

**Git and github**

**Assignment**



February 23, 2023

JSpiders

Basavanagudi

**GitHub Theory**

## Version Control

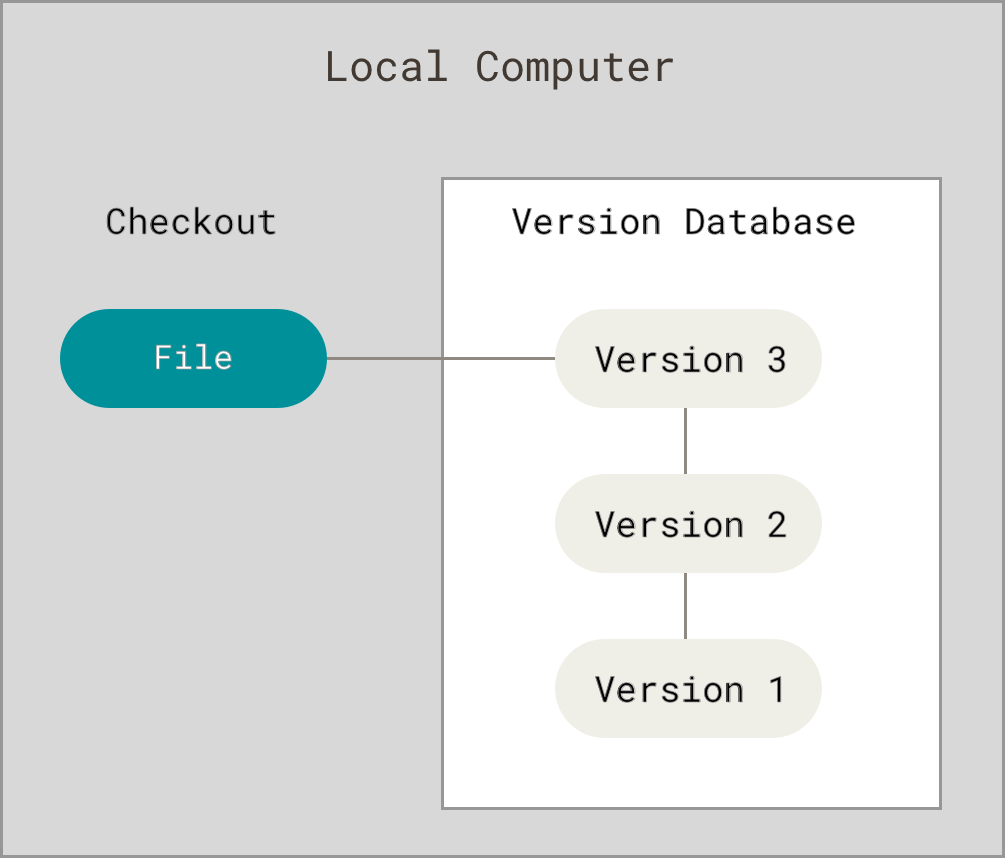
What is “version control”, and why should you care? Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later. For the examples in this book, you will use software source code as the files being version controlled, though in reality 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.

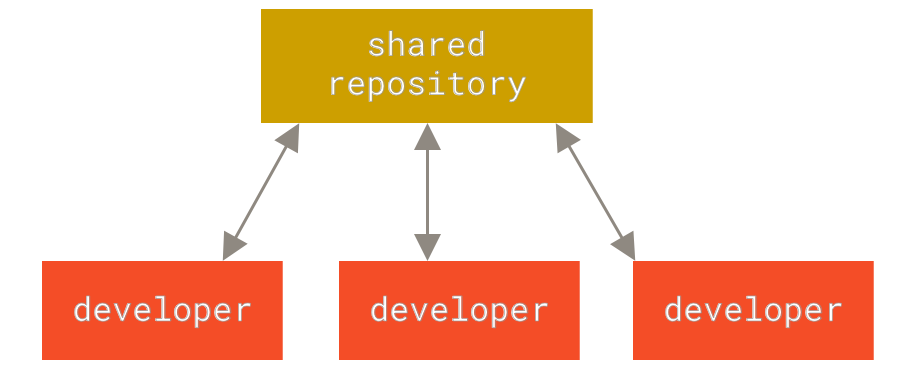


*Figure 1. Local version control*

One of the most popular VCS tools was a system called RCS, which is still distributed with many computers today. [RCS](https://www.gnu.org/software/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.



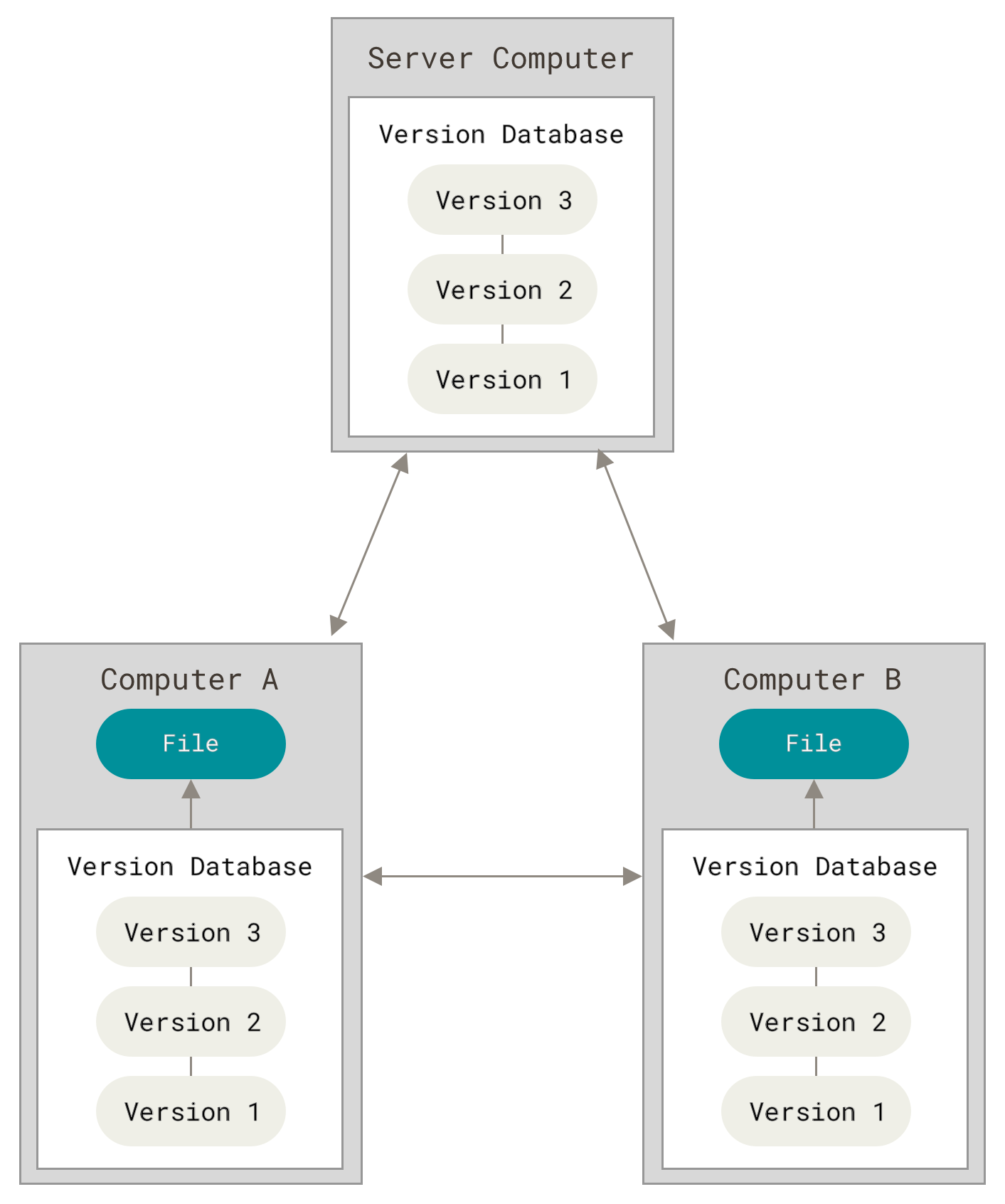
*Figure 2. Centralized version control*

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.



*Figure 3. Distributed version control*

**What is Git Clone?**

The git clone command is used to create a copy of a specific repository or branch within a repository. When you clone a repo you get a copy of the entire history of the repo. The command used for cloning any repository is:

**git clone <repository-link>**

**HTTP vs SSH**

**SSH**

SSH (Secure Shell) is a public-key cryptography protocol that ensures no one can intercept or change the data during the transfer. Since it is more difficultto set up, it is not as widespread as HTTPS, but it offers greater data integrity and security.

However, firewalls on some systems refuse to allow SSH connections on the default port, which can further complicate the setup. Additionally, some operating systems don't have SSH clients installed by default.

**Why Use SSH for Git?**

The purpose of establishing an SSH connection is to encrypt data exchanged between the client and the server. SSH connections are based on a key pair - a private key on a remote server and the corresponding public key on the local system.

Using SSH keys means there is no need to provide the username and password for each action, for example, pushing or pulling changes or signing commits.

**HTTPS**

HTTPS (Hyper Text Transfer Protocol Secure) is a more widespread network protocol that uses SSL/TLS data encryption. Since it is easier to configure than SSH, HTTPS is more common but provides a lower data security level since it doesn't use public-key cryptography.

Git with HTTPS uses token-based authentication to establish connections on port 443 via the Public/Private Pair authentication mode. Port 443 is open in almost every firewall, which isn’t always the case for SSH.

The downside of using HTTPS is that every action, such as git fetch, git pull, or git push asks for your username and password.

**Why Use HTTPS for Git?**

The main purpose of HTTPS is to allow secure data transfers between the client and server. HTTPS facilitates Git setup as there is no need to create SSH keys for each machine from which you want to access the repository.

The authentication is performed through a Personal Access Token, which acts as a unique password and allows users to additionally secure their account with 2FA.

**The benefits of using HTTPS for Git are:**

**Simple setup**. HTTPS is easy to set up, requiring only the repo URL and the clone command.

**Availability**. HTTPS is available on every operating system and has very few firewall restrictions.

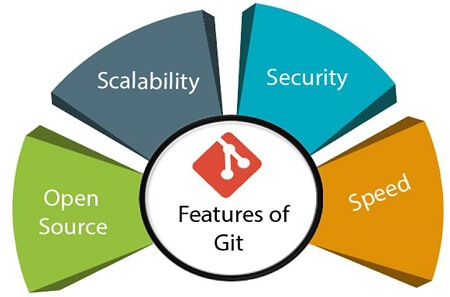
**Portability.** Access the repository from any machine by providing the username and password/token.

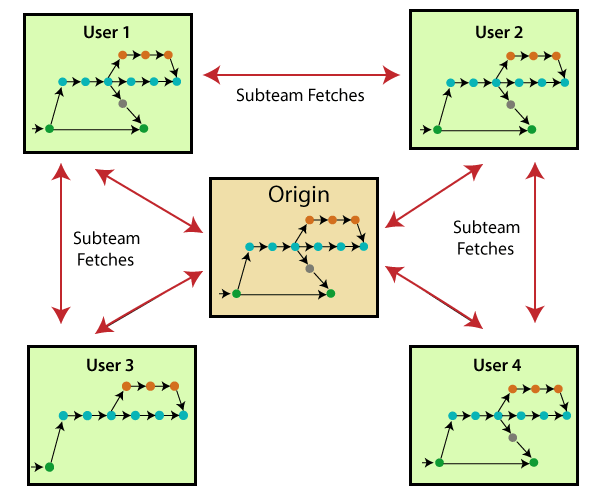
**Local Repository and Remote Repository**

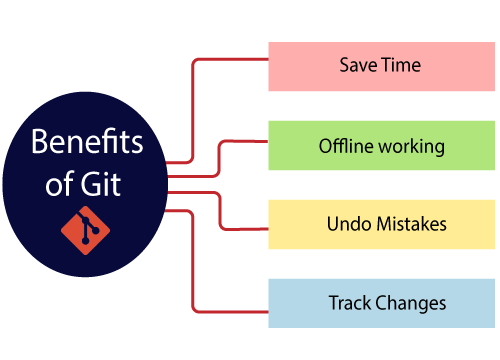
Local repositories reside on the computers of team members.

Remote repositories are hosted on a server that is accessible for all team members -most likely on the internet or on a local network.

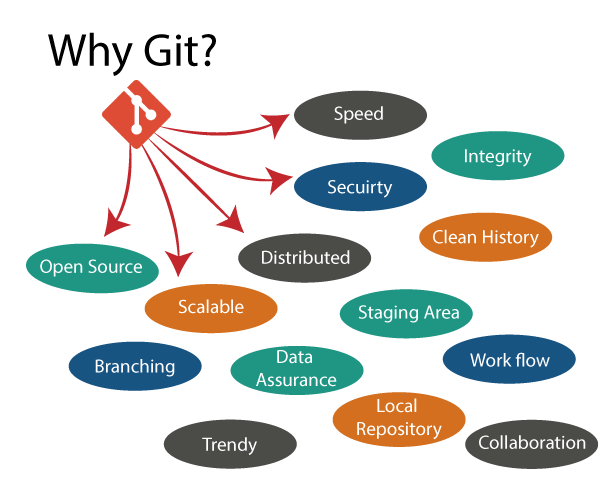
Technically, a remote repository doesn't differ from a local one: it contains branches, commits, and tags just like a local repository.







below diagram shows the **Advantages** of git



# **Git Tools**

To explore the robust functionality of Git, we need some tools. Git comes with some of its tools like Git Bash, Git GUI to provide the interface between machine and user. It supports inbuilt as well as third-party tools.

Git comes with built-in GUI tools like **git bash**, **git-gui**, and **gitk** for committing and browsing. It also supports several third-party tools for users looking for platform-specific experience.

### GitBash

Git Bash is an application for the Windows environment. It is used as Git command line for windows. Git Bash provides an emulation layer for a Git command-line experience. Bash is an abbreviation of **Bourne Again Shell**. Git package installer contains Bash, bash utilities, and Git on a Windows operating system.

# **Git Terminology**

Git is a tool that covered vast terminology and jargon, which can often be difficult for new users, or those who know Git basics but want to become Git masters. So, we need a little explanation of the terminology behind the tools. Let's have a look at the commonly used terms.

**Some commonly used terms are:**

### [Branch](https://www.javatpoint.com/git-branch)

A branch is a version of the repository that diverges from the main working project. It is an essential feature available in most modern version control systems. A Git project can have more than one branch. We can perform many operations on Git branch-like rename, list, delete, etc.

Cherry-picking in Git is meant to apply some commit from one branch into another branch. In case you made a mistake and committed a change into the wrong branch, but do not want to merge the whole branch. You can revert the commit and cherry-pick it on another branch.

### [Clone](https://www.javatpoint.com/git-clone)

The **git clone** is a Git command-line utility. It is used to make a copy of the target repository or clone it. If I want a local copy of my repository from GitHub, this tool allows creating a local copy of that repository on your local directory from the repository URL.

### [Fetch](https://www.javatpoint.com/git-fetch)

It is used to fetch branches and tags from one or more other repositories, along with the objects necessary to complete their histories. It updates the remote-tracking branches.

### [Master](https://www.javatpoint.com/git-origin-master)

Master is a naming convention for Git branch. It's a default branch of Git. After cloning a project from a remote server, the resulting local repository contains only a single local branch. This branch is called a "master" branch. It means that "master" is a repository's "default" branch.

### [Merge](https://www.javatpoint.com/git-merge)

Merging is a process to put a forked history back together. The git merge command facilitates you to take the data created by git branch and integrate them into a single branch.

### [Origin](https://www.javatpoint.com/git-origin-master)

In Git, "origin" is a reference to the remote repository from a project was initially cloned. More precisely, it is used instead of that original repository URL to make referencing much easier.

### [Pull/Pull Request](https://www.javatpoint.com/git-pull)

The term Pull is used to receive data from GitHub. It fetches and merges changes on the remote server to your working directory. The **git pull command** is used to make a Git pull.

Pull requests are a process for a developer to notify team members that they have completed a feature. Once their feature branch is ready, the developer files a pull request via their remote server account. Pull request announces all the team members that they need to review the code and merge it into the master branch.

### [Push](https://www.javatpoint.com/git-push)

The push term refers to upload local repository content to a remote repository. Pushing is an act of transfer commits from your local repository to a remote repository. Pushing is capable of overwriting changes; caution should be taken when pushing.

### [Remote](https://www.javatpoint.com/git-remote)

In Git, the term remote is concerned with the remote repository. It is a shared repository that all team members use to exchange their changes. A remote repository is stored on a code hosting service like an internal server, GitHub, Subversion and more.

In case of a local repository, a remote typically does not provide a file tree of the project's current state, as an alternative it only consists of the .git versioning data.

### [Repository](https://www.javatpoint.com/git-repository)

In Git, Repository is like a data structure used by VCS to store metadata for a set of files and directories. It contains the collection of the file as well as the history of changes made to those files. Repositories in Git is considered as your project folder. A repository has all the project-related data. Distinct projects have distinct repositories.

### [Tag](https://www.javatpoint.com/git-tag)

Tags make a point as a specific point in Git history. It is used to mark a commit stage as important. We can tag a commit for future reference. Primarily, it is used to mark a projects initial point like v1.1. There are two types of tags.

1. Light-weighted tag
2. Annotated tag

### [Git Revert](https://www.javatpoint.com/git-revert)

In Git, the term revert is used to revert some commit. To revert a commit, **git revert** command is used. It is an undo type command. However, it is not a traditional undo alternative.

### [Git Flow](https://www.javatpoint.com/git-flow)

GitFlow is a **branching model** for Git, developed by **Vincent Driessen**. It is very well organized to collaborate and scale the development team. Git flow is a collection of Git commands. It accomplishes many repository operations with just single commands.

### [Git Rm](https://www.javatpoint.com/git-rm)

In Git, the term rm stands for **remove**. It is used to remove individual files or a collection of files. The key function of git rm is to remove tracked files from the Git index. Additionally, it can be used to remove files from both the working directory and staging index.

### [Git Fork](https://www.javatpoint.com/git-fork)

A fork is a rough copy of a repository. Forking a repository allows you to freely test and debug with changes without affecting the original project.

Great use of using forks to propose changes for bug fixes. To resolve an issue for a bug that you found, you can:

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

# **12 Git Commands**

There are many different ways to use Git. Git supports many command-line tools and graphical user interfaces. The Git command line is the only place where you can run all the Git commands.

The following set of commands will help you understand how to use Git via the command line.

## Basic Git Commands

Here is a list of most essential Git commands that are used daily.

1. [Git Config command](https://www.javatpoint.com/git-commands#config-command)
2. [Git init command](https://www.javatpoint.com/git-commands#init-command)
3. [Git clone command](https://www.javatpoint.com/git-commands#clone-command)
4. [Git add command](https://www.javatpoint.com/git-commands#add-command)
5. [Git commit command](https://www.javatpoint.com/git-commands#commit-command)
6. [Git status command](https://www.javatpoint.com/git-commands#status-command)
7. [Git push Command](https://www.javatpoint.com/git-commands#push-command)
8. [Git pull command](https://www.javatpoint.com/git-commands#pull-command)
9. [Git Branch Command](https://www.javatpoint.com/git-commands#branch-command)
10. [Git Merge Command](https://www.javatpoint.com/git-commands#merge-command)

Let's understand each command in detail.

## 1) Git config command

This command configures the user. The Git config command is the first and necessary command used on the Git command line. This command sets the author name and email address to be used with your commits. Git config is also used in other scenarios.

**Syntax**

1. $ git config --global user.name "ImDwivedi1"
2. $ git config --global user.email "Himanshudubey481@gmail.com"

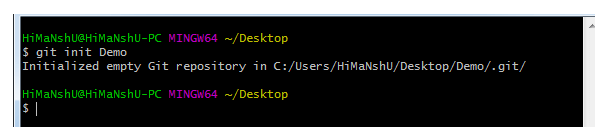
### 2) Git Init command

This command is used to create a local repository.

**Syntax**

1. $ git init Demo

The init command will initialize an empty repository. See the below screenshot.

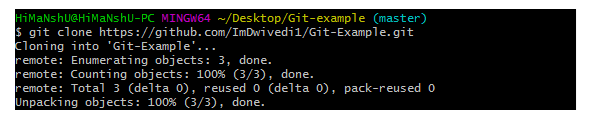


### 3) Git clone command

This command is used to make a copy of a repository from an existing URL. If I want a local copy of my repository from GitHub, this command allows creating a local copy of that repository on your local directory from the repository URL.

**Syntax**

1. $ git clone URL



### 4) Git add command

This command is used to add one or more files to staging (Index) area.

**Syntax**

To add one file

1. $ git add Filename

To add more than one file

1. $ git add\*

Git Commands

### 5) Git commit command

Commit command is used in two scenarios. They are as follows.

**Git commit -m**

This command changes the head. It records or snapshots the file permanently in the version history with a message.

**Syntax**

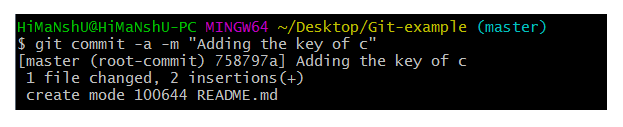
1. $ git commit -m " Commit Message"

**Git commit -a**

This command commits any files added in the repository with git add and also commits any files you've changed since then.

**Syntax**

1. $ git commit -a



### 6) Git status command

The status command is used to display the state of the working directory and the staging area. It allows you to see which changes have been staged, which haven't, and which files aren?t being tracked by Git. It does not show you any information about the committed project history. For this, you need to use the git log. It also lists the files that you've changed and those you still need to add or commit.

**Syntax**

1. $ git status



### 7) Git push Command

It is used to upload local repository content to a remote repository. Pushing is an act of transfer commits from your local repository to a remote repo. It's the complement to git fetch, but whereas fetching imports commits to local branches on comparatively pushing exports commits to remote branches. Remote branches are configured by using the git remote command. Pushing is capable of overwriting changes, and caution should be taken when pushing.

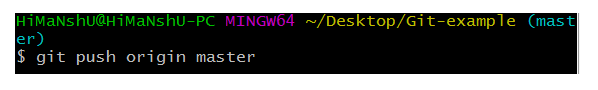
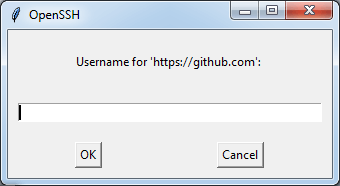
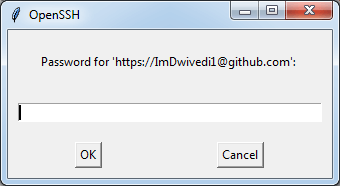
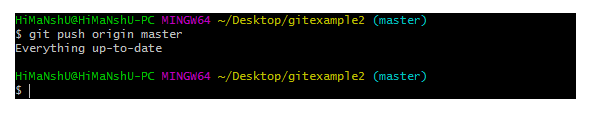
Git push command can be used as follows.

**Git push origin master**

This command sends the changes made on the master branch, to your remote repository.

**Syntax**

1. $ git push [variable name] master

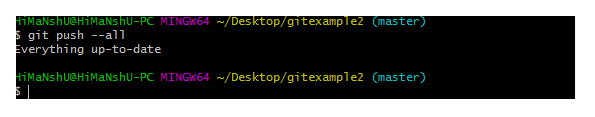
  
  
  


**Git push -all**

This command pushes all the branches to the server repository.

**Syntax**

1. $ git push --all

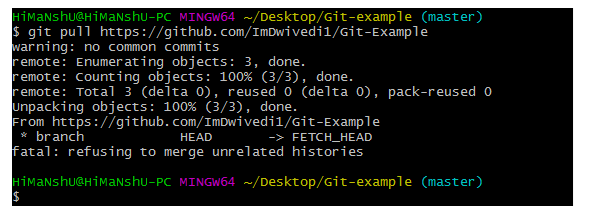


### 8) Git pull command

Pull command is used to receive data from GitHub. It fetches and merges changes on the remote server to your working directory.

**Syntax**

1. $ git pull URL



### 9) Git Branch Command

This command lists all the branches available in the repository.

**Syntax**

1. $ git branch



### 10) Git Merge Command

This command is used to merge the specified branches history into the current branch.

**Syntax**

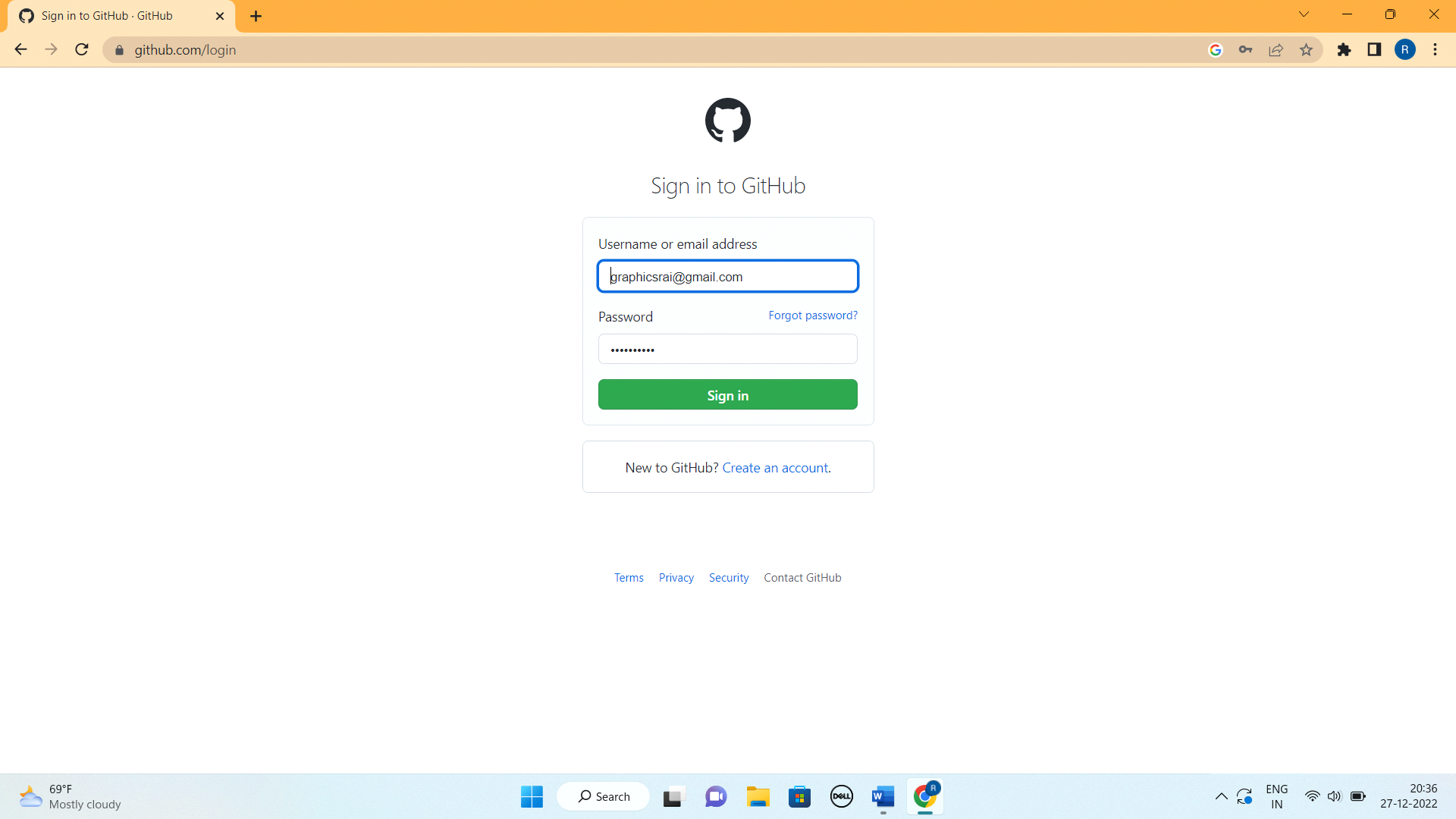
1. $ git merge Branch Name



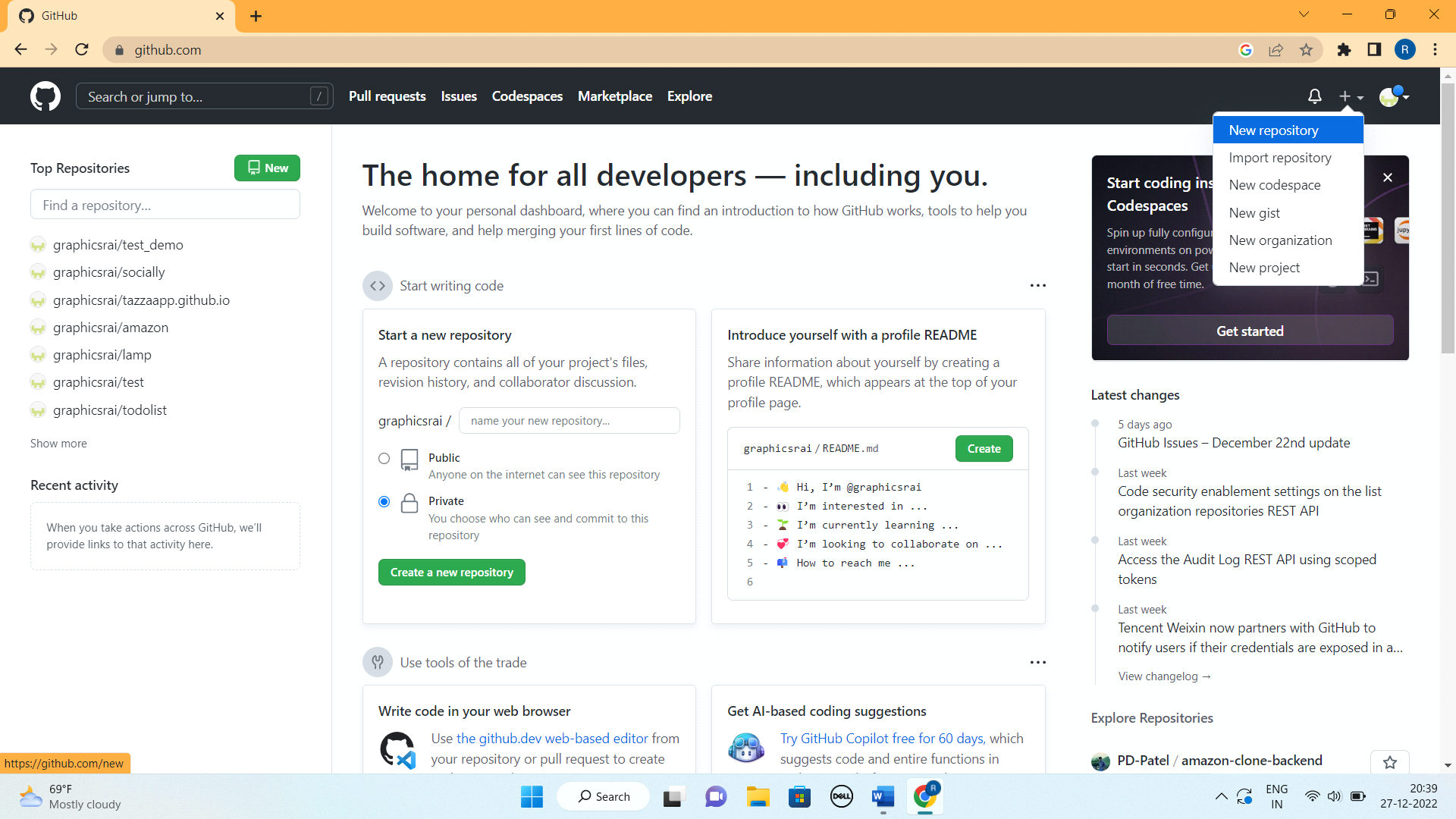
**GitHub Practical**

**How to Create a git repository steps are as follows.**

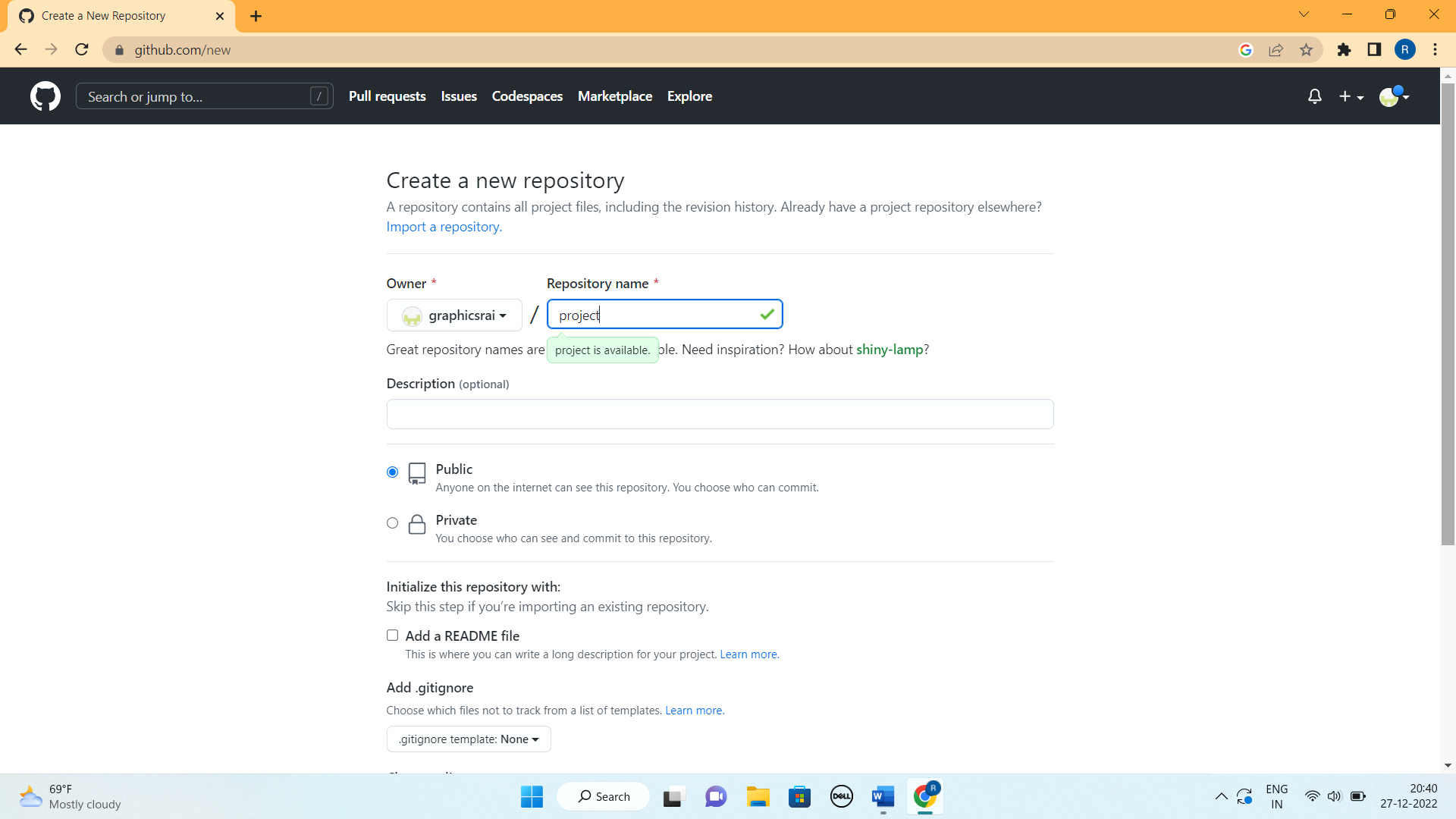
* **Step 1:** Login to [https://github.com](https://github.com/) account.



* **Step 2:**  Create a New repository by choosing + icon in right top of page



* **Step 3:** Give the repository name what you want.

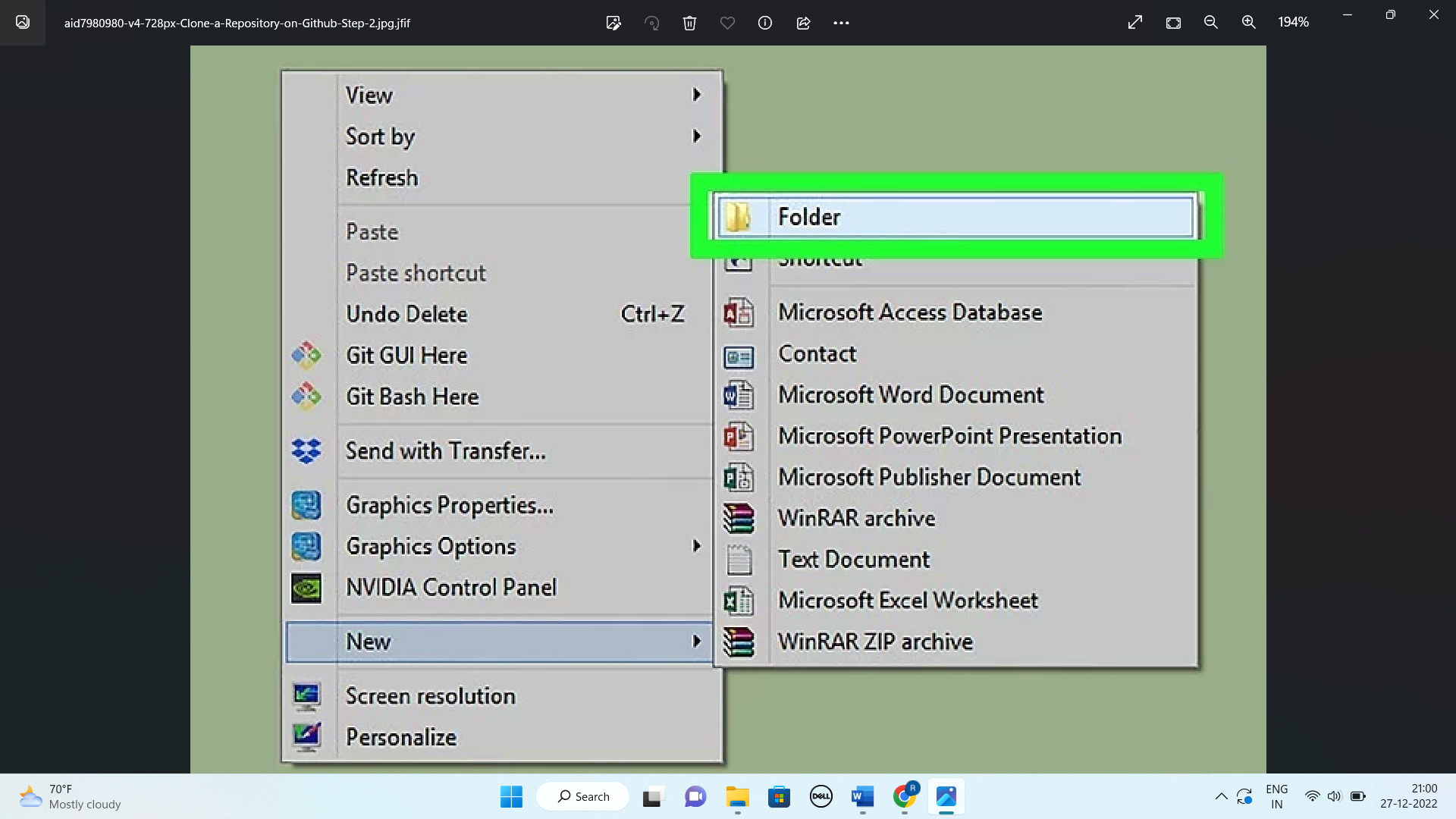


* **Step 4:** Now click on Create Repository Button. And it will create a new repository.

**How to clone a repository into your local repository steps are as follows.**

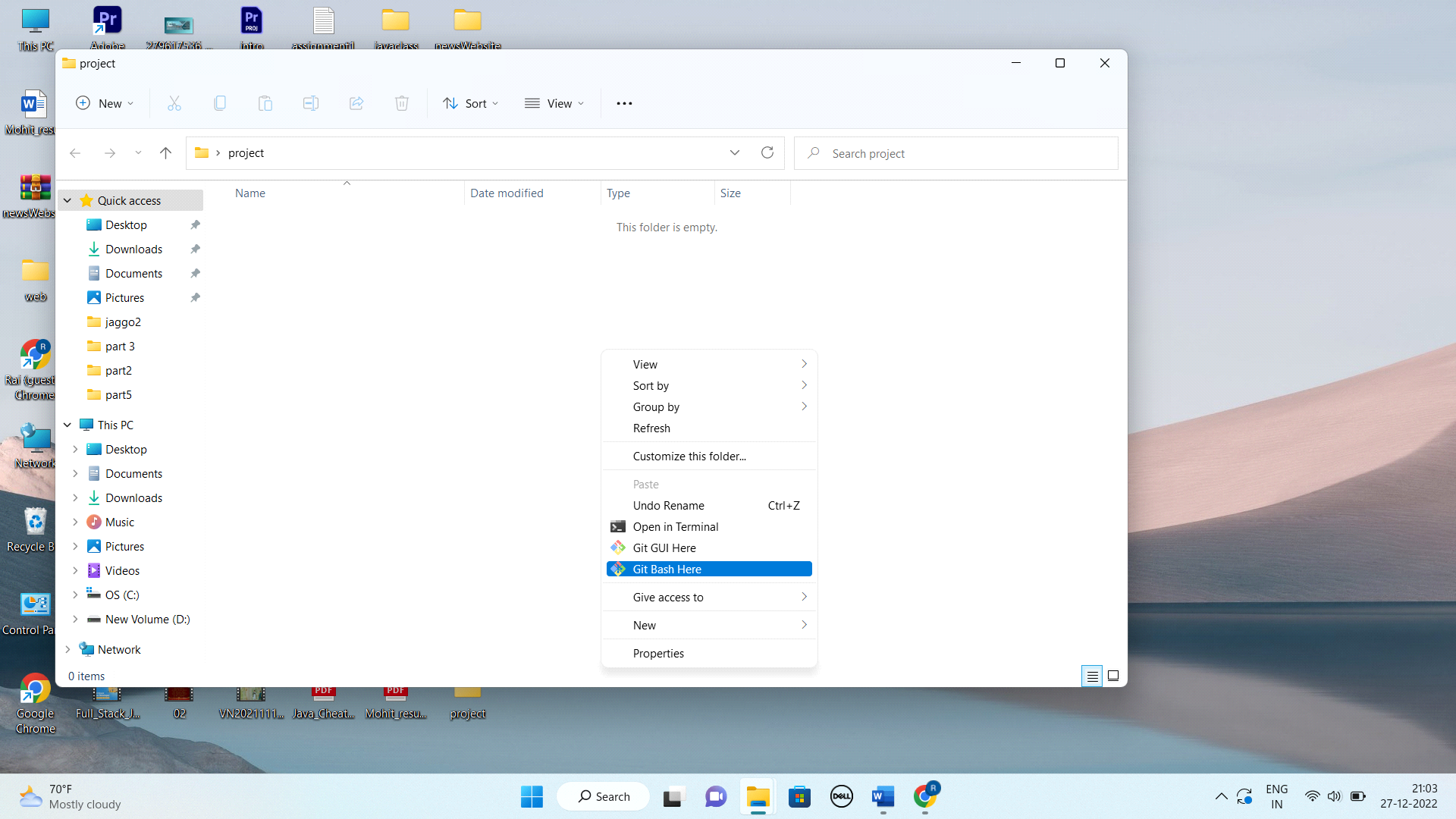
We can clone a repository in 3 ways.

* **Using the command line**
* **Step 1: Create a directory for your repository.** Navigate to a location of your choosing in your computer. Then right-click and select “New Folder”.

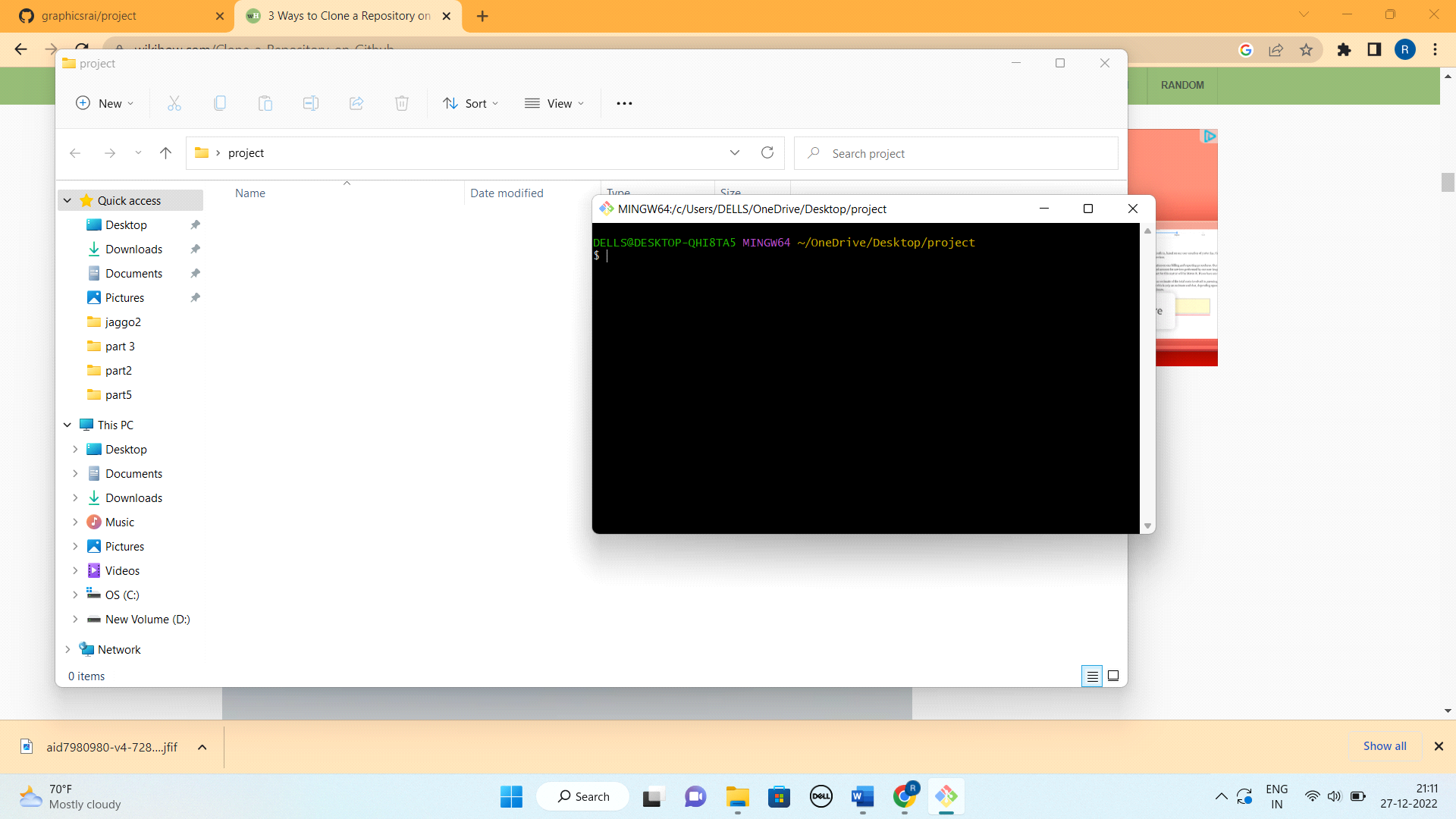


* **Step 2:** Open new created folder and right click inside it and choose **Git Bash Here** option.

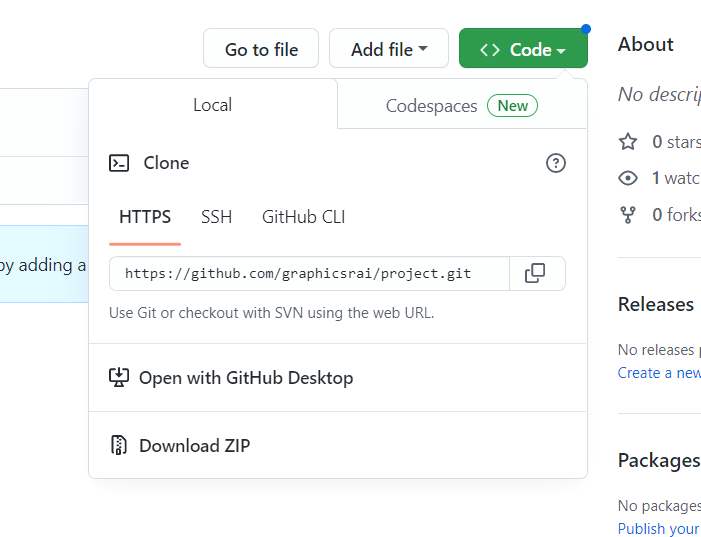
It will open git command line.



* **Step 3:** It will navigate to your target directory in the command line.



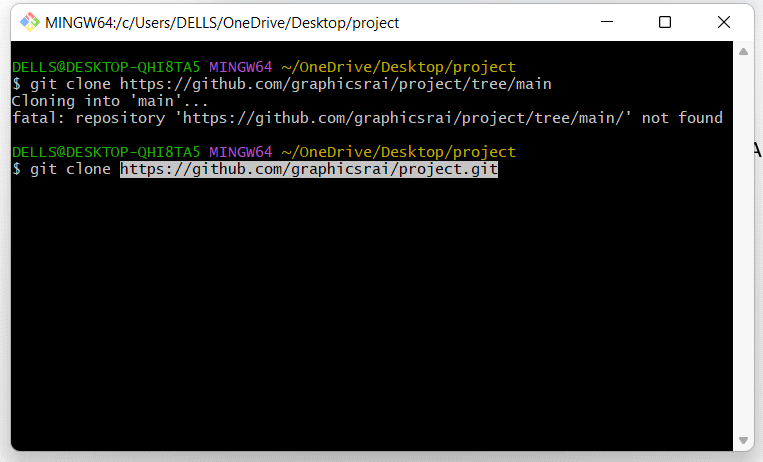
* **Step 4:** Copy the source location. Click the source location (Go to the Code button and inside code button click on HTTPS and copy the url.
* Go to the github account and copy the repository url.



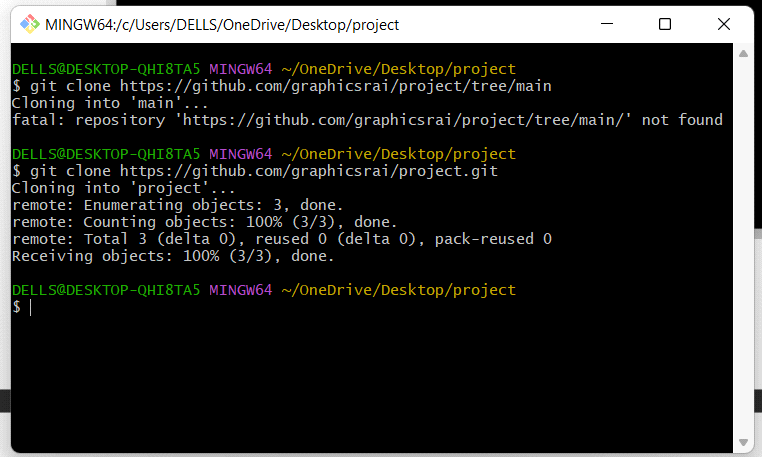
And go to the git bash command line again.

* **Step 5:**  And type the command in git bash command line. And paste the copied url followed by the command.

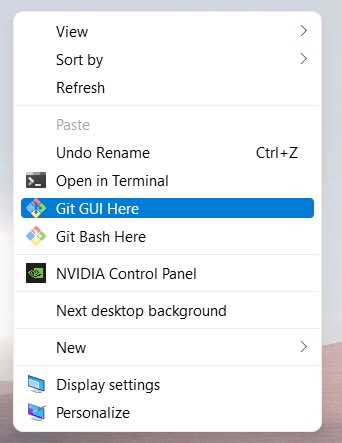
**$ git clone** [**https://github.com/graphicsrai/project.git**](https://github.com/graphicsrai/project.git)**. a**nd press enter.



It will clone your project to the local repository

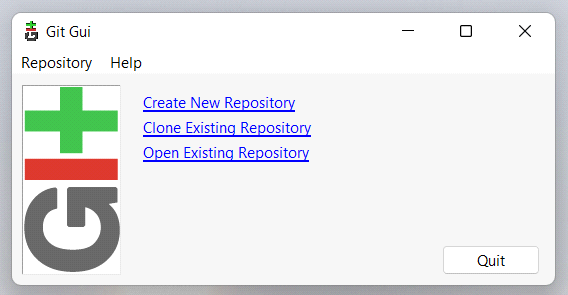


* **Using the Git GUI**
* **Step 1:** Copy the source location. Click the source location (Go to the Code button and inside code button click on HTTPS and copy the url.
* **Step 2:** Right click where you want to create the local repository folder and select the git GUI option. In my case am doing it on desktop.



**Step 3:** Click “Clone Repository”. This is the first option on the boot splash screen.

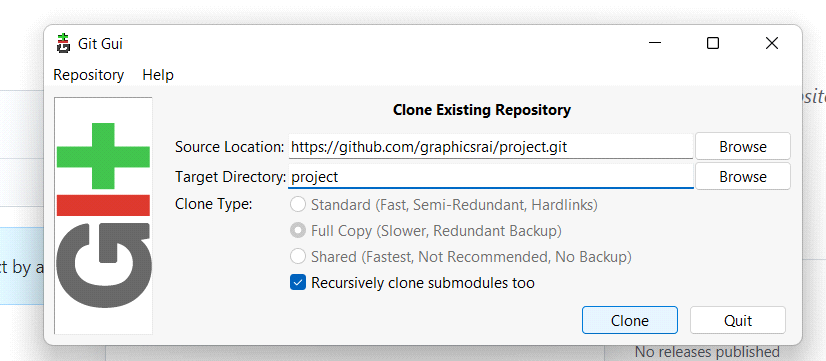
* You can also select “Clone” from “Repository” dropdown menu.



**Step 4:** Paste the copied like form GitHub in the Source Location,

and provide the folder name in which you want to clone the project in Target Directory.

And press clone. It will clone your project in to this folder and you can find this folder on desktop.

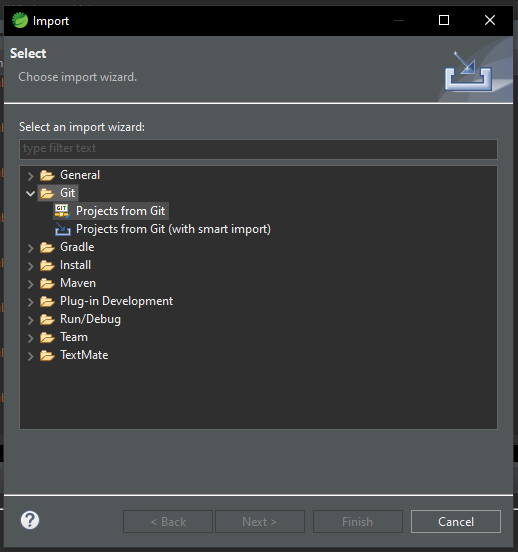


**3. Using the Spring tool suite**

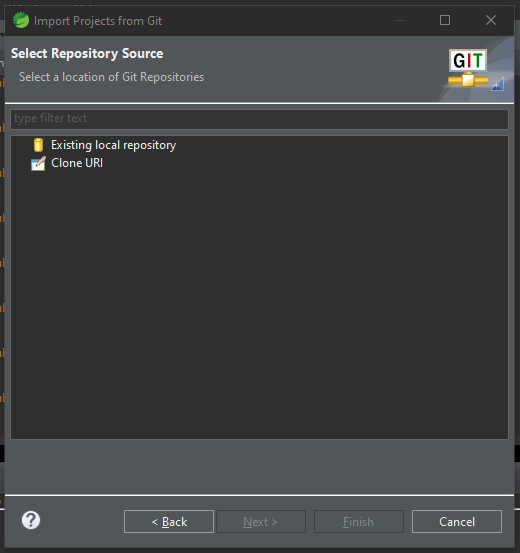
* **Step 1:** Copy the source location. Click the source location (Go to the Code button and inside code button click on HTTPS and copy the url.
* **Step 2:**  Open STS and click on File then go to import



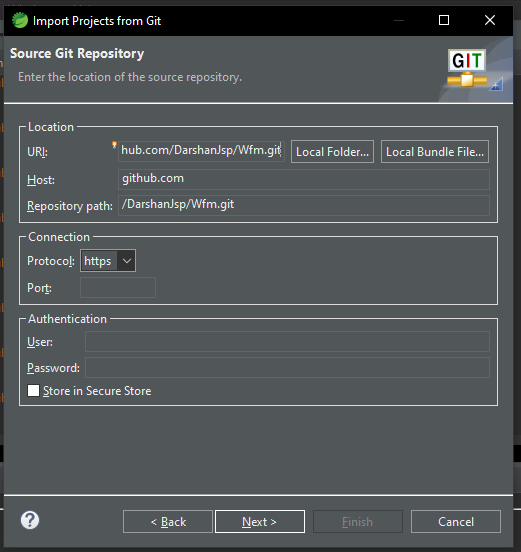
**Step 3:** Go to Git and select Projects from Git. and press next button.



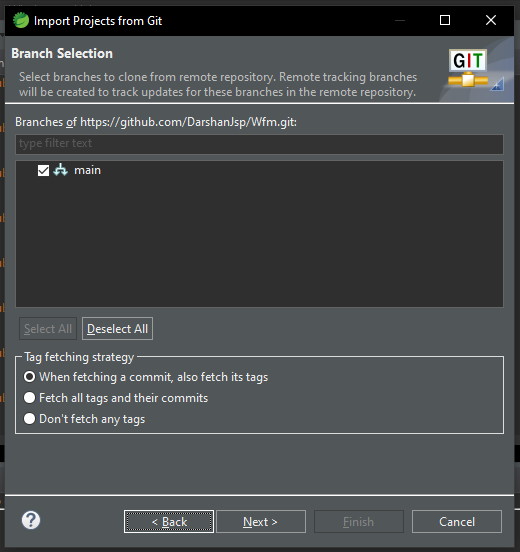
**Step 4:** Select Clone URI and press enter.



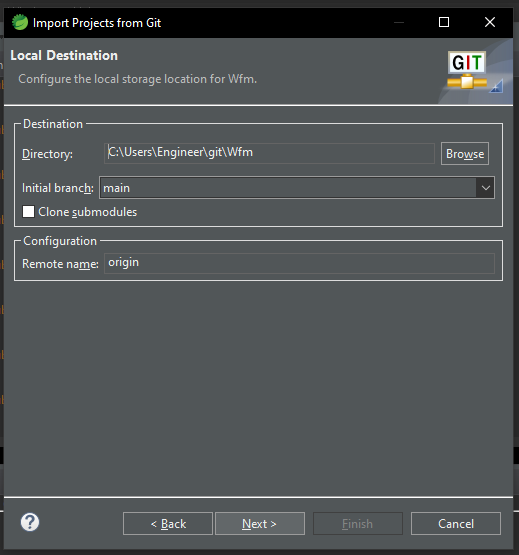
**Step 5:** Now paste URL copied from GitHub in URI box and press next



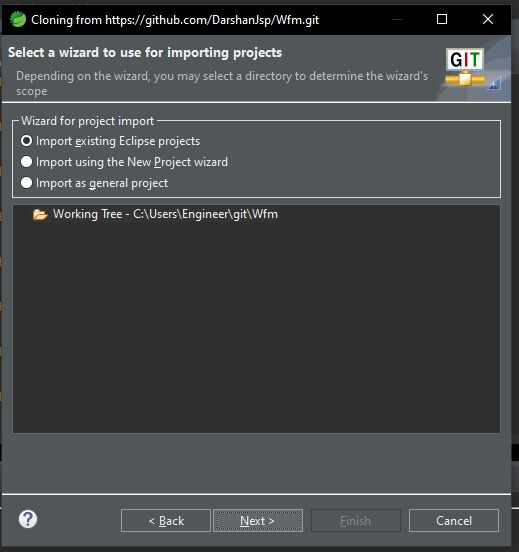
**Step 6:** Check main and press next.



**Step 7:** Change the directory where you want to clone the files from GitHub, and press next button.



**Step 8:** Press next.



It will clone your repository to the local provided directory. you can check in respected directory.