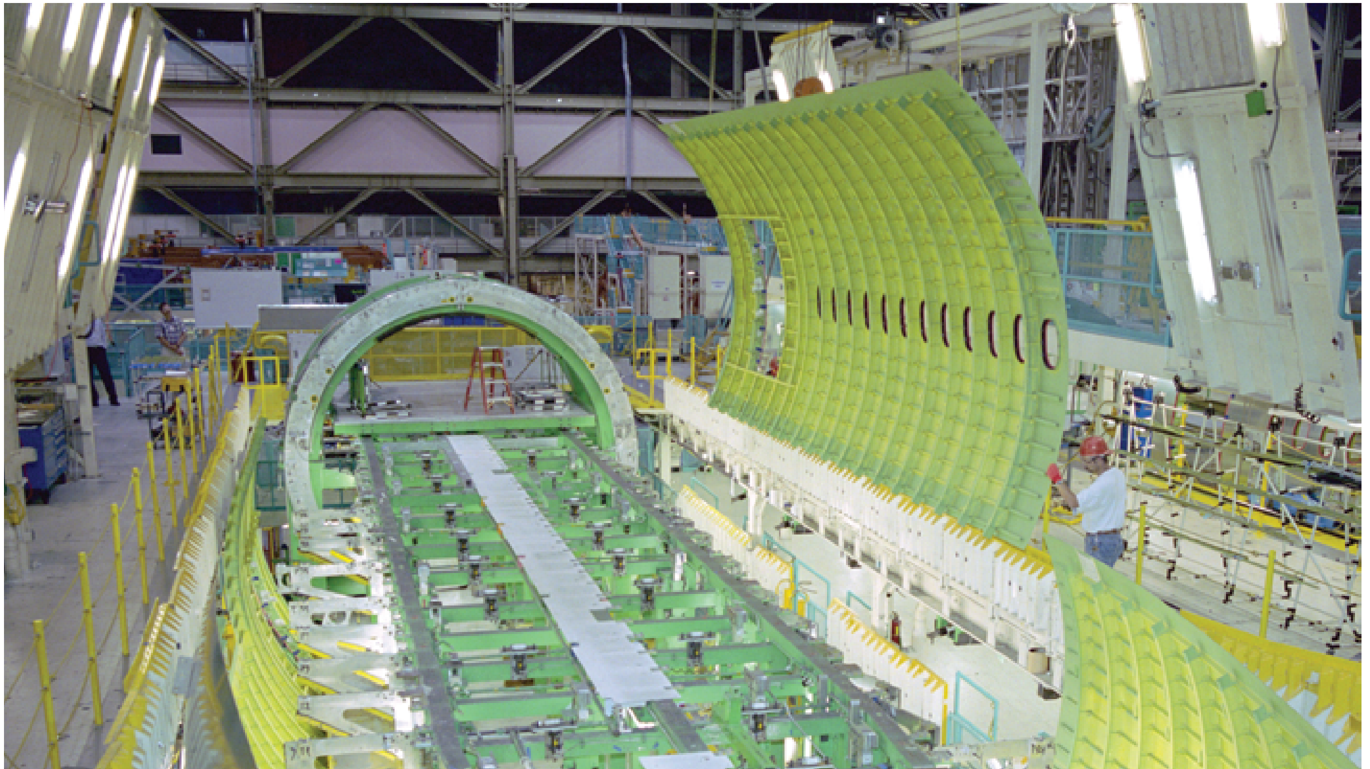
Subversion

# Building a better CVS

Subversion [1] is a free source code manager and version control system intended to replace CVS. We explain why you should consider changing to this new system and the pitfalls you may find. BY DAVID NEARY



Most open source developers have, at some stage, come across CVS. It is the de facto standard SCM (Source Code Manager) on free software projects. As such, it has a huge user base, and has earned a reputation as a good piece of software.

The primary goal of the Subversion project is "to build a compelling replacement for CVS in the open source community". In other words, it is designed to implement all of the functionality of CVS, with a familiar interface, while fixing its design flaws, and offering much improved functionality.

THE AUTHOR

*Dave Neary discovered Linux in 1997, and, apart from a flirtation with FreeBSD, has never looked back. He is an occasional contributor to the GIMP, and is listed as co-author of Gnect, one of the gnome games. He lives and works in Lyon.*

Unusually for an open source project, Subversion has had a number of full-time developers employed to work on it since the project's inception. CollabNet is paying the salaries of several developers, and holds the copyright on code in the project. The code is release under a BSD/Apache style licence.

Subversion is alpha software – this means that it is considered acceptable for public release, but is still in active development. Features are still being finished or refined, and bugs are found and fixed all the time. For those who are worried about using alpha grade software to house their projects, however, it's worth noting that Subversion's developers have confidence in it – the project has been self-hosting for a year and a half, with no data loss.

## A better CVS

For those who have used CVS for years with no problems, you might be asking yourselves what I meant by design flaws in the last section. CVS has a number of problems, primarily caused by its dependency on the RCS file format for versioning files. The issues addressed by Subversion include the following.

### Atomic commits

If you are making a change to a source code repository, and you commit that change, one of the fundamental princi-

ples of both database and version control systems is that either your entire change is accepted or your entire change is rejected. This behaviour is called atomicity (this is the A in ACID). In CVS, this is not guaranteed.

Atomicity is only guaranteed on a file-by-file basis. This means that if you are committing changes to 10 files, and someone else starts a commit at roughly the same time as you which changes the 8th file in your list, the changes for the first 7 files in your change get accepted, and the rest gets rejected. After this happens, it is very likely that the repository will be in an inconsistent state until you resolve the conflict you've just found and commit the rest of your change.

In addition, because CVS has no way of grouping changes to a number of files together, it isn't even possible to revert your partial commit while ensuring that the successful commit which happened at the same time doesn't get reverted as well.

An illustration might help explain the problem. Tom and Dick are working on the same source code tree which has 3 files in it – a.c, b.c and c.c. By coincidence, Tom and Dick try to commit changes at the same time. They both check that they are up to date with the repository, and then at the same time they try to commit their changes.

While Tom is writing his changes to a.c, Dick starts writing to b.c. Before Dick has finished, Tom starts writing to c.c. Dick's commit finds that Tom has locked c.c, and informs him that his c.c is no longer up to date. Dick does an update to get Tom's changes, and finds that there's a conflict in c.c which he has to resolve (perhaps in conference with Tom).

Meanwhile, Harry checks out the sources and has Tom's changes to a.c and c.c, and Dick's change to b.c, but not Dick's change to c.c. He tries to build the project, and finds that he can't. In brief, until Tom and Dick resolve Dick's conflict, neither Tom, Dick or Harry has a working copy of the source code.

Subversion implements atomic commits. When you commit to a Subversion repository, you start a transaction with the repository, and if any part of the commit fails, the transaction is rolled back and the entire commit is rejected.

## Files & versioning history

CVS has no way to rename files and keep versioning history. Renaming "file1" to "file2" in CVS means doing the following:

```
$ mv file1 file2
$ cvs remove file1
$ cvs add file2
$ cvs commit
```

This creates a new "file2" with no record of a common history with the old "file1" (which is now stored in the Attic).

In Subversion, the above operation is performed by

```
$ svn move file1 file2
$ svn commit
```

and the common history of "file1" and "file2" is conserved.

In addition, Subversion has dramatically increased the things that can be versioned. Directories and file metadata, as well as renamed or copied files, all have their own versioning. This means that not only can you move and copy files, you can move and copy directories too.

These copies are very cheap, because they're lazy copies – the first copy is similar to a hard-link to a particular version of the directory. As you change files on the branch, only those files you change get copied onto the branch. This means that making and maintaining branches is a cheap operation, both in terms of space in the repository and in terms of time.

## Branching and tagging

When we tag a repository in CVS, every file in the directory tree we are tagging is "stamped" with the tag. Likewise, when we are branching, the branch tag is created in each file affected. This means that branching and tagging are expensive operations for big repositories and directory trees, which has a cost proportional to the number of files being branched or tagged.

Subversion has made both branching and tagging constant time operations. In fact, Subversion makes no distinction between a tag and a branch. Both of these are implemented simply by copying the directory you are tagging, and have a cost the same as any other copy.

Logically there is no difference between branching and tagging – a tag is a copy of a group of files at a certain point of time, and a branch is a copy of a group of files at a certain point in time which can be changes independently of the rest of the tree. In brief, a tag is a branch that we don't change. In Subversion, this is the case. If you change a file in a tagged tree and commit, the tag suddenly becomes a branch.

In addition, requesting a file off a branch in CVS requires time proportional to the number of revisions since the branch point with the HEAD branch, plus the number of revisions made in HEAD since the branch point. RCS files store the latest revision in HEAD as full text, and any time you request the text of another version, it has to be constructed. This means, for a file on a branch,

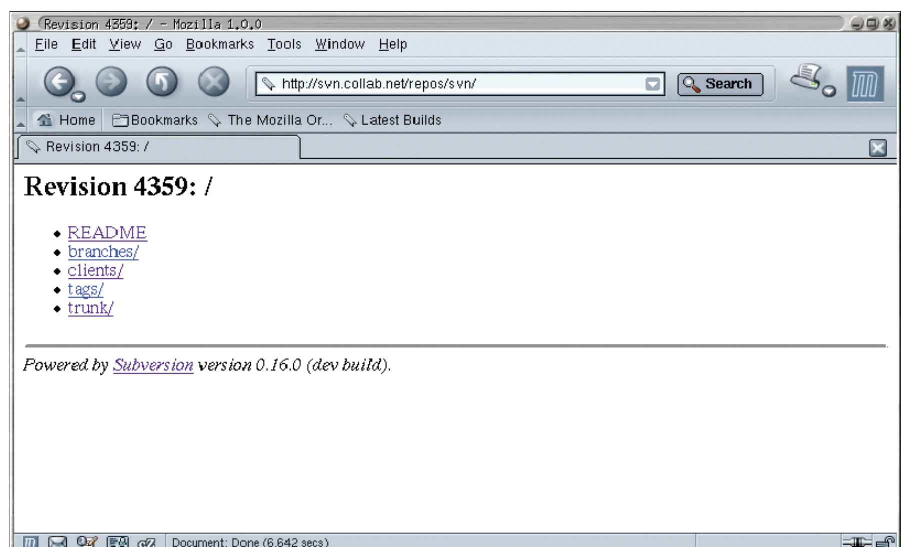



Figure 1: Screenshot of Mozilla pointing at the Subversion repository

reverting all changes from HEAD to the branch point, and then applying all the changes made on the branch to arrive at the complete file as we see it.

Because of this, diffs, branch switches and check-outs are all roughly proportional to the number of revisions on a file in CVS when those operations are on a branch. All branch and tag operations in Subversion are constant time.

## Binary diffs

Storing binary files in CVS is something of a nightmare. Because the RCS file format is essentially text based, any changes to a binary file resulted in the replacement of the old file. If the file is a 100K image that gets changed for every release, like the GIMP splash screen, then that one file ends up taking up many megabytes of space in the repository.

Subversion uses a diffing algorithm called Vdelta to provide efficient binary diffing, meaning that storing postscript or pdf documents which change frequently doesn't pose the same problems as it does for CVS. The diffing algorithm is also extremely efficient on text only files.

## Client-server communication

When we change a file locally when using CVS, and we want to know the difference between our changed copy and the repository version, the entire file is sent to the server, the diff is done there, and the result is sent back to the client. Similar operations are performed for all operations involving locally modified files, such as updates, commits and merges. This means that the cost of these operations is proportional to the size of the locally modified files, rather than the size of the change.

The philosophy of Subversion is different. The reasoning is that disk space has become a more plentiful resource than bandwidth in recent years, and therefore we should minimise use of the latter, even if there is a cost in the former. When we update a Subversion repository, a copy of the latest repository revision is made locally, as well as being patched into our local copy.

Because of this, diffs are sent in both directions by Subversion. That is, if we modify a local file and commit, only the differences between our local file and the most recent revision we have locally are sent to the server, meaning a lot less use of bandwidth.

In addition, because we have a pristine copy of the repository locally, there is a clear distinction between server operations and local operations – for example, finding out which local files have been modified, and the changes we have made to those files, are operations which can be performed without any access to the server. In fact, the cost of subversion operations in general is proportional to the size of the change, rather than the size of the repository or the size of the files being changed.

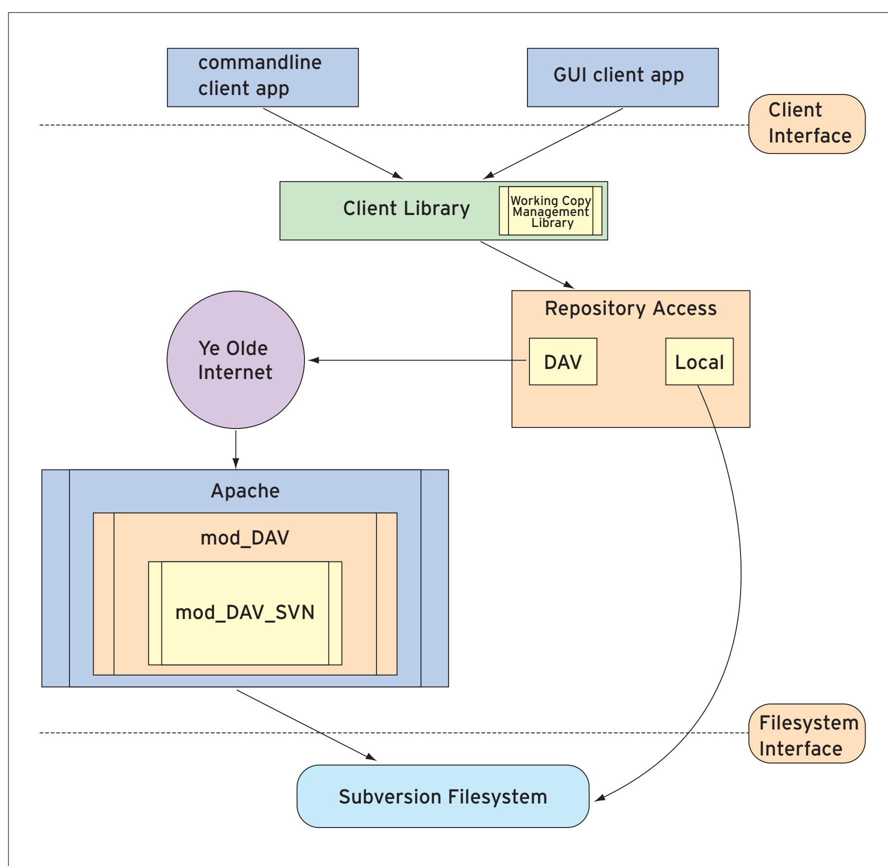



**Figure 2: Subversion design, copyright Brian Fitzpatrick, published under the Apache licence**

## Subversion's design – Repository versions

Subversion does not version files like CVS. Instead, it versions the repository as a whole. When you commit to the repository, a transaction is started, the changes you make are added to the repository, and if no problem occurs, the new repository with your changes is committed with its version number.

There are a number of advantages of this scheme over the CVS scheme. The most important is that this is the mechanism which is used to give atomic commits. It also gives a way to get a group of changes which were made at the same time (a changeset) very easily – you just get the difference between two successive repository versions.

## Apache as a server

Subversion uses the WebDAV and DeltaV extensions to the HTTP protocol for client/server communications. In practice, this means that it uses Apache 2 with a specialised module to do server operations, and the client talks standard HTTP/WebDAV. This means that on the server side, Subversion profits from a stable and well-tested network server.

Using Apache also gives several other useful features for free – client/server authentication is done using Apache's htpasswd mechanism, secure client/server communications are provided by modssl, and wire compression is supported with mod_deflate. In addition, Subversion repositories get a web interface for free – just point your browser at the root directory of the repository.

For those of you who are worried about having to install and administer

Apache to have a remote subversion server, there is also a lightweight, standalone server included with Subversion. This has not been extensively tested so far, but supports the same functionality as a CVS pserver.

## Modular design

Subversion has been designed from the start to be a client-server application. It is a set of libraries, each of which has a particular function, and a number of command line utilities. It has a very modular design.

Unlike CVS, Subversion has one library which performs all client operations, which means that writing graphical clients does not mean bolting a GUI onto the existing command line interface. Since the API is all in one library, a graphical client calls the same functions that the command line client does.

Subversion currently has two graphical clients – a wxWindows based client called rapidSVN and a gtk+ based client written in python called gsvn.

## A Few Words of Warning

Subversion isn't perfect – bugs are found & fixed every day, and there's a reason why it's called alpha software. There are several major problems outstanding which you might come across which have workarounds. Unlike some products, the Subversion project is upfront about these problems – they even have a page [5] dedicated to telling you about the worst problems it has.

Some of the reasons why you might not want to switch all your sources to Subversion straight away are listed below.

## Interrupted check-outs

Currently if you are checking out files from a repository there is no facility to resume the checkout where you left off, as with CVS. However, this problem is scheduled for resolution before 1.0. If you don't have many users checking out sources over 56Kb modem connections that go down all the time, this probably isn't a major issue.

## CVS to Subversion migration

Converting existing CVS repositories to Subversion while keeping version history is something which is more or less required. There is such a tool, but it is currently incomplete.

Because of CVS's file-based versioning, it is very difficult to migrate branches and tags, which are implemented as copies, to Subversion. Support for CVS should be complete before a 1.0 release, but this is a difficult problem.

```
 --revprop               : operate on a revision property (use with -r)

dave@bolsh:~/work/articles$ svn proplist -v *
Properties on 'svn_browser.png':
  svn:mime-type : image/png
Properties on 'svn_design.png':
  svn:mime-type : image/png
dave@bolsh:~/work/articles$ ls
CVS_linuxmagazine.txt         Subversion_resume_linuxmagazine.txt
CVS_resume_linuxmagazine.txt  svn_browser.png
Subversion_linuxmagazine.txt  svn_design.png
dave@bolsh:~/work/articles$ svn status
?      .xvpics
M      Subversion_linuxmagazine.txt
A      svn_browser.png
A      svn_design.png
dave@bolsh:~/work/articles$ rm -rf .xvpics/
dave@bolsh:~/work/articles$ svn commit -m "Almost finished draft"
Sending        Subversion_linuxmagazine.txt
Adding  (bin)  svn_browser.png
Adding  (bin)  svn_design.png
Transmitting file data ...
Committed revision 10.
dave@bolsh:~/work/articles$
```

**Figure 3: A typical subversion session with stuff like svn status, update, proplist and commit**

## Getting started

The easiest way to see the benefits of Subversion is to use it. Getting started is a little tricky for the moment, but as the technologies used become more commonly used, this will certainly get easier.

## Subversion client

To get started, you will need a subversion client, and the ability to create your first repository. The database which Subversion uses to store your files in the repository is Berkeley DB version 4 [6].

For Debian users, Subversion is included in unstable. For Red Hat and Mandrake (and other RPM based distributions), there are RPMs, available off the project download page. For everyone else, you can build from sources. The latest release source tarballs are available on the Subversion site [7].

The latest stable release of Subversion is version 0.17 and it requires DB 4.0.14. To build from sources, look at section II of the install guide [8]. The basic procedure is the usual ./configure, make, make install.

If you want to build from the latest sources, the requirements are a little more complicated – see the install guide for details.

## Setting up a repository

Once you have the binaries installed, you will want to set up your first repository. Detailed instructions on how to do this are available in the README document in the project sources [9]. To create a repository, just run the following command:

```
svnadmin create /path/to/repos
```

### Listing 1: A recommended directory structure

```
/
  trunk
    project1
    project2
  branches
    project1
      // Create project1 branches here
    project2
      // Create project2 branches here
  tags
    project1
      // Create project1 tags here
    project2
      // Create project2 tags here
```

### Listing 2: Creating a directory structure

```
$ mkdir tmp
$ mkdir tmp/trunk tmp/branches tmp/tags
$ mkdir tmp/trunk/project tmp/branches/project tmp/tags/branches
$ cd tmp
$ svn import file:///repos_path . -m "Initialise directory structure"
```

Make sure that you have permission to write to the directory */path/to/repos*, and that the directory exists.

The Subversion developers recommend that you set up an initial directory structure (which you can always add to or change later) to put in place a branching and tagging system. Keeping track of where you made a certain tag or branch can be tricky without a system. Setting up a basic structure in all your repositories to make this easier is probably a good idea.

There are several ways to set up this structure, the easiest is to create the directory structure yourself as in Listing 2, and use svn import to import the lot into the repository in one go.

Once you have your basic filesystem structure in place, you can now import your sources into the repository, and you're ready to go:

```
$ cd project
$ svn import file:///repos_path
/trunk/project . -m
"Import project sources"
```

Check them out to see that all has gone well:

```
$ svn co file:///repos_path
/trunk/project svn_project
```

And we can remove our old project sources, and use subversion:

```
$ rm -rf project
$ mv svn_project project
```

## Apache 2 & Subversion

To get a Subversion network server up & running on your machine, you will first need to have Apache 2.0 [9] installed, including APR and APRutil. In addition, you'll need to install Neon [10], which is the implementation of the WebDAV and DeltaV which the Subversion client uses to talk to Apache.

The latest stable release of Subversion at the time of writing requires Apache 2.0.44 (or a pre-release) and Neon 0.23.5.

Untar Neon into the svn source tree in the neon directory, and configure *--with-apxs=/path/to/apxs*. Make sure that you compile Subversion without debug flags, unless you also have a debug build of apache – otherwise the subversion module won't load up.

After building and installing, make the necessary changes to Apache config file to add the mod_dav_svn module, and to indicate the location and access method for the repository, and then start apache.

Now if you point your web browser at *http://localhost/repository* (or whatever location you told Apache), you will see your source tree. If you want to restrict access to the source code, or enable secure tunnelling, you should consult the documentation for Apache 2.

## Learn Subversion in 3 minutes

Below is a short list of the most common subversion commands along with a description of what they do. A slightly more detailed list is available in the Subversion Quick Guide [11], and a comprehensive description of all commands available in Subversion is available in the Subversion book [3].

## Conclusion

Subversion is a natural successor to CVS. It has come very far in just two and a half years of development, and is stable enough to be used for source projects as a direct replacement for CVS.

There are some obstacles to it's widespread adoption, one of which is that the client software for subversion is not yet as widely available as the CVS client. In addition, if you already have your sources in a CVS repository, migration to Subversion isn't perfect yet.

However, as we have seen here, it has an impressive feature set, and is very well tested at this stage. It is fast approaching beta status, and is getting more and more polished.

If you are planning on importing sources into a new repository, and you were thinking of using CVS, Subversion might be just the program you should consider. ■

### Common Subversion commands

| Command | Description |
|---|---|
| co/checkout | Check out a copy of the source code to a working copy. |
| ci/commit | Commit your local changes to the repository |
| up/update | Update your working copy to reflect changes to the repository since your last update |
| status | New in Subversion – Summarise your local changes, without talking to the repository, or summarise resources that are out of date without updating. |
| add, remove/rm | Add or remove files to/from version control. |
| copy/cp, move/mv | New in Subversion – Copy/move a file or directory to another file, keeping old version history. |
| merge | Equivalent to cvs update -j - merge changes from another location into the working copy. |
| switch | Change your working copy to use another branch |
| diff | Get differences between your working copy and the last updated sources (new to Subversion), or the current repository |
| log | Show log entries for resource |
| propadd, proplist, propdel, propview | New to Subversion – Manipulate metadata on a file. You can associate arbitrary data to any file or directory. A certain number of metadata keys have a special meaning (for example, svn:mime-type). |

### INFO


[1] Subversion home page: *http://subversion.tigris.org*

[2] CVS home page: *http://www.cvshome.org/*

[3] Subversion book: *http://svnbook.red-bean.com/book.html*

[4] Subversion sources: *http://svn.collab.net/repos/svn/*

[5] Inconveniences page: *http://subversion.tigris.org/inconveniences.html*

[6] Berkeley DB home page: *http://www.sleepycat.com/*

[7] Subversion downloads: *http://subversion.tigris.org/servlets/ProjectDocumentList*

[8] INSTALL: *http://svn.collab.net/repos/svn/trunk/INSTALL*

[9] README: *http://svn.collab.net/repos/svn/trunk/README*

[10] Apache2 home page: *http://httpd.apache.org/*

[11] Neon home page: *http://www.webdav.org/neon/*

[12] Quick Reference: *http://subversion.tigris.org/files/documents/15/177/foo.ps*