Common Configuration Management Tasks: How to Do Them with Subversion*

Tom Verhoeff

Contents

*`subversion.apache.org`

1 The Big Picture

When using a *centralized* Configuration Management System (CMS), a project has one central *repository* (Master Library and Archive Library) and multiple *working copies* (Development Libraries). Each working copy starts as a copy of (part of) the repository, and changes independently. These changes can be fed back into the repository, and from there propagate to other working copies. This may give rise to *conflicts*, when two independent changes are not compatible. Conflicts must subsequently be *resolved*.

We summarize how to accomplish some common configuration management tasks using Subversion. We use Unix command-line terminology. Terminology for other Subversion tools, such as *TurtoiseSVN*, is usually the same, but their operation may differ.

2 Subversion Help

- `svn help`
- `svn help` *`command`*, in particular for arguments and options
- *The Subversion Book* (freely available on the web: `svnbook.red-bean.com`)

Some commands have alternative forms (not shown here). See `svn help`.

3 Create New Empty Repository

Needs to be done only once. Often done by an administrator.

- `svnadmin create` *`path`*, where *`path`* is an existing empty directory

4 Obtain Access to Repository

Use the repository's URL with an appropriate (network) protocol, e.g. `https`, for (remote) access to the repository; you need appropriate permissions. Ask the repository administrator.

5 Inspect Repository

- `svn list` *`URL`* shows listing (`svn list -v` shows details)
- `svn info` *`URL`* shows general information
- `svn log` *`URL`* shows log messages
- `svn cat` *`URL`* shows file contents
- `svn diff` *`URL1 URL2`* shows line-by-line differences
- `svn blame` *`URL`* shows who edited which lines

6 Modify Repository

- `svn mkdir URL` creates a new directory
- `svn delete URL` removes a file or directory (but not from the history)
- `svn move sourceURL destURL` (also used for *renaming*)
- `svn copy sourceURL destURL` (also used for *tagging* and *branching*)
- `svn import path URL` adds the contents of a directory tree
- `svn commit [path]` incorporate changes from working copy (see §11)

7 Create Working Copy

There are three scenarios:

1. You have nothing, and want to start with something from the repository.
2. You have nothing (yet), neither has the repository, and you want to start from scratch.
3. You have a directory tree with files not yet in the repository, and want to convert it into a working copy, adding *some* or *all* of these files to the repository.

7.1 Create Working Copy from Repository

The repository already has all the files, and you now want a fresh local working copy with these files.

- `svn checkout URL`

N.B. Watch out if you do this inside an already existing working copy!

N.B. The name (of the root directory) of this working copy does not have to be the same as in the repository: `svn checkout URL path` .

7.2 Create Empty Working Copy

You want to start with an empty working copy to fill the repository later.

Create a new (empty) directory in an appropriate place in the repository (`svn mkdir URL` ; you must choose where it belongs in the repository), and check out this empty directory (`svn checkout URL [path]`).

7.3 Convert Existing Directory Tree into Working Copy

A directory tree already exists outside the repository, and you want to turn this into a working copy, where *some* of these files will be added to the repository.

Do the preceding step to create an empty directory in the repository, then check it out into the existing directory (this will not affect existing files), and flag the files (incl. subdirectories) that need to go into the repository with `svn add` . These are then put into the repository at tne next `svn commit` .

To put (some of) the files in directory `mytree` into the repository:

1. `svn mkdir URL/repos-name-of-mytree`
2. `svn checkout URL/repos-name-of-mytree mytree`
3. `svn add -N path` for all (sub)directories *without* adding their contents
4. `svn add path` for all files/directories to end up in the repository
   N.B. (sub)directory contents are added recursively.
5. `svn commit`

When a complete directory tree already exists outside the repository, and you want *all* these files to be added to the repository, you can also consider using `svn import [path] URL` and then `svn checkout` to another (!) location.

8 Inspect Working Copy

- `svn info` shows where a working copy came from
- `svn log` shows log messages
- `svn diff` shows line-by-line differences
- `svn blame` shows who edited which lines when
- `svn status` shows what will happen when committing
- `svn status -q` minimizes output
- `svn status -u` also shows what will happen when updating

9 Modify Working Copy

Changes to the working copy are typically not propagated into the repository until an explicit `svn commit` is done.

Use local (non-SVN) commands for editing files that already exist in the repository.

Create new file that eventually needs to end up in the repository: create locally, then `svn add`. Also works for new subdirectories (`svn add -N path` works non-recursively, i.e., without also adding all files in the directory), but `svn mkdir path` is preferred.

Deletion, renaming, and copying (when intended to go into the repository) are better *not done locally*, but (to preserve history) should be done through

- `svn delete path`
- `svn move source-path dest-path` (also used for *renaming*)
- `svn copy source-path dest-path`

Revert to previous revision: `svn revert`.

N.B. The working copy can (and often will) contain additional files (e.g., created by the "build" process) that are not intended to end up in the repository. Setting *properties* may make life easier, so that Subversion will not bother you about these auxiliary files.

10 Update Working Copy from Repository

- `svn update` N.B. Could give rise to *conflicts*.

11 Update Repository from Working Copy

- `svn commit`

Include a brief description of the purpose of this commit (*log message*). Remember that later a diff can be used to tell you what was changed, but a diff does not tell you why. Possibly do several commits, each with its own appropriate log message.

12 Update Repository from Patch

If you don't have write permission to the repository (often the case in open source projects), then you can communicate changes via *patches*:

- `svn diff > changes.patch`: Contributor produces a patch file.
- Contributor sends the patch file `changes.patch` to a developer.
- `patch -p0 -i changes.patch`: Developer applies the patch file.
  N.B. Execute in same subdirectory (of developer's working copy) where it was created.
- Developer reviews the patch.
- `svn commit`: Developer commits the change when accepting the patch.

13 Get Rid of Working Copy

1. Check whether there are any uncommited changes: `svn status`
2. If necessary, commit changes: `svn commit`
3. Just in case: `svn cleanup` to finish unfinished business
4. Now it is safe to delete the working copy directory tree (if necessary, it can be recovered from the repository).

14 Basic Work Cycle

Also see *The Subversion Book*, Chapter 2.

Step	**Commands**	**Remarks**
Incorporate external changes	`svn status -u` `svn update`	Only needed with multiple working copies

Make your changes	edit contents
	`svn add`
	`svn delete`
	`svn copy`
	`svn move`
	`svn mkdir`
		Do not simply use `rm, cp, mv`

Examine your changes	`svn status` `svn diff`	
Undo some changes	`svn revert`	Only when needed
Resolve conflicts	`svn update` `svn resolved`	Can only occur with mulitple working copies
Commit your changes	`svn commit`	Include appropriate log message

15 Make svn ignore certain files in working copy

- `svn propedit svn:ignore` ***dir-path*** one file name per line (wildcards allowed); in this directory, `svn` will ignore all files matching a pattern; these properties are versioned and need to be commited like any other change

16 Not Covered

- Installing server and client software for Subversion
- Choosing a directory tree structure for the repository:
  `trunk`, `tags`, `branches`
- What to put in the repository and what not
- When to commit (quality criteria)
- How to deal with binary files (incl. use of appropriate properties)
- Revisions and how to identify them with option `-r`:
  `revision number`, `HEAD`, `BASE`, `COMMITTED`, `PREV`, { "*`date`*" }
- How to handle conflicts (incl. `svn resolved`)
- Properties (`propedit`, `propget`, `proplist`, `propset`, `propdel`)
- Tagging (via `svn copy` *`URL`* in `tags` subdirectory)
- Branching (via `svn copy` *`URL`* in `branches` subdirectory)
- Merging: `svn merge`, also consider Python script `svnmerge.py`[1]
- `svn lock`, `svn unlock` claim/release exclusive access to items
- `svn switch` converts a working copy to another revision
- `svn import` imports a directory tree, but not from a working copy
- `svn export` extracts a directory tree, but not as a working copy

[1] See: Subversion Wiki about `svnmerge.py`