Introduction to Subversion

Josh Glover `<jmglov@jmglov.net>`
(2005/03/05)

About

- Subversion is “a compelling replacement for CVS”
- Provides you with a way to record the revision history of any document
- You can select versions by revision number, date, tag, or branch
- Allows for concurrent editing of files

SVN vs. CVS

- Since Subversion is meant to be a replacement for CVS, it supports most of CVS's features (exceptions usually come when a CVS “feature” could more correctly be called a “bug”)
- SVN offers a command-line interface that is very familiar that of CVS
- Directories, renames, and file meta-data are versioned, unlike CVS
- Commits are truly atomic
- Revision numbers and log messages are per commit, not per-file as in CVS

Client / Server Architecture

- Server can run as an Apache module, using WebDAV / DeltaV, giving you all of Apache's fine-grained configuration and access controls “for free”
- Server can also run over SSH (very similar to CVS) or on a local machine
- Client keeps a lot more data in the working copy than does the CVS client, so you can actually get some work done off-line!
- Client and server were designed to use bandwidth efficiently, by transmitting diffs in both directions whenever possible

Terminology

- **Repository**: central location where a Subversion server stores the files associated with one or more projects
- **Subversion Root**: directory where all **repositories** are located
- **Working Copy**: local copy of the root directory or any subdirectory of a **repository**

Creating a Subversion Root

- A Subversion Root is nothing more than a directory where you decide to keep all of your Subversion repositories
- You may create a repository anywhere, so you do not technically need a Subversion Root, but creating one or more is recommended for the sake of organisation

SVN Commands

- Just like CVS, SVN uses a sub-command interface:
  `svn <command> {<options>} {<arguments>}`
- Unlike CVS, SVN has no concept of global versus command-specific options. This is A Good Thing (TM).
- Common options:

```
--force            be forceful
-q                 be quiet
-r <rev>[:<rev>]   operate on this revision(s)
-v                 be verbose
--targets <file>   reads <file> for additional args
-N                 non-recursive (like CVS)
```

- The most useful command: `help`

Creating a Repository

- Create your Subversion Root directory:
  ```
  mkdir ~/svnroot
  ```
- Choose a data-store: Berkeley DB or FSFS
  http://svnbook.red-bean.com/en/1.1/ch05.html#svn-ch-5-sect-1.3
- To create a repository with a FSFS data-store (recommended):
  ```
  svnadmin create --fs-type fsfs \
   ~/svnroot/test-fsfs-repos
  ```
- To create a repository with a Berkeley DB data-store (do so at your own peril!):
  ```
  svnadmin create ~/svnroot/test-bdb-repos
  ```

Why not Berkeley DB?

- Prone to “wedging”, i.e. getting stuck in an inconsistent state due to various synchronisation issues
- When this happens, the repository must be manually recovered by the Subversion Administrator
- I have never seen data loss due to BDB wedging, only loss of valuable programmer time

Projects and Repositories

- Unlike in CVS, where revision numbers apply only to individual files, Subversion revision numbers apply not only to all of the files, directories and metadata in a project, but repository-wide!
- Because of this, I recommend using a separate repository for each project

Importing a New Project

- Create the repository

```
svnadmin create --fs-type fsfs \
 /home/jmglov/svnroot/test-new-project
```

- Create the project skeleton directory:

```
for i in branches tags trunk; do
  mkdir -p test-new-project/$i
done
```

- Import the skeleton:

```
cd test-new-project
svn import -m 'initial import' \
 file:///home/jmglov/svnroot/test-new-project
```

- Checkout the project:

```
cd ..
rm -rf test-new-project
svn co file:///home/jmglov/svnroot/test-new-project
```

Importing an Existing Project

- Create the repository

```
svnadmin create --fs-type fsfs \
 /home/jmglov/svnroot/test-exist-project
```

- Create the project skeleton directory and copy the project files into it:

```
for i in branches tags trunk; do
  mkdir -p test-exist-project/$i
done
rsync -av ~/coding/exist-project/ \
 ~/test-exist-project/trunk/
```

- Import the skeleton, then checkout the project as for the new project

Importing a CVS Project (1/2)

- Install `cvs2svn`: http://cvs2svn.tigris.org/
- Basic usage:
  `cvs2svn -s <svn_repos> <cvs_repos>`
- Always run the first time with the `--dry-run` flag, which just simulates the conversion
- You do not need to create the repository first, as `cvs2svn` will do so for you; however, if you want a repository with an FSFS data store, you must create it first and pass the `--existing-svnrepos` flag to `cvs2svn`

Importing a CVS Project (2/2)

- Example:

```
svnadmin create --fs-type fsfs \
 /home/jmglov/svnroot/test-cvs-project

cvs2svn --existing-svnrepos \
 -s /home/jmglov/svnroot/test-cvs-project \
 /data/cvsroot/cvs-project

svn co \
 file:///home/jmglov/svnroot/test-cvs-project
```

- Note that `cvs2svn` creates the proper `branches/`, `tags/`, and `trunk/` subdirectories automatically!

Working Copies

- A “working copy” is anything that has been checked out of a repository
- The working copy contains all of the files and directories in the project, plus a special “administrative directory”, `.svn`, in each directory in the working copy
- The administrative directory contains one file of interest, entries, which is an XML record of each file and sub-directory that is under Subversion's control in the current directory of the working copy

`svn status` (1/2)

- `svn status` is used to list information about files in the working copy
- Use the `-u` option to force Subversion to query the repository (it normally only checks the `.svn/entries` file, which is great for offline work, but you will miss changes that have been made in other working copies and commited)
- Use the -v option to increase verbosity (SVN normally only shows “interesting files”, i.e. ones that have changed or are not under SVN control)

`svn status` (2/2)

- Running `svn stat -uv` in our newly imported project's working directory yields what we would expect:

```
                 1          1 jmglov         .
Status against revision:      1
```

- In the first line, the first field that we see is the working revision, the second is the repository revision, and the last is the pathname
- The second line tells us what the current repository revision is
- This means that only the current directory, “.”, is under control, and it is up to date
- Run `svn help stat` for more information

Adding a File (1/2)

- Let's write a haiku:

```
cd ~/test-new-project/trunk/
cat >haiku.txt <<'EOF'
CVS no good?
Subversion to the rescue:
Branching is easy!
EOF
```

- Just like CVS:

```
svn add haiku.txt
```

- And just as in CVS, adding something to the repository is a two-step process; you must commit your changes to the repository:

```
svn commit -m 'added' haiku.txt
```

  (more on committing later)

Adding a File (2/2)

- One improvement over CVS: `svn add` adds recursively, so you can add whole directory trees at once!
- Interesting options to `svn add`:

`--auto-props`	automatic properties (more on this later)

`svn commit` (1/2)

- Use `svn commit` to communicate your changes to the repository, which makes it available to other working copies (these may be yours on different machines, or may belong to other developers on your project)
- Standard usage:
  ```
  svn commit <file> {<file>}
  ```
- Useful options to svn commit:

`-m <msg>`	specify a log message
`--editor-cmd <cmd>`	specify an editor for log messages

`svn commit` (2/2)

- SVN will normally open an editor (usually `vi`, but your `$EDITOR` environment variable is honoured) and prompted to enter a log message:

```
--This line, and those below, will be ignored--

A    freeform.txt
```

- Save and exit to use the log message and commit; exit without saving and SVN will prompt:

```
Log message unchanged or not specified
a)bort, c)ontinue, e)dit
```

- You may pass a log message on the command-line by using the `-m` flag:

```
svn commit -m 'added' haiku.txt
```

`svn update` (1/2)

- Use `svn update` to synchronise your working copy with the repository
- But how do you know when you need to update?

```
: jmglov@laurana; svn stat -u
       *          3   freeform.txt
Status against revision:      4
```

- Of course, updating can never hurt anything (unless you specifically don't want newer code)
- Standard invocation:

```
svn up
```

`svn update` (2/2)

- Useful options to `svn update`:

`-r <rev>`	update to revision `<rev>` instead of HEAD
`--diff3-cmd <cmd>`	use `<cmd>` as the merge command

- For example, you decide you liked freeform.txt better in revision 3:

```
: jmglov@laurana; svn stat -u
Status against revision:      4
: jmglov@laurana; svn up -r 3 freeform.txt
U  freeform.txt
Updated to revision 3.
: jmglov@laurana; svn stat -u
      *        3   freeform.txt
Status against revision:      4
```

`svn diff` (1/3)

- In the course of editing, it is quite likely that you make so many changes to a file that, come time to commit, you have no idea how the file that you want to commit differs from the latest version in the repository
- `svn diff` to the rescue!
- Standard usage:
  `svn diff <filename>`
- Output is a good old unified diff
- Running `svn diff` with no arguments is like running `diff -ru`

svn diff (2/3)

```
: jmglov@laurana; svn diff freeform.txt
Index: freeform.txt
===================================================================
--- freeform.txt        (revision 4)
+++ freeform.txt        (working copy)
@@ -1,3 +1,7 @@
+Freeform Poem
+by Josh Glover
+---------------
+
 I am full
 of angst and thus care
 Not at all for metre or even punctuation!
@@ -5,3 +9,10 @@

 Or not, your choice
 My friend
+
+This poem sucks as do
+many that ignore metre; I fart in
+Their
+general direction! I wave my
+private Parts
+at their AUNTIES!
```

`svn diff` (3/3)

- Useful options to `svn diff`:

`-r <rev>[:<rev>]`	diff against this revision or range of revisions
`-x <args>`	bundled args to GNU diff
`--diff-cmd <cmd>`	use this diff command
`--no-diff-deleted`	ignore deleted files (diffs would otherwise be entire file)

- To see everything that changed in revision 124:

```
svn diff -r 124
```

- To see everything that changed between revisions 100 and 125 for file `foobar.c`:

```
svn diff -r 100:125 foobar.c
```

svn log (1/2)

- If you get into the habit of always writing meaningful log messages, svn log can be a great way to figure out what changed
- Standard usage: `svn log <filename>`

```
: jmglov@laurana; svn log freeform.txt
------------------------------------------------------------------------
r5 | jmglov | 2005-03-29 23:22:21 -0500 (Tue, 29 Mar 2005) | 1 line

added title, another stanza at the end
------------------------------------------------------------------------
r4 | jmglov | 2005-03-29 21:25:50 -0500 (Tue, 29 Mar 2005) | 1 line

added a new stanza at the end
------------------------------------------------------------------------
r3 | jmglov | 2005-03-29 21:02:05 -0500 (Tue, 29 Mar 2005) | 2 lines

added

------------------------------------------------------------------------
```

`svn log` (2/2)

- Useful options:

`-r <rev>[:<rev>]`	show log message(s) for this revision or range of revisions
`--stop-on-copy`	do not go back beyond a copy operation in the revision history
`--xml`	XML output (useful for web apps, n'est ce pas?)

`svn blame` (1/2)

- When working concurrently on a project with other developers, sooner or later, someone will introduce a bug that breaks everything
- `svn blame` provides a way to know who done it, and when
- Standard usage: `svn blame <filename>`
- Useful options:

`-r <rev>[:<rev>]`	operate on this revision or range of revisions

svn blame (2/2)

```
: jmglov@laurana; svn blame freeform.txt
    22    plyah8r Freeform Poem 7h47 5uX0rZ!
     5     jmglov by Josh Glover
     5     jmglov --------------
     5     jmglov
     3     jmglov I am full
     3     jmglov of angst and thus care
     3     jmglov Not at all for metre or even punctuation!
     3     jmglov Fear me.
     4     jmglov
     4     jmglov Or not, your choice
     4     jmglov My friend
     5     jmglov
     5     jmglov This poem sucks as do
     5     jmglov many that ignore metre; I fart in
     5     jmglov Their
     5     jmglov general direction! I wave my
     5     jmglov private Parts
     5     jmglov at their AUNTIES!
```

`svn revert`

- Sometimes, you've just bolloxed your working copy up so badly that the only way forward is to throw it all away and start from the last known good revision
- Standard usage: `svn revert <filename>`
- Useful options:

`-R`	act recursively

`svn copy` (1/2)

- `svn copy` provides for the case when you want to “fork” a file (e.g. a library has gotten too big and you want to split it up into several .c files)
- Standard usage:
  ```
  svn copy <old_file> <new_file>
  ```
- Note that you need to commit to make the repository notice the copy:
  ```
  svn cp freeform.txt iambic-pentametre.txt
  svn commit \
   -m 'going to rework freeform poem in proper metre' \
   freeform.txt iambic-pentametre.txt
  ```

`svn copy` (2/2)

- SVN remembers the history of the copied file
- To find out where the copy was actually made, use the `--stop-on-copy` switch to `svn log`:

```
: jmglov@laurana; svn log --stop-on-copy iambic-pentametre.txt
------------------------------------------------------------------------
r9 | jmglov | 2005-03-30 00:06:31 -0500 (Wed, 30 Mar 2005) | 1 line

first attempt to smooth this over
------------------------------------------------------------------------
r7 | jmglov | 2005-03-29 23:57:53 -0500 (Tue, 29 Mar 2005) | 1 line

going to rework freeform poem in proper metre
------------------------------------------------------------------------
```

- This indicates that `iambic-pentametre.txt` was copied in revision 7

svn move

- CVS provided no easy way to rename files, but luckily, SVN does!
- Standard usage:

```
svn move <old_file> <new_file>
```

- Note that you need to commit to make the repository notice the move, and you need to commit both the old file and the new one (SVN treats this as a copy and a delete):

```
svn mv haiku.txt svn-haiku.txt
svn commit -m 'renamed haiku' \
 haiku.txt svn-haiku.txt
```

Conflicts (1/5)

- If you use Subversion for long enough (especially in large projects with many developers), you will eventually see a dreaded conflict:

```
: jmglov@laurana; svn up
C  haiku.txt
Updated to revision 11.
```

- The big, fat “C” means that SVN tried to merge the differences in the file automatically, but failed
- All is not lost! Unlike CVS's fairly primitive conflict resolution, SVN gives you a wealth of options.

Conflicts (2/5)

- There is the old-school way, opening the file in a text editor and fixing it:

```
<<<<<<< .mine
+-----------------+
| Haiku           |
| by Josh Glover |
+-----------------+
=======
Haiku
by Josh Glover
+-+-+-++-+-+-+
>>>>>>> .r11

CVS no good?
Subversion to the rescue:
Branching is easy!
```

Conflicts (3/5)

- The conflicts are delimited by the
```
<<<<<<< .mine
[...]
=======
[...]
>>>>>>> .r11
```
- The “.mine” after the row of less-thans indicates that the section up until the equal signs is from the working copy
- The “.r11” after the row of greater-thans indicates that the section above is from revision 11
- Simply merge the two together to get what you want, then removed the lines containing the less-thans, greater-thans, and equal signs

Conflicts (4/5)

- More often than not, one version is just wrong
- SVN knows this, and gives you a few versions to easily choose from:

```
: jmglov@laurana; ls haiku.txt*
haiku.txt  haiku.txt.mine  haiku.txt.r10  haiku.txt.r11
```

- If you are sure that the version in your working copy is right, simply:

```
cp haiku.txt.mine haiku.txt
```

- Likewise, if you want to go with the newest revision in the repository:

```
cp haiku.txt.r11 haiku.txt
```

- Note these are not `svn copy` commands--no need to involve SVN just yet

Conflicts (5/5)

- When you have fixed the file one way or another, be sure to let SVN know:

```
: jmglov@laurana; svn resolved haiku.txt
Resolved conflicted state of 'haiku.txt'
: jmglov@laurana; ls haiku.*
haiku.txt
```

- SVN cleans up the temporary but oh-so-helpful files, and you are ready to commit:

```
svn commit \
 -m 'added a fancy border to the title' \
 haiku.txt
```

Properties (1/3)

- SVN even keeps metadata under revision control
- This is enabled by “properties”, which are simply key/value pairs
- `svn propset <property> <value> <path>`
- For example, to set the copyright on all files in the trunk:

```
cd trunk/
svn propset copyright \
 'Copyright (c) 2005 and onwards,'\
' Josh Glover <jmglov@jmglov.net>' \
 -R ./
svn commit -m 'set copyright'
```

Properties (2/3)

- To list properties: `svn proplist <path>`:

```
: jmglov@laurana; svn proplist traditional/haiku.txt
Properties on 'traditional/haiku.txt':
  copyright
```

- To display the values of properties:
  `svn propget <property> <path>`:

```
: jmglov@laurana; svn propget copyright traditional/haiku.txt
Copyright (c) 2005 and onwards, Josh Glover <jmglov@jmglov.net>
```

- Or `svn proplist --verbose <path>`:

```
: jmglov@laurana; svn proplist --verbose traditional/haiku.txt
Properties on 'traditional/haiku.txt':

  copyright : Copyright (c) 2005 and onwards, Josh Glover <jmglov@jmglov.net>
```

- To change properties, use `svn propset` again or:
  `svn propedit <property> <path>`
- To delete properties:
  `svn propdel <property> <path>`

Properties (3/3)

- Special properties:
  - `svn:eol-style` end-of-line markers
  - `svn:executable` if set, SVN will set the exec bit
  - `svn:externals`
  - `svn:mime-type`
  - `svn:ignore` set on a directory; multi-line list of patterns for files to ignore
  - `svn:keywords` in file: `$keyword$`
    - `LastChangedDate`
    - `LastChangedRevision`
    - `LastChangedBy`
    - `HeadURL`
    - `Id`

Binary Files

- CVS made you jump through hoops when dealing with binary files, but not SVN:

```
: jmglov@laurana; svn add gentoo-matrix-aq-1024x768.jpg
A  (bin)  gentoo-matrix-aq-1024x768.jpg
```

- Look at that! SVN noticed that my JPEG is a binary file, and will handle it as such
- SVN's DeltaV algorithm can actually handle “diffs” of binary files, so having lots of revisions does not bloat the repository like it would in CVS, which has to store **the entire file for every revision**!

Branching / Tagging (1/2)

- Where SVN really shines is in its handling of branches and tags
- CVS made this possible, but in all but the most trivial of cases, complications would inevitably arise
- Tagging:

```
svn up
cd ../tags
svn cp ../trunk 2005-03-30_PRE_RELEASE
svn commit \
 -m 'made a pre-release tag' \
 2005-03-30_PRE_RELEASE
```

Branching / Tagging (2/2)

- Branching:

```
svn up
cd ../branches
svn cp ../trunk BLEEDING_EDGE
svn commit \
 -m 'created branch for new features' \
 BLEEDING_EDGE
```

- Looks a lot like a tag, right?
- Branches and tags are interchangeable: commit to a “tag” and it becomes a
  “branch” automatically!
- If you do this, you may find it useful to move the “tag” into the `branches/` directory:

```
cd ../
svn mv tags/2005-03-30_PRE-RELEASE \
 branches/2005-03-30_PRE-RELEASE
svn commit -m 'turned tag into branch' \
 tags/2005-03-30_PRE-RELEASE \
 branches/2005-03-30_PRE-RELEASE
```

Merging (1/4)

- OK, so you've branched, done lots of development on the branch, and are ready to merge back to the trunk (trunk is really nothing more than the “main” branch)
- Use `svn merge`:

  `svn merge -r <rev>:<rev> <source>`
- How does one determine the starting and ending revisions for the range?

```
: jmglov@laurana; svn log --stop-on-copy ../branches/BLEEDING_EDGE
------------------------------------------------------------------------
r13 | jmglov | 2005-03-30 11:55:04 -0500 (Wed, 30 Mar 2005) | 1 line

created branch for new features
------------------------------------------------------------------------
: jmglov@laurana; svn stat -u
Status against revision:     16
```

Merging (2/4)

- Now carry out the merge:

```
: jmglov@laurana; svn merge -r 13:16 \
 file:///home/jmglov/svnroot/\
 test-new-project/branches/BLEEDING_EDGE
A   modern
A   post-modern
A   post-modern/freeform.txt
A   traditional
A   traditional/haiku.txt
A   traditional/iambic-pentametre.txt
D   haiku.txt
D   freeform.txt
D   iambic-pentametre.txt
```

- You can also use the HEAD revision, but I don't recommend it, for a reason that you will see in a couple of slides

Merging (3/4)

- Like most commands, `svn merge` only affects your working copy
- This gives you a chance to inspect the merge and possibly modify it before committing:

```
: jmglov@laurana; svn stat -u
D              16   haiku.txt
D              16   freeform.txt
D              16   iambic-pentametre.txt
A  +            -   modern
A  +            -   post-modern/freeform.txt
A  +            -   post-modern
A  +            -   traditional/haiku.txt
A  +            -   traditional/iambic-pentametre.txt
A  +            -   traditional
Status against revision:     16
```

Merging (4/4)

- When you are ready to commit, scroll back up to your `svn merge` command in your shell's history (the up arrow in Bash, or Ctrl-r for a reverse history search):

```
svn merge -r 13:16 \
 file:///home/jmglov/svnroot/\
 test-new-project/branches/BLEEDING_EDGE
```

- Edit it into a `svn commit` command:

```
svn commit \
 -m 'merged branches/BLEEDING_EDGE'\
' (-r 13:16) into trunk'
```

Handy Scripts

- I have written a few simple shell scripts that can save you a bit of typing
- `svn-changed` shows the files that were modified by a specific revision
- `svn-gen-patch` generates patches for one or more revisions
- `svn-merge` automates the merge / commit dance
- Get them at: `http://www.jmglov.net/unix/scripts/`

Hook Scripts (1/6)

- Subversion provides a mechanism for policy enforcement (e.g. repository path-based access control, enforcement of coding conventions), tracking development activity, and performing fine-grained repository backups
- This mechanism is “hook scripts”
- Hook scripts are simply executable programs (often written in a scripting language) that reside in the repository on the Subversion server, and are triggered by some repository event

Hook Scripts (2/6)

- Once triggered, the hook script is provided with enough information on the triggering event to determine exactly who done what to whom with what when and where
- The script may, by virtue of its output or return status, allow the action, disallow it, or in some cases, suspend it
- Hooks reside in the `hooks/` subdirectory of the repository, and when a new repository is created, the `hooks/` subdirectory is automatically populated with template scripts
- Subversion repositories currently implement five hooks

Hook Scripts (3/6)

- `start-commit`
  - Run before the commit transaction is created
  - Typically used to implement repository-level access controls
  - Passed two arguments:
    - Path to repository
    - Username attempting to commit
  - A non-zero exit value will result in the commit being disallowed, and any output written to standard error will be reported to the user's Subversion client

Hook Scripts (4/6)

- `pre-commit`
  - Run when the commit transaction is complete, but before it is committed
  - Typically used to implement access controls based on content or location *within* the repository
  - Passed two arguments:
    - Path to repository
    - Name of transaction being committed (which must be fed to `svnlook` to extract specifics)
  - A non-zero exit value will result in the commit being disallowed, and any output written to standard error will be reported to the user's Subversion client
  - `http://www.jmglov.net/unix/scripts/svn-hooks/pre-commit`

Hook Scripts (5/6)

- `post-commit`
  - Run when the commit transaction is committed, and a new revision is created
  - Typically used to provide email notification of commits or to perform fine-grained backups of the repository
  - Passed two arguments:
    - Path to repository
    - New revision that was committed (which must be fed to `svnlook` to extract specifics)
  - Exit code is ignored
  - http://www.jmglov.net/unix/scripts/svn-hooks/post-commit
  - http://www.jmglov.net/unix/scripts/svn-hooks/commit-email.pl

Hook Scripts (6/6)

- `pre-revprop-change`, `post-revprop-change`
  - Run before / after changes to unversioned revision properties (e.g. `svn:log` commit message property)
  - Typically used keep track of changes to properties using an external mechanism
  - Passed four arguments:
    - Path to repository
    - Revision whose property is being / has been modified
    - Username attempting the change
    - Name of the property
  - Non-zero exit code for `pre-revprop-change` will result in the property change being disallowed; exit code is ignored for `post-revprop-change`

Systems Administration (1/4)

- Subversion is not just for developers: it is also handy for systems administration tasks
- Two examples:
  - Config files
  - Dotfiles

Systems Administration (2/4)

- Config files: use Subversion to track changes to /etc:
  - Create a new repository:

```
svnadmin create --fs-type fsfs \
 /data/svnroot/config-files
```

  - Import /etc

```
for i in branches tags trunk; do
 mkdir -p config-files/$i
done
sudo rsync -av /etc config-files/trunk/
sudo chown -R jmglov:jmglov \
 config-files/trunk/
cd config-files/
svn import -m 'initial import' \
 file:///data/svnroot/config-files
```

Systems Administration (3/4)

- Checkout a working copy:
```
cd ..
rm -rf config-files
svn co \
 file:///data/svnroot/config-files
```
- Make a change to a file:
```
cd config-files/trunk
vim etc/make.conf
sudo cp etc/make.conf /etc/make.conf
```
- Test the change
- Commit the new revision:
```
svn commit -m 'added “sse2” to USE' \
 etc/make.conf
```

Systems Administration (4/4)

- Dotfiles can be handled the same way, where dotfiles are copied from the working directory into `${HOME}`, or your dotfiles can actually be symlinks into your working copy, i.e.:

```
: jmglov@laurana; ls -l .xemacs
lrwxrwxrwx  1 jmglov jmglov 22 Aug  3 04:25 .xemacs -> dotfiles/trunk/.xemacs
```

- For both config files and dotfiles, you can use the trunk for common files, and make branches for differences among boxen (e.g. my laptop's LCD runs at a different resolution than my desktop's monitor, so I might make branches to hold my two differing versions of `/etc/X11/xorg.conf`)

Finis

- You now know enough about Subversion to make you dangerous!
- Go forth, my sons (and daughters, as applicable), and use SVN to save thy revision history!
- A good next step is the Subversion book, which is published by O'Reilly as <u>Version Control with Subversion</u>, but also available for free online: http://svnbook.red-bean.com/
  (or if you run Gentoo Linux, in your `/usr/share/docs/` directory, provided you emerged subversion with the “docs” USE flag set)