Git Tutorial



**Git tutorial** provides basic and advanced concepts of Git and GitHub. Our Git tutorial is designed for beginners and professionals.

Git is a modern and widely used **distributed version control** system in the world. It is developed to manage projects with high speed and efficiency. The version control system allows us to monitor and work together with our team members at the same workspace.

This tutorial will help you to understand the distributed version control system Git via the command line as well as with [GitHub](https://www.javatpoint.com/what-is-github). The examples in this tutorial are performed on **Windows**, but we can also perform same operations on other operating systems like **Linux (Ubuntu)** and **MacOS**.

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**.

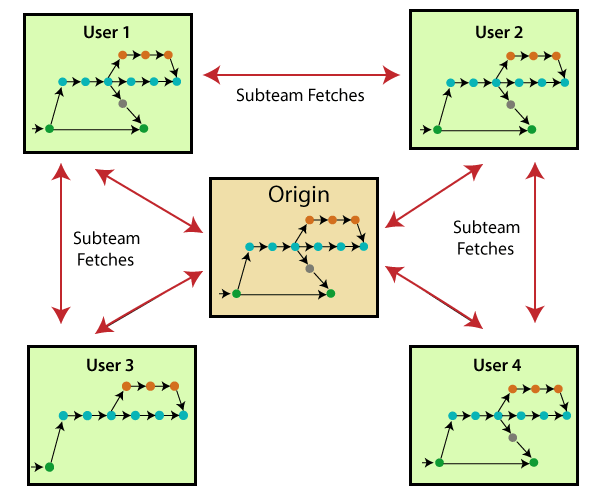
Git is easy to learn, and has fast performance. It is superior to other SCM tools like Subversion, CVS, Perforce, and ClearCase.

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 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.

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.

# 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

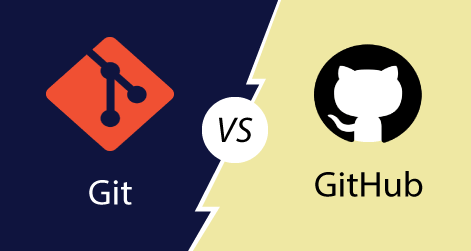
GitHub can be separated as the Git and the Hub. GitHub service includes access controls as well as collaboration features like task management, repository hosting, and team management.

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.

Git vs GitHub

**Git is an open-source distributed version control system** which is available for everyone at zero cost. It is designed to handle minor to major projects with speed and efficiency. It is developed to co-ordinate the work among programmers. The version control allows you to track and work together with your team member at the same workspace.



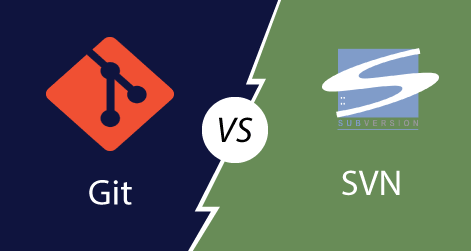
While **GitHub is a Git repository hosting service**. It is a web-based service. GitHub facilitates with all of the features of distributed version control and source code management (SCM) functionality of Git. It also supports some of its characteristics in a single software tool.

To better understand the similarities and differences between Git and GitHub, look at the following points.

|  |  |
| --- | --- |
| **Git** | **GitHub** |
| Git is a distributed version control tool that can manage a programmer's source code history. | GitHub is a cloud-based tool developed around the Git tool. |
| A developer installs Git tool locally. | GitHub is an online service to store code and push from the computer running the Git tool. |
| Git focused on version control and code sharing. | GitHub focused on centralized source code hosting. |
| It is a command-line tool. | It is administered through the web. |
| It facilitates with a desktop interface called Git Gui. | It also facilitates with a desktop interface called GitHub Gui. |
| Git does not provide any user management feature. | GitHub has a built-in user management feature. |
| It has minimal tool configuration feature. | It has a market place for tool configuration. |

Git vs SVN

Apache Subversion or **SVN is one of the most popular centralized version control systems**. Now, SVN's popularity is on the decrease, but there are still millions of projects stored in it. It can continue to be actively maintained by an open-source community. In SVN, you can check out a single version of the repository. It stores data in a central server. The drawback of the SVN is, it has the entire history on a local repository which limits you. You can only do commits, diffs, logs, branches, merges, file annotations, etc.



While, **Git is a popular distributed version control system**, which means that you can clone your repository. Thus you can get a complete copy of your entire history of that project. This means you can access all your commits.

**Git has more advantages than SVN**. It is much better for those developers who are not always connected to the master repository. Also, it is much faster than SVN.

To better understand the differences between Git and Subversion. Let's have a look at following significance points.

|  |  |
| --- | --- |
| **Git** | **SVN** |
| It's a distributed version control system. | It's a Centralized version control system |
| Git is an SCM (source code management). | SVN is revision control. |
| Git has a cloned repository. | SVN does not have a cloned repository. |
| The Git branches are familiar to work. The Git system helps in merging the files quickly and also assist in finding the unmerged ones. | The SVN branches are a folder which exists in the repository. Some special commands are required For merging the branches. |
| Git does not have a Global revision number. | SVN has a Global revision number. |
| Git has cryptographically hashed contents that protect the contents from repository corruption taking place due to network issues or disk failures. | SVN does not have any cryptographically hashed contents. |
| Git stored content as metadata. | SVN stores content as files. |
| Git has more content protection than SVN. | SVN's content is less secure than Git. |
| Linus Torvalds developed git for Linux kernel. | CollabNet, Inc developed SVN. |
| Git is distributed under GNU (General public license). | SVN is distributed under the open-source license. |

# How to Install Git on Windows

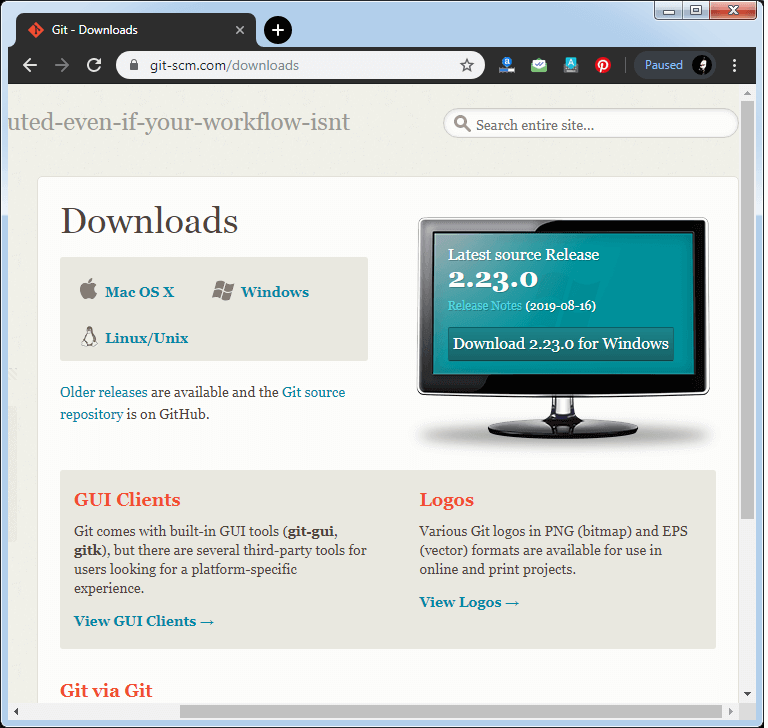
To use Git, you have to install it on your computer. Even if you have already installed Git, it's probably a good idea to upgrade it to the latest version. You can either install it as a package or via another installer or download it from its official site.

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



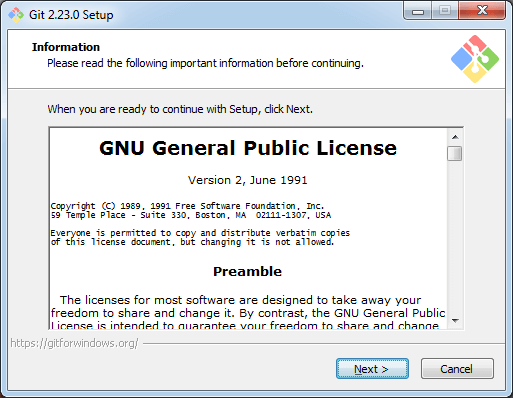
Click on the package given on the page as **download 2.23.0 for windows**. The download will start after selecting the package.

Now, the Git installer package has been downloaded.

## Install Git

**Step2**

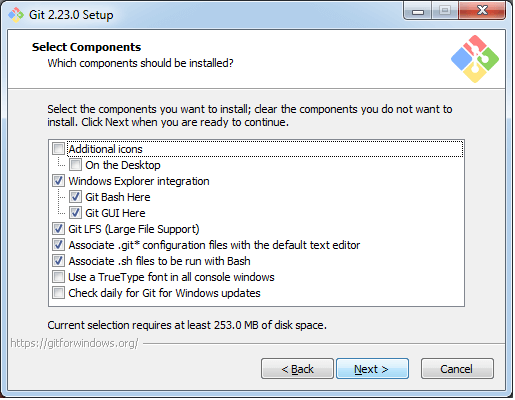
Click on the downloaded installer file and select **yes** to continue. After the selecting **yes** the installation begins, and the screen will look like as



Click on **next** to continue.

**Step3**

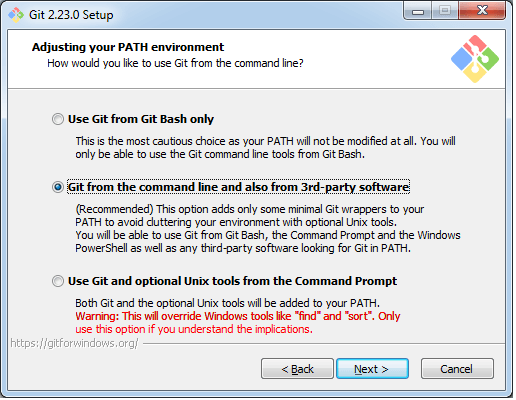
Default components are automatically selected in this step. You can also choose your required part.



Click next to continue.

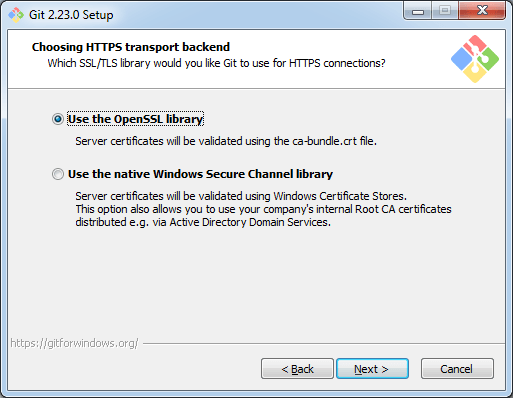
**Step4**

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



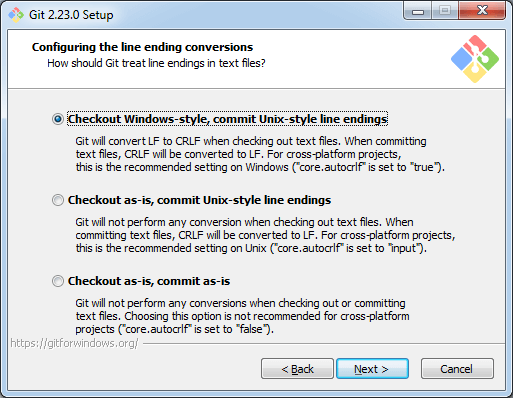
**Step5**

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



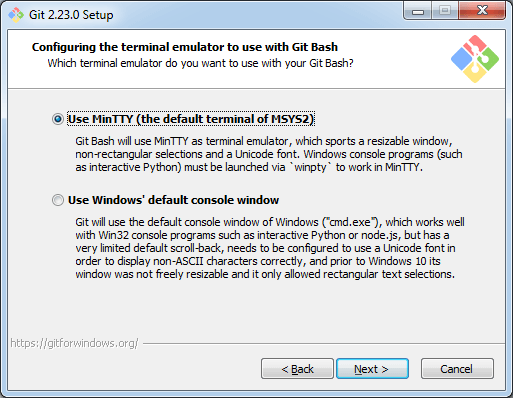
**Step6**

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



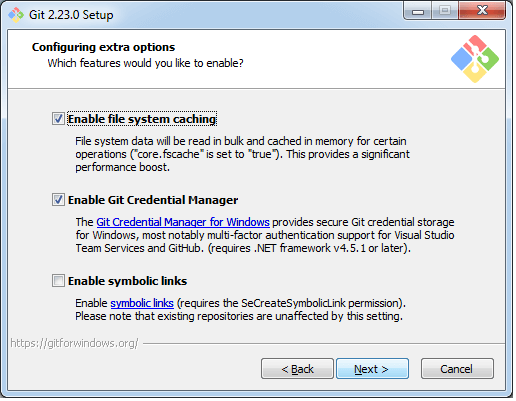
**Step7**

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



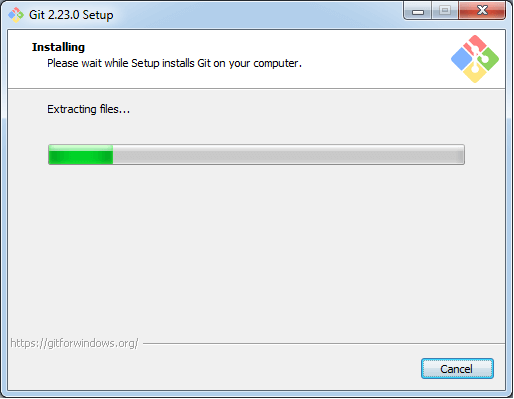
**Step8**

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.



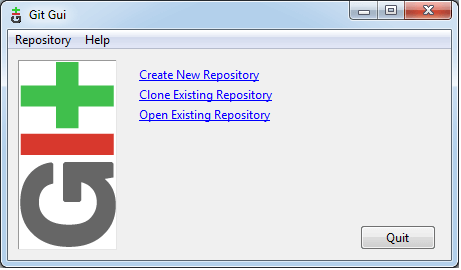
**Step9**

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.

[**Next →**](https://www.javatpoint.com/git-flow)[**← Prev**](https://www.javatpoint.com/git-terminology)

# Git command line

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.

* [Git Config command](https://www.javatpoint.com/git-commands#config-command)
* [Git init command](https://www.javatpoint.com/git-commands#init-command)
* [Git clone command](https://www.javatpoint.com/git-commands#clone-command)
* [Git add command](https://www.javatpoint.com/git-commands#add-command)
* [Git commit command](https://www.javatpoint.com/git-commands#commit-command)
* [Git status command](https://www.javatpoint.com/git-commands#status-command)
* [Git push Command](https://www.javatpoint.com/git-commands#push-command)
* [Git pull command](https://www.javatpoint.com/git-commands#pull-command)
* [Git Branch Command](https://www.javatpoint.com/git-commands#branch-command)
* [Git Merge Command](https://www.javatpoint.com/git-commands#merge-command)
* [Git log command](https://www.javatpoint.com/git-commands#log-command)
* [Git remote command](https://www.javatpoint.com/git-commands#remote-command)

Let's understand each command in detail.

### **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"

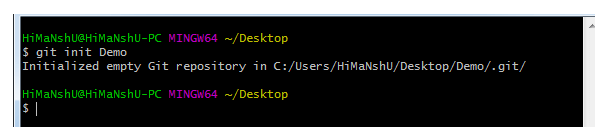
### **Git Init command**

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.

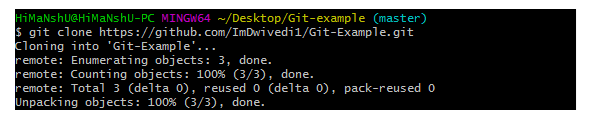


### **Git clone command**

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



### **Git add command**

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

**Syntax**

To add one file

1. $ git add Filename

To add more than one file

1. $ git add\*

Git Commands

### **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**

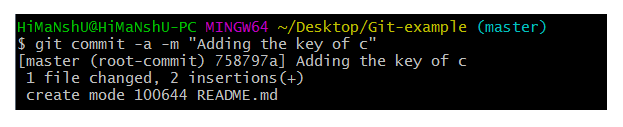
1. $ git commit -m " Commit Message"

**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

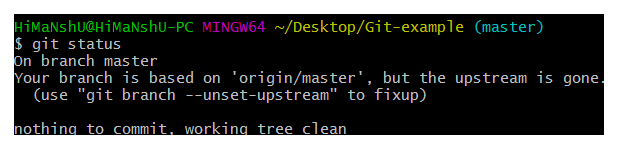


### **Git status command**

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



### **Git push Command**

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

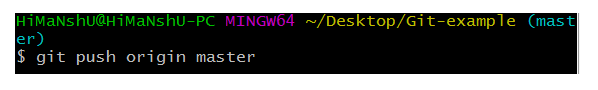
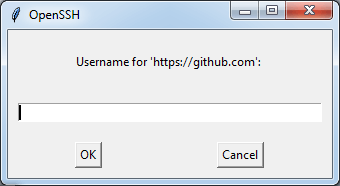
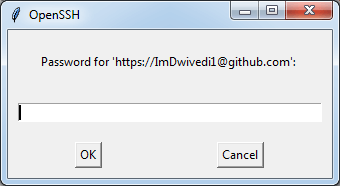
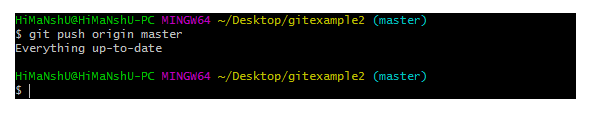
Git push command can be used as follows.

**Git push origin master**

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

**Syntax**

1. $ git push [variable name] master

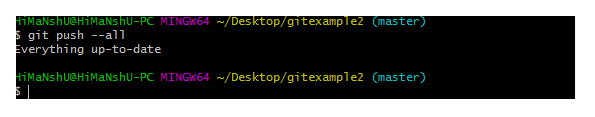
  
  
  


**Git push -all**

This command pushes all the branches to the server repository.

**Syntax**

1. $ git push --all

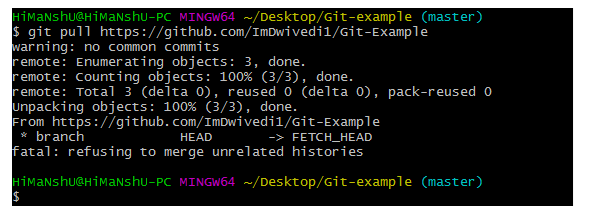


### **Git pull command**

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

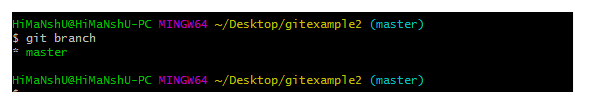


### **Git Branch Command**

This command lists all the branches available in the repository.

**Syntax**

1. $ git branch



### **Git Merge Command**

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

**Syntax**

1. $ git merge BranchName

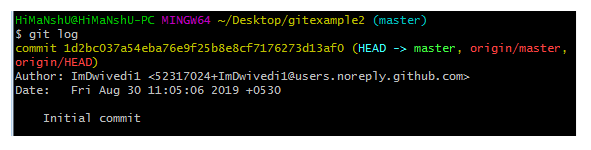


### **Git log Command**

This command is used to check the commit history.

**Syntax**

1. $ git log

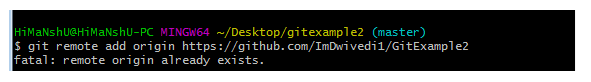


By default, if no argument passed, Git log shows the most recent commits first. We can limit the number of log entries displayed by passing a number as an option, such as -3 to show only the last three entries.

1. $ git log -3

### **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.



The **Hello World** project is a time-honored tradition in computer programming. It is a simple exercise that gets you started when learning something new. Let’s get started with GitHub!

**You’ll learn how to:**

* Create and use a repository
* Start and manage a new branch
* Make changes to a file and push them to GitHub as commits
* Open and merge a pull request

## What is GitHub?

GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere.

This tutorial teaches you GitHub essentials like repositories, branches, commits, and Pull Requests. You’ll create your own Hello World repository and learn GitHub’s Pull Request workflow, a popular way to create and review code.

#### No coding necessary

To complete this tutorial, you need a [GitHub.com account](http://github.com/) and Internet access. You don’t need to know how to code, use the command line, or install Git (the version control software GitHub is built on).

**Tip:** Open this guide in a separate browser window (or tab) so you can see it while you complete the steps in the tutorial.

## Step 1. Create a Repository

A **repository** is usually used to organize a single project. Repositories can contain folders and files, images, videos, spreadsheets, and data sets – anything your project needs. We recommend including a README, or a file with information about your project. GitHub makes it easy to add one at the same time you create your new repository. It also offers other common options such as a license file.

Your hello-world repository can be a place where you store ideas, resources, or even share and discuss things with others.

### To create a new repository

1. In the upper right corner, next to your avatar or identicon, click  and then select **New repository**.
2. Name your repository hello-world.
3. Write a short description.
4. Select **Initialize this repository with a README**.



Click **Create repository**.

## Step 2. Create a Branch

**Branching** is the way to work on different versions of a repository at one time.

By default your repository has one branch named main which is considered to be the definitive branch. We use branches to experiment and make edits before committing them to main.

When you create a branch off the main branch, you’re making a copy, or snapshot, of main as it was at that point in time. If someone else made changes to the main branch while you were working on your branch, you could pull in those updates.

This diagram shows:

* The main branch
* A new branch called feature (because we’re doing ‘feature work’ on this branch)
* The journey that feature takes before it’s merged into main



Have you ever saved different versions of a file? Something like:

* story.txt
* story-joe-edit.txt
* story-joe-edit-reviewed.txt

Branches accomplish similar goals in GitHub repositories.

Here at GitHub, our developers, writers, and designers use branches for keeping bug fixes and feature work separate from our main (production) branch. When a change is ready, they merge their branch into main.

### To create a new branch

1. Go to your new repository hello-world.
2. Click the drop down at the top of the file list that says **branch: main**.
3. Type a branch name, readme-edits, into the new branch text box.
4. Select the blue **Create branch** box or hit “Enter” on your keyboard.



Now you have two branches, main and readme-edits. They look exactly the same, but not for long! Next we’ll add our changes to the new branch.

## Step 3. Make and commit changes

Bravo! Now, you’re on the code view for your readme-edits branch, which is a copy of main. Let’s make some edits.

On GitHub, saved changes are called commits. Each commit has an associated commit message, which is a description explaining why a particular change was made. Commit messages capture the history of your changes, so other contributors can understand what you’ve done and why.

#### Make and commit changes

1. Click the README.md file.
2. Click the  pencil icon in the upper right corner of the file view to edit.
3. In the editor, write a bit about yourself.
4. Write a commit message that describes your changes.
5. Click **Commit changes** button.



These changes will be made to just the README file on your readme-edits branch, so now this branch contains content that’s different from main.

## Step 4. Open a Pull Request

Nice edits! Now that you have changes in a branch off of main, you can open a pull request.

Pull Requests are the heart of collaboration on GitHub. When you open a pull request, you’re proposing your changes and requesting that someone review and pull in your contribution and merge them into their branch. Pull requests show diffs, or differences, of the content from both branches. The changes, additions, and subtractions are shown in green and red.

As soon as you make a commit, you can open a pull request and start a discussion, even before the code is finished.

By using GitHub’s [@mention system](https://help.github.com/articles/about-writing-and-formatting-on-github/#text-formatting-toolbar) in your pull request message, you can ask for feedback from specific people or teams, whether they’re down the hall or 10 time zones away.

You can even open pull requests in your own repository and merge them yourself. It’s a great way to learn the GitHub flow before working on larger projects.

#### Open a Pull Request for changes to the README

Click on the image for a larger version

| **Step** | **Screenshot** |
| --- | --- |
| Click the  **Pull Request** tab, then from the Pull Request page, click the green **New pull request** button. | [pr-tab](https://guides.github.com/activities/hello-world/pr-tab.gif) |
| In the **Example Comparisons** box, select the branch you made, readme-edits, to compare with main (the original). | [branch](https://guides.github.com/activities/hello-world/pick-branch.png) |
| Look over your changes in the diffs on the Compare page, make sure they’re what you want to submit. | [diff](https://guides.github.com/activities/hello-world/diff.png) |
| When you’re satisfied that these are the changes you want to submit, click the big green **Create Pull Request** button. | [create-pull](https://guides.github.com/activities/hello-world/create-pr.png) |
| Give your pull request a title and write a brief description of your changes. | [pr-form](https://guides.github.com/activities/hello-world/pr-form.png) |

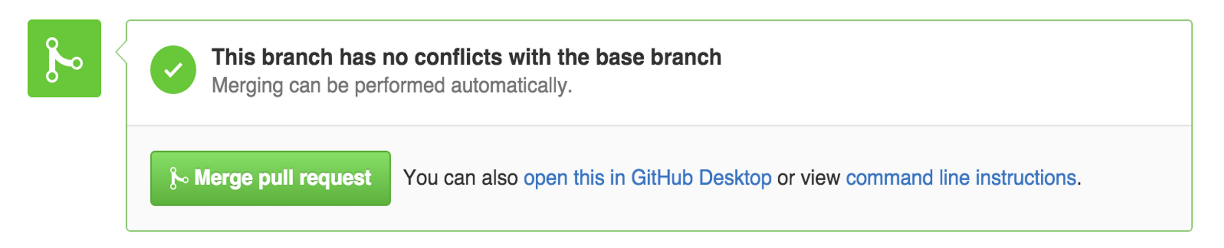
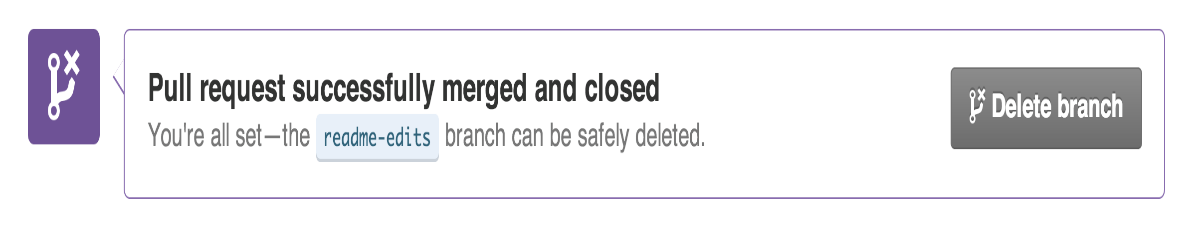
When you’re done with your message, click **Create pull request**!

**Tip**: You can use [emoji](https://help.github.com/articles/basic-writing-and-formatting-syntax/#using-emoji) and [drag and drop images and gifs](https://help.github.com/articles/file-attachments-on-issues-and-pull-requests/) onto comments and Pull Requests.

## Step 5. Merge your Pull Request

In this final step, it’s time to bring your changes together – merging your readme-edits branch into the main branch.

1. Click the green **Merge pull request** button to merge the changes into main.
2. Click **Confirm merge**.
3. Go ahead and delete the branch, since its changes have been incorporated, with the **Delete branch** button in the purple box.

echo "# Cognizantres" >> README.md

git init

git add README.md

git commit -m "first commit"

git branch -M main

git remote add origin https://github.com/pavitrapatil2017/Cognizantres.git

git push -u origin main

### …or push an existing repository from the command line

git remote add origin https://github.com/pavitrapatil2017/Cognizantres.git

git branch -M main

git push -u origin main