Scratchbox migration to Subversion

Janne Kataja

janne.kataja@movial.fi

Scratchbox migration to Subversion

by Janne Kataja

Published $Date: 2004/09/16 14:24:09 $

Revision history

Version:	Author:	Description:
2004-09-01	Kataja	Corrections
2004-08-31	Kataja	Proposal
2004-08-03	Kataja	Initial version

Table of Contents

Chapter 1. Introduction

This document is a study on changing Scratchbox development version control system to Subversion. Currently Scratchbox project uses CVS. Many of the problems with CVS are due to CVS being built upon RCS. RCS only tracks changes in single files and doesn't allow concurrent users. CVS adds support for modules and multiple users as an afterthought. Subversion is a version control system that is designed to replace CVS and overcome it's limitations.

This document covers basic terms and usage of Subversion. Also explains how to set up repository access to that repository, what Debian packages are needed, and finally how to convert an existing repository from CVS to Subversion without loss of information. The tutorial sections in this document are written for system administrators and developers with some experience in using CVS.

Subversion and CVS both are licensed under open source license. CVS is licensed under GNU GPL. Subversion is licensed under Apache-style license. Subversion depends on Berkeley DB, which in turn is dual-licensed open source and commercial. Commercial licenses are not required for usage.

1.1. Conclusions

Recommend changing version control system from CVS to Subversion. Any showstopper problems didn't arise. An existing repository can be converted from CVS to Subversion with process that is described elsewhere in this document. This conversion will not result in loss information, except for CVS revision numbering.

Changing CVS to Subversion requires the following work tasks.

- Install Subversion software to server and clients.
- Convert existing CVS repository to Subversion.
- Change in the documentation, website and possibly source code. These have usage instructions and references to e.g. CVS file locations.
- Change website update script and IRC channel bot notify script hooks.
- Developer training.

There are some advantages in changing to Subversion.

- Subversion *supports transactions*. Commits to the repository are processed as atomic transactions. Either all or nothing gets published.
- The Subversion *commands also work on directories*. Directories can be added, copied, moved or deleted. In CVS, directories are meant to be permanent, and changes need to be done in hand. This possibly leads to problems, for example when any checked out workspaces are invalidated.

- Subversion uses *binary diff algorithm*. Changes to binary files are versioned as well. Only the latest revision of a binary file gets stored inside a CVS repository.
- *Copying is a cheap*, constant time operation. The functional equivalent of tagging and branching is done by copying files within the repository.
- Subversion allows to attach *metadata properties* to files and paths.
- *Directories and properties are versioned* just like files.

The main concern in Subversion is that it uses Berkeley DB repository as a backend. The Berkeley DB database files are non-portable, and might break between version changes and different architectures. Next version of Subversion (release 1.1) will include FSFS, which is an alternative backend implementation. It uses ordinary filesystem directories to store files, even if those files are still in a binary-only format. See fsfs information page (http://web.mit.edu/ghudson/info/fsfs) for more information. CVS stores repository in an ordinary filesystem, with revision diff information stored in text format.

Chapter 2. Usage

The svn commands also work on directories. Directories can be added, copied, moved or deleted. Changes only take place after **svn commit**. Directories and properties are versioned just like files.

The Subversion repository is assigned a repository global revision number after each commit. Subversion revisions apply to the whole repository. Revision N represents the state of the repository after Nth commit. A file in the repository might be identical in revision N and N+1.

2.1. Accessing the repository

file://

Local file-based access. file:// access uses repository database files on the local host. Example checkout of *scratchbox* module trunk using file based access:

```
$ svn checkout file:///work/svn/scratchbox/trunk work-dir
```

Note: Subversion repository which uses Berkeley DB as a backend shouldn’t be accessed over a network filesystem.

http:// and https://

http:// or secure access over https://

Example checkout of *scratchbox* module trunk from scratchbox.org using http based access:

```
$ svn checkout http://scratchbox.org/repos/scratchbox/trunk work-dir
```

Apache 2.0 based Subversion server is based on WebDAV protocol, which is in turn based on http. This allows to use other programs than a Subversion aware client to access the repository. WebDAV is supported by many other programs, like file managers (Windows Explorer, Nautilus), office applications (Openoffice, Microsoft Office), etc. It’s also possible to go to repository URL with a browser and browse the repository contents as it appears on the latest revision.

svn://

Connects to a server running **svnserve** daemon using a custom protocol. Example checkout of *scratchbox* module trunk from scratchbox.org using anonymous svnserve access:

```
$ svn checkout svn://scratchbox.org/scratchbox/trunk work-dir
```

svn+ssh://

Connects to remote machine using **ssh** and then runs **svnserve** command on that machine. Checkout *scratchbox* module trunk from scratchbox.org over ssh tunnel:

```
$ svn checkout svn+ssh://scratchbox.org/work/svn/scratchbox/trunk work-dir
```

2.2. Repository layout

Before importing any projects, consider the repository layout. Subversion handles tags and branches as ordinary directories. Repository locations for those are a matter of convention.

- **/** is the repository root. Add new projects under here.
  - **project/** is a project module that contains everything that is related to a project.
    - **trunk/** is a project's main development line.
    - **tags/** contains snapshots of the main development line at a certain revision. Contents shouldn't be changed!
    - **branches/** contains various branches of the main development line.

Note: Tags location should only be used as a read-only snapshot. Tags are used to give a symbolic name to a project at a certain revision (example release). Because Subversion doesn't process **tags** any different than other directories, the files inside a tagged release can be changed. Please ensure that users will treat this location as should!

In CVS, tag and branch names cannot start with a digit or contain periods. This ends up up with branches example **SBOX_0_9_8**. As Subversion tags and branches are no different from directories, they can be named more freely, example **scratchbox-0.9.8**.

2.3. Tags and branches

Subversion has no concept of tagging or branching. These operations are substituted with copying files within the repository. Tags and branches exists as separate directories inside the repository and there is no difference between these operations. Repository locations for tags and branches are a matter of convention (see below for a suggested layout).

Making a copy using **svn copy** is a cheap, constant time operation. Subversion doesn't actually duplicate the data inside the repository. Example below tags trunk as Scratchbox 1.0 release location.

```
# svn copy http://scratchbox.org/repos/scratchbox/trunk \
          http://scratchbox.org/repos/scratchbox/tags/scratchbox-1.0 \
     -m "tagging scratchbox 1.0 release"
```

After the operation completes, the new repository location **/scratchbox/tags/scratchbox-1.0** remains as a snapshot copy of the project trunk **/scratchbox/trunk** at that time.

A copy can be also made from the basis of a revision number. This is specified by passing the parameter **-r N** to the **svn copy** command, where **N** is a revision number.

Chapter 3. Install

3.1. Prerequisites

Required Debian packages (version level).

- cvs2svn (0.1263)
- libsvn0 (1.0.5)
- subversion (1.0.5)

3.2. Backing up the repository

Subversion uses Berkeley DB as a backend database. Berkeley DB includes hot backup support that allows making a backup while the database is in use. Naively copying the files while the database is in use could result in an unusable backup. The problem with Berkeley DB files is that they are non-portable. Those files might break between version changes and different architectures.

A portable backup should be created using **svnadmin dump** command instead. Using this command, the current repository is saved into a portable dump file. The making of and restoration of the backups takes longer, as each commit is reproduced. Using this command it's also possible to make incremental dump files. Gzip compresses a dump to approx. one quarter of it's original size. Example **/etc/cron.daily/svnbackup** backup script follows. The script creates snapshot files svn-*date*-rev*revision*.dump.gz, where date is the current date in **yyyymmdd** format and **revision** the latest repository global version number. Create the target directory (here **/var/backups/svn**) before running this.

```
$ cat /etc/cron.daily/svnbackup
#!/bin/sh

SVN=/work/svn
BACKUP=/var/backups/svn

svnadmin dump ${SVN} | gzip
    > ${BACKUP}/svn-‘date +%Y%m%d‘-rev‘svnlook youngest ${SVN}‘.dump.gz
```

The portable dump file contents can be recovered using **svnadmin load** command. It reads contents from stdin and playbacks the original revisions to the repository as they appear in the dump. Example follows.

```
$ zcat /var/backups/svn/svn-20040823-rev2272.dump.gz | svnadmin load /work/svn
```

3.3. Providing access to the repository

Subversion's network access layer is modular. Access to the repository can be set up using different methods: local file based, custom svn protocol, custom svn protocol over ssh, and http including https. The repository can be accessed with multiple methods at the same time, given that every process can read and write the Berkeley DB files. Processes should also write to the database files using a friendly umask.

3.3.1. Browser access

If the repository is accessed using Apache WebDAV-based server, it is also possible to go to repository URL with a browser and browse the repository. Contents are shown as they appear on the latest revision. Older revisions cannot be viewed.

Third party software is required to browse the repository history and run diffs. ViewCVS is a Python CGI-script that was originally written to browse CVS repositories. The latest ViewCVS development version installed from source **vievcvs-1.0-dev** also supports browsing Subversion repositories. See ViewCVS homepage (http://viewcvs.sourceforge.net/) for more information.

Apache 2.0 module **mod_svn_view** provides, like ViewCVS, a web-based view of a Subversion repository. It is different from ViewCVS because it uses the Subversion libraries directly. It generates a simple XML output that can be run through XSL Transformations via mod_transform to generate a customized look. See mod_svn_view homepage (http://www.outoforder.cc/projects/apache/mod_svn_view/) for more information.

3.3.2. Access using svnserve custom protocol

Running **svnserve** in daemon mode provides an access to the repository. Clients connect using a custom protocol. Option **-r** restricts access to a repository root path.

```
# svnserve -d -r /work/svn/
```

Configurations file **/work/svn/conf/svnserve.conf** sets the user database file and access realms for authenticated and anomymous access. Usernames and passwords need to be typed in the users database file set with **password-db** directive. Authentication using e.g. PAM or LDAP is not possible. Passwords are not sent in plaintext over the network.

3.3.3. Authenticated access using ssh tunnel

Users will connect using SSH and run **svnserve** command.

All SSH users should have read and write permissions to the repository. Users will also require a sane umask when e.g. Berkeley DB logfiles are generated in repository database directory. The first **svnserve** lying on the $PATH should be a wrapper script that sets **umask 002** and then executes the real **svnserve** binary, for example as follows.

```
$ cat /usr/local/bin/svnserve
#!/bin/sh
umask 002
/usr/bin/svnserve "$@"
```

Access to individual modules inside the repository cannot be restricted in this access method.

3.3.4. Apache

Apache 2.0 can provide access to a repository over http or https using the module **mod_dav_svn**. Additional required Debian packages (version level) are as follows.

- apache2
- libapache2-svn (1.0.6)

Install the above package. Then enable Subversion modules **dav**, **dav_fs** and **dav_svn**. To enable a module, add symlinks to modules' **.load** and **.conf** files from **/etc/apache2/mods-available/** to **/etc/apache2/mods-enabled/** directory.

Example access with no restrictions, so that an anonymous user would have full rights. The **Location** block sets urls that begin with **http://scratchbox.org/repos/** to be served with **DAV** module mod_dav_**svn**. Subversion repository **SVNPath** is kept in **/work/svn**.

```
<Location /repos>
DAV svn
SVNPath /work/svn
</Location>
```

There are several ways to set up access restrictions to repository location. It is possible to use any authentication modules that Apache provides, for example basic authentication, LDAP or host IP. Add the following lines to **Location** block to add basic authentication. Also, create the users passwords file **/etc/apache2/svn.passwd** with **htpasswd** tool.

```
AuthType Basic
AuthName "Subversion Repository"
AuthUserFile /etc/apache2/svn.passwd
Require valid-user
```

Users can be authenticate with LDAP authentication using **mod_auth_ldap**.

```
AuthType Basic
```

```
AuthName "Subversion repository"
AuthLDAPURL ldap://ldap.scratchbox.org/dc=scratchbox,dc=org?uid?sub
Require valid-user
```

Users can be authenticated by LDAP group authentication as well.

```
Require group cn=developers,ou=groups,dc=scratchbox,dc=org
```

To allow anonymous chekcout access to the repository, add the following line to **Location** block before **Require**. First tries to authenticate the user using basic authentication, then falls back to anonymous access.

```
Satisfy any
```

Apache module **mod_authz_svn** allows to define access restrictions on a directory basis. For this, specify access policy control file with **AuthzSVNAccessFile** directive in **Location** block.

```
AuthzSVNAccessFile /etc/apache2/dav_svn.authz
```

Also create a subversion access policy control file to **/etc/apache2/dav_svn.authz**, as follows. User groups have to be provided in this file, searching them from LDAP is not supported.

```
[groups]
developers = crash,test,dummy

[/]
# give read access for all
* = r
# read/write to developers group
@developers = rw

[/project/trunk]
# read/write to luser
luser = rw
```

A complete configuration example follows. This example sets up authenticated access using basic authentication, path-based restrictions and allow anonymous checkouts.

```
<Location /repos>
DAV svn
SVNPath /work/svn

AuthType Basic
AuthName "Subversion Repository"
AuthUserFile /etc/apache2/svn.passwd

AuthzSVNAccessFile /etc/apache2/dav_svn.authz

Satisfy any
```

```
Require valid-user
</Location>
```

Repository access does not show up in Apache access log.

3.3.5. WebDAV autoversioning

Subversion includes autoversioning support for WebDAV shares. This allows clients to access a WebDAV share while the server is transparently versioning any changes. This option is set with **SVNAutoversioning** directive in Apache location block. Unfortunately, autoversioning interoperability support was not yet stable enough for real usage. The autoversioning support was tested against OpenOffice 1.1 and Microsoft Word 2000.

Chapter 4. Conversion

Follow this procedure to convert an existing CVS repository to Subversion.

4.1. Backing up CVS repository

Backup the CVS repository before proceeding. It also might be a good idea to remove write permissions from **CVSROOT** so that no one will make a commit while the conversion is unfinished.

4.2. Create the repository

Run the following command to create a repository at **/work/svn**. Do not create the repository on a network filesystem. It might work, but more likely might corrupt the repository. Berkeley DB uses direct mmap and requires strict POSIX locking semantics.

```
# svnadmin create /work/svn
```

The directory **/work/svn/db** now contains a Berkeley DB environment. Repository users, also **www-data**, should have write permissions there. New logfiles that are generated in this directory need to be owned by the group as well. Write permissiosn are required also for read-only access. Access restrictions will be taken care of in an upper level (Apache server).

```
# chmod 2775 /work/svn/db /work/svn/dav
# chmod 0775 /work/svn/db/*
```

4.3. Importing an existing CVS repository

There are different ways to migrate an existing CVS repository to Subversion. This example uses **cvs2svn** for a full conversion of trunk, tags and branches. The conversion also includes historical data on the files.

As of version 0.1263-1, **cvs2svn** tool doesn't support multicomponent paths, with no plans to fix this [1]. Use the following script as a workaround to this problem. Essentially, it first uses **cvs2svn** script to convert a module, and then re-orders the layout. [2]. Note that the procedure might be broken if file names include spaces.

```
#!/bin/sh

MODULE=$1
SVN=/work/svn
```

```
CVS=/work/cvs/scratchbox
MSG="re-order repository layout"

# Convert from CVS to SVN using cvs2svn
cvs2svn --existing-svnrepos -s $SVN $CVS/$MODULE

# Then re-order repository layout
svn mkdir file://$SVN/$MODULE -m "$MSG"
svn move file://$SVN/branches file://$SVN/$MODULE/branches -m "$MSG"
svn move file://$SVN/tags file://$SVN/$MODULE/tags -m "$MSG"
svn move file://$SVN/trunk file://$SVN/$MODULE/trunk -m "$MSG"
```

Run the above script to import all CVS modules to Subversion.

```
$ for module in crocodile fakeroot-net sb-toolchains scratchbox \
    scratchbox-libs www chroot-uid ee linux-headers sbrsh \
    scratchbox-base scratchbox-utils ; do
        cvs2svn.sh $module ;
done
```

4.4. Importing file properties

The conversion script does not convert all of the file properties properly. A checkout of the modules is needed to change the properties. If disk space is at low, then the modules can be processed one at a time. Change the working directory to a drive with enough space for holding copies of all trunks, tags and branches. Checkout everything from the repository to a local working copy.

Note: Because of how Subversion handles metadata, the imported properties don't apply retrospectively. Metadata is versioned over time, just like the file contents, and there is no easy way to change the history. Properties will though persist to the later revisions of files.

```
$ cd /tmp
$ svn checkout file:///work/svn/
```

Note: Running the above checkout command takes some time. Go and have a cup of coffee.

Now change the working directory to the local copy.

```
$ cd svn
```

Filenames and filename patterns that should not be attempted to be stored in the repository can be set by attaching **svn:ignore** property to a directory. The files that are listed in that property are also ignored when displaying informational messages. Setting this property has the same effect that **.cvsignore** file in CVS. Run the following command to replace **.cvsignore** files with **svn:ignore** properties.

> **Note:** Subversion doesn't support special filename **!** lines like **.cvsignore** files.

```
$ find -name .cvsignore | while read file ; do
    svn propset svn:ignore -F $file "‘dirname $file‘" ;
    svn delete -m "replaced .cvsignore with svn:ignore" $file
done
```

Keywords like $Author$, $Date$, Id, etc. are not automatically expanded in Subversion like they are in CVS. Property **svn:keywords** needs to be set for the completion to take place. The following command will grep for Author, Date and Id keywords and add relevant keyword expansion properties for those files. Any existing keywords are lost.

> **Note:** Keyword expansion could damage binary files if they contain text that can be interpreted as keywords.

```
$ find . -type f -a ’(’ -path ’*/.*’ -prune -o -print ’)’ | while read file ; do
    plist=""
    for p in Author Date Id Rev ; do
        if grep -q ’\$’${p}’:’ "$file" ; then
            plist="${p} $plist"
        fi
    done
    if [ "$plist" != "" ] ; then
        svn propset svn:keywords \"${plist%% }\" $file
    fi
done
```

Subversion uses executable property to control exec bit set on repository files. This should be set for e.g. build scripts stored in the repository. Run the following command to search for files with exec bits set from the CVS directories, then attach **svn:executable** properties to those files in the Subversion side. Note that the script sets those bits only only starting from the current revision number.

```
$ for module in crocodile fakeroot-net sb-toolchains scratchbox \
    scratchbox-libs www chroot-uid ee linux-headers sbrsh \
    scratchbox-base scratchbox-utils ; do
        find /work/cvs/scratchbox/$module -type f -perm +1 | grep -v Attic | while read f ;
            f=${f/\/work\/cvs\/scratchbox\/$module\//}
            f=${f%%,v}
            svn propset svn:executable ON $module/trunk/$f \
                $module/tags/*/$f $module/branches/*/$f
    done
done
```

Now commit the changes made

```
$ svn commit -m "imported svn:ignore, svn:keywords and svn:executable properties"
```

This completes repository import from CVS.

4.5. Cleaning up

This step is not necessary but might be convenient. CVS branch names cannot start with a digit or contain periods. ending up with branches for example called v1_2_3 instead of 1.2.3. Existing branches and tags may be renamed to be more verbose.

Notes

1. See related bug report (http://cvs2svn.tigris.org/issues/show_bug.cgi?id=7)
2. Based on cvs2svn.sh (http://cvs2svn.tigris.org/nonav/issues/showattachment.cgi/8/cvs2svn.sh). Changed REPOS parameter to repository install path (here **/work/svn**), added CVS parameter to CVS root path (here **/work/cvs/scratchbox**) and also added **existing-svnrepos** parameter to **cvs2svn** call

Appendix A. Basic usage of the svn client

Subversion command-line client **svn** interface is modelled after **cvs**. See also **svn help** for more commands and options.

A.1. Checkout files

svn checkout URL [PATH]

Checkout a local copy from the repository.

```
$ svn checkout https://scratchbox.org/repos/scratchbox/trunk work-dir
```

Note: Checkout the trunk of a project, and not the project itself. If you check out a complete project, your local copy contains a copy of every branch and tag.

A.2. Adding and removing content

svn add PATH

Adds a local path to the repository.

svn import PATH URL

Recursively import a copy of path to repository.

```
$ svn import README https://scratchbox.org/repos/scratchbox/trunk
```

svn delete PATH
svn delete URL

Delete a file or a directory from a local copy or from the repository.

A.3. Tagging and branching

svn copy SOURCE DEST

Copies a file or a directory in local copy or inside the repository. Tags and branches are also produced with this command. Making a copy is a cheap, constant time operation. Example below tags the project trunk.

```
$ svn copy https://scratchbox.org/repos/scratchbox/trunk \
   https://scratchbox.org/repos/scratchbox/tags/project-1.0 \
                  -m "tagging project 1.0 release"
```

A.4. Merging between revisions

Merge applies the differences between two revisions. It results only in modifications to a local working copy.

svn merge -r N:M SOURCE [PATH]

SOURCE can be a URL or a local working copy item. Revisions N and M are the ones to be compared.

To merge changes that have taken place in a branch, compare the initial branch state to the current branch state. Use svn log to see when the branch was created in (example revision 1337). The current branch state is always the HEAD revision.

```
$ svn merge -r 1337:HEAD \
        https://scratchbox.org/repos/scratchbox/branches/my-branch
```

svn merge URL1[@N] URL2[@M] [DEST]

Applies the differences between the two source URLs to the local working copy.

A.5. Updating working copy

svn update [PATH]

Updates local copy.

svn status [PATH]]

Prints out all files that have local modifications. Subversion keeps a cache of the original files that are used for comparison. The repository is not connected.

A.6. Committing changes

svn commit [PATH]

Send local changes to the repository. Commit is processed as an atomic transaction (either all or nothing gets published).

A.7. Properties

Subversion supports versioned metadata properties to be attached to files and paths. Properties are set using svn propset command. Perhaps there could be a policy to set **license** property on certain files.

```
$ svn propset license ’GPL’ foo.c
```

Subversion has reserved namespace of special properties starting ’svn:’. For example, **svn:executable** property is used to control exec bit set on repository files. This should be set for e.g. build scripts stored in the repository.

```
$ svn propset svn:executable ON configure
```

Filenames and filename patterns that should not be attempted to be stored in the repository can be set by attaching **svn:ignore** property to a directory. The files that are listed in that property are also ignored when displaying informational messages. Setting this property has the same effect that **.cvsignore** file in CVS.

```
$ svn propset svn:ignore "foo.o" .
```

Keywords like $Author$, $Date$, Id, etc. are not automatically expanded in Subversion like they are in CVS. Property **svn:keywords** needs to be set for the completion to take place.

```
$ svn propset svn:keywords Id foo.c
```

See [2, chapter 7, section 2] for a complete list of special properties.

Bibliography

[1] *cvs2svn project homepage (http://cvs2svn.tigris.org/)* .

[2] *Version Control with Subversion (http://svnbook.red-bean.com/svnbook/index.html) Ben Collins-Sussman Brian W. Fitzpatrick C. Michael Pilato* .