Help Topic: The SVN Update Command

Spring 2021

Michael Benjamin, mikerb@mit.edu
Department of Mechanical Engineering
MIT, Cambridge MA 02139

The SVN `update` Command

The `svn update` command lets you refresh your locally checked out repository with any changes in the repository HEAD on the server. It also tells you *what* has been changed, added, deleted. If a change has been made to a file you have also changed locally, svn will try to merge those changes. If unsuccessful, it will report a conflict. More on this is discussed below. More info can always be found by Googling "Subversion book" and reading the full PDF online free, or just typing `svn help update` anytime on the command line.

Basic usage of the update `update` command

The basic syntax for the `svn update` command is:

```
$ svn update        (invoked somewhere within previously checked out tree)
```

The `svn update` command requires a network connection to the server and perhaps a password if it is not an anonymous read-only repository. The command may be invoked anywhere in the locally checked out tree. *If it is invoked in a subtree, updates only for that subtree are pulled from the server.* The update will work recursively for all directories within the directory it is invoked. This latter behavior is usually what you want, but can be overridden with the `--depth` argument, not covered here, but covered in the online Subversion book.

Deciphering the output of the svn `update` command

In the simplest case, when your locally checked out tree is up to date with the server, you will see a two-line response like the following:

```
$ svn update
Updating '.':
At revision 2675.
```

The revision number output is convenient since you and co-developers can easily confirm that you're working with the same version from the server after you both do an update.

Beware of the common "gotcha". Two developers both perform an update and confirm that they have the same revision - but their code is acting differently. This can drive people bonkers. The thing to remember is that, even though you both have the same revision, one or both of you may have local modifications that you have not yet checked into the server. Very likely one of these local modifications is causing the difference in observed behavior. Check for local modifications using the `svn status` command, discussed in a separate help topic or in the svn book.

Here's an example output after an update:

```
$ svn update
Updating '.':
U    Figure.h
A    Object.h
G    Items.cpp
A    Object.cpp
```

This indicates that:

- The file `Figure.h` was updated with changes pulled in from the server. There were *no* local modifications to this file prior to the update.
- The files `Object.h` and `Object.cpp` are newly added from the server to your local checkout.
- The file `Items.cpp` had changes known to the server and separate changes in the local copy of the file successfully merged. (Note the server will not have the local changes until you commit the changes to the server.)

The primary cases you will likely see the most frequently are below. (See `svn help update` for a more complete list.)

```
A  Added
D  Deleted
U  Updated
C  Conflict
G  Merged
```

The topic of a conflict (C) is discussed next.

Dealing with a conflict after an svn `update`

Subversion is pretty good at merging two files with separate changes. It usually gets things right, and in my experience errs on the side of being conservative. It has never merged two files that didn't deserve merging. But sometimes flags as a conflict an issue that is easily resolved when it hits human eyes. In the case above, if the file `Items.cpp` was *not* able to be merged by the svn algorithms, you would have see this instead:

```
$ svn update
Updating '.':
U    XWFigure.h
A    XKObject.h
C    XWItems.cpp
A    XKObject.cpp
Updated to revision 303.
Conflict discovered in file 'Items.cpp'.
Select: (p) postpone, (df) show diff, (e) edit file, (m) merge,
        (mc) my side of conflict, (tc) their side of conflict,
        (s) show all options:
```

The choices are a fairly self-explanatory. I choose (e) most of the time unless I just happen to know that my local change is worth discarding, in which case I choose (tc). The (m) option puts you into an interactive tool that helps you de-conflict. The (df) option just shows you the difference between two files. The (p) option just kicks the can down the road and leaves the file marked as conflicted. You won't be able to check in local changes until that conflict is resolved.

The one worth special mention is the (e) option. It's probably best to deal with any conflicts as they come up, and dealing with them during the update process is my own general practice. Selecting the (e) option launches an editor in your terminal window. The editor will show a file with the conflicts delimited as by example below.

```
<<<<<<< .mine
// string fruit = "apples";
=======
// string fruit = "pears";
>>>>>>> .r305
```

You can resolve the conflict by picking one version of a block of code and deleting the other (and the delimiter lines), and saving the file. Once you have quit the editor, you are prompted with the same set of options with one additional option, the (r) option:

```
Select: (p) postpone, (df) show diff, (e) edit file, (m) merge,
        (r) mark resolved, (mc) my side of conflict,
        (tc) their side of conflict, (s) show all options:
```

By choosing the (r) option you are acknowledging to svn that all is good from your perspective, for that file. It will then continue with the rest of the update.

Configuring SVN with an editor of your choosing

Getting back to that issue of launching an editor to resolve conflicts as described above. "Which editor?" you may ask. The editor chosen is determined by how you have certain environment variables set in your shell configuration. It will first look for the value of the `SVN_EDITOR` variable, and fall back to the value for the `EDITOR` environment variable. If these are not set, svn may not allow you to select the (e) option.

If you're a `tcsh` user, add the following lines to your `.cshrc` file, with the editor of your choosing:

```
setenv EDITOR 'emacs'
setenv SVN_EDITOR 'emacs'
```

If you're a `bash` user, add the following lines to your `.bashrc` file, with the editor of your choosing:

```
export EDITOR=emacs
ecport SVN_EDITOR=emacs
```

To check the prevailing value of an environment variable, e.g., `EDITOR`, just type this on the command line: `echo $EDITOR`.

(The `EDITOR` is also relevant when checking in (committing) code and adding a commit comment.)