Take control

LINUX FORMAT On the DVD
» Bazaar 1.13.1
» Git 1.6.2.4
» Subversion 1.6.1

Juliet Kemp compares three of the big contenders for managing your data changes: Bazaar, Subversion, and Git.

Our expert

Juliet Kemp thinks that version control is the best thing since automated backups, especially since the time she accidentally deleted 2,500 words then hit **:w**.

Version control systems are indispensable if you're working on a multi-person project, and they're pretty damn useful even if you're just working solo. Keeping a full history of the changes you've made gives you a basic backup and enables you to revert back to an earlier version if you screw something up.

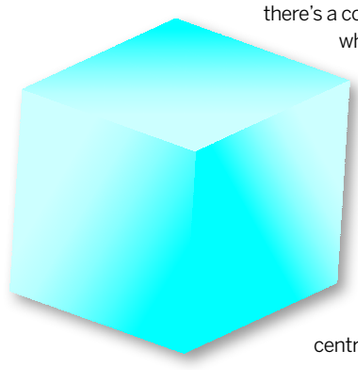
But with so many options available, from the rather dated *CVS* onwards, which one is best? What about distributed versus centralised? We look at three of the big names – *Bazaar*, *Subversion* and *Git* – to give you an idea of which one might best suit you and your project, whether that's large-scale software, small-scale coding, keeping track of config files or anything else that might spring to mind.

> **"There are two main types of version control system available."**

Client–server vs distributed

There are two main types of version control system available: client–server and distributed. There are also local-only systems, such as *RCS*, which operate on a single machine at a time, but those are very little-used now – it's both easier and more flexible to use a more modern system, even if you're only operating it locally.

Client-server systems work on a centralised model, where there's a copy of the current code on a central server, which users check out in order to work on locally. When a user has finished making their changes, they update against the central version (in case other people have made changes in the meantime), deal with any conflicts that might have arisen, and then check their code into the server, whereupon other people can check it out again.



Distributed systems are structured on a peer-to-peer basis: instead of one centralised repository, everyone has their own repository, and you synchronise by exchanging change-sets in the form of patches, or by merging branches. In practice, however, most projects of any significant size will have a single copy nominated as the main development branch, but this is a social difference rather than a technical one.

Both systems have advantages and disadvantages. Some of the advantages of distributed systems are:

» They provide a full backup of the codebase and change history with each branch, and there are many branches.
» It's easier to work without a network connection, because you can commit changes to your own local repository.
» Collaborating directly with other developers is easier, because you don't have to go through a central system.
» It's easier to create and destroy branches, and therefore easier to conduct experiments when developing!
» Some people see it as more empowering, encouraging new people to get involved in a project.
» It's possible to have multiple 'central' branches for different uses (stable, development or release branches, for example).
» Committing, viewing history, and other similar operations are fast, because there's no need to access a central server.
» Merging is in general much easier.

Centralised systems also have advantages:

» It's possible for a single person or entity to keep control of the whole history and project access (this can obviously be seen as an advantage in some circumstances and a disadvantage in others!).
» A 'master version' of the code is kept centrally, rather than having multiple competing versions.
» The central server can be explicitly designed and set up to be fault-tolerant, rather than relying on lots of people's personal machines.

In short, both types of versioning methods have their advantages, although distributed systems are becoming increasingly popular these days.

Bazaar

» **URL** http://bazaar-vcs.org
» **Licence** GPL
» **Used by** MySQL, Gnash, Squid

Distributed, but designed to support more centralised workflows as well.

Bazaar (*bzr* on the command line) is a distributed system that calls itself 'version control for human beings'. It aims to support a variety of types of workflow, giving you significant control over the way you choose to work and to use version control. It's also possible to use *Bazaar* with other version control systems, or with the repositories from other systems (such as *CVS* or *Subversion*).

Bazaar can be used with either a distributed workflow style, with small task branches for each new feature and developers using a local mirror branch to send changes back to the shared server; or with a more standard centralised version control style where developers regularly commit directly to the shared server. It also works well for personal single-user projects. The declared aim of the *Bazaar* team is that the software should fit the way you work, rather than you having to mould your working style to the software.

One nice feature of *Bazaar*, especially when you're working on your own, is that (unlike with *Subversion*) you don't have to create a repository, import your files, and then check out a working copy. You just work from within your project directory and *Bazaar* does its tracked changing from there. Of course, one downside to this is that it's more complicated to back the repositories up: you need to either keep all your projects as subdirectories of one main directory, or make sure that all of your directories are being backed up. (Which is not, of course, such a bad idea anyway.) It does also mean that a slip with **rm -rf** will take out your repository. This makes it much easier to start a project: instead of having to import your code and then check it out again, you can just initialise a new project from within your directory. You can also use a separate repository directory and check out a branch from that repository, if you prefer to work in that slightly more centralised way. Repositories are easy to set up, using the **init-repo** command.

> **"Bazaar enables you to commit changes without a net connection."**

Work offline

The distributed nature of *Bazaar* enables you to work and commit changes without a net connection. You can do this with centralised version control by having a local repository; but that can cause problems when you want to merge back in with the main repository. The way that *Bazaar* operates makes this localised version control easy: you download from a main project with the **bzr branch** command, which then creates a local branch on your own machine for you. You can work from this branch, or create further sub-branches as you like, and commit as often as you like. You can merge changes from the parent with **bzr merge**, and then when you're happy with your code, you can create a patch to send upstream with the **bzr send -o patchname.patch** command. Whoever owns the parent branch can merge the patch in or not as they prefer (using the same commands as when merging a branch). While in theory *Bazaar* enables you to operate without a central project tree, most projects will maintain a central tree and merge changes into that.

› **Getting a diff from *Bazaar* and checking the repository status.**

Bazaar's merge algorithm supports merging multiple branches, and will locate the most recent common ancester. It can also weave branches together, and can deal with some fairly complicated setups. However, it does require that the branches being used have some common ancestor (unlike *Git*, which will merge entirely unrelated trees).

Bazaar also supports cherrypicking, which is when you merge some changes from a branch (say up to version 104, or versions 105–7) but not all of them. You can also temporarily shelve changes that you're working on (take them out of your working tree, to return it to an earlier state, perhaps to make it easier apply a large upgrade/update from the parent branch), and then unshelve them when you want them back. This is useful when you're working on multiple patches, or when you want to assess other people's patches.

As with *Subversion*, hooks (scripts run before or after particular actions) are available.

Usefully for larger projects, *Bazaar* can be linked in with bugtracking solutions. By using the **--fixes** notation, you can associate a bug number in a particular bugtracking system (there's support for Bugzilla, Launchpad, Trac, and Roundup, among others). So this:

```
bzr commit --fixes project:23400 -m "Stores user birthdates properly"
```

will add a link in the log to bug 23,400 in the Bugzilla tracker for Project. (There's support for easy configuration for Bugzilla and Trac.)

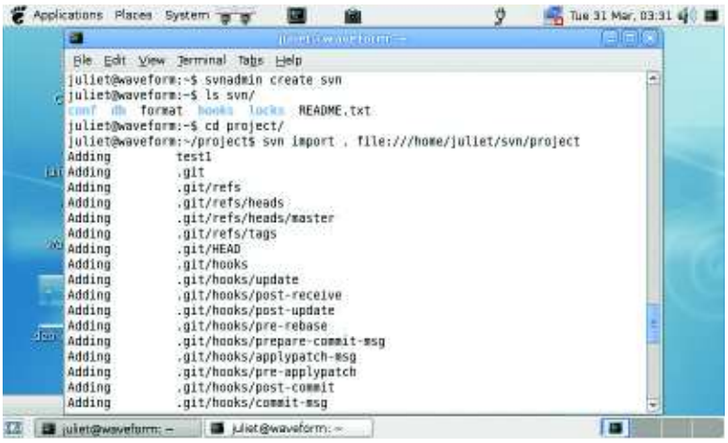
Other contenders

Perforce Popular setup using a centralised client–server model. *Perforce* is under active development, but the downside is that it's proprietary: it's free for up to two users or for OSS projects, but $900 per seat otherwise.
CVS Released in 1986, this is just about the oldest full version control system around. It's centralised, with some very well-publicised drawbacks (such as the high cost of branching/merging). It's still in use and still maintained, but new features are no longer being added, and for a new project you'd be better off with another option.
Mercurial Another distributed system in active development. It has a neat patch queue system, and the command line abbreviation is *hg*, which should remind you of your chemistry lessons.

Subversion

» **URL** http://subversion.tigris.org
» **Licence** Apache Licence
» **Used by** KDE, Python, Ruby, Mono, Google Code.

Centralised system designed to fix some of the problems with CVS.



› **Setting up and importing into a repository. This directory already has a *Git* repository, so all that information is also being imported.**

Subversion was designed as a successor to the very popular *CVS*, fixing some of its most notable problems or irritations in the process. It works on a client–server model, as *CVS* does. Your central repository can be local (accessed via **file://**) or remote (accessed via **http://** or **https://**, or via the custom **svn://** or **svn+ssh://** protocol).

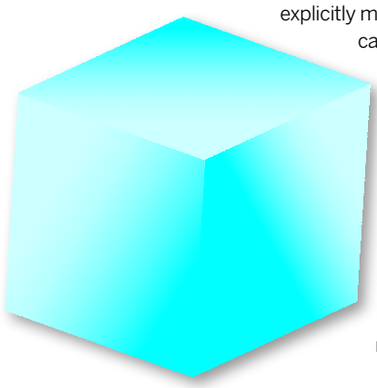
Unlike with *Bazaar*, you always have to set up a central repository (whether locally or remotely) before you start; so the process of getting your files under version control is slightly more effort. Once they're in there, you have to check them back out again to get a working copy

In terms of the change/check/commit cycle (change the file, check for conflicts, commit the change), the commands and basic operation are much the same as with any other version control system. To some extent, once you've got started with one system you have a head start on all the others, as many of the commands are similar.

> **"It's relatively simple to merge a branch back in with a trunk."**

Conflict resolution

If you encounter a conflict within *Subversion*, you have to explicitly mark the conflict as 'resolved' before you can commit the file. This can occasionally seem like a nuisance, but it does reduce the chances of a conflict being accidentally committed. (Although you can of course just remove the flag without actually resolving the conflict!)



Repositories can be branched and tagged, as they can in other systems, and it is relatively simple to merge a branch back in with the trunk. However, merging multiple branches, or cross-merging between branches, can be difficult; this is something that distributed systems handle far better than client–server systems.

You can also merge and separate whole repositories – the admin tools available for *SVN* include *svndumpfilter*, which enables you to filter out particular projects. In general, however, *Subversion* isn't designed for the same level of branch management flexibility as distributed systems are. There's no integral command to take a patch file and merge that into your tree; you have to use the standalone tool *patch*, which can cause problems with deleted or merged files.

Tagged commits

Subversion has a system of properties whereby you can attach versioned metadata to your files. You can set pretty much any human-readable label you like to be a property: it's a neat way of maintaining extra data about your files.

```
$ svn propset test "test property value" myfile.txt
$ svn proplist myfile.txt
Properties on 'myfile.txt'
  test
$ svn propget test myfile.txt test property value
```

There are some special properties, beginning with an **svn:** prefix, that do particular things, for example you can set the **svn:ignore** property on specific files and they will thereafter be ignored.

You can also, as with *Bazaar* and *Git*, set hooks: scripts to be run when particular things happen. These are useful for jobs such as checking that code will build correctly before a commit is allowed, removing trailing whitespace, changing tabs to spaces (or vice versa), and sending emails to your fellow developers after you've made a commit. (And, of course, anything else you can think of and have the ability to write a script for!)

Resources

» **Bazaar**
Bazaar in 5 minutes
http://doc.bazaar-vcs.org/bzr.dev/en/mini-tutorial/index.html
Guide to switching from other VCS tools
http://bazaar-vcs.org/BzrSwitching

» **Git**
Official Git tutorial
www.kernel.org/pub/software/scm/git/docs/gittutorial.html
SVN-to-*Git* crash course
http://git-scm.com/course/svn.html

» **Subversion**
Version Control with Subversion, an O'Reilly book available for free online
http://svnbook.red-bean.com

» **General version control**
Wikipedia article comparing various systems
http://en.wikipedia.org/wiki/Comparison_of_revision_control_software
Email from Linus Torvalds discussing the advantages of a distributed system
http://lwn.net/Articles/246381

Git

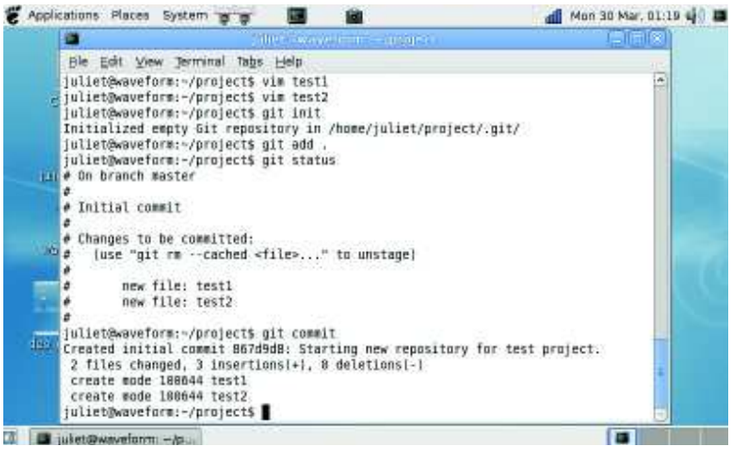
» **URL** http://git-scm.com
» **Licence** GPL
» **Used by** the Linux kernel, Gnome, Perl, X.org, VLC, Android

Highly distributed and very fast.

Git, like *Bazaar*, is another distributed version control option, initially created by Linus Torvalds for Linux kernel development.

One of the core features of *Git* is its support for non-linear development processes: the idea that changes will be repeatedly merged as they are passed around reviewers (as happens with the Linux kernel development process). In practice, this means that it's very easy to merge branches, and even to merge entirely unrelated, independent branches or trees that have no common ancestor. This also means that it's possible to merge unversioned code or files into an existing versioned tree: something which neither *Subversion* nor *Bazaar* can handle straightforwardly. *Git* is also designed to be fast, to deal with large projects quickly.

Git's distributed nature means that, like *Bazaar*, each working copy carries its own repository around with it (in the **.git** subdirectory), rather than the repository living in a central location as with *SVN*. Again, this means that it's easy to get a new project under version control – in the project directory, execute **git init; git add .; git commit** – but it also means that backing up is slightly more complicated. Again, if you want to you can set up your own version of a locally centralised repository.



› **Setting up a directory as a *Git* repository is painless. The # lines in the middle are the status output.**

It's compatible...

Like *Bazaar*, *Git* works with *Subversion*: you can use a *Subversion* repository directly with *Git*, using the **git-svn** commands. This can be massively useful if you just want to try it out, or if you're working with a project that uses *SVN* and doesn't intend to change.

Although largely similar, the commands are slightly different in a couple of cases from the ones used by *SVN* and *Bazaar*. There are a couple of really neat changes: for example, **git diff** automatically uses **less** as a pager rather than you having to remember to run it through a pipe.

It also has an interesting security feature: the history is stored in such a way that the name of a revision depends on the history to that point. Once the revision is published, it can't be changed without the change being visible.

In practice, this means that revisions are identified with SHA1 IDs: 160-bit hex numbers. The downside to this is that it's harder to use a revision number to identify a particular revision to work with, since they're long and complicated. However, *Git* will autocomplete for you, and there's always cut and paste.

Tags in *Git* are extremely powerful. You can attach an arbitrary description to the tag: in some cases, projects store a whole release announcement as the description. The name of the tagger is stored, and the tag can be PGP signed, thus, again, confirming not only the person's identity, but also the validity of the revision, history and tree through the revision ID system.

Unlike *Subversion*, when you merge branches, the full history of both branches is preserved, and branches can be repeatedly merged. *Git* really does put a big priority on flexibility and the ability to merge repeatedly and from multiple directions. As well as dealing well with patches (changesets), it also has strong support for applying patches that come in by email. You can directly feed in a mailbox with patch emails and it will grab the patches and apply them. There's also the *StGIT* tool for maintaining sets of patches.

> **"Git has strong support for dealing with patches that come in by email."**

Get your hooks in

As with both *Bazaar* and *Subversion*, *Git* has support for hooks: scripts that are set to run before or after particular events (eg checking for trailing whitespace before running a commit and exiting if any is found; or sending an email after a commit).

One slight problem *Git* does have is an inefficient use of space: each new object is stored as a separate file. To get around this, files are intermittently 'packed' together to save space.

Verdict

All three of the version control systems compared here are really good pieces of software: what you use depends on what your requirements are.

For an ultra-distributed setup, with lots of developers working largely independently, *Git* has major advantages. If you're working on your own, using a distributed system can also make sense, because it's so easy to create a new repository, even from an existing directory. And the easier you make it for yourself to use version control, the more likely you are to do it. However, make sure your backups are happening regularly!

For a more centralised project, *Subversion* has advantages – and there's plenty of support available for it. *Bazaar* is good as a bridge of sorts between centralised and distributed systems: despite its being distributed, it's easy to use in a more centralised way if that suits your project better.

Happily, the cost of experimenting with all of these various version control methods is low, especially for the distributed systems – so it's easy just to pick one and start using version control straight away, and then switch systems if you want to try another one at a later stage. LXF