**Nuclear Engineering Laboratory**
M O N T E C U C C O L I N O

# memorandum

*Nuclear Reactor Physics*
*Department of Energetic and Nuclear Engineering*
*and of Environmental Control (DIENCA)*
*University of Bologna*

Doc. ID: LIN-M02.1.2007
Subject: Subversion scratchbook
Date: December 20, 2007

Author: Giacomo Grasso
Phone: +39-051-6441708
e-mail: giacomo.grasso@mail.ing.unibo.it

## Subversion Quick reference guide

Subversion is a free/open-source version control system. That is, Subversion manages all data contained in a project over time, remembering every change ever made to its files and directories. This allows every user accessing the repository to recover older versions of the data, or examine the history of how data changed. In this regard, many people think of a version control system as a sort of time machine.

Subversion stores all versioned data in a central repository, i.e. a central common folder. This directory contains (among other things) a collection of database files. Subversion has no concept of a project. The repository is therefore just a virtual versioned filesystem, a large tree that can hold anything at wish.

Currently, the subversion package does not set up a repository, so it is needed to set up it manually. One possible location for a repository is in /var/local/repos (this is just a sample location; to keep general configuration about it from here on the repository location will be addressed as <*repos*>). To set up the central repository, the Subversion command is:

```
$ svnadmin create <repos>
```

It is interesting to notice that it is possible to make the repository writable by a web server, by setting:

```
$ chown -R www-data:www-data <repos>
```

To allow access to the repository via user authentication, it is then needed to add (or uncomment) the following lines in `/etc/apache2/mods-available/dav_svn.conf` (config file location refers to Apache2 web server):

```
<Location /repos>
  DAV svn
  SVNPath <repos>
  AuthType Basic
  AuthName "Subversion repository"
  AuthUserFile /etc/subversion/passwd
  <LimitExcept GET PROPFIND OPTIONS REPORT>
    Require valid-user
```

```
    </LimitExcept>
  </Location>
```

and create the user authentication file with the command:

```
$ htpasswd2 -c /etc/subversion/passwd some-username
```

Restarting Apache2 the new Subversion repository will be accessible via the <*remote_repos*> http://hostname/repos URL.

To import any sort of project (a collection of files and directories) into the Subversion repository, it is first needed to organize all the related elements into a single directory called *myproject* within the <*local*> path. Once the tree of data is ready to go, it can be imported into the repository with the **svn import** command:

```
$ svn import <local>/myproject file://<repos>/myproject -m "Initial log message"

Adding <local>/myproject/file1
...
Committed revision 1.
```

or, via the web server:

```
$ svn import <local>/myproject <remote_repos>/myproject -m "Initial log message"
```

How can be seen, the

```
file://<repos>
```

and the

```
<remote_repos>
```

syntaxes are equivalent each other, and both referring the same physical repository.

Now the repository contains this tree of data. The committed files cannot be accessed by directly peeking into the repository; they're all stored within a database. But the repository's imaginary filesystem now contains a top-level directory named *myproject*, which in turn contains the project data. It is important to note that the original <*local*>*/myproject* directory is unchanged; Subversion is unaware of it (in fact, it is even possible to delete that directory at all!). In order to start manipulating repository data, it is therefore needed to create a new working copy of the data, a sort of private workspace, asking Subversion to check out a working copy of the *myproject* directory in the repository:

```
$ svn checkout file://<repos>/myproject workdir

A myproject/file1
```

```
...
Checked out revision 1.
```

To publish the new changes to other users, the project must be committed back to the repository by means of the Subversion's **commit** command:

```
$ svn commit filestocommit

Sending filetocommit1
...
Transmitting file data.
Committed revision revN.
```

When a new Subversion repository is created, it begins its life at revision zero and each successive commit increases the revision number by one. After every commit completes, the Subversion client informs the user of the new revision number *revN*. The **svn commit** command sends all the new changes to the repository. When committing a change, the user is therefore forced to supply a log message, describing the changes brought to the project. Such a log message will be attached to the new revision created. If the log message is brief, it can be easily supplied on the command line using the –**message** (or **-m**) option:

```
$ svn commit --message "New version log message"

Sending filetocommit1
...
Transmitting file data.
Committed version revN.
```

However, if a more exhaustive log message has been composed, it is possible to tell Subversion to get the message from a file by passing the filename with the –**file** switch:

```
$ svn commit --file logmsgfile
```

To prevent uncommented development of the project, if the user fails to specify either the –**message** or –**file** switch, then Subversion will automatically launch the user's favorite editor for composing a log message. To retrieve informations about the history of a file or directory, it is possible to use the **svn log** command. **svn log** will provide a record of users' related changes to a file or directory, at what revision it changed, the time and date of that revision, and, if it was provided, the log message that accompanied the commit.

```
$ svn log
------------------------------------------------------------------------
rrevN | user | date time permissions | N line
log message of revision revN
------------------------------------------------------------------------
...
```

The typical work cycle looks like this:

- Update the working copy
  - **svn update**
- Make changes
  - **svn add**
  - **svn delete**
  - **svn copy**
  - **svn move**
- Examine changes
  - **svn status**
  - **svn diff**
  - **svn revert**
- Merge others' changes into the working copy
  - **svn update**
  - **svn resolved**
- Commit changes
  - **svn commit**