# Vrsion Control Systems – What are they ?

## Subversion Version Control System

Article written by Jinan Kordab – Programmer

Outline

1. Subversion in a nutshell
2. Branching and tagging
3. What others say
4. Links

Subversion in a nutshell

Subversion is a centralized system for sharing information, and it differs from other version control systems in many ways. Some questions can be asked in subversion like:

- What did this directory or file ( any kind of file, not limited to code files that programmers use) contain last wednesday?

- Who was the last person to change this file, and what changes did he make?

- It supports directory versioning (Files and Directories are versioned)

- Every newly added file begins with a fresh history of its own without **inheriting** the history of another file.

- Supports transactions (Atomic commits) meaning that either all the data is commited or no data is commited at all.

- Choice of network layers (plugs into Apache HTTP server as an extension module)

- A more lightweight standalone subversion server process is also available that can be tunneled over SSH (Secure Shell Protocol).

- As far as security , Subversion is an Open Source project, which is implemented as a collection of shared C libraries with well defined APIs, that can be used to write additional security modules.

- Supports SSL i-e HTTPS

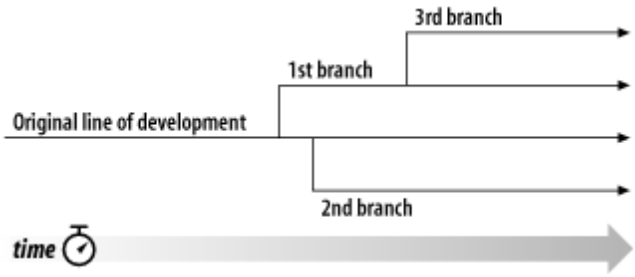
- Can manage any type of files, not limited to programming source code files such as *.cs

Branching and Merging

Branching helps solve the following case scenario: X and Y are assigned to work on the same project for a long time. X needs to use some files to do minor bug fixes and minor changes, while Y needs to change, update, and add some more functionality and modules to the same files that X is using. If Y starts changing the files radically, X may find it confusing and lose his/her work flow. The typical solution is wait for X to finish his/her work and then assign Y to do his work on the same files.

Branching saves this dilemma, by creating a separate branch (line of development) separate from the other, containing the same working files, and while X fixes only minor bugs, Y can implement new features, or change the same files totally, but on a different **branch** or line of development.

In Subversion, branching is simply a word that means **copying** one project, file, or whole directory to a separate destination where different line of development can take place, thus the word branching. *$ svn copy* command is used.




**Pic A. Different branches emerging from single line of development**

Merging simply means comparing two directories, or repositories and the differences are applied to a *working directory.* $ svn merge command is used. The command takes three arguments

1. Initial repository tree, or initial directory
2. A final repository tree
3. A working copy to accept the fifferences as local changes ( often called the target of the merge)

If the results of the merging are satisfying, one can commit the changes using *$svn commit* command or revert back ( transaction rollback ) all the merging using $ svn revert command.

Syntax for *$svn merge* command:

```
$ svn merge http://svn.example.com/repos/branch1@150 \
http://svn.example.com/repos/branch2@212 \
my-working-copy
```

Some common problems with Subversion Merging

- If one repeatedly merges changes from one branch to another, accidentally one may merge the same change twice i-e subversion is *not* such an application that prevents double-application of changes to a branch (working folder) . Prevention for this type of scenario involves *human factor* , at least for the current version of Subversion, where one should commit a *log message* where spesifying revision number ( or range of revisions ) that are being merged into the branch, and later on use *$svn log* command to review which changes a specific branch contains.
- Merging of two repositories must only be done into a working copy that has no local edits and has been recently updated. If done otherwise, some problems might occur.
- As mentioned earlier, *$svn revert* command can be used to revert the merging to specific directory or repository, **but** if one has merged into a working copy that already has local modifications, the changes applied to a merge will be mixed with pre-existing ones and running *$svn revert* command is no longer an option.
- If *$svn merge* is used on two **unrelated trees ,** the delta, or the differences that should be merged into the working copy , could not apply cleanly, meaning that the differences may not be correctly understood, after which *$svn merge* command will complain about “Skipped Targets”

Edample of *$svn merge* complaining about skipped targets:

```
$ svn merge -r 1288:1351
http://svn.example.com/repos/branch
U foo.c
U bar.c
```

```
Skipped missing target: 'baz.c'

U glub.c

C glorb.h

$
```

Solution to this problem is fortunately easy enough, and is done by running a rollback of *recursive revert* on the changes done by the merge. *$svn revert – recursive* function can be used, then deleting any unversioned files left behind after revert and rerun merge again.

- Caution should be taken when merging two copies or renames from different branches, especially with renamed files, since merge will delete the original file and replace it with the new one .

Solution:

Most merges involve comparing trees that are ancestrally related to one another, and therefore s**vn merge** defaults to this behavior. Occasionally, however, one may want the merge command to compare two unrelated trees. For example, you may have imported two source-code trees representing different vendor releases of a software project . If you asked **svn merge** to compare the two trees, you'd see the entire

first tree being deleted, followed by an add of the entire second tree! In these situations, one will want **svn merge** to do a path-based comparison only, ignoring any relations between files and directories. Adding the *--ignore-ancestry* option to your merge command will solve this problem, and it will behave just like **svn diff**. (And conversely, the *--notice-ancestry* option will cause **svn diff** to behave ike the merge command.)

<u>What Others Say</u>

In the previous releases of Subversion, most users were unsure on how to use $svn commands, since the manual was very short. In the last build, the manual is expanded with many use cases and all API function descriptions.

Since Subversion is an Open Source project that was eveolved from the deficiency of previous version control systems, one can say that it is productive and efficient CVS.

Links

A link to ***AnkhSVN,*** a Subversion Plugin for Visual Studio. This software allows you to perform the most common version control operations directly from inside the Microsoft Visual Studio IDE. http://ankhsvn.open.collab.net/

A link to Subversion project home http://subversion.tigris.org/

# INSTALLING SUBVERSION ON A SINGLE MACHINE

Since most operating systems, including Windows XP, are not already equipped with Subversion, it needs to be installed from its project initiators, tigris.org from the current working link http://subversion.tigris.org/project packages.html

Subversion is distributed both as source code and binary packages.

Precompiled packages (binary), is the best choice, since Subversion relies on some other packages as well.

The following steps will be takes to configure subversion (server and client)

1. Download Subversion x.x.x-setup (latest) that comes as windows installer package
2. Run the setup

If Repository will be used on local lan, and no encryption of data is needed, the best simple solution is to use svn scheme, i-e **SVNSERVE**, which is a small server that listens to connections, allows repository access and supports simple authentication. This option is suitable for teams in small private LAN.

**SVNSERVE** is great, but it needs to be restarted each time windows is closed, limiting the productivity

**If you want SVNSERVE to run whenever your windows server boots, then it must be installed as a service .** Svnservice, a simple svn wrapper is available from http://dark.clansoft.dk/˜mbn/svnservice/

For client side, TurtoiseSVN can be used, it comes with TortoisePink which already has SSH ( Secure Shell Protocol ).

**TortoiseSVN 1.4.8**, built against ***Subversion 1.4.6***. Released 16. February 2008.

Will be downloaded from http://tortoisesvn.tigris.org/

## SUBVERSION CHARACTERISTICS

- Subversion is a centralized system for sharing information
- svn - subversions command line client program
- - Security
    - -implemented as a collection of shared C libraries with well defined APIs
    - maintainable and usable by many other application languages
- -Choice of network layers
    - plugs into APACHE HTTP server as an extension module
    - a more lightweight standalone subversion server PROCESS is also available can be tunneled over SSH
- -Versioned metadata(data that describes data, in this case files and folders have properties) one can create and store key=value pairs for folders and files.NOTE: properties are also versioned
- - Supports transactions(Atomic commits) Either all the data is commited, or no data is commited at all
- -Every newly added file begins with a fresh clean history if its own without inheriting the history of another file
- - Supports directory versioning(files AND directories are versioned)

PRESENTATION SLIDES