Git - Basic Concepts

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).

In this chapter, we will concentrate only on distributed version control system and especially on Git. 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.

Advantages of Git

Free and open source

Git is released under GPL’s open source license. It is available freely over the internet. You can use Git to manage property projects without paying a single penny. As it is an open source, you can download its source code and also perform changes according to your requirements.

Fast and small

As most of the operations are performed locally, it gives a huge benefit in terms of speed. Git does not rely on the central server; that is why, there is no need to interact with the remote server for every operation. The core part of Git is written in C, which avoids runtime overheads associated with other high-level languages. Though Git mirrors entire repository, the size of the data on the client side is small. This illustrates the efficiency of Git at compressing and storing data on the client side.

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.

Security

Git uses a common cryptographic hash function called secure hash function (SHA1), to name and identify objects within its database. Every file and commit is check-summed and retrieved by its checksum at the time of checkout. It implies that, it is impossible to change file, date, and commit message and any other data from the Git database without knowing Git.

No need of powerful hardware

In case of CVCS, the central server needs to be powerful enough to serve requests of the entire team. For smaller teams, it is not an issue, but as the team size grows, the hardware limitations of the server can be a performance bottleneck. In case of DVCS, developers don’t interact with the server unless they need to push or pull changes. All the heavy lifting happens on the client side, so the server hardware can be very simple indeed.

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** − You modify a file from the working directory.

**Step 2** − You add these files to the staging area.

**Step 3** − You 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 **B**inary **L**arge **Ob**ject. 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 -1 .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.

[tom@CentOS tom\_repo]$ pwd

/home/tom/tom\_repo

[tom@CentOS tom\_repo]$ cat .git/config

[core]

repositoryformatversion = 0

filemode = true

bare = false

logallrefupdates = true

[remote "origin"]

url = gituser@git.server.com:project.git

fetch = +refs/heads/\*:refs/remotes/origin/\*

<https://github.blog/2022-02-02-build-ci-cd-pipeline-github-actions-four-steps/>

## What is a CI/CD pipeline?

A CI/CD pipeline automates the process of software delivery. It builds code, runs tests, and helps you to safely deploy a new version of the software. CI/CD pipeline reduces manual errors, provides feedback to developers, and allows fast product iterations.

CI/CD pipeline introduces automation and continuous monitoring throughout the lifecycle of a software product. It involves from the integration and testing phase to delivery and deployment. These connected practices are referred as CI/CD pipeline.

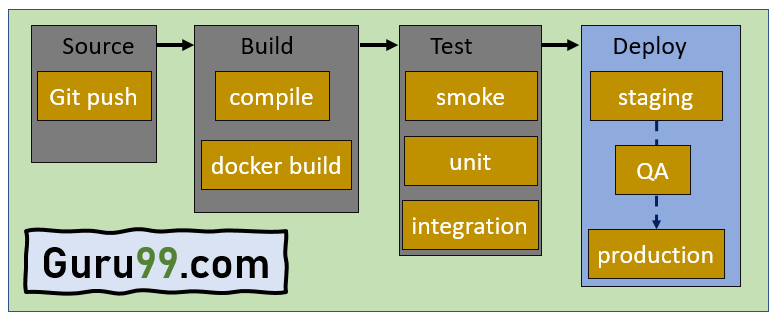
## What is Continuous Integration, Continuous Delivery, and Continuous Deployment?

* **Continuous integration** is a software development method where members of the team can integrate their work at least once a day. In this method, every integration is checked by an automated build to search the error.
* **Continuous delivery** is a software engineering method in which a team develops software products in a short cycle. It ensures that software can be easily released at any time.
* **Continuous deployment** is a software engineering process in which product functionalities are delivered using automatic deployment. It helps testers to validate whether the codebase changes are correct, and it is stable or not.

## Stages of a CI/CD pipeline

A CI/CD pipeline is a runnable specification of the steps that any developer should perform to deliver a new version of any software. Failure in each and every stage triggers a notification via email, Slack, or other communication platforms. It enables responsible developers to know about the important issues.

Here are the important Stages of CI/CD pipeline:

Stages of CI/CD pipeline

### Source Stage

In the source stage, CI/CD pipeline is triggered by a code repository. Any change in the program triggers a notification to the CI/CD tool that runs an equivalent pipeline. Other common triggers include user-initiated workflows, automated schedules, and the results of other pipelines.

### Build Stage

This is the second stage of the CI/CD Pipeline in which you merge the source code and its dependencies. It is done mainly to build a runnable instance of software that you can potentially ship to the end-user.

Programs that are written in languages like C++, Java, C, or Go language should be compiled. On the other hand, JavaScript, Python, and Ruby programs can work without the build stage.

Failure to pass the build stage means there is a fundamental project misconfiguration, so it is better that you address such issue immediately.

### Test Stage

Test Stage includes the execution of automated tests to validate the correctness of code and the behaviour of the software. This stage prevents easily reproducible bugs from reaching the clients. It is the responsibility of developers to write automated tests.

### Deploy Stage

This is the last stage where your product goes live. Once the build has successfully passed through all the required test scenarios, it is ready to deploy to live server.

## Example of CI/CD Pipeline

Here is example of CI/CD pipeline:

* **Source Code Control:** Host code on GitHub as a private repository. This will help you to integrate your application with major services and software.
* **Continuous integration:** Use continuous integration and delivery platform CircleCI and commit every code. When the changes notify, this tool will pull the code available in GitHub and process to build and run the test.
* **Deploy code to UAT:**Configure CircleCI to deploy your code to AWS UAT server.
* **Deploy to production:**You have to reuse continuous integration steps for deploying code to UAT.

## CI/CD pipeline Best Practices

Here is a CI/CD pipeline best practices:

* Write up the current development process therefore, you can know the procedures that require to change and one that can be easily automated.
* Start off with a small proof of project before going ahead and complete whole development process at once.
* Set up a pipeline with more than one stage in which fast fundamental tests run first.
* Start each workflow from the same, clean, and isolated environment.
* Run open source tools that cover everything from code style to security scanning.
* Setup a better code hub to continuously check the quality of your code by running the standard set of tests against every branch.
* Peer code review each pull request to solve a problem in a collaborative manner.
* You have to define success metrics before you start the transition to CD automation. This will help you to consistently analyze your software, developing progress help refining where needed.

## Advantages of CI/CD pipelines

Here are the pros/ benefits of CI/CD Pipeline:

* Builds and testing can be easily performed manually.
* It can improve the consistency and quality of code.
* Improves flexibility and has the ability to ship new functionalities.
* CI/CD pipeline can streamline communication.
* It can automate the process of software delivery.
* Helps you to achieve faster customer feedback.
* CI/CD pipeline helps you to increase your product visibility.
* It enables you to remove manual errors.
* Reduces costs and labour.
* CI/CD pipelines can make the software development lifecycle faster.
* It has automated pipeline deployment.
* A CD pipeline gives a rapid feedback loop starting from developer to client.
* Improves communications between organization employees.
* It enables developers to know which changes in the build can turn to the brokerage and to avoid them in the future.
* The automated tests, along with few manual test runs, help to fix any issues that may arise.

https://www.simplilearn.com/tutorials/jenkins-tutorial/jenkins-installation-on-windows