svn-buildpackage - maintaining Debian packages with Subversion

Eduard Bloch

$LastChangedDate: 2005-03-29 03:19:19 +0200 (Di, 29 MÃ¤r 2005) $

Copyright Notice

See ‘Copyright’ on page 19 for details.

Contents

1 Introduction **1**

1.1 Purpose . . . 1

1.2 Why a version control system? . . . 1

1.3 Features . . . 2

1.4 Contents overview . . . 2

1.5 Popular repository layouts . . . 3

2 Getting started **5**

2.1 Quick guide . . . 5

2.2 Basic svn usage . . . 5

2.3 Creating Subversion repository . . . 6

2.4 Using by multiple developers . . . 6

2.4.1 SVN over SSH . . . 7

2.4.2 Anonymous access . . . 8

3 Importing Debian packages **9**

3.1 Importing from existing source package files . . . 9

3.2 On-Build-Time merging . . . 10

4 Common tasks **11**

4.1 Checkout . . . 11

4.2 Building the package . . . 11

4.3 Working with source . . . 11

4.4 Handling new upstream versions . . . 12

4.5 Finalizing the Revision . . . 12

5 Command reference **13**

5.1 svn-inject . . . 13

5.1.1 NAME . . . 13

5.1.2 SYNOPSIS . . . 13

5.1.3 OPTIONS . . . 13

5.2 svn-buildpackage . . . 14

5.2.1 NAME . . . 14

5.2.2 SYNOPSIS . . . 14

5.2.3 DESCRIPTION . . . 14

5.2.4 OPTIONS . . . 14

5.2.5 CONFIGURATION FILE . . . 15

5.2.6 DIRECTORY LAYOUT HANDLING . . . 16

5.2.7 ENVIRONMENT VARIABLES . . . 16

5.2.8 RECOMMENDATIONS . . . 17

5.3 svn-upgrade . . . 17

5.3.1 NAME . . . 17

5.3.2 SYNOPSIS . . . 17

5.3.3 DESCRIPTION . . . 17

6 Further documentation **19**

6.1 Various links . . . 19

6.2 Copyright . . . 19

Chapter 1

Introduction

1.1 Purpose

This short document is only intended to give a short help in converting packages to Subversion management. It is primarily intended for developers not really familiar with Subversion or CVS management and/or converting from maintaining their packages using common tools (dpkg-dev, devscripts) only to version control system Subversion.

1.2 Why a version control system?

But the first question may be: why use a version control system at all? Look at how the source is handled by the Debian package. First, we have the pure upstream source, which is often maintained by another person. The upstream author has his own development line and releases the source in snapshots (often called releases or program versions).

The Debian maintainer adds an own set of modifications, leading to an own version of the upstream package. The difference set between this two version finally ends in Debian's .diff.gz files, and this patchset is often appliable to future upstream versions in order to get the "Debian versions".

So the obvious way to deal with source upgrades/changes is using local copies, patch, different patchutils and scripts to automate all this, eg. uupdate. However, it often becomes nasty and uncomfortable, and there is no way to undo changes that you may do by mistakes.

At this point, the Subversion system can be used to simplify that work. It does the same things that you normaly would do by-hand but keeps it in an own archive (a repository). It stores the development lines of Upstream and Debian source, keeping them in different directories (different branches). The branches are wired internally (the VCS "knows" the history of the file and tracks the differences between the Upstream and Debian versions). When a new upstream version is installed, the differences between the old and new upstream versions and the Debian version are merged together.

You can create snapshots of your Debian version ("tag" it) and switch back to a previous state, or see the changes done in the files. You can store when commiting the file to the repository or place custom tags on the files ("properties") serving various purposes.

1.3 Features

svn-buildpackage and other scripts around it has been created to do follow things:

- keep Debian package under revision control, which means storing different versions of files in a Subversion repository
- allow easy walking back trough time using svn command
- easy retrieval of past versions
- keep track of upstream source versions and modified Debian versions
- easy installation of new upstream versions, merging the Debian changes into it when needed (similar to the uupdate program)
- automated package building in clean environment, notifying about uncommited changes
- create version tags when requested to do the final build and update changelog when needed
- allow co-work of multiple Debian developers on the same project
- auto-configure the repository layout, making it easy to use by people without knowing much about Subversion usage (mostly you need only the add, rm and mv commands of svn)
- allow to store only the Debian specific changes in the repository and merge them into the upstream source in the build area (which nicely completes build systems like dpatch or dbs)
- If wished, keep the upstream tarballs inside of the repository

1.4 Contents overview

There are currently three scripts provided by the svn-buildpackage package:

- svn-inject: script used to insert an existing Debian package into a Subversion repository, creating the repository layout as needed.
- svn-buildpackage: exports the contents of the directory associated with the starting directory from the Subversion repository to the clean environment and build the package there

- svn-upgrade: similar to uupdate, upgrades the trunk to a new upstream version, preserving and merging Debian specific changes

1.5 Popular repository layouts

There are different ways to store the packages in the repositories (or in multiple repositories at your choice). `svn-buildpackage` normaly expects a directory structure similar to the one well described in the Subversion Book, which looks like:

```
packageA/
   trunk/
   branches/
   branches/upstream
   tags/

projectB/
   trunk/
   branches/
   branches/developerFoo
   tags/
```

packageA above may be a typical upstream-based source package and a projectB may be a Debian native package with a separate branch created by developerFoo for his own experiments. See Subversion Book/Branches (`http://svnbook.red-bean.com/html-chunk/ch04s02.html`) for more details about using Subversion branches.

Also note that Tags work quite different than those in CVS. Subversion does not maintain magic tags associated with some files. Instead, it tracks the file state and moves, so Tagging something means creating a copy (inside of the Repository, harddisk-space efficient) of a certain version of the file set. So the Debian branch of the released package source is contained in `trunk/` and is tagged by copying (mirroring) the trunk tree to `tags/DEBIAN-REVISION`. The same happens for the upstream releases. In addition, the most recent upstream version is mirrored to `branches/upstream/current`. After few package upgrade cycles, the directory tree may look so:

```
# svn ls -R file:///home/user/svn-repo/dev/translucency
branches/
branches/upstream/
branches/upstream/0.5.9/
branches/upstream/0.5.9/AUTHORS
branches/upstream/0.5.9/COPYING
...
branches/upstream/0.6.0/
branches/upstream/0.6.0/AUTHORS
```

```
branches/upstream/0.6.0/COPYING
...
branches/upstream/current/
branches/upstream/current/AUTHORS
branches/upstream/current/COPYING
... same stuff as in 0.6.0 ...
tags/
tags/0.5.9-1/
...
tags/0.5.9-1/debian/
tags/0.5.9-1/debian/README.Debian
...
tags/0.6.0-1/
tags/0.6.0-1/AUTHORS
...
tags/0.6.0-1/debian/
tags/0.6.0-1/debian/README.Debian
tags/0.6.0-1/debian/changelog
...
trunk/
trunk/AUTHORS
trunk/COPYING
... trunk where 0.6.0-2 is beeing prepared ...
```

svn-buildpackage also supports the second repository layout suggested in the Subversion Book (function/package) but svn-inject prefers the one documented above. Both svn-buildpackage and svn-upgrade should be able to auto-detect the repository layout and the location of package files.

In theory, you do not have to follow that examples and place the trunk, branches and tags directory on the locations you like more. But svn-buildpackage and other scripts won't locate the files automaticaly so you will need to edit the .svn/deb-layout file in your working directory and set paths. See the old abstract (`file:///usr/share/doc/svn-buildpackage/CONFIG`) about how auto-detection works and the config example (`file:///usr/share/doc/svn-buildpackage/examples/config.example`).

Finaly, the working directory structure on your development system may look so:

```
dev/ # base directory, may be under version control or not
dev/foo # trunk directories of various packages
dev/bar # contents correspond to trunk, see above
dev/tarballs # where "orig" tarballs are stored, may be under VC or not
dev/build-area # where the packages are exported temporarily and built
```

Chapter 2

Getting started

Besides of the packages that are installed by dependencies when you install `svn-buildpackage`, you may need `ssh` and the obligatory tool chain: `dpkg-dev`, `build-essential` and all the packages they pull into the system.

2.1 Quick guide

Here is a quick guide for those who wish to build an existing package using an existing, public available SVN repository. To create own repositories, skip this section and look for more details below.

- svn co <svn://server/path/to/trunk> package
- mkdir tarballs
- cp dir-where-you-keep-the-source/package_version.orig.tar.gz tarballs/

  NOTE: you need the upstream source tarballs, stored under a usual dpkg-source-compatible filename in tarballs/
- cd package
- svn-buildpackage -us -uc -rfakeroot

2.2 Basic svn usage

You need only few commands to start using svn with svn-buildpackage scripts. If you wish to learn more about it, read parts of the the Subversion Book (`http://svnbook.red-bean.com/html-chunk/`). The most used commands are:

- `add` – put new files unto the revision contol

- `rm` – remove the files from the repository
- `mv` – move files around, leting revision control system know about it
- `commit` – commit your changes to the repository
- `resolved` – tell svn that you have resolved a conflict
- `diff` – creates a "diff -u" between two versions, specified by file revision number or by date. See the `diff --help` output.
- `cat -r Revision` – useful to browse in some previous revision of the file

If you are familiar with CVS you will probably know almost all you need.

2.3 Creating Subversion repository

The main Subversion repository is easily created with:

```
svnadmin create repo-directory
```

For our example, we choose the name `svn-deb-repo` and put it in `/home/user`.

If you plan to keep many packages in the one repository including upstream tarballs, consider to put it on a hard disk with much free space and good performance (especially short disk access times) since the repository will grow and the filesystem may become fragmented over time.

2.4 Using by multiple developers

Multiple developers with local access to the repository may share it using a common group. To do so, create a new group and add all developers to it. Run "chgrp -R sharedGroup repdir ; chmod -R g+s repdir" for the shared group and the repository directory. Now, on local access to this repository everybody will create files with the appropriate group setting. However, the developers will need to set a liberal umask before using svn (like "0022").

If somebody resists to do so, there is still a brute-force solution: fix the permissions with a post-commit script. However, this is an "unsound" solution and may lead to ALL KINDS OF PROBLEMS. MAKE SURE THAT YOU ARE AWARE OF THE POSSIBLE CONSEQUENCES BEFORE YOU OPEN THE PANDORA BOX. See Debian BTS (`http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=240630`) for details. When you damage your repository, don't blame me and remember that there is "svnadmin recover".

```
#!/bin/sh

# POST-COMMIT HOOK
# The following corrects the permissions of the repository files

REPOS="$1"
REV="$2"

chgrp -R sharedGroup $REPOS
# replace sharedGroup with your group
chmod -R g+r $REPOS
chmod -R g+w $REPOS
```

2.4.1 SVN over SSH

To run Subversion over SSH, you basically need a shell on the target system and a subversion repository located there which is created following the description above. The repository must be configured for access by the system users of the remote system.

Assuming that your user name on the client system is the same as on the server side, there is not much to configure. Just change the protocol specificaton from `file://` to `svn+ssh://remoteusername@server-hostname` in all examples showed in this manual.

If you wish to use fsh over ssh (what I recommend because it is a significant speed-up with repeating operations), you will specify a custom transport method in subverions's configuration. To do so, edit the file `~/.subversion/config` and add the section `[tunnels]` to it, following by your custom transport definition. Example:

```
# personal subversion config with custom ssh tunnel command
[tunnels]
# SSH account on svn.d.o
# compression is enabled in the ssh config
deb = fsh -l blade
# SSH account for NQ intranet, set fix username
nq = ssh -C -l zomb
```

You can use the new defined tunnels in a similar ways as described above but replace `svn+ssh` with `svn+tunnelname`, so the final URL looks like:

```
svn+deb://svn.debian.org/svn/myproject/ourpackage/trunk
```

.

2.4.2 Anonymous access

You can allow outsiders to have anonymous (read-only) access using the svnserve program, as described in the Subversion documentation.

Another method is using HTTP/WebDAV with Apache2. More about a such setup can be found in the Subversion Book and the SubversionApache2SSL Howto (http://wiki.debian.net/?SubversionApache2SSLHowto). svn.debian.org (http://svn.debian.org/) is an example site granting anonymous access to some selected projects hosted there.

Chapter 3

Importing Debian packages

3.1 Importing from existing source package files

The `svn-inject` utility is intended to import already packaged source packages into a new subdirectory of the repository, creating the repository layout as needed. Normally, it takes two arguments: the .dsc file of your package and the base URL of the Subversion repository.

```
svn-inject translucency_*dsc file:///tmp/z
cp /tmp/translucency_0.6.0.orig.tar.gz /tmp/tarballs || true
mkdir -p translucency/branches/upstream
tar  -z -x -f /tmp/translucency_0.6.0.orig.tar.gz
mv * current
svn -q import -m"Installing original source version" translucency file:///tmp
svn -m Tagging upstream source version copy file:///tmp/z/translucency/branch
upstream/current file:///tmp/z/translucency/branches/upstream/0.6.0 -q
svn -m Forking to Trunk copy file:///tmp/z/translucency/branches/upstream/cur
dpkg-source -x /tmp/translucency_0.6.0-1.dsc
dpkg-source: extracting translucency in translucency-0.6.0
svn_load_dirs file:///tmp/z/translucency/trunk . *
...
Running /usr/bin/svn propset svn:executable  initscript
Running /usr/bin/svn propset svn:executable  debian/rules
Running /usr/bin/svn propset svn:executable  mounttest.sh
Running /usr/bin/svn propset svn:executable  mount.translucency
Running /usr/bin/svn propget svn:eol-style base.h
Running /usr/bin/svn propget svn:eol-style Makefile
Running /usr/bin/svn propget svn:eol-style translucency.8
Running /usr/bin/svn commit -m Load translucency-0.6.0 into translucency/trun

Running /usr/bin/svn update
Cleaning up /tmp/svn_load_dirs_jD7OenzVjI
```

```
Storing trunk copy in /tmp/translucency.
svn co file:///tmp/z/translucency/trunk /tmp/translucency -q
svn propset svn:executable 1 debian/rules -q
svn -m"Fixing debian/rules permissions" commit debian -q
Done! Removing tempdir.
Your working directory is /tmp/translucency - have fun!
```

If you omit the URL, `svn-inject` will try to use the URL of the current directory as base URL. I would not rely on this, however.

3.2 On-Build-Time merging

A special feature of svn-buildpackage is so called mergeWithUpstream-mode. Many projects do not want to keep the whole upstream source under revision control, eg. because of the large amount of required disc space and process time. Sometimes it makes sense to keep only the `debian/` directory any maybe few other files under revision control.

The task of exporting the source from repository and adding it to the upstream source before building becomes annoying the time. But the `svn-buildpackage` tools automate most of this work for you: they switch to so called mergeWithUpstream-mode if a special flag has been detected: the mergeWithUpstream (Subversion) property of the `debian` directory. `svn-buildpackage` will merge the trunk with upstream source on build time and `svn-upgrade` will only update the changed files in this case.

To enable this feature during the inital import of the source package, simply add the `-o` switch to the `svn-inject` call and it will prepare the source for with mergeWithUpstream-mode: reduce the set of files to those modified for Debian and set the mergeWithUpstream property.

But what, if you decide to switch to mergeWithUpstream-mode after the package has been injected? To do this, checkout the whole repository, remove the files not changed in the Debian package from both upstream source and Debian branch (`svn rm`) and set the mergeWithUpstream property in the `trunk` directory with `svn propset mergeWithUpstream 1`.

If you actually decide to stop using the mergeWithUpstream-mode, unset the mergeWithUpstream property as follows: `svn propdel mergeWithUpstream debian/`.

Chapter 4

Common tasks

4.1 Checkout

svn-inject will do the initial checkout for you. If you need another working copy, run

```
svn co protocol://repository-base-url/yourpackage
```

4.2 Building the package

Change to your trunk directory and run:

```
svn-buildpackage -us -uc -rfakeroot
```

You may recognice the options above – they are passed directly to the build command (`dpkg-buildpackage` by default). Normaly, the build is done in another directory (exporting the source with cp-la-like method). If you wish the resulting packages to be placed in the directory above, use the `--svn-move` option. To run Lintian after the build, use `--svn-lintian` option. More options are described in the manpage ('svn-buildpackage' on page 14).

4.3 Working with source

Every time when you add or modify something, `svn-buildpackage` won't let you proceed unless suspicious files are in the clean state (unless you use the `--svn-ignore` switch). You use the commands described in 'Basic svn usage' on page 5 to register the new files (or move or delete the old ones) and commit the changes to the repository.

4.4 Handling new upstream versions

Upgrading with new upstream version normaly happens in two steps:

- the `current` tree in the upstream branch is upgraded with the source from the new upstream package (the old version is kept in repository in `branches/upstream/oldVersion`).
- The version in `trunk/` becomes upgraded by merging the changes between the upstream versions into the `trunk/` directory.

The script `svn-upgrade` (formerly `svn-uupdate`) does both things for you and also creates a new changelog entry. The first step is done internaly by using a third party script (svn_load_dirs, see Subversion book for documentation), the second step is done with the merge command of svn. Just run `svn-upgrade` from you local working directory (which corresponds the `trunk/` checkout product).

After running `svn-upgrade` some files may be in conflicting state. This is naturaly happens if you have modified some files in the upstream package and now upstream did something similar on the same positions so `svn merge` was confused.

When `svn-upgrade` complains about files in conflicting state, fix them manualy. When done, use the `svn resolved` command to mark them as clean and `svn commit` to update the repository.

4.5 Finalizing the Revision

When you are ready to upload a new revision of your package, everything builds fine, the changelog is cleaned up and the package is tested, you can do the final build and tag the end version. To do so, add `--svn-tag` switch and after the package is built, it will be tagged (by creating a copy of the `trunk/` directory as said above).

Chapter 5

Command reference

5.1 svn-inject

5.1.1 NAME

svn-inject - puts a Debian source package into Subversion repository

5.1.2 SYNOPSIS

svn-inject [*options*] <*package*>.dsc <*repository URL*>

5.1.3 OPTIONS

-h print this message

-v Make the command verbose

-q Hide less important messages.

-l Layout type. 1 (default) means package/{trunk,tags,branches,...} scheme, 2 means the {trunk,tags,branches,...}/package scheme. 2 is not implemented yet.

-t directory Specify the directory where the .orig.tar.gz files are stored on the local machine.

-d , –do-like=directory Looks at the working directory of some other package and uses its base URL, tarball storage directory and similar checkout target directory.

-c number Checkout nothing (0), trunk directory (1) or everything (2) when the work is done.

-o Put only files that are actually touched in the .diff file under the version control. svn-inject sets the mergeOnUpstream property automaticaly, see HOWTO.

5.2 svn-buildpackage

5.2.1 NAME

svn-buildpackage - build Debian packages from SVN repository

5.2.2 SYNOPSIS

svn-buildpackage [*OPTIONS ...*] [*OPTIONS for dpkg-buildpackage*]

5.2.3 DESCRIPTION

Builds Debian package within the SVN repository. The source code repository must be in the format created by svn-inject, and this script must be executed from the work directory (trunk/package).

By default, the working directory is used as the main source directory (assuming the whole upstream source is beeing stored in the repository). The alternative is so called "merge mode". With this method, only the debian directory (and maybe some other modified files) are stored in the repository. At build time, the contents of the svn trunk are copied to the extracted tarball contents (and can overwrite parts of it). To choose this working model, set the svn property mergeWithUpstream on the Debian directory ("svn propset mergeWithUpstream 1 debian").

The default behaviour is following: Check the working directory, complain on uncommited files (also see –svn-ignore-new) Copy the orig tarball to the build area if neccessary (also see –svn-no-links) Extract the tarball (in merge mode) or export the svn work directory to the build directory (also see below and –svn-no-links) Build with dpkg-buildpackage (also see –svn-builder, –svn-lintian, etc.) Create a changelog entry for the future version

5.2.4 OPTIONS

-h , –help Show the help message

–svn-verbose More verbose program output

–svn-dont-clean Don't run debian/rules clean (default: clean first)

–svn-dont-purge Don't run remove the build directory when the build is done. (Default: wipe after successfull build)

–svn-export Just export the working directory and do neccessary code merge operations, then exit.

–svn-no-links Don't use file links but try to export or do hard copies of the working directory (default: use links where possible). This is useful if your package fails to build because some files, empty directories, broken links, ... cannot not be transported with in the default link-copy mode.

–svn-ignore-new Don't stop on svn conflicts or new/changed files

–svn-tag Final build: Tag, export, build cleanly & make new changelog entry

–svn-tag-only, –svn-only-tag Don't build the package, do only the tag copy

–svn-retag If an existing target directory has been found while trying to create the tag copy, remove the target directory first.

–svn-lintian Run lintian in the build area when done

–svn-move When done, move the created files (as listed in .changes) to the parent directory, relativ to the one where svn-buildpackage was started.

–svn-move-to =... Specifies the target directory to move generated files to.

–svn-pkg =packagename Overrides the detected package name, use with caution. May be set too late during the processing (ie. still have the old value when expanding shell variables).

–svn-override =var=value,anothervar=value Overrides any config variable that has been autodetected or found in .svn/deb-layout.

–svn-builder =COMMAND Specifies alternative build command instead of dpkg-buildpackage, eg. debuild, pdebuild... WARNING: shell quotation rules do not completely apply here, better use wrappers for complex constructs. Using this option may break –svn-lintian and –svn-move functionality.

–svn-pass-diff Experimental function to generate the .diff.gz contents using svn and pass it to dpkg-buildpackage. Requires a hacked dpkg-buildpackage script.

–svn-prebuild, –svn-postbuild, –svn-pretag, –svn-posttag Commands (hooks) to be executed before/after the build/tag command invocations. Examples to fetch the orig tarball from a local pool (trying pool/libX/... to):

```
svn-buildpackage --svn-prebuild='a="wget -c http://mymirror/debian/main/pool/

Multiple retries with a bashism:

svn-buildpackage --svn-prebuild='wget -c http://mymirror/debian/main/pool/{'e

Or using a bounty, see below...

svn-b --svn-prebuild="wget http://mymirror/debian/main/pool/$guess_loc"
```

5.2.5 CONFIGURATION FILE

svn-buildpackage's behaviour can be modified using the file *~/.svn-buildpackage.conf* . It is basicaly a list of the long command line options (without leading minus signs), one argument per line (but without quotes embrassing multi-word arguments). The variables are expanded

with the system shell if potential shell variables were found there. Avoid ~ sign because of unreliable expansion, better use $HOME instead. Example:

```
svn-builder=debuild -EPATH

svn-no-links

svn-override=origDir=$HOME/debian/upstream/$PACKAGE

# svn-ignore-new

#svn-lintian
```

5.2.6 DIRECTORY LAYOUT HANDLING

By default, svn-buildpackage expects a configuration file with path/url declaration, .svn/deb-layout. The values there can be overriden with the *–svn-override* option, see above. If a config file could not be found, the settings are autodetected following the usual assumptions about local directories and repository layout. In addition, the contents of a custom file debian/svn-deblayout will be imported during the initial configuration. Package maintainers can store this file in the repository to pass correct defaults to new svn-buildpackage users. The format is the same as in the file .svn/deb-layout.

5.2.7 ENVIRONMENT VARIABLES

Following environment variables are exported by *svn-buildpackage* and can be used in hook commands or the package build system.

PACKAGE , *package* The source package name

SVN_BUILDPACKAGE Version of svn-buildpackage

TAG_VERSION , *debian_version* The complete Debian version string, also used for the tag copy

non_epoch_version Same as debian_version but without any epoch strings

upstream_version Same as debian_version but without Debian extensions

guess_loc Guessed upstream source package name in the pool, something like libm/libmeta-html-perl_3.2.1.0.orig.tar.gz

DIFFSRC (experimental) shows the location of generated diff file Following variables are understood by svn-buildpackage:

FORCETAG Tells to ignore the signs for an incomplete changelog

FORCEEXPORT Export upstream source from the repository even if mergeWithUpstream property is set

5.2.8 RECOMMENDATIONS

First thing, using shell aliases makes sense. Example for the Bash:

```
alias svn-b="svn-buildpackage -us -uc -rfakeroot --svn-ignore"
alias svn-br="svn-b --svn-dont-purge --svn-reuse"
alias svn-bt="svn-buildpackage --svn-tag -rfakeroot"
```

That simply has the meaning: "just build no matter what", the same without re-exporting on repeated builds and "build for upload and tag". To access remote repositories, the shell access (via ssh) is the easiest way to create the link. However, it requires to enter the password and this happens more often with svn-buildpackage. There are workarounds: use ssh key without passphrase (insecure and still slow connection) or use the fsh tool which keeps a persistent connection in background and behaves like a rsh/ssh command. For details, see svn-buildpackage manual. Another way to get a remote link is using the Subversion DAV module (with SSL and Apache user authentication), see svn-buildpackage HOWTO manual for details.

5.3 svn-upgrade

5.3.1 NAME

svn-upgrade - upgrade source package from a new upstream revision

5.3.2 SYNOPSIS

svn-upgrade *newtarball* [*OPTIONS* ...]

5.3.3 DESCRIPTION

svn-upgrade modifies a Debian package source located in a Subversion repository, upgrading it to a new upstream release. The repository filesystem tree must be in the format created by svn-inject.

-V , –version STRING Forces a different upstream version string

-c , —clean Runs "make clean" and removed the debian/ directory in the new source.

-P , –packagename STRING Forces a different package name

-v , –verbose More verbose program output

-r , –replay-conflicting Extra cleanup run: replaces all conflicting files with upstream versions. Review of "svn status" output before doing that could make sense.

Tarballs must be compressed with gzip or bzip2.

Chapter 6

Further documentation

6.1 Various links

- Subversion Homepage: http://subversion.tigris.org/
- The Subversion Book: http://svnbook.red-bean.com/ or file:///usr/share/doc/subversion/book/
- Subversion vs. CVS and others: http://better-scm.berlios.de/

6.2 Copyright