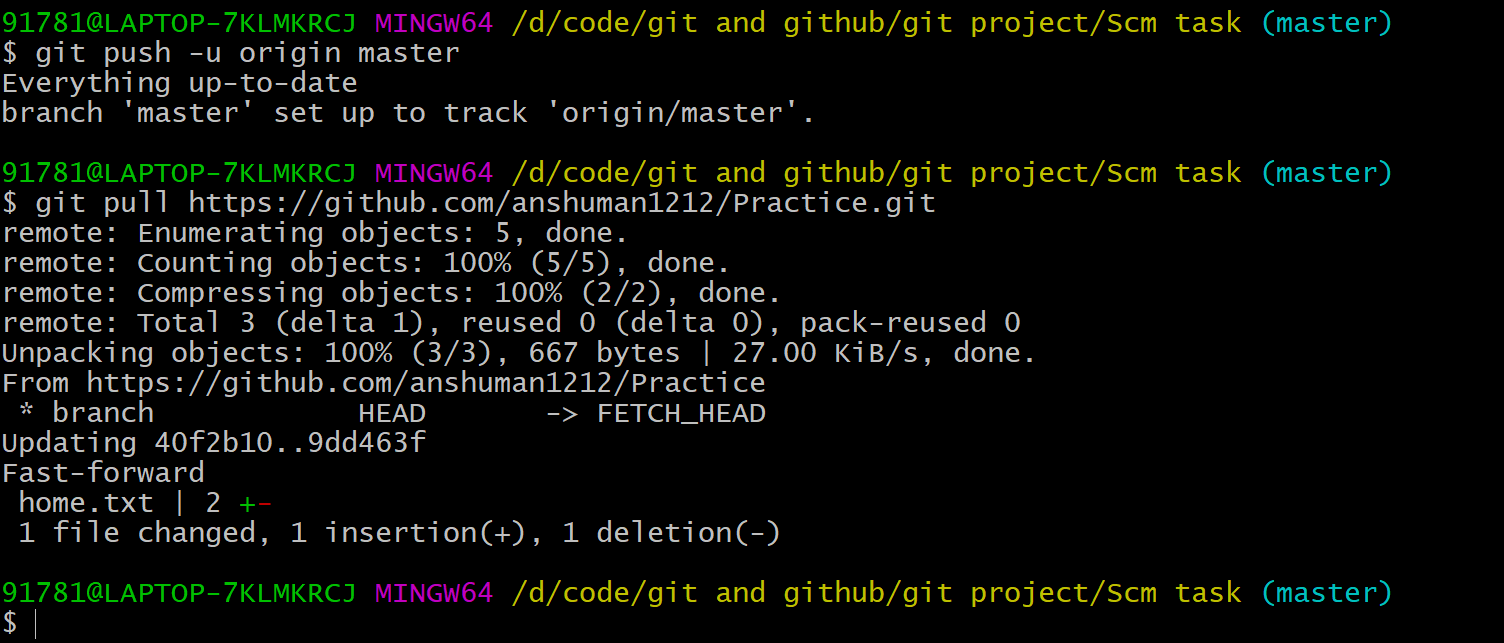
**git push :**

This command is used to push our repository.



**git pull :**

This command is used to pull a repository.



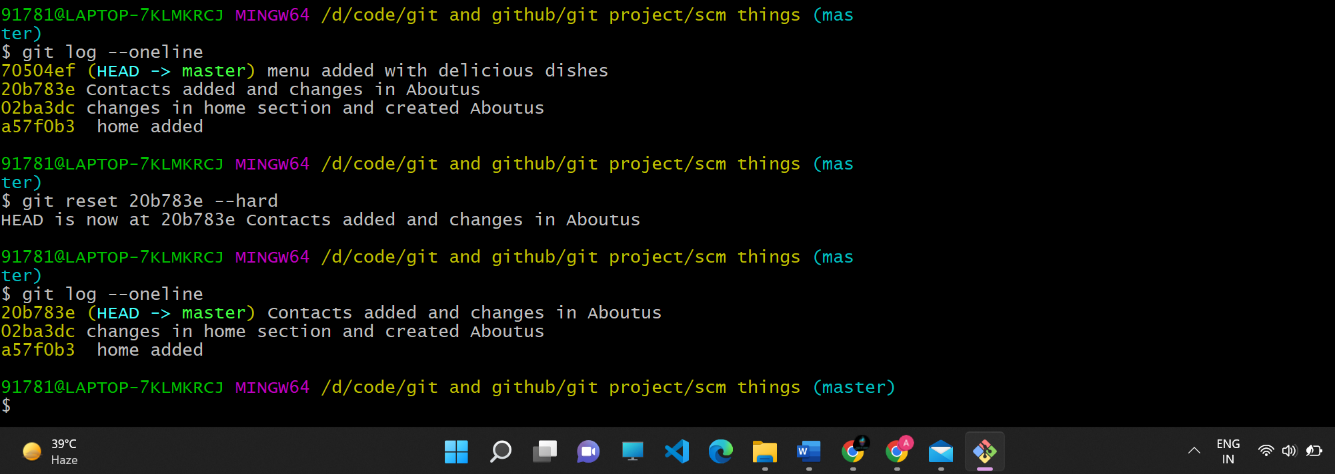
**git remote remove [remote name] :**

This command is to remove the remote.



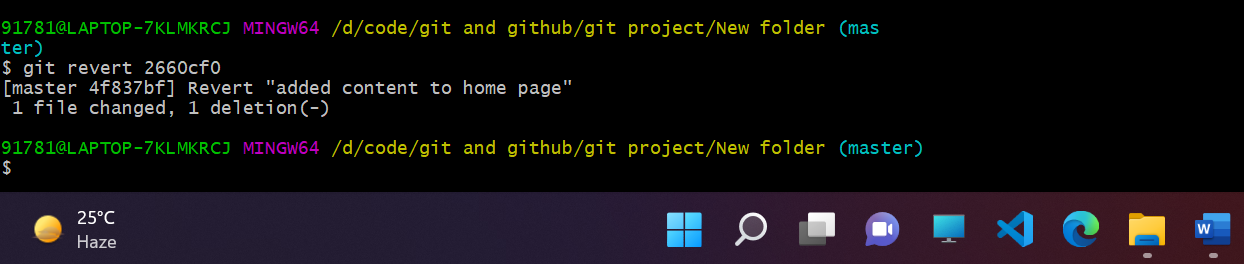
**git reset [checksome] :**

This command is used to restore the commit upto the checksome.



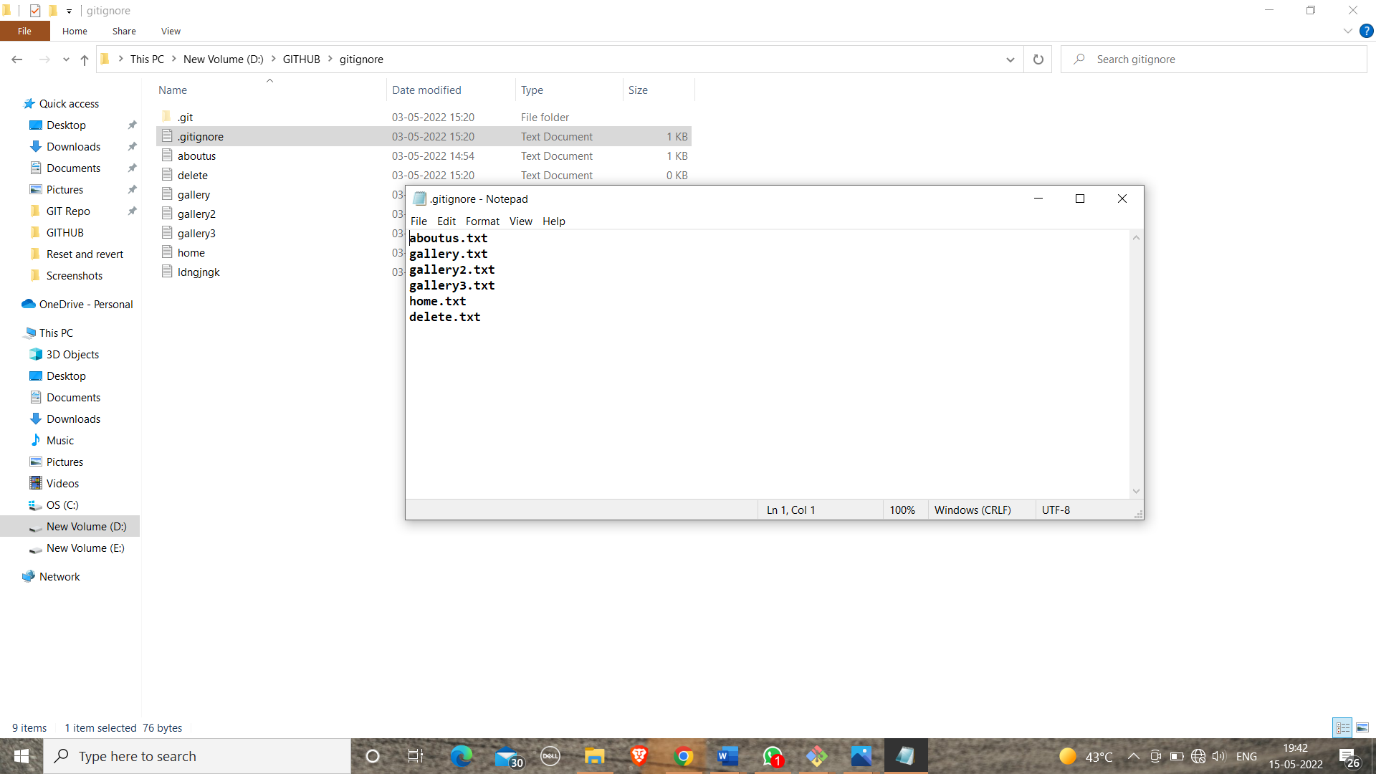
**git revert [checksome] :**

This command is used to restore the commit upto the checksome and added a new checksome of revert.



**.gitignore.txt :**

It is a text file used to ignore files which is not usable for us.



It ignore all the files which is inside .gitignote file.

## 5.Workflow and Discussion.

# During the development of the project we kept in mind the digital framework and financial network online. And we took an approach on how to create a infrastructure that enables cashless economy. The development of this project started with a collaboration of the team while implementing the basic framework and overview and later the project development started which was at last uploaded on Git-Hub via collaboration

|  |  |  |
| --- | --- | --- |
| **S.No.** |  |  |
| 1 | https://www.geeksforgeeks.org/version-control-systems/ |  |
| 2 | https://www.perforce.com/blog/vcs/what-is-version-control |  |
| 3 | https://medium.com/@lanceharvieruntime/version-control-why-do-we-need-it-1681f4888cec |  |
| 4 | https://en.wikipedia.org/wiki/Git |  |
| 5 | https://git-scm.com/ |  |
| 6 | https://www.simplilearn.com/tutorials/git-tutorial/git-vs-github |  |

6. Reference