# Version Control Systems

Tony Finch ⟨fanf2@cam.ac.uk⟩

Wednesday 18 April 2011

**Abstract**

Version control systems – the pragmatics.
Following an introduction to the subject from Jon Warbrick.

## Version control timeline

http://codicesoftware.blogspot.co.uk/2010/11/version-control-timeline.html

## SCCS – "source code control system"

- 1972 – SNOBOL, IBM OS/MVT
- 1973 – C, Unix
- PWB → System V → Solaris
- Specified by POSIX
- GNU CSSC "compatibly stupid source control" www.gnu.org/software/cssc
- www.cam.ac.uk

SCCS was developed at Bell Labs. It was part of the Unix-based Programmers' Workbench (PWB), and from there it was inherited by System V Unix, thence commercial Unixes such as Solaris, and standardized as part of POSIX. The GNU project has a free clone called CSSC.

It has some important proprietary successors which I'll come back to later.

We use it in the Computing Service to manage some of the content on the main web server. I believe this was originally just because it comes with Solaris.

**RCS – "revision control system"**

- 1982 – C, Unix
- free / open source
- Managed Mail Domains
- Hermes system configuration
- ~~PWF~~ MCS Linux package lists

Another imaginitive name.

RCS has a similar mode of operation to SCCS. The repository is a subdirectory of the working directory. You have to lock a file before you can edit it, to prevent multiple concurrent edits from accidentally conflicting. Locks can be inconvenient, but they are simpler than merging changes and resolving conflicting edits.

We use the locking feature for managed mail domains. A hypothetical more modern user interface would have nothing to do with version control.

RCS is still used for configuration management in a couple of places mainly for historical reasons.

**CVS – "concurrent versions system"**

- 1986 – shell scripts wrapping RCS
- 1990 – reimplemented in C
  - vendor branches
- 1995 – access repository over network
- 1996 – `cvsweb` repository browser
- Hermes source code & package configuration

CVS is a big step towards modern version control. Unlike SCCS and RCS, the repository is separate from a checked out working copy.

CVS is "concurrent" in the sense that multiple people can check out and edit a file at the same time. If someone else checks in changes to a file while you are editing, you must use the `update` command to merge those changes into your copy before you can check in your own changes.

Remote access to a CVS repository usually uses `ssh`.

I set up the Hermes repository soon after arriving, based on my experience of using it whe contributing to the Apache HTTP server and FreeBSD projects. It replaced some of our use of RCS.

We use the "vendor branch" feature to help maintain local customizations to some of the software we run on Hermes, most notably David Carter's extensive improvements to the Cyrus IMAP server.

### Subversion / `svn` – "CVS done right"

- 2000 – Cross-platform
- Apache APR, Berkeley DB, WebDAV
- Multi-file changesets
- Cheap branches
- Better authentication & access control
- DSpace / Unix Support / Development Group

Subversion was intended to be an evolutionary improvement over CVS.

With CVS you can commit changes for multiple files with one command, but the changes are recorded separately: each file has its own version counter, and if you are changing lots of files the revision timestamps will be different. Changes that copy or move files are a massive pain.

Subversion fixes these problems. It has "changesets" that record modifications to several files as an atomic unit. Its other advantage is its greater friendliness to non-Unix developers.

Subversion took quite a long time to become usable. Its repository storage was not particularly reliable until Subversion 1.1 was released in 2004, and it had very weak merge support until version 1.5 was released in 2008.

I'm not a big fan of it, since it is not a big enough improvement over CVS to be worth the effort of repository conversion, and it is much much slower to use.

Our `svn` repository was set up by Tom De Mulder.

### Early distributed version control systems

- 1991 – `smoosh` Sun WorkShop TeamWare 1999 – BitKeeper *Proprietary popularizer*

- 2001 – GNU Arch / `tla` *Free but loony*
- 2003 – Monotone *Merkle crypto hash tree*
- 2003 – Darcs *Patch algebra & Haskell*
- 2003 – `svk` *Distributed Subversion*

At Sun they took a different approach to concurrent development from CVS. Instead of having a central repository with multiple working trees, they based TeamWare on cloning SCCS trees which include both the working copy and the history files.

This allows each developer to record the history of their work-in-progress by committing to their local repository, without affecting the central shared repository. They have an independent development branch.

Promiscuous branching is only helpful if it is easy to merge two sets of changes back together. The `smoosh` program did this by combining SCCS history files.

BitKeeper is basically "TeamWare done right" – it has changesets rather than handling changes to each file independently. It is important because it was used to host Linux kernel development between 2002 and 2005, which did a lot to popularize distributed version control.

The other programs listed on this slide were all a bit too weird to be popular.

GNU Arch had a needlessly baroque command line interface and a prickly developer. It is no longer maintained, but it has a successor, Bazaar, about which more in a moment.

Monotone provided a lot of inspiration for Git. It has exceptionally paranoid cryptographic integrity checking, which makes it rather slow. It is still maintained.

Darcs was popular for a while amongst developers of a mathematical bent. Sadly it has problems scaling up to cope with large and/or long-running projects.

SVK uses Subversion for repository storage. Adding DVCS features allows developers to make commits when not attached to the network. SVK has been displaced by `git`'s Subversion interoperability features.

### Current popular distributed version control systems

- `git www.git-scm.com` *Swiss-army chainsaw*
- Mercurial / `hg mercurial.selenic.com`
- Bazaar / `bzr bazaar.canonical.com`
- Fossil `www.fossil-scm.org`

From most popular downwards:

The big beast of modern version control is Git, initially developed by Linus Torvalds and now maintained by Junio Hamano. It is very fast and handles large projects well. It has masses of features but they aren't particularly well organized. You have to know rather too much about its slightly peculiar data model to use it effectively.

Both Git and Mercurial were created in response to a massive falling-out between Larry McVoy (creator of BitKeeper) and the Linux project in 2005. Mercurial is not as popular or fast as Git but is rather more user-friendly, especially for Windows users.

Bazaar also dates from 2005 but took longer to attain a decent amount of performance and stability. It is now comparable to Mercurial. It is developed by Canonical and used a lot by the Ubuntu project.

Fossil is written by the author of SQLite. As well as version control it also has a built-in bug tracker and wiki. It emphasizes easy deployment.

They all provide multiple ways to share a repository, with differing degrees of convenience, performance, and access control: local filesystem, for small-scale use; via HTTP; via ssh; via custom protocols.

### Code hosting services

- `github.com`
- `gitorious.org` → `git`
- `bitbucket.org` → `hg`, `git`
- `code.google.com` → `svn`, `hg`, `git`
- `launchpad.net` → `bzr`
- `savannah.gnu.org`
- `alioth.debian.org`
- `sourceforge.net` → many

An advantage of DVCS is that you can easily publish a copy of your repository on a code hosting service, without having to rely on them as the only authoritative copy (as was the case in the early days of SourceForge when the choice was between `cvs` and `svn`).

Bitbucket started off as the Mercurial alternative to GitHub.

Google Code started off supporting only Subversion, then added Mercurial, and later Git.

Launchpad is Canonical's project hosting site.

**Git in the Computing Service**

Speculation...

- Repository hosting?
  - CS internal
  - Managed service
- Config. management
  - Starling

There is currently not much use of Git for collaboration within the Computing Service. It would help to have a central place to store shared repositories.

There is a package called `gitolite` which would be suitable for CS internal Git hosting. Unfortunately it does not have any way to delegate management so it doesn't scale up to very large organizations. There is `gitorious` which could be the basis of a managed Git hosting service, but it doesn't support private repositories which is one of the main reasons for running our own rather than leaving it to GitHub.

Ben Harris has been working on a system called "Starling" which uses Git for system configuration management. If it is successful it will replace Hermes's use of RCS and rdist, and motivate us to upgrade from CVS too!

"I'm an egotistical bastard, and I name all my projects after myself. First Linux, now git." — Linus Torvalds