**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.
* **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.
* **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.
* **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.

* **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.
* **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 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.

* **Install Git on Windows**
* **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>.

**Install Git**

* **Step2**

Click on the downloaded installer file and select **yes** to continue. After the selecting **yes** the installation begins.

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**.

* It facilitates with three features.
* Create New Repository
* Clone Existing Repository
* Open Existing Repository
* **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.

* **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.

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

**Setting email id**

The Git uses this email id for each commit.

$ 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.

$ 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.

$ git config -list

This command will list all your settings.

**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.

* **Basic Git Commands**
* **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**

$ git config --global user.name "ImDwivedi1"

$ git config --global user.email "Himanshudubey481@gmail.com"

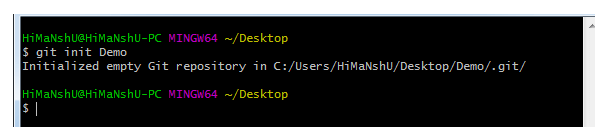
* **Git Init command**

This command is used to create a local repository.

**Syntax**

$ git init Demo

The init command will initialize an empty repository. See the below screenshot.

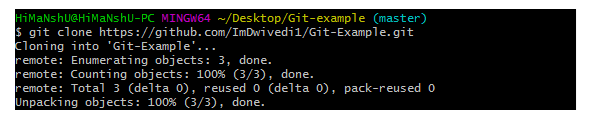


* **Git clone command**

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**

$ 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

$ git add Filename

To add more than one file

$ 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**

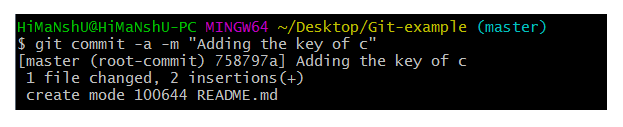
$ git commit -m " Commit Message"

**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**

$ git commit -a

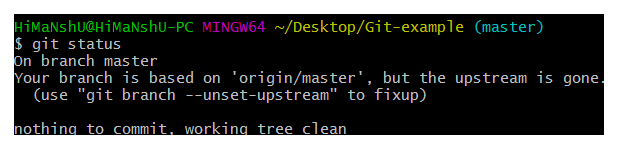


* **Git status command**

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**

$ 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.

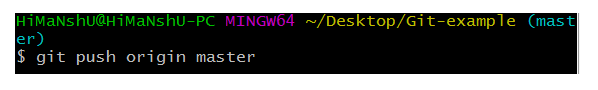
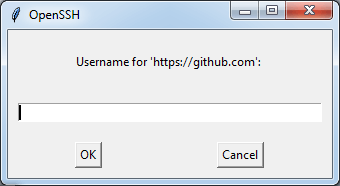
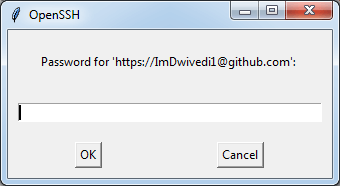
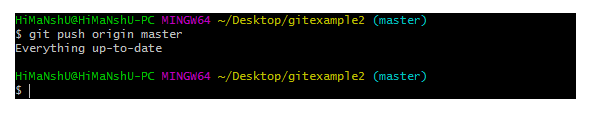
Git push command can be used as follows.

* **Git push origin master**

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

**Syntax**

$ git push [variable name] master

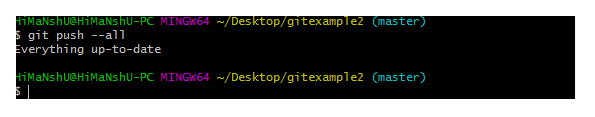
  
  
  


* **Git push -all**

This command pushes all the branches to the server repository.

**Syntax**

$ git push --all

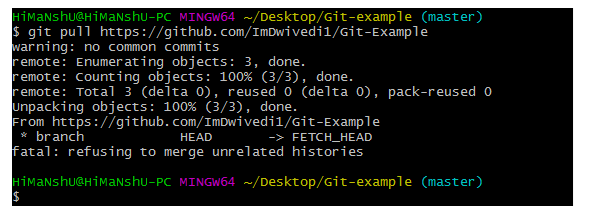


* **Git pull command**

Pull command is used to receive data from GitHub. It fetches and merges changes on the remote server to your working directory.

**Syntax**

$ git pull URL

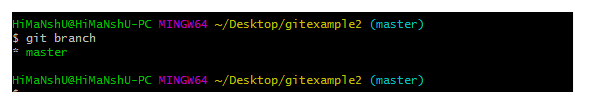


* **Git Branch Command**

This command lists all the branches available in the repository.

**Syntax**

$ git branch



* **Git Merge Command**

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

**Syntax**

$ git merge BranchName

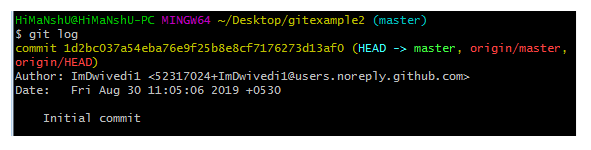


* **Git log Command**

This command is used to check the commit history.

**Syntax**

$ git log

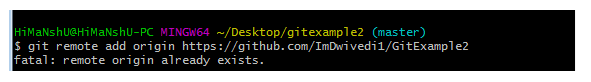


By default, if no argument passed, Git log shows the most recent commits first. We can limit the number of log entries displayed by passing a number as an option, such as -3 to show only the last three entries.

$ 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.



* **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.

The files being removed must be ideal for the branch to remove. No updates to their contents can be staged in the index. Otherwise, the removing process can be complex, and sometimes it will not happen. But it can be done forcefully by **-f** option.

* **The git rm command**

The git rm command is used to remove the files from the working tree and the index.

If we want to remove the file from our repository. Then it can be done by the git rm command. Let's take a file say newfile.txt to test the rm command. The git rm command will be operated as:

$ git rm <file Name>

The above command will remove the file from the Git and repository. The git rm command removes the file not only from the repository but also from the staging area. If we check the status of the repository, then it will show as deleted. Consider the below output:

