UsingSVN

Misc info

Whatever you do, don't checkout /commons as you will be checking out a copy of every tag, trunk, and branch and it will take forever and a day. *Instead*, check out individual components, or (more likely) check out "trunks-proper" or "trunks-sandbox".

Checkout all components

```
svn co https://svn.apache.org/repos/asf/commons/trunks-proper/ commons/trunks-proper
svn co https://svn.apache.org/repos/asf/commons/trunks-sandbox/ commons/trunks-sandbox
```

Checkout individual components* _(Don't forget to checkout commons-build!)

```
svn co https://svn.apache.org/repos/asf/commons/proper/commons-build/trunk commons/trunks-proper/commons-build
svn co https://svn.apache.org/repos/asf/commons/proper/dbcp/trunk commons/trunks-proper/dbcp
```

```
svn co https://svn.apache.org/repos/asf/commons/proper/commons-build/trunk commons/trunks-sandbox/commons-build
svn co https://svn.apache.org/repos/asf/commons/sandbox/cache/trunk commons/trunks-sandbox/cache
```

== Getting directory listings==

If you don't know the exact path to the component you wish to access, use 'svn ls' to get a directory listing, eg:

```
svn ls http://svn.apache.org/repos/asf/commons
```

A web browser can also be used to view the repository structure, just by entering the URL that you would use for subversion operations. Note that this shows only the latest version of everything (directories and files).

Importing

```
svn import https://svn.apache.org/repos/asf/commons/sandbox/PROJECT_NAME/trunk
```

Where PROJECT_NAME is the name of the package you want to import.

Differences between CVS and SVN

Normally CVS "tags" are simply used to mark a set of files so that you can retrieve that same set later. In this case, the equivalent in subversion is just to use
svn cp {from} {to}
eg
svn cp https://..../trunk https://.../tags/beta1 to save the current state of the trunk as a directory "beta1". The copy command makes a "light-weight copy", essentially a sort of hard link with copy-on-write so updates don't affect the original source.

If the trunk versions move on, and you later want beta1 to include one of the updated files, then update what "beta1" refers to by relinking from the beta1 directory to the version you really want:

- removing the file (link) you no longer need from the "tag" dir
  eg svn rm https://...../tags/..../foo.txt [1]

- copying back in (ie link to) the version you want
  eg svn cp
  -r 100 https://..../trunk/.../foo.txt [2]
  https://..../tags/.../foo.txt [3]

[1] or if you have a working copy of the tag dir, you can
do svn rm and svn commit
[2] if you want the latest version, just omit the -r 100.
[3] or if you have a working copy of the tag, copy to your working
copy then svn commit it.

Performing the copy again (ie linking to the *updated* file [together with its history] from the tag) is what Brett means by "copy with history again". Using "svn merge" doesn't do the same thing, because it is effectively generating a patch file then applying it to your version; the differences get merged in, but not the history.

Alternatively, if the "updated" tag is meant to look mostly or completely like the new trunk, then just use "svn update" to ensure your trunk working copy looks like the stuff you want to tag, then delete the old "beta1" directory, and recreate it. Copies are cheap!

There isn't any difference between a directory created with the intent of just using it as a "tag" and a directory created with the intent of using it as a branch. The convention of putting the scn copy into subdirs "{project}/branches" and "{project}/tags" are traditionally used to *indicate the intent* of the copy, but they are functionally identical.

For some other traditional uses of CVS tags, it might be better to use subversion "properties" (see svn set-prop).

Maven 1 config*

project.properties*

```
maven.changelog.factory=org.apache.maven.svnlib.SvnChangeLogFactory
```

*

project.xml*

```
  <repository>
   <connection>scm:svn:http://svn.apache.org/repos/asf/commons/proper/${pom.artifactId.substring(8)}/trunk<
/connection>
    <url>http://svn.apache.org/repos/asf/commons/proper/${pom.artifactId.substring(8)}/trunk</url>
  </repository>
```

Commons repo

- http://svn.apache.org/repos/asf/commons/

Software

- http://subversion.tigris.org/
- http://tortoisesvn.tigris.org/
- http://subclipse.tigris.org/ (Add http://subclipse.tigris.org/update as an update site in Eclipse's update manager)

If you are trying to get the latest version of subeclipse, do not use Eclipse's "search for updates of existing features", instead use "search for new features." Eclipse kept telling me that there was no version greater than 0.9.22 while this fellow was telling me he'd just released 0.9.26. Running search for new features against the subversion.tigris.org site quickly found what I was looking for. Any commons devs who want to try subeclipse but are stuck on 0.9.22 should try this.

- http://esvn.umputun.com/_ Client for Linux, Mac, Windows_
- http://www.cygwin.com/ (Linux-like environment for Windows with command-line svn available)

Documentation

- http://svnbook.red-bean.com/
- http://subversion.tigris.org/faq.html
- http://www.apache.org/dev/version-control.html#https-svn
- http://jakarta.apache.org/site/source.html

Subversion client config

Add this to your subversion client configuration file:*

Note:_' make sure the settings are merged into the appropriate section if it already exists, as duplicate section names can cause problems.

Possible locations of your subversion client configuration

- Windows: C:\Documents and Settings\yourname\Application Data\Subversion\config
- Linux: ~/.subversion/config

See: http://www.apache.org/dev/svn-eol-style.txt for the canonical list of auto-props settings.