Contents

[Introduction to Source Control Management 2](#_Toc94531081)

[VCS, CVCS, DVCS 2](#_Toc94531082)

[Centralized Version Control System 2](#_Toc94531083)

[Distributed Version Control System 2](#_Toc94531084)

[Basic Difference with Pros and Cons 3](#_Toc94531085)

# Introduction to Source Control Management

## VCS, CVCS, DVCS

* **VCS**: Version Control System.
* **CVCS**: Centralized VCS.
  + Subversion.
* **DVCS**: Distributed VCS.
  + Git (*Written in C*).
  + Mercurial (*Written in Python*).
  + Bazaar (*Written in Python*).

## Centralized Version Control System

In centralized source control, there is a server and a client. The server is the master repository that contains all of the versions of the code. To work on any project, firstly user or client needs to get the code from the master repository or server. So the client communicates with the server and pulls all the code or current version of the code from the server to their local machine. In other terms we can say, you need to take an update from the master repository and then you get the local copy of the code in your system. So once you get the latest version of the code, you start making your own changes in the code and after that, you simply need to commit those changes straight forward into the master repository. Committing a change simply means merging your own code into the master repository or making a new version of the source code. So everything is centralized in this model.

There will be just one repository and that will contain all the history or version of the code and different branches of the code. So the basic workflow involves in the centralized source control is getting the latest version of the code from a central repository that will contain other people’s code as well, making your own changes in the code, and then committing or merging those changes into the central repository.

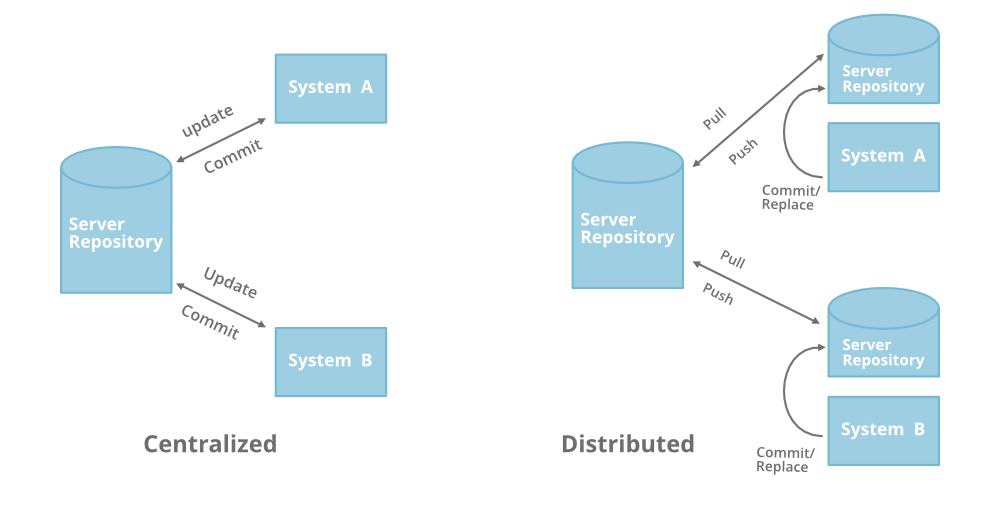
## Distributed Version Control System

In distributed version control most of the mechanism or model applies the same as centralized. The only major difference you will find here is, instead of one single repository which is the server, here every single developer or client has their own server and they will have a copy of the entire history or version of the code and all of its branches in their local server or machine. Basically, every client or user can work locally and disconnected which is more convenient than centralized source control and that’s why it is called distributed.

You don’t need to rely on the central server, you can clone the entire history or copy of the code to your hard drive. So when you start working on a project, you clone the code from the master repository in your own hard drive, then you get the code from your own repository to make changes and after doing changes, you commit your changes to your local repository and at this point, your local repository will have ‘*change sets*‘ but it is still disconnected with the master repository (master repository will have different ‘***sets of changes***‘ from each and every individual developer’s repository), so to communicate with it, you issue a request to the master repository and push your local repository code to the master repository. Getting the new change from a repository is called “**pulling**” and merging your local repository’s ‘set of changes’ is called “**pushing**“.

It doesn’t follow the way of communicating or merging the code straight forward to the master repository after making changes. Firstly you commit all the changes in your own server or repository and then the ‘set of changes’ will merge to the master repository.

Below is the diagram to understand the difference between these two in a better way: 



## Basic Difference with Pros and Cons

* Centralized version control is easier to learn than distributed. If you are a beginner, you’ll have to remember all the commands for all the operations in DVCS and working on DVCS might be confusing initially. CVCS is easy to learn and easy to set up.
* DVCS has the biggest advantage in that it allows you to work offline and gives flexibility. You have the entire history of the code in your own hard drive, so all the changes you will be making in your own server or to your own repository which doesn’t require an internet connection, but this is not in the case of CVCS.
* DVCS is faster than CVCS because you don’t need to communicate with the remote server for each and every command. You do everything locally which gives you the benefit to work faster than CVCS.
* Working on branches is easy in DVCS. Every developer has an entire history of the code in DVCS, so developers can share their changes before merging all the ‘sets of changes to the remote server. In CVCS it’s difficult and time-consuming to work on branches because it requires to communicate with the server directly.
* If the project has a long history or the project contain large binary files, in that case, downloading the entire project in DVCS can take more time and space than usual, whereas in CVCS you just need to get few lines of code because you don’t need to save the entire history or complete project in your own server so there is no requirement for additional space.
* If the main server goes down or it crashes in DVCS, you can still get the backup or entire history of the code from your local repository or server where the full revision of the code is already saved. This is not in the case of CVCS, there is just a single remote server that has entire code history.
* Merge conflicts with other developer’s code are less in DVCS. Because every developer work on their own piece of code. Merge conflicts are more in CVCS in comparison to DVCS.
* In DVCS, sometimes developers take the advantage of having the entire history of the code and they may work for too long in isolation which is not a good thing. This is not in the case of CVCS.