

Git and GitHub

➤ Version Control System

Version Control System (VCS) is a software that helps software developers to work together and maintain a complete history of their work.

Listed below are the functions of a VCS –

- Allows developers to work simultaneously.
- Does not allow overwriting each other's changes.
- Maintains a history of every version.

Following are the types of VCS –

- Centralized version control system (CVCS).
- Distributed/Decentralized version control system (DVCS).

Git falls under distributed version control system.

➤ Distributed Version Control System

- Centralized version control system (CVCS) uses a central server to store all files and enables team collaboration.
- But the major drawback of CVCS is its single point of failure, i.e., failure of the central server. Unfortunately, if the central server goes down for an hour, then during that hour, no one can collaborate at all. And even in a worst case, if the disk of the central server gets corrupted and proper backup has not been taken, then you will lose the entire history of the project. Here, distributed version control system (DVCS) comes into picture.
- DVCS clients not only check out the latest snapshot of the directory but they also fully mirror the repository. If the server goes down, then the repository from any client can be copied back to the server to restore it.
- Every checkout is a full backup of the repository. Git does not rely on the central server and that is why you can perform many operations when you are offline.
- You can commit changes, create branches, view logs, and perform other operations when you are offline. You require network connection only to publish your changes and take the latest changes.

➤ Git

- Git is a distributed version-control system for tracking changes in source code during software development.
- It is designed for coordinating work among programmers, but it can be used to track changes in any set of files. Its goals include speed, data integrity, and support for distributed, non-linear workflows.
- Git was created by Linus Torvalds in 2005 for development of the Linux kernel, with other kernel developers contributing to its initial development.
- As with most other distributed version-control systems, and unlike most client-server system, every Git directory on every computer is full-fledged and full version-tracking abilities, independent of network access or a central server.
- Git's design was inspired by BitKeeper and Monotone. Git was originally designed as a low-level version-control system engine.

Advantages:

- **Performance**

- ✓ Git performs very strongly and reliably when compared to other version control systems.
- ✓ New code changes can be easily committed, version branches can be effortlessly compared and merged, and code can also be optimized to perform better.
- ✓ Algorithms used in developing Git take the full advantage of the deep knowledge stored within, with regards to the attributes used to create real source code file trees, how files are modified over time and what kind of file access patterns are used to recall code files as and when needed by developers.
- ✓ Git primarily focus upon the file content itself rather than file names while determining the storage and file version history. Object formats of Git repository files use several combinations of delta encoding and compression techniques to store metadata objects and directory contents.

- **Security**

- ✓ Git is designed specially to maintain the integrity of source code.
- ✓ File contents as well as the relationship between file and directories, tags, commits, versions etc. are secured cryptographically using an algorithm called SHA1 which protects the code and change history against accidental as well as malicious damage.

- **Flexibility**

- ✓ A key design objective of Git is the kind of flexibility it offers to support several kinds of nonlinear development workflows and its efficiency in handling both small scale and large scale projects as well as protocols.
- ✓ It is uniquely designed to support tagging and branching operations and store each and every activity carried out by the user as an integral part of "change" history. Not all VCSs support this feature.

- **Wide acceptance**
 - ✓ Git offers the type of performance, functionality, security and flexibility that most developers and teams need to develop their projects.
 - ✓ When compared to other VCS (version control system) Git is most widely accepted system owing to its universally accepted usability and performance standards.
- **Quality open source project**
 - ✓ Git is a widely supported open source project with over ten years of operational history. People maintaining the project are very well matured and possess a long term vision to meet the long term needs of users by releasing staged upgrades at regular intervals of time to improve functionality as well as usability.
 - ✓ Quality of open source software made available on Git is heavily scrutinized a countless number of times and businesses today depend heavily on Git code quality.
- **Distributed**
 - ✓ Git is distributed in nature. Distributed means that the repository or the complete code base is mirrored onto the developers system so that he can work on it only.
- **Implicit backup**
 - ✓ The chances of losing data are very rare when there are multiple copies of it. Data present on any client side mirrors the repository, hence it can be used in the event of a crash or disk corruption.
- **Easier branching**
 - ✓ CVCS uses cheap copy mechanism, if we create a new branch, it will copy all the codes to the new branch, so it is time-consuming and not efficient. Also, deletion and merging of branches in CVCS is complicated and time-consuming. But branch management with Git is very simple. It takes only a few seconds to create, delete, and merge branches.

➤ DVCS Terminologies

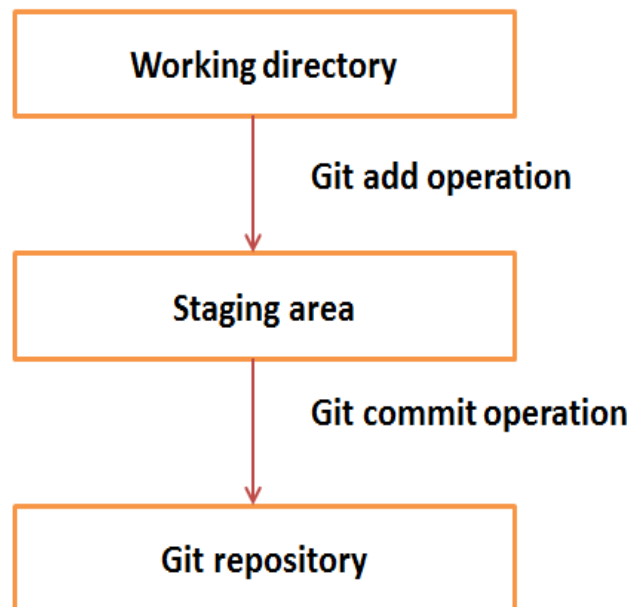
- **Local Repository**
 - ✓ Every VCS tool provides a private workplace as a working copy.
 - ✓ Developers make changes in their private workplace and after commit, these changes become a part of the repository.
 - ✓ Git takes it one step further by providing them a private copy of the whole repository. Users can perform many operations with this repository such as add file, remove file, rename file, move file, commit changes, and many more.
- **Working Directory and Staging Area or Index**
 - ✓ The working directory is the place where files are checked out. In other CVCS, developers generally make modifications and commit their changes directly to the repository. But Git uses a different strategy.
 - ✓ Git doesn't track each and every modified file. Whenever you do commit an operation, Git looks for the files present in the staging area. Only those files present in the staging area are considered for commit and not all the modified files.

Let us see the basic workflow of Git.

Step 1 – Modify a file from the working directory.

Step 2 – Add these files to the staging area.

Step 3 – Perform commit operation that moves the files from the staging area. After push operation, it stores the changes permanently to the Git repository.



Suppose you modified two files, namely “sort.c” and “search.c” and you want two different commits for each operation. You can add one file in the staging area and do commit. After the first commit, repeat the same procedure for another file.

First commit

```
[bash]$ git add sort.c
```

adds file to the staging area

```
[bash]$ git commit -m “Added sort operation”
```

Second commit

```
[bash]$ git add search.c
```

adds file to the staging area

```
[bash]$ git commit -m “Added search operation”
```

➤ **Blobs**

- ✓ Blob stands for Binary Large Object. Each version of a file is represented by blob.
- ✓ A blob holds the file data but doesn’t contain any metadata about the file. It is a binary file, and in Git database, it is named as SHA1 hash of that file.
- ✓ In Git, files are not addressed by names. Everything is content-addressed.

➤ **Trees**

- ✓ Tree is an object, which represents a directory. It holds blobs as well as other sub-directories.
- ✓ A tree is a binary file that stores references to blobs and trees which are also named as SHA1 hash of the tree object.

➤ **Commits**

- ✓ Commit holds the current state of the repository.
- ✓ A commit is also named by SHA1 hash. You can consider a commit object as a node of the linked list.
- ✓ Every commit object has a pointer to the parent commit object. From a given commit, you can traverse back by looking at the parent pointer to view the history of the commit.
- ✓ If a commit has multiple parent commits, then that particular commit has been created by merging two branches.

➤ **Branches**

- ✓ Branches are used to create another line of development. By default, Git has a master branch, which is same as trunk in Subversion.
- ✓ Usually, a branch is created to work on a new feature. Once the feature is completed, it is merged back with the master branch and we delete the branch.
- ✓ Every branch is referenced by HEAD, which points to the latest commit in the branch. Whenever you make a commit, HEAD is updated with the latest commit.

➤ **Tags**

- ✓ Tag assigns a meaningful name with a specific version in the repository.
- ✓ Tags are very similar to branches, but the difference is that tags are immutable. It means, tag is a branch, which nobody intends to modify.
- ✓ Once a tag is created for a particular commit, even if you create a new commit, it will not be updated. Usually, developers create tags for product releases.

➤ **Clone**

- ✓ Clone operation creates the instance of the repository.
- ✓ Clone operation not only checks out the working copy, but it also mirrors the complete repository. Users can perform many operations with this local repository. The only time networking gets involved is when the repository instances are being synchronized.

➤ **Pull**

- ✓ Pull operation copies the changes from a remote repository instance to a local one.
- ✓ The pull operation is used for synchronization between two repository instances.
- ✓ This is same as the update operation in Subversion.

➤ **Push**

- ✓ Push operation copies changes from a local repository instance to a remote one.
- ✓ This is used to store the changes permanently into the Git repository. This is same as the commit operation in Subversion.

➤ **HEAD**

- ✓ HEAD is a pointer, which always points to the latest commit in the branch.
- ✓ Whenever you make a commit, HEAD is updated with the latest commit. The heads of the branches are stored in `.git/refs/heads/` directory.

```
[CentOS]$ ls -l .git/refs/heads/  
master
```

```
[CentOS]$ cat .git/refs/heads/master  
570837e7d58fa4bccd86cb575d884502188b0c49
```

➤ **Revision**

- ✓ Revision represents the version of the source code. Revisions in Git are represented by commits. These commits are identified by SHA1 secure hashes.

➤ **URL**

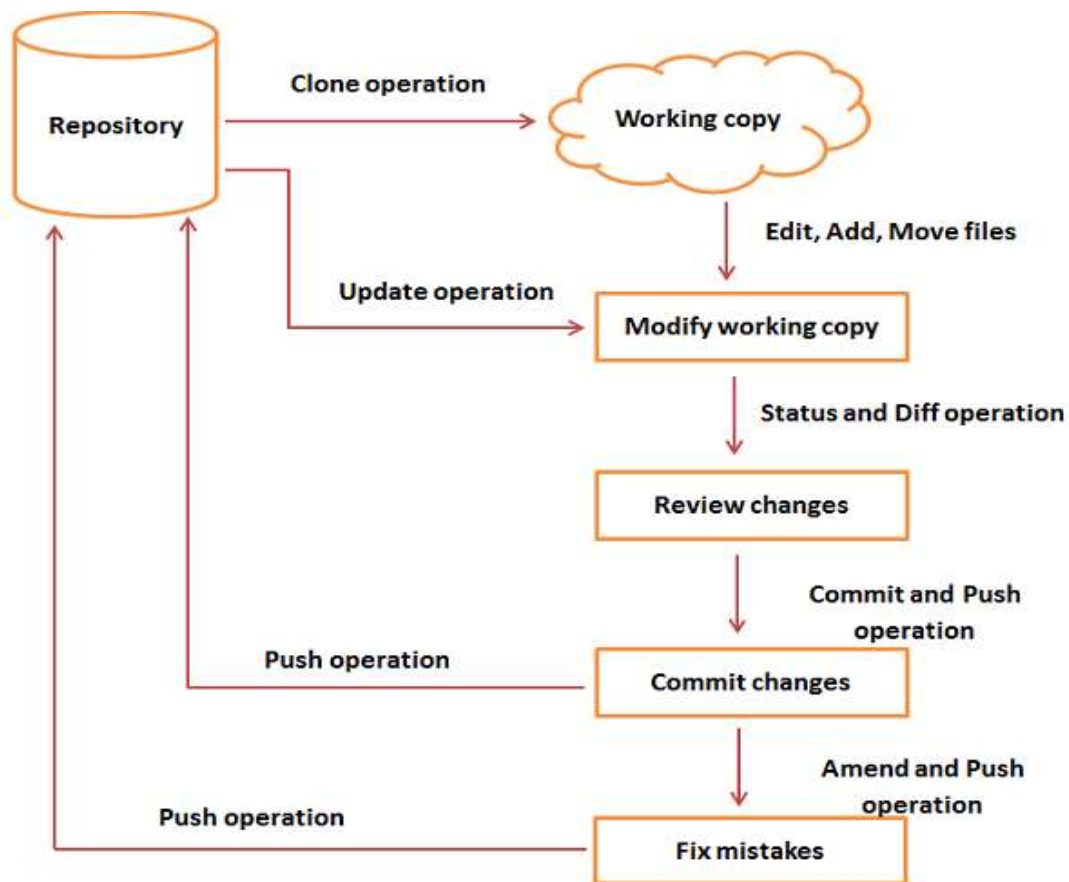
- ✓ URL represents the location of the Git repository. Git URL is stored in config file.

Git Cycle

General workflow is as follows –

- You clone the Git repository as a working copy.
- You modify the working copy by adding/editing files.
- If necessary, you also update the working copy by taking other developer's changes.
- You review the changes before commit.
- You commit changes. If everything is fine, then you push the changes to the repository.
- After committing, if you realize something is wrong, then you correct the last commit and push the changes to the repository.

- Shown below is the pictorial representation of the work-flow.



Commands:

1. **Init :** This command is used to create a new repository.
\$ git init
2. **Add:** This command is used to add files in index.
\$ git add file.txt
3. **Clone:** This command is used for repository checking out purposes.
\$ git clone /path/to/repository
4. **Commit:** This command is used to commit changes to the head.
\$ git commit -m "Added program file"
5. **Status:** This command displays the list of changed files along with the files that are yet to be added or committed.
\$ git status
6. **Push:** This command sends the made changes to the master branch of the remote repository associated with the working directory.
\$ git push origin master
7. **Pull:** This command is used to merge all the changes present on the remote repository to the local working directory.
\$ git pull
8. **Checkout:** This command is used to create branches or to switch between them.
\$ git checkout -b <branch-name>
9. **Show :** This command is used to view information about any git object.
\$ git show

10. Remote: This command allows user to connect the local repository to a remote server.

```
$ git remote -v
```

```
$ git remote add origin
```

11. Merge : This command is used to merge a branch into active branch.

```
$ git merge <branch-name>
```

12. Reset: This command is used to reset index and working directory to last commit's state.

```
$ git reset --hard HEAD
```

13. Rm: This command is used to remove files from the index and working directory.

```
$ git rm file.txt
```

➤ **GitHub**

GitHub is an open-source repository hosting service, sort of like a cloud for code. It hosts your source code projects in a variety of different programming languages and keeps track of the various changes made to every iteration. The service is able to do this by using git, a revision control system that runs in the command line interface.

Other sources are similar to GitHub—including BitBucket, Microsoft Team Foundation Server, and more—but the sheer size of the community should be important to you if you want as many people as possible to see your project. As of 2018, GitHub reported having more than 28 million users, significantly more than its competitors.

The other differences involve cost. GitHub offers private repositories only at an additional cost, while a few of the other services offer private repositories for free. However, these typically come with limited storage and bandwidth.

Using GitHub makes it easier to collaborate with colleagues and peers and look back at previous versions of your work. If you aren't already using GitHub for your coding projects, here are a few reasons to consider doing so.

- **Have Your Code Reviewed by the Community**

Your project is a skeleton. It does what you want it to do, but you're not always sure how the wider population will implement it—or if it even works for everyone.

Fortunately, when you post your project on GitHub, the wider community of programmers and hobbyists can download and evaluate your work. They can give you a heads-up on possible issues such as conflicts or unforeseen dependency issues.

- **GitHub Is a Repository**

Because GitHub is a repository, it allows your work to get out in front of the public. Moreover, it is one of the largest coding communities around, so using it can provide wide exposure for your project and for you. The more people you have to review your project, the more attention and use it is likely to attract.

- **Collaborate and Track Changes in Your Code across Versions**

As when using Microsoft Word or Google Drive, you can have a version history of your code so that is not lost with every iteration. GitHub also tracks changes in a changelog, so you can know exactly what is changed each time. This feature is especially helpful for looking back in time and quickly identifying changes a collaborator made.

- **Use Multiple Integration Options**

GitHub can integrate with common platforms such as Amazon and Google Cloud, as well as services such as Code Climate to track your feedback, and it can highlight syntax in more than 200 different programming languages.

- **Follow the Open-Source Trend**

Many companies and organizations, large and small, are moving to open-source solutions. EnterpriseDB offers Postgres, an open-source database, and states on its website that open-source technologies allow for greater flexibility in a constantly changing environment like technology. It cites the Department of Defence and the Consumer Financial Protection Bureau as large U.S. agencies that have moved to open source in order to respond quickly to technology changes.

Open-source projects tend to be more flexible because they respond more rapidly to market demands. Closed-source programs might exist in a bubble while trying to convince a target market of its value as opposed to being genuinely responsive. GitHub provides a community where programmers are constantly working to solve current problems and making solutions available to the public.

- **Find Talent for Your Organization**

Because of the breadth of the GitHub community, you can sometimes find programmers working on similar projects or who have skills, experiences, or a vision that offers a good fit for your organization. By being a part of the community, you can identify these people, work with them, and possibly even bring them on board to work for you.

- **Develop and Implement a Management Strategy**

You likely have multiple people working on projects at the same time, and many of them may be in different locations and possibly even different countries. With the ability to collaborate on a project through GitHub, you can establish a system for different collaborators to work together without stepping on each other's toes.

➤ The “Hub” in GitHub

Git is a command-line tool, but the center around which all things involving Git revolve is the hub—GitHub.com—where developers store their projects and network with like-minded people.

1. Repository

A repository (usually abbreviated to “repo”) is a location where all the files for a particular project are stored. Each project has its own repo, and you can access it with a unique URL.

2. Forking a Repo

“Forking” is when you create a new project based off of another project that already exists. This is an amazing feature that vastly encourages the further development of programs and other projects. If you find a project on GitHub that you’d like to contribute to, you can fork the repo, make the changes you’d like, and release the revised project as a new repo. If the original repository that you forked to create your new project gets updated, you can easily add those updates to your current fork.

3. Pull Requests

You’ve forked a repository, made a great revision to the project, and want it to be recognized by the original developers—maybe even included in the official project/repository. You can do so by creating a pull request. The authors of the original repository can see your work, and then choose whether or not to accept it into the official project. Whenever you issue a pull request, GitHub provides a perfect medium for you and the main project’s maintainer to communicate.

4. Social networking

The social networking aspect of GitHub is probably its most powerful feature, allowing projects to grow more than just about any of the other features offered. Each user on GitHub has their own profile that acts like a resume of sorts, showing your past work and contributions to other projects via pull requests.



Project revisions can be discussed publicly, so a mass of experts can contribute knowledge and collaborate to advance a project forward. Before the advent of GitHub, developers interested in contributing to a project would usually need to find some means of contacting the authors—probably by email—and then convince them that they can be trusted and their contribution is legit.

5. Changelogs

When multiple people collaborate on a project, it’s hard to keep track revisions—who changed what, when, and where those files are stored. GitHub takes care of this problem by keeping track of all the changes that have been pushed to the repository.

▲ Git Vs GitHub

- Git is a distributed version control tool that can manage a development project's source code history, while GitHub is a cloud based platform built around the Git tool.
- Git is a tool a developer installs locally on their computer, while GitHub is an online service that stores code pushed to it from computers running the Git tool.
- The key difference between Git and GitHub is that Git is an open-source tool developers install locally to manage source code, while GitHub is an online service to which developers who use Git can connect and upload or download resources.
- One way to examine the differences between GitHub and Git is to look at their competitors. Git competes with centralized and distributed version control tools such as Subversion, Mercurial, ClearCase and IBM's Rational Team Concert. On the other hand, GitHub competes with cloud-based SaaS and PaaS offerings, such as GitLab and Atlassian's Bitbucket.

	
1. It is a software	1. It is a service
2. It is installed locally on the system	2. It is hosted on Web
3. It is a command line tool	3. It provides a graphical interface
4. It is a tool to manage different versions of edits, made to files in a git repository	4. It is a space to upload a copy of the Git repository
5. It provides functionalities like Version Control System Source Code Management	5. It provides functionalities of Git like VCS, Source Code Management as well as adding few of its own features