CollabNet

Which SCM? The Pros and Cons of Git and Subversion

git enterprise-grade

http://www.collab.net

Table of Contents

Has Git Killed Subversion & CVS?

Video killed the radio star, and the Internet killed both. Many believe Git is on it's way to killing Subversion (which all but killed CVS), but let's let the numbers speak for themselves.

There are over 20 different open-source and commercial version control systems (VCS) in use today – some getting more media coverage than others. In particular, distributed VCS (DVCS) like Git and Mercurial are attracting a lot of attention due to the popularity of Linus Torvald, GitHub and the recent acquisition of Bitbucket by Atlassian. I wondered what developers are actually using for their software development process so I started my research.

I pulled data on the Codesion VCS's in use – Subversion, Git and CVS. Interestingly enough, the old dinosaur CVS is still being used for new projects today, but showing a trending decline in favor of Subversion and Git. While this analysis only looks at the data on the Codesion system, it seems to be pretty reflective of the overall industry trends. Perhaps I'll have the time someday to put together a crawler which surveys VCS's running public web servers to gauge uptake beyond Codesion's data set.

Subversion

Subversion is easily the most popular and its usage is growing larger each month. Its centralized architecture make it easy to maintain a security hierarchy, access control, and backups, which is why it's preferred in many enterprise organizations over Perforce, VSS, and ClearCase (see figure 2 below).

Subversion is still widely popular amongst the open source community, most notably the Apache Software Foundation, FreeBSD, GCC, Django, Ruby, ExtJS, PHP, Python and MediaWiki.

CVS

CVS is being used for new projects, proving that the once dominate open-source VCS is still preferred by a segment of developers. Perhaps its the familiarity of CVS that keeps that keeps attracting people, or perhaps some just don't need all the bells and whistles that newer VCS's offer. CVS's core strength is in it's simplicity to keep revisions of code and share it with colleagues.

Either way, it's interesting all the same to compare the buzz of the newer systems with what people are actually using. That said the usage is definitely trending downwards and I'd expect it to be completed replaced in the next couple of years.

Git

Usage of Git with Codesion has remained pretty steady since we launched the beta in mid-2010. It's hard to draw too much from this trend without considering what is happening with other Git hosting providers. According to a 2009 Dr. Dobbs survey published by Forrester Research (see figure 2 below), Git ranks as the 7th most used VCS with 2.7% of those surveyed ranking it as their primary SCM. The survey included 1,020 developers who spend at least 30% of their time writing code.

If the constant media attention is a leading indicator, Git continues to gain in popularity - especially amongst the open-source, OSX and Linux development community. I fully expect this trend to continue upwards for Git over the next year. It still remains to be seen how Git will be adopted in the enterprise.

Overall

There does seem to be an explosion of core open source developer tools these days, not just in the (D)VCS space but also in other areas like databases, where there are more than 70 open source databases. I still remember only a few years ago when all you had to choose from was MySQL, PostgreSQL, SQLite and Berkeley DB. I expect this list to consolidate as clear winners emerge, and other fall behind due to lack of uptake, founding developers loosing interest or projects being abandoned. I suspect the same will be the case for the (D)VCS world, as a select few are adopted by the developer community, both in the private sector and open source community.

Subversion still attracts the largest number of developers world-wide, but Git and Mercurial are the fastest growing. I fully expect Git and Mercurial to continue up the adoption path as DVCS become more appealing. Subversion will hold strong given its centralized architecture, which many organizations prefer for the reasons pointed out above. I expect each of these 3 systems (SVN, Git, and Hg) to live happily along side each other filling different needs amongst developers, much like the databases of old.

What VCS to Choose?

If you're new to version control make a decision based on the system alone. You should consider what type of software you're developing (open source, proprietary, small or large project), what development practices you're using, and what internal security policies exist in your organization. Your answers will help lead you down the path to the best VCS for your project.

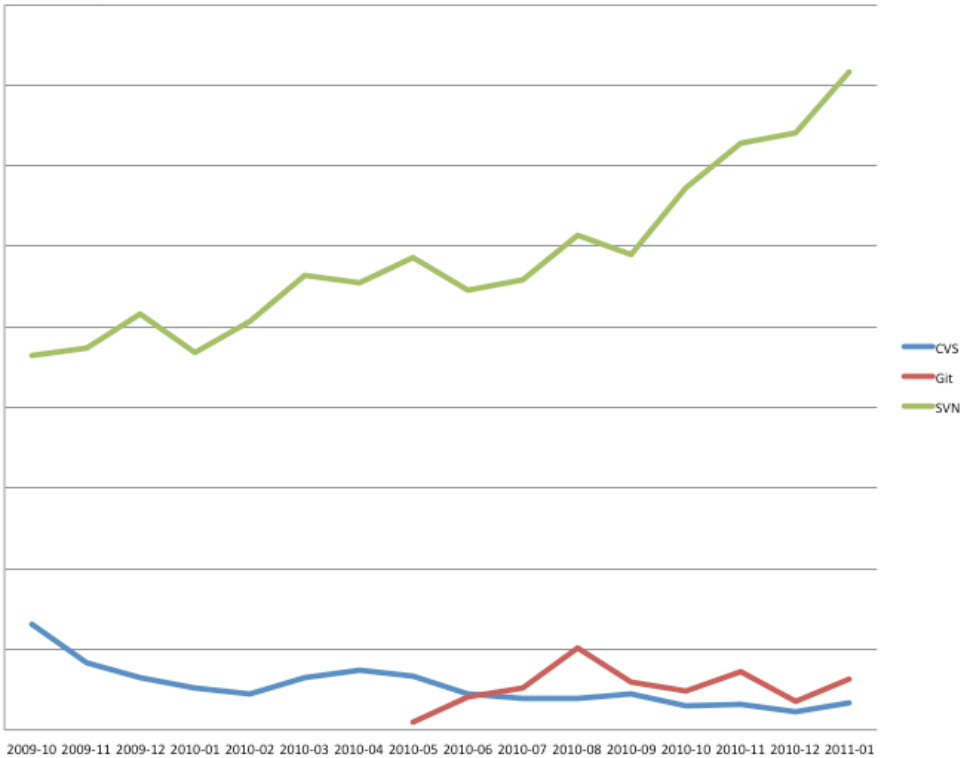


Fig 1: Graph showing the number of new projects created each month for each VCS. Data sourced from Codesion.com

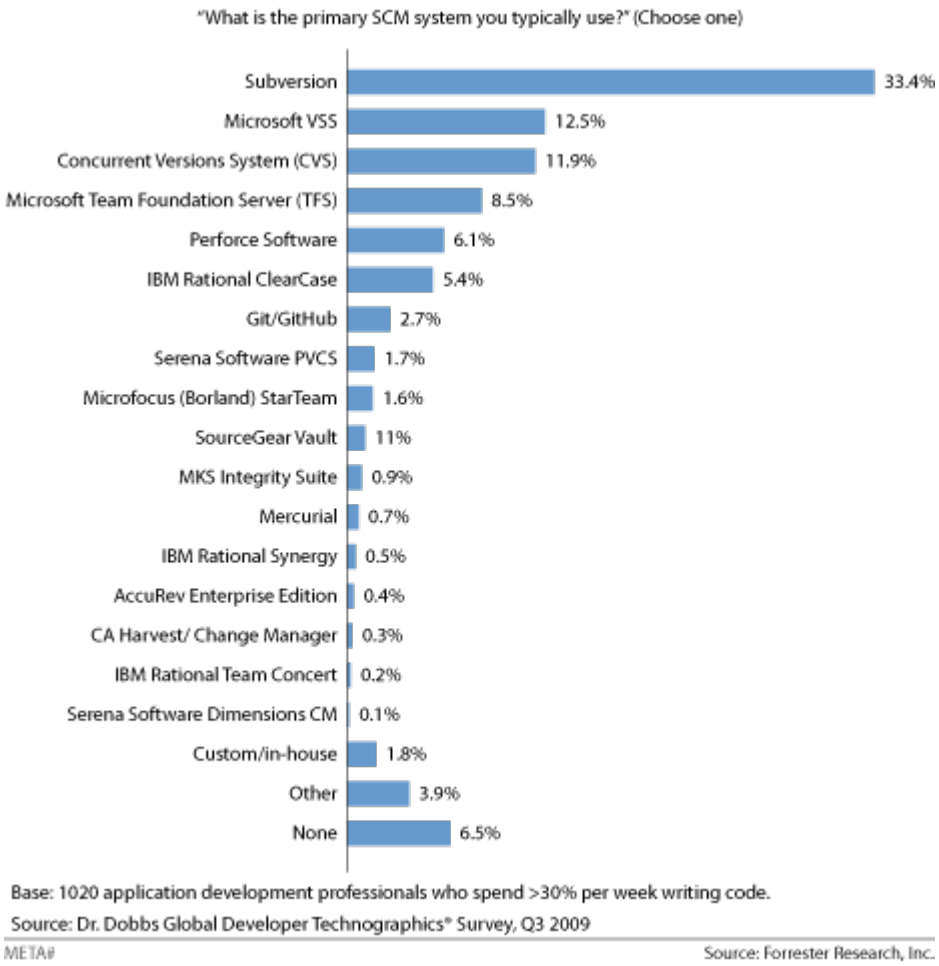


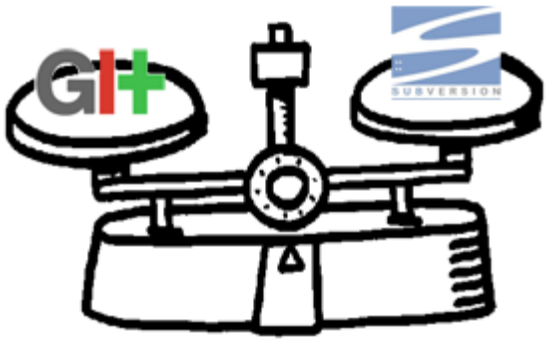
Fig 2: VCS usage patterns across a variety of systems. Source: [Jeffrey Hammond, Forrester research]

Subversion or Git? Decisions, decisions ….

by Jack Repenning

Are you facing a difficult choice between version control systems? Are you having trouble sorting out the alternatives? Are you bewildered by all the opinions you find on Google? Let's see if I can help sort this all out.

CollabNet (the original stewards of the Subversion project) provides both Subversion and Git, either through our enterprise TeamForge product or through our CloudForge Platform. We think there's a place in the universe for each. Here's how to find yourself in that universe.



The short form

Here are some specific recommendations. If you want to stop reading at the end of this section, you should make out just fine.

- **If you have compelling requirements for a single, certain, master copy of your work, use Subversion.** You can do this with Git, so long as there are no slip-ups. But you can't do anything else with Subversion (slip-ups or no), and "compelling requirements" like Sarbanes-Oxley are happier with guarantees than possibilities.
- If you plan to maintain parallel, largely shared but permanently somewhat different lines of the same product, use Git. One common example: perhaps you have a large product that you customize for each customer. The customizations are permanent, and generally not shared among code lines, but most of the code is common to all. Git was designed for just this case (in Git terms, local customizations to the common core, and occasional feature or bug-fix contributions back up-tree).

Neither of those? Take your pick, you should be fine with either tool.

The Basics

What do you want from your version control system? A lot of this is the same for everyone, and both tools accomplish these tasks just fine. If all you care about is the basics, you could easily just flip a coin and get on with your work, confident that your choice would be sound. These "basics" include:

- Store all versions of your files ("version control")
- Associate versions of each file with appropriate versions of all other files ("configuration management")
- Allow many people to work on the same files, toward a common goal or release ("concurrency")
- Allow groups of people to work on substantially the same files, but each group towards its own goal or release ("branching")
- Recover, at any time, a coherent configuration of file versions that correspond to some goal or release, either for investigation or extension like bug fixing ("release management")

Both systems do all these things quite well. If these things are truly all you care about … flip that coin! All the discussion and decision has to do with other details, details that are always secondary to the basics, but can become crucial to particular projects or environments.

So, if you haven't flipped that coin and gone away, let's look at the differences.

But first, some demythologization

Here are a few things you may have heard that just aren't true, or aren't true any longer:

- Git hates windows. No, it doesn't. It used to, but now there are good Git integrations with Windows Explorer (TortoiseGIT, SmartGIT) and most of the major IDEs (Visual Studio, Eclipse, Netbeans), and you no longer need to use the Unix emulation environment Cygwin.
- Git is only for hackers. As a Unix-centric, command-line only tool, Git was originally anathema for many workers. But with all the GUI integrations, it's considerably more friendly, within reach of nearly everyone (see Windows integrations above, plus non-Windows tools like GitX, Coda via GitX or Tower, Emacs).
- Git is hard to learn. Well, it's a lot easier to learn now, anyway, thanks to those GUIs, and to improvements in the command-line UI and documentation.

On the other hand

- Subversion is slow on Windows. The latest Subversion releases, up to the current 1.7, have made great strides in Windows performance.
- Subversion merging is hard. Well, it's a lot easier now, anyway, thanks to the continuing progress on "merge tracking."

So, what's different?

As you've surely heard, the key difference is that Subversion is "centralized," while Git is "distributed." You can go many other places to learn about what that means in their implementation, I don't spend time on that here. But I will talk about why it matters to you.

Local versioning

Git's distributed model means that you have full version control operations purely locally on your workstation. Of course, the changes you make locally are not visible to anyone else, until you do something else (Git's "push" or "pull"), but you can check changes in so they're not lost, create branches, merge among branches, back up to older versions, and browse and compare versions - all without a network. The one cost in all that is, you have to learn some additional commands (push, pull, clone) and workflows, if you ever expect anyone else to see your work. You can do everything locally that any version control system can do … and you also have extra steps required to make your work visible to others, kind of an inescapable trade-off. On balance, local versioning is a significant convenience for the developer, and a big part of the popularity of Git.

Guaranteed centralization

Most processes have some reason to want one copy of everything, somewhere, that's reliably known to be "the official version." With Subversion, that's the central repository. With Git, that's … some repository or other: the tool doesn't privilege any one repository over another, but users and conventions can. With Git, you have to remember to push or pull all changes into the designated central repository - not hard to remember, especially if you support it with other conventions, such as only building releases from the central, but still an extra step. With Subversion, there's nothing to remember or agree to: there is only one repository. By definition, anything that's checked in at all is checked in to the central. One place where this can become particularly important is in "governance," where external constraints like Sarbanes-Oxley or escrow contracts demand hands-off guarantees. On balance, enterprises with governance constraints value the guarantees of Subversion.

(Governance, traceability, ALM, and DevOps all require broader support than just the code and files, of course: there has to be integration among the code repository, the tests, the deployed product, and the tracking system. Fortunately, CollabNet now integrates both Subversion and Git with these other components.)

Throw-away work

Somewhat paradoxically, there are times when you want to throw away (or keep private) some work. Git is better at throwing things away than Subversion is: you keep the potentially throw-away work in its own repository until you decide whether to make it official. If you decide to toss it, there are virtually no left-overs in the repositories you keep. By contrast, in Subversion, everything goes into the one central repository, and you don't have that same option. Typically in Subversion, you don't actually "throw away" such work, but only leave it on some branch you never look at. But it's still there, taking up space, and possibly cluttering the history browser. The possibility of throw-away work is a fairly big consideration in open-source work (and in fact it was one of the key design considerations for Git), but rather less so for enterprise work (where throw-away work is also throw-away salaries, equipment, lighting, and all the other employee expenses), so enterprises typically avoid ever doing it in the first place. I wrote about handling throw-away work at some length a while back.

Un-recommendations

Some things are possible, but I wouldn't recommend them - at least, not to someone who has to ask:

- Git-SVN. Git is able to pull files out of a Subversion repository, store them in a Git repository (allowing all that local version control, and some inter-git-repository push/pull/merge), and then push the results back into Subversion. It sounds like the best of both worlds, doesn't it? But you end up having to be an above-average expert in both systems, and parts of the workflow are very slow. Some folks really like it. Maybe you would, too. But it's not what you'd call "mainstream."
- Cygwin. Cygwin is a system for providing a lot of Unix-like capabilities on a Windows system. Early Git Windows implementations ran inside Cygwin. Again, this ends up forcing you to be an above-average expert in two very different systems. If you're already steeped in Unix and Windows both, it can be handy. But it's a high price to pay for just adding a little Git. Fortunately, Git Windows installation no longer requires Cygwin.

So, which one's for you? Try a free, fully functional 30 day trial of Subversion and Git on CloudForge to help make this choice.

For More Information

CollabNet is your one-stop shop for enterprise-grade Git management.

Subscribe to Git Blogs: http://blogs.collab.net/email-subscribe?catName=Git

24/7 Git Support: http://www.collab.net/downloads/git-enterprise

Enterprise Git Management: http://www.collab.net/downloads/git-enterprise

Git Toolkit: www.collab.net/gotgit<http://www.collab.net/gotgit>

CONTACT US

Corporate Headquarters
8000 Marina Blvd, Suite 600
Brisbane, CA 94005
United States
Phone: +1 (650) 228-2500
Toll Free: +1 (888) 778-9793

Topics trending now

Many of the latest technology announcements have implications for PaaS and cloud development that will serve agile businesses everywhere.

- Enterprise Cloud Development, www.collab.net/ecd
- Continuous Integration, www.collab.net/getci
- 5 Things your Development Team need to be doing now, www.collab.net/5things

About CollabNet

CollabNet is the recognized leader in enterprise cloud development and Agile ALM, with more than 7,000 global customers that range from single workgroups to large enterprises. Its deep open source roots include the creation of Subversion, the industry leading version control system with millions of users. CollabNet helps enterprise customers build and deploy better software through its focus on collaboration, enterprise Agile methods and cloud development and computing. Many CollabNet customers improve productivity by as much as 70 percent, while reducing costs by 80 percent. Its solutions include TeamForge®, the industry-leading Agile ALM platform for distributed development, ScrumWorks® Pro for Agile project management, Subversion Edge for managed source code management, Codesion for cloud-based development and deployment, and a range of Agile-based training, consulting and transformation services. For more information, please visit (www.collab.net)

CollabNet, Inc.
8000 Marina Blvd.,
Suite 600
CA 94005

Tel +1 650 228 2500
Fax +1 650 228 2501
www.collab.net
info@collab.net

Blog blogs.collab.net
Twitter twitter.com/collabnet
Facebook www.facebook.com/collabnet
LinkedIn www.linkedin.com/company/collabnet-inc