

Version Control with Subversion

Software development projects typically use version control systems like subversion, cvs, or git to manage source code files and coordinate efforts of many developers. In bioinformatics and other scientific applications, we use version control systems not only for source code management but also as a way to track analyses, data files, manuscript drafts, and other resources.

In this in-class lab assignment, you'll get started using the subversion version control system and the class subversion repository to manage files for your projects and assignments. You'll import your python file for all future assignments into your personal space within the class subversion repository. When you leave the lab, you'll install a subversion client on your home computer and use it to download a copy of your checked-in source code and continue your work. You'll also use subversion (instead of email) to hand in assignments from now on.

Why use version-control? There are a lot of good reasons, and one of the most important is that version-control systems automate a lot of the file merging functions you would have to do by hand when working on a project with other developers or scientists.

For example, let's say you and some partners in the class are editing the same file `big_project_one.py`. You and your partners wisely decide to divide up the work, assigning different function implementations to different team members.

Without a version control system, you would then have to merge the various changes you have made by hand into a single master file. This would typically be a painful and error-prone process even with a one file, much less many different ones. And you would have to repeat it with each bug-fix.

Subversion, as with other version control systems, will automate much of this process for you. How it works in practice is quite simple. Each team member checks out a version of a master copy of the same repository, which typically contains a collection of files belonging to a project. The repository also captures the entire history of each file, including all changes made to each dating from the file's addition to the repository. Each team member then edits his or her checked-out versions independently. Once they are happy with their edits, they "check in" their changes to the project repository, also providing a log message describing their work. At that point, the subversion program running on the server merges the changes into the central, master copy of the file and increments the version number associated with the repository. Then, all the other developers and scientists using those same files can run an `svn update` command that allows them to access the new versions. When they do that, their local checked-out copies of the files get updated to include the changes. Their local versions get merged with the master copies from the repository, but none of their local changes are lost.

Background reading:

Subversion - A Quick Tutorial

<http://aymanh.com/subversion-a-quick-tutorial>

(Note: you won't run `svn admin` when using the class subversion repository.)

Version Control with Subversion

<http://svnbook.red-bean.com/en/1.1/index.html>

Dealing with conflicts:

<http://svnbook.red-bean.com/en/1.1/ch03s05.html#svn-ch-3-sect-5.4>

Subversion features that will matter to you:

Subversion has a global revision number for the whole repository. This means that you can very easily retrieve snapshots of your entire source code and data file tree as they existed at a given revision number. For example, let's say you do some research using revision number three of your source code tree. You then write a paper on your results and send it to a journal, while making a note of which revision of the source code you used to do the work. Weeks pass, and you continue to make changes and improvements to the source code tree, perhaps even modifying some of the algorithms you used for the paper. Weeks or months later, you receive the comments back from the reviewers, who want you to make some modifications to your analysis or perhaps do more work. Because all your code is under version control in subversion, you can easily retrieve a snapshot of your entire source code tree, plus data files, as they existed when you submitted the paper. This saves you endless headaches of trying to re-construct complex analyses using archived files and whatever notes you may (or may not) have made while doing the work.

Subversion is a client and a server. This means that you can easily make working copies of all your files on any computer where the subversion client (`svn`) is installed. This saves you the the work of transferring files from computer to computer. For example, you could work on a class project at home, check in your work to the repository, come to class, and then check out all of your work onto the lab computer.

Getting a copy of the repository. Scott Wood set up a class subversion repository for you to use. Each student in the class should have a directory within the repository to work with – it will be named for your user id.

To check out a copy of your part of the repository, you would use the command `svn co` (for `svn checkout`.)

For example, a student named Mary Stevens (user name `mstev`) would type:

```
$ svn co https://cci-subv.uncc.edu/svn/binf_prog --username mstev 6111
```

This command would check out a copy of directories and files and place these in a new directory (folder) called 6111. Only files for which user `mstev` has “read” permissions in the repository will be checked out. These will include a directory called `mstev` and another directory called `public`.

On a Windows computer, you would download a Windows subversion client. One good option for this is TortoiseSVN. You can also use the `svn` client that comes with the `cygwin` distribution, if you have installed `cygwin`.

To get TortoiseSVN, see: <http://tortoisesvn.tigris.org/>.

Subversion lets you work with others easily. Depending on the system setup, subversion makes it possible for other users to view and/or modify the files (source code, data, etc) in your repository. In addition, subversion tracks all changes you or other users make to your shared files in the repository. Each time some-one checks in a change to the repository, subversion will invite them

to write a log message, which should describe their changes. Subversion allows you to compare new and old versions of files and also read these log messages.

Compare two revisions of the same file:

```
$ svn compare -r R1:R2 [filename]
```

Revert to previous revisions of a repository:

```
$ svn update -r R
```

Subversion also makes it (relatively) easy to resolve potential conflicts between files. For example, imagine that two students (Mary and Steve) are working together on a project. Mary edits a function a file (let's call it `funcs.py`) and Steve edits the same function in his checked-out copy of the same file. Then, Mary checks in her version to the repository. When Rick tries to check in his version, he'll get a message telling him that a newer version is available and that he needs to update his copy. When he runs the update, he'll get another message letting him know that there is a conflict between the most recent version of the file (in the repository) and his own, modified version. At this point, subversion will automatically make copies in his working directory of all relevant versions of the file and will invite him to resolve the conflict by examining the versions by hand. Once he has decided how to resolve the conflict, he runs

```
svn resolved
```

and checks in a final, authoritative version of the file.

Subversion Lab.

1. Open a terminal. Type `man svn` and read the man pages for the `svn` command. Then, type `help svn`. You should see the following output, or something similar. Commands you will use most often are shown in bold:

```
$ svn help
usage: svn <subcommand> [options] [args]
Subversion command-line client, version 1.4.4.
Type 'svn help <subcommand>' for help on a specific subcommand.
Type 'svn --version' to see the program version and RA modules
  or 'svn --version --quiet' to see just the version number.

Most subcommands take file and/or directory arguments, recursing
on the directories.  If no arguments are supplied to such a
command, it recurses on the current directory (inclusive) by
default.

Available subcommands:
  add
  blame (praise, annotate, ann)
  cat
  checkout (co)
  cleanup
  commit (ci)
```

```

copy (cp)
delete (del, remove, rm)
diff (di)
export
help (?, h)
import
info
list (ls)
lock
log
merge
mkdir
move (mv, rename, ren)
propdel (pdel, pd)
propedit (pedit, pe)
propget (pget, pg)
proplist (plist, pl)
propset (pset, ps)
resolved
revert
status (stat, st)
switch (sw)
unlock
update (up)

```

Subversion is a tool for version control.

For additional information, see <http://subversion.tigris.org/>

see:

http://www.cci.uncc.edu/coit_new/techsupport/Subversion_HowTo.html

2. Scott Wood has set up a single subversion repository that we will use for the class. The repository has the following structure:

```

binf_prog/user/[your files go here]
binf_prog/class/[you can read these files, but you can't modify them]
binf_prog/public/[you can read and modify these files]

```

Open a terminal window.

3. Check a copy of your part of the repository using svn.

```
svn co https://cci-subv.uncc.edu/svn/binf_prog --username [user]
```

Subversion will prompt you for a password. Enter the password you got from the instructor (via email or in person).

4. Change directories into the newly-created directory `binf_prog` and list the contents of the directory. (**Quiz:** What Unix command lists files in the current working directory?)

You should see a directory named for your user id. For example, if your user id is `mstev`, you would see a listing of the following directories:

```
mstev
public
class
```

5. Change into your user directory. This is the directory where you will add files and directories and do all your work.

6. Create a new directory called `hw5` using the `svn mkdir` command:

```
svn mkdir hw5
```

7. Place your latest version of your homework 5 file (`hw5.py`) into that directory, making sure to name it `hw5.py`. Add it to the repository using the `svn add` command:

```
svn add hw5.py
```

8. Commit your changes using `svn ci` (`svn checkin`):

```
svn ci
```

9. What should happen next is that your default editor (emacs on the Mac computers in the lab) will launch. Type in a message describing this new change to the repository. Then save the file (by default subversion will assign a file name) and quit emacs. (**Reminder:** `C-x C-s` to save, `C-x C-c` to quit.) You should then see some messages printed to the console summarizing the changes you've checked in.

10. At home or another computer, check a copy of the files as you did above. Edit the file as usual, and then, when you have reached a stopping point, check in your changes.

To get full credit on this lab assignment, check in your most recent `hw5.py` file by the beginning of the next class period. It does NOT have to be your final version – you can check in new versions of `hw5.py` many times before the due date. The TA will grade the version that was current as of the due date.