Software Development Methodologies

Developing an application in a team :

Team

- An application developer is a critical part of technical and/or project management teams responsible for ensuring user needs are met through the deployment and updates of software.
- Application developers can be found in almost every industry sector, in any company interested in pushing new software and updates out to their end users on a routine basis.
- The application developer could be responsible for working with a team to deploy releases to internal or external clients.

Understanding the Application Deployment Lifecycle

The application deployment lifecycle typically involves

the following key stages: Initial Planning > Design > Development > Testing > Deployment > Support.

Roles and Responsibilities of Application Developers

Application developers have a number of responsibilities that fall on their shoulders. These have to do with managing the application lifecycle, knowledge of certain principles in coding, support and collaboration efforts.

Coding and Design

It stands to reason that a very basic tenant of application development is an in-depth knowledge of coding and application design principles. Depending on the specialization, application developers need to know the right programming language to code for the operating system they are designing for.

Typical programming languages include:

- Java / JavaScript
- C++
- Python; and
- PHP

Issues developers face when working in a team

Lack of Diverse Skills and Interests

If a team consists of members with similar skills and interests, the purpose of the team may not be achieved. Consider the scope of the project before selecting team members to determine which skills and interests best serve the team. Choose team members who have a range of different skill sets and interests.

Poor Communication

When individuals work as part of a team, communication during all phases of the project is a key component. If a team member treats his role as an independent one, communication suffers. If the team leader doesn't communicate with the team members, the project flow suffers. Hold team meetings to discuss the progress of the project. Ask team members about their successes and challenges in team work to gauge the level of communication.

Lack of Leadership

Teams need leaders to offer a sense of purpose and direction. Lack of effective leadership challenges effective team development. Without a strong leader to guide the team and hold members accountable, the team may lose morale and momentum. Commit yourself to monitoring the team's performance and offering support as needed. If you can't be present during team meetings, appoint a team leader. Introduce the leader and his role to the team to prevent competition for the role from other team members.

Role Confusion

Even though a team works together to achieve a goal, each person needs to know his specific role within the team, according to WP ERP. Otherwise, role confusion results. When team members lack an understanding of their specific roles or choose not to follow through with their roles, the team cannot develop as a cohesive and well-functioning unit. As a small business owner, develop specific, well-defined roles for each team member. Explain each role, in detail, to each team member to avoid confusion. Monitor team members to make sure they adhere to their assigned role.

Interpersonal Conflict

Conflict within a team is inevitable and one of the major problems in team work. How team members deal with conflict is critical to team development. Invest time in training for yourself and your employees in conflict resolution skills. With the skills to effectively resolve conflict, your team can work together to stay on track without your constant intervention.

Poor Work Environment

A team needs a specific place to meet. It needs to be quiet and conducive to effective teamwork. If the team is virtual, the members need to agree on a set of virtual tools – communication, file sharing, project and task monitoring – to help the team work together to accomplish goals. Not having a designated place or online management tools as part of team development can hinder progress.

Introduction to code versioning system

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.

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

Complete change history of the file

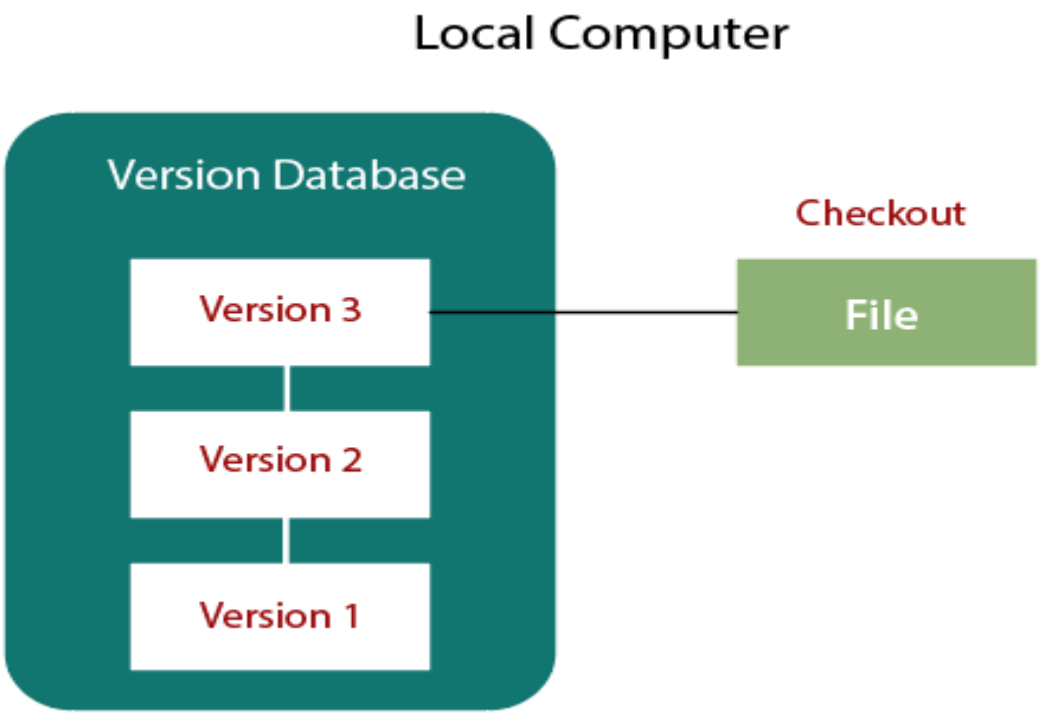
Simultaneously working

Branching and merging

Traceability

Types of Version Control System

Localized Version Control Systems



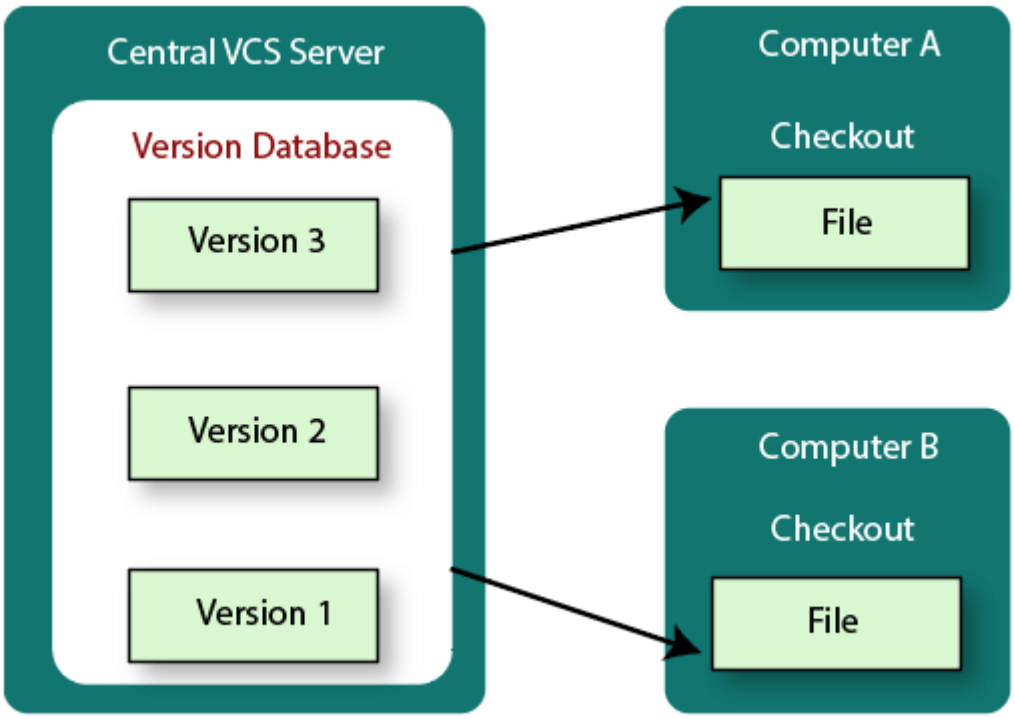
The localized version control method is a common approach because of its simplicity. But this approach leads to a higher chance of error. In this approach, you may forget which directory you're in and accidentally write to the wrong file or copy over files you don't want to.

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 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.

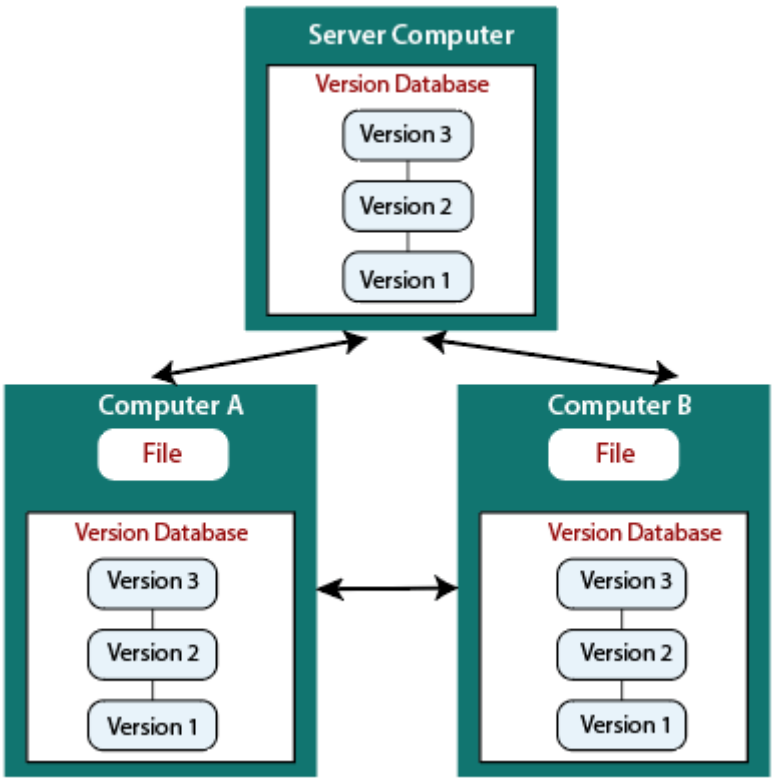
A local version control system facilitates with a server software component which stores and manages the different versions of the files.

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 of 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.

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

- **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 feature**s 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.

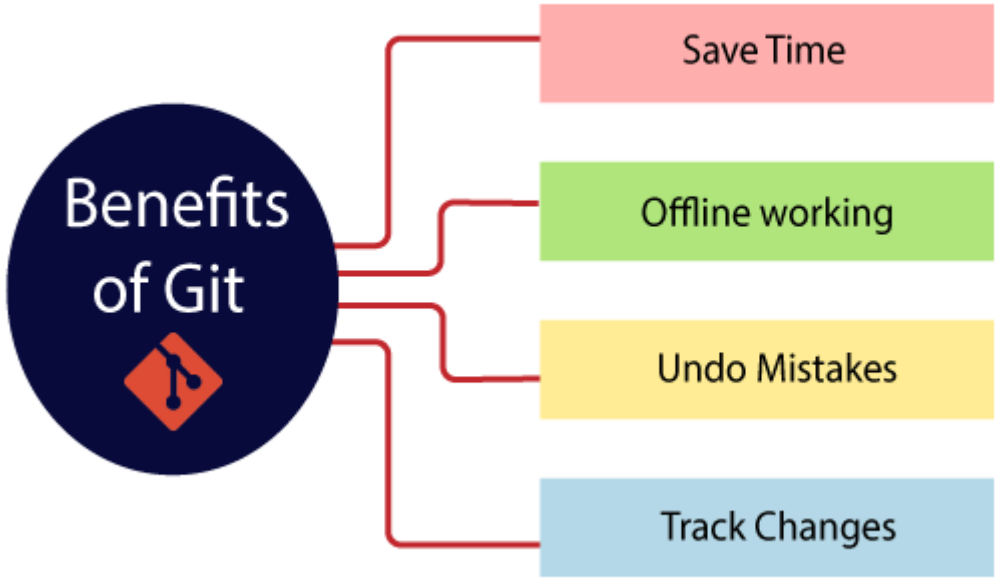
    - The core benefit of branching is if we want to push something to a remote repository, we do not have to push all of our branches. We can select a few of our branches, or all of them together.

- **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.



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.

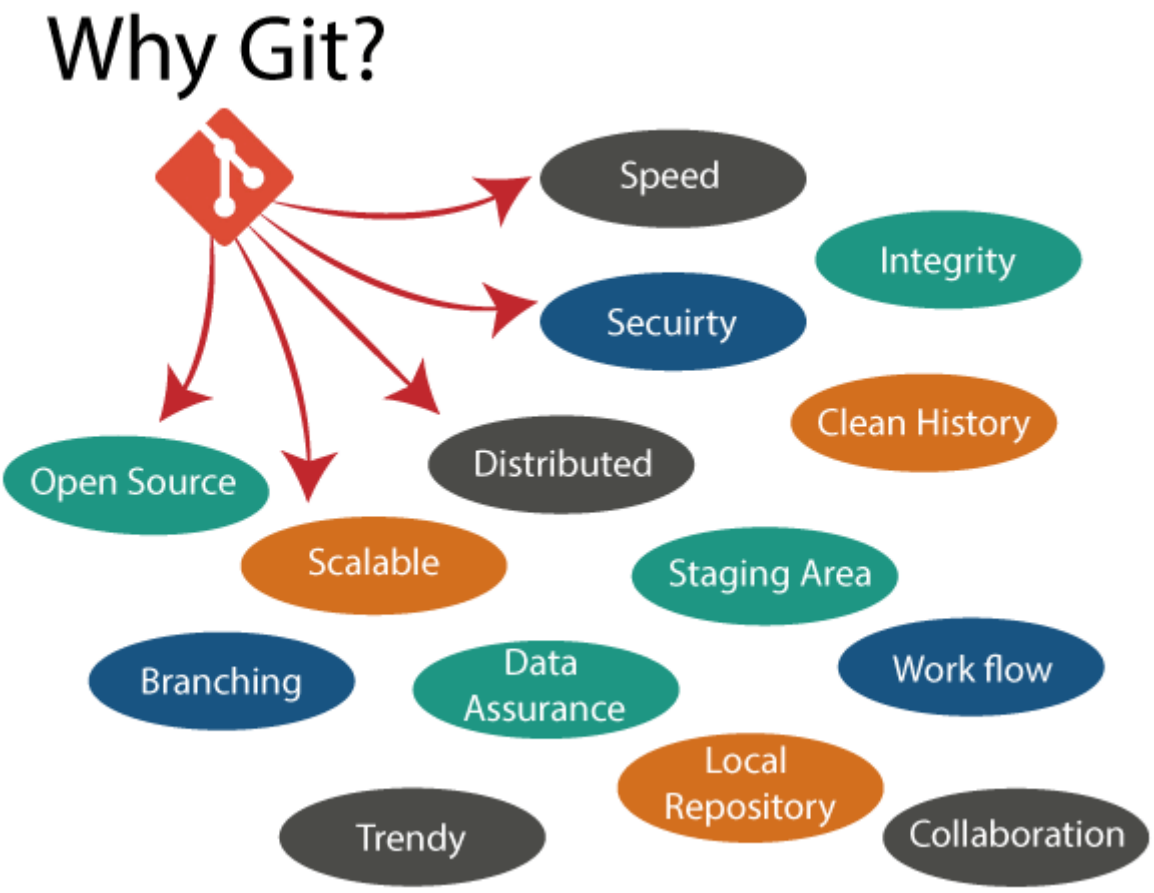
Undo Mistakes

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?



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.

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
- Wikisc

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.

How to Install Git on Windows

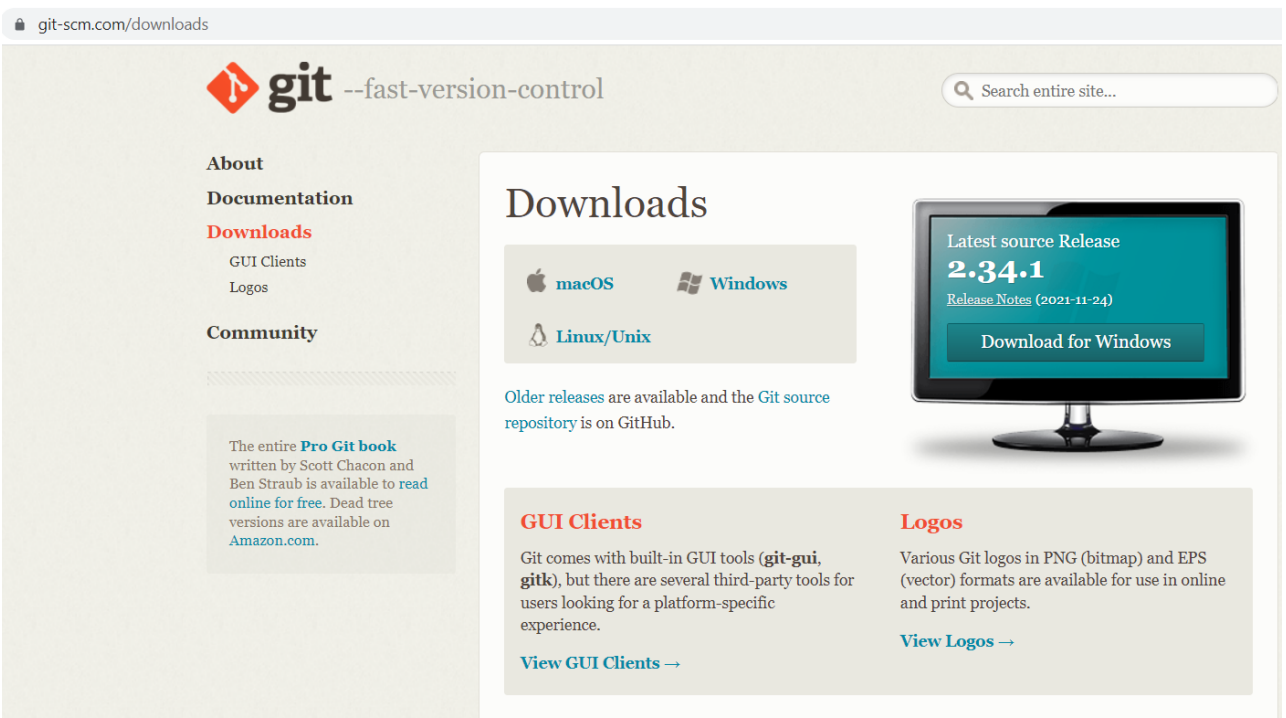
To use Git, you have to install it on your computer. Even if you have already installed Git, it's probably a good idea to upgrade it to the latest version. You can either install it as a package or via another installer or download it from its official site.

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



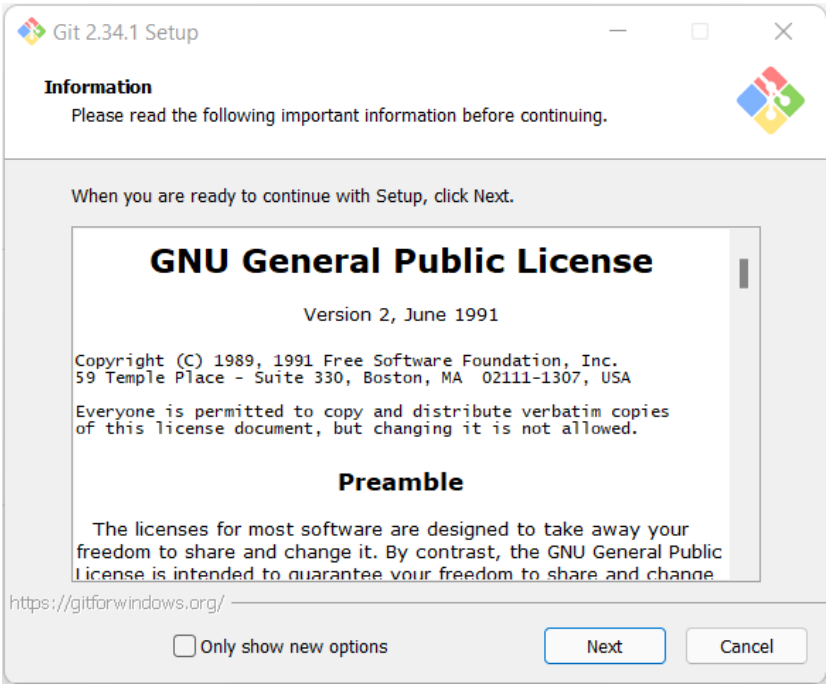
Click on the package given on the page as **download 2.34.1 for windows**. The download will start after selecting the package.

Now, the Git installer package has been downloaded.

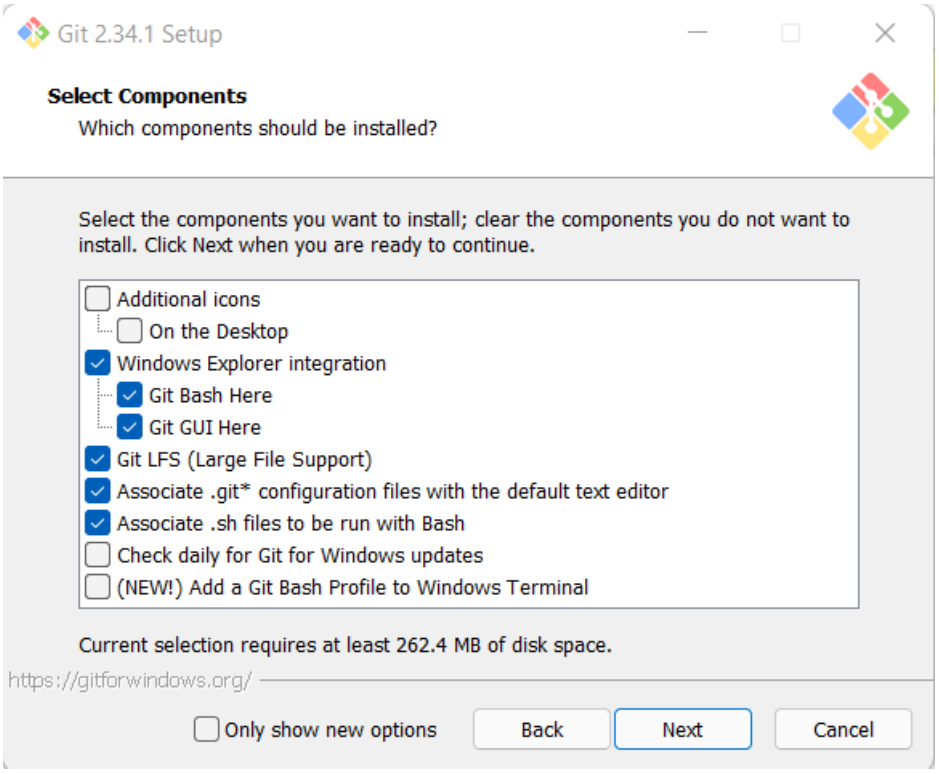
Install Git

Step2

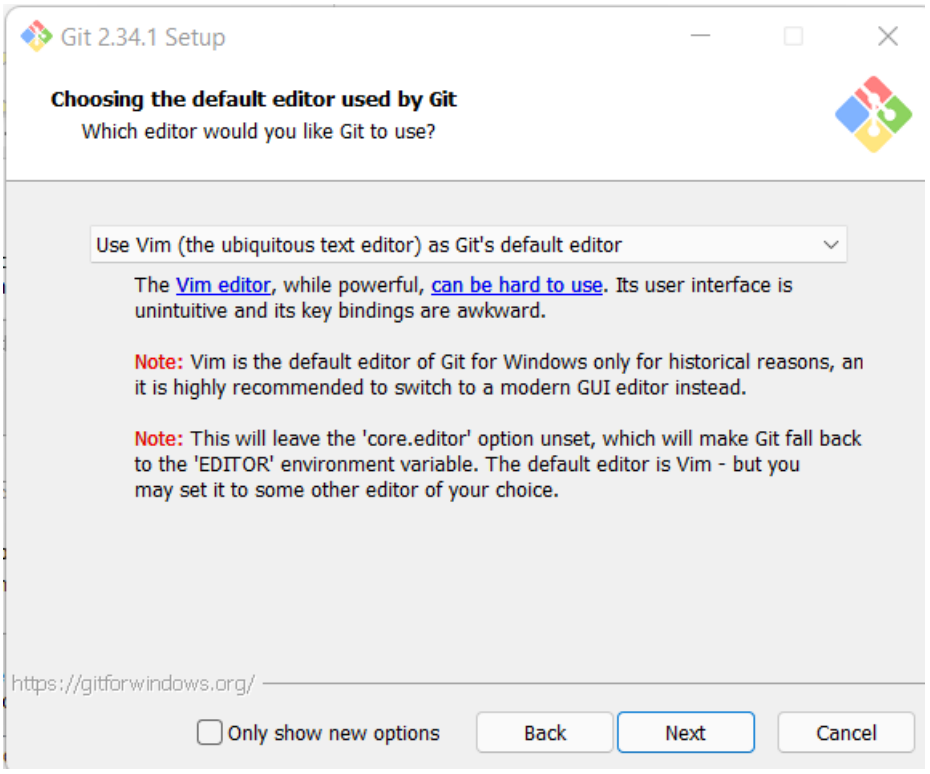
Click on the downloaded installer file and select yes to continue. After the selecting yes the installation begins, and the screen will look like as



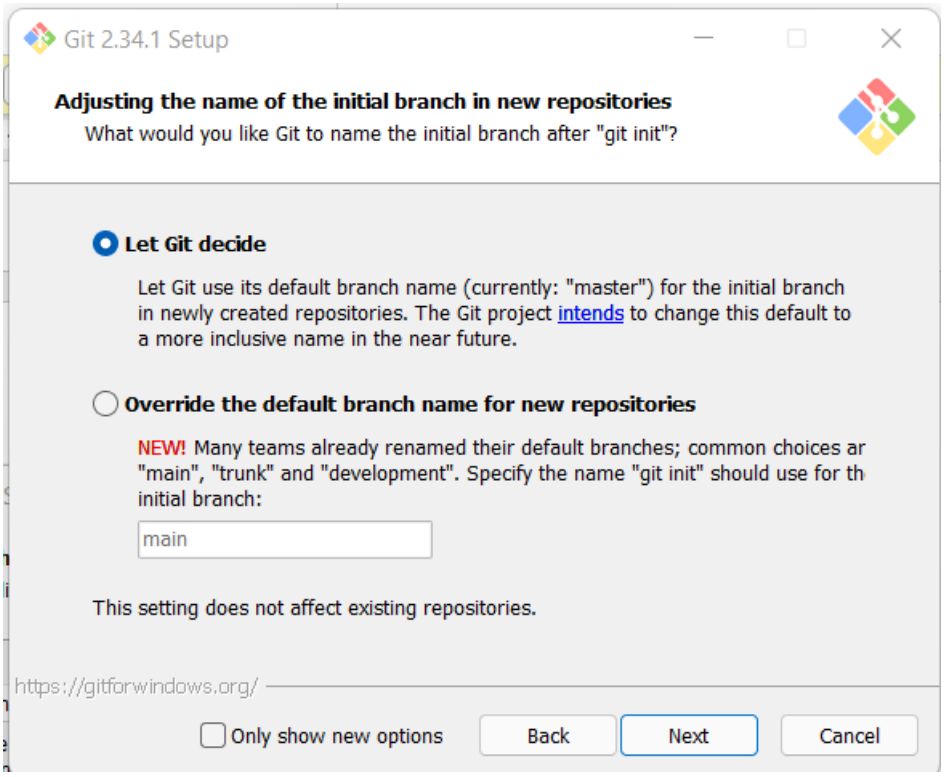
Next…..



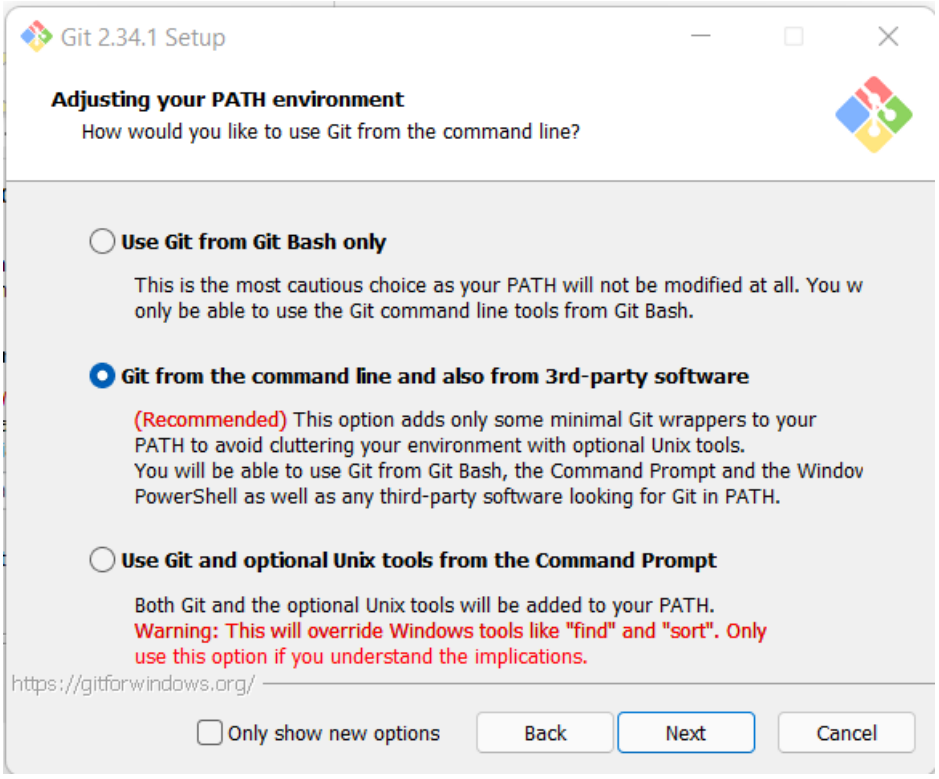
Next……


Git 2.34.1 Setup
Choosing the default editor used by Git
Which editor would you like Git to use?
Use Vim (the ubiquitous text editor) as Git's default editor
The Vim editor, while powerful, can be hard to use. Its user interface is unintuitive and its key bindings are awkward.
Note: Vim is the default editor of Git for Windows only for historical reasons, an it is highly recommended to switch to a modern GUI editor instead.
Note: This will leave the 'core.editor' option unset, which will make Git fall back to the 'EDITOR' environment variable. The default editor is Vim - but you may set it to some other editor of your choice.
https://gitforwindows.org/
Only show new options
Back
Next
Cancel

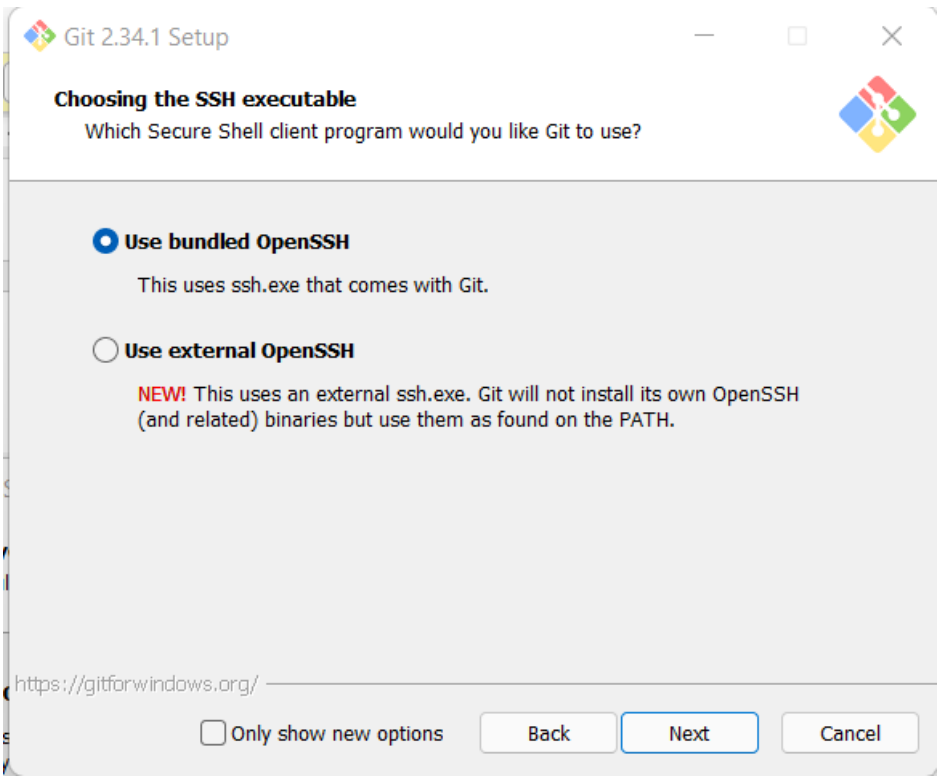
Next…..


Git 2.34.1 Setup
Adjusting the name of the initial branch in new repositories
What would you like Git to name the initial branch after "git init"?
Let Git decide
Let Git use its default branch name (currently: "master") for the initial branch in newly created repositories. The Git project intends to change this default to a more inclusive name in the near future.
Override the default branch name for new repositories
NEW! Many teams already renamed their default branches; common choices ar "main", "trunk" and "development". Specify the name "git init" should use for th initial branch:
main
This setting does not affect existing repositories.
https://gitforwindows.org/
Only show new options
Back
Next
Cancel

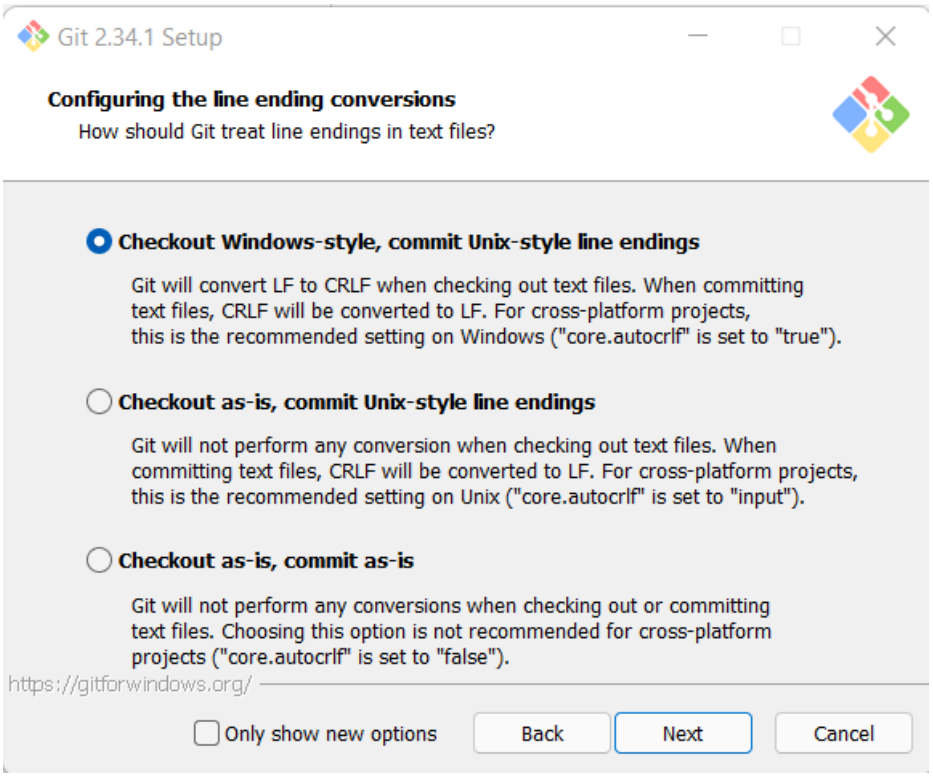
Next….


Git 2.34.1 Setup
Adjusting your PATH environment
How would you like to use Git from the command line?
Use Git from Git Bash only
This is the most cautious choice as your PATH will not be modified at all. You w
only be able to use the Git command line tools from Git Bash.
Git from the command line and also from 3rd-party software
(Recommended) This option adds only some minimal Git wrappers to your
PATH to avoid cluttering your environment with optional Unix tools.
You will be able to use Git from Git Bash, the Command Prompt and the Windov
PowerShell as well as any third-party software looking for Git in PATH.
Use Git and optional Unix tools from the Command Prompt
Both Git and the optional Unix tools will be added to your PATH.
Warning: This will override Windows tools like "find" and "sort". Only
use this option if you understand the implications.
https://gitforwindows.org/
Only show new options
Back
Next
Cancel

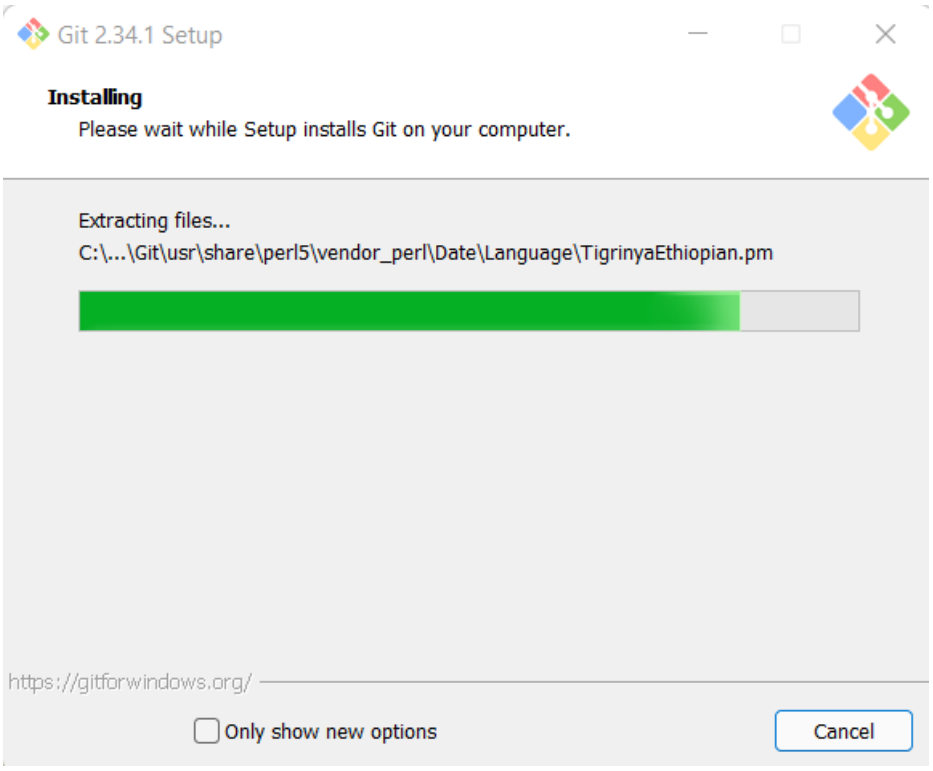
Next….


Git 2.34.1 Setup
Choosing the SSH executable
Which Secure Shell client program would you like Git to use?
Use bundled OpenSSH
This uses ssh.exe that comes with Git.
Use external OpenSSH
NEW! This uses an external ssh.exe. Git will not install its own OpenSSH
(and related) binaries but use them as found on the PATH.
https://gitforwindows.org/
Only show new options
Back
Next
Cancel

Next….



Next….



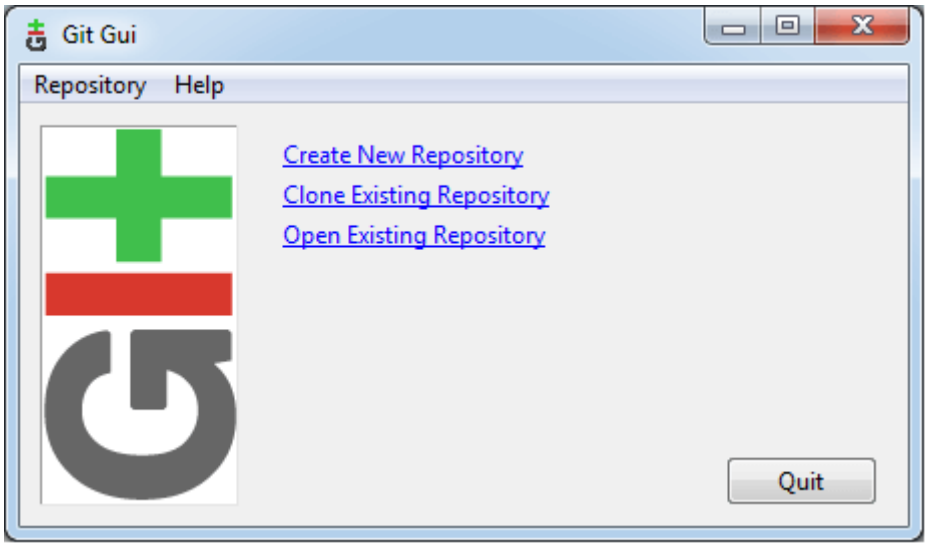
Then Finish

Open cmd in Admin mode and type

>> Git –version

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

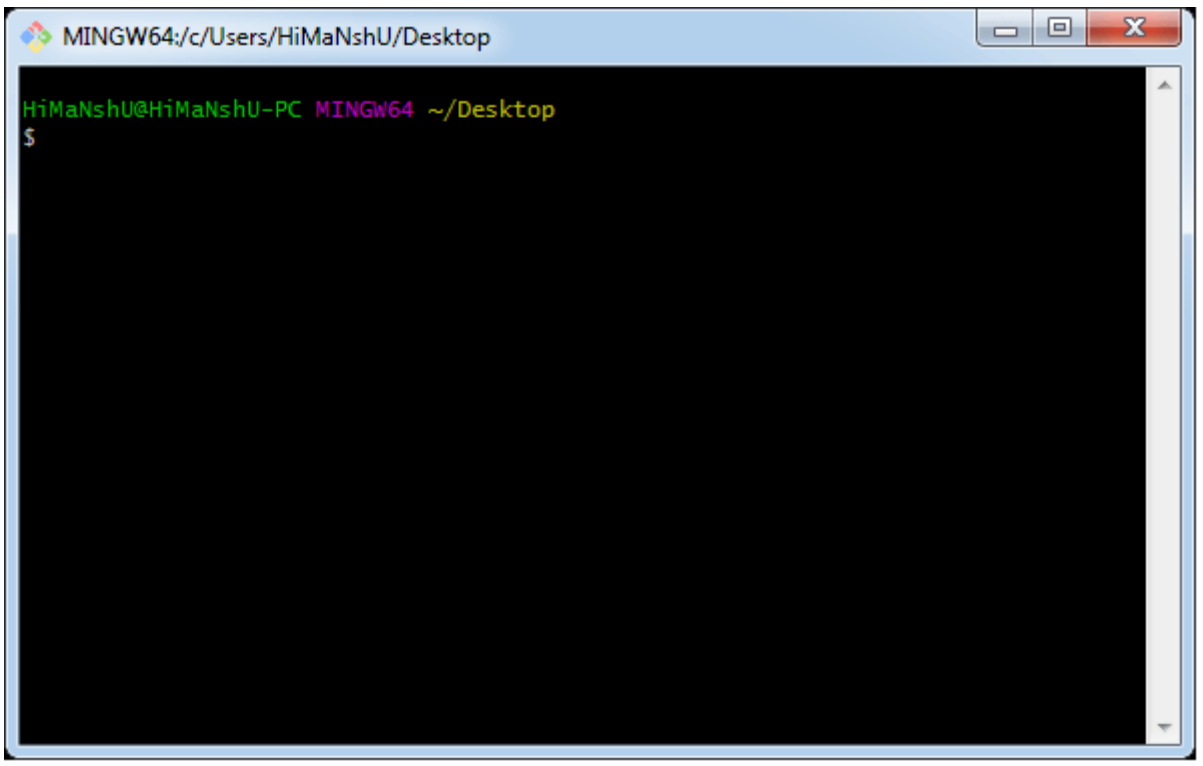
The Git Gui looks like as



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.

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.

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 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.

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.

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.

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 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.