Using Subversion

Berthold Gunreben
`<berthold.gunreben AT suse DOT de>`
Thomas Schraitle
`<thomas.schraitle AT suse DOT de>`

$Id: svndoc.xml 217 2006-08-21 13:00:48Z thomas-schraitle $

Contents

1 Quick Start

Subversion is, like CVS, a tool for managing source files. It keeps track of your work and changes in a set of files and directories. It allows developers and writers to collaborate. The common area is the *repository*, where all files are stored. The repository can reside on a local directory, but it is usually accessible over a network.

Working with Subversion involves the following steps:

1 You *check out* your working directory from the repository. This needs to be done only once.

2 You *modify* your files, for example, insert new sections or delete some paragraphs.

3. You can check the *status* of your working directory.

4. If you have write permission, you *check in* your changes to the repository. With read access only, you must find someone with write permission if you want to keep these changes.

5. At the same time, other users with write access to the repository might check in their modifications. To get these changes, *update* your working directory.

6. If you change a file and another person changed it too, you might get *conflicts* with the next update. This rare case occurs if two people edit the same lines of the same file. Subversion cannot decide which version is better, so you must resolve the conflict manually.

For more information, see [svn]. Although you can work with Subversion on Windows, this guide is targeted to Linux.

2 Basic Work Cycle

For your daily work, use the `svn` command. It knows `--help`. You can even get the options for a certain Subversion command, for example `svn update --help`. This prints the available options for the update procedure of Subversion.

A typical work cycle looks like this (taken from the Subversion Book):

Procedure 1 *A typical Work Cycle during Development*

1. Update your working copy

   - `svn update`, see Section 6, “Updating your Working Directory” (page 7)

2. Make changes

   - `svn add`, see Section 10, “Adding and Removing Files” (page 10)
   - `svn delete`, see Section 10, “Adding and Removing Files” (page 10)
   - `svn copy`, see Section 11, “Copying and Renaming Files” (page 11)

- `svn move`, see Section 11, “Copying and Renaming Files” (page 11)

3 Examine your changes

- `svn status`, see Section 7, “Examine your Changes” (page 7)
- `svn diff`, see Section 8, “Diffing your Changes” (page 9)
- `svn revert`, see Section 12, “Reverting Changes” (page 11)

4 Merge others' changes

- `svn merge`
- `svn resolved`, see Section 13, “Resolving Conflicts” (page 12)

5 Commit your changes

- `svn commit`, see Section 14, “Checking in your Modifications” (page 13)

6 Tag your work after finishing

- `svn copy`, see Section 9, “Tagging your current trunk” (page 10)

3 Requirements

Before you start, you should know about Subversion:

1. Subversion is platform neutral; it runs on Linux, MacOS, Windows and some others
2. You can run Subversion with the help of a GUI (graphical user interface) or by typing the respective commands in a shell

There are some packages that are essential or might be useful if you work with Subversion:

Table 1 *Essential and useful packages*

Package	Description
`subversion`	This is essential. You can not work without this package
`subversion-doc`	The documentation of Subversion. If you need in-depth information, this is the right place to look.
`kdesvn`	GUI for Subversion. This let you access the repository through a GUI.

The configure check needs additional packages:

- `make`
- `automake`
- `autoconf`

4 Setting up your Subversion Working Environment

Before you work with Subversion, you should check whether all relevant packages are installed. Proceed as follows:

Procedure 2 *Setting up your Subversion Working Environment*

1. Find a place where you have lots of free space and write access. It is important that you do not run out of space. Between 2 and 3 GB should be enough, but more is always better.

2. Configure Subversion

   - Check, if you have a directory `.subversion` in your home directory. If not, create it:

     ```
     mkdir ~/.subversion
     ```

3 Insert the following line into your `~/.bashrc` or `~/.bashrc.local`:

```
export SVNROOT="https://forgesvn1.novell.com/svn/novdoc"
```

This simplifies the typing of your commands.

5 Checking out

The first step when working with Subversion is to check out your files from a repository. This step must be done only once, but it is essential.

Procedure 3 *Checking out*

1 Open a shell

2 Check out an init script that does most of the tedious work (one line):

```
svn cat $SVNROOT/trunk/bin/initdoc.py > initdoc.py
```

You have to enter the password `anonymous` twice.

3 Set the excecution permission:

```
chmod +x initdoc.py
```

4 Run the command. Use as first parameter the directory where you want to store the hole stylesheets and book source codes:

```
./initdoc.py WORKING_DIRECTORY
```

This script creates a directory `SUSEDOC/` inside `WORKING_DIRECTORY`.

Change the language

Your language is determined by the `LANG` environment variable.

5 Change to your directory:

```
cd WORKING_DIRECTORY
```

6 Updating your Working Directory

From time to time, you should update your working directory to get changes others have made:

1. Open a shell and go to the directory where the book file resides, for example, `books`:

   ```
   cdWORKING_DIRECTORY/doc/trunk/books/
   ```

2. Run the update process:

   ```
   svn update
   ```

   The `svn update` might give an output, see Section 7, “Examine your Changes” (page 7) for more information.

The next action depends on the status of the file. If you have modified a file, you probably want to check in your changes. If your file is added or removed, a commit would solve this problem. For files with a conflict, resolve the conflict before handing it to Subversion.

7 Examine your Changes

Before you checking in your changes, it is always a good idea to take a look at exactly what you have changed. Subversion prints the status of your working directory with `svn status`. The output of this commands contains six columns:

First column (content, file and directory changes)
    This is the most important column

	(space) no modifications
`A`	Added
`C`	Conflicted
`D`	Deleted

`G`	Merged
`I`	Ignored
`M`	Modified
`R`	Replaced
`X`	item is unversioned, but is used by an external definition
`?`	item is not under version control
`!`	item is missing (removed by non-svn command) or incomplete
`~`	versioned item obstructed by some item of a different kind

Second column (Property changes)
This is probably the second most important column.

	(space) no modifications
`C`	Conflicted
`M`	Modified

Third column (working directory locked?)

	(space) not locked
`L`	Locked

Fourth column (history)

	(space) no history scheduled with commit
`+`	history scheduled with commit

Fifth column (Switching)
: Only used for switching and is not relevant for us at the moment. See `svn status --help` for more information.

Sixth column (Repository lock token)
: Not relevant for us at the moment. See `svn status --help` for more information.

8 Diffing your Changes

If you do not know, which changes did you make, try `svn diff`. You can diff your files even if you are not online—Subversion holds a copy of the last revision.

To get the differences between your current, modified file and the last revision, run:

```
svn diff foo
```

To get the difference between your working copy and the trunk revision, run:

```
svn diff
```

However, you can also diff between revisions that are already checked in. Subversion knows the following *revision keywords*:

Revision Keywords (taken from the Subversion Book)

HEAD
: The latest revision in the repository.

BASE
: The revision you last updated to

COMMITTED
: The revision of the latest change to the file when you last updated. Or in other words: The last revision in which an item changed before (or at) BASE.

PREV
: The revision just *before* the last revision in which an item changed. (Technically, COMMITTED − 1.)

Here are some examples (taken from the Subversion Book):

```
svn diff --revision PREV:COMMITED foo
# shows the last change committed to foo

svn log --revision HEAD
# shows log message for the latest repository commit

svn diff --revision BASE:HEAD
# shows all commit logs since you last updated
```

9 Tagging your current trunk

This section is probably useful to SUSE employees only. However it can be informative for other as well.

Tags are “snapshots of a project in time”. To create a tag for our books, enter:

```
svn copy $SVNROOT/trunk/books/en \
         $SVNROOT/tags/books/en/TAG_NAME
```

Replace *`TAG_NAME`* with an appropriate name.

10 Adding and Removing Files

If you need to add new files or remove old files, use the commands `svn add` and `svn remove`.

To add a new file, enter:

```
svn addNEW_FILE
```

To delete an old file, enter:

```
svn removeOLD_FILE
```

If you are finished with adding or deleting files, commit your changes:

```
svn commit
```

11 Copying and Renaming Files

Unless CVS, Subversion can copy a file and retain its history. Subversion copies a file from a source to a destination. There are four possibilities where the source and destination are located:

- From a working copy to a working copy (WC -> WC)
- From a working copy to an URL (WC -> URL)
- From an URL to a working copy (URL -> WC)
- From an URL to an URL (URL -> URL). Used to branch & tag

In general, you can copy a file `A` to a file `B` with the following command:

```
svn copy A B
```

After a `svn status` you can see the new file `B` is scheduled for addition. It looks like if you have added and removed a file.

To rename or move a file from a source to a destination, use `svn move` or `svn rename` (the two commands are the same, they are just synonyms.) For example:

```
svn rename X Y
```

This renames `X` to `Y`. After a `svn status` you can see the renamed file is scheduled for addition.

After you copied or renamend the files, you must check in if you want to save it in the repository:

```
svn ci -m"Copied file" A B
svn ci -m"Renamend file" X Y
```

12 Reverting Changes

If you want to discard your local changes, you can use the `svn revert` command. It reverts any local changes to a file, directory or any property changes. It can revert scheduling operations like addition, renaming or deletion only. For example:

```
svn add foo
svn revert foo

svn rename foo bar
svn revert foo bar

svn delete foo
svn revert foo
```

No targets to `svn revert` is a protection

The Subversion book [svn] (page 15) says: “If you provide no target to `svn revert`, it will do nothing—to protect you from accidentally losing changes in your working copy, `svn revert` requires you to provide at least one target.”

13 Resolving Conflicts

Conflicts occur only when all of the following conditions are met:

- Two users edit the same file.
- These two users also edit the same line.
- One of them commits the changes and the other updates.

If you update your working directory and Subversion detects a conflict, it changes the contents of the file and inserts “conflict marks”:

```
<<<<<<< .mine
  This is a nice line.
=======
  This is a short line.
>>>>>>> .r4
```

You have the following possibilities:

1. Remove your version and update the file. You get the latest version from Subversion. This solution works if you do not want to keep your changes. Or
2. Resolve the conflict.

To resolve the conflict, proceed as follows:

1 Open the file containing the conflict.

2 Search for conflict marks. These marks have the general format:

```
<<<<<<< FILENAME

=======

>>>>>>>.last revision number from repository
```

3 Decide which version is preferred (yours or the repository's), or merge the two lines, delete the conflict marks, and save the result.

4 Remove the conflict state with:

```
svn resolvedFILENAME
```

The command `svn resolved` fixes some bookkeeping data.

5 Check in your changes

```
svn ciFILENAME
```

14 Checking in your Modifications

After you have changed, added, or removed files in your local working directory, you have to commit your files back to the Subversion repository:

1 Change your directory to the book sources.

2 Run:

```
svn ciNAME_OF_YOUR_FILE(S)
```

Replace *`NAME_OF_YOUR_FILE(S)`* with the name of your changed file(s). The editor vi opens.

Check the files before commit

It is possible to check in with the command `svn ci`. However, practice showed, that you get modified files that you do not want to check in at this stage. For this reason it is probably a good idea to always insert the files in the command line.

3. Press i to insert your login message.

4. Close the editor with : – w – q. Subversion commits your file(s) with your commit message.

5. If you want to cancel the commit, press : – q – ! in the vi editor. Subversion lets you determine your actions. Abort with a.

Checking in is always a good Idea

Do not hesitate to check in your changes. If you have checked your changes back in our Subversion repository, you are protected from hard disk crash or other loss of data. In other words: Subversion is your backup system.

Please make sure that you always check in valid XML.

15 Viewing History

If you want to see the history of your files use the `svn log` command. For example:

```
svn log xml/help.xml
------------------------------------------------------------------------
r2913 | tom | 2005-12-19 13:34:17 +0100 (Mo, 19 Dez 2005) | 1 line

docmanager: Property »doc:status« set to »proofed«
------------------------------------------------------------------------
r2912 | tom | 2005-12-19 13:33:19 +0100 (Mo, 19 Dez 2005) | 2 lines

worked on comments

------------------------------------------------------------------------
r2877 | rwalter | 2005-12-18 18:52:07 +0100 (So, 18 Dez 2005) | 1 line

docmanager: Property »doc:status« set to »comments«
------------------------------------------------------------------------
r2876 | rwalter | 2005-12-18 18:49:53 +0100 (So, 18 Dez 2005) | 3 lines
```

```
proofing and comments
...
```

It is more convenient to pipe it to less to browse a long history:

```
svn log xml/help.xml | less
```

Quit with q.

16 Restore an old Revision

Sometimes you need to restore an old revision from the Subversion repository. Do the following:

1. Check if your respective file is unchanged. If there are changes that you do not want to loose, check in first otherwise run `svn revert FILENAME`
2. Determine your revision with `svn log`. See also Section 15, "Viewing History" (page 14).
3. Restore your revision and save it; for example (replace *REVISION* from the last step):

   ```
   svn cat -rREVISION \
     $SVNROOT/trunk/books/en/xml/foo.xml > xml/foo.xml
   ```

4. Check in your changes with a meaningful log message.

Links

[svn] *Version Control with Subversion*. http://svnbook.red-bean.com/ .
Ben Collins-Sussman. Brian W. Fitzpatrick. Michael Pilato.