Managing Source Code With Subversion

Jonathan Oxer

May 3rd, 2005: Linux Users Victoria

Source Code Management

Source Code Management systems (SCMs) rock. Definitely the single most useful tool for a development team, ranking second only to a good text editor (just!).

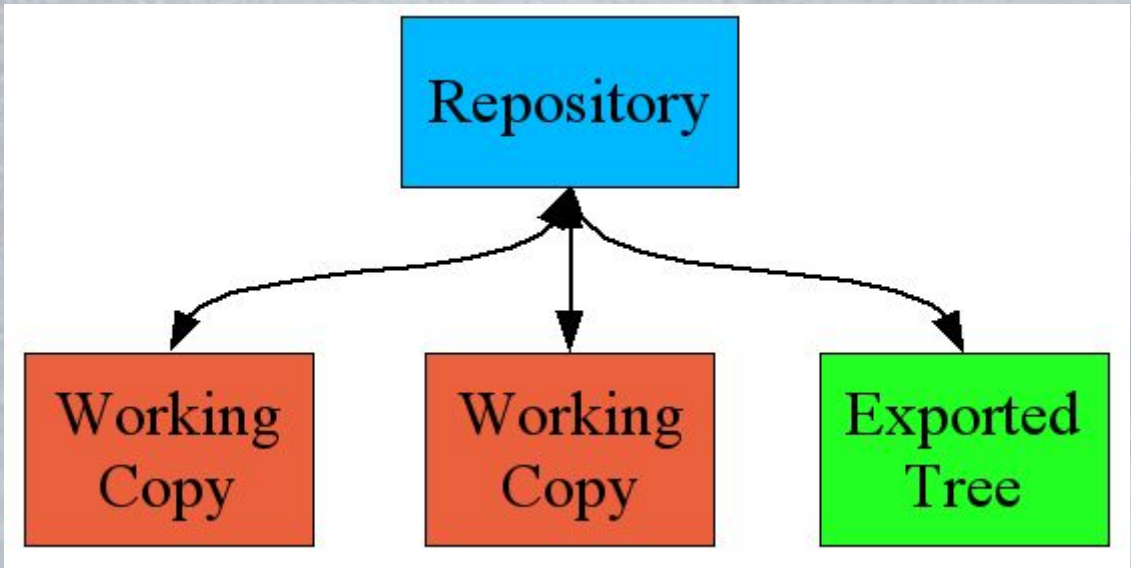
SCMs allow multiple developers to work collaboratively on the same codebase and tracks which changes have been made by which developers. They also attempt to automatically merge changes together when possible, and provide a mechanism to assist with manual merging when necessary.

Well known SCMs include CVS, Subversion, ARCH, BitKeeper and Visual SourceSafe.

SCMs: Centralised vs Decentralised

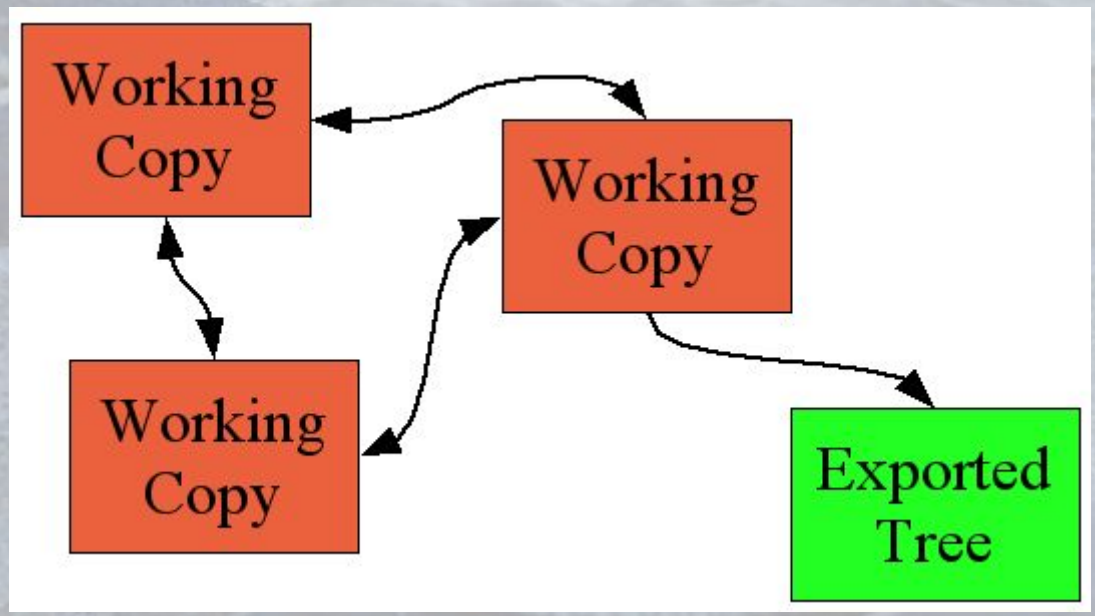
SCMs generally fall into one of two categories: centralised and decentralised.

Centralised systems use a master “repository” with all developers working on a checked-out “working copy”. All communication is between the repository and the working copies:



SCMs: Centralised vs Decentralised

Decentralised systems have no one master repository, being structured more like a peering network of working copies:



Newer SCMs are tending toward being decentralised.

SCMs: Changeset vs Patch Oriented

SCMs also generally fall into one of another two categories: changeset or patch oriented.

Changeset oriented SCMs work in terms of “revisions” or “versions” of the source tree or files, and changes are sequential.

Patch oriented SCMs work in terms of discrete “patches” which are applied to the codebase. Developers can choose to apply certain patches and discard others.

Patch-oriented SCMs are a relatively recent development and less developers are familiar with them than with changeset SCMs.

Which SCM To Choose?

Not quite a religious war, but developers often have strong preferences. CVS is very widely used and many developers are familiar with it but it's quite dated.

Subversion is rapidly taking over as the most widely used SCM in the F/OSS world: it's essentially a “work-alike” re-write of CVS, so many developers already have the workflow wired into their hind-brain.

ARCH-based systems are very interesting if you're willing to go to just a bit more effort to get your brain around it.

For an overview of SCMs see better-scm.berlios.de

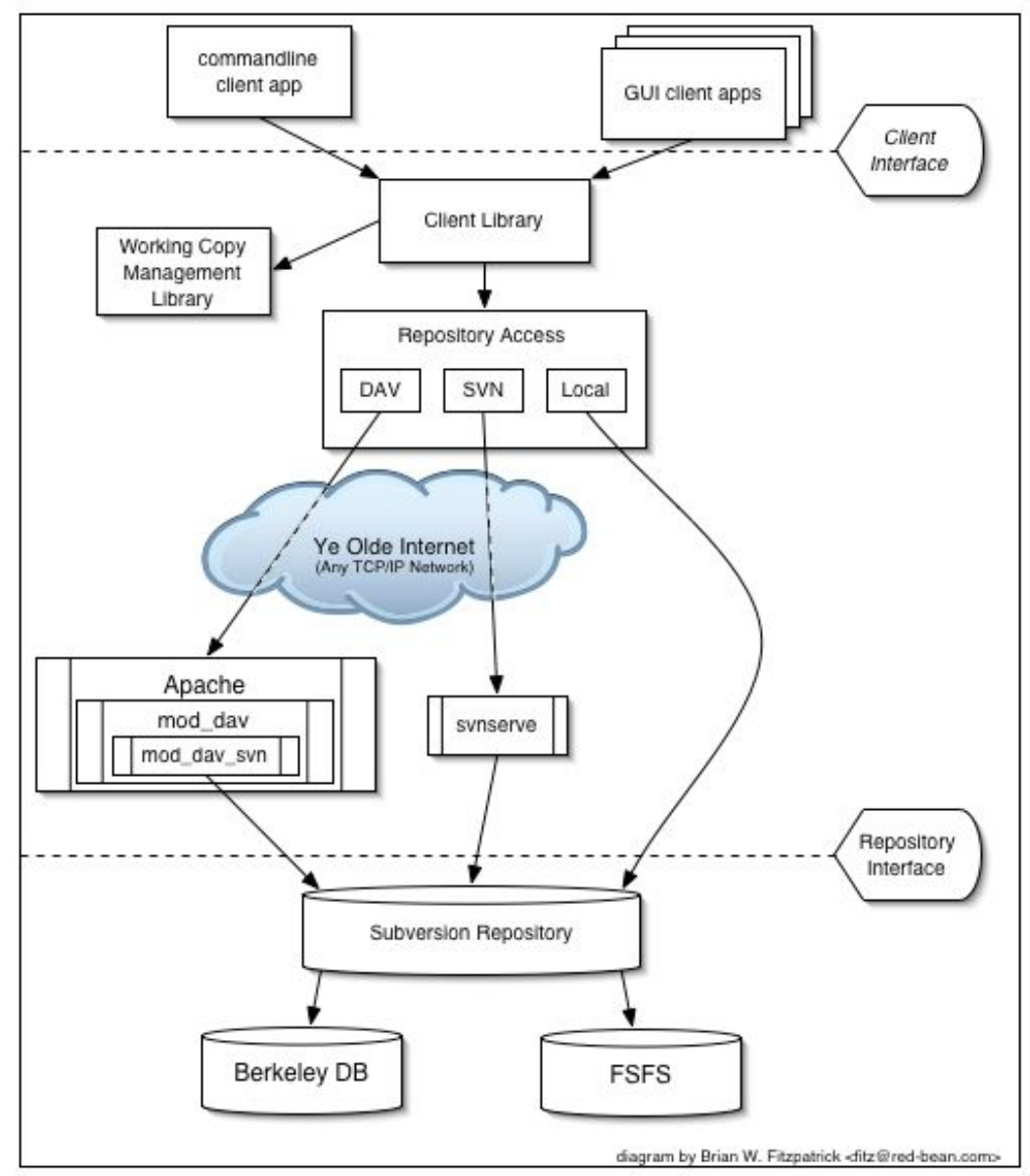
Subversion Basics

Subversion comes in two parts:

1. The server, which consists of both the server-proper and a bunch of original and third party admin tools for managing repositories. The server is built using the Apache2 libraries and can be configured to allow various access methods such as WebDAV, SSH and local filesystem.

2. Client tools, which allow you to connect to a server and manage a local working copy. Subversion provides powerful CLI tools and a number of third parties provide GUI tools for those who insist on point-n-click.

Subversion Architecture



Server Setup

1. Install using the packaging system of your distro or fetch the source and build it yourself. Best to use a distro package if you can: building from scratch isn't too bad but it uses a bunch of libraries including Apache and BDB so save yourself the versioning headaches if possible.

2. Create yourself a test repository somewhere:

```
svnadmin create /var/lib/subversion/luv
```

3. Set write privileges for your Apache2 user:

```
chown -R www-data /var/lib/subversion/luv
```

Server Setup

4. Add an entry to your “apache2.conf” so it knows how to reference your repository:

```
<Location /luv>
    DAV svn
    SVNPath /var/lib/subversion/luv
    AuthType Basic
    AuthName “LUV SVN Repo”
    AuthUserFile /etc/subversion/luv-passwords
    Require valid-user
</Location>
```

5. Create a password file and add a user:

```
htpasswd2 -c /etc/subversion/luv-passwords testuser
```

Test Your Server

Your repo will now be accessible by WebDAV through your Apache2 server, so all you have to do is point a browser at it!

http://localhost/luv

Hopefully everything will be working, but if not the things to check are:

- Has Apache2 restarted properly? Check error log for DAV loading errors
- Check permissions on the repo directory
- Check authentication setup (user listed in password file?)

Optional: Install WebSVN

WebSVN is a great tool for browsing your repositories. It gives you access to a bunch of meta information you wouldn't normally see through the web interface:

websvn.tigris.org

(or “apt-get install websvn” for the enlightened among us!)

Finally The Good Stuff! Adding Files

At this point you need to import some files into the repo. Technically you can import pretty much anything you like, but for a software project I recommend you set yourself up a structure like this:

```
project/branches/
project/tags/
project/trunk/
```

Then stick your project files into 'trunk'.

You can just put your project files at the top level but as you'll see soon that would limit you down the track.

Import Your Project Tree

Almost there!

Import your project tree into the repo:

```
svn import project http://localhost/luv
```

Now your files are in the repo. Point a browser at it or use WebSVN to see for yourself.

Check Out A Working Copy

At this point your project tree is untouched and pretty useless because it's not being managed by subversion: **don't** make changes in it. Instead you need to “check out” a local working copy and start working in that.

```
svn checkout http://localhost/luv/trunk project2
```

You'll then have a working copy of the source tree in a directory called “project2”, and it will be exactly the same as your original project tree with the exception that it's being managed by Subversion. Subversion will track any changes you make to any files within the working copy.

Working The Working Copy

Now it gets interesting. The basic commands to remember are:

svn checkout <repos_path> <local_dir>
svn update (up)
svn status (st)
svn commit (ci) (-m “commit message”)
svn diff
svn revert
svn add
svn mv
svn rm
svn mkdir

Basic Workflow

99% of the time the workflow is very simple:

After making changes to the working copy, “svn up” to make sure you're up to date with changes made by other developers and merge them into your working copy.

Use “svn status” to see the status of the files in your working copy.

Commit your changes back up to the repo with “svn commit”.

The other 1% is when you have to deal with conflicts.

Conflict Resolution

Sometimes changes “conflict”: that is, they are close to each other within the file and Subversion can't figure out how to merge them. When that happens it asks you to fix the problem and you'll be blocked from committing back to the repo.

If “svn status” shows files with a “C” status, you need to fix them manually. Look for markers like this:

```
<<<<<<< .mine
>>>>>>> .r23
```

Both versions of the section of code will be included in the file. Edit the file until you're happy with the result.

Conflict Resolution

When you're happy with the result you need to tell Subversion that you've fixed the problem:

```
svn resolved filename.c
```

You'll then be free to commit to the repo.

Meta Data: Object Properties

Subversion lets you store arbitrary meta-data with objects, including directories:

svn propset <propertyname> 'property value' file.c
svn propedit <propertyname> file.c
svn proplist file.c
svn proplist --verbose file.c

Branching, Tagging And Merging

Sometimes it's not enough to just have one tree: sometimes you need many. A typical situation is having a stable tree and a development tree, allowing bug fixes to be committed to the stable tree without forcing the dev tree to be released.

Subversion takes the concept of branching and executes it in a very simple way: in fact, it has no special internal “branch” mechanism at all. It lets you use the “copy” command to snapshot any arbitrary part of the tree and keep working on it as if it's your own private working copy, or copy it and label it with a tag for posterity.

Then later you can use “svn merge” to migrate changes between your branch and the trunk.

Tag

A “tag” is really just another copy of the tree, but one that no-one commits to. In fact it's identical to a branch: the only difference is how we agree to use it.

A tag can be created from an arbitrary revision, or by copying a mixed working copy.

Backups: Dump And Restore

Subversion stores its back-end data in a BDB database, and provides tools to dump the contents (including full history) in a portable way and also to load a dumpfile.

```
svnadmin dump /var/lib/subversion/luv > mybackup
```

Likewise a restore:

```
svnadmin load /var/lib/subversion/luv < mybackup
```

This is very useful for backups, migrating repos, and handling major upgrades to Subversion itself. Watch for repo ownership though.

Exporting The Tree

Sometimes you don't want to create a working copy full of ".svn" directories: you just want to output a snapshot. It's a very bad idea to leak working copies out to the world because they contain a bunch of hidden meta-data.

Use "svn export" just like "svn checkout":

```
svn export http://localhost/luv myexport
```

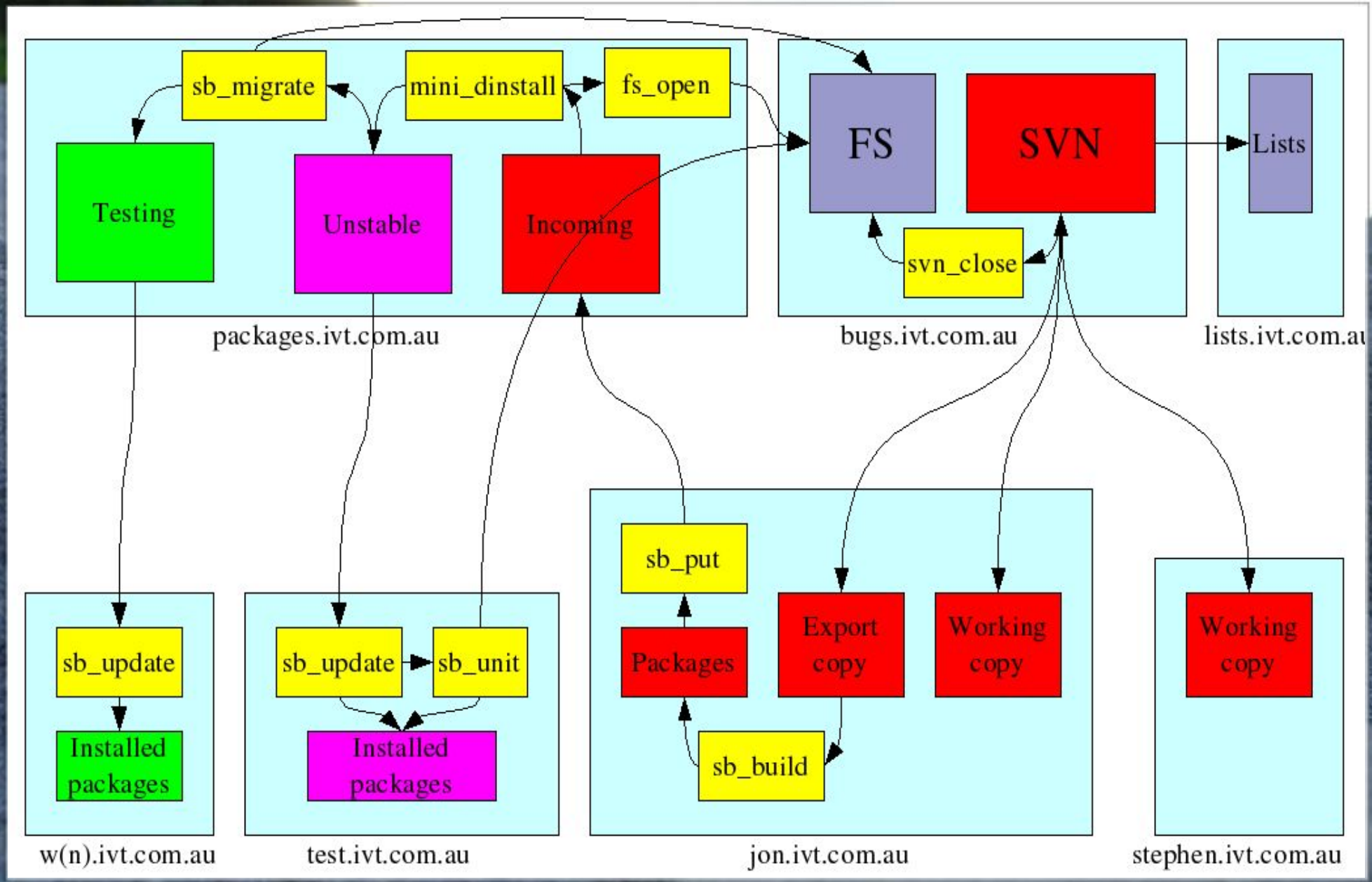
You can then use the exported tree for release packaging, etc.

Automate Your World: Hook Scripts

When an operation is performed in Subversion it checks for the existence of a number of different scripts inside the repo and executes them if it can find them:

- **start-commit**: Run before a transaction even begins, and doesn't have access to much information.
- **pre-commit**: Run after a transaction has been done but before it's committed, at which point it's possible to examine what the transaction does and make decisions.
- **post-commit**: Run straight after a transaction is committed.
- **pre-revprop-change** / **post-revprop-change**: Used for managing versioned properties.

The Automatic World



Questions? Need More Info?

These slides are online at:

jon.oxer.com.au/talks

The main Subversion site is:

subversion.tigris.org

“The SVN Book” can be downloaded from:

svnbook.red-bean.com