



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.

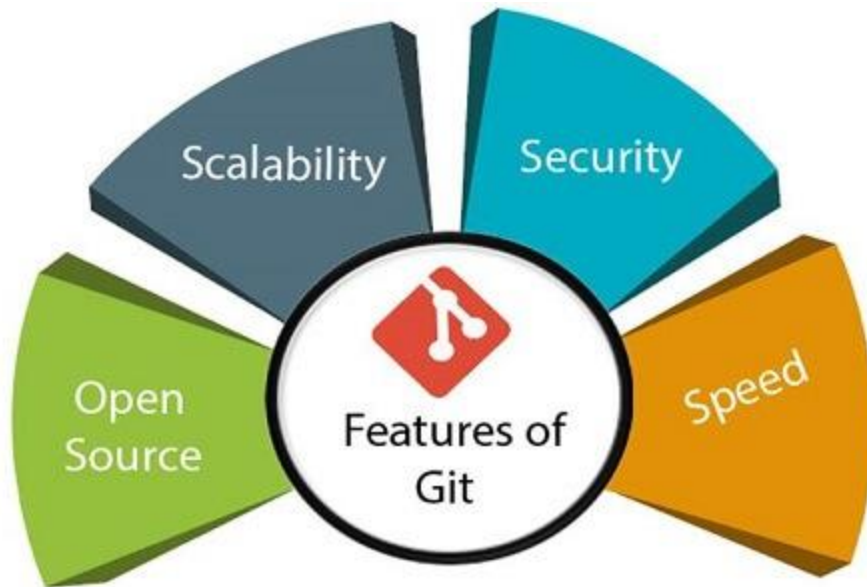
Git is foundation of many services like **GitHub** and **GitLab**, but we can use Git without using any other Git services. Git can be used **privately** and **publicly**.

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**.

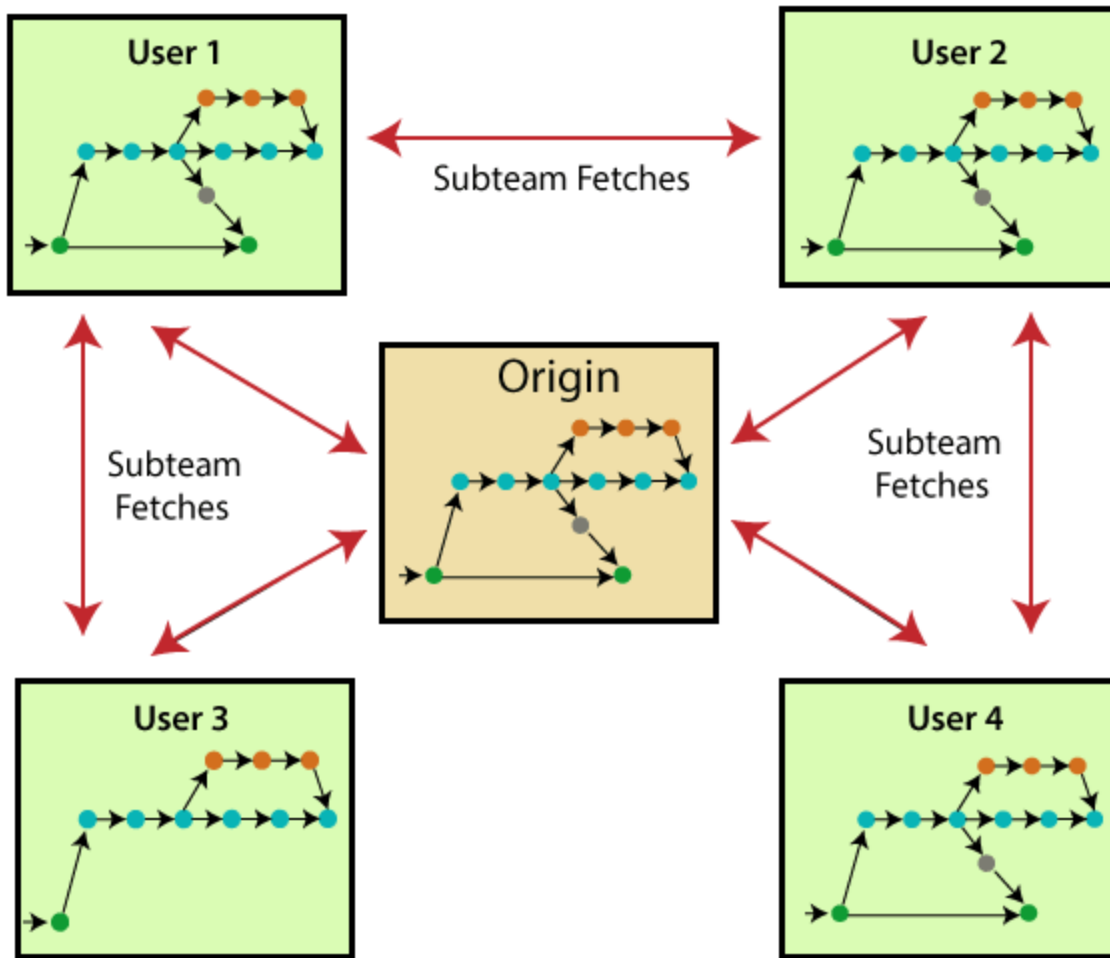
Git is easy to learn, and has fast performance. It is superior to other SCM tools like Subversion, CVS, Perforce, and ClearCase.

Features of Git

Some remarkable features of Git are as follows:



- **Open Source**
Git is an **open-source tool**. It is released under the **GPL** (General Public License) license.
- **Scalable**
Git is **scalable**, which means when the number of users increases, the Git can easily handle such situations.
- **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.



- **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.

- **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.

- **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.

- **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.

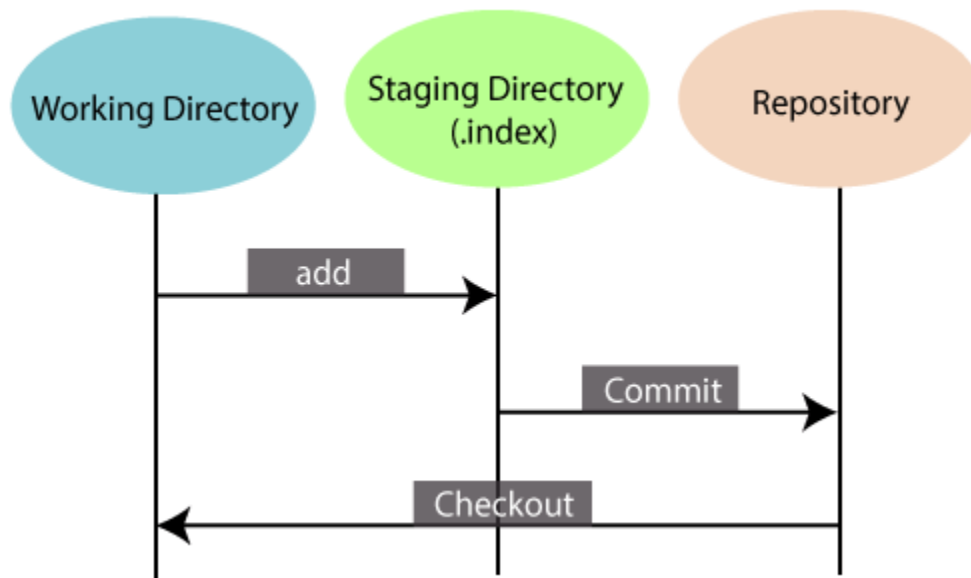
- **Data Assurance**

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.

- **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.**

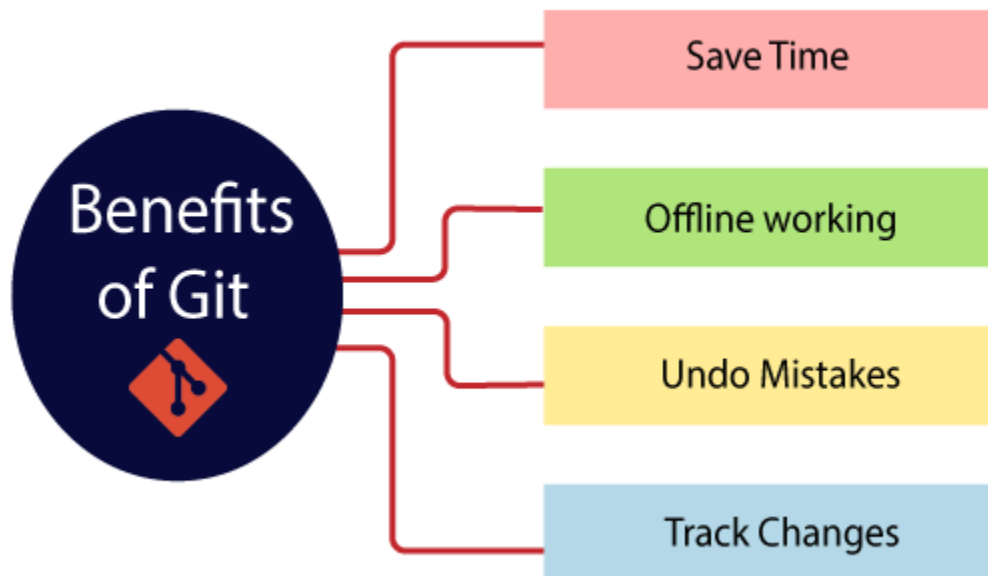
- **Maintain the clean history**

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.

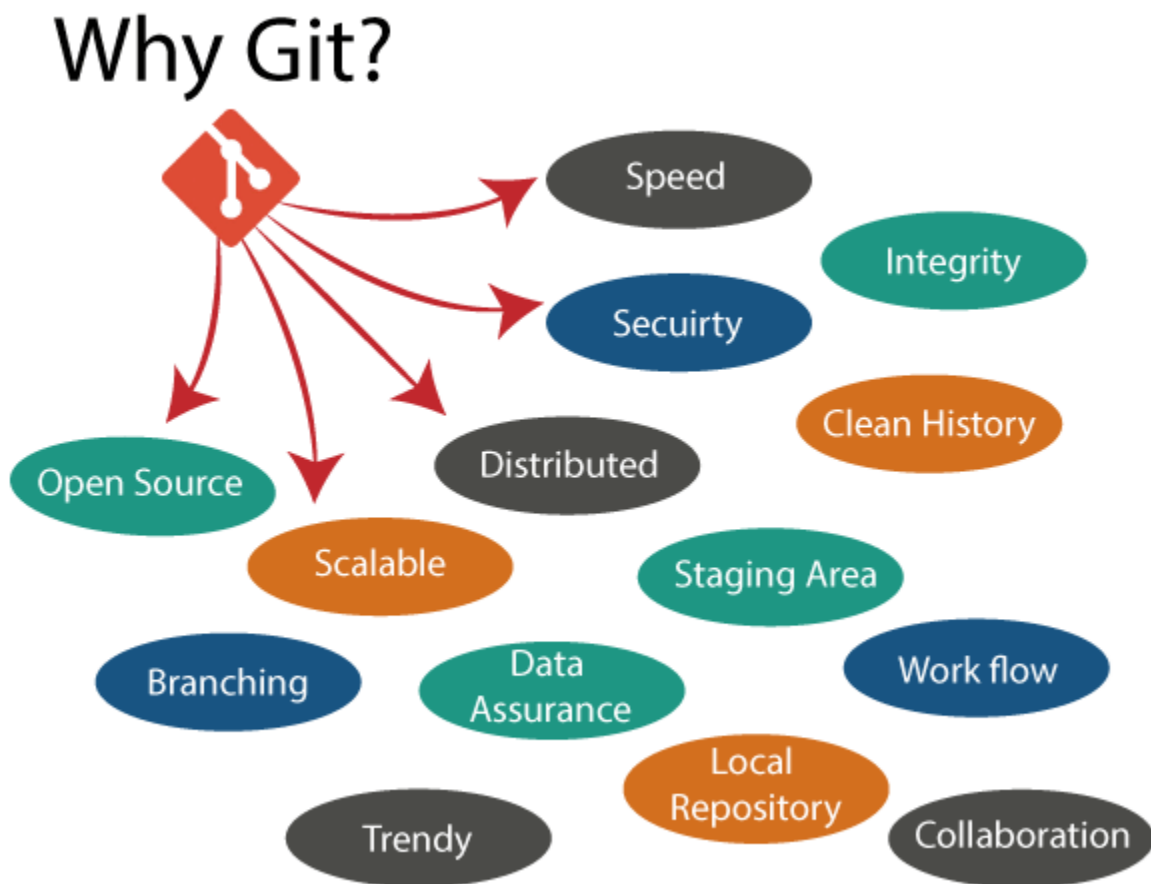
Some **significant benefits** of using Git are as follows:



- **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.
- **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.
- **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.
- **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.



- **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.

- **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.

- **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.

- **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.

- **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.

Prerequisites

Git is not a programming language, so you should have the basic understanding of Windows commands only.

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.



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

Benefits of GitHub

Difference between git and gitHub

Programming language wordings are very intuitive these days. By hearing the name of a particular language, we start imagining what all it will be.

Java and **Javascript** are very similar to the names ham and hamster, the logo of **python** is intertwined with the image of snakes.



So, someone looking at git and github would find any apparent connection between them. Let us see git and github in detail with the differences between them.

Git



There are many words to define **git**, but it is an open-source distributed version control system in simpler words.

Let us break each component in the definition and understand it.

- **Open-source** - A type of computer software released under a specific license. The users are given permissions to use the code, modify the code, give suggestions, clone the code to add new functionality. In other words, if the software is open-source, it is developed collaboratively in a public manner. The open-source softwares is cheaper, more flexible, and lasts longer than an authority or a company. The products in the source code include code, documents, formats for the users to understand and contribute to it. Using open-source a project can be expanded to update or revise the current features. Unix and Linux are examples of open-source softwares.
- **Control system** - The work of a control system is to track the content. In other words, git is used to storing the content to provide the services and features to the user.
- **Version Control system** - Just like an app has different updates due to bugs and additional feature addition, version changes, git also supports this feature. Many developers can add their code in parallel. So the version control system easily manages all the updates that are done previously. Git provides the feature of branching in which the updated code can be done,

and then it can be merged with the main branch to make it available to the users. It not only makes everything organized but keeps synchronization among the developers to avoid any mishap. Other examples of version control systems are Helix core, Microsoft TFS, etc.

- **Distributed version control system** - Here distributed version control system means if a developer contributes to open source, the code will also be available in his remote repository. The developer changes his local repository and then creates a pull request to merge his changes in the central repository. Hence, the word distributed means the code is stored in the central server and stored in every developer's remote system.

Why is git needed?

When a team works on real-life projects, git helps ensure no code conflicts between the developers. Furthermore, the project requirements change often. So a git manages all the versions. If needed, we can also go back to the original code. The concept of branching allows several projects to run in the same codebase.

GitHub



By the name, we can visualize that it is a Hub, projects, communities, etc. **GitHub** is a **Git repository** hosting service that provides a web-based graphical interface. It is the largest community in the world. Whenever a project is open-source, that particular repository gains exposure to the public and invites several people to contribute.

The source code of several projects is available on github which developers can use in any means.

Using github, many developers can work on a single project remotely because it facilitates collaboration.

Features of gitHub

- Using github the project managers can collaborate, review and guide the developers regarding any changes. This makes project management easy.
- The github repositories can be made public or private. Thus allowing safety to an organization in case of a project.
- GitHub has a feature of pull requests and issues in which all the developers can stay on the same page and organize.
- All the codes and their documentation are in one place in the same repository. Hence it makes easy code hosting.
- There are some special tools that github uses to identify the vulnerabilities in the code which other softwares do not have. Hence there is safety among the developers from code start till launch.
- Github is available for mobile and desktops. The UI is so user-friendly that it becomes straightforward to get comfortable with and use it.

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.

Git Version Control System

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.

Play Video

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

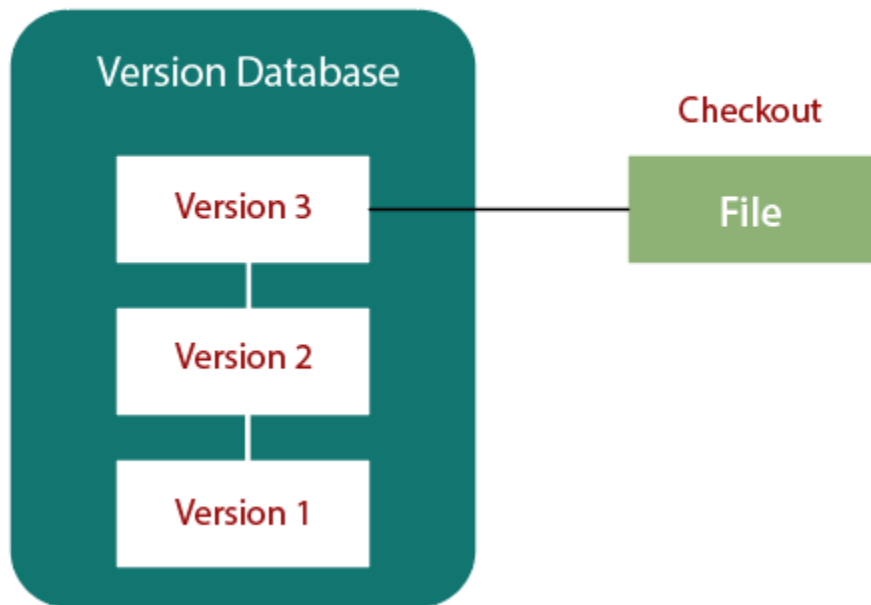
- Complete change history of the file
- Simultaneously working
- Branching and merging
- Traceability

Types of Version Control System

- Localized version Control System
- Centralized version control systems
- Distributed version control systems

Localized Version Control Systems

Local Computer



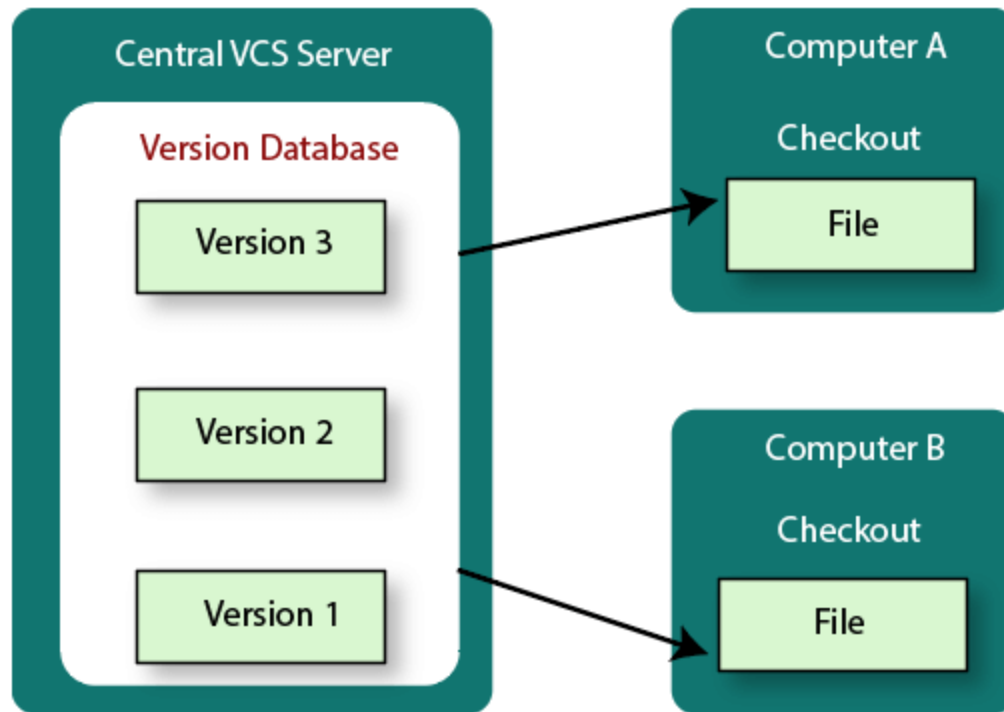
The localized version control method is a common approach because of its simplicity. But this approach leads to a higher chance of error. In this approach, you may forget which directory you're in and accidentally write to the wrong file or copy over files you don't want to.

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

Centralized Version Control System

The developers needed to collaborate with other developers on other systems. The localized version control system failed in this case. To deal with this problem, Centralized Version Control Systems were developed.



These systems have a single server that contains the versioned files, and some clients to check out files from a central place.

Centralized version control systems have many benefits, especially over local VCSs.

- Everyone on the system has information about the work what others are doing on the project.
- Administrators have control over other developers.
- It is easier to deal with a centralized version control system than a localized version control system.
- A local version control system facilitates with a server software component which stores and manages the different versions of the files.

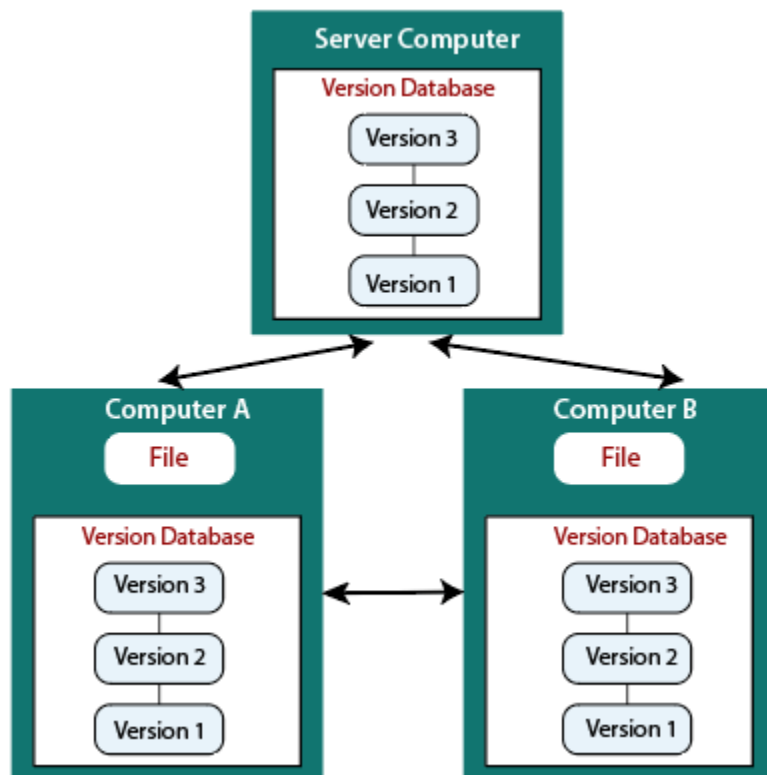
It also has the same drawback as in local version control system that it also has a single point of failure.

Distributed Version Control System

Centralized Version Control System uses a central server to store all the database and team collaboration. But due to single point failure, which means the failure of the central

server, developers do not prefer it. Next, the Distributed Version Control System is developed.

In a Distributed Version Control System (such as Git, Mercurial, Bazaar or Darcs), the user has a local copy of a repository. So, the clients don't just check out the latest snapshot of the files even they can fully mirror the repository. The local repository contains all the files and metadata present in the main repository.



DVCS allows automatic management branching and merging. It speeds up most operations except pushing and pulling. DVCS enhances the ability to work offline and does not rely on a single location for backups. If any server stops and other systems were collaborating via it, then any of the client repositories could be restored by that server. Every checkout is a full backup of all the data.

These systems do not necessarily depend on a central server to store all the versions of a project file.

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 SVN (Subversion) and CVS .	The popular tools of DVCS are Git and Mercurial .
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

How to Install Git on Windows

To use Git, you have to install it on your computer. Even if you have already installed Git, it's probably a good idea to upgrade it to the latest version. You can either install it as a package or via another installer or download it from its official site.

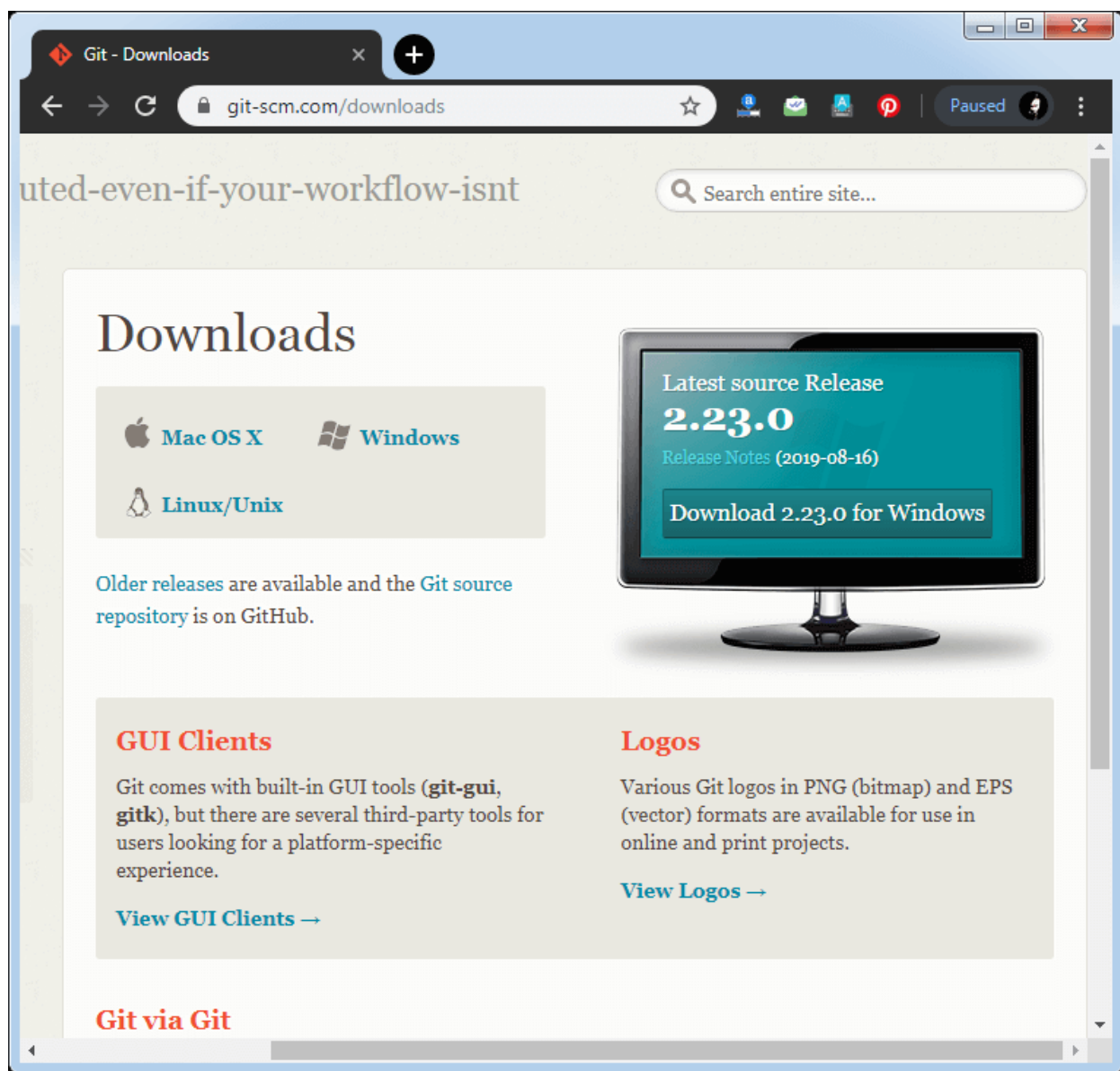
Now the question arises that how to download the Git installer package. Below is the stepwise installation process that helps you to download and install the Git.

How to download Git?

Step1

To download the Git installer, visit the Git's official site and go to download page. The link for the download page is <https://git-scm.com/downloads>. The page looks like as

Play Video



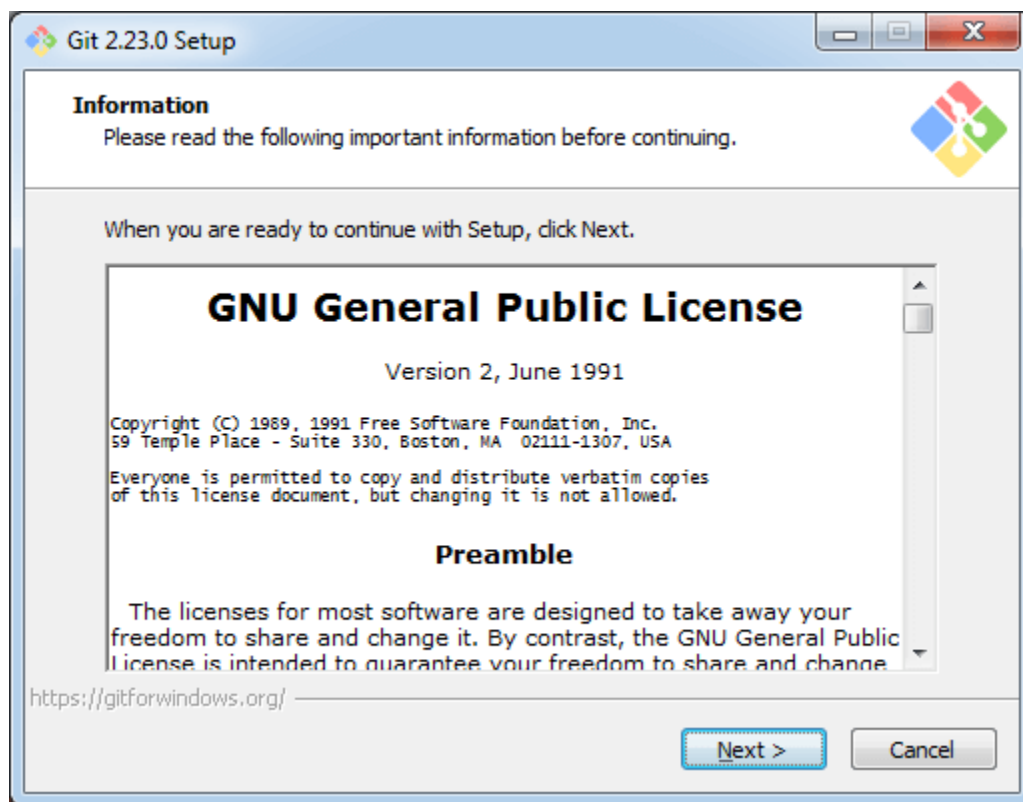
Click on the package given on the page as **download 2.23.0 for windows**. The download will start after selecting the package.

Now, the Git installer package has been downloaded.

Install Git

Step2

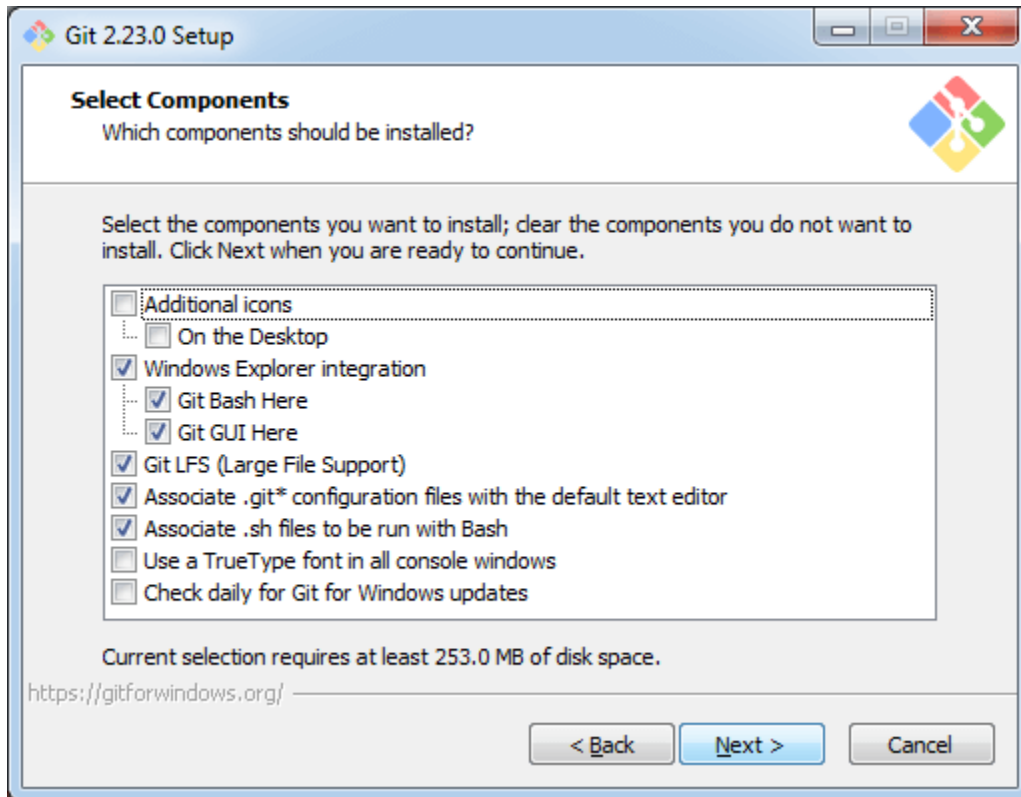
Click on the downloaded installer file and select **yes** to continue. After the selecting **yes** the installation begins, and the screen will look like as



Click on **next** to continue.

Step3

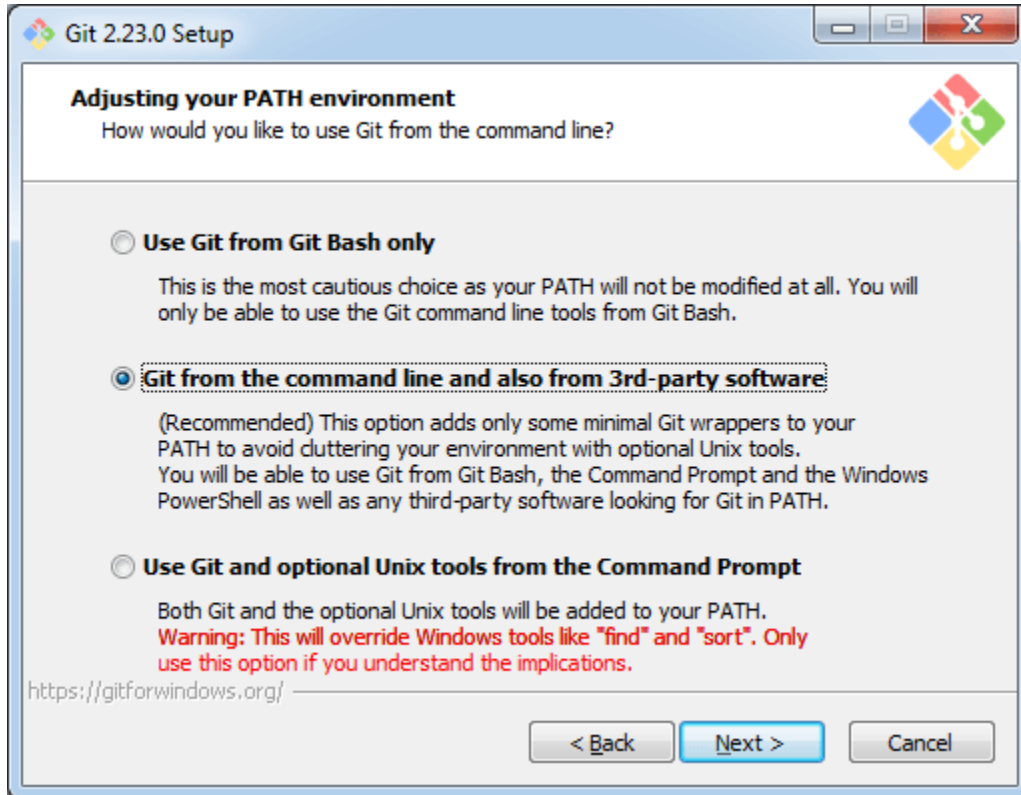
Default components are automatically selected in this step. You can also choose your required part.



Click next to continue.

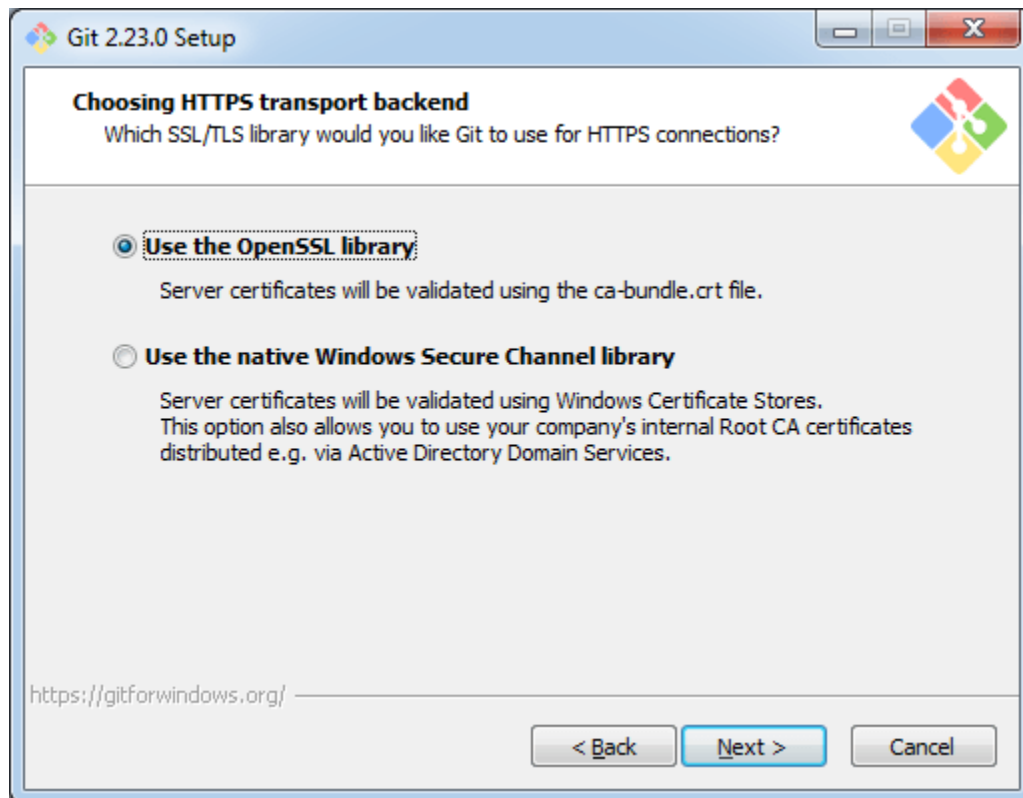
Step4

The default Git command-line options are selected automatically. You can choose your preferred choice. Click **next** to continue.



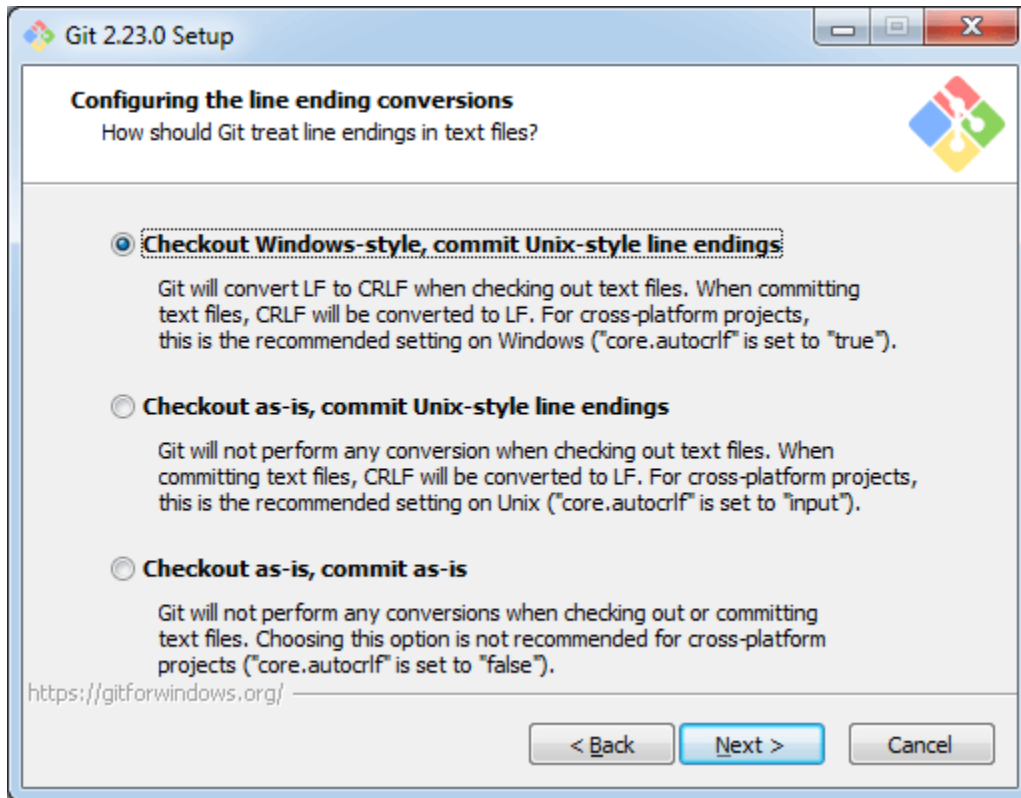
Step5

The default transport backend options are selected in this step. Click **next** to continue.



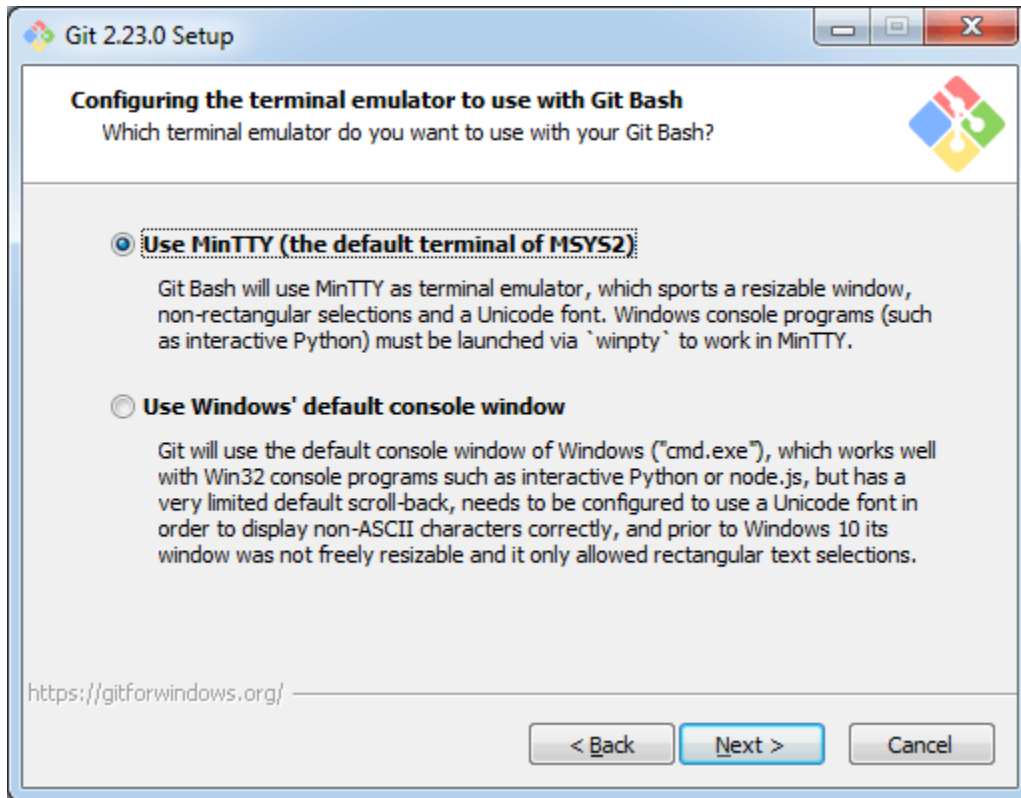
Step6

Select your required line ending option and click next to continue.



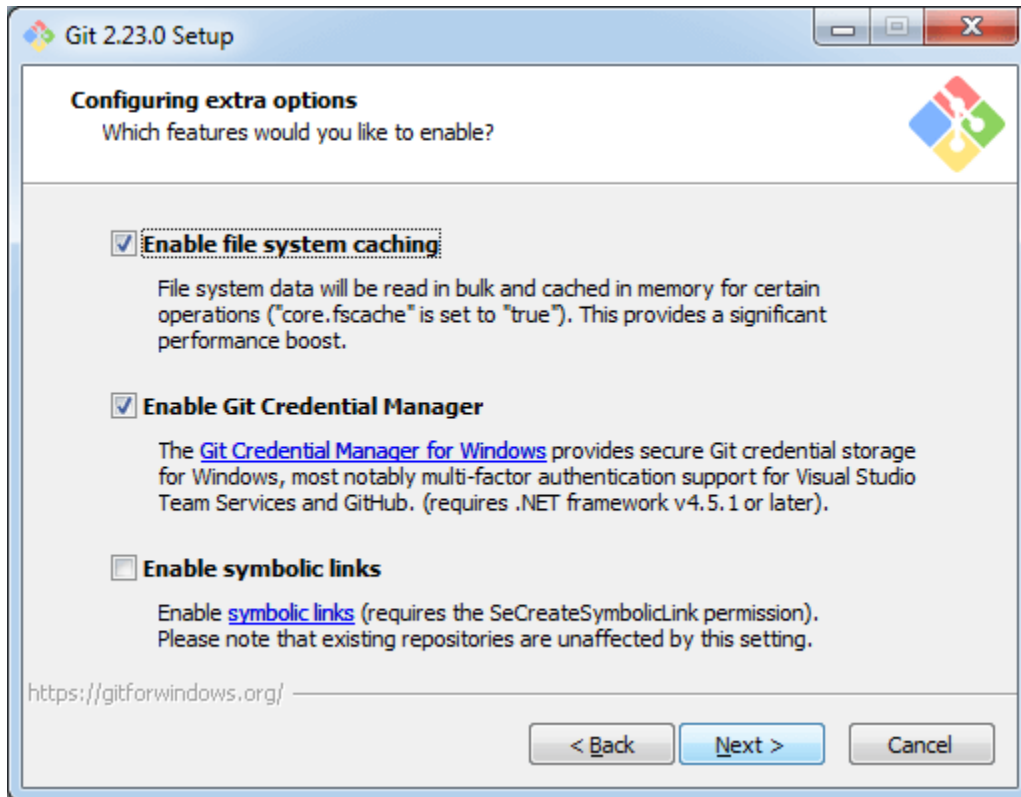
Step7

Select preferred terminal emulator clicks on the **next** to continue.



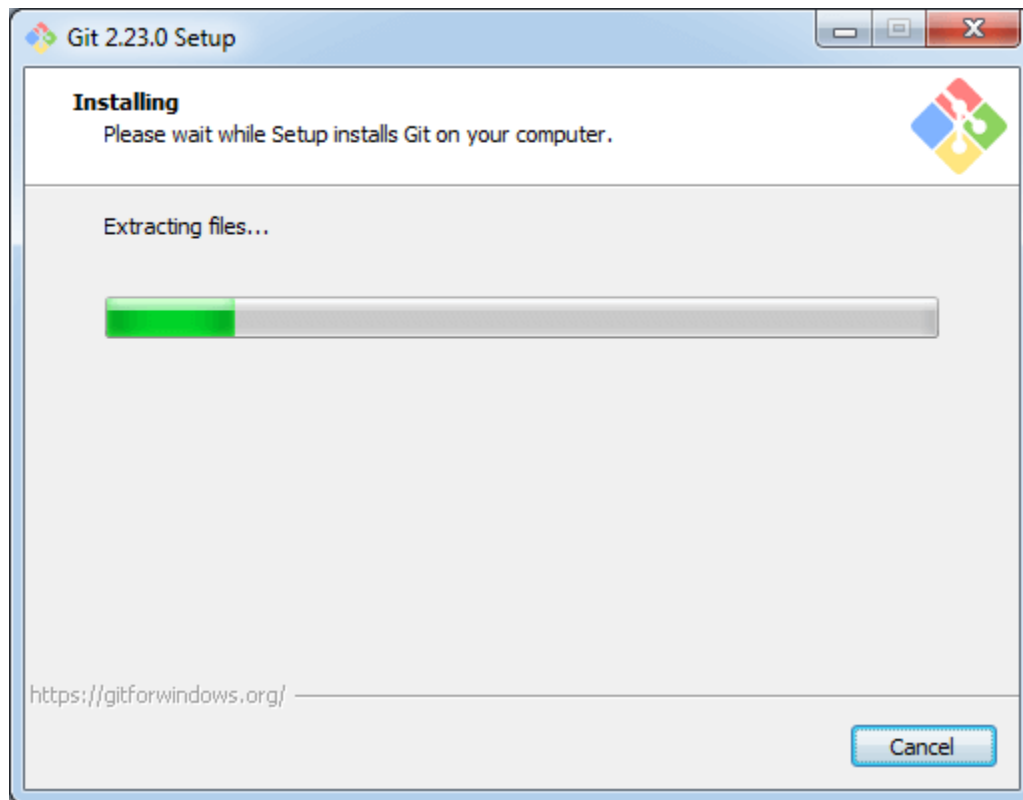
Step8

This is the last step that provides some extra features like system caching, credential management and symbolic link. Select the required features and click on the **next** option.



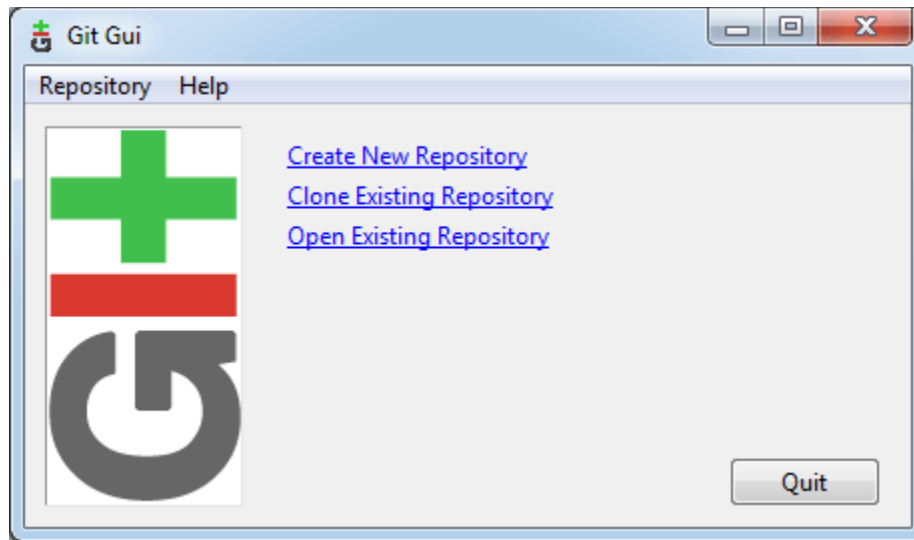
Step9

The files are being extracted in this step.



Therefore, The Git installation is completed. Now you can access the **Git Gui** and **Git Bash**.

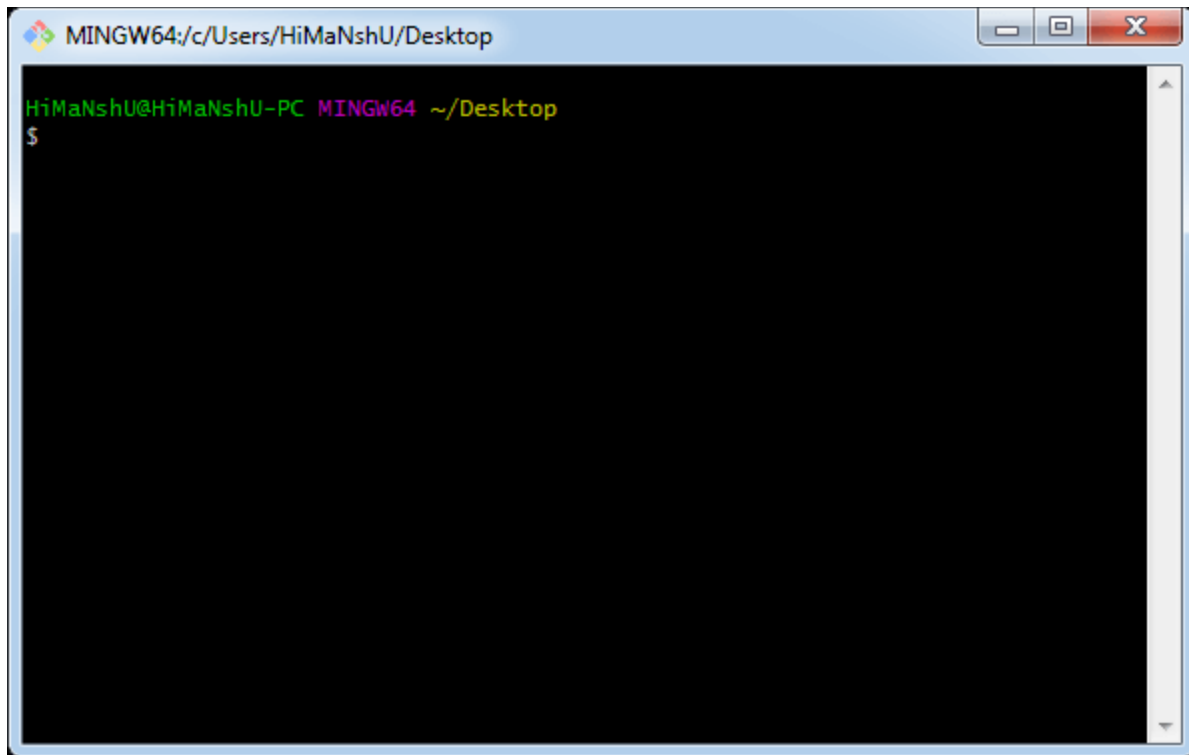
The **Git Gui** looks like as



It facilitates with three features.

- Create New Repository
- Clone Existing Repository
- Open Existing Repository

The **Git Bash** looks like as



Git Environment Setup

The environment of any tool consists of elements that support execution with software, hardware, and network configured. It includes operating system settings, hardware configuration, software configuration, test terminals, and other support to perform the operations. It is an essential aspect of any software.

It will help you to understand how to set up Git for first use on various platforms so you can read and write code in no time.

The Git config command

Git supports a command called **git config** that lets you get and set configuration variables that control all facets of how Git looks and operates. It is used to set Git configuration values on a global or local project level.

Setting **user.name** and **user.email** are the necessary configuration options as your name and email will show up in your commit messages.

Setting username

The username is used by the Git for each commit.

1. `$ git config --global user.name "Himanshu Dubey"`

Setting email id

The Git uses this email id for each commit.

1. `$ git config --global user.email "himanshudubey481@gmail.com"`

There are many other configuration options that the user can set.

Setting editor

You can set the default text editor when Git needs you to type in a message. If you have not selected any of the editors, Git will use your default system's editor.

To select a different text editor, such as Vim,

1. `$ git config --global core.editor Vim`

Checking Your Settings

You can check your configuration settings; you can use the **git config --list** command to list all the settings that Git can find at that point.

1. `$ git config -list`

This command will list all your settings. See the below command line output.

Output

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop
$ git config --list
core.symlinks=false
core.autocrlf=true
core.fscache=true
color.diff=auto
color.status=auto
color.branch=auto
color.interactive=true
help.format=html
rebase.autosquash=true
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
http.sslbackend=openssl
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge --skip -- %f
filter.lfs.process=git-lfs filter-process --skip
filter.lfs.required=true
credential.helper=manager
gui.recentrepo=C:/Git
user.email=dav.himanshudubey481@gmail.com
user.name=Himanshu Dubey
```

Colored output

You can customize your Git output to view a personalized color theme. The **git config** can be used to set these color themes.

Color.ui

1. `$ Git config -global color.ui true`

The default value of color.ui is set as auto, which will apply colors to the immediate terminal output stream. You can set the color value as true, false, auto, and always.

Git configuration levels

The git config command can accept arguments to specify the configuration level. The following configuration levels are available in the Git config.

- local
- global
- system

--local

It is the default level in Git. Git config will write to a local level if no configuration option is given. Local configuration values are stored in **.git/config** directory as a file.

--global

The global level configuration is user-specific configuration. User-specific means, it is applied to an individual operating system user. Global configuration values are stored in a user's home directory. **~/.gitconfig** on UNIX systems and **C:\Users\%.gitconfig** on windows as a file format.

--system

The system-level configuration is applied across an entire system. The entire system means all users on an operating system and all repositories. The system-level configuration file stores in a **gitconfig** file off the system directory. **\$(prefix)/etc/gitconfig** on UNIX systems and **C:\ProgramData\Git\config** on Windows.

The order of priority of the Git config is local, global, and system, respectively. It means when looking for a configuration value, Git will start at the local level and bubble up to the system level.

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.

Git Package Tools

Git provides powerful functionality to explore it. We need many tools such as commands, command line, Git GUI. Let's understand some essential package tools.

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

Bash is a standard default shell on Linux and macOS. A shell is a terminal application which is used to create an interface with an operating system through commands.

By default, Git Windows package contains the Git Bash tool. We can access it by right-click on a folder in Windows Explorer.

Git Bash Commands

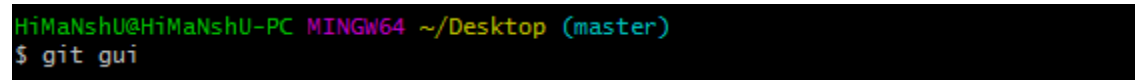
Git Bash comes with some additional commands that are stored in the **/usr/bin** directory of the Git Bash emulation. Git Bash can provide a robust shell experience on Windows. Git Bash comes with some essential shell commands like **Ssh, scp, cat, find**.

Git Bash also includes the full set of Git core commands like **git clone, git commit, git checkout, git push**, and more.

Git GUI

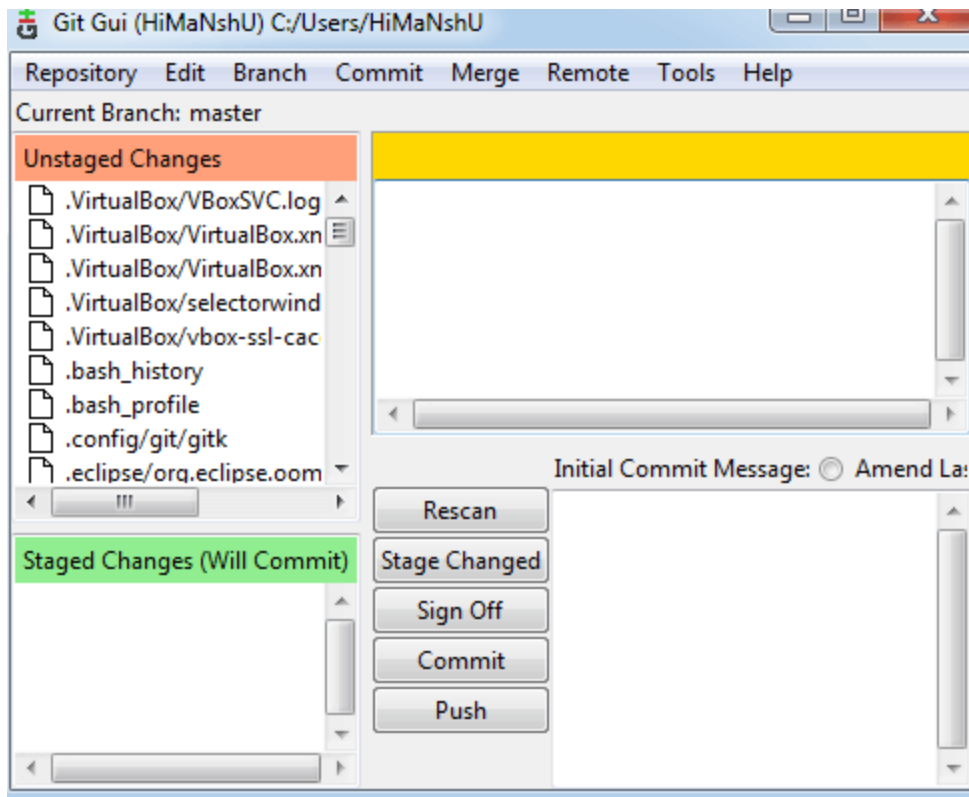
Git GUI is a powerful alternative to Git BASH. It offers a graphical version of the Git command line function, as well as comprehensive visual diff tools. We can access it by simply right click on a folder or location in windows explorer. Also, we can access it through the command line by typing below command.

1. `$ git gui`



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

A pop-up window will open as Git gui tool. The Git GUI's interface looks like as:



Git facilitates with some built-in GUI tools for committing (git-gui) and browsing (gitk), but there are many third-party tools for users looking for platform-specific experience.

Gitk

gitk is a graphical history viewer tool. It's a robust GUI shell over **git log** and **git grep**. This tool is used to find something that happened in the past or visualize your project's history.

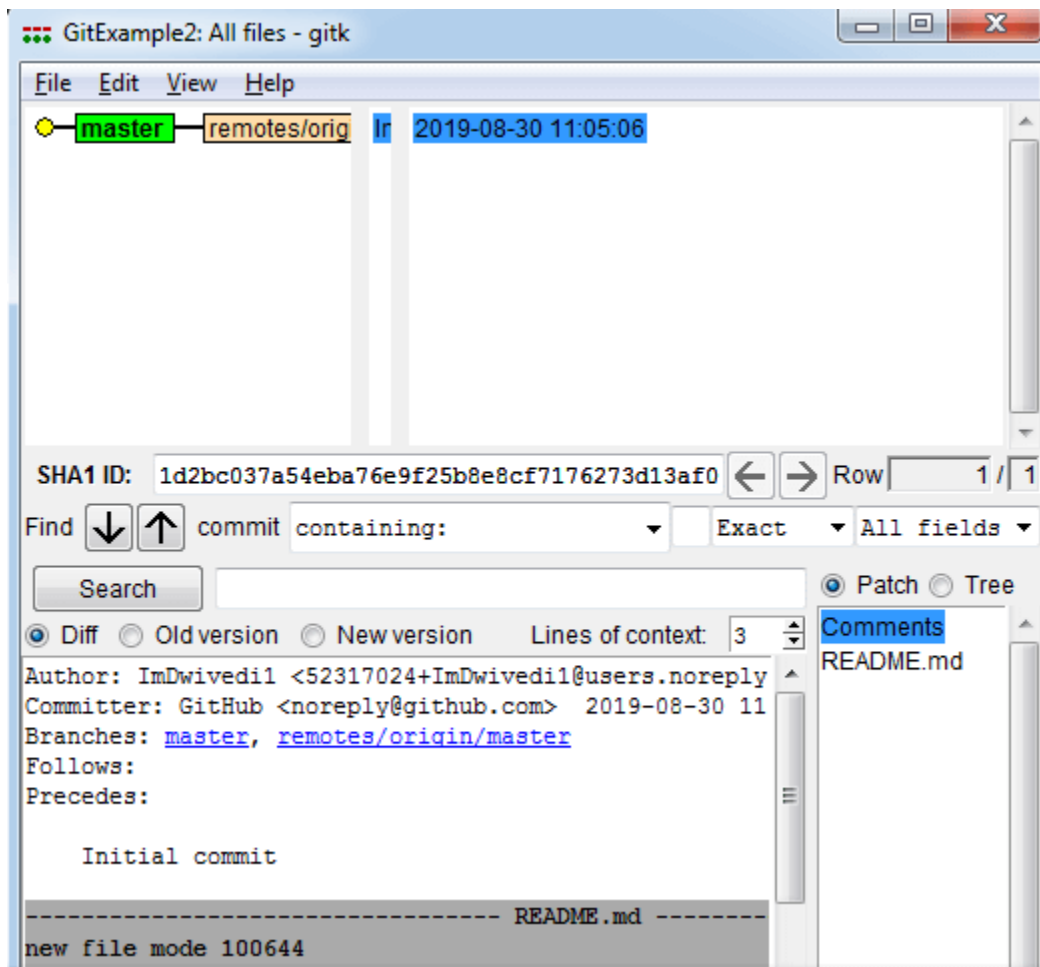
Gitk can invoke from the command-line. Just change directory into a Git repository, and type:

1. `$ gitk [git log options]`

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

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

This command invokes the gitk graphical interface and displays the project history. The Gitk interface looks like this:



Gitk supports several command-line options, most of which are passed through to the underlying git log action.

Git Third-Party Tools

Many third-party tools are available in the market to enhance the functionality of Git and provide an improved user interface. These tools are available for distinct platforms like Windows, Mac, Linux, Android, iOS.

A list of popular third party Git tools are as follows:

Tools	Platforms					Price	License Type
	Windows	Mac	Linux	Android	iOS		
SourceTree	Yes	Yes	No	No	No	Free	Proprietary
GitHub Desktop	Yes	Yes	No	No	No	Free	MIT
TortoiseGit	Yes	No	No	No	No	Free	GNU GPL
Git Extensions	Yes	Yes	Yes	No	No	Free	GNU GPL
GitKraken	Yes	Yes	Yes	No	No	Free/\$29/\$49	Proprietary
SmartGit	Yes	Yes	Yes	No	No	\$79/user/free for non-commercial use	Proprietary
Tower	Yes	Yes	No	No	No	\$79/user (30 days free trial)	Proprietary
Git Up	No	Yes	No	No	No	Free	GNU GPL
GitEye	Yes	Yes	Yes	No	No	Free	Proprietary
gitg	Yes	No	Yes	No	No	Free	GNUGPL
Git2Go	No	No	No	No	Yes	Free with in-app purchases	Proprietary

GitDrive	No	No	No	No	Yes	Free with in-app purchases	Proprietary
GitFinder	No	Yes	No	No	No	\$24.95	Proprietary
SnailGit	No	Yes	No	No	No	&9.99/Lite version	Proprietary
Pocket Git	No	No	No	Yes	No	1.99€	Proprietary
Sublime Merge	Yes	Yes	Yes	No	No	\$99/user, \$75 annual business sub, free eval	Proprietary

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

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.

Checkout

In Git, the term checkout is used for the act of switching between different versions of a target entity. The **git checkout** command is used to switch between branches in a repository.

Cherry-Picking

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

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

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.

HEAD

HEAD is the representation of the last commit in the current checkout branch. We can think of the head like a current branch. When you switch branches with git checkout, the HEAD revision changes, and points the new branch.

Index

The Git index is a staging area between the working directory and repository. It is used as the index to build up a set of changes that you want to commit together.

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

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

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

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

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.

Rebase

In Git, the term rebase is referred to as the process of moving or combining a sequence of commits to a new base commit. Rebasing is very beneficial and visualized the process in the environment of a feature branching workflow.

From a content perception, rebasing is a technique of changing the base of your branch from one commit to another.

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

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.

Stashing

Sometimes you want to switch the branches, but you are working on an incomplete part of your current project. You don't want to make a commit of half-done work. Git stashing allows you to do so. The **git stash command** enables you to switch branch without committing the current branch.

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

Upstream And Downstream

The term upstream and downstream is a reference of the repository. Generally, upstream is where you cloned the repository from (the origin) and downstream is any project that integrates your work with other works. However, these terms are not restricted to Git repositories.

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 Reset

In Git, the term reset stands for undoing changes. The **git reset** command is used to reset the changes. The git reset command has three core forms of invocation. These forms are as follows.

- Soft
- Mixed
- Hard

Git Ignore

In Git, the term ignore used to specify intentionally untracked files that Git should ignore. It doesn't affect the Files that already tracked by Git.

Git Diff

Git diff is a command-line utility. It's a multiuse Git command. When it is executed, it runs a diff function on Git data sources. These data sources can be files, branches, commits, and more. It is used to show changes between commits, commit, and working tree, etc.

Git Cheat Sheet

A Git cheat sheet is a summary of Git quick references. It contains basic Git commands with quick installation. A cheat sheet or crib sheet is a brief set of notes used for quick reference. Cheat sheets are so named because the people may use it without no prior knowledge.

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 Squash

In Git, the term squash is used to squash previous commits into one. Git squash is an excellent technique to group-specific changes before forwarding them to others. You can merge several commits into a single commit with the powerful interactive rebase command.

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

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.

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
2. Git init command
3. Git clone command
4. Git add command
5. Git commit command
6. Git status command
7. Git push Command
8. Git pull command
9. Git Branch Command
10. Git Merge Command
11. Git log command
12. Git remote 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.

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop
$ git init Demo
Initialized empty Git repository in C:/Users/HiMaNshU/Desktop/Demo/.git/

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop
$ |
```

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

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/Git-example (master)
$ git clone https://github.com/ImDwivedi1/Git-Example.git
Cloning into 'Git-Example'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
```

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*

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/Git-example (master)
$ git add README.md
```

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`

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/Git-example (master)
$ git commit -a -m "Adding the key of c"
[master (root-commit) 758797a] Adding the key of c
1 file changed, 2 insertions(+)
create mode 100644 README.md
```

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`

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/Git-example (master)
$ git status
On branch master
Your branch is based on 'origin/master', but the upstream is gone.
    (use "git branch --unset-upstream" to fixup)

nothing to commit, working tree clean
```

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.

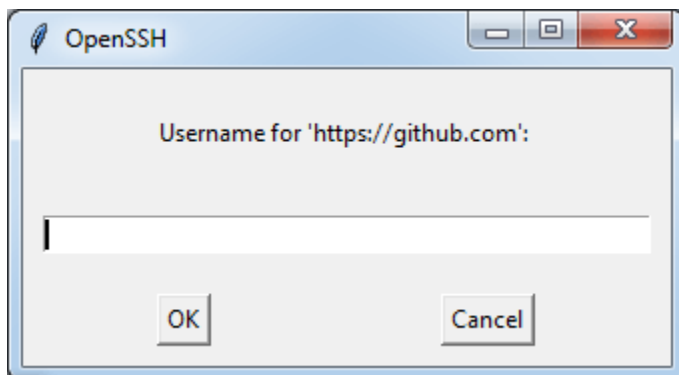
Git push origin master

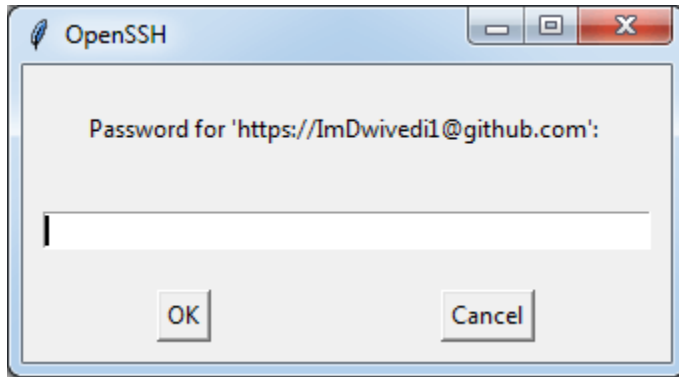
This command sends the changes made on the master branch, to your remote repository.

Syntax

1. `$ git push [variable name] master`

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/Git-example (master)
$ git push origin master
```





```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/gitexample2 (master)
$ git push origin master
Everything up-to-date

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/gitexample2 (master)
$
```

Git push -all

This command pushes all the branches to the server repository.

Syntax

1. `$ git push --all`

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/gitexample2 (master)
$ git push --all
Everything up-to-date

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/gitexample2 (master)
$
```

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`


```

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/Git-example (master)
$ git pull https://github.com/ImDwivedi1/Git-Example
warning: no common commits
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/ImDwivedi1/Git-Example
 * branch            HEAD       -> FETCH_HEAD
fatal: refusing to merge unrelated histories

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

```

9) Git Branch Command

This command lists all the branches available in the repository.

Syntax

1. \$ git branch

```

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/gitexample2 (master)
$ git branch
* master

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/gitexample2 (master)
$

```

10) Git Merge Command

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

Syntax

1. \$ git merge BranchName

```

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/gitexample2 (master)
$ git merge master
Already up to date.

```

11) Git log Command

This command is used to check the commit history.

Syntax

1. \$ git log

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/gitexample2 (master)
$ git log
commit 1d2bc037a54eba76e9f25b8e8cf7176273d13af0 (HEAD -> master, origin/master,
origin/HEAD)
Author: ImDwivedi1 <52317024+ImDwivedi1@users.noreply.github.com>
Date:   Fri Aug 30 11:05:06 2019 +0530

    Initial commit
```

By default, if no argument passed, Git log shows the most recent commits first. We can limit the number of log entries displayed by passing a number as an option, such as -3 to show only the last three entries.

1. \$ git log -3

12) Git remote Command

Git Remote command is used to connect your local repository to the remote server. This command allows you to create, view, and delete connections to other repositories. These connections are more like bookmarks rather than direct links into other repositories. This command doesn't provide real-time access to repositories.

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/gitexample2 (master)
$ git remote add origin https://github.com/ImDwivedi1/GitExample2
fatal: remote origin already exists.
```