Introduction to Subversion

A Version Control System

Duncan Murdoch

Department of Statistical and Actuarial Sciences
University of Western Ontario

November 27, 2006

Outline

1. What is version control for?
   - Peace of mind
   - Managing multiple copies
2. What is Subversion?
   - Version control choices
   - Subversion details
3. Some Practical Advice
   - What to commit, and where
   - Using Subversion in DSAS
4. Further Resources

Backup

- The First Law of Computing:[1]
  1. Back it up.
- The Second Law of Computing:
  2. Do it now.
- You never know when you'll have a hardware failure, you'll make a stupid mistake, or you'll just change your mind.
- Version control lets you revert to earlier versions of files, by keeping a record of all old versions in a central repository.

[1] Fred Davidson, *Principles of Statistical Data Handling*

Backup

- The First Law of Computing:[1]
  1. Back it up.
- The Second Law of Computing:
  2. Do it now.
- You never know when you'll have a hardware failure, you'll make a stupid mistake, or you'll just change your mind.
- Version control lets you revert to earlier versions of files, by keeping a record of all old versions in a central repository.

[1] Fred Davidson, *Principles of Statistical Data Handling*

Backup

- The First Law of Computing:[1]
  1. Back it up.
- The Second Law of Computing:
  2. Do it now.
- You never know when you'll have a hardware failure, you'll make a stupid mistake, or you'll just change your mind.
- Version control lets you revert to earlier versions of files, by keeping a record of all old versions in a central repository.

[1] Fred Davidson, *Principles of Statistical Data Handling*

Change log

- In a big project that takes a long time to complete (e.g. a thesis or a paper), you often make changes and then forget exactly why you had to make them.
- Version control systems encourage you to record a short note describing the purpose of each change. These are very useful months or years later to remind yourself of what was in your mind.
- *R was entered into version control in September, 1997. Since then it has had approximately 40,000 changes.*

Collaboration

- When working with co-authors, it is difficult to co-ordinate who will work on what. If two authors make changes to the same file, it is very easy for one set of changes to get lost.
- Version control provides strategies for dealing with this. Multiple authors can edit copies of the same files at the same time, and no changes will be lost.

Distributed files and staging

- I work regularly on 6 different computers: two laptops, two desktops, two servers. It is hard to co-ordinate with myself!
- Version control lets each computer get a copy of a current project while online, then I can work offline, and merge the changes later.
- Version control lets me develop web pages on any local machine, then when all the files look okay and work together, update the web server to include them.

Archival snapshots

- When sending documents or computer code to others, it's important to know exactly what you sent. You may want to keep working on the project, but when you get comments back, they may refer to specific lines of the older files.
- Version control lets you make a snapshot of the state of a large number of files at a fixed point in time, *using almost no disk space*. It just records which versions of each file are current. Later you can reconstruct that version.

Branching

- In some projects, you want several branches to remain active at once.
- In R, there is the *R-patched* branch where only bug fixes are allowed, the *R-devel* branch where new changes are developed, various temporary branches where people try out changes to see how well they work, and a large number of inactive branches related to previous releases.
- Version control lets all of these co-exist in one central *repository*. They are all stored very efficiently: only the differences between versions are kept, not whole copies of everything.

Version control systems

- There are a lot of different version control systems: RCS, CVS, Subversion, SVK, BitKeeper, Arch, Git, Perforce, Visual SourceSafe, . . .
- Subversion is a free, open-source, multi-platform one, available from `http://subversion.tigris.org`.
- TortoiseSVN is a front-end to Subversion for Windows users, available from `http://tortoisesvn.tigris.org`.
- *TortoiseSVN is one of the reasons I prefer Windows to Unix.*

The Subversion repository

- Subversion manages files in a hierarchical file system, just like Windows or Unix.
- Unlike those, it is a *versioned* file system. Conceptually, it contains a complete copy of the file system for every revision.
- The R repository `https://svn.r-project.org/R` contains about 40,000 versions of the file system.
- Storage is very efficient: only the *deltas* are saved between versions, i.e. enough information to convert one version into the next. Since most versions only contain small changes from the previous ones, it doesn't use much space.
- Some other information is saved to allow bigger jumps too, for efficiency: the *skip-delta algorithm*.

The Subversion repository

- Subversion manages files in a hierarchical file system, just like Windows or Unix.
- Unlike those, it is a *versioned* file system. Conceptually, it contains a complete copy of the file system for every revision.
- The R repository `https://svn.r-project.org/R` contains about 40,000 versions of the file system.
- Storage is very efficient: only the *deltas* are saved between versions, i.e. enough information to convert one version into the next. Since most versions only contain small changes from the previous ones, it doesn't use much space.
- Some other information is saved to allow bigger jumps too, for efficiency: the *skip-delta algorithm*.

The Subversion repository

- Subversion manages files in a hierarchical file system, just like Windows or Unix.
- Unlike those, it is a *versioned* file system. Conceptually, it contains a complete copy of the file system for every revision.
- The R repository `https://svn.r-project.org/R` contains about 40,000 versions of the file system.
- Storage is very efficient: only the *deltas* are saved between versions, i.e. enough information to convert one version into the next. Since most versions only contain small changes from the previous ones, it doesn't use much space.
- Some other information is saved to allow bigger jumps too, for efficiency: the *skip-delta algorithm*.

Working copies

- Users check out a particular revision of some part of the file system, e.g.
  `svn checkout https://svn.r-project.org/R/trunk`
- To Windows or Unix, it looks like regular hierarchy of files, but extra version information is saved in `.svn` subdirectories.
- Use `svn update` to get the latest version. No need to give the repository again, it's stored in the `.svn` information.

Saving changes

- Subversion can tell whether you have modified any of the files, and exactly what the modifications are: *even when you are offline*.
- Use `svn status` to see the status of all of your files.
- TortoiseSVN will show the status in Explorer.
- Use `svn commit -m"Some message"` to save your changes once you are done (and online).

Saving changes

- Subversion can tell whether you have modified any of the files, and exactly what the modifications are: *even when you are offline*.
- Use `svn status` to see the status of all of your files.
- TortoiseSVN will show the status in Explorer.
- Use `svn commit -m"Some message"` to save your changes once you are done (and online).

Versioning works best on text files

- Subversion knows how to merge changes in text files, so if two users modify the same file, it's not a problem (unless they both modify the same line).
- It doesn't try to merge binary files, it makes you choose one set of changes or the other.
- The delta calculations within Subversion work best on text files; often a small change to a binary file affects large parts of it.
- Rule of thumb: If you had to write it, commit it. If it is generated by the computer, don't.

Versioning works best on text files

- Subversion knows how to merge changes in text files, so if two users modify the same file, it's not a problem (unless they both modify the same line).
- It doesn't try to merge binary files, it makes you choose one set of changes or the other.
- The delta calculations within Subversion work best on text files; often a small change to a binary file affects large parts of it.
- Rule of thumb: If you had to write it, commit it. If it is generated by the computer, don't.

Versioning works best on text files

- Subversion knows how to merge changes in text files, so if two users modify the same file, it’s not a problem (unless they both modify the same line).
- It doesn’t try to merge binary files, it makes you choose one set of changes or the other.
- The delta calculations within Subversion work best on text files; often a small change to a binary file affects large parts of it.
- Rule of thumb: If you had to write it, commit it. If it is generated by the computer, don’t.

Once committed, files are hard to remove

- Only the administrator of the repository can permanently delete a file, and it's slow and time consuming.
- Don't commit confidential information (e.g. grades, passwords, etc.) unless it is encrypted: assume it will be around forever!
- Don't commit very large files unless they are very important: they may make your repository inconveniently large.

Once committed, files are hard to remove

- Only the administrator of the repository can permanently delete a file, and it's slow and time consuming.
- Don't commit confidential information (e.g. grades, passwords, etc.) unless it is encrypted: assume it will be around forever!
- Don't commit very large files unless they are very important: they may make your repository inconveniently large.

Once committed, files are hard to remove

- Only the administrator of the repository can permanently delete a file, and it's slow and time consuming.
- Don't commit confidential information (e.g. grades, passwords, etc.) unless it is encrypted: assume it will be around forever!
- Don't commit very large files unless they are very important: they may make your repository inconveniently large.

How to organize your repository

- The standard organization is good:

/trunk/...	Active versions of files
/tags/<tagname>/...	Snapshots
/branches/<branchname>/...	Branches

- A useful branch is /branches/inactive. Move trunk directories here when you don't need them now, but may in the future, e.g. old papers, course materials for old courses, etc.

How to use it in DSAS

- Currently Subversion is not installed on fisher or our other servers, except on my account: but you are free to use it.
- Add `/home/faculty/murdoch/bin` to the end of your PATH.
- It is easy to install in Windows; you can use only TortoiseSVN, but I recommend getting both TortoiseSVN and the command line version.

How to use it in DSAS

- Currently Subversion is not installed on fisher or our other servers, except on my account: but you are free to use it.
- Add `/home/faculty/murdoch/bin` to the end of your PATH.
- It is easy to install in Windows; you can use only TortoiseSVN, but I recommend getting both TortoiseSVN and the command line version.

Setting up a repository

- There are several different ways to set up a server.
- For a single user, `ssh` access is easiest. As long as you have `ssh` access to your account, you can use this.
- For a large shared repository, `http` or `https` access using Apache is best. This needs to be set up by a system administrator.

The Subversion book

- *Version Control with Subversion* is an excellent book, available in print from O'Reilly.
- Also available for free at `http://www.svnbook.com`.
- There are several other books around; take a look!

Web resources

- Besides the book, other online resources include a number of short presentations mainly aimed at coders, e.g. the one referenced on our web page.
- The Subversion Users mailing list is very helpful.