Comparison between Centralized and Distributed Version Control Systems [closed]

Asked 16 years, 3 months ago Modified 7 years, 6 months ago

Viewed 69k times

79

As it currently stands, this question is not a good fit for our Q&A format. We expect answers to be supported by facts, references, or expertise, but this question will likely solicit debate, arguments, polling, or extended discussion. If you feel that this question can be improved and possibly reopened, visit the help center for guidance.

Closed 12 years ago.

What are the **benefits and drawbacks** with using **Centralized versus Distributed** Version Control Systems (DVCS)? Have you run into any problems in DVCS and how did you safeguard against these problems? *Keep the discussion tool agnostic and flaming to minimum.*

For those wondering what DVCS tools are available, here is a list of the best known free/open source DVCSs:

- Git, (written in C) used by the Linux Kernel and Ruby on Rails.

- Mercurial, (written in Python) used by Mozilla and OpenJDK.
- Bazaar, (written in Python) used by Ubuntu developers.
- Darcs, (written in Haskell).

version-control comparison dvcs

Share

Improve this question

Follow

edited Jun 20, 2017 at 9:56

community wiki
6 revs, 4 users 51%
Spoike

11 A lot of answers for not being "constructive" – Catchops Aug 15, 2013 at 14:26

I can say it's been constructive for me. – vercellop Mar 28, 2019 at 19:00

1 “For those wondering what DVCS tools are available”—Why not list the centralized ones? – Guildenstern May 13, 2023 at 18:17

15 Answers

Sorted by: Highest score (default)

From my answer to a different question:

60

Distributed version control systems (DVCSs) solve different problems than Centralized VCSs. Comparing them is like comparing hammers and screwdrivers.

Centralized VCS systems are designed with the intent that there is One True Source that is Blessed, and therefore Good. All developers work (checkout) from that source, and then add (commit) their changes, which then become similarly Blessed. The only real difference between CVS, Subversion, ClearCase, Perforce, VisualSourceSafe and all the other CVCSes is in the workflow, performance, and integration that each product offers.

Distributed VCS systems are designed with the intent that one repository is as good as any other, and that merges from one repository to another are just another form of communication. Any semantic value as to which repository should be trusted is imposed from the outside by process, not by the software itself.

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. If you need both, you're fsck'd.

Share Improve this answer Follow

edited May 23, 2017 at 12:18
Community Bot
1 ● 1

answered Sep 21, 2008 at 13:55
Craig Trader
15.7k ● 6 ● 40 ● 56

13 You can have a DVCS locally to manage your personal branches and stuff and later convert (export) them to the standard centralized repos. Many people is finding that a very comfortable working model, where you are forced to use centralized. – Vinko Vrsalovic Jan 7, 2009 at 9:56

4 Distribution version control systems are a clean superset of centralized ones. If you want the centralized control you get with a centralized version control system you can have that with a distributed one. In fact, I would argue that it's even easier to exercise because of the way most distributed version control systems allow you to work with changesets as a discreet entity that you can add or remove at will. – Omnifarious Oct 28, 2010 at 15:37

4 I am with Omnifarious, your answer could be misleading. In particular the statement "if your project or organization wants centralized control, then a DVCS is a non-starter" only makes sense if read alongside your clarifying comment. – Colin Jack Jun 28, 2011 at 19:44

3 Disagreed. Using git, for example, with a single bare/OneTrue repo is just hard work, and is completely against the git philosophy. Other programs do this much better. If your organisation wants to keep a single centralised

master repo, git does *not* provide a usable superset of, for example, svn. And, if you want a central repo, and you give your devs git, you can be absolutely certain that they'll screw things up for you. Been there. – EML May 8, 2017 at 14:11

2 @EML I've worked in an environment that used git and they liked to pretend it was centralized and it created huuuge headaches with merges. I've also seen git done right. I am now in an environment that uses central version control and I think switching to git could be a huge learning curve for a lot of people. – Kyle Burkett Aug 7, 2017 at 14:19

47

To those who think distributed systems don't allow authoritative copies please note that there are plenty of places where distributed systems have authoritative copies, the perfect example is probably Linus' kernel tree. Sure lots of people have their own trees but almost all of them flow toward Linus' tree.

That said I use to think that distributed SCM's were only useful for lots of developers doing different things but recently have decided that anything a centralized repository can do a distributed one can do better.

For example, say you are a solo developer working on your own personal project. A centralized repository might be an obvious choice but consider this scenario. You are away from network access (on a plane, at a park, etc) and want to work on your project. You have your

> local copy so you can do work fine but you really want to commit because you have finished one feature and want to move on to another, or you found a bug to fix or whatever. The point is that with a centralized repo you end up either mashing all the changes together and commiting them in a non-logical changeset or you manually split them out later.
>
> With a distributed repo you go on business as usual, commit, move on, when you have net access again you push to your "one true repo" and nothing changed.
>
> Not to mention the other nice thing about distributed repos: full history available always. You need to look at the revision logs when away from the net? You need to annotate the source to see how a bug was introduced? All possible with distributed repos.
>
> Please please don't believe that distributed vs centralized is about ownership or authoritative copies or anything like that. The reality is distributed is the next step in evolution of SCM's.

Share Improve this answer Follow

edited May 25, 2017 at 9:20

Mukesh Y.
949 ● 1 ● 16 ● 36

answered Sep 21, 2008 at 14:32

manicmethod
886 ● 1 ● 8 ● 13

Saying that anything a CVCS can do, a DVCS can do it better is an overstatement. For example, your example is wrong, at least in a sense. Centralized systems, like TFS, *do* support trees in the form of 'code shelves'.
– Nikhil Vandanapu Sep 13, 2019 at 21:14

What does this quote? – Guildenstern May 13, 2023 at 18:30

27

Not really a comparison, but here are what big projects are using:

Centralized VCSes

- **Subversion**

  Apache, GCC, Ruby, MPlayer, Zope, Plone, Xiph, FreeBSD, WebKit, ...

- **CVS**

  CVS

Distributed VCSes

- **git**

  Linux kernel, KDE, Perl, Ruby on Rails, Android, Wine, Fedora, X.org, Mediawiki, Django, VLC, Mono, Gnome, Samba, CUPS, GnuPG, Emacs ELPA...

- **mercurial (hg)**

  Mozilla and Mozdev, OpenJDK (Java), OpenSolaris, ALSA, NTFS-3G, Dovecot, MoinMoin, mutt, PETSc, Octave, FEniCS, Aptitude, Python, XEmacs, Xen, Vim, Xine...

- **bzr**

  Emacs, Apt, Mailman, MySQL, Squid, ... also promoted within Ubuntu.

- **darcs**

  ghc, ion, xmonad, ... popular within Haskell community.

- **fossil**

  SQLite

Share Improve this answer Follow

edited Jun 20, 2017 at 9:55

community wiki
11 revs, 5 users 77%
sastanin

2 This answer needs to be upvoted. – Spoike Jan 11, 2009 at 12:08

6 The task was to "Keep the discussion tool agnostic", so this answer does not really help. – Weidenrinde May 13, 2009 at 9:17

SQLite does *not* use CVS or any other centralized VCS. They use their own distributed VCS fossil-scm.org. – Baruch Dec

25, 2013 at 8:16

Thanks, @baruch. Fixed. The answer was written long time ago and needs to be updated. Many projects have switched (I suspect mostly from Subversion to Git). It's a community wiki, feel free to edit. – sastanin Jan 8, 2014 at 8:53

21

W. Craig Trader said this about DVCS and CVCS:

> If you need both, you're fsck'd.

I wouldn't say you're *fsck'd* when using both. Practically developers who use DVCS tools usually try to merge their changes (or send pull requests) against a central location (usually to a release branch in a release repository). There is some irony with developers who use DVCS but in the end stick with a centralized workflow, you can start to wonder if the Distributed approach really is better than Centralized.

There are some advantages with DVCS over a CVCS:

- The notion of uniquely recognizable commits makes sending patches between peers painless. I.e. you make the patch as a commit, and share it with others developers who need it. Later when everyone wants to merge together, that particular commit is recognized and can be compared between branches, having less chance of merge conflict. Developers tend to send patches to each other by USB stick or e-mail regardless of versioning tool you use.

Unfortunately in the CVCS case, version control will register the commits as seperate, failing to recognize that the changes are the same, leading to a higher chance of merge conflict.

- You can have local experimental branches (cloned repositories can also be considered a branch) that you don't need to show to others. That means, breaking changes don't need to affect developers if you haven't pushed anything upstream. In a CVCS, when you still have a breaking change, you may have to work offline until you've fixed it and commit the changes by then. This approach effectively defeats the purpose of using versioning as a safety net but it is a necessary evil in CVCS.
- In today's world, companies usually work with off-shore developers (or if even better they want to work from home). Having a DVCS helps these kind of projects out because it eliminates the need of a reliable network connection since everyone has their own repo.

…and some disadvantages that usually have workarounds:

- *Who has the latest revision?* In a CVCS, the trunk usually has the latest revision, but in a DVCS it may not be plainly obvious. The workaround is using rules of conduct, that the developers in a project have to come to an agreement in which repo to merge their work against.

- Pessimistic locks, i.e. a file is locked when making a check-out, are usually not possible because of concurrency that may happen between repositories in DVCS. The reason file locking exists in version control is because developers want to avoid merge conflicts. However, locking has the disadvantage of slowing development down as two developers can't work on same piece of code simultaneously as with a long transaction model and it isn't full proof warranty against merge conflicts. The only sane ways regardless of version control is to combat big merge conflicts is to have good code architecture (like low coupling high cohesion) and divide up your work tasks so that they have low impact on the code (which is easier said than done).
- In proprietary projects it would be disastrous if the whole repository becomes publically available. Even more so if a disgruntled or malicious programmer gets hold of a cloned repository. Source code leakage is a severe pain for proprietary businesses. DVCS's makes this plain simple as you only need to clone the repository, while some CM systems (such as ClearCase) tries to restrict that access. However in my opinion, if you have an enough amount of dysfunctionality in your company culture then no version control in the world will help you against source code leakage.

Share Improve this answer Follow

edited May 23, 2017 at 10:31

Community Bot

1 ● 1

answered Sep 21, 2008 at 16:24

Spoike

122k ● 45 ● 142 ● 158

10

During my search for the right SCM, I found the following links to be of great help:

1. Better SCM Initiative : Comparison. Comparison of about 26 version control systems.
2. Comparison of revision control software. Wikipedia article comparing about 38 version control systems covering topics like technical differences, features, user interfaces, and more.
3. Distributed version control systems. Another comparison, but focussed mainly on distributed systems.

Share Improve this answer Follow

edited Sep 21, 2008 at 19:08

answered Sep 21, 2008 at 14:12

Nobby
121 ● 5

Note that information about Git in "Better SCM Initiative : Comparison" isn't entirely correct. Also set of features seems to me geared towards centralized VCS and CVS-like systems. – Jakub Narębski Jul 28, 2009 at 18:51

8

To some extent, the two schemes are equivalent:

- A distributed VCS can trivially emulate a centralised one if you just always push your changes to some designated upstream repository after every local commit.

- A centralised VCS won't usually be able to emulate a distributed one quite as naturally, but you can get something very similar if you use something like quilt on top of it. Quilt, if you're not familiar with it, is a tool for managing large sets of patches on top of some upstream project. The idea here is that the DVCS commit command is implemented by creating a new patch, and the push command is implemented by committing every outstanding patch to the centralised VCS and then discarding the patch files. This sounds a bit awkward, but in practice it actually works rather nicely.

Having said that, there are a couple of things which DVCSes traditionally do very well and which most centralised VCSes make a bit of a hash of. The most important of these is probably branching: a DVCS will make it very easy to branch the repository or to merge branches which are no longer needed, and will keep track of history while you do so. There's no particular reason why a centralised scheme would have trouble with this, but historically nobody seems to have quite gotten it right yet. Whether that's actually a problem for you depends on how you're going to organise development, but for many people it's a significant consideration.

The other posited advantage of DVCSes is that they work offline. I've never really had much use for that; I mostly do development either at the office (so the repository's on the local network) or at home (so there's ADSL). If you do a lot of development on laptops while traveling then this might be more of a consideration for you.

There aren't actually very many gotchas which are specific to DVCSes. There's a slightly greater tendency for people to go quiet, because you can commit without pushing and it's easy to end up polishing things in private, but apart from that we haven't had very many problems. This may be because we have a significant number of open source developers, who are usually familiar with the patch-trading model of development, but incoming closed source developers also seem to pick things up reasonably quickly.

Share Improve this answer Follow

answered Sep 21, 2008 at 20:59

Steven Smith

6

I have been using subversion for many years now and I was really happy with it.

Then the GIT buzz started and I just had to test it. And for me, the main selling point was branching. Oh boy. Now I no longer need to clean my repository, go back a few version or any of the silly things I did when using subversion. Everything is cheap in dvcs. I have only tried fossil and git though, but I have used perforce, cvs and subversion and it looks like dvcs all have really cheap branching and tagging. No longer need to copy all code to one side and therefore merging is just a breeze.

Any dvcs can be setup with a central server, but what you get is among other things

You can checkin any small change you like, as Linus says if you need to use more than one sentence to describe what you just did, you are doing too much. You can have your way with the code, branch, merge, clone and test all locally without causing anyone to download huge amount of data. And you only need to push the final changes into the central server.

And you can work with no network.

So in short, using a version control is always a good thing. Using dvcs is cheaper (in KB and bandwidth), and I think it is more fun to use.

To checkout Git : http://git-scm.com/
To checkout Fossil : http://www.fossil-scm.org
To checkout Mercurial : https://www.mercurial-scm.org

Now, I can only recommend dvcs systems, and you easily can use a central server

Share Improve this answer Follow

edited Jun 20, 2017 at 9:54
Vadim Kotov
8,274 ● 8 ● 50 ● 63

answered Mar 13, 2009 at 9:38
Trausti Thor
3,774 ● 34 ● 43

4

Distributed VCS are appealing in many ways, but one disadvantage that will be important to my company is the issue of managing non-mergable files (typically binary, e.g. Excel documents). Subversion deals with this by supporting the "svn:needs-lock" property, which means you must get a lock for the non-mergable file before you edit it. It works well. But that work-flow requires a centralised repository model, which is contrary to the DVCS concept.

So if you want to use a DVCS, it is not really appropriate for managing files that are non-mergable.

Share Improve this answer Follow

answered Jun 21, 2009 at 10:04

community wiki
Craig McQueen

3

The main problem (aside from the obvious bandwidth issue) is **ownership**.

That is to be sure to different (geographic) site are not working on the same element than the other.

Ideally, the tool is able to assign ownership to a file, a branch or even a repository.

To answer the comments of this answer, you really want the tool to tell you who owns what, and *then*

communicate (through phone, IM or mail) with the distant site.
If you have not ownership mechanism... you will "communicate", but often too late ;) (i.e.: after having done concurrent development on an identical set of files in the same branch. The commit can get messy)

Share Improve this answer Follow

edited Sep 21, 2008 at 18:06

answered Sep 21, 2008 at 13:41

VonC
1.3m ● 558 ● 4.7k ● 5.6k

1 This can be remedied via a technique known as "communication" :) – bk1e Sep 21, 2008 at 14:59

This is one of the strengths of a CVS, but also a palliative for the underlying problem. Merging with a DVCS tends to be less messy. One of the main advantages of a DVCS is serving geographically distributed teams! – riezebosch Sep 10, 2013 at 9:27

3

For me this is another discussion about a personal taste and it's rather difficult to be really objective. I personally prefer [Mercurial](#) over the other DVCS. I like to write hooks in the same language as [Mercurial](#) is written in and the smaller network overhead - just to say some of my own reasons.

Share Improve this answer

edited Jun 20, 2017 at 9:56

Follow

Vadim Kotov
8,274 ● 8 ● 50 ● 63

answered Sep 21, 2008 at 13:44

unexist
2,528 ● 23 ● 28

2

Everybody these days is on the bandwagon about how DVCSs are superior, but Craig's comment is important. In a DVCS, each person has the entire history of the branch. If you are working with a lot of binary files, (for example, image files or FLAs) this requires a huge amount of space and you can't do diffs.

Share Improve this answer

Follow

answered Feb 6, 2010 at 18:10

community wiki
elmonty

With Git, files are identified by content, meaning the same file is stored only once, even when they appear in multiple branches. And eventually files will be packed by storing deltas when there are too many loose objects around. Source: git-scm.com/book/en/Git-Internals-Packfiles
– riezebosch Sep 10, 2013 at 9:18

And bandwidth. And (as I've found at several large sites), increased merge conflict resolution that goes way wrong more often than not. – galaxis Aug 3, 2017 at 20:52

1

I have a feeling that Mercurial (and other DVCS) are more sophisticated than the centralised ones. For instance, merging a branch in Mercurial keeps the complete history of the branch whereas in SVN you have to go to the branch directory to see the history.

Share Improve this answer Follow

answered Sep 21, 2008 at 13:52
Roman Plášil
1,960 ● 2 ● 17 ● 27

1

Another plus for distributed SCM even in solo developer scenario is if you, like many of us out there, have more than one machine you work on.

Lets say you have a set of common scripts. If each machine you work on has a clone you can on demand update and change your scripts. It gives you:

1. a time saver, especially with ssh keys
2. a way to branch differences between different systems (e.g. Red Hat vs Debian, BSD vs Linux, etc)

Share Improve this answer Follow

edited Jun 21, 2009 at 9:41

community wiki
2 revs, 2 users 78%
Spoike

Even on a single machine it is amazing to be able to snapshot your pet project once in a while! – riezebosch Sep 10, 2013 at 9:20

0

W. Craig Trader's answer sums up most of it, however, I find that personal work style makes a huge difference as well. Where I currently work we use subversion as our One True Source, however, many developers use git-svn on their personal machines to compensate for workflow issue we have (failure of management, but that's another story). In any case. its really about balancing what feature sets make you most productive with what the organization needs (centralized authentication, for example).

Share Improve this answer Follow

answered Sep 21, 2008 at 14:01

Mark Kegel

4,510 ● 3 ● 23 ● 21

0

A centralised system doesn't necessarily prevent you from using separate branches to do development on. There doesn't need to be a single true copy of the code base, rather different developers or teams can have different branches, legacy branches could exist etc.

What it does generally mean is that the repository is centrally managed - but that's generally an advantage in a company with a competent IT department because it means there's only one place to backup and only one place to manage storage in.

Share Improve this answer Follow

answered Sep 21, 2008 at 16:34

MarkR
63.5k ● 15 ● 119 ● 154

With DSCM's every developer has a copy of the repository. How much backup do you want. – Ikke Jan 25, 2010 at 16:08

With back-ups, the driving force is consistency, not number/versions of it. We want to be cautious about conflating the separate concepts that source repo back-ups and decentralized code repo serving, esp in a commercial enterprise that has more of an emphasis on (and is more conducive to) centralized control, predicated on much greater (dependency on) repo availability assurance. – galaxis Aug 3, 2017 at 20:59