

## Version Control with Git:-

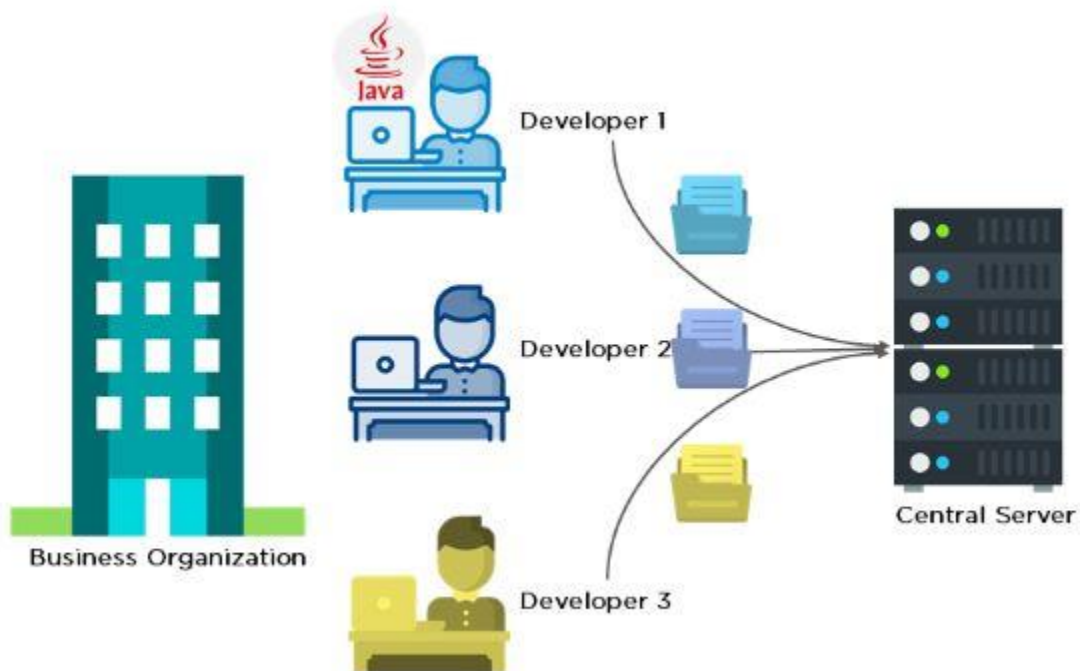
1. Introduction to version control
2. Basics of Git
3. Installing and Configuring Git
4. Common Git Commands
5. Branching and Merging Strategies
6. Working with Remote Repositories

### 1. Introduction to version control

## Getting Started with Git

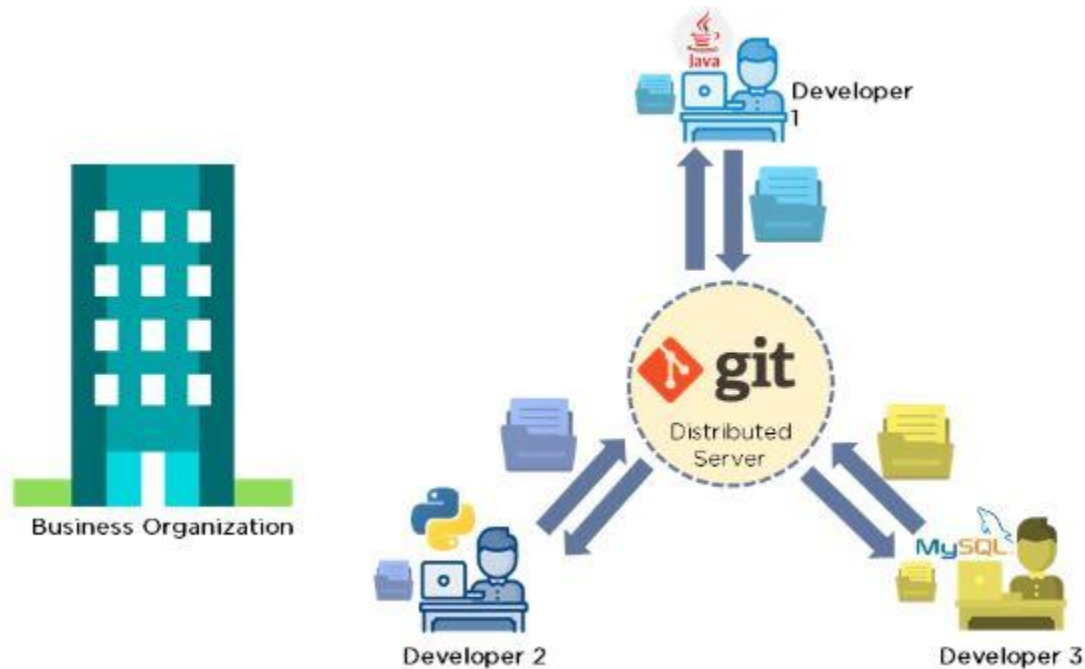
Before diving deep, let's explain a scenario before Git:

- Developers used to submit their codes to the central server without having copies of their own
- Any changes made to the source code were unknown to the other developers
- There was no communication between any of the developers



Now let's look at the scenario after Git:

- Every developer has an entire copy of the code on their local systems
- Any changes made to the source code can be tracked by others
- There is regular communication between the developers



## 1.1 Git Version Control System(VCS)

A version control system is a software that tracks changes to a file or set of files over time so that you can recall specific versions later. It also allows you to work together with other programmers.

The version control system is a collection of software tools that help a team to manage changes in a source code. It uses a special kind of database to keep track of every modification to the code.

Developers can compare earlier versions of the code with an older version to fix the mistakes.

### • Benefits of the Version Control System

The Version Control System is very helpful and beneficial in software development; developing software without using version control is unsafe. It provides backups for uncertainty. Version control systems offer a speedy interface to developers. It also allows software teams to preserve efficiency and agility according to the team scales to include more developers.

Some key benefits of having a version control system are as follows.

- Complete change history of the file
- Simultaneously working
- Branching and merging
- Traceability

# Types of Version Control System

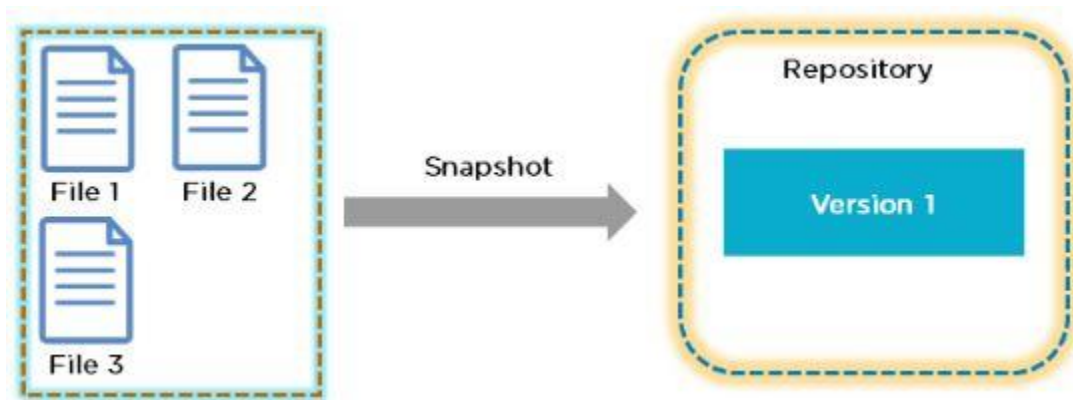
- a. Localized version Control System
- b. Centralized version control systems
- c. Distributed version control systems

## a. Localized Version Control Systems

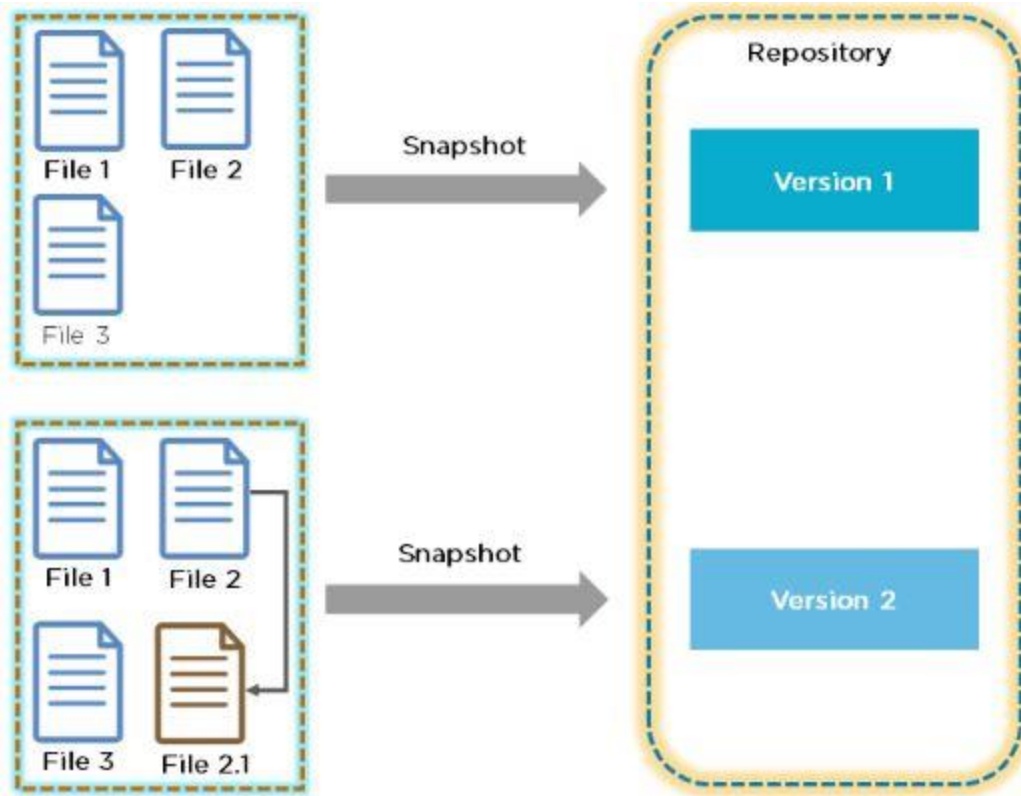
The localized version control method is a common approach because of its simplicity. But this approach leads to a higher chance of error. In this approach, you may forget which directory you're in and accidentally write to the wrong file or copy over files you don't want to.

To deal with this issue, programmers developed local VCSs that had a simple database. Such databases kept all the changes to files under revision control. A local version control system keeps local copies of the files.

The diagram below shows there are three files in the local system. A snapshot of these files are stored in the remote repository as Version 1.



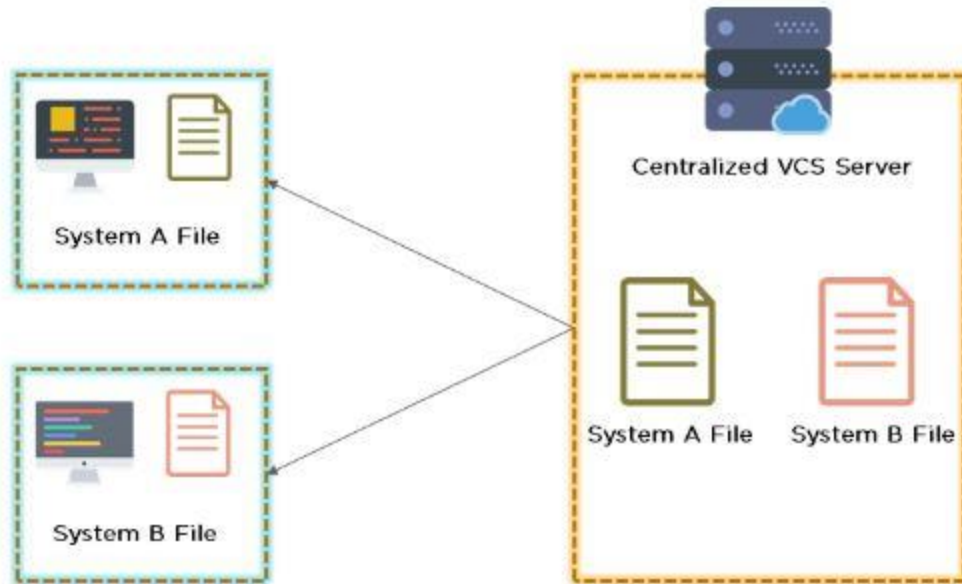
The following diagram includes a few changes:



There have been some changes to file 2 and is updated to file 2.1. This change is stored as Version 2 in the repository. VCS enables you to track the history of a file collection. Each version captures a snapshot of the files at a certain point in time, and the VCS allows you to switch between these versions.

The major drawback of Local VCS is that it has a single point of failure.

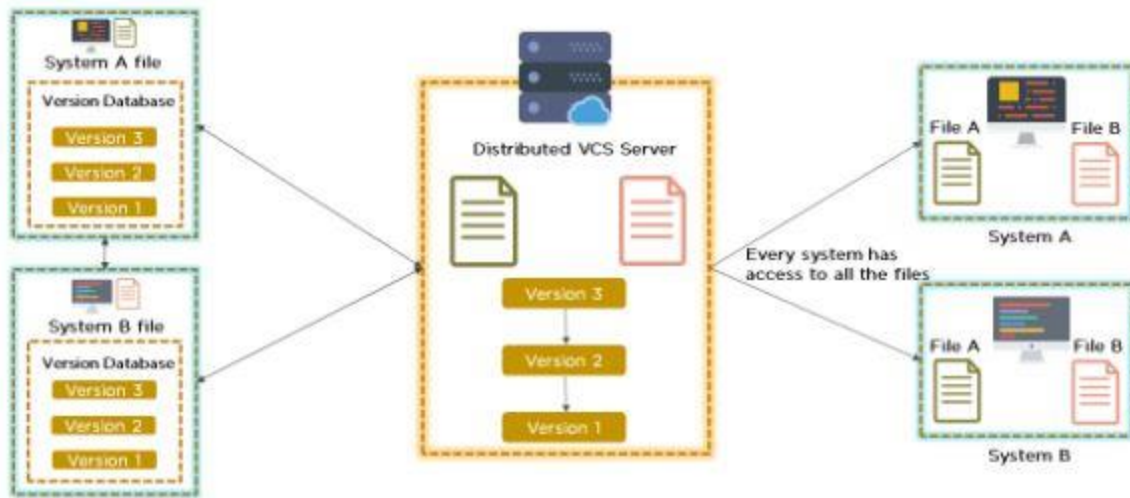
## **b. Centralized Version Control System**



- Uses a central server to store all the files
- Every operation is performed directly on the repository
- All the versions of the file are stored on the Central VCS server
- In case the central server crashes, the entire data of the project will be lost. Hence, distributed VCS was introduced.

### **c. Distributed Version Control System**

- Every programmer has a copy of all the versions of the code on their local systems
- Distributed VCS moves from the client-server approach of central VCS to a peer-to-peer approach
- They can update their local repositories with new data from the central server and changes are reflected in the principal repository
- Git is one such distributed VCS tool



## • Difference between Centralized Version Control System and Distributed Version Control System

Centralized Version Control Systems are systems that use **client/server** architecture. In a centralized Version Control System, one or more client systems are directly connected to a central server. Contrarily the Distributed Version Control Systems are systems that use **peer-to-peer** architecture.

There are many benefits and drawbacks of using both the version control systems. Let's have a look at some significant differences between Centralized and Distributed version control system.

Centralized Version Control System	Distributed Version Control System
In CVCS, The repository is placed at one place and delivers information to many clients.	In DVCS, Every user has a local copy of the repository in place of the central repository on the server-side.
It is based on the client-server approach.	It is based on the client-server approach.
It is the most straightforward system based on the concept of the central repository.	It is flexible and has emerged with the concept that everyone has their repository.
In CVCS, the server provides the latest code to all the clients across the globe.	In DVCS, every user can check out the snapshot of the code, and they can fully mirror the central repository.
CVCS is easy to administrate and has additional control over users and access by its server from one place.	DVCS is fast comparing to CVCS as you don't have to interact with the central server for every command.
The popular tools of CVCS are <b>SVN</b> (Subversion) and <b>CVS</b> .	The popular tools of DVCS are <b>Git</b> and <b>Mercurial</b> .

CVCS is easy to understand for beginners.	DVCS has some complex process for beginners.
If the server fails, No system can access data from another system.	if any server fails and other systems were collaborating via it, that server can restore any of the client repositories

## 2. Basics of Git

### What is Git?

**Git** is an **open-source distributed version control system**. It is designed to handle minor to major projects with high speed and efficiency. It is developed to co-ordinate the work among the developers. The version control allows us to track and work together with our team members at the same workspace.

Git is foundation of many services like **GitHub** and **GitLab**, but we can use Git without using any other Git services. Git can be used **privately** and **publicly**.

Git was created by **Linus Torvalds** in **2005** to develop Linux Kernel. It is also used as an important distributed version-control tool for **the DevOps**.

Git is easy to learn, and has fast performance. It is superior to other SCM(Source code management) tools like Subversion, CVS(Concurrent Versions System), Perforce, and ClearCase.

### • Features of Git

Some remarkable features of Git are as follows:



#### 1. Open Source



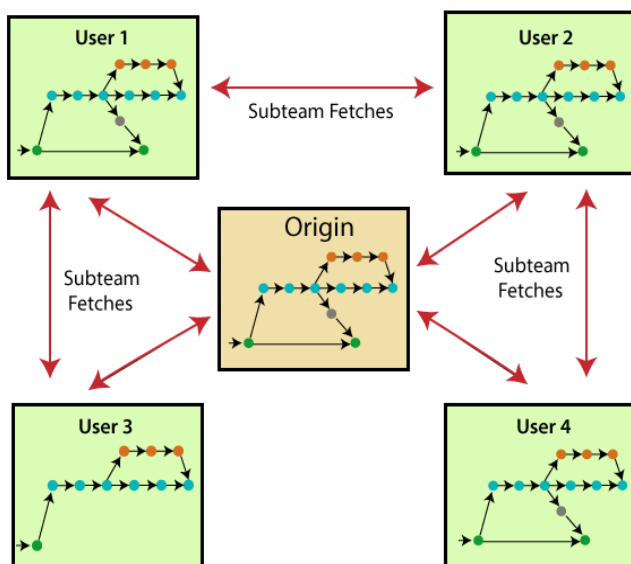
Git is an **open-source tool**. It is released under the **GPL** (General Public License) license.

## 2. Scalability

Git is **scalable**, which means when the number of users increases, the Git can easily handle such situations.

## 3. Distributed

One of Git's great features is that it is **distributed**. Distributed means that instead of switching the project to another machine, we can create a "clone" of the entire repository. Also, instead of just having one central repository that you send changes to, every user has their own repository that contains the entire commit history of the project. We do not need to connect to the remote repository; the change is just stored on our local repository. If necessary, we can push these changes to a remote repository.



## 4. Security

Git is secure. It uses the **SHA1 (Secure Hash Function)** to name and identify objects within its repository. Files and commits are checked and retrieved by its checksum at the time of checkout. It stores its history in such a way that the ID of particular commits depends upon the complete development history leading up to that commit. Once it is published, one cannot make changes to its old version.

## 5. Speed

Git is very **fast**, so it can complete all the tasks in a while. Most of the git operations are done on the local repository, so it provides a **huge speed**. Also, a centralized version control system continually communicates with a server somewhere. Performance tests conducted by Mozilla showed that it was **extremely fast compared to other VCSs**. Fetching version history from a locally stored repository is much faster than fetching it from the remote server. The **core part of Git is written in C**, which **ignores** runtime overheads associated with other high-level languages. Git was developed to work on the Linux kernel; therefore, it is **capable** enough to **handle**



**large repositories** effectively. From the beginning, **speed** and **performance** have been Git's primary goals.

## 6. Supports non-linear development

Git supports **seamless branching and merging**, which helps in visualizing and navigating a non-linear development. A branch in Git represents a single commit. We can construct the full branch structure with the help of its parental commit.

## 7. Branching and Merging

**Branching and merging** are the **great features** of Git, which makes it different from the other SCM tools. Git allows the **creation of multiple branches** without affecting each other. We can perform tasks like **creation**, **deletion**, and **merging** on branches, and these tasks take a few seconds only. Below are some features that can be achieved by branching:

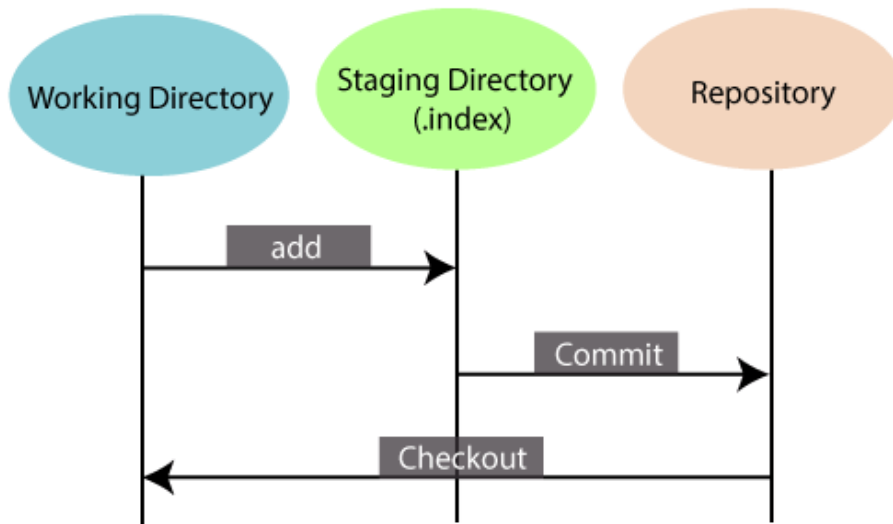
- We can **create a separate branch** for a new module of the project, commit and delete it whenever we want.
- We can have a **production branch**, which always has what goes into production and can be merged for testing in the test branch.
- We can create a **demo branch** for the experiment and check if it is working. We can also remove it if needed.
- The core benefit of branching is if we want to push something to a remote repository, we do not have to push all of our branches. We can select a few of our branches, or all of them together.

## 8. Data Assurance

The Git data model ensures the **cryptographic integrity** of every unit of our project. It provides a **unique commit ID** to every commit through a **SHA algorithm**. We can **retrieve** and **update** the commit by commit ID. Most of the centralized version control systems do not provide such integrity by default.

## 9. Staging Area

The **Staging area** is also a **unique functionality** of Git. It can be considered as a **preview of our next commit**, moreover, an **intermediate area** where commits can be formatted and reviewed before completion. When you make a commit, Git takes changes that are in the staging area and make them as a new commit. We are allowed to add and remove changes from the staging area. The staging area can be considered as a place where Git stores the changes. Although, Git doesn't have a dedicated staging directory where it can store some objects representing file changes (blobs). Instead of this, it uses a file called index.



Another feature of Git that makes it apart from other SCM tools is that **it is possible to quickly stage some of our files and commit them without committing other modified files in our working directory.**

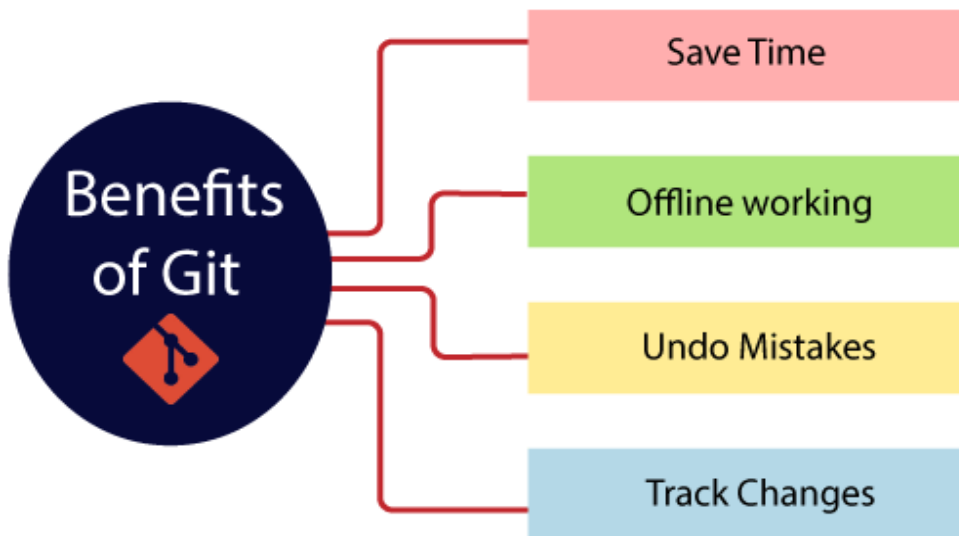
## 10. Maintain the clean history

Git facilitates with Git Rebase; It is one of the most helpful features of Git. It fetches the latest commits from the master branch and puts our code on top of that. Thus, it maintains a clean history of the project.

## • Benefits of Git

A version control application allows us to **keep track** of all the changes that we make in the files of our project. Every time we make changes in files of an existing project, we can push those changes to a repository. Other developers are allowed to pull your changes from the repository and continue to work with the updates that you added to the project files.

Some **significant benefits** of using Git are as follows:



### 1. Saves Time

Git is lightning fast technology. Each command takes only a few seconds to execute so we can save a lot of time as compared to login to a GitHub account and find out its features.

### 2. Offline Working

One of the most important benefits of Git is that it supports **offline working**. If we are facing internet connectivity issues, it will not affect our work. In Git, we can do almost everything locally. Comparatively, other CVS like SVN is limited and prefer the connection with the central repository.

### 3. Undo Mistakes

One additional benefit of Git is we can **Undo** mistakes. Sometimes the undo can be a savior option for us. Git provides the undo option for almost everything.

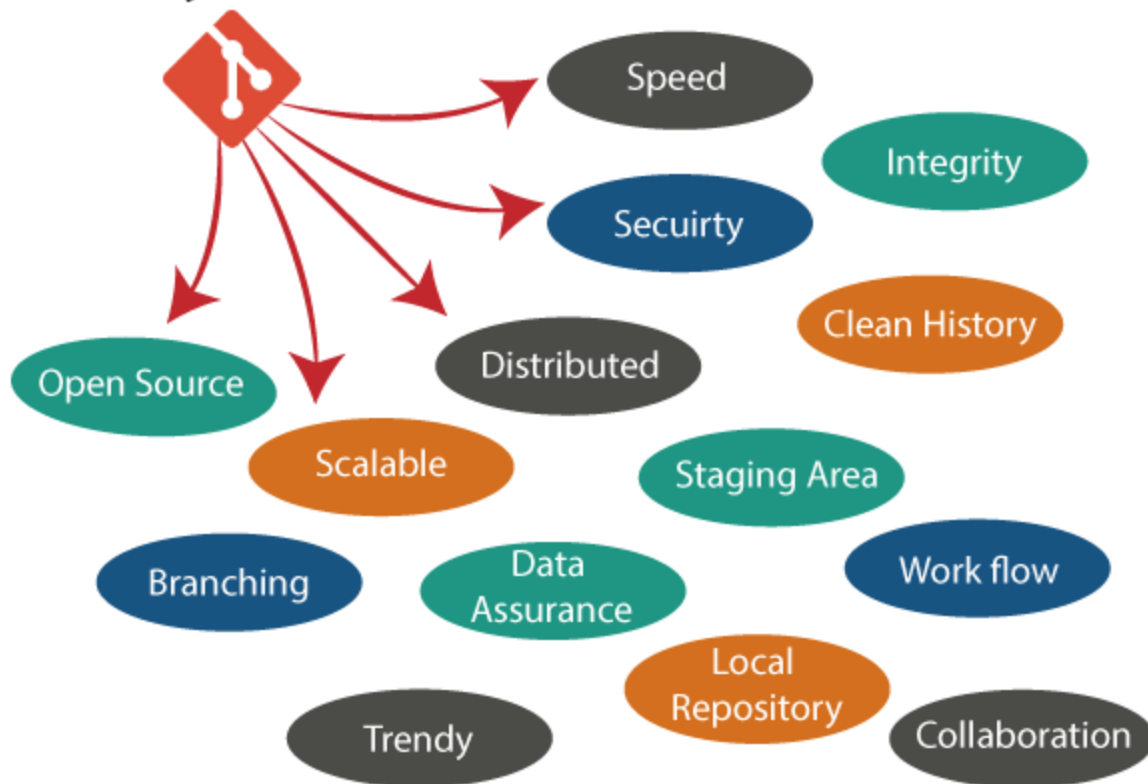
### 4. Track the Changes

Git facilitates with some exciting features such as **Diff**, **Log**, and **Status**, which allows us to track changes so we can **check the status**, **compare** our files or branches.

## • Why Git?

We have discussed many **features** and **benefits** of Git that demonstrate the undoubtedly Git as the **leading version control system**. Now, we will discuss some other points about why should we choose Git.

# Why Git?



## 1. Git Integrity

Git is **developed to ensure** the **security** and **integrity** of content being version controlled. It uses checksum during transit or tampering with the file system to confirm that information is not lost. Internally it creates a checksum value from the contents of the file and then verifies it when transmitting or storing data.

## 2. Trendy Version Control System

Git is the **most widely used version control system**. It has **maximum projects** among all the version control systems. Due to its **amazing workflow** and features, it is a preferred choice of developers.

## 3. Everything is Local

Almost All operations of Git can be performed locally; this is a significant reason for the use of Git. We will not have to ensure internet connectivity.

## 4. Collaborate to Public Projects

There are many public projects available on the GitHub. We can collaborate on those projects and show our creativity to the world. Many developers are collaborating on public projects. The collaboration allows us to stand with experienced developers and learn a lot from them; thus, it takes our programming skills to the next level.

## 5. Impress Recruiters

We can impress recruiters by mentioning the Git and GitHub on our resume. Send your GitHub profile link to the HR of the organization you want to join. Show your skills and influence them through your work. It increases the chances of getting hired.

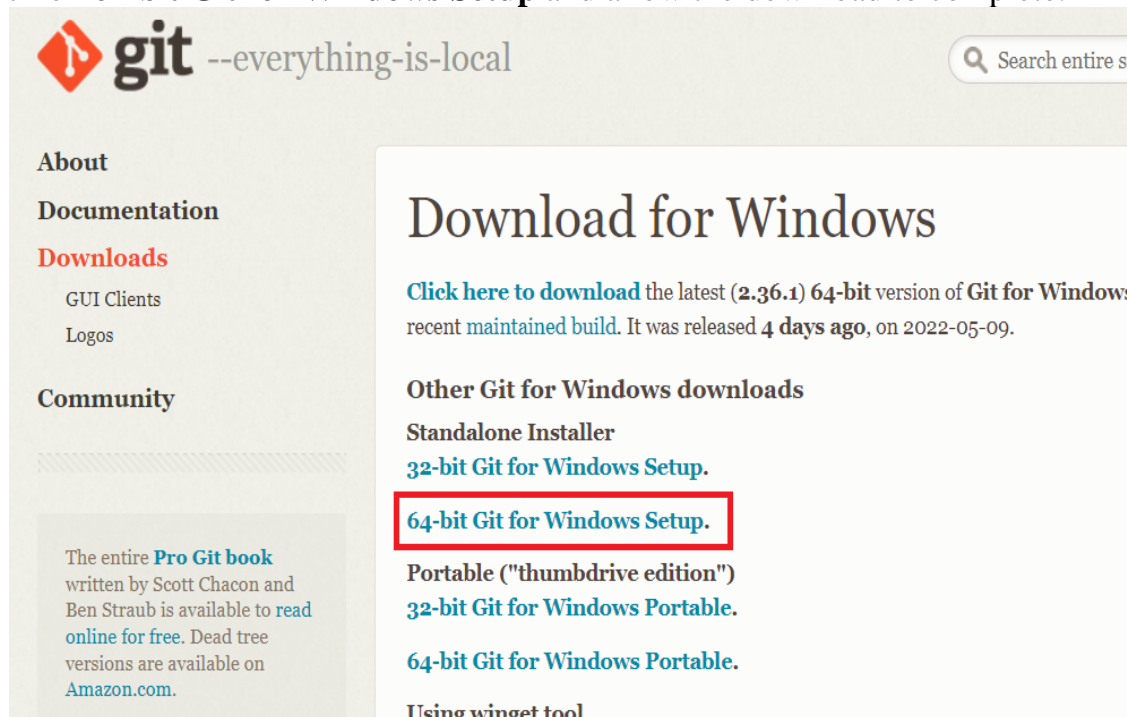
## 3. Installing and Configuring Git

### Steps to download and install Git on Windows

#### Downloading

**Step 1:** Go to the official website: <https://git-scm.com>

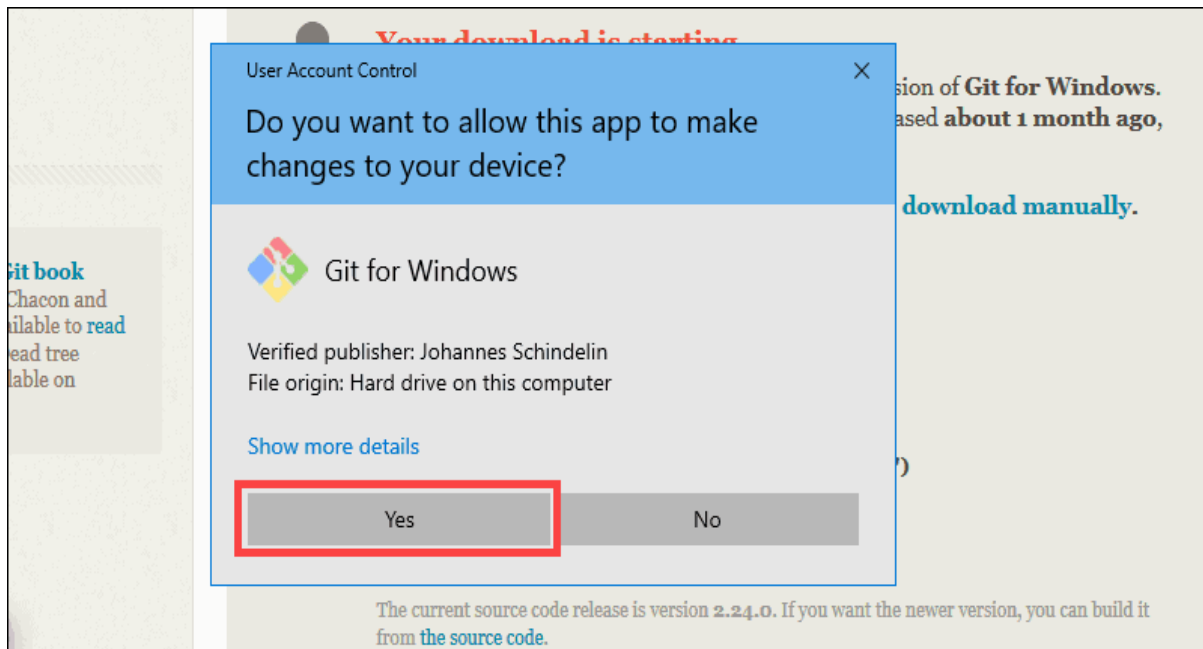
**Step 2:** Click on **64-bit Git for Windows Setup** and allow the download to complete.



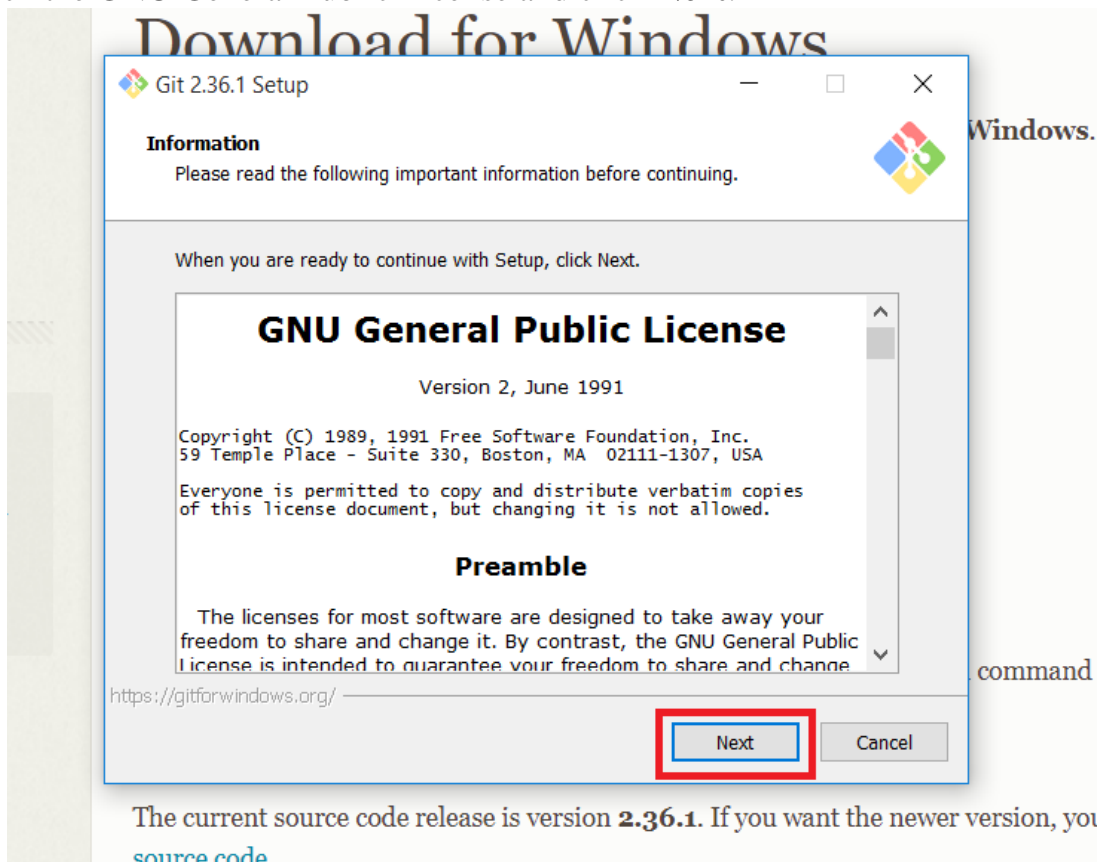
#### Extract and Launch Git Installer

**Step 3:** Go to your download location and double-click the file to launch the installer.

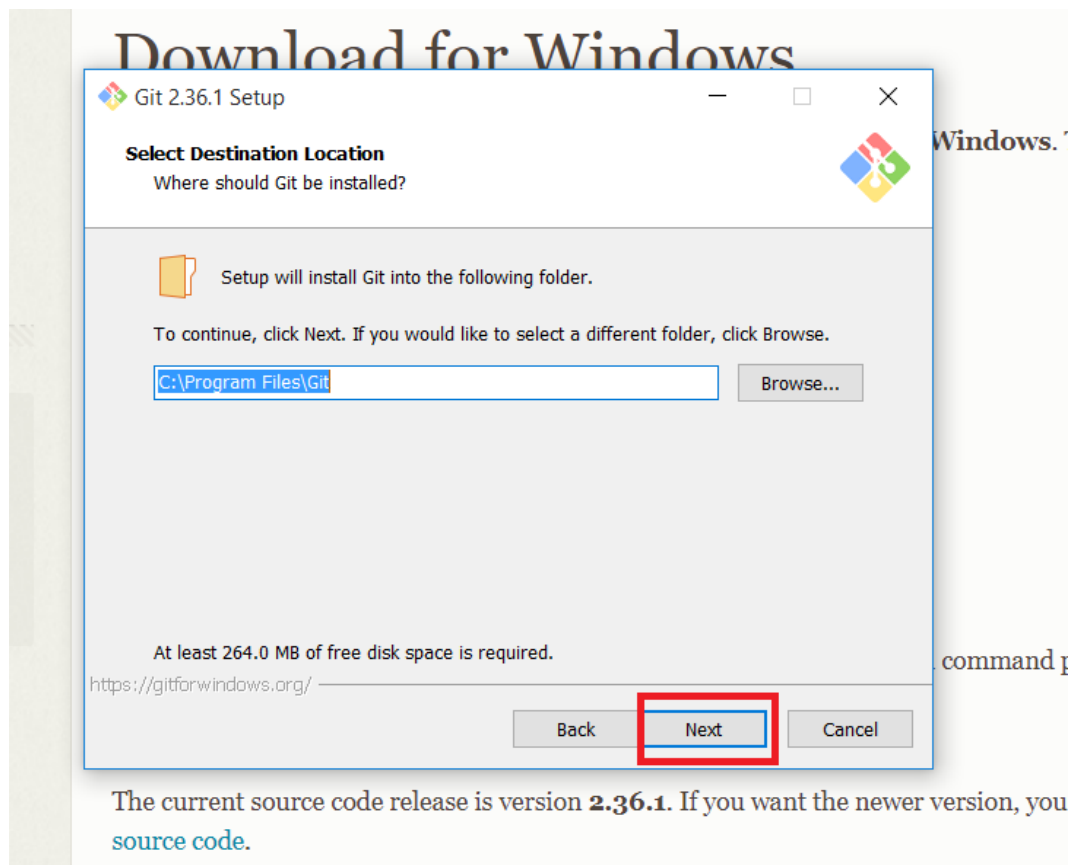
**Step 4:** Allow the app to modify your device by selecting Yes in the User Account Control window that appears.



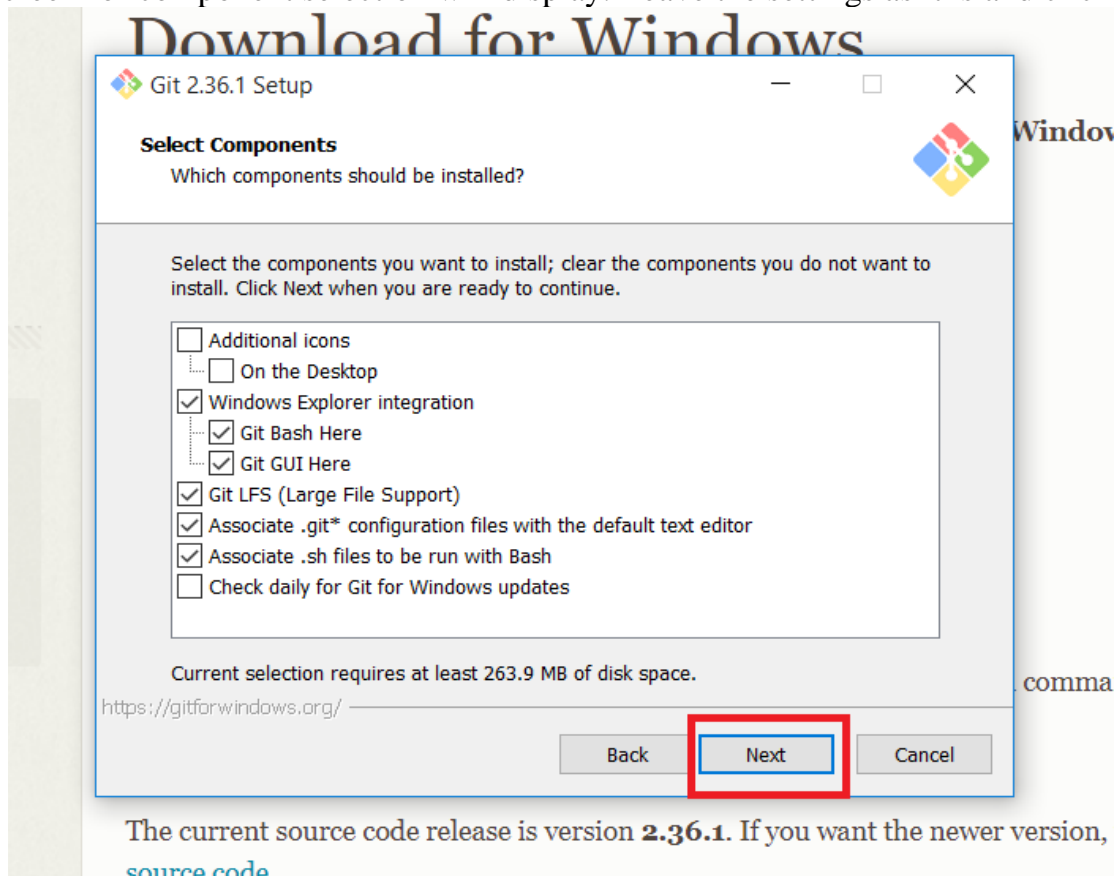
**Step 5:** Check the GNU General Public License and click **Next**.



**Step 6:** Select the install location. If you don't have a reason to modify it, leave it to default and click **Next**.

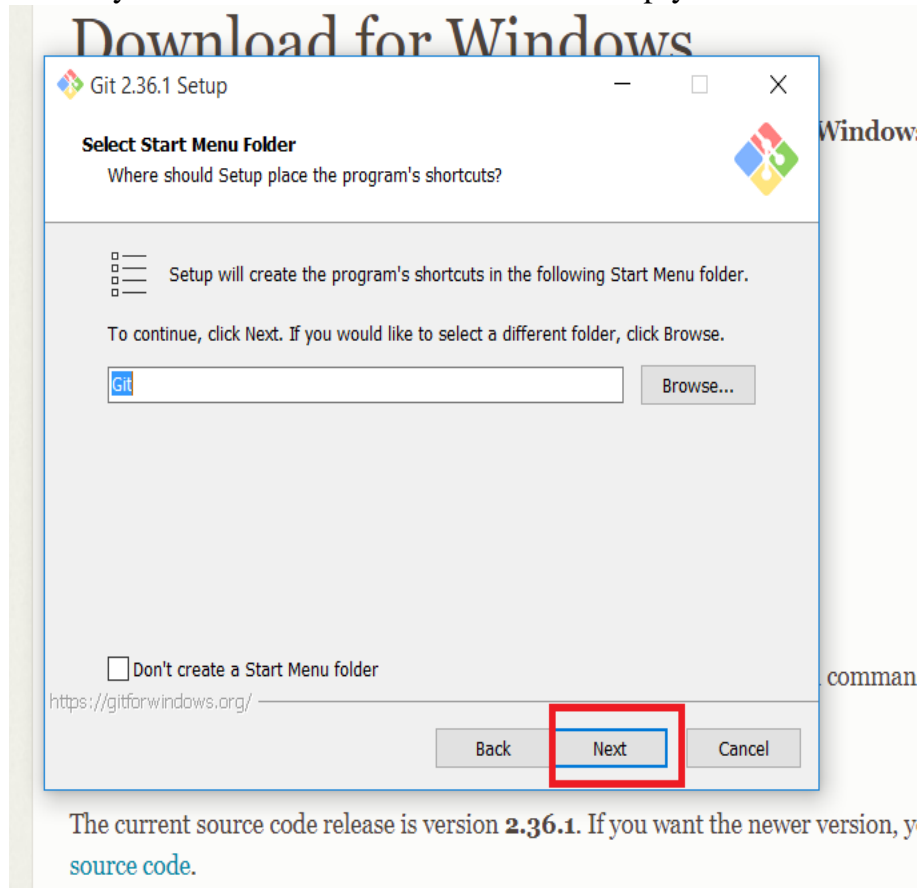


**Step 7:** A screen for component selection will display. Leave the settings as it is and click **Next**.

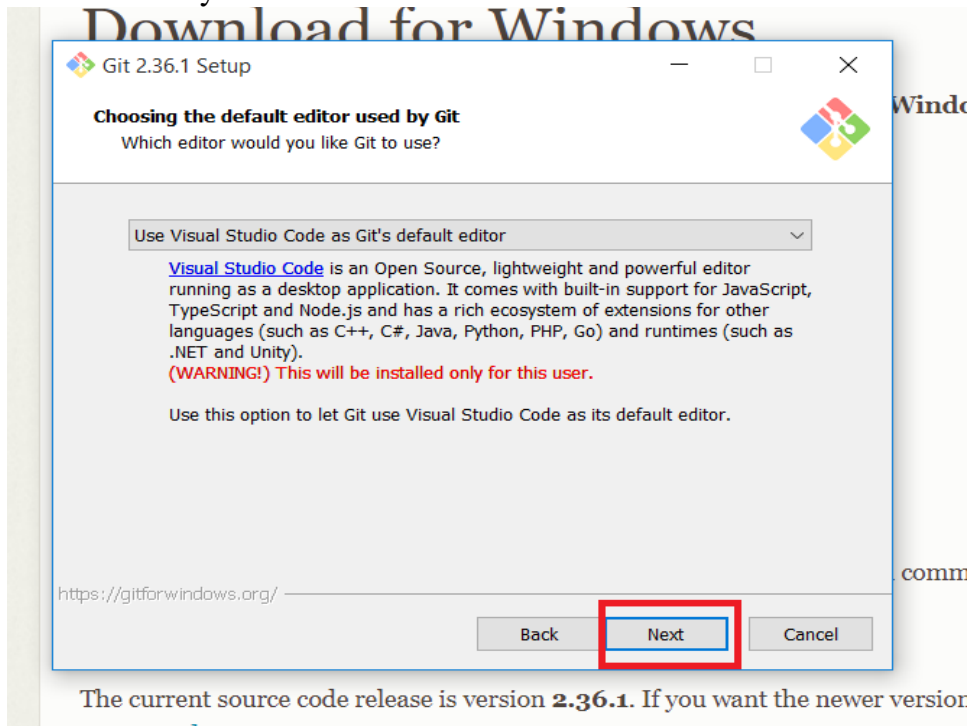




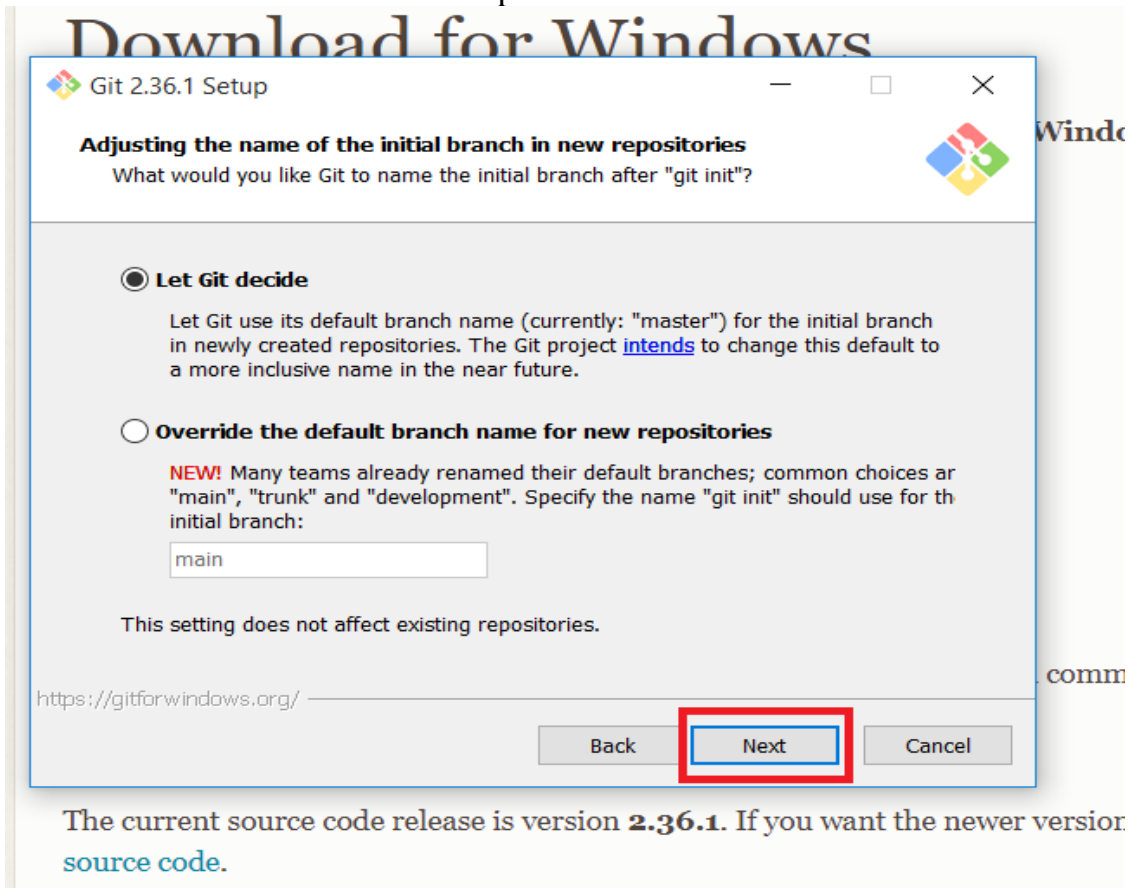
**Step 8:** The installer asks you to create a start menu folder. Simply click **Next**.



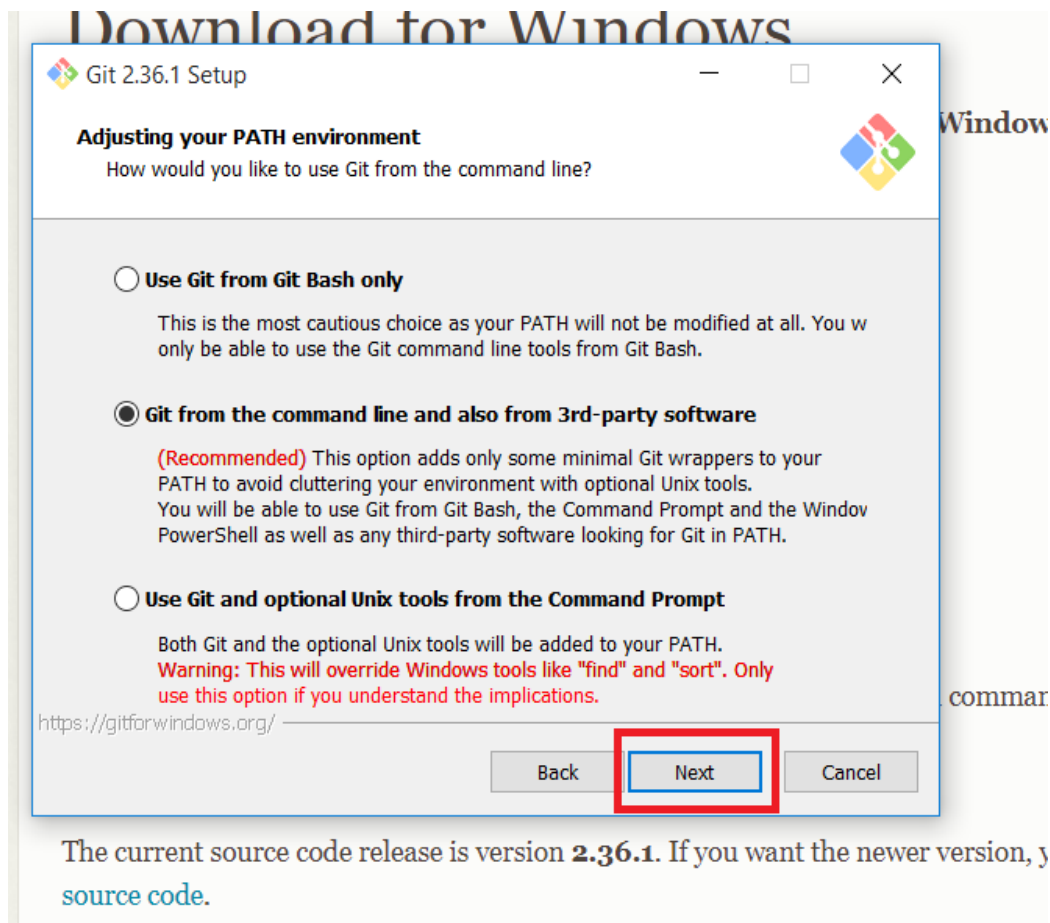
**Step 9:** Choose the text editor you want to use with Git and click **Next**.



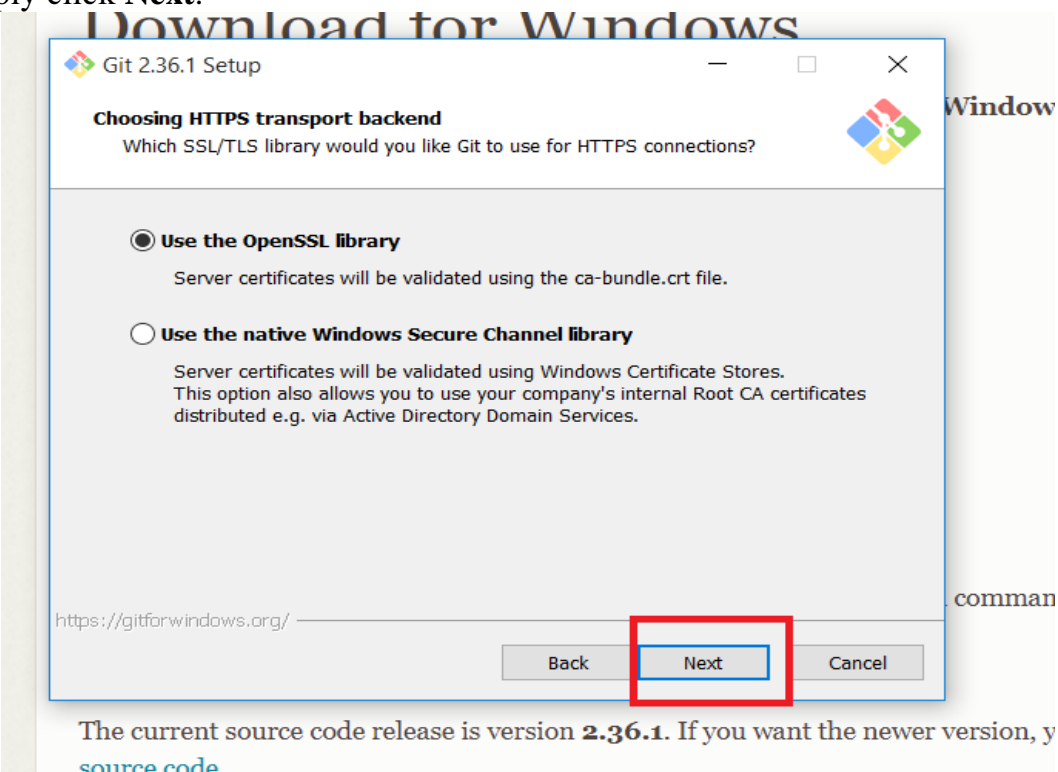
**Step 10:** The following step allows you to give your original branch a new name. 'Master' is the default. Leave the default choice selected and press the **Next** button.



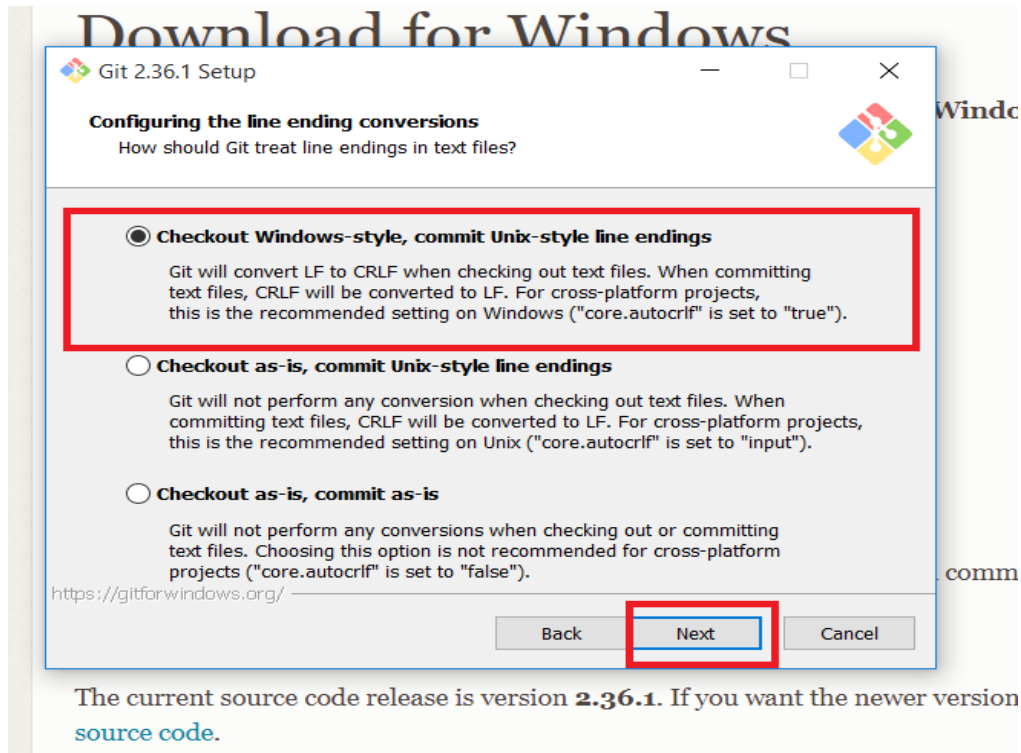
**Step 11:** You can adjust the PATH environment during this installation phase. When you run a command from the command line, the PATH is the default set of folders that are included. Continue by selecting the middle (recommended) option and clicking **Next**.



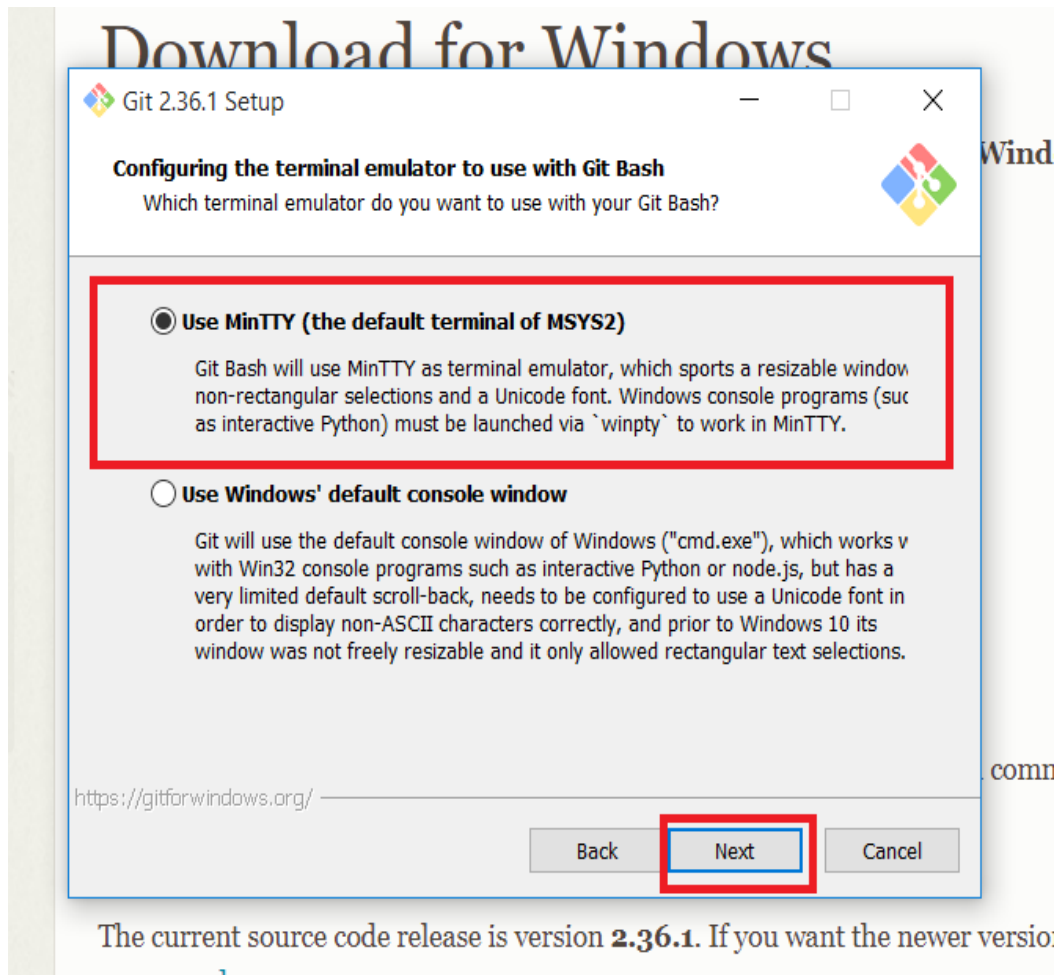
**Step 12:** The following option concerns server certificates. The default choice is used by the majority of users. Simply click **Next**.



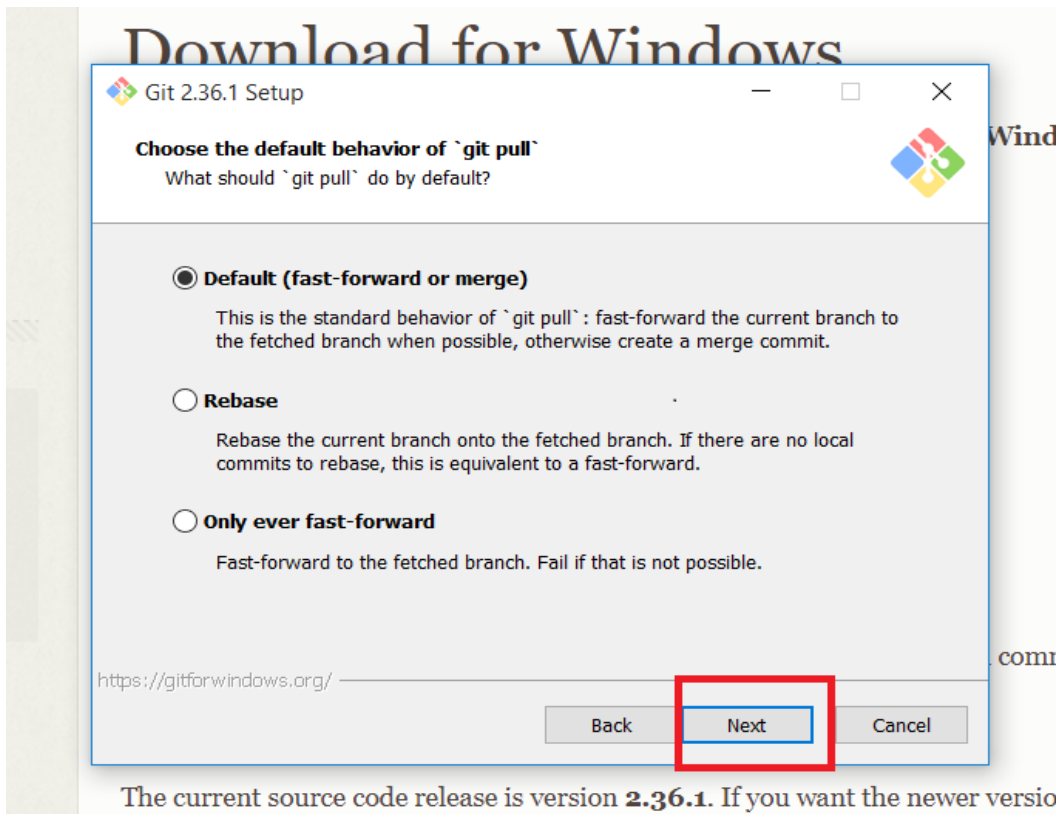
**Step 13:** This step deals with how data is structured, and altering this option may create issues. So, it is advised to leave the default selection.



**Step 14:** Select the terminal emulator that you wish to use. Because of its features, the default MinTTY is suggested. Click **Next**.

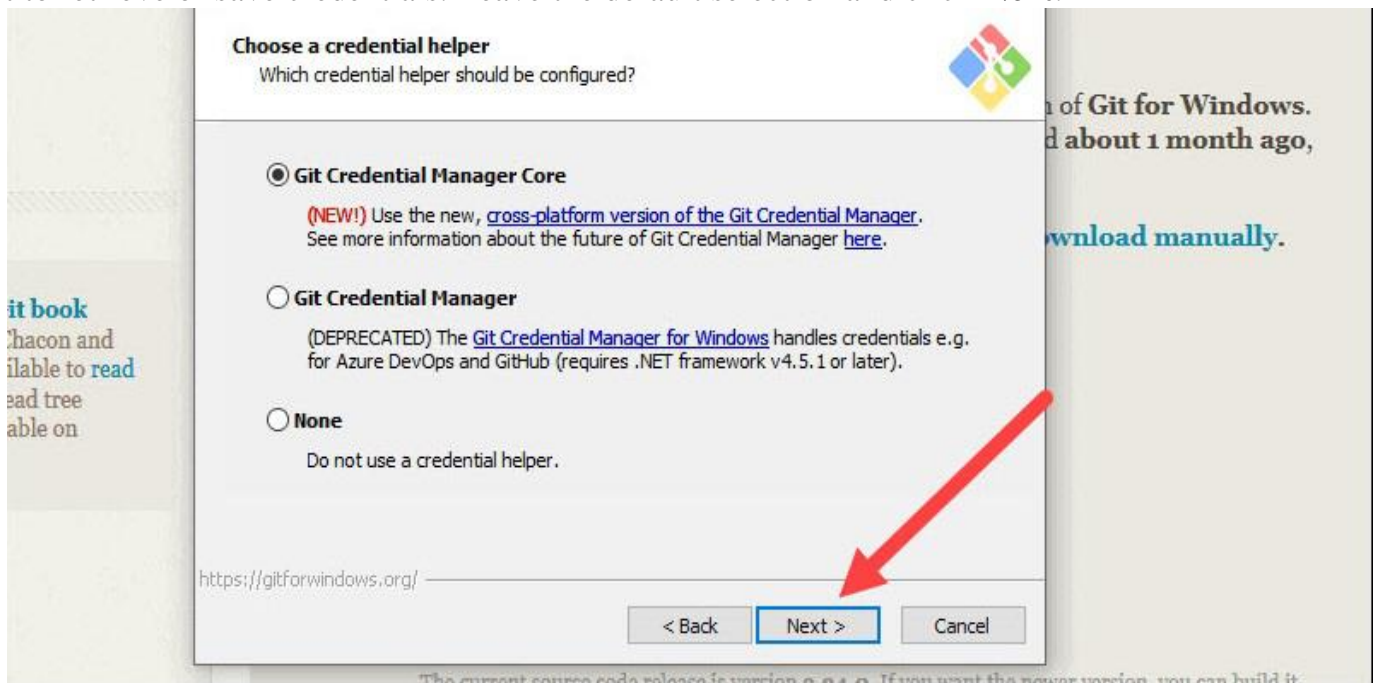


**Step 15:** The installer now prompts you to specify what the git pull command should perform. Leave the default selected option and click **Next**.

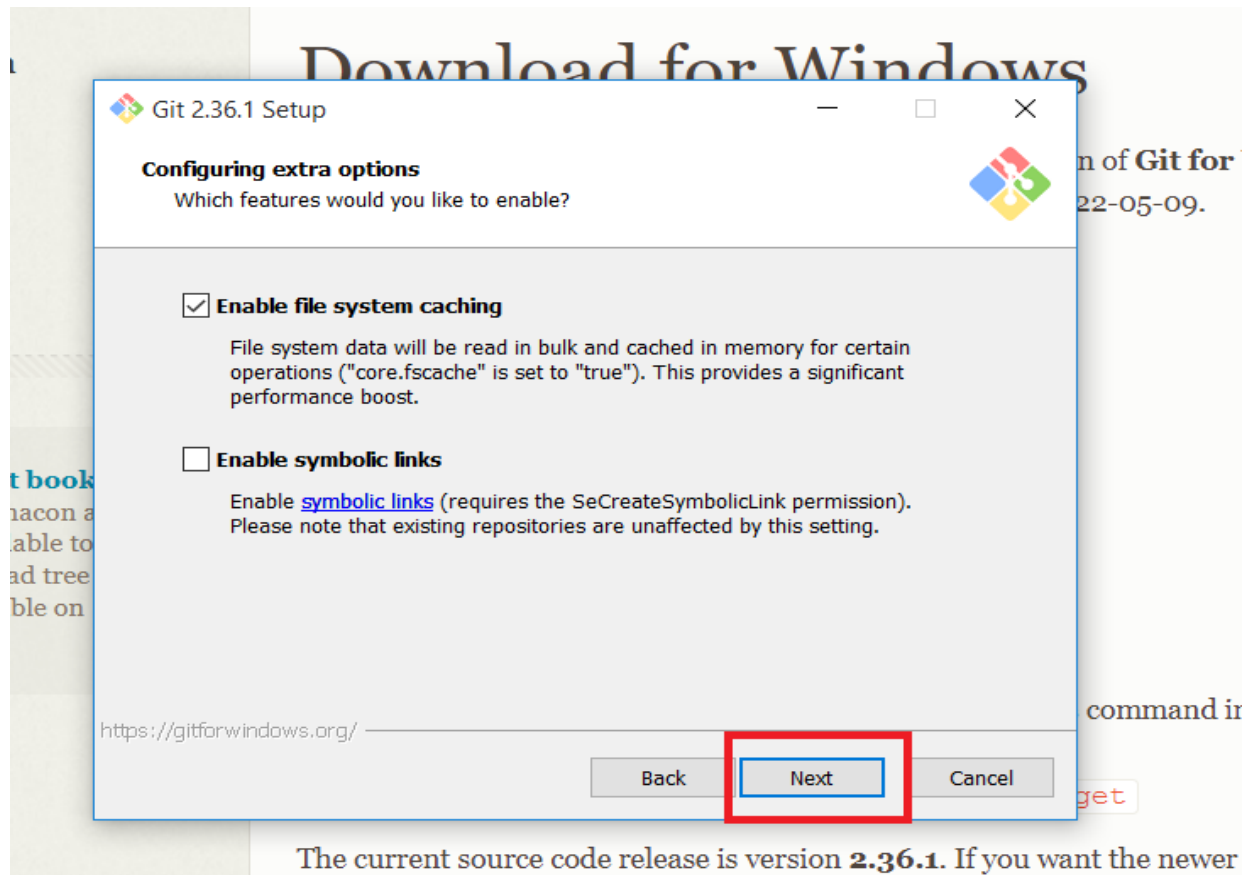


The current source code release is version **2.36.1**. If you want the newer version

**Step 16:** The next step is to decide which credential helper to employ. Credential helpers are used by Git to retrieve or save credentials. Leave the default selection and click **Next**.

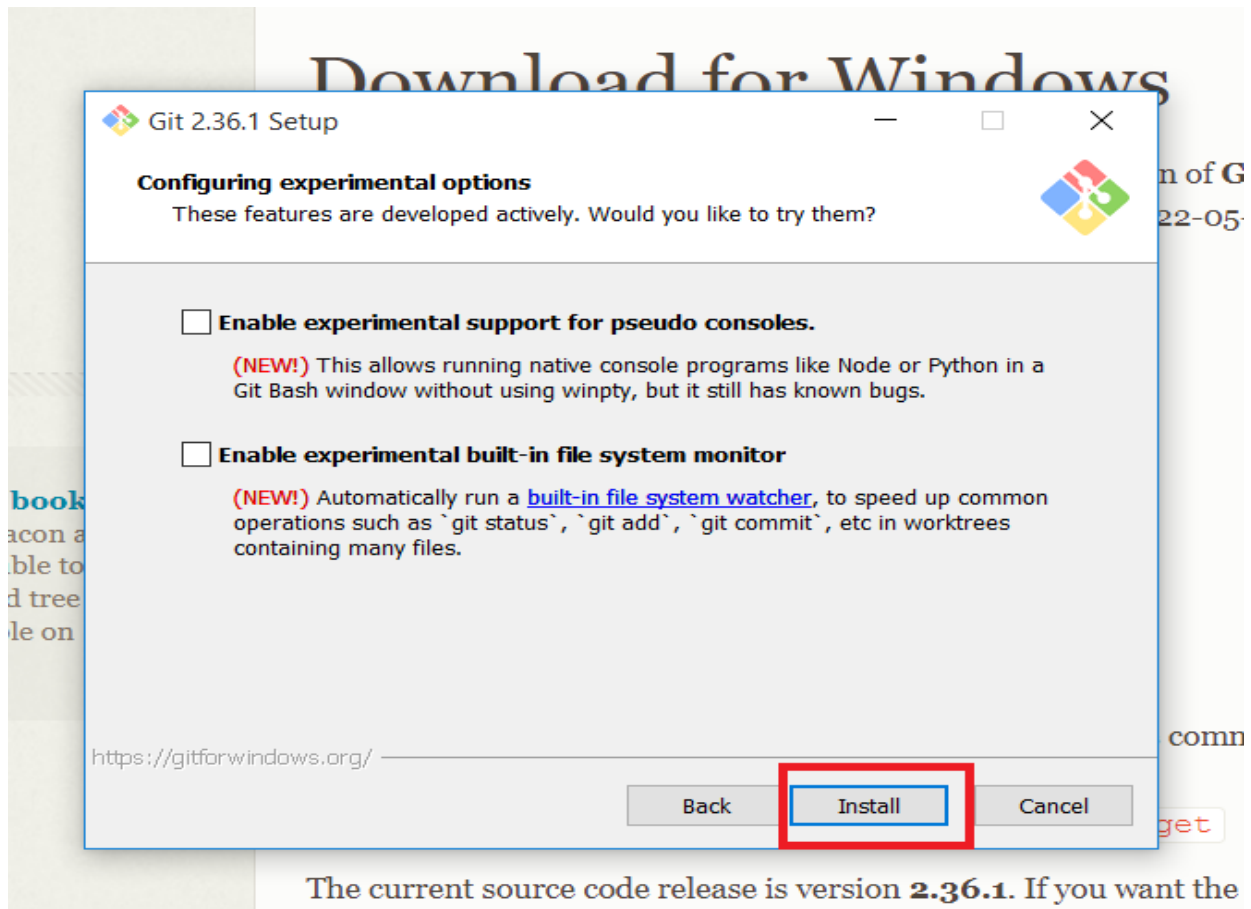


**Step 17:** Although the default choices are suggested, this step allows you to select which additional features to activate.

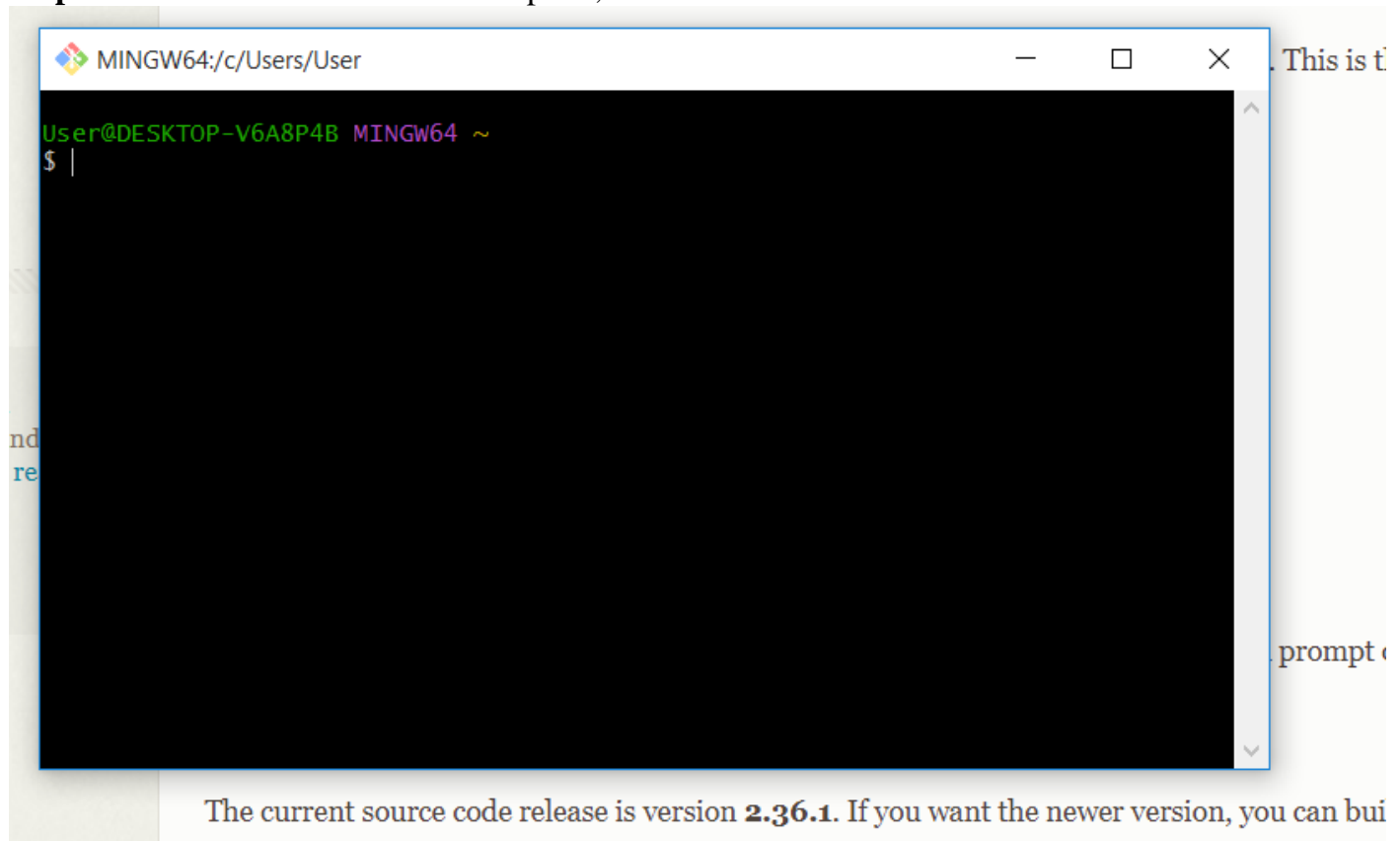


**Step 18:** Git offers to install some experimental features. Leave them unchecked and click **Install**.





**Step 19:** Once the installation is complete, launch the Git bash.



**Command** to create git repository in folder and store files and track changes.

- Check the version of Git.

```
SSPL-LP-DNS-YT0+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~  
$ git --version  
git version 2.18.0.windows.1
```

- Create a “week1” repository in the local system.
- Move to the week1 repository.

```
MINGW64/f:/Anusha/week1  
Admin@DESKTOP-T3DAU1H MINGW64 ~ (master)  
$ pwd  
/c/Users/Admin  
Admin@DESKTOP-T3DAU1H MINGW64 ~ (master)  
$ cd "F:\Anusha\week1"
```

- Create a new git instance for a project.

## 1.git init

- The command git init is used to create an empty Git repository.
- After the git init command is used, a .git folder is created in the directory with some subdirectories. Once the repository is initialized, the process of creating other files begins.

```
Admin@DESKTOP-T3DAU1H MINGW64 /f/Anusha/week1  
$ git init  
Initialized empty Git repository in F:/Anusha/week1/.git/
```

- Set up global config variables with github account username and email- If you are working with other developers, you need to know who is checking the code in and out, and to make the changes.

## 2.git config

- The gitconfig command is used initially to configure the user.name and user.email. This specifies what email id and username will be used from a local repository.
- When gitconfig is used with --global flag, it writes the settings to all repositories on the computer.

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

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

```
Admin@DESKTOP-T3DAU1H MINGW64 /f/Anusha/week1 (master)
$ git config --global user.name "devisar"

Admin@DESKTOP-T3DAU1H MINGW64 /f/Anusha/week1 (master)
$ git config --global user.email "anupenugonda1998@cmritonline.ac.in"
```

```
MINGW64/f/Anusha/week1
Admin@DESKTOP-T3DAU1H MINGW64 /f/Anusha/week1 (master)
$ git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw64/etc/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
pull.rebase=false
credential.helper=manager
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=master
user.name=devisar
user.email=anupenugonda1998@cmritonline.ac.in
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.logallrefupdates=true
core.symlinks=false
core.ignorecase=true
```

### 3. git help

- If in case you need help, use the following commands:

```
SSPL-LP-DNS-YT0+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~
$ git help config

SSPL-LP-DNS-YT0+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~
$ git config --help
```

This will lead you to the Git help page on the browser, which will display the following:

## git-add(1) Manual Page

### NAME

git-add - Add file contents to the index

### SYNOPSIS

```
git add [--verbose | -v] [--dry-run | -n] [--force | -f] [--interactive | -i] [--patch | -p]
        [--edit | -e] [--[no-]all | --[no-]ignore-removal | [--update | -u]]
        [--intent-to-add | -N] [--refresh] [--ignore-errors] [--ignore-missing] [--renormalize]
        [--chmod={+|-}x] [--] [<pathspec>...]
```

```
SSPL-LP-DNS-YT0+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~
$ git help add
```

This will lead you to the Git help page on the browser, which will display the following:

## git-add(1) Manual Page

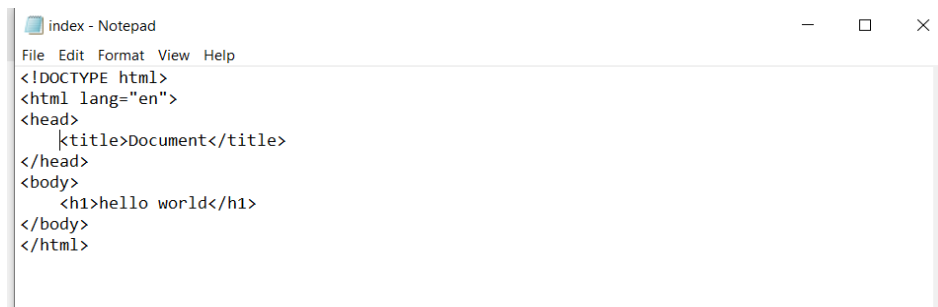
### NAME

git-add - Add file contents to the index

### SYNOPSIS

```
git add [--verbose | -v] [--dry-run | -n] [--force | -f] [--interactive | -i] [--patch | -p]
        [--edit | -e] [--[no-]all | --[no-]ignore-removal | [--update | -u]]
        [--intent-to-add | -N] [--refresh] [--ignore-errors] [--ignore-missing] [--renormalize]
        [--chmod={+|-}x] [--] [<pathspec>...]
```

- Create a file called index.html in the week1 folder; write something and save it.



```
index - Notepad
File Edit Format View Help
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Document</title>
</head>
<body>
  <h1>hello world</h1>
</body>
</html>
```

## 4. Common Git Commands

### GIT Repository

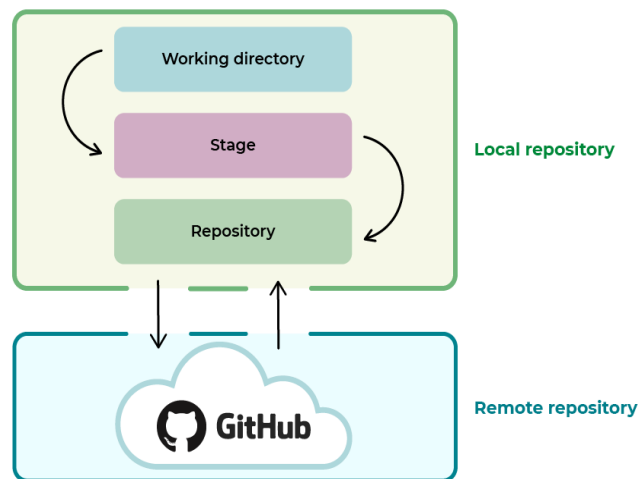
Repositories in GIT contain a collection of files of various different versions of a Project. These files are imported from the repository into the local server of the user for further updations and modifications in the content of the file. A VCS or the Version Control System is used to create these versions and store them in a specific place termed as a repository.

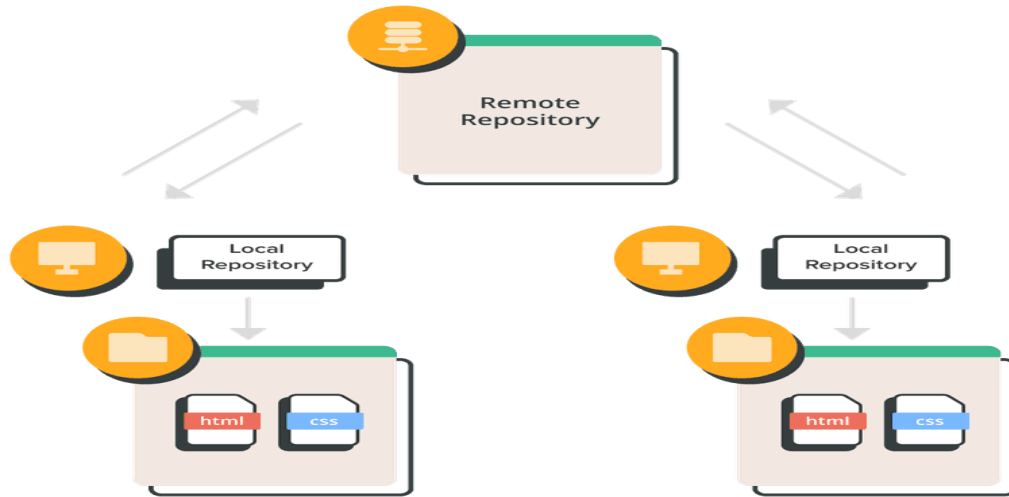
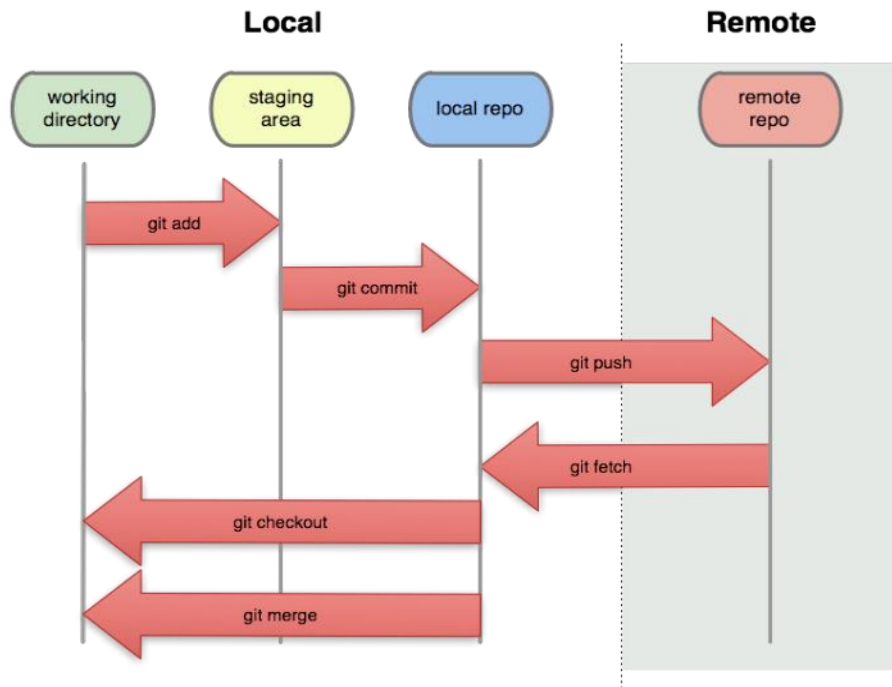
**Repositories in Git are of two types:**

**Local Repository:** Git allows the users to perform work on a project from all over the world because of its Distributive feature. This can be done by cloning the content from the Central repository stored in the GitHub on the user's local machine. This local copy is used to perform operations and test them on the local machine before adding them to the central repository.

**Remote Repository:** Git allows the users to sync their copy of the local repository to other repositories present over the internet. This can be done to avoid performing a similar operation by multiple developers. Each repository in Git can be addressed by a shortcut called remote.

Git provides tools to perform work on these repositories according to the needs of the user. This workflow of performing modifications to a Repository is referred to as the Working Tree.





**Fig. Working Flow of Local Repository and Remote Repository**

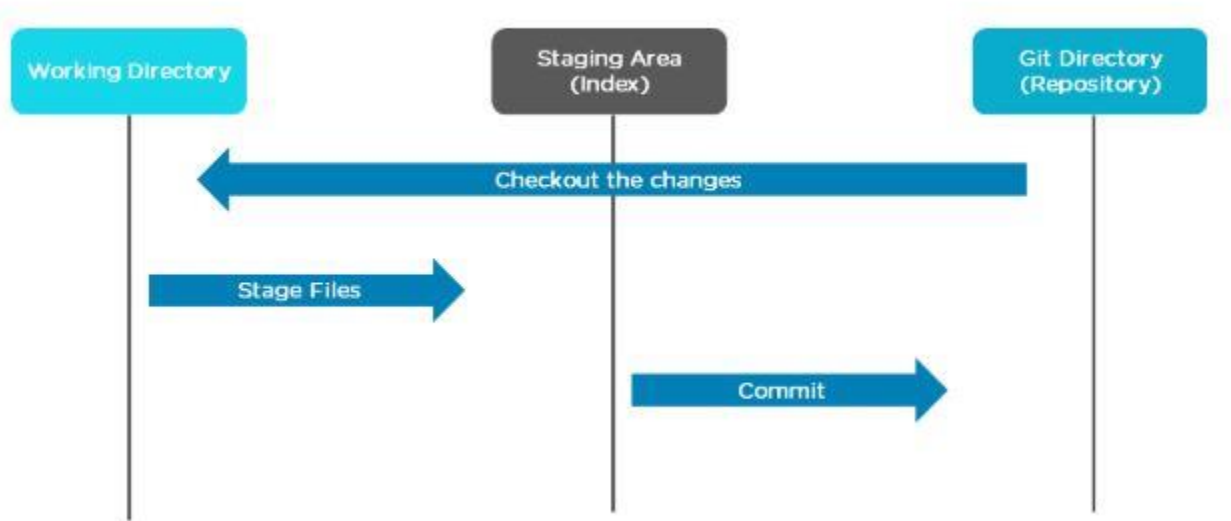
## Working with A Local Repository

Git allows the users to perform work on a project from all over the world because of its Distributive feature. This can be done by cloning the content from the Central repository stored in the GitHub on the user's local machine. This local copy is used to perform operations and test them on the local machine before adding them to the central repository.

The Git workflow is divided into three states:

- Working directory - Modify files in your working directory

- Staging area (Index) - Stage the files and add snapshots of them to your staging area
- Git directory ( Local Repository) - Perform a commit that stores the snapshots permanently to your Git directory. Checkout any existing version, make changes, stage them and commit.



**Command** to create git repository in folder and store files and track changes.

- Check the version of Git.

```

SSPL-LP-DNS-YT0+SimpleLearn@SSPL-LP-DNS-YT01 MINGW64 ~
$ git --version
git version 2.18.0.windows.1
  
```

- Create a “week1” repository in the local system.
- Move to the week1 repository.

```

MINGW64/f/Anusha/week1
Admin@DESKTOP-T3DAU1H MINGW64 ~ (master)
$ pwd
/c/Users/Admin
Admin@DESKTOP-T3DAU1H MINGW64 ~ (master)
$ cd "F:\Anusha\week1"
  
```

- Create a new git instance for a project.

## 1.gitinit

- The command git init is used to create an empty Git repository.



- After the git init command is used, a .git folder is created in the directory with some subdirectories. Once the repository is initialized, the process of creating other files begins.

```
Admin@DESKTOP-T3DAU1H MINGW64 /f/Anusha/week1
$ git init
Initialized empty Git repository in F:/Anusha/week1/.git/
```

- Set up global config variables with github account username and email- If you are working with other developers, you need to know who is checking the code in and out, and to make the changes.

## 2.gitconfig

- The gitconfig command is used initially to configure the user.name and user.email. This specifies what email id and username will be used from a local repository.
- When gitconfig is used with --global flag, it writes the settings to all repositories on the computer.

**gitconfig --global user.name "user name"**

**gitconfig --global user.email "email id"**

```
Admin@DESKTOP-T3DAU1H MINGW64 /f/Anusha/week1 (master)
$ git config --global user.name "devisar"

Admin@DESKTOP-T3DAU1H MINGW64 /f/Anusha/week1 (master)
$ git config --global user.email "anupenugonda1998@cmritonline.ac.in"
```

```
MINGW64/f/Anusha/week1
Admin@DESKTOP-T3DAU1H MINGW64 /f/Anusha/week1 (master)
$ git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw64/etc/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
pull.rebase=false
credential.helper=manager
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=master
user.name=devisar
user.email=anupenugonda1998@cmritonline.ac.in
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.logallrefupdates=true
core.symlinks=false
core.ignorecase=true
```

## 3. git help

- If in case you need help, use the following commands:

```
SSPL-LP-DNS-YT0+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~
$ git help config

SSPL-LP-DNS-YT0+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~
$ git config --help
```

This will lead you to the Git help page on the browser, which will display the following:

```
git-add(1) Manual Page

NAME

git-add - Add file contents to the index

SYNOPSIS

git add [--verbose | -v] [--dry-run | -n] [--force | -f] [--interactive | -i] [--patch | -p]
[--edit | -e] [--[no-]all | --[no-]ignore-removal | [--update | -u]]
[--intent-to-add | -N] [--refresh] [--ignore-errors] [--ignore-missing] [--renormalize]
[--chmod=(+|-)x] [--] [<pathspec>...]
```

```
SSPL-LP-DNS-YT0+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~
$ git help add
```

This will lead you to the Git help page on the browser, which will display the following:

```
git-add(1) Manual Page

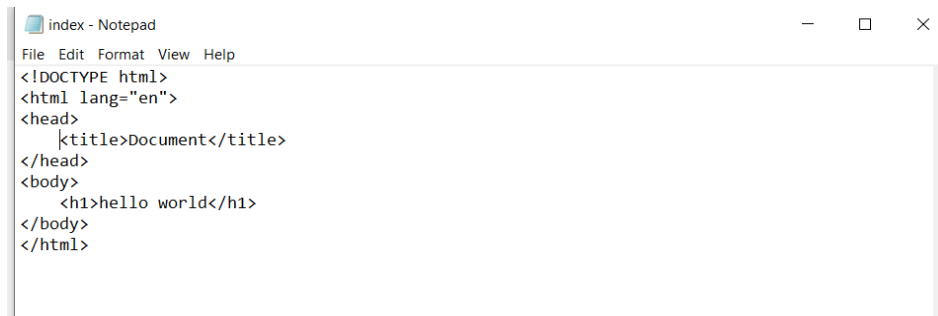
NAME

git-add - Add file contents to the index

SYNOPSIS

git add [--verbose | -v] [--dry-run | -n] [--force | -f] [--interactive | -i] [--patch | -p]
[--edit | -e] [--[no-]all | --[no-]ignore-removal | [--update | -u]]
[--intent-to-add | -N] [--refresh] [--ignore-errors] [--ignore-missing] [--renormalize]
[--chmod=(+|-)x] [--] [<pathspec>...]
```

- Create a file called index.html in the week1 folder; write something and save it.

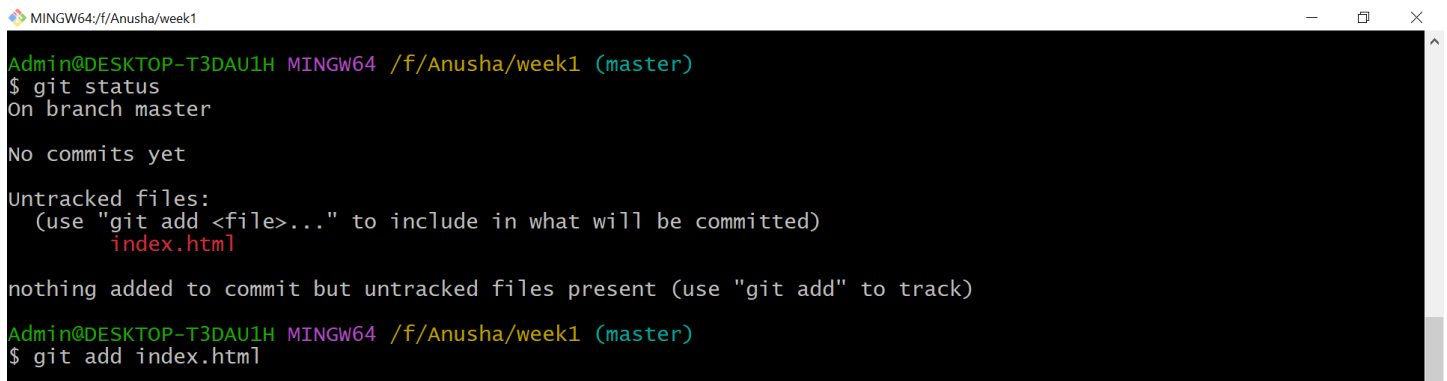


```
index - Notepad
File Edit Format View Help
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Document</title>
</head>
<body>
  <h1>hello world</h1>
</body>
</html>
```

### 3.git add

- Add command is used after checking the status of the files, to add those files to the staging area.
- Before running the commit command, "git add" is used to add any new or modified files.

#### git add . (or) git add file name



```
Admin@DESKTOP-T3DAU1H MINGW64 /f/Anusha/week1 (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        index.html

nothing added to commit but untracked files present (use "git add" to track)
Admin@DESKTOP-T3DAU1H MINGW64 /f/Anusha/week1 (master)
$ git add index.html
```

### 4.git commit

- The commit command makes sure that the changes are saved to the local repository.
- The command "git commit -m <message>" allows you to describe everyone and help them understand what has happened.

#### git commit -m "commit message"

```

Admin@DESKTOP-T3DAU1H MINGW64 /f/Anusha/week1 (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   index.html

Admin@DESKTOP-T3DAU1H MINGW64 /f/Anusha/week1 (master)
$ git commit -m "index file committing"
[master (root-commit) f7aecd2] index file committing
 1 file changed, 19 insertions(+)
 create mode 100644 index.html

```

## 5.git status

- The git status command tells the current state of the repository.
- The command provides the current working branch. If the files are in the staging area, but not committed, it will be shown by the git status. Also, if there are no changes, it will show the message no changes to commit, working directory clean.

```

Admin@DESKTOP-T3DAU1H MINGW64 /f/Anusha/week1 (master)
$ git status
On branch master
nothing to commit, working tree clean

```

## 6.git log

- The git log command shows the order of the commit history for a repository.
- The command helps in understanding the state of the current branch by showing the commits that lead to this state.

```

Admin@DESKTOP-T3DAU1H MINGW64 /f/Anusha/week1 (master)
$ git log
commit f7aecd23caba4bec5f2adda3ca163fe5e3f78581 (HEAD -> master)
Author: devisar <anupenugonda1998@cmritonline.ac.in>
Date:   Tue Mar 12 19:17:44 2024 +0530

    index file committing

```

## 7.git diff

- Make any necessary changes to the file and save.

```
index - Notepad
File Edit Format View Help
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Document</title>
</head>
<body>
  <h1>hello world</h1>
  <h2>Hello students</h2>
</body>
</html>
```

- Now that you've made changes to the file, you can compare the differences since your last commit.
- Use command **git diff**

```
F:\Anusha\FSD>git diff
diff --git a/index.html b/index.html
index f18cfea..a9ce83b 100644
--- a/index.html
+++ b/index.html
@@ -5,5 +5,6 @@
  </head>
  <body>
    <h1>hello world</h1>
+   <h2>Hello students</h2>
  </body>
</html>
\ No newline at end of file

F:\Anusha\FSD>
```

- Save modified data to git repository using command **git commit -a -m "modified"**

```
C:\Windows\System32\cmd.exe
F:\Anusha\FSD>git status
On branch anu
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   index.html

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    week5 program/
    week6/week6.0/
    week7.zip

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

F:\Anusha\FSD>git commit -a -m "modified"
[anu 95664b7] modified
1 file changed, 1 insertion(+)

F:\Anusha\FSD>git status
On branch anu
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    week5 program/
    week6/week6.0/
    week7.zip

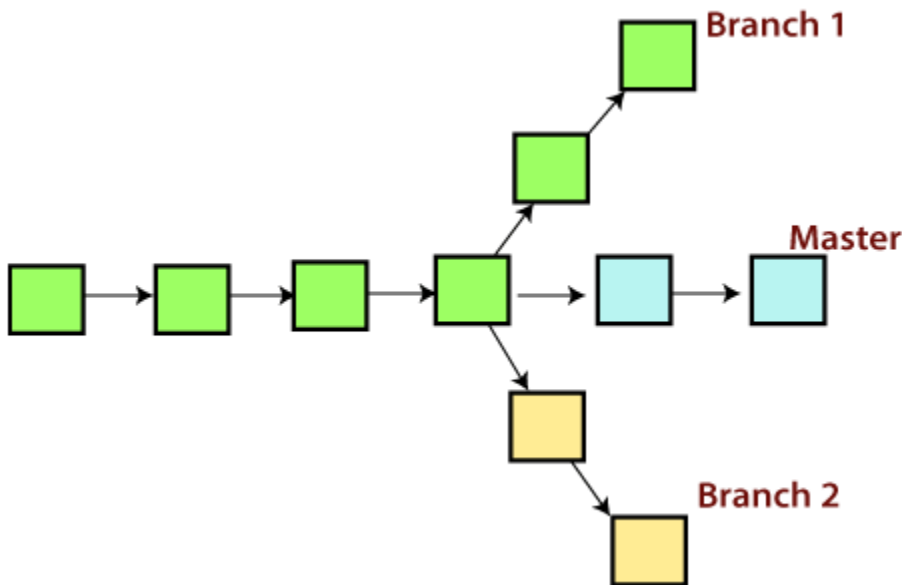
nothing added to commit but untracked files present (use "git add" to track)

F:\Anusha\FSD>
```

## 5. Branching and Merging Strategies

### Git Branch

A branch is a version of the repository that diverges from the main working project. It is a feature available in most modern version control systems. A Git project can have more than one branch. These branches are a pointer to a snapshot of your changes. When you want to add a new feature or fix a bug, you spawn a new branch to summarize your changes. So, it is complex to merge the unstable code with the main code base and also facilitates you to clean up your future history before merging with the main branch.



### Git Master Branch

The master branch is a default branch in Git. It is instantiated when first commit made on the project. When you make the first commit, you're given a master branch to the starting commit point. When you start making a commit, then master branch pointer automatically moves forward. A repository can have only one master branch.

Master branch is the branch in which all the changes eventually get merged back. It can be called as an official working version of your project.

### Operations on Branches

We can perform various operations on Git branches. The **git branch** command allows you to **create**, **list**, **rename** and **delete** branches. Many operations on branches are applied by git checkout and git merge command. So, the git branch is tightly integrated with the **git checkout** and **git merge** commands.

**The Operations that can be performed on a branch:**

#### Create Branch

You can create a new branch with the help of the **git branch** command. This command will be used as:

### Syntax:

1. \$ git branch <branch name>

### Output:

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git branch B1
```

This command will create the **branch B1** locally in Git directory.

## List Branch

You can List all of the available branches in your repository by using the following command.

Either we can use **git branch - list** or **git branch** command to list the available branches in the repository.

### Syntax:

1. \$ git branch --list

**or**

1. \$ git branch

### Output:

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git branch
  B1
  branch3
* master

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git branch --list
  B1
  branch3
* master
```

Here, both commands are listing the available branches in the repository. The symbol \* is representing currently active branch.

## Delete Branch

You can delete the specified branch. It is a safe operation. In this command, Git prevents you from deleting the branch if it has unmerged changes. Below is the command to do this.

### Syntax:



1. `$ git branch -d<branch name>`

#### Output:

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git branch -d B1
Deleted branch B1 (was 554a122).
```

This command will delete the existing branch B1 from the repository.

The **git branch d** command can be used in two formats. Another format of this command is **git branch D**. The **'git branch D'** command is used to delete the specified branch.

1. `$ git branch -D <branch name>`

## Delete a Remote Branch

You can delete a remote branch from Git desktop application. Below command is used to delete a remote branch:

#### Syntax:

1. `$ git push origin -delete <branch name>`

#### Output:

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git push origin --delete branch2
To https://github.com/ImDwivedi1/GitExample2
- [deleted]          branch2

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$
```

As you can see in the above output, the remote branch named **branch2** from my GitHub account is deleted.

## Switch Branch

Git allows you to switch between the branches without making a commit. You can switch between two branches with the **git checkout** command. To switch between the branches, below command is used:

1. `$ git checkout<branch name>`

#### Switch from master Branch

You can switch from master to any other branch available on your repository without making any commit.

#### Syntax:

1. \$ git checkout <**branch** name>

### Output:

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git checkout branch4
Switched to branch 'branch4'
```

As you can see in the output, branches are switched from **master** to **branch4** without making any commit.

## Rename Branch

We can rename the branch with the help of the **git branch** command. To rename a branch, use the below command:

### Syntax:

1. \$ git branch -m <**old** branch name><**new** branch name>

### Output:

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git branch -m branch4 renamedB1

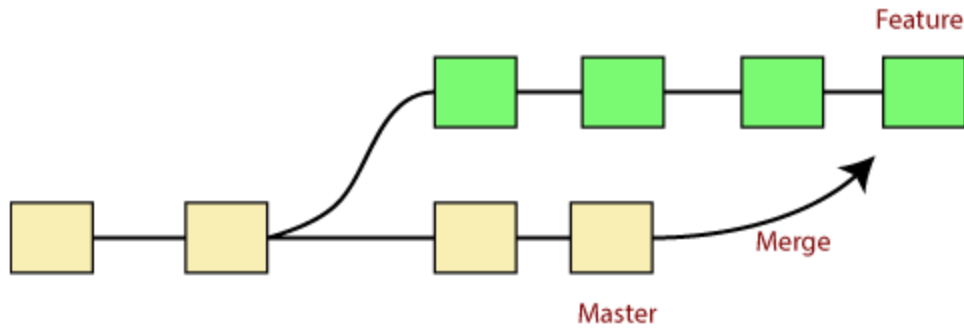
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git branch
* master
  renamedB1

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ |
```

As you can see in the above output, **branch4** renamed as **renamedB1**.

## Git Merge

In Git, the merging is a procedure to connect the forked history. It joins two or more development history together. The git merge command facilitates you to take the data created by git branch and integrate them into a single branch. Git merge will associate a series of commits into one unified history. Generally, git merge is used to combine two branches.



It is used to maintain distinct lines of development; at some stage, you want to merge the changes in one branch. It is essential to understand how merging works in Git.

In the above figure, there are two branches **master** and **feature**. We can see that we made some commits in both functionality and master branch, and merge them. It works as a pointer. It will find a common base commit between branches. Once Git finds a shared base commit, it will create a new "merge commit." It combines the changes of each queued merge commit sequence.

## The "git merge" command

The git merge command is used to merge the branches.

The syntax for the git merge command is as:

1. `$ git merge <query>`

It can be used in various context. Some are as follows:

### Scenario1: To merge the specified commit to currently active branch:

Use the below command to merge the specified commit to currently active branch.

1. `$ git merge <commit>`

The above command will merge the specified commit to the currently active branch. You can also merge the specified commit to a specified branch by passing in the branch name in <commit>. Let's see how to commit to a currently active branch.

See the below example. I have made some changes in my project's file **newfile1.txt** and committed it in my **test** branch.

```

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (test)
$ git add newfile1.txt

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (test)
$ git commit -m "edited newfile1.txt"
[test d2bb07d] edited newfile1.txt
1 file changed, 1 insertion(+), 1 deletion(-)

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (test)
$ git log
commit d2bb07dc9352e194b13075dcfd28e4de802c070b (HEAD -> test)
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Wed Sep 25 11:27:44 2019 +0530

    edited newfile1.txt

commit 2852e020909dfe705707695fd6d715cd723f9540 (test2, master)
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Wed Sep 25 10:29:07 2019 +0530

    newfile1 added

```

Copy the particular commit you want to merge on an active branch and perform the merge operation. See the below output:

```

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (test)
$ git checkout test2
Switched to branch 'test2'

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (test2)
$ git merge d2bb07dc9352e194b13075dcfd28e4de802c070b
Updating 2852e02..d2bb07d
Fast-forward
 newfile1.txt | 2 +-
1 file changed, 1 insertion(+), 1 deletion(-)

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (test2)
$

```

In the above output, we have merged the previous commit in the active branch test2.

## Merge Branch

Git allows you to merge the other branch with the currently active branch. You can merge two branches with the help of **git merge** command. Below command is used to merge the branches:

### Syntax:

1. \$ git merge <branch name>

### Output:

```

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git merge renamedB1
Already up to date.

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$

```

From the above output, you can see that **the master branch merged** with **renamedB1**. Since I have made no-commit before merging, so the output is showing as already up to date.

## Example:-

- Create three more text files in the local repository - “info1.txt”, “info2.txt”, “info3.txt”.



- Create a branch “first\_branch” and merge it to the main (master) branch.

```
SSPL-LP-DNS-YT0+SimpliLearn@SSPL-LP-DNS-YT01 MINGW64 ~/test (master)
$ git branch first_branch
```

The above command creates a branch.

```
SSPL-LP-DNS-YT0+SimpliLearn@SSPL-LP-DNS-YT01 MINGW64 ~/test (master)
$ git checkout first_branch
Switched to branch 'first_branch'
M       info.txt
```

The above command switches to the new branch from the master branch.

```
SSPL-LP-DNS-YT0+SimpliLearn@SSPL-LP-DNS-YT01 MINGW64 ~/test (first_branch)
$ git add info3.txt
```

The above command creates and adds “info3.txt” to the first\_branch.

```
SSPL-LP-DNS-YT0+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~/test (first_branch)
$ git commit -m "make some changes to first_branch"
[first_branch b14d609] make some changes to first_branch
Committer: Simplilearn <Simplilearn@SSPL-LP-DNS-YT01.blrsimplilearn.com>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly:

    git config --global user.name "Your Name"
    git config --global user.email you@example.com

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

    git commit --amend --reset-author

1 file changed, 1 insertion(+)
create mode 100644 info3.txt
```

The above command makes a commit to the first\_branch.

```
SSPL-LP-DNS-YT0+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~/test (first_branch)
$ ls
info.txt  info1.txt  info2.txt  info3.txt
```

The above command shows that the new branch has access to all the files.

```
SSPL-LP-DNS-YT0+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~/test (first_branch)
$ git checkout master
Switched to branch 'master'
M       info.txt
```

```
SSPL-LP-DNS-YT0+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~/test (master)
$ ls
info.txt  info1.txt  info2.txt
```

The above command shows that the master branch does not have an “info3.txt” file.

```

SSPL-LP-DNS-YT0+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~/test (master)
$ git merge first_branch
Updating 3ae78fd..b14d609
Fast-forward
 info3.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 info3.txt

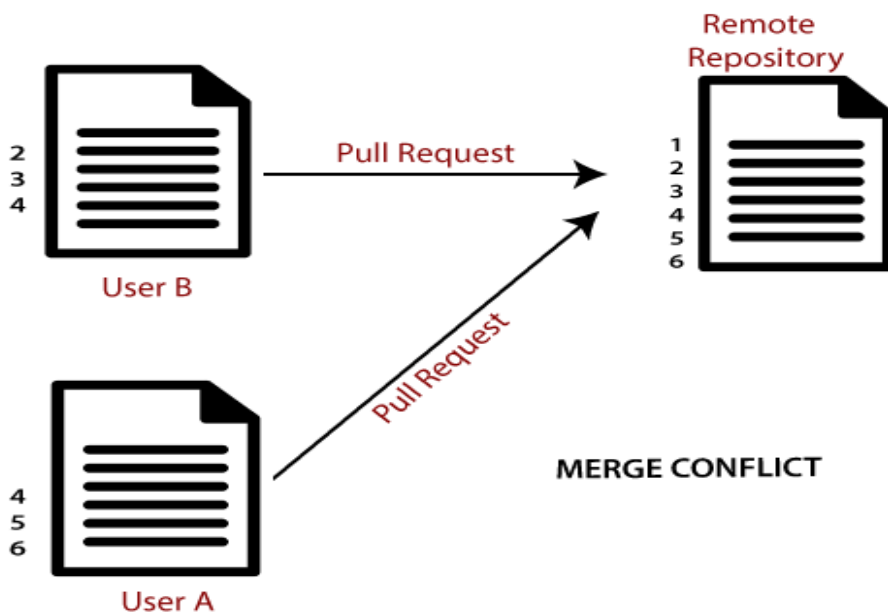
SSPL-LP-DNS-YT0+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~/test (master)
$ ls
info.txt  info1.txt  info2.txt  info3.txt

```

The above command is used to merge “first\_branch” with the master branch. Now, the master branch has “info3.txt” file.

## Git Merge Conflict

When two branches are trying to merge, and both are edited at the same time and in the same file, Git won't be able to identify which version is to take for changes. Such a situation is called merge conflict. If such a situation occurs, it stops just before the merge commit so that you can resolve the conflicts manually.



Let's understand it by an example.

Suppose my remote repository has cloned by two of my team member **user1** and **user2**. The user1 made changes as below in my projects index file.

```

bash_profile x index.html x index.html x
1  <head>
2  <body>
3  <title> This is a Git example</Title>
4  <h1> Git is a version control</h1>
5  </head>
6  </body>

```

Update it in the local repository with the help of git add command.

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/user1repo (master)
$ git add index.html
```

Now commit the changes and update it with the remote repository. See the below output:

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/user1repo (master)
$ git commit -m "edited by user1"
[master fe4ef27] edited by user1
1 file changed, 1 insertion(+)

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/user1repo (master)
$ git push origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 2 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 345 bytes | 345.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/ImDwivedi1/Git-Example
   039c01b..fe4ef27  master -> master

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/user1repo (master)
```

Now, my remote repository will look like this:

ImDwivedi1 edited by user1		Latest commit fe4ef27 6 minutes ago
Demo	Create Demo	9 days ago
README.md	Create README.md	29 days ago
index.html	edited by user1	6 minutes ago
new file	add new file	9 days ago
newfile2	newfile2	8 days ago

It will show the status of the file like edited by whom and when.

Now, at the same time, **user2** also update the index file as follows.

```
bash_profile x index.html x index.html x
1  <head>
2  <body>
3  <title> This is a Git example</Title>
4  <h2> Git is a version control system</h2>
5  </head>
6  </body>
```

User2 has added and committed the changes in the local repository. But when he tries to push it to remote server, it will throw errors. See the below output:



```

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/user2repo (master)
$ git add index.html

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/user2repo (master)
$ git commit -m " edited by user2"
[master 3ee71e0] edited by user2
1 file changed, 1 insertion(+)

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/user2repo (master)
$ git push origin master
To https://github.com/ImDwivedi1/Git-Example
! [rejected] master -> master (fetch first)
error: failed to push some refs to 'https://github.com/ImDwivedi1/Git-Example'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/user2repo (master)
$

```

In the above output, the server knows that the file is already updated and not merged with other branches. So, the push request was rejected by the remote server. It will throw an error message like **[rejected] failed to push some refs to <remote URL>**. It will suggest you to pull the repository first before the push. See the below command:

```

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/user2repo (master)
$ git pull --rebase origin master
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 3 (delta 1), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/ImDwivedi1/Git-Example
* branch master -> FETCH_HEAD
039c01b..fe4ef27 master -> origin/master
First, rewinding head to replay your work on top of it...
Applying: edited by user2
Using index info to reconstruct a base tree...
M index.html
Falling back to patching base and 3-way merge...
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
error: failed to merge in the changes.
hint: Use 'git am --show-current-patch' to see the failed patch
Patch failed at 0001 edited by user2
Resolve all conflicts manually, mark them as resolved with
"git add/rm <conflicted_files>", then run "git rebase --continue".
You can instead skip this commit: run "git rebase --skip".
To abort and get back to the state before "git rebase", run "git rebase --abort"
.

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/user2repo (master|REBASE 1/1)
$ |

```

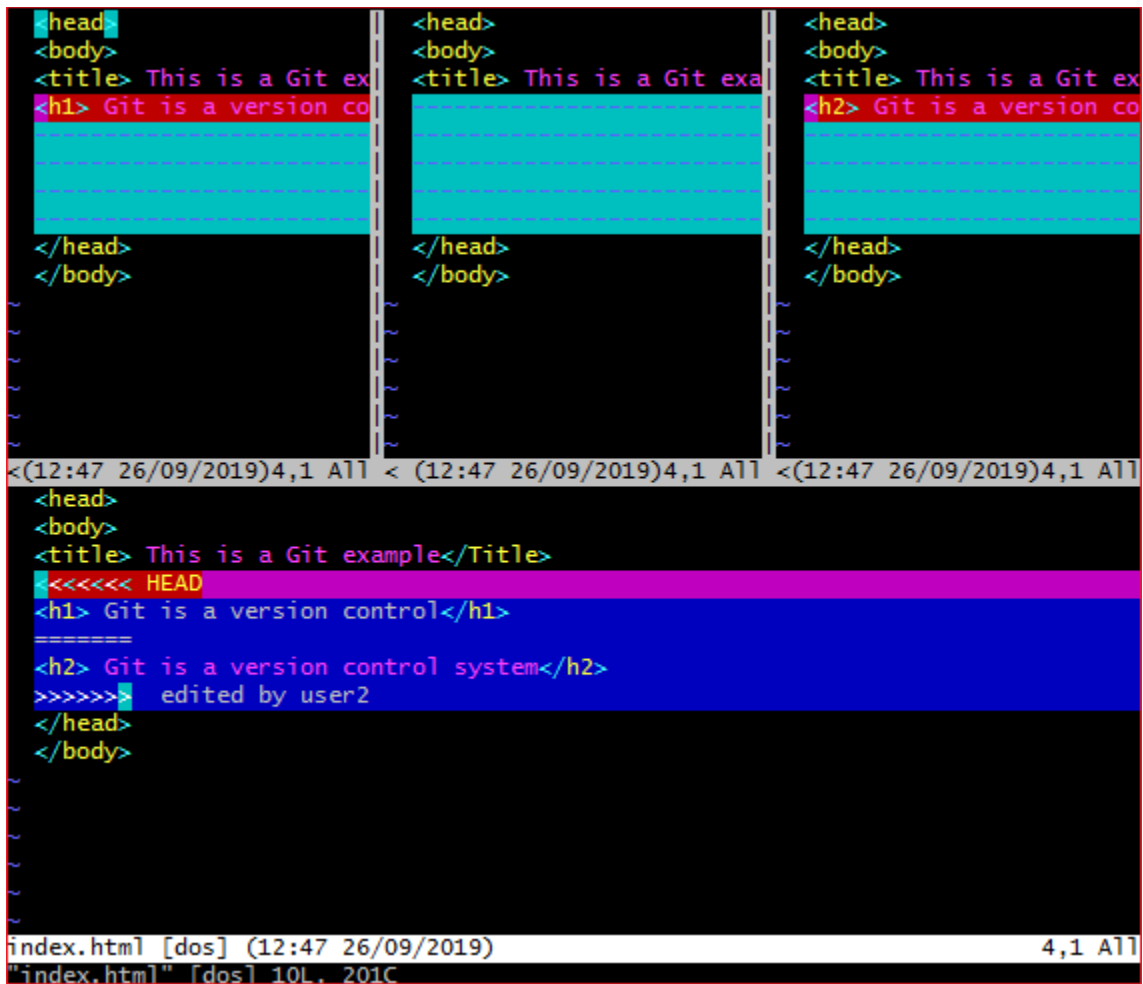
In the given output, git rebase command is used to pull the repository from the remote URL. Here, it will show the error message like **merge conflict in <filename>**.

# Resolve Conflict:

To resolve the conflict, it is necessary to know whether the conflict occurs and why it occurs. Git merge tool command is used to resolve the conflict. The merge command is used as follows:

1. \$ git mergetool

In my repository, it will result in:



```
<head>
<body>
<title> This is a Git example
<h1> Git is a version control system
</head>
</body>

<head>
<body>
<title> This is a Git example
<h1> Git is a version control system
</head>
</body>

<head>
<body>
<title> This is a Git example
<h2> Git is a version control system
</head>
</body>

<(12:47 26/09/2019)4,1 All < (12:47 26/09/2019)4,1 All <(12:47 26/09/2019)4,1 All
<head>
<body>
<title> This is a Git example</Title>
<<<<<< HEAD
<h1> Git is a version control</h1>
=====
<h2> Git is a version control system</h2>
>>>>>> edited by user2
</head>
</body>

index.html [dos] (12:47 26/09/2019) 4,1 All
"index.html" [dos] 10L, 201C
```

The above output shows the status of the conflicted file. To resolve the conflict, enter in the insert mode by merely pressing **I** key and make changes as you want. Press the **Esc** key, to come out from insert mode. Type the: **w!** at the bottom of the editor to save and exit the changes. To accept the changes, use the rebase command. It will be used as follows:

1. \$ git rebase --continue

Hence, the conflict has resolved. See the below output:

```

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/user2repo (master|REBASE 1/1)
$ git rebase --continue
Applying: edited by user2

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/user2repo (master)
$ git push origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 2 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 373 bytes | 124.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/ImDwivedi1/Git-Example
fe4ef27..b3db7dc master -> master

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/user2repo (master)
$

```

In the above output, the conflict has resolved, and the local repository is synchronized with a remote repository.

To see that which is the first edited text of the merge conflict in your file, search the file attached with conflict marker <<<<<<. You can see the changes from the **HEAD** or base branch after the line <<<<<< **HEAD** in your text editor. Next, you can see the divider like =====. It divides your changes from the changes in the other branch, **followed by >>>>>> BRANCH-NAME**. In the above example, user1 wrote "<h1>Git is a version control</h1>" in the base or HEAD branch and user2 wrote "<h2>Git is a version control</h2>".

Decide whether you want to keep only your branch's changes or the other branch's changes, or create a new change. Delete the conflict markers <<<<<<, =====, >>>>>> and create final changes you want to merge.

## 6. Working with Remote Repositories

### What is GitHub?

GitHub is a Git repository hosting service. GitHub also facilitates with many of its features, such as access control and collaboration. It provides a Web-based graphical interface.

GitHub is an American company. It hosts source code of your project in the form of different programming languages and keeps track of the various changes made by programmers.

It offers both **distributed version control and source code management (SCM)** functionality of Git. It also facilitates with some collaboration features such as bug tracking, feature requests, task management for every project.

#### What is a repository in GitHub?

A repository is the most basic element of GitHub. It's a place where you can store your code, your files, and each file's revision history. Repositories can have multiple collaborators and can be either public or private. To create a new repository, go to <https://github.com/new>.



## • Features of GitHub

GitHub is a place where programmers and designers work together. They collaborate, contribute, and fix bugs together. It hosts plenty of open source projects and codes of various programming languages.

Some of its significant features are as follows.

- Collaboration
- Integrated issue and bug tracking
- Graphical representation of branches
- Git repositories hosting
- Project management
- Team management
- Code hosting
- Track and assign tasks
- Conversations
- **Wikisc:-** Every repository on GitHub.com comes equipped with a section for hosting documentation, called a wiki. You can use your repository's wiki to share long-form content about your project, such as how to use it, how you designed it, or its core principles.

## • Benefits of GitHub

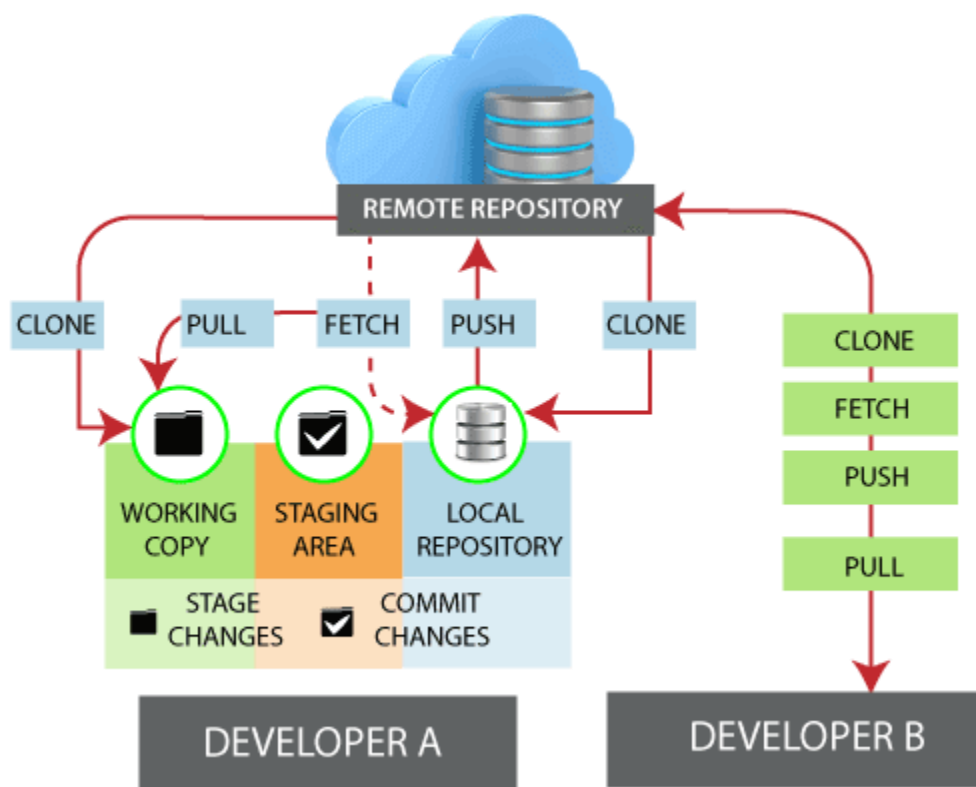
GitHub can be separated as the Git and the Hub. GitHub service includes access controls as well as collaboration features like task management, repository hosting, and team management.

The key benefits of GitHub are as follows.

- It is easy to contribute to open source projects via GitHub.
- It helps to create an excellent document.
- You can attract recruiter by showing off your work. If you have a profile on GitHub, you will have a higher chance of being recruited.
- It allows your work to get out there in front of the public.
- You can track changes in your code across versions.

## Working with Remote Repository

The developers can perform many operations with the remote server. These operations can be a clone, fetch, push, pull, and more. Consider the below image:



a. First Create GitHub Account and Login

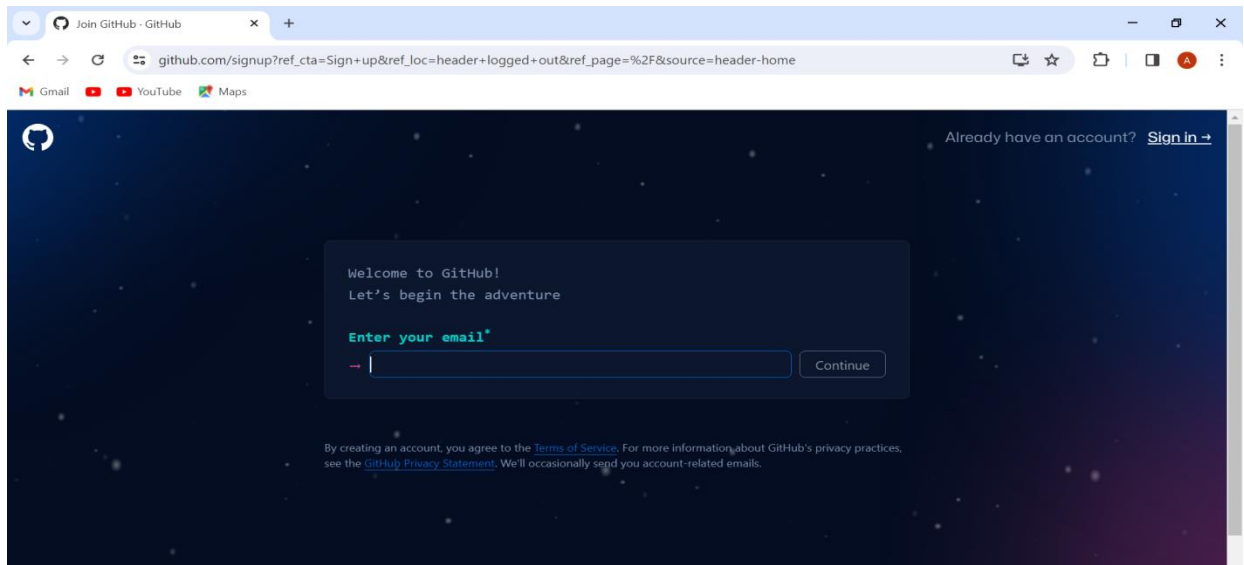


Fig.1.Github sign in and sign up page

b. Next create new Repository name week1.0

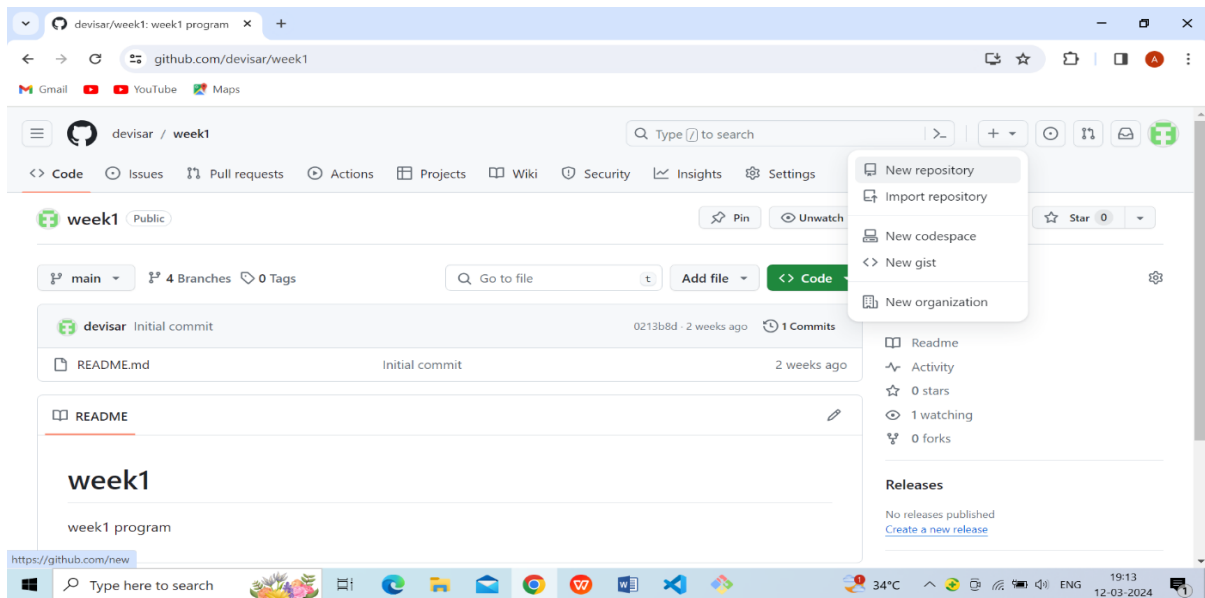


Fig.2.creating new repository page

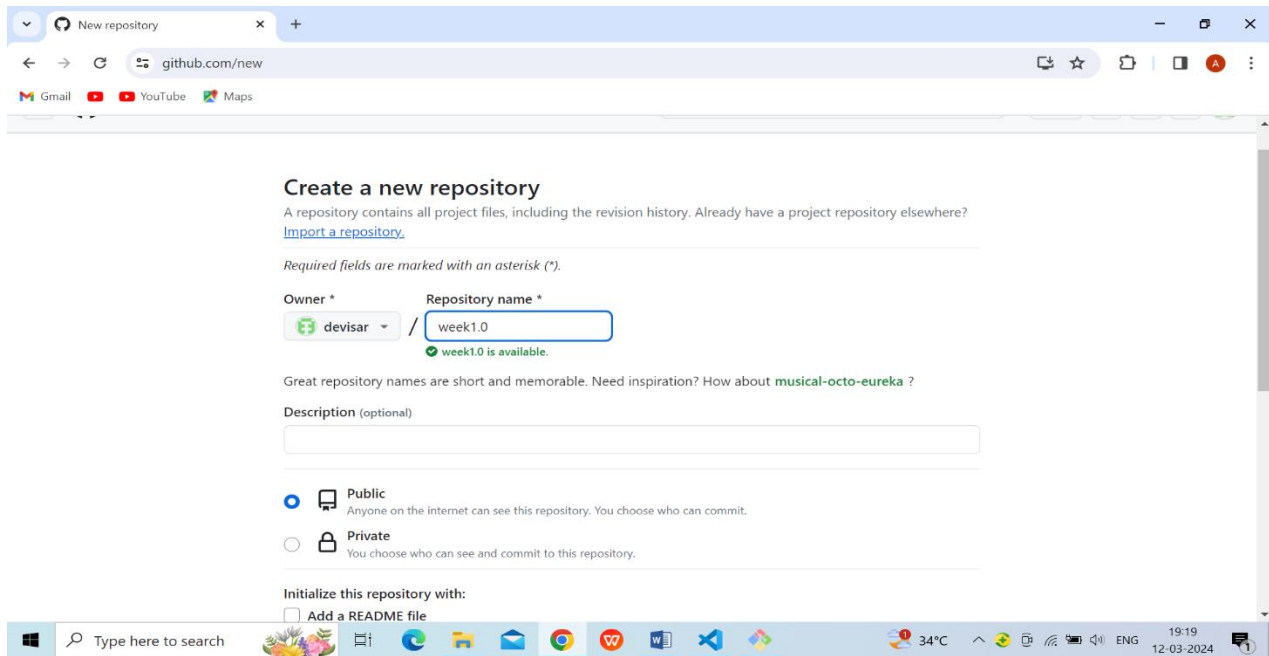


Fig.3.Enter repository name page

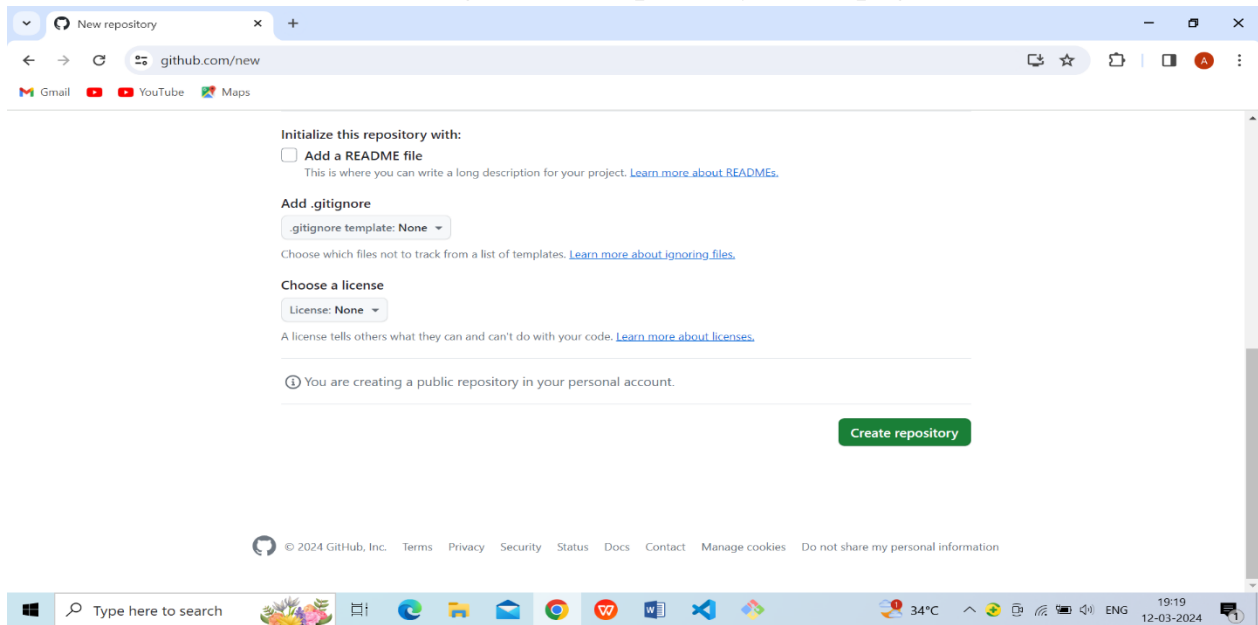


Fig.4.Click button to create repository page

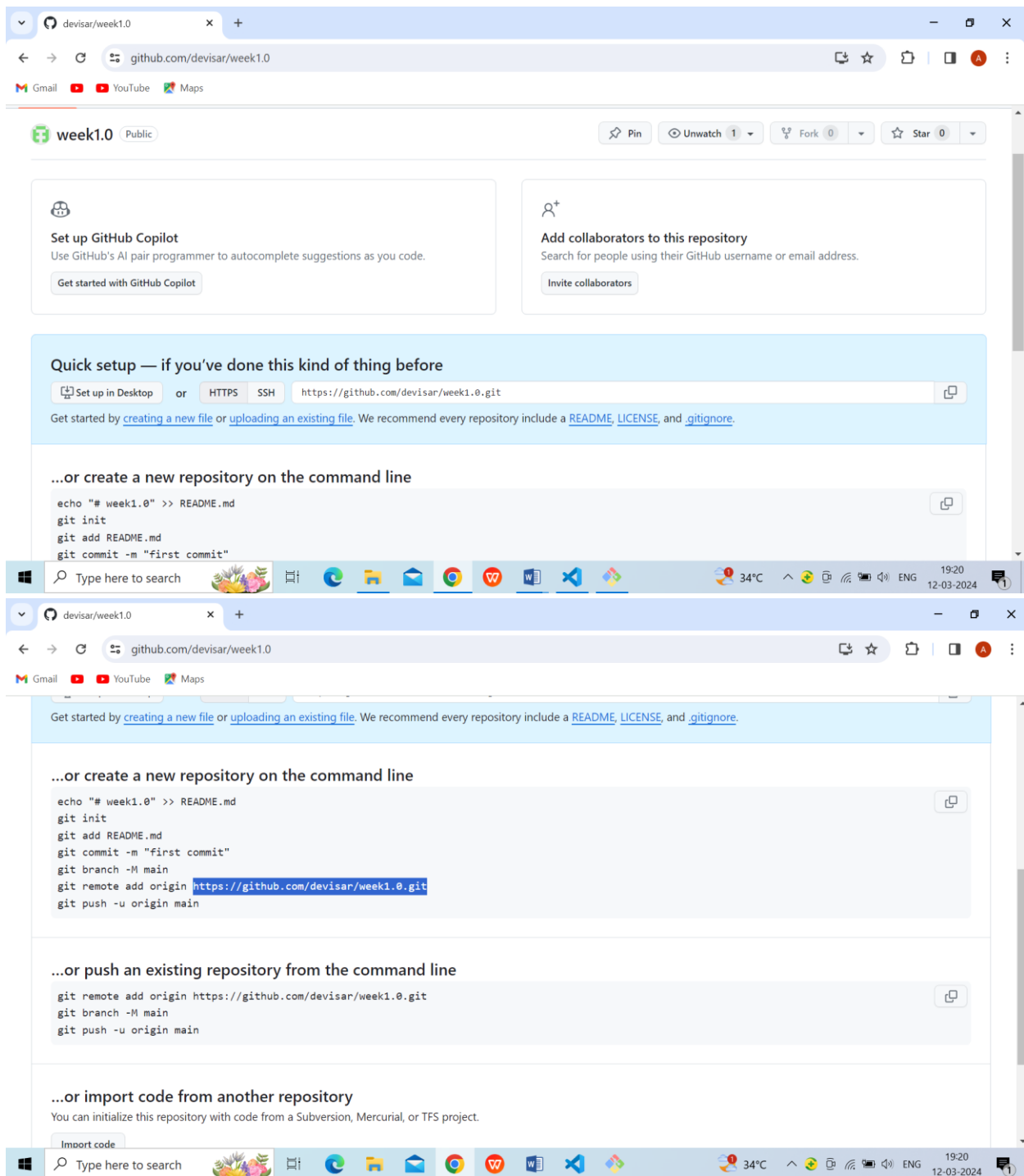


Fig.5.After opening page

- Connect the local repository to your remote repository.

## 1.git remote

- The git remote command is used to create, view, and delete connections to other repositories.



- The connections here are not like direct links into other repositories, but as bookmarks that serve as convenient names to be used as a reference.

## Git remote add origin <address>

```
Admin@DESKTOP-T3DAU1H MINGW64 /f/Anusha/week1 (master)
$ git remote add origin "https://github.com/devisar/week1.0.git"

Admin@DESKTOP-T3DAU1H MINGW64 /f/Anusha/week1 (master)
$ git remote -v
origin https://github.com/devisar/week1.0.git (fetch)
origin https://github.com/devisar/week1.0.git (push)
```

- Push the file to the remote repository.

## 2.git push

- The command `git push` is used to transfer the commits or pushing the content from the local repository to the remote repository.
- The command is used after a local repository has been modified, and the modifications are to be shared with the remote team members.

## git push -u origin master

```
Admin@DESKTOP-T3DAU1H MINGW64 /f/Anusha/week1 (master)
$ git push -u origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 688 bytes | 344.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/devisar/week1.0.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
```

Refresh your repository page on GitHub. You will get your local file on your remote GitHub repository

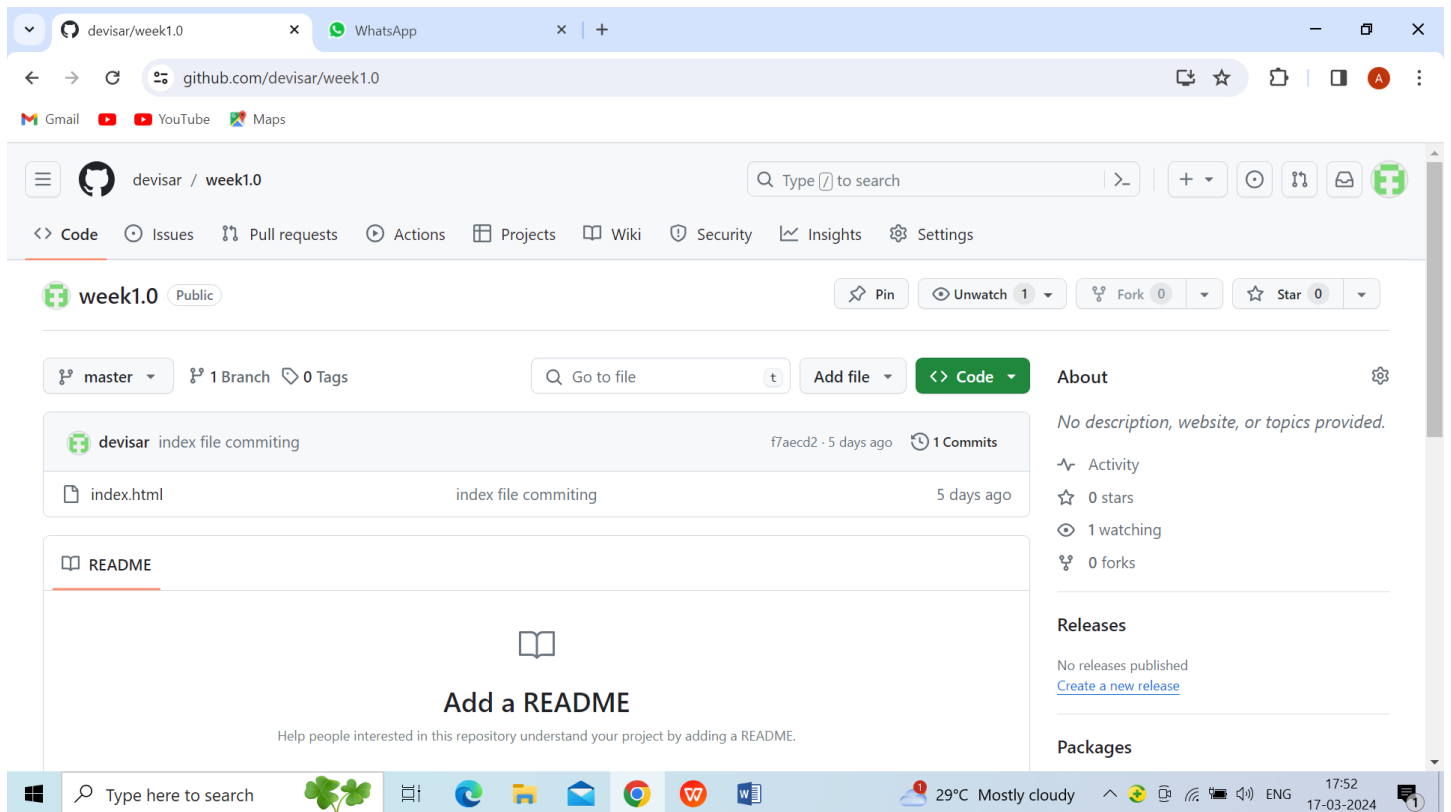


Fig.6.Now File is uploaded

## 3.remote commands

### 3.1 Check your Remote

To check the configuration of the remote server, run the **git remote** command. The git remote command allows accessing the connection between remote and local. If you want to see the original existence of your cloned repository, use the git remote command. It can be used as:

**git remote**

And

**git remote -v**

```
F:\Anusha\FSD>git remote
origin

F:\Anusha\FSD>git remote -v
origin https://github.com/devisar/FSD-LAB.git (fetch)
origin https://github.com/devisar/FSD-LAB.git (push)

F:\Anusha\FSD>
```

## 3.2 Remove Remote

You can remove a remote connection from a repository. To remove a connection, perform the git remote command with **remove** or **rm** option. It can be done as:

**Syntax:**

1. **\$git remote rm <destination>**

Or

1. **\$ git remote remove <destination>**

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git remote rm origin

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git remote -v

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$
```

In the above output, I have removed remote server "origin" from my repository.

## 3.3 Git Remote Rename

Git allows renaming the remote server name so that you can use a short name in place of the remote server name. Below command is used to rename the remote server:

**Syntax:**

1. **\$ git remote rename <old name><new name>**

**Output:**

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git remote rename origin hd

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git remote -v
hd      https://github.com/ImDwivedi1/GitExample2 (fetch)
hd      https://github.com/ImDwivedi1/GitExample2 (push)
```

In the above output, I have renamed my default server name origin to hd. Now, I can operate using this name in place of origin. Consider the below output:

```

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git pull hd master
From https://github.com/ImDwivedi1/GitExample2
 * branch          master       -> FETCH_HEAD
 * [new branch]     master       -> hd/master
Already up to date.

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git pull origin master
fatal: 'origin' does not appear to be a git repository
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.

```

In the above output, I have pulled the remote repository using the server name `hd`. But, when I am using the old server name, it is throwing an error with the message "'origin' does not appear to be a git repository." It means Git is not identifying the old name, so all the operations will be performed by a new name.

### 3.4 Git Change Remote (Changing a Remote's URL)

We can change the URL of a remote repository. The `git remote set` command is used to change the URL of the repository. It changes an existing remote repository URL.

#### Git Remote Set:

We can change the remote URL simply by using the `git remote set` command. Suppose we want to make a unique name for our project to specify it. Git allows us to do so. It is a simple process. To change the remote URL, use the below command:

1. **\$ git remote set-url <remote name><newURL>**

The **remote set-url** command takes two types of arguments. The first one is `<remote name>`, it is your current server name for the repository. The second argument is `<newURL>`, it is your new URL name for the repository. The `<new URL>` should be in below format: **`https://github.com/URLChanged`**

Consider the below image:

```

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git remote set-url origin https://github.com/URLChanged

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git remote -v
origin https://github.com/URLChanged (fetch)
origin https://github.com/URLChanged (push)

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$

```

In the above output, I have changed my existing repository URL as **https://github.com/URLChanged** from **https://github.com/ImDwivedi1/GitExample2**. It can be understood by my URL name that I have changed this. To check the latest URL, perform the below command:

## 4. Git Clone Command

The **git clone** is a command-line utility which is used to make a local copy of a remote repository. It accesses the repository through a remote URL.

Usually, the original repository is located on a remote server, often from a Git service like GitHub, Bitbucket, or GitLab. The remote repository URL is referred to the **origin**.

### Syntax:

1. \$ git clone <repository URL>
- 

## Git Clone Repository

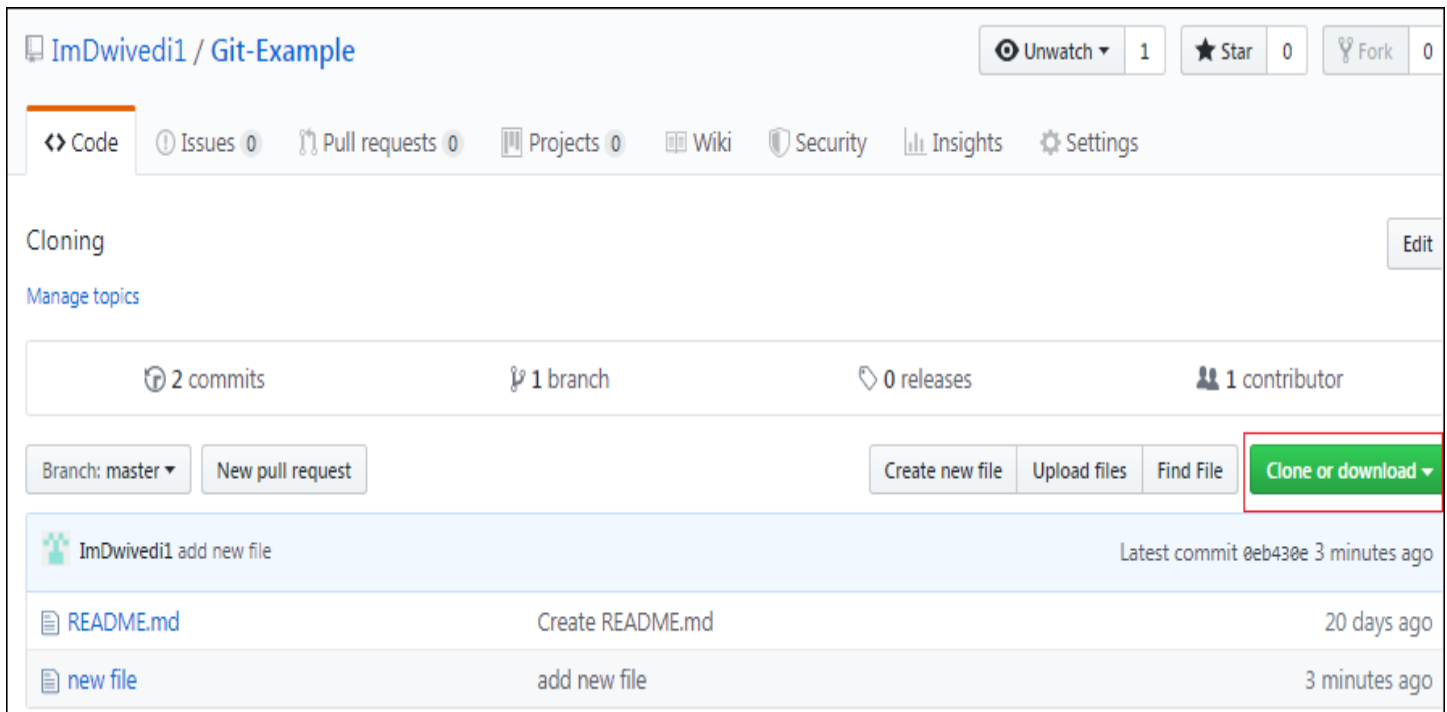
Suppose, you want to clone a repository from GitHub, or have an existing repository owned by any other user you would like to contribute. Steps to clone a repository are as follows:

Step 1:

Open GitHub and navigate to the main page of the repository.

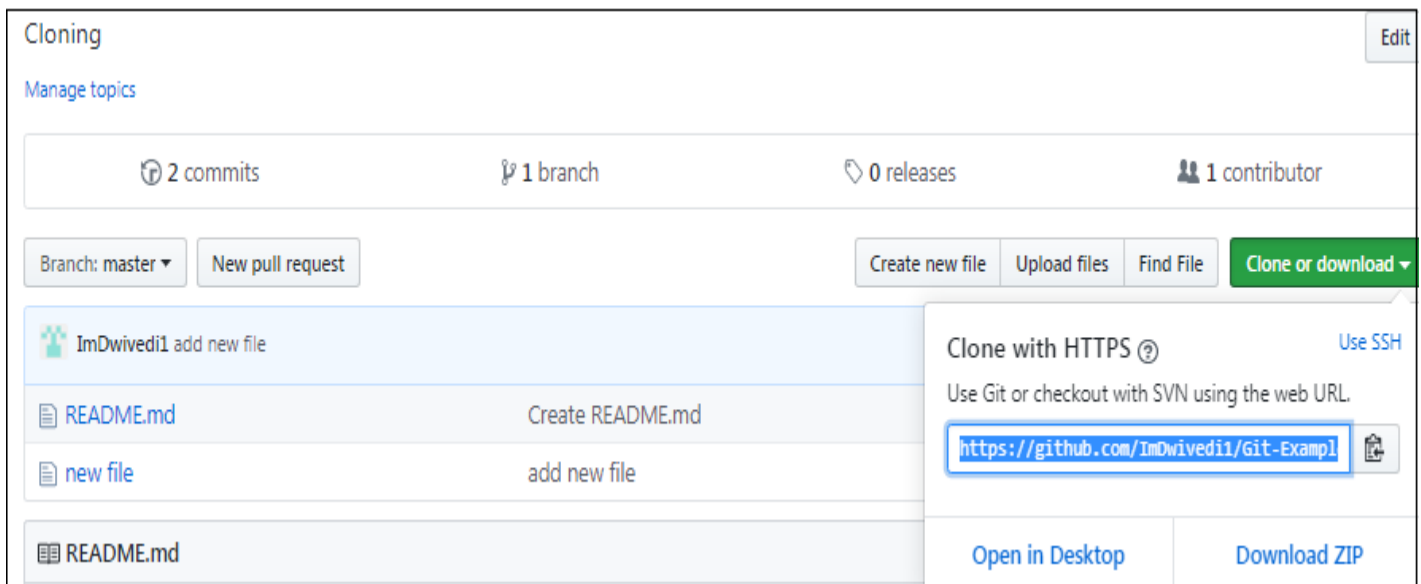
Step 2:

Under the repository name, click on **Clone or download**.



Step 3:

Select the **Clone with HTTPS** section and **copy the clone URL** for the repository. For the empty repository, you can copy the repository page URL from your browser and skip to next step.



Step 4:

Open Git Bash and change the current working directory to your desired location where you want to create the local copy of the repository.

Step 5:

Use the git clone command with repository URL to make a copy of the remote repository. See the below command:

1. `$ git clone https://github.com/ImDwivedi1/Git-Example.git`

Now, Press Enter. Hence, your local cloned repository will be created. See the below output:

```
HiManshU@HiManshU-PC MINGW64 ~/Desktop (master)
$ cd "new folder"

HiManshU@HiManshU-PC MINGW64 ~/Desktop/new folder (master)
$ git clone https://github.com/ImDwivedi1/Git-Example.git
Cloning into 'Git-Example'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (6/6), done.

HiManshU@HiManshU-PC MINGW64 ~/Desktop/new folder (master)
$
```

## Git Clone Branch

Git allows making a copy of only a particular branch from a repository. You can make a directory for the individual branch by using the git clone command. To make a clone branch, you need to specify the branch name with -b command. Below is the syntax of the command to clone the specific git branch:

### Syntax:

1. `$ git clone -b <Branch name> <Repository URL>`

See the below command:

1. `$ git clone -b master https://github.com/ImDwivedi1/Git-Example.git "new folder(2)"`

```
HiManshU@HiManshU-PC MINGW64 ~/Desktop/new folder(2) (master)
$ git clone -b master https://github.com/ImDwivedi1/Git-Example.git
Cloning into 'Git-Example'...
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 9 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (9/9), done.

HiManshU@HiManshU-PC MINGW64 ~/Desktop/new folder(2) (master)
$ |
```

In the given output, only the master branch is cloned from the principal repository Git-Example.

## 5. "git pull" command

The pull command is used to access the changes (commits) from a remote repository to the local repository. It updates the local branches with the remote-tracking branches. Remote tracking branches are branches that have been set up to push and pull from the remote repository. Generally, it is a collection of the fetch and merges command. First, it fetches the changes from remote and combined them with the local repository.

The syntax of the git pull command is given below:

### Syntax:

1. `$ git pull <option> [<repository URL> <refspec>...]`

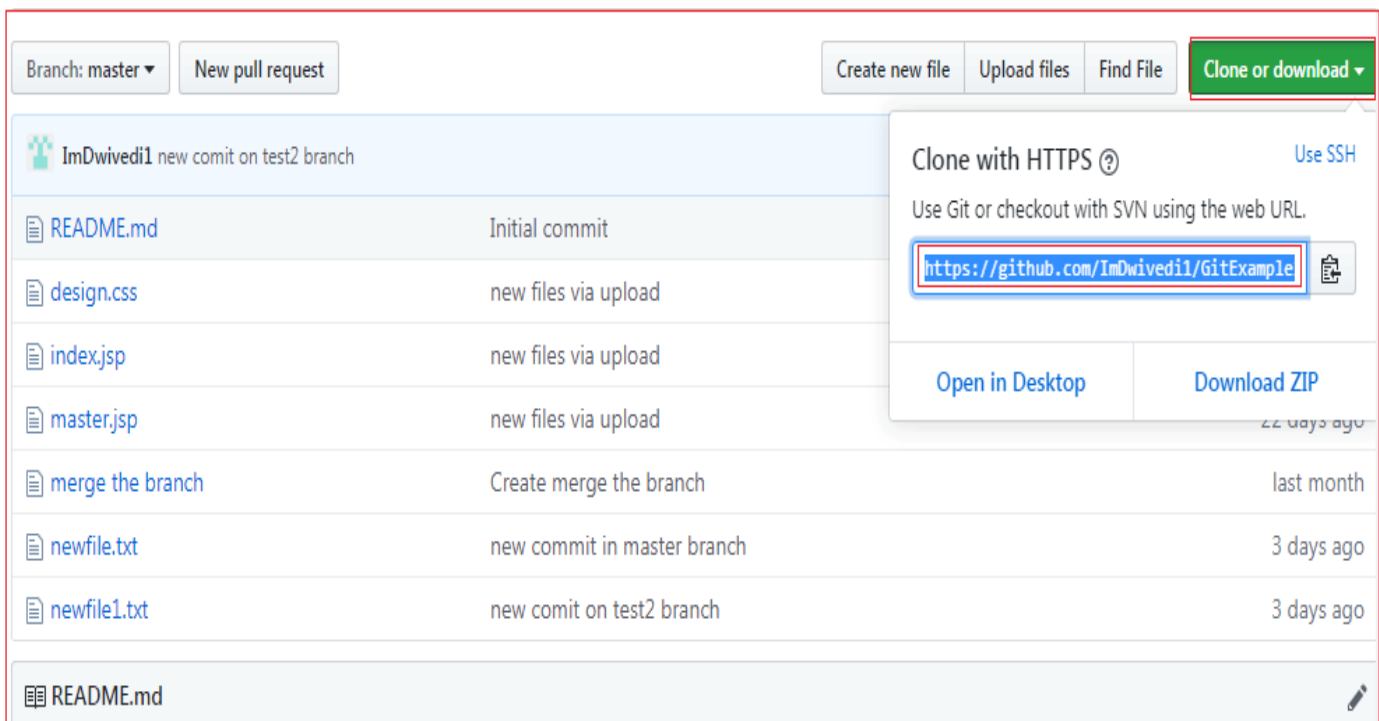
In which:

**<option>:** Options are the commands; these commands are used as an additional option in a particular command. Options can be **-q** (quiet), **-v** (verbose), **-e**(edit) and more.

**<repository URL>:** Repository URL is your remote repository's URL where you have stored your original repositories like GitHub or any other git service. This URL looks like:

1. `https://github.com/ImDwivedi1/GitExample2.git`

To access this URL, go to your account on GitHub and select the repository you want to clone. After that, click on the **clone** or **download** option from the repository menu. A new pop up window will open, select **clone with https option** from available options. See the below screenshot:





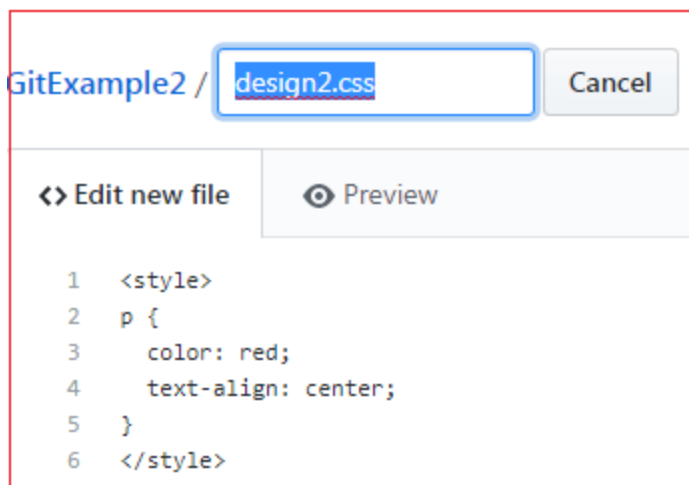
Copy the highlighted URL. This URL is used to Clone the repository.

**<Refspec>:** A ref is referred to commit, for example, head (branches), tags, and remote branches. You can check head, tags, and remote repository in **.git/ref** directory on your local repository. **Refspec** specifies and updates the refs.

### How to use pull:

It is essential to understand how it works and how to use it. Let's take an example to understand how it works and how to use it. Suppose I have added a new file say **design2.css** in my remote repository of project GitExample2.

To create the file first, go to create a file option given on repository sub-functions. After that, select the file name and edit the file as you want. Consider the below image.



Go to the bottom of the page, select a commit message and description of the file. Select whether you want to create a new branch or commit it directly in the master branch. Consider the below image:

## Commit new file

CSS file

See the proposed CSS file.

☒ Commit directly to the `master` branch.  
☐ Create a new branch for this commit and start a pull request. [Learn more about pull requests.](#)

Commit new file

Cancel

Now, we have successfully committed the changes.

To pull these changes in your local repository, perform the `git pull` operation on your cloned repository. There are many specific options available for pull command. Let's have a look at some of its usage.

## Default git pull:

We can pull a remote repository by just using the `git pull` command. It's a default option. Syntax of `git pull` is given below:

### Syntax:

1. `$ git pull`

### Output:

```

HiManShU@HiManShU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git pull
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/ImDwivedi1/GitExample2
   f1ddc7c..0a1a475  master       -> origin/master
Updating f1ddc7c..0a1a475
Fast-forward
 design2.css | 6 ++++++
 1 file changed, 6 insertions(+)
 create mode 100644 design2.css

HiManShU@HiManShU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ |

```

In the given output, the newly updated objects of the repository are fetched through the git pull command. It is the default version of the git pull command. It will update the newly created file **design2.css** file and related object in the local repository. See the below image.

Name	Date modified	Type	Size
.git	10/1/2019 2:47 PM	File folder	
newfolder3	9/21/2019 11:37 AM	File folder	
design	9/19/2019 6:10 PM	Cascading Style S...	1 KB
design2	10/1/2019 2:47 PM	Cascading Style S...	1 KB
index	9/19/2019 6:10 PM	JSP File	2 KB
master	9/19/2019 6:10 PM	JSP File	1 KB
merge the branch	9/20/2019 6:05 PM	File	1 KB
newfile	9/28/2019 12:56 PM	Text Document	1 KB
newfile1	9/28/2019 4:01 PM	Text Document	1 KB
README	9/19/2019 6:10 PM	MD File	1 KB

As you can see in the above output, the design2.css file is added to the local repository. The git pull command is equivalent to **git fetch origin head** and **git merge head**. The head is referred to as the ref of the current branch.

## Git Pull Remote Branch

Git allows fetching a particular branch. Fetching a remote branch is a similar process, as mentioned above, in **git pull command**. The only difference is we have to copy the URL of the particular branch we want to pull. To do so, we will select a specific branch. See the below image:

Branch: edited ▾ View #2

Create new file Upload files Find File Clone or download ▾

This branch is 1 commit ahead, 8 commits behind master. #2 Compare

ImDwivedi1 Update Index.jsp ... Latest commit 00fcce5 17 days ago

README.md	Initial commit	last month
design.css	new files via upload	22 days ago
index.jsp	Update Index.jsp	17 days ago
master.jsp	new files via upload	22 days ago
merge the branch	Create merge the branch	last month

README.md

In the above screenshot, I have chosen my branch named **edited** to copy the URL of the edited branch. Now, I am going to pull the data from the edited branch. Below command is used to pull a remote branch:

### Syntax:

1. `$ git pull <remote branch URL>`

### Output:

```
HiManshu@HiManshu-PC MINGW64 ~/Desktop/Demo (master)
$ git pull https://github.com/ImDwivedi1/GitExample2.git
remote: Enumerating objects: 38, done.
remote: Counting objects: 100% (38/38), done.
remote: Compressing objects: 100% (25/25), done.
remote: Total 38 (delta 13), reused 19 (delta 7), pack-reused 0
Unpacking objects: 100% (38/38), done.
From https://github.com/ImDwivedi1/GitExample2
* branch          HEAD      -> FETCH_HEAD

HiManshu@HiManshu-PC MINGW64 ~/Desktop/Demo (master)
$
```

In the above output, the remote branch **edited** has copied.

## 6.git fetch command

The **"git fetch" command** is used to pull the updates from remote-tracking branches. Additionally, we can get the updates that have been pushed to our remote branches to our local machines. As we know, a branch is a variation of our repositories main code, so the remote-tracking branches are branches that have been set up to pull and push from remote repository.

# How to fetch Git Repository

We can use fetch command with many arguments for a particular data fetch. See the below scenarios to understand the uses of fetch command.

## Scenario 1: To fetch the remote repository:

We can fetch the complete repository with the help of fetch command from a repository URL like a pull command does. See the below output:

### Syntax:

1. `$ git fetch < repository Url >`

### Output:

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/Git-Example (master)
$ git fetch https://github.com/ImDwivedi1/Git-Example.git
warning: no common commits
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (6/6), done.
From https://github.com/ImDwivedi1/Git-Example
 * branch                HEAD              -> FETCH_HEAD

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/Git-Example (master)
$ |
```

In the above output, the complete repository has fetched from a remote URL.

If you want to add working directory use command **git merge**

## Differences between Git and GitHub:-

**Git:** Git is a distributed version control system for tracking changes in source code during software development. It is designed for coordinating work among programmers, but it can be used to track changes in any set of files. Its goals include speed, data integrity, and support for distributed, non-linear workflows.

**GitHub:** GitHub is a web-based Git repository hosting service, which offers all of the distributed revision control and source code management (SCM) functionality of Git as well as adding its own features.

Below is a table of differences between Git and GitHub:

S.No.	Git	GitHub
-------	-----	--------

<b>S.No.</b>	<b>Git</b>	<b>GitHub</b>
1.	Git is a software.	GitHub is a service.
2.	Git is a command-line tool	GitHub is a graphical user interface
3.	Git is installed locally on the system	GitHub is hosted on the web
4.	Git is maintained by linux.	GitHub is maintained by Microsoft.
5.	Git is focused on version control and code sharing.	GitHub is focused on centralized source code hosting.
6.	Git is a version control system to manage source code history.	GitHub is a hosting service for Git repositories.
7.	Git was first released in 2005.	GitHub was launched in 2008.
8.	Git has no user management feature.	GitHub has a built-in user management feature.
9.	Git is open-source licensed.	GitHub includes a free-tier and pay-for-use tier.
10.	Git has minimal external tool configuration.	GitHub has an active marketplace for tool integration.
11.	Git provides a Desktop interface named GitGui.	GitHub provides a Desktop interface named GitHub Desktop.
12.	Git competes with CVS, Azure DevOps Server, Subversion, Mercurial, etc.	GitHub competes with GitLab, Bit Bucket, AWS Code Commit, etc.

## Key Git Commands

- **git init:** Initialize a new repository.
- **git clone <repo-url>:** Clone a remote repository to your local machine.
- **git status:** Check the status of the working directory.
- **git add <file>:** Add changes to the staging area.
- **git commit -m "message":** Commit the staged changes.
- **git push origin <branch>:** Push changes to a remote repository.
- **git pull origin <branch>:** Fetch and merge changes from a remote repository.
- **git branch:** View local branches.
- **git checkout <branch>:** Switch to a different branch.
- **git merge <branch>:** Merge changes from one branch into another.
- **git log:** View commit history.
- **git diff:** View the changes between commits or working files.
- **git branch -d <branch>:** Delete a local branch.