Introduction to Subversion

Subversion is an open source revision control system that allows one or more users to easily share and maintain collections of files. Most of the material here is distilled from http://svnbook.red-bean.com.

Subversion availability

To determine if you have subversion installed on your system, type

```
$ svn --version
```

How the repository is set up

Determine your repository URL. For example, the EMC repository is located at https://svn.ncep.noaa.gov/emc with a mirror on the vapor R&D machine at file:///gpfs/v/svn/emc. Various projects are maintained in the same repository. Each project contains three directories: `trunk`, for mainline development; `branches`, for branch development off the mainline; and `tags`, for labeling "frozen" snapshots of your system.

Revision numbers

When a Subversion repository is created it starts off at revision zero and each successive commit increments the revision number by one. Note that, unlike CVS, the revision number is repository-wide so *any* commit increments the revision number. Typically you don't have to worry about the value of the revision number, but there are situations – in particular, merging – where it is helpful.

Importing files into subversion

To import a new project into the repository, first move to the location containing the directory hierarchy, let's call it `projX`, you wish to import, e.g.

```
$ cd $HOME/projects
```

Then, use the `svn import` command,

```
$ svn import -m "New src" projX https://svn.ncep.noaa.gov/emc/X/trunk
```

This imports the directory hierarchy `projX` into `emc/X/trunk` in the repository. The `-m "New src"` option sets the log message for this import; without it the default editor (usually emacs) is invoked to allow you to enter a log message.

Checking out files

The above import does *not* place the local `projX` hierarchy under version control. To get a versioned hierarchy you need to obtain a working copy. First, move to a location where you want to create your workspace,

```
$ cd $HOME/workspace
```

Then use the `svn checkout` command,

```
$ svn checkout https://svn.ncep.noaa.gov/emc/X/trunk projX
```

You will notice that a new directory named `projX` was created. Additional projects can also be checked into the same workspace area,

```
$ svn checkout https://svn.ncep.noaa.gov/emc/crtm/trunk CRTM
$ svn checkout https://svn.ncep.noaa.gov/emc/gsi/trunk GSI
$ ls
CRTM    GSI     projX
```

If you enter any versioned directory, you may notice there is a hidden directory named `.svn` present. ***This is where Subversion stores internal information, and you should not modify any of its contents.*** Once you have successfully checked out your project into your workspace, you should consider deleting the original sources so you aren't tempted to edit them directly and bypass Subversion.

Editing files

Once Subversion has created a working copy of your project(s), you are free to edit, compile, debug, and test as usual – they're the same as when they were unversioned.

Merging your changes back into the repository

Since each user can check out a working copy of a project, the changes you make are only visible to yourself. When you are happy with the changes, you must commit them into the repository to make them available to other users and developers. But what if another user has changed the same file(s)? Subversion handles this by requiring you to update your working copy to the current repository version before you can commit your own changes. This is done using the `svn update` command,

```
$ svn update hello.f90
```

This will merge any changes (assuming there were no conflicts) in the repository into your working copy. You then compile and test the code again to make sure it still works. If Subversion finds a conflict, it will notify you where the conflict exists and it is up to you (in coordination with other users) to resolve it.

Now that your working copy is up to date, you are ready to commit your changes to the repository using the `svn commit` command,

```
$ svn commit hello.f90
```

At this point, Subversion will start an editor to allow you to enter a log message that describes the change. A good log message briefly describes not just what the change was, but *why* the change was made. It is recommended that users follow GNU Change Log format when typing the log message. When you exit the editor, Subversion will commit your changes to the repository, where they will become visible to all users.

Checking the status of files

When you have a working copy checked out, you will occasionally want to know which files are up to date, and which have been locally modified. This is done using the `svn status` command,

```
$ svn status
```

Note that this form of the command only indicates *locally* modified items; that is, what you have changed in your working copy since your last update. If want to indicate what items in your working copy are out-of-date, use the `-u` (or `--show-updates`) switch,

```
$ svn status -u
```

If there is no output, then everything is up-to-date.

To see the commit log messages for a particular item, use the `svn log` command,

```
$ svn log hello.f90
```

To see the differences (if any) between the working copy of a file and the version of the file since your last update (the "base" revision), use the `svn diff` command,

```
$ svn diff hello.f90
```

Adding new files

When you create a new file, or include an already existing one, in your working copy, it remains local until you commit it to the repository. You must first schedule the file for addition using the `svn add` command, and then commit it,

```
$ svn add newfile.f90
A         newfile.f90
$ svn commit -m "Initial commit" newfile.f90
```

Deleting files

Removing a file from your working copy does not remove it from the Subversion repository. Similarly to the `add` subcommand, you must first schedule the file for deletion using the `svn delete` command – which also deletes it from your working copy – and then commit it,

```
$ svn delete oldfile.f90
D         oldfile.f90
$ svn commit -m "Removed file" oldfile.f90
Deleting       oldfile.f90
Transmitting file data .......
Committed revision 5.
```

Subsequent updates and checkouts will no longer include deleted files. However, specifying a revision number with the `update` subcommand can restore older versions of the file,

```
$ svn update -r4 oldfile.f90
A         oldfile.f90
Updated to revision 4.
```

Two important things to remember:

1) always use Subversion (not operating system) commands to delete files. This will prevent you from unwittingly deleting a locally modified file.

2) you can always retrieve a deleted file from the repository by specifying the appropriate revision number to an update.

Renaming files

Subversion provides a shortcut compared to the usual delete-then-add procedure: the `svn move` command followed by, as always, a commit,

```
$ svn move thisfile.f90 thatfile.f90
A         thatfile.f90
D         thisfile.f90
$ svn commit -m "Renamed this to that file" thisfile.f90 thatfile.f90
```

Copying files

This section is included only to reinforce the point to *not* use operating system commands to operate on files. Use the `svn copy` command,

```
$ svn copy thatfile.f90 otherfile.f90
A         otherfile.f90
$ svn status
A  +   otherfile.f90
$ svn commit -m "Copied other from that file" otherfile.f90
```

So, why use the `copy` subcommand; and what does the "+" mean in the `status` subcommand output? Using the Subversion command to copy a file preserves the history of the file, and the "+" in column four of the status output indicates that that history is scheduled for commit. If you did the following,

```
$ cp thatfile.f90 otherfile.f90
$ svn add otherfile.f90
A         otherfile.f90
$ svn status
A      otherfile.f90
```

then none of the previous history of `otherfile.f90` is preserved – the file is a brand new one in the eyes of Subversion.

Undoing changes

If you haven't committed, then the `svn revert` command will undo all local edits and scheduling operations (e.g. files and directories you have scheduled for addition or deletion),

```
$ svn status changedfile.f90 addedfile.f90
A      addedfile.f90
M      changedfile.f90
$ svn revert changedfile.f90 addedfile.f90
Reverted changedfile.f90
Reverted addedfile.f90
```