

|  |
| --- |
|  |

**CVCS v/s DVCS**

**guide**

# Table of Contents

[Table of Contents …………………………………………………………………………………………………. 2](#_Toc390877804)

1. [Introduction ……………………………………………………………………………………………………………. 3](#_Toc390877805)

**2.1** [**Centralized Version Control System(CVCS) ………………………………………………………… 3**](#_Toc390877806)

**2.2** [**A Typical Centralized Version Control Workflow…………………………………………….. 4**](#_Toc390877807)

[**2.3 Core Advantages of CVCS……………………………………………………………………………………. 4**](#_Toc390877808)

[**2.4 Core Disadvantages of CVCS………………………………………………………………………………. 5**](#_Toc390877809)

[**3.1 Distributed or Decentralized Version Control System(DVCS)……………………….. 5**](#_Toc390877810)

[**3.2 Core Advantages of DVCS………………………………………………………………………………….**](#_Toc390877811)**. 6**

[**3.3 Core Disadvantages of DVCS…………………………………………………………………………….. 7**](#_Toc390877811)

[**4. Diagrammatic explanation of the difference in the two workflows…………………….. 8**](#_Toc390877810)

[**5.1 Advantages of DVCS over CVCS……………………………………………………………………………… 9**](#_Toc390877810)

[**5.2 Advantages of CVCS over DVCS…………………………………………………………………………. 9**](#_Toc390877811)

[**6.1 Choice of a Version Control System……………………………………………………………………… 10**](#_Toc390877810)

[**6.2 Problems solved by DVCS…………………………………………………………………………………..11**](#_Toc390877811)

[6.3 Scenario of better cure from data loss(SVN v/s Git)………………………………………..11](#_Toc390877811)

[**7. Conclusion………………………………………………………………………………………………………………… 12**](#_Toc390877810)

# 

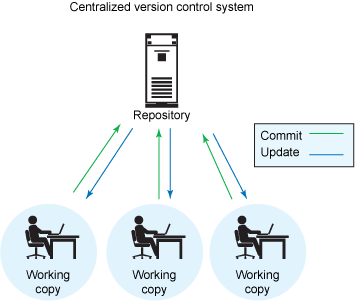
1. **Introduction**

Version Control system

* Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.
* If you screw things up or lose files, you can easily recover.
* Almost all “real” projects use some kind of version control
* Any company with a clue uses some kind of version control
* Companies without a clue are bad places to work!!!

There are many version control systems out there. Often they are divided into two groups: “centralized” and “distributed”.

**2.1 Centralised version control system**



* Centralized version control systems are based on the idea that there is a single “central” copy of your project somewhere (probably on a server), and programmers will “commit” their changes to this central copy.
* “Committing” a change simply means recording the change in the central system. Other programmers can then see this change. They can also pull down the change, and the version control tool will automatically update the contents of any files that were changed.
* Centralized version control solves the problem of saving a full copy of the file even if only a single line changed. Programmers no longer have to keep many copies of files on their hard drives manually, because the version control tool can talk to the central copy and retrieve any version they need on the fly.

Some of the most common centralized version control systems you may have heard of or used are CVS, Subversion (or SVN) and Perforce.

# 2.2 A typical centralised version control workflow

Workflow for adding a new feature or fixing a bug in your project will usually look something like this:

* Pull down any changes other people have made from the central server.
* Make your changes, and make sure they work properly.
* Commit your changes to the central server, so other programmers can see them.

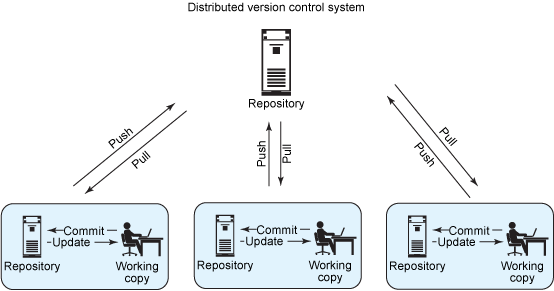
**2.3 Core Advantages of CVCS**

* Relatively easy to set up.
* Provide developers with a clear view (everyone knows at some level what the others do).
* Enable admin to control the workflow (who to do what).

**2.4 Core Disadvantages of CVCS**

* If the main server goes down, the developers can’t save versioned changes (single point of failure).
* Remote commits are slow.
* Unsolicited changes might ruin the development.
* If the hard disk of the central database becomes corrupted, the entire history could be lost.

**3.1 Distributed(or Decentralized) Version Control System(DVCS)**



* These systems do not necessarily rely on a central server to store all the versions of a project’s files. Instead, every developer “clones” a copy of a repository which has the full history of the project on their own hard drive & also the metadata of the original.
* This method may sound wasteful, but in practice, it’s not a problem. Most programming projects consist mostly of plain text files (and maybe a few images), and disk space is so cheap that storing many copies of a file doesn’t create a noticeable dent in a hard drive’s free space. Modern systems also compress the files to use even less space.
* The act of getting new changes from a repository is usually called “pulling,” and the act of moving your own changes to a repository is called “pushing”. In both cases, you move changesets (changes to files groups as coherent wholes), not single-file diffs.
* One common misconception about distributed version control systems is that there cannot be a central project repository. This is simply not true – there is nothing stopping you from saying “this copy of the project is the authoritative one.” This means that instead of a central repository being required by the tools you use, it is now optional and purely a social issue.

Some of the most common centralized version control systems you may have heard of or used are Git, Mercurial, Bazaar, Bitkeeper.

**3.2 Core Advantages of DVCS**

* **Everyone has their own local sandbox.**

No more giant check-ins ; your incremental history is in your repo.

* **DVCS works offline.**

You can happily stay on your local machine, checking in and undoing, no matter if the “server” is down.

* **DVCS is fast.**

Commits and reverts are all done locally & there’s no sketchy network or server to ask for old revisions from a year ago.

* **DVCS handles changes very well.**

Every change has a guid ('Globally Unique Identifier') which makes it easy to track.

* **Branching and merging is easy.**

Because every developer “has their own branch”, every shared change is like reverse integration.

The guids make it easy to automatically combine changes and avoid duplicates.

* **With DVCS, there is less management.**

DVCS systems are easy to get running since there is no “always-running” server software to install.

Also, DVCS systems may not require you to “add” new users since you can just pick what URLs to pull from.

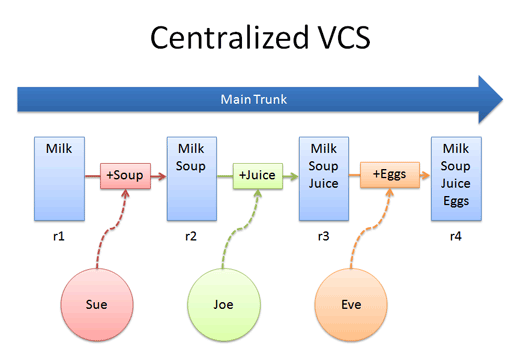
**3.3 Core Disadvantages of DVCS**

* **You still need a backup.**
* **There’s not really a “latest version”.**

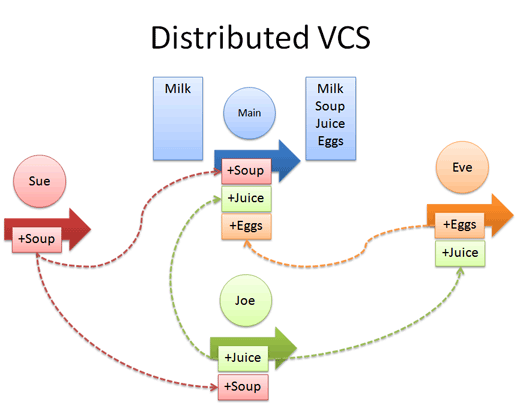
If there’s no central location, you don’t immediately know where to see for the latest version.

* **There aren’t really revision numbers.**

Instead, people refer to change numbers which are not intuitive. But, you can tag releases with meaningful names.



**4. Diagrammatic explanation of the difference in the two workflows**



**5.1 Advantages of DVCS over CVCS**

The act of cloning an entire repository gives distributed version control tools several advantages over centralized systems:

* Performing actions other than pushing and pulling changesets is extremely fast because the tool only needs to access the hard drive, not a remote server.
* Committing new changesets can be done locally without anyone else seeing them. Once you have a group of changesets ready, you can push all of them at once.
* Everything but pushing and pulling can be done without an internet connection. So you can work on a plane, and you won’t be forced to commit several bugfixes as one big changeset.
* Since each programmer has a full copy of the project repository, they can share changes with one or two other people at a time if they want to get some feedback before showing the changes to everyone.

**5.2 Advantages of CVCS over DVCS**

To be quite honest, there are almost no disadvantages to using a distributed version control system over a centralized one. Distributed systems do not prevent you from having a single “central” repository, they just provide more options on top of that.

There are only two major inherent disadvantages to using a distributed system:

* If your project contains many large, binary files that cannot be easily compressed, the space needed to store all versions of these files can accumulate quickly.
* If your project has a very long history (50,000 changesets or more), downloading the entire history can take an impractical amount of time and disk space.

The authors and contributors of modern distributed version control systems are working on solving these problems, but at the moment, no bundled, built-in features solve them.

**6.1 Choice of Version Control System**

In due course of our evaluation, we found that subversion(cvcs) is versatile and a feature rich tool which user friendly tools and interfaces. This would be an automatic choice for project development teams which work in a closely knit work environment. It has a small learning curve and a new user can be brought up to speed very quickly. But on the down side, if the project involves a lot of Merging and Conflict in future, it might be difficult to manage them when volume increases.

In this case, Git has an upper hand. It has a lot of handy features, very effective in exposing merge conflicts and managing multiple parallel versions. Although it has a steep learning curve, it is very versatile and works well in a dynamic project environment. Open Source projects often look towards Git as an effective VCS due to its versatility. Distributed approach outweighs the pros and outdoes the cons of Centralized approach.

**6.2 Problems solved by DVCS**

|  |  |
| --- | --- |
| Problem in cvcs | Solved by dvcs |
| Single point of failure. | Each client connected to remote repository is a version control system and can perform very well as a server in case of failure of the remote server. |
| Remote commits are slow. | Introduced concept of local commits: no network, wicked fast. |
| Unsolicited changes that may break your build. | Developer pushes their changes continuously.  Integrators review the code changes and can bring about changes to the reference code asynchronously. |
| Merging is painful. | Merging is breeze. |

**6.3 Scenario of data-loss(SVN v/s Git)**

In case a user’s hard drive crashes before he pushes his changes, that danger exists with a centralized system too.However, the centralized system has a far worse disaster scenario. What if that “IT-guaranteed backup” of your subversion server you’ve been so confident in turns out to be corrupted? What if the backup job was modified by some guy who was late for his daughter’s recital and didn’t quite get it done right?

With a DVCS, you have no loss of data whatsoever. Just figure out who has the most recent commit – they have a COMPLETE copy of the repository. Within minutes you can have another centralized repository setup with absolutely no loss of data.

But you say – the same thing is true of subversion! We all have a copy of the project on our hard drive.No, you don’t. You all have a copy of one VERSION of your application. It may not even be working or complete. And all those previous versions? Log entries? Metadata? Gone. Unless your backup is good, you’re hosed.

If fault-tolerance is crucial to your organization, your safest bet is DVCS with a backed-up shared repository. That way, you’re completely covered even if a backup should fail.

## Version control systems aim to solve a specific problem that programmers face: “storing and sharing multiple versions of code files.” If you’re a programmer of any kind and you don’t use any kind of version control, you should start right now. It will make your life easier.

**7. Conclusion**

The real choice between using one type or the other is organizational -- if your project or organization wants centralized control, then a DVCS is a non-starter. If your developers are expected to work all over the country/world, without secure broadband connections to a central repository, then DVCS is probably your salvation.