Subversion Basics

Overview

This is intended to serve as a basic introduction to the Subversion source code repository tool. Basic terminology will be provided along with examples of commands.

Subversion uses a file system paradigm. So many terms like copying, deleting, moving, etc. are similar to their file system analogs.

Terminology

version control - the management of changes to code or other files using a central tool or service (Subversion being one example)

repository - the remote copy of the code on the Subversion server; often abbreviated to "repo" or sometimes called the "remote repository"

working copy - your local copy of an SVN folder within a repository, including any local changes you have made

structure node - a container within a repository that can have files in it or another structure node; similar to a directory in a file system

folder - aka structure node; also sometimes used to refer to directories in the local working copy that represent the local copy of these folders

global revision - every repository has a global revision which tags its current contents and is incremented after each commit; in Subversion commands these will always start with 'r' as in 'r1234'

commit - push your changes to the repository (making a new global revision)

trunk - the remote repository's current, main revision of a project; usage of this term is only by convention as it usually just refers to a certain folder called *tr unk* in the repository

revision - globally unique number in the repository starting with "r" that tags the entire state of the repository after a commit

tag - a copy of the trunk from a certain point in time, which should not be modified once created; usually kept in a structure node called *tags*

branch - a copy of the *trunk* or some other structure node which represents a development fork of the code; usually kept in a node called *branches* under the project's directory in the repository

merge - combining together two copies of (usually) the same node in order to merge their changes

revert - undo local, uncommitted changes in the working copy and replace with them with the current copy from the repo

URL - a resource locator used for some SVN commands; in the case of Subversion these will look like *svn://repo/path/to/something*

relative path - the relative path to a file or directory in the working copy (as opposed to a full URL); many svn commands can use relative paths from within a working copy and not just absolute URLs

repository root - the base URL of the repository such as svn://svn.freehep.org/hps/

head revision - the most recent global revision of the repository

^ - The "^" character can be used to specify the repository root for any command that accepts a Subversion URL.

Repository Structure

SVN Repositories may contain many code projects.

These are separated by different folders in the repository, usually under the root directory, but potentially in any sub-folder.

```
project1
project2
project3
[etc.]
```

Each project folder will typically have three sub-folders.

```
project1
    trunk
    branches
    tags
```

trunk should contain the main development branch of the code. Most user commits will happen on trunk.

branches contains folders that are copies of the *trunk*, or some other folder, made by users; these represent development branches of the code that are eventually merged back into trunk or abandoned.

tags contains copies of the trunk that should not (really ever) be modified; typically, these are made when releasing software versions by a build tool or script.

SVN Client

The primary way to interact with a Subversion repository is through the *svn* command in a command terminal.

You can check if you have this installed on the terminal by typing

```
svn
```

Should the command not be found, then it would be a good idea to install it.

If you are using Linux, this is almost certainly already installed for you by your distro.

Otherwise, you could try installing it (e.g. for Yum users).

```
yum install subversion
```

On OSX, it should be bundled with the Xcode development tools which you can get in the app store.

Ignoring Files

You will probably want to configure the SVN client to ignore certain types of files that you might have in your working copy but do not want to be visible.

You can add global file patterns that should be ignored to the config file at *~/.subversion/config* which should have been created by Subversion (if it does not exist then execute the *svn* command in a terminal).

These are some example settings.

```
global-ignores = *.class *.jar .classpath .project .settings
```

The filter will apply when using the *status* command and ignored files will not be considered by *add*, etc..

Commands

All commands described below assume you are in a terminal using bash on Linux or a similar Unix OS; also, most of these commands assume that the current working directory in the command terminal represents an SVN structure node that has been checked out.

Most of the arguments used for these example commands are completely bogus and, depending on the command, should be replaced by valid paths to files in your working copy or the remote repository (should you actually want to execute those commands!).

If you really want to experiment with some of the more advanced commands (branching, merging, copying, etc.) you might think about setting up your own test repository as doing this kind of stuff on an actual production repository if you are just testing or fooling around is not polite.

Checkout a Folder from SVN

Most commonly, you will checkout folders from SVN to local directories using a command such as this.

```
svn co svn://example.org/repo/some/dir
```

svn:// is the protocol for communicating with the SVN server.

example.org is the host name of the server.

repo is the name of the SVN repository (servers can have multiple repositories).

some/dir is a folder in the repository.

You may checkout the entire repository.

```
svn co svn://example.org/repo/
```

But it is probably not a good idea.

Show Working Copy Information

This command will print out general repository information like your current revision number and the repository root.

```
svn info
```

This command will list detailed information about changes and additions to your local working copy compared with the repository.

```
svn status
```

Each file listed with this command will have a letter next to it representing its status. The meaning of these abbreviation is fully described here.

Show Changes in Working Copy

You can use the *diff* command to show changes between the repository and your working copy.

```
svn diff | less
```

This will display the changes in diff format between working copy files and the repository.

You can also specify a file or dir

```
svn diff some/file/or/dir
```

Show Folder Contents

You can list the contents of a directory in the repository using this command in your working copy.

```
cd my-working-copy; svn ls some/relative/path/
```

Or you may also use a URL in which case the command need not be executed within a working copy.

```
svn ls svn://repo/some/path/
```

This command ignores (always?) your working copy and always looks at the folder on the server.

Update Working Copy from Repository

An entire working copy can be updated by executing a command like this.

```
cd hps-java-trunk; svn up
```

The command may also be executed with files or directories to limit the local files that are affected.

```
svn up some/dir some/file another/file
```

You should periodically update your working copy from the repository to keep it up to date.

Add Files

This command can be used to add files to the repository.

```
svn add rel/path/to/file1 rel/path/to/file2
```

These changes will be pushed to the remote repository when you execute a commit command

> **SVN add command**
>
> Don't use *svn add* without providing arguments as you may inadvertently add files to your commit which should not be in there.
>
> Also be careful using directories as by default SVN will add unknown files unless they are excluded (e.g. in user's Subversion config file)

Delete Files

This command can be used to delete files.

```
svn rm rel/path/to/file1 rel/path/to/file2
```

The deletion will occur in the repository when you commit.

When you execute the *rm* command, Subversion will not by default leave a local copy of the file and so will delete it immediately!

Commit Local Changes

This command is used to commit your local changes to the repository, including deletions, changes and additions which have been done on the working copy.

```
svn commit -m "committing some stuff" path/to/file1 optional/path/to/file2
```

Usually it is good to include a list of files that should be specifically affected by the commit. Otherwise, you may inadvertently commit changes.

Files must be explicitly added or deleted in order to be included in a commit using the *add* or *rm* sub-commands (described above).

Any changes to files that SVN knows about in the working copy will be included in the commit automatically.

Copy Files

Similar to how a file system works, files can be copied to directories. In Subversion, this will essentially fork the file from the source version of it where it can be independently changed.

Here is an example of making a copy of a file with a new name.

```
svn cp some/file1 another/file2
```

This copies the file *file1* to *file2* in another directory.

Or a directory may also be used as the target, which will keep the file's original name.

```
svn cp some/file1 another/dir
```

Create a Directory

There are two ways to make a directory in a Subersion project.

Firstly, you may create a directory locally and then add it.

```
cd some-svn-project; mkdir newdir; svn add newdir
```

Or you can execute a command which will affect the remote repository directly.

```
svn mkdir path/to/newdir
```

In both cases, you will need to commit in order for these changes to be pushed to the repository.

Merging Changes

> **Advanced Command**
>
> Merging is an advanced command. Always use '--dry-run' to check the results of a merge before executing it.

Merging involves combining changes from one version of the repository with another.

Two different types of merges will be covered here, though this is far from covering all the ways in which this command can be used.

When working on a branch, you will periodically want to merge from trunk to your branch.

```
cd my-hps-java-branch; svn merge ^/java/trunk
```

Similarly, you eventually will want to merge back into trunk from the branch once you are done with it.

```
cd hps-java-trunk; svn merge ^/java/branches/my-hps-java-branch
```

Merging can also be used to undo bad commits.

See for examples:

http://stackoverflow.com/questions/13330011/how-to-revert-an-svn-commit

This page shows the full syntax for the merge command.

Making a Branch

Creating a development branch is done using the copy command to copy the current version of the trunk into a branches folder.

```
svn cp -m "Creating development branch." ^/projects/java/trunk ^/projects/java/branches/my-dev-branch
```

Now *my-dev-branch* will be a complete copy of *trunk* at the time when it was copied.

Making a Tag

Tagging is actually performed by using the copy command.

```
svn cp -m "Creating a tag." ^/projects/hps/java/trunk ^/projects/hps/java/tags/mytag
```

Typically, tags are not created by hand but through an automated build/release system.

Tags are only by convention in SVN as the directories are typically not made read-only. If you are checking out and modifying something with *tags* in its path, you're probably doing it wrong.

Reverting Local Changes

You can replace a file in your working copy with the repository's current version using a command like this.

```
svn revert path/to/broken/local/file
```

You can also recursively revert entire directories and their sub-directories should you really want to revert a lot of changes.

```
svn revert -R path/to/some/dir
```

You will want to be careful with this, as you can easily lose your work if it is has not been committed yet and gets reverted!

Warnings and Tips

- **Do not checkout an SVN project into another SVN project.** This has the potential to confuse both you and the Subversion client (and repository).
- **Do not checkout and modify tags of the code.** By convention, a node representing a tag in SVN, usually with "tags" as one of the sub-directories in its path, should not be modified once it is created. Changes should only be made on branches or the trunk.
- **Be careful when using 'svn add' on a directory.** When used without file arguments, the *svn add* command will include all files and sub-directories. You may not want this, so it is better to explicitly list files when adding them.
- **Be careful when using 'svn add' and 'svn commit' without any arguments or with directories.** This may cause many files to be added to the repository that should not actually be tracked and will subsequently need to be removed. Once a file is added, it is tracked "forever" by the repository and never really deleted unless the repository is rebuilt with that commit stripped out. So please double check that you are adding only the files that are intended to be included. Before making a large or complicated commit, first use *svn status* in your working copy to see what files will be committed so you can verify that the status is correct.
- **Use the '--dry-run' argument before executing complex or dangerous commands.** This will show you what will happen without the command actually being executed.
- **Do not be too dependent on your IDE's Subversion plugin.** Make sure to have a compatible SVN command line client available e.g. where the SVN minor version number is the same as your IDE's "connector" version (Eclipse), so that you are able to execute shell commands. You should also be careful of certain GUI commands for adding files, as they will often include many files in the commit that you do not want (which is why typically it is preferable to list out all files that should be in a commit or add them individually).
- **Update your working copy regularly.** This will make sure you do not cause conflicts by modifying a file that someone has already made changes to in the repository. Also, the longer you wait between updates, the more chance that you have changes in your working copy which are not compatible with the repository's version.
- **Do not develop code in your working copy for an extended period of time without committing it.** This is especially true if there is the possibility that someone else is working on the same file. When possible, prefer checking in your work at the end of each day, or, at a minimum, the end of every week.
- **Use branches when making an extensive set of complex changes to the trunk.** This will ensure that the trunk is not broken by committing unstable code, and it will also allow you to commit changes that "break everything" without affecting anyone else using the trunk.

More Information

The Red Bean SVN Book is an excellent source of information for all things Subversion.