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**.

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.
- **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.
- **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.

- **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.

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
- **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, compares** our files or branches.

How to Install Git on Windows

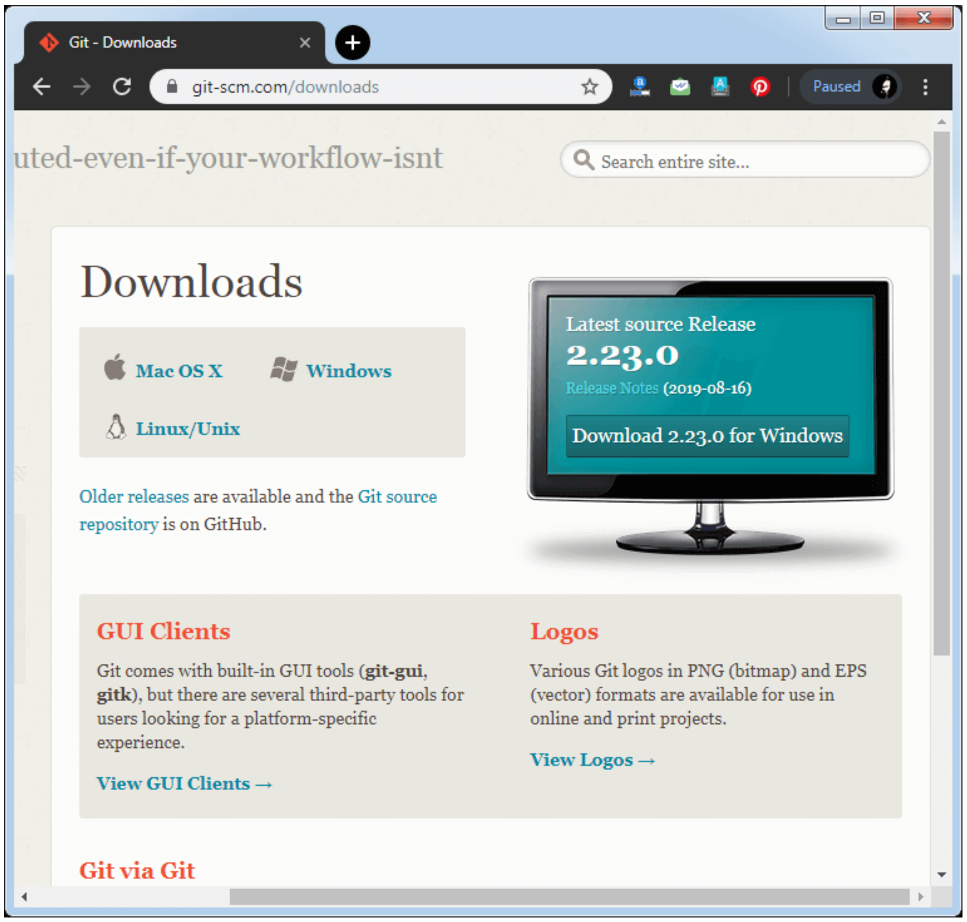
To use Git, you have to install it on your computer. Even if you have already installed Git, it's probably a good idea to upgrade it to the latest version. You can either install it as a package or via another installer or download it from its official site.

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



Downloads

Mac OS X Windows

Linux/Unix

Older releases are available and the Git source repository is on GitHub.

Latest source Release
2.23.0
Release Notes (2019-08-16)
Download 2.23.0 for Windows

GUI Clients

Git comes with built-in GUI tools (**git-gui**, **gitk**), but there are several third-party tools for users looking for a platform-specific experience.

View GUI Clients →

Logos

Various Git logos in PNG (bitmap) and EPS (vector) formats are available for use in online and print projects.

View Logos →

Git via Git

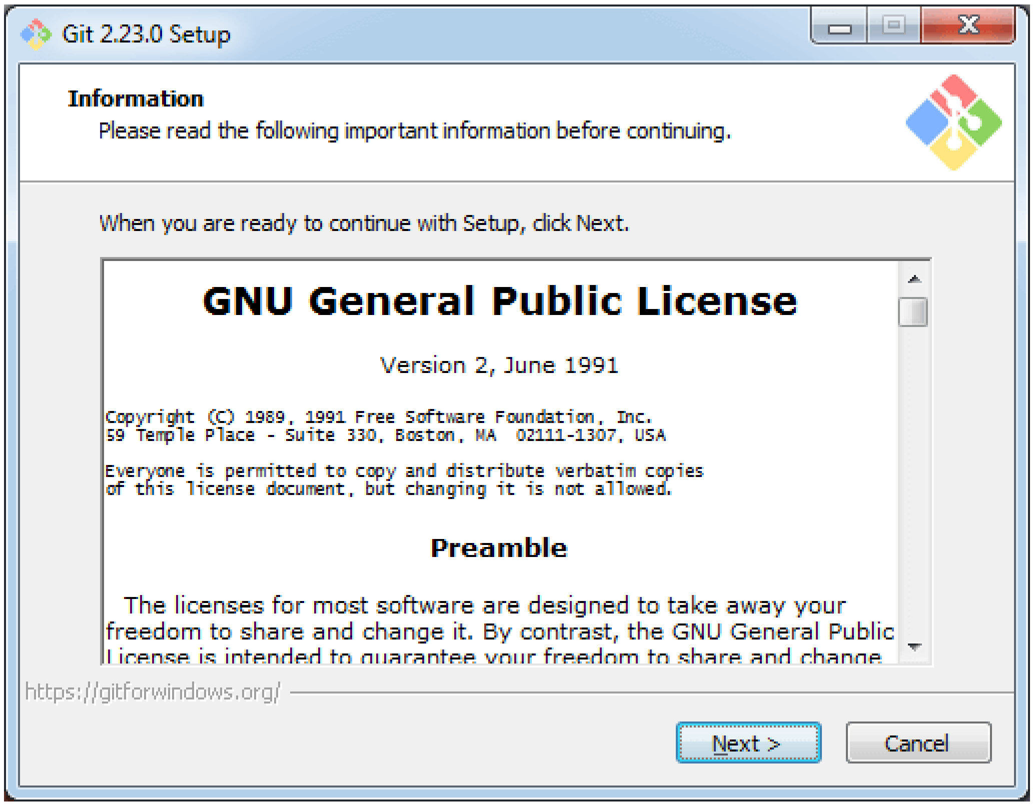
Click on the package given on the page as **download 2.23.0 for windows**. The download will start after selecting the package.

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



Please read the following important information before continuing.

When you are ready to continue with Setup, click Next.

GNU General Public License

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place - Suite 330, Boston, MA 02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

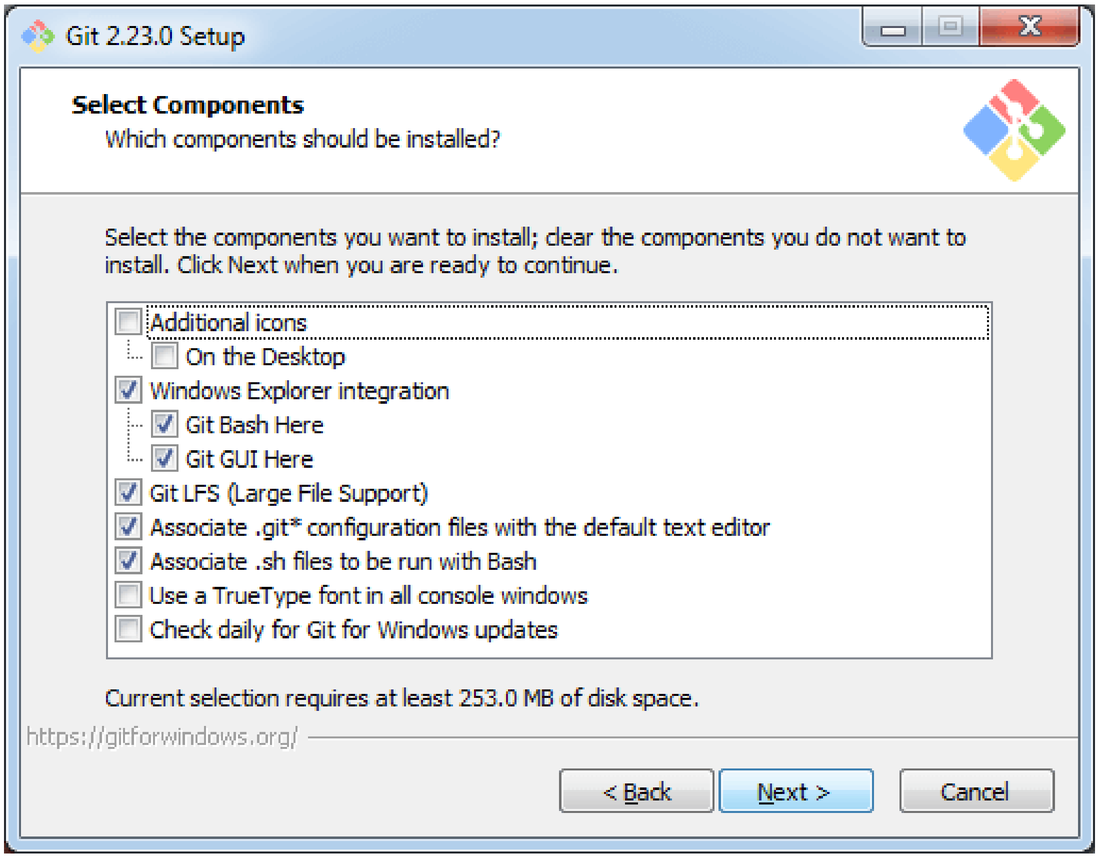
The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change

https://gitforwindows.org/

Click on **next** to continue.

Step3

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



Select Components

Which components should be installed?

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

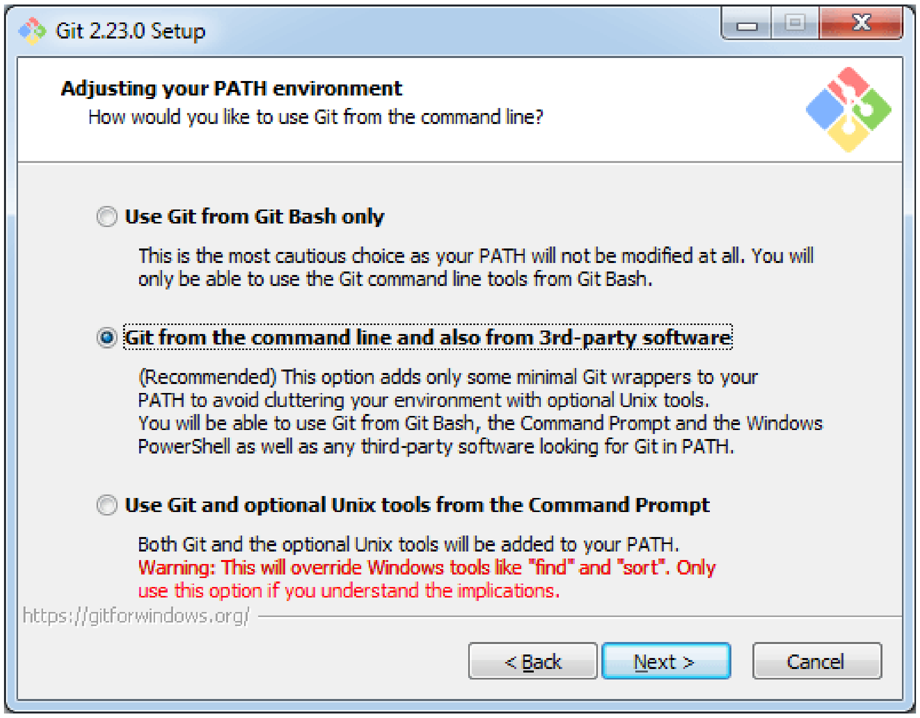
- [ ] Additional icons
  - [ ] On the Desktop
- [✓] Windows Explorer integration
  - [✓] Git Bash Here
  - [✓] Git GUI Here
- [✓] Git LFS (Large File Support)
- [✓] Associate .git* configuration files with the default text editor
- [✓] Associate .sh files to be run with Bash
- [ ] Use a TrueType font in all console windows
- [ ] Check daily for Git for Windows updates

Current selection requires at least 253.0 MB of disk space.

Click next to continue.

Step4

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



Git 2.23.0 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 will 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 Windows PowerShell as well as any third-party software looking for Git in PATH.

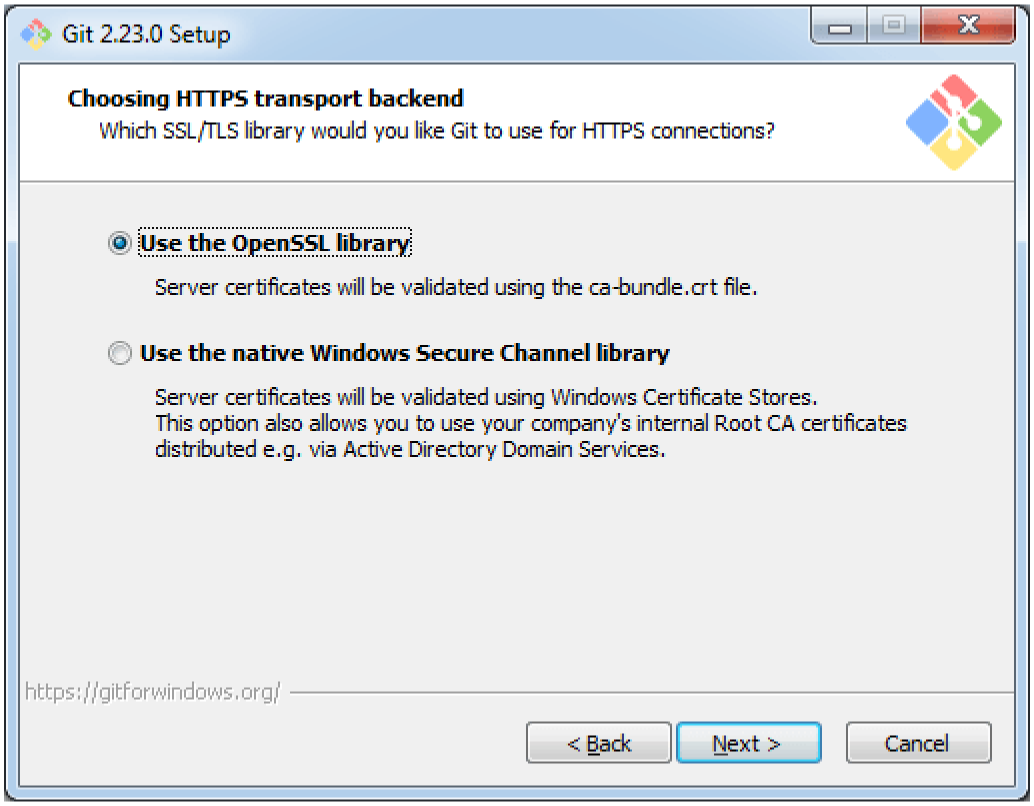
( ) **Use Git and optional Unix tools from the Command Prompt**

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/

Step5

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



Git 2.23.0 Setup

Choosing HTTPS transport backend

Which SSL/TLS library would you like Git to use for HTTPS connections?

(●) **Use the OpenSSL library**

Server certificates will be validated using the ca-bundle.crt file.

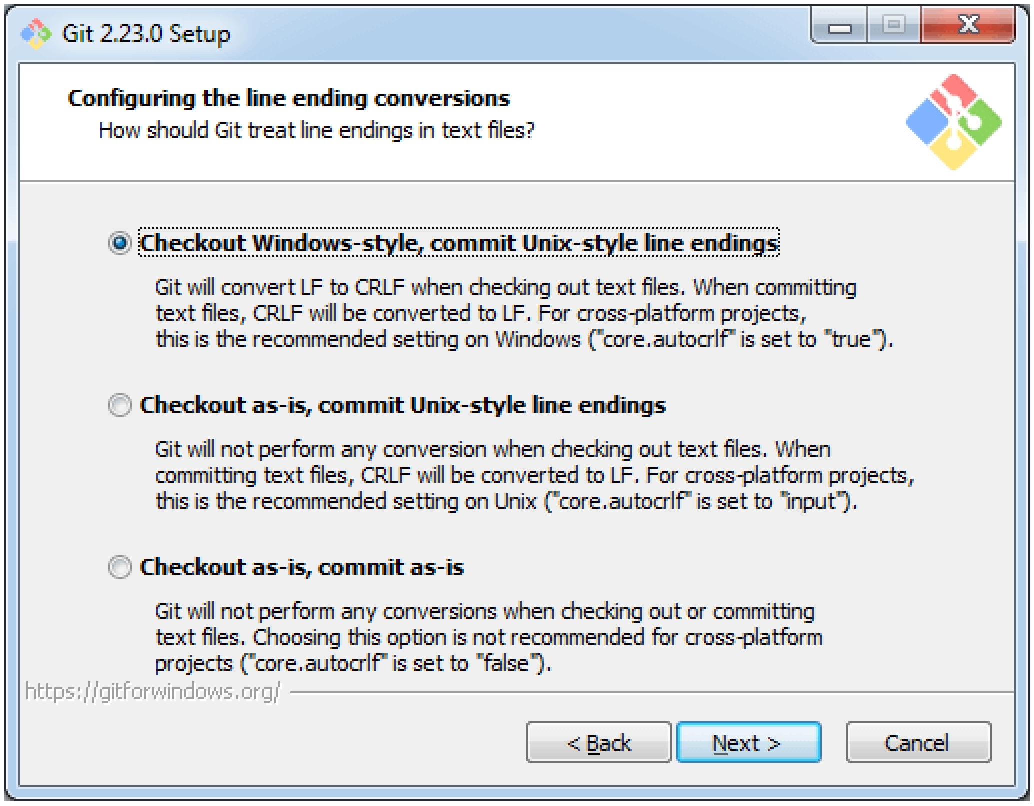
( ) **Use the native Windows Secure Channel library**

Server certificates will be validated using Windows Certificate Stores. This option also allows you to use your company's internal Root CA certificates distributed e.g. via Active Directory Domain Services.

https://gitforwindows.org/

Step6

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


Git 2.23.0 Setup

Configuring the line ending conversions

How should Git treat line endings in text files?

[✓ **Checkout Windows-style, commit Unix-style line endings**]

Git will convert LF to CRLF when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Windows ("core.autocrlf" is set to "true").

Checkout as-is, commit Unix-style line endings

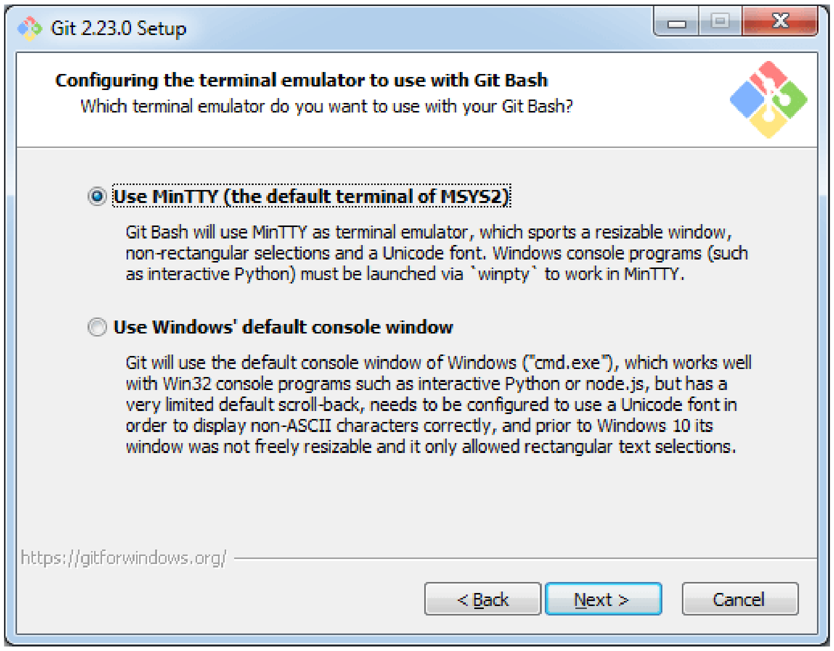
Git will not perform any conversion when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Unix ("core.autocrlf" is set to "input").

Checkout as-is, commit as-is

Git will not perform any conversions when checking out or committing text files. Choosing this option is not recommended for cross-platform projects ("core.autocrlf" is set to "false").

Step7

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


Git 2.23.0 Setup

Configuring the terminal emulator to use with Git Bash

Which terminal emulator do you want to use with your Git Bash?

[✓] **Use MinTTY (the default terminal of MSYS2)**

Git Bash will use MinTTY as terminal emulator, which sports a resizable window, non-rectangular selections and a Unicode font. Windows console programs (such as interactive Python) must be launched via `winpty` to work in MinTTY.

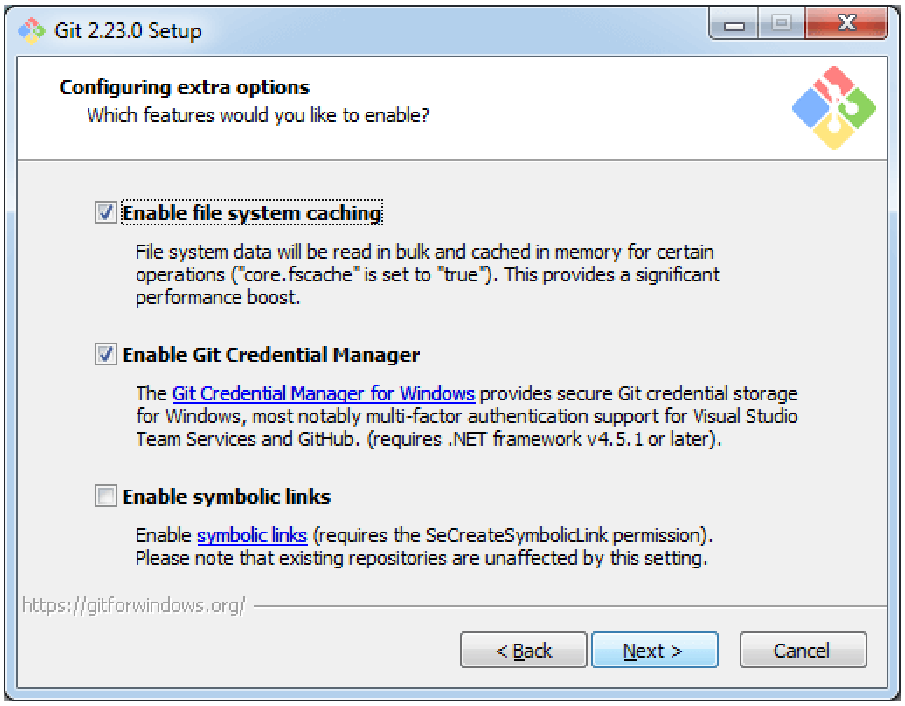
[ ] **Use Windows' default console window**

Git will use the default console window of Windows ("cmd.exe"), which works well with Win32 console programs such as interactive Python or node.js, but has a very limited default scroll-back, needs to be configured to use a Unicode font in order to display non-ASCII characters correctly, and prior to Windows 10 its window was not freely resizable and it only allowed rectangular text selections.

https://gitforwindows.org/

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.


Git 2.23.0 Setup

Configuring extra options

Which features would you like to enable?

[✓] **Enable file system caching**

File system data will be read in bulk and cached in memory for certain operations ("core.fscache" is set to "true"). This provides a significant performance boost.

[✓] **Enable Git Credential Manager**

The Git Credential Manager for Windows provides secure Git credential storage for Windows, most notably multi-factor authentication support for Visual Studio Team Services and GitHub. (requires .NET framework v4.5.1 or later).

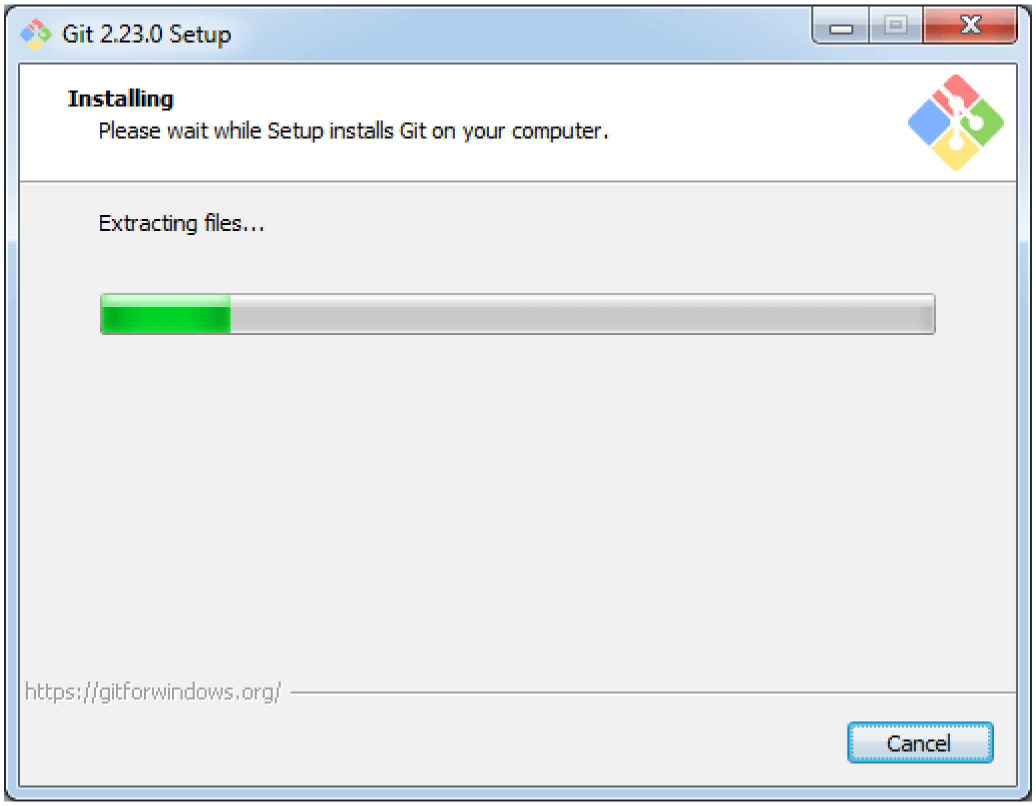
[ ] **Enable symbolic links**

Enable symbolic links (requires the SeCreateSymbolicLink permission). Please note that existing repositories are unaffected by this setting.

https://gitforwindows.org/

Step9

The files are being extracted in this step.



Installing

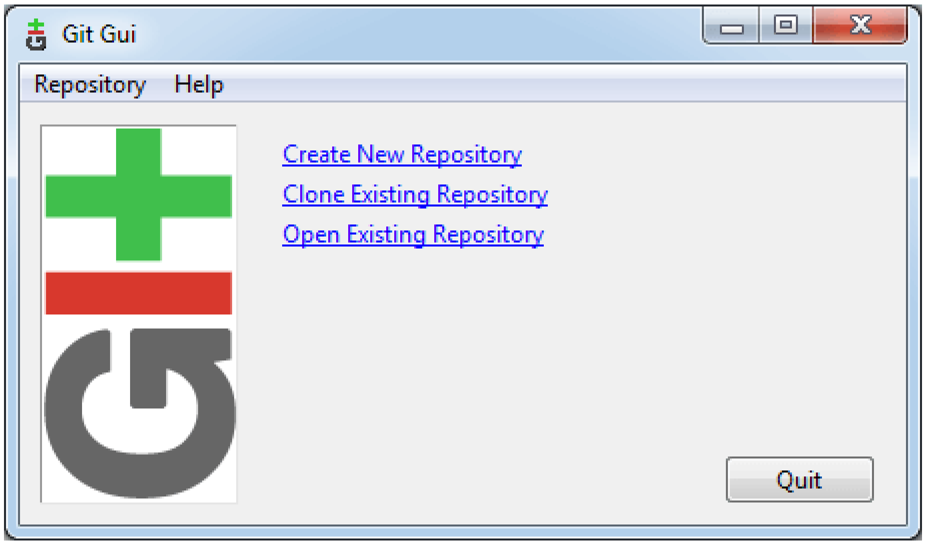
Please wait while Setup installs Git on your computer.

Extracting files...

https://gitforwindows.org/

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

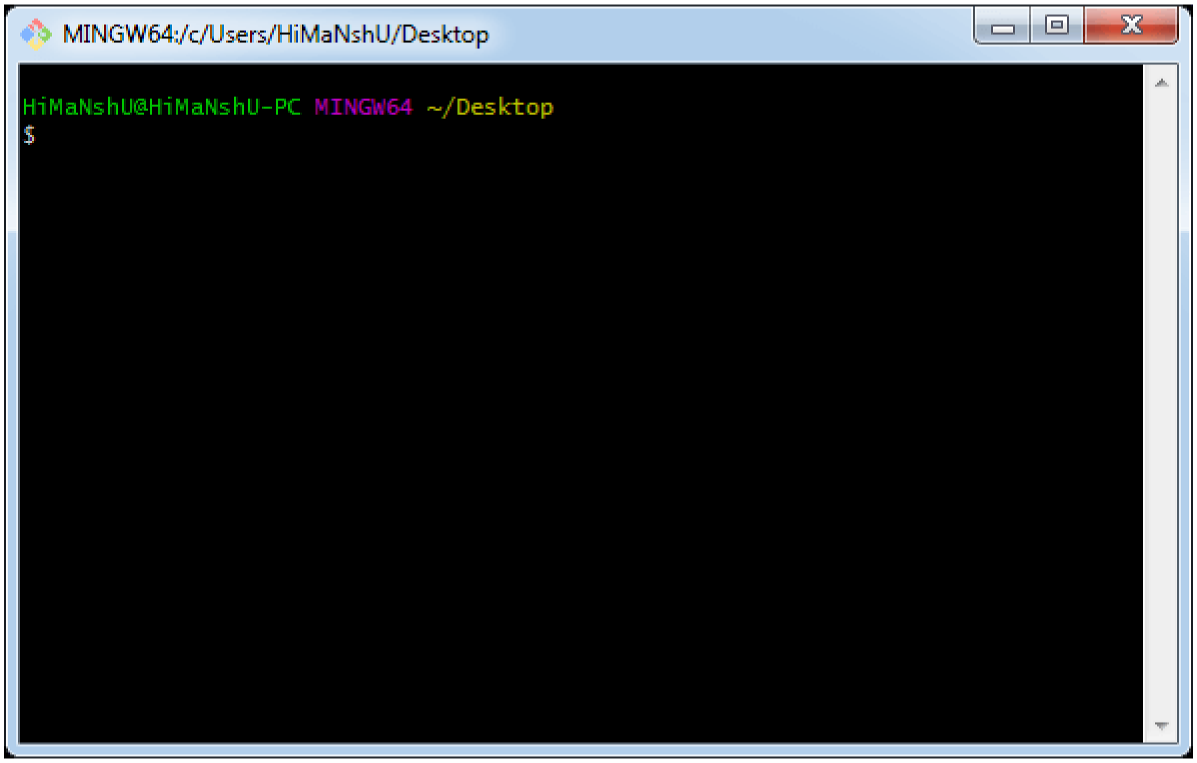
The **Git Gui** looks like as



It facilitates with three features.

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

The **Git Bash** looks like as



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

Staging & commits

Git Init

The git init command is the first command that you will run on Git. The git init command is used to create a new blank repository. It is used to make an existing project as a Git project. Several Git commands run inside the repository, but init command can be run outside of the repository.

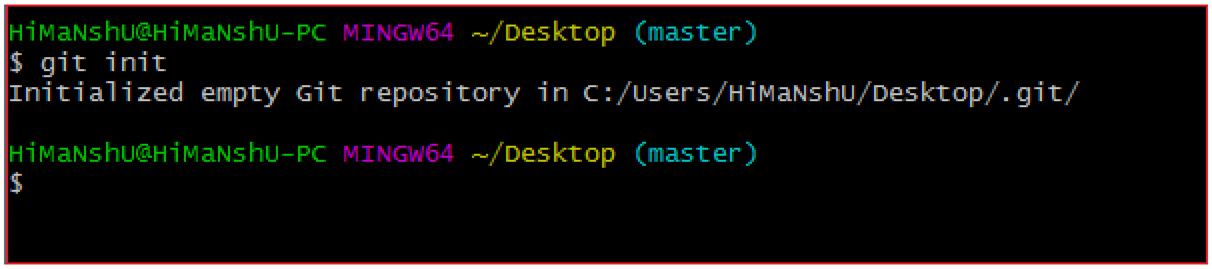
The git init command creates a .git subdirectory in the current working directory. This newly created subdirectory contains all of the necessary metadata. These metadata can be categorized into objects, refs, and temp files. It also initializes a HEAD pointer for the master branch of the repository.

Create a Repository for a Blank (New) Project:

To create a blank repository, open command line on your desired directory and run the init command as follows:

1. **$ git init**

The above command will create an empty .git repository. Suppose we want to make a git repository on our desktop. To do so, open Git Bash on the desktop and run the above command. Consider the below output:



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

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

Git Add

The git add command is used to add file contents to the Index (Staging Area).This command updates the current content of the working tree to the staging area. It also prepares the staged content for the next commit. Every time we add or update any file in our project, it is required to forward updates to the staging area.

The git add command is a core part of Git technology. It typically adds one file at a time, but there some options are available that can add more than one file at once.

The "index" contains a snapshot of the working tree data. This snapshot will be forwarded for the next commit.

The git add command can be run many times before making a commit. These all add operations can be put under one commit. The add command adds the files that are specified on command line.

Git add files

Git add command is a straight forward command. It adds files to the staging area. We can add single or multiple files at once in the staging area. It will be run as:

```
$ git add <File name>
```

Git Commit

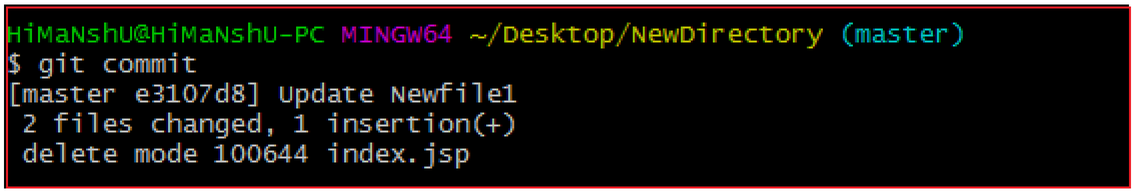
It is used to record the changes in the repository. It is the next command after the git add. Every commit contains the index data and the commit message. Every commit forms a parent-child relationship. When we add a file in Git, it will take place in the staging area. A commit command is used to fetch updates from the staging area to the repository.

The staging and committing are co-related to each other. Staging allows us to continue in making changes to the repository, and when we want to share these changes to the version control system, committing allows us to record these changes.

The git commit command

The commit command will commit the changes and generate a commit-id. The commit command without any argument will open the default text editor and ask for the commit message. We can specify our commit message in this text editor. It will run as follows:

```
$ git commit
```



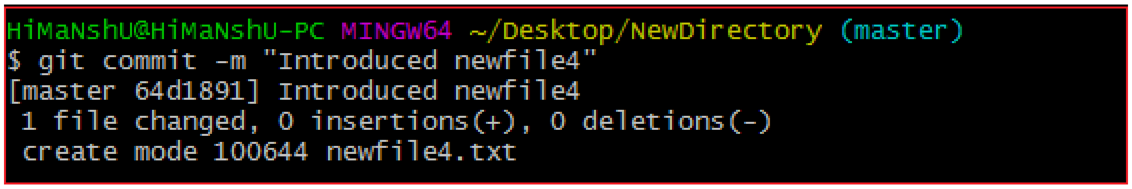
```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/NewDirectory (master)
$ git commit
[master e3107d8] Update Newfile1
 2 files changed, 1 insertion(+)
 delete mode 100644 index.jsp
```

Git commit -m

The -m option of commit command lets you to write the commit message on the command line. This command will not prompt the text editor. It will run as follows:

$ git commit -m "Commit message."

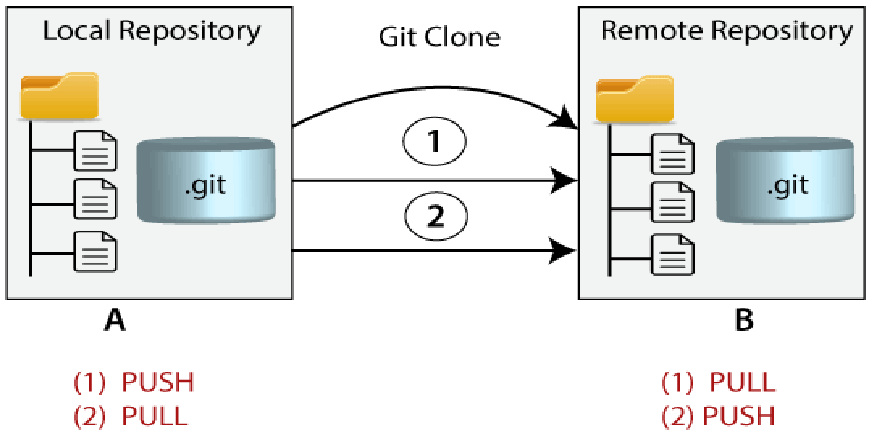
The above command will make a commit with the given commit message. Consider the below output:



```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/NewDirectory (master)
$ git commit -m "Introduced newfile4"
[master 64d1891] Introduced newfile4
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 newfile4.txt
```

Git Clone

In Git, cloning is the act of making a copy of any target repository. The target repository can be remote or local. You can clone your repository from the remote repository to create a local copy on your system. Also, you can sync between the two locations.



Git Clone Command

Syntax:

```
git clone <repository 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**.

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:

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

Git Origin Master

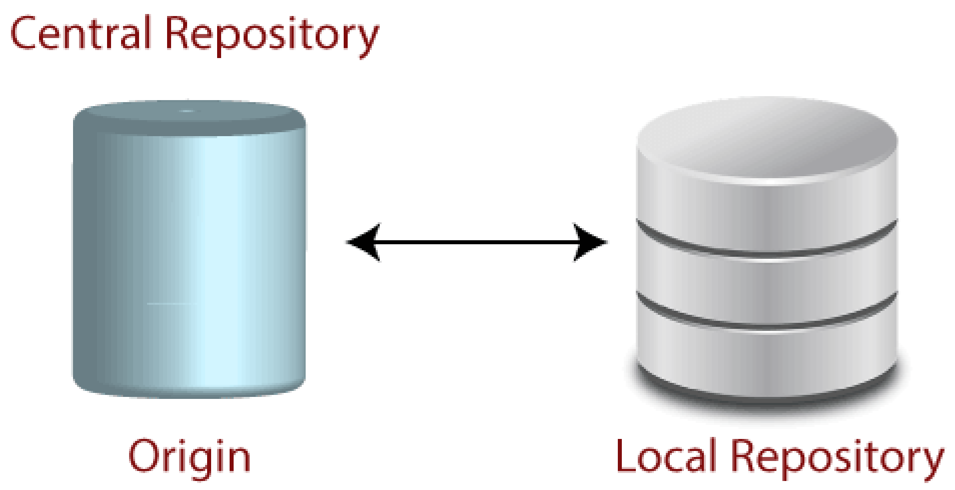
The term "git origin master" is used in the context of a remote repository. It is used to deal with the remote repository. The term origin comes from where repository original situated and master stands for the main branch.

Git 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.

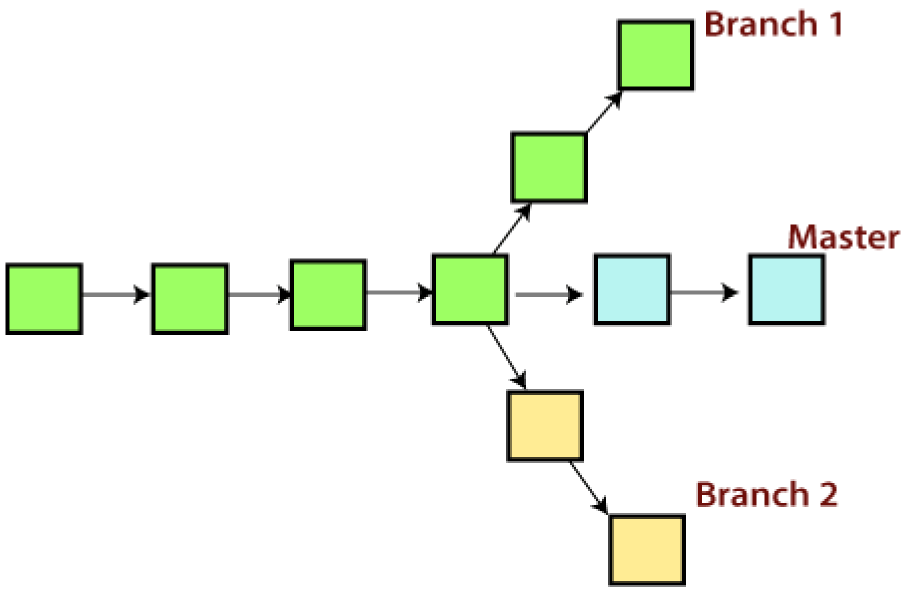
Git Origin

In Git, The term origin is referred to the remote repository where you want to publish your commits. The default remote repository is called **origin**, although you can work with several remotes having a different name at the same time. It is said as an alias of the system.



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.



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.

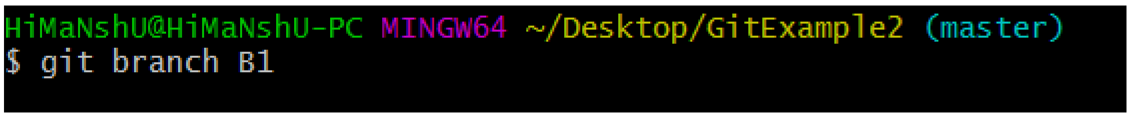
Create Branch

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

Syntax:

$ 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.

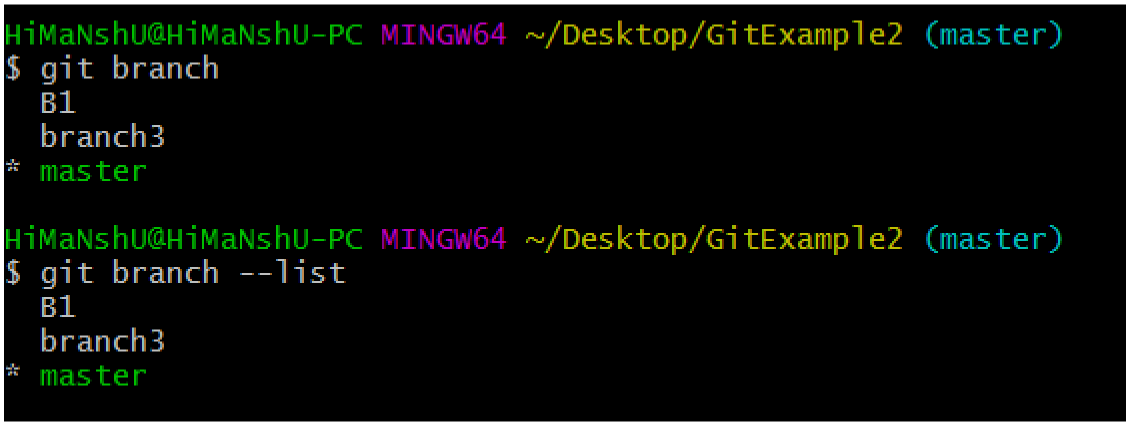
Syntax:

$ git branch --list

or

$ 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.

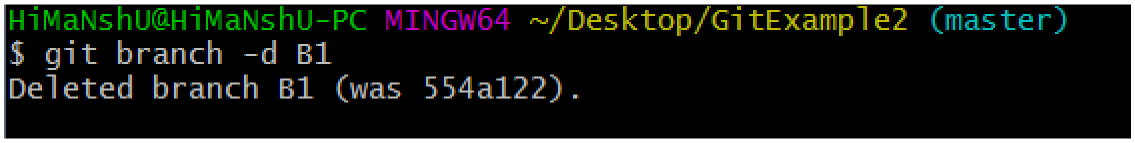
Delete Branch

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:

$ 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.

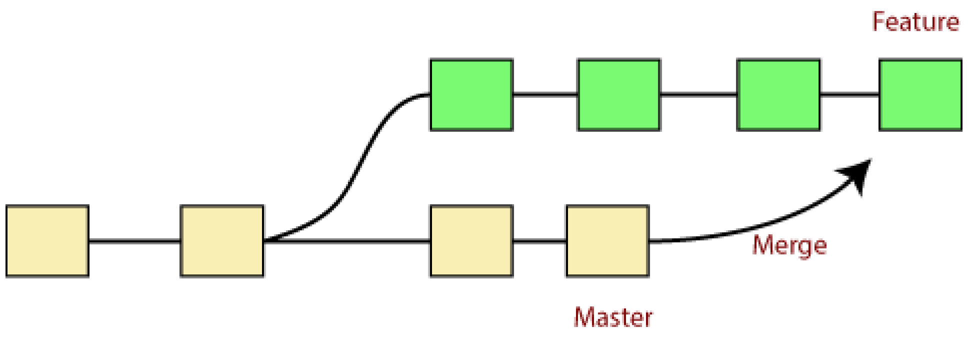
Switch Branch

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:

$ git checkout<branch name>

Git Merge and Merge Conflict

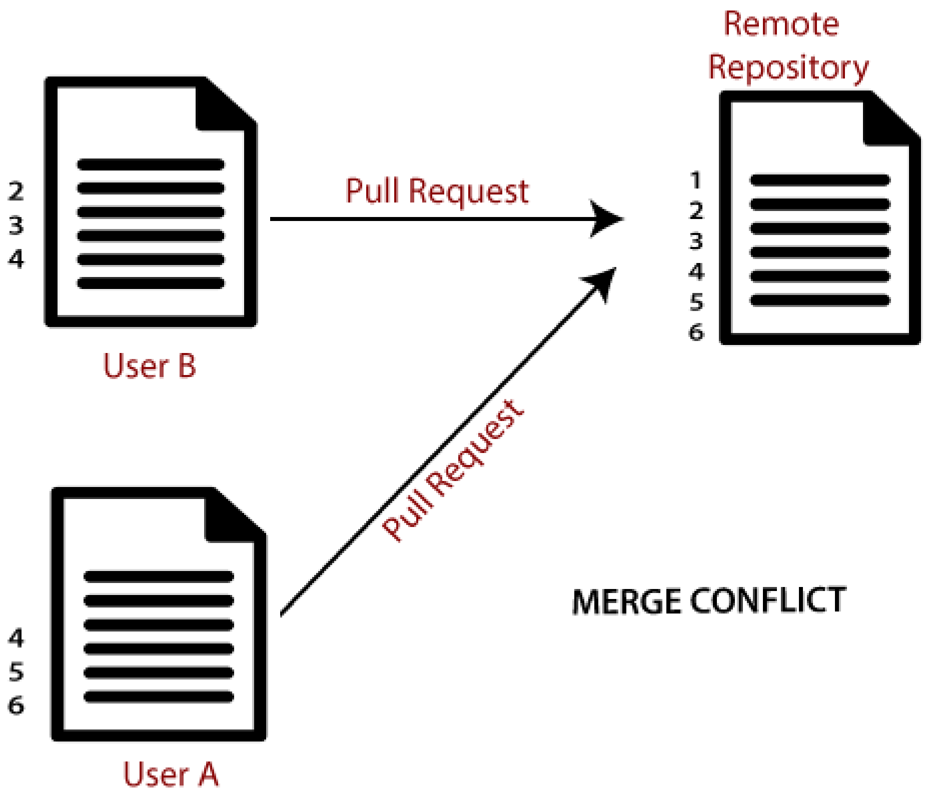
In Git, the merging is a procedure to connect the forked history. It joins two or more development history together. The git merge command facilitates you to take the data created by git branch and integrate them into a single branch. Git merge will associate a series of commits into one unified history.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

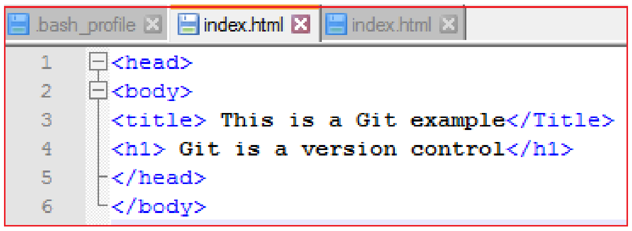
Git Merge Conflict

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.



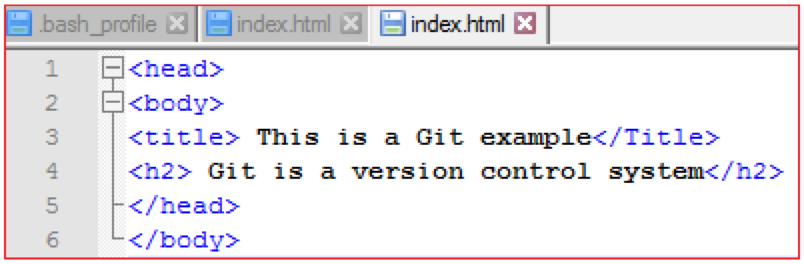
Let's understand it by an example.

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.



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

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



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

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.

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:

$ git mergetool

To accept the changes, use the rebase command. It will be used as follows:

$ git rebase --continue