(Re)Introduction to Subversion

Paul van Delst

Betty Petersen Memorial Library
Technical Seminar Series
Introduction

• The goal of this seminar is to
  – Introduce new users to version control concepts in the context of subversion.
  – Describe the typical features of subversion, and how you might use them in a regular work day.
  – Discuss how we can use this tool to simplify, change, or adapt how we typically manage our software.

• What is your experience with subversion, or version control in general?

• What do you want to know about subversion, or version control?
Outline

• What is subversion? What it is not.
• Version control in the WWB.
• Basic organisation: trunk, branches, and tags
• Revision numbering
• Basic Capabilities
  – Importing, checking out, editing and updating files
  – Checking the status of files
  – Adding and deleting files
  – Renaming and copying files
  – Undoing local changes
• Branching and merging
  – Undoing committed changes
• Tagging
• Configuration
  – Some configuration tips.
What is Subversion?

• Most of the material here is distilled from
  http://svnbook.red-bean.com

• What is subversion?
  – Subversion is an open source revision control system that allows one or
    more users to easily share and maintain collections of files.

• What it is not.
  – Magic.
  – It is not a substitute for management.
  – It is not a substitute for developer communication.

• There is nothing inherently special about subversion. Many other
  revision control systems exist.
  – Git, Mercurial, darcs, CVS, Perforce, ClearCase, etc.
Version control in the WWB (1)

• NCEP/EMC repository is located at
  
  https://svn.ncep.noaa.gov/emc

  The vapor mirror (read-only!) is located at
  
  file:///gpfs/v/svn/emc

  (These locations will change very soon, so parameterise them!)

• NESDIS/STAR researchers use subversion to manage the
  Microwave Integrated Retrieval System (MIRS) software. Their
  repository is located at
  
  /net/backup/backup

  on their network.

• NCEP/CPC is planning to use subversion to manage the FET
  project,
  
  http://fet.hwr.arizona.edu/ForecastEvaluationTool

  an interactive web tool to evaluate historical skill of long range
  forecasts.
Version control in the WWB (2)

- Not everyone uses subversion for version control

- NESDIS/STAR IOSSPDT (don’t ask me what that stands for…. the group includes Walter Wolf and Co) researchers use ClearCase to manage their software.
  - NPOESS related software
  - GOES-R related software
Basic organisation

• Recommended repository layout has three main directories:
  – trunk
  – branches
  – tags

• trunk
  – The main line of development.
  – Typically always in an “almost ready for release” state.

• branches
  – This is where non-trivial development is done.
  – Experimental development branch names: EXP-desc
  – Release branch name: RB-rel

• tags
  – This is where snapshots and releases go.
  – Snapshots: name.revnum.YYYY-MM-DD
  – Code releases: REL-rel
  – No development (otherwise it would be a branch)
Example: CRTM Project

Revision 3367
/crtm

[Parent Directory]
branches/
tags/
trunk/

Powered by Subversion 1.4.3 (r23084)
Example: CRTM Project branches

Revision 3367
/crtm/branches/src

[Parent Directory]
EXP-DISORT/
EXP-Extra_Layers/
EXP-Multiple_Algorithm/
EXP-RTSolution_Speedup/
EXP-RTTOV/
EXP-SOI/
EXP-SSU/
RB-1.0/
RB-1.1/
RB-1.2/
Example: CRTM Project tags
Revision Numbering

- When a subversion repository is created it starts at revision 0.

- Each subsequent commit increments the revision number by 1.

- Unlike CVS, the revision number is repository-wide so any commit increments the revision number.
  - Not a big deal; just don’t be surprised when you get back from vacation and find an update to your working copy is 100’s of revisions beyond where you left it – even if the code in that directory was not changed.

- Typically, don’t worry about the revision number value.
  - BUT: for some operations, like merging, keeping track of revision numbers is helpful.
Basic Capability - Importing Files

- Importing a new project into the repository, let’s call it `projX`. First, move to the location of your un(sub)versioned code:
  
  ```
  $ cd $HOME/projects
  $ ls
  projX  projY  projZ
  ```

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

- Directory hierarchy `projX` imported into `emc/X/trunk` in the repository.

- Log message
  
  - 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 interactively enter a log message.
Basic Capability - Checking Out Files (1)

• The previous import does not place the local projX hierarchy under version control.

• To get a local versioned hierarchy you need to obtain a working copy. Also known as a sandbox.

• Move to 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

• Additional projects can also be checked out,
  $ 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
In your working copy 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.

When you have successfully checked out your project into your workspace, you should consider deleting the original sources.

You then won’t be tempted to edit these unversioned sources and bypass Subversion.
Editing files

- Once you have created a working copy of your project(s)....
- Edit, compile, debug, and test as usual.
- The files are the same as when they were unversioned.
Basic Capability - Updating Files

• You’re happy with changes you’ve made to code, and you want to commit them to the repository.

• What if another user has changed and committed the same file(s)?
  – This is where developer communication is important

• Subversion handles this by requiring you to update your working copy to the current repository version before you can commit.

• Do this via the `svn update` command

  $ svn update hello.f90
  U hello.f90
  Updated to revision 1356.

• This merges any changes (assuming no conflicts) in the repository into your working copy.
What if there is a conflict?

• What if your `svn update` command produces this
  
  $ svn update hello.f90
  C hello.f90
  Updated to revision 1356.

• Look at a file listing
  
  $ ls hello.f90*
  hello.f90 hello.f90.mine hello.f90.r1274 hello.f90.r1356
What if there is a conflict?

• What if your `svn update` command produces this
  
  ```
  $ svn update hello.f90
  C hello.f90
  Updated to revision 1356.
  ```

• Look at a file listing
  
  ```
  $ ls hello.f90*
  hello.f90  hello.f90.mine  hello.f90.r1274  hello.f90.r1356
  ```

  This is the merged file containing *conflict markers* to highlight the conflicted areas.
What if there is a conflict?

• What if your `svn update` command produces this

  $ svn update hello.f90
  C hello.f90
  Updated to revision 1356.

• Look at a file listing

  $ ls hello.f90*
  hello.f90 hello.f90.mine hello.f90.r1274 hello.f90.r1356

This is your working copy file with your local edits as it existed before the update. No conflict markers.
What if there is a conflict?

• What if your `svn update` command produces this

```
$ svn update hello.f90
C hello.f90
Updated to revision 1356.
```

• Look at a file listing

```
$ ls hello.f90*
hello.f90 hello.f90.mine hello.f90.r1274 hello.f90.r1356
```

This is the file that you checked out before you made your local edits. The “BASE” revision.
What if there is a conflict?

- What if your `svn update` command produces this
  
  ```bash
  $ svn update hello.f90
  C hello.f90
  Updated to revision 1356.
  ```

- Look at a file listing
  
  ```bash
  $ ls hello.f90*
  hello.f90 hello.f90.mine hello.f90.r1274 hello.f90.r1356
  ```

This is the file that Subversion got from the server when you updated your working copy. The “HEAD” revision
What if there is a conflict?

- What if your `svn update` command produces this
  
  ```
  $ svn update hello.f90
  C hello.f90
  Updated to revision 1356.
  ```

- Look at a file listing
  
  ```
  $ ls hello.f90*
  hello.f90  hello.f90.mine  hello.f90.r1274  hello.f90.r1356
  ```

- How to resolve?
  - Merge the conflicted text by hand.
  - Copy one of the temporary files on top of your working file.
  - Run `svn revert hello.f90` to throw away all your local changes.

- Run `svn resolved`. This will tell Subversion you have resolved the conflict. *Subversion will not overwrite local changes unless you explicitly tell it to do so.*
Now your working copy is up to date, commit your changes to the repository using the `svn commit` command:

```
$ svn commit hello.f90
<Enter log message>
Sending hello.f90
Transmitting file data .......
Committed revision 1357.
```

Subversion will start an editor allowing 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.
- Recommend users follow GNU Change Log format so the subversion logs can be used to construct ChangeLog files.

When you exit the editor, Subversion will commit your changes to the repository, where they will become visible to all users.
Basic Capability - File Status (1)

- To determine what files are up to date and which have been locally modified in your working copy, use the `svn status` command
  
  ```
  $ svn status
  ```

- Note that this form only indicates *locally* modified items; what you have changed since your last update. *The repository is not accessed.*

- To indicate which items in your working copy are out of date, the `-u` (or `--show-updates`) switch should be used
  
  ```
  $ svn status -u
  ```

- If there is no output, then everything is up to date.
Basic Capability - File Status (2)

- To view the commit log messages for an item, use the `svn log` command
  
  $ svn log hello.f90

- To see the differences (if any) between your 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

If there is no output, there are no differences.
**Basic Capability - Adding 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.

- Before you can commit, you must schedule the file for addition using the `svn add` command
  
  ```
  $ svn add newfile.f90
  A  newfile.f90
  ```

- Now you can commit the file to the repository
  
  ```
  $ svn commit -m "Initial commit" newfile.f90
  Sending     newfile.f90
  Transmitting file data .......
  Committed revision 1358.
  ```

- If you are adding an entire directory, populate the directory first and then add the directory – this will work recursively on all its contents.
Basic Capability - Deleting Files (1)

- Removing a file from your working copy does not remove it from the repository.

- Similarly to the add subcommand, you must first schedule the file for deletion using the `svn delete` command
  
  $ svn delete oldfile.f90
  D     oldfile.f90

  Note that this also deletes the file from your working copy.

- Now you can commit the file to the repository
  
  $ svn commit -m “Removed file” oldfile.f90
  Deleting     oldfile.f90
  Transmitting file data ........
  Committed revision 1359.
Basic Capability - Deleting Files (2)

- Subsequent updates and checkouts will no longer include deleted files.

- BUT: specifying a revision number with the `update` subcommand can restore older versions of the file
  
  ```bash
  $ svn update -r1358 oldfile.f90
  A        oldfile.f90
  Updated to revision 1358.
  ```

- Two very important things to remember:
  1) *Always* use Subversion (not OS) 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.
Basic Capability - Renaming Files

- Subversion provides a shortcut compared to the usual delete-then-add procedure.

- The `svn move` command
  
  $ svn move thisfile.f90 thatfile.f90
  A thatfile.f90
  D thisfile.f90

- Followed, as always, by a commit
  
  $ svn commit -m "Renamed this to that file" thisfile.f90
  thatfile.f90
  Deleting thisfile.f90
  Adding thatfile.f90
  Transmitting file data .......
  Committed revision 1360.
This section included to reinforce the point to \textit{not} use OS commands to operate on files.

Use the \texttt{svn copy} command

\begin{verbatim}
$ svn copy thatfile.f90 otherfile.f90
A otherfile.f90
$ svn status
A + otherfile.f90
\end{verbatim}

But why use copy? And what does the “+” mean?
Basic Capability - Copying Files

- This section included to reinforce the point to *not* use OS commands to operate on files.

- Use the `svn copy` command
  
  ```
  $ svn copy thatfile.f90 otherfile.f90
  A  otherfile.f90
  $ svn status
  A  + otherfile.f90
  ```

  But why use copy? And what does the “+” mean?

  Using the Subversion command to copy a file preserves the history of that file.
Basic Capability - Copying Files

- This section included to reinforce the point to *not* use OS commands to operate on files.

- Use the `svn copy` command
  
  `$ svn copy thatfile.f90 otherfile.f90`  
  `A` otherfile.f90
  `$ svn status`  
  `A + otherfile.f90`

  But why use copy? And what does the “+” mean?

  The “+” in the status output indicates that the history is also scheduled for addition.
Basic Capability - Copying Files

• This section included to reinforce the point to *not* use OS commands to operate on files.

• Use the `svn copy` command
  
  ```
  $ svn copy thatfile.f90 otherfile.f90
  A    otherfile.f90
  $ svn status
  A   +    otherfile.f90
  ```

  But why use copy? And what does the “+” mean?

• Doing the following
  
  ```
  $ cp thatfile.f90 otherfile.f90
  $ svn add otherfile.f90
  A    otherfile.f90
  ```

  means none of the previous history of `otherfile.f90` is preserved. It is a brand new file in the eyes of Subversion.
Basic Capability - Undoing Changes

• If you haven’t committed, then the `svn revert` command will undo:
  – local edits
  – scheduling operations, i.e. 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
  ```

• If you supply no targets, the `revert` subcommand will do nothing.

⚠️ To reiterate: You *lose* the local changes you have made to
`changedfile.f90`. The BASE revision of the file is what will exist in
your working copy after the `revert` subcommand is executed.
Branching and Merging (1)

• Recall the main repository structure
  – trunk: mainline development.
  – branches: non-trivial development.
  – tags: NO development.

• Merging branches to and from the trunk is one operation where keeping track of the revision number(s) is very helpful

• Pre-v1.5 Subversion does not track information about merge operations, so one could accidentally merge the same change twice potentially leading to conflicts.

• Developers must manually track merge info.

• Use the commit log message to keep track of the revision number, or range of revisions, that are being merged.

• Reviewing the svn log output will tell you what changes have already been merged and allow you to construct subsequent merge commands.
Branching and Merging (2)

• Ensure your trunk working copy is up to date
  
  $ svn update  
  At revision 1360.  

• Create a branch off the trunk. Let’s call it EXP-MyBranch.
  
  $ svn copy trunk branches/EXP-MyBranch  
  $ svn commit  
  EXP-MyBranch branch. Created from r1360.  
  Adding branches/EXP-MyBranch  
  Committed revision 1361.

Creating a branch is done via a copy command. There is nothing inherently “branchy” about a branch – it is a branch because we say it is so.
Branching and Merging (2)

- Ensure your trunk working copy is up to date
  
  $ svn update
  At revision 1360.

- Create a branch off the trunk. Let’s call it EXP-MyBranch.
  
  $ svn copy trunk branches/EXP-MyBranch
  $ svn commit
  EXP-MyBranch branch. Created from r1360.
  Adding branches/EXP-MyBranch
  Committed revision 1361.

Identify the revision from which the branch was created in the log message when you initially commit the branch.
Branching and Merging (2)

- Ensure your trunk working copy is up to date
  
  ```
  $ svn update
  At revision 1360.
  ```

- Create a branch off the trunk. Let’s call it EXP-MyBranch.
  
  ```
  $ svn copy trunk branches/EXP-MyBranch
  $ svn commit
  EXP-MyBranch branch. Created from r1360.
  Adding branches/EXP-MyBranch
  Committed revision 1361.
  ```

- Development proceeds on the branch. Each commit log message for the branch should begin with the branch name, e.g.
  
  EXP-MyBranch branch.

  ```
  src:Coefficients/EmisCoeff subdirectory.
  * EmisCoeff_Define.f90: <...log message text...>
  ```
Branching and Merging (2)

• Ensure your trunk working copy is up to date
  
  $ svn update
  At revision 1360.

• Create a branch off the trunk. Let’s call it EXP-MyBranch.
  
  $ svn copy trunk branches/EXP-MyBranch
  $ svn commit
  EXP-MyBranch branch. Created from r1360.
  Adding branches/EXP-MyBranch
  Committed revision 1361.

• Development proceeds on the branch. Each commit log message for
  the branch should begin with the branch name, e.g.

  EXP-MyBranch branch.

  src:Coefficients/EmisCoeff subdirectory.

  * EmisCoeff_Define.f90: <...log message text...>

  First line of log message identifies the branch.
Branching and Merging (2)

- Ensure your trunk working copy is up to date
  
  \$ svn update
  
  At revision 1360.

- Create a branch off the trunk. Let’s call it EXP-MyBranch.
  
  \$ svn copy trunk branches/EXP-MyBranch
  \$ svn commit
  
  EXP-MyBranch branch. Created from r1360.
  Adding branches/EXP-MyBranch
  Committed revision 1361.

- Development proceeds on the branch. Each commit log message for the branch should begin with the branch name, e.g.
  
  EXP-MyBranch branch.

| src:Coefficients/EmisCoeff subdirectory. |
| * EmisCoeff_Define.f90: <...log message text...> |

GNU ChangeLog format adopted for the CRTM.
Branching and Merging (2)

• Ensure your trunk working copy is up to date
  $ svn update
  At revision 1360.

• Create a branch off the trunk. Let’s call it EXP-MyBranch.
  $ svn copy trunk branches/EXP-MyBranch
  $ svn commit
  EXP-MyBranch branch. Created from r1360.
  Adding branches/EXP-MyBranch
  Committed revision 1361.

• Development proceeds on the branch. Each commit log message for
  the branch should begin with the branch name, e.g.
  EXP-MyBranch branch.
  src:Coefficients/EmisCoeff subdirectory.
  * EmisCoeff_Define.f90: <...log message text...>

• One can then search log message output for all instances of commits to
  a particular branch.
Branching and Merging (3)

• Let’s say you’ve finished with **EXP-MyBranch** development for now. You’ve tested the final changes and committed them at r1417.

• Now you want to merge your branch into your **trunk** working copy.

• Remember the revision numbers!
  – Branch created at **r1361** (check the **svn log** output if you forgot)
  – Branch development ended at **r1417**.

• Make sure your trunk working copy is “clean”
  – No local edits
  – Up to date
Branching and Merging (4)

• Use the `svn merge` command. Preview your merge with the `--dry-run` switch

  $$\texttt{svn merge --dry-run -r1361:1417 https://.../EXP-MyBranch}$$

• If the preview is o.k., do the merge and run your tests. Remember, the merge is `local` to your working copy since you haven’t committed yet.
Branching and Merging (4)

• Use the `svn merge` command. Preview your merge with the `--dry-run` switch
  
  $ svn merge --dry-run -r1361:1417 https://.../EXP-MyBranch

• If the preview is o.k., do the merge and run your tests. Remember, the merge is `local` to your working copy since you haven’t committed yet.

• When you’re ready to commit the trunk merges, indicate the merged revisions in the log message
  
  `Merged EXP-MyBranch changes r1361:1417 into the trunk`

Specify the range of revisions included in this merge.
Branching and Merging (4)

• Use the `svn merge` command. Preview your merge with the `--dry-run` switch
  
  $ svn merge --dry-run -r1361:1417 https://.../EXP-MyBranch

• If the preview is o.k., do the merge and run your tests. Remember, the merge is local to your working copy since you haven’t committed yet.

• When you’re ready to commit the trunk merges, indicate the merged revisions in the log message
  
  Merged EXP-MyBranch changes r1361:1417 into the trunk

• The log messages now contain a record of what was merged, what revisions were merged, and what they were merged into (e.g. you could merge trunk changes into a branch)

• Future EXP-MyBranch merges will start at r1418.
Undoing Committed Changes

• You mistakenly committed and you want to undo.
  – This is different from using `svn revert` before a commit.

• Let’s say you’re at r1480 and you want to get back to r1475 and use that to continue work.

• Use the `svn merge` command on your working copy, but with the revision numbers in reverse order,
  $ svn merge -r1480:1475 https://…

• When you’re ready to commit, indicate what you did in the log message
  `Reverse merged r1480:1475 to undo commits`

• The log message for the commit now contains a record of what revisions were undone.
Tagging

• When you want to take a snapshot of your current development (in trunk or branch), or create a release, you tag the code in question.

• Tag the experimental development with a date. Let’s use the previous convention and call it `EXP-MyBranch.rev1481.2009-03-24`

  $ pwd
  branches/EXP-MyBranch
  $ cd ..
  $ svn copy EXP-MyBranch tags/EXP-MyBranch.rev1481.2009-03-24
  $ svn commit
  `EXP-MyBranch.rev1481.2009-03-24` tag
  Adding `tags/EXP-MyBranch.rev1481.2009-03-24`
  Committed revision 1482.
Tagging

• When you want to take a snapshot of your current development (in trunk or branch), or create a release, you tag the code in question.

• Tag the experimental development with a date. Let’s use the previous convention and call it `EXP-MyBranch.rev1481.2009-03-24`

```bash
$ pwd
branches/EXP-MyBranch
$ cd ..
$ svn copy EXP-MyBranch tags/EXP-MyBranch.rev1481.2009-03-24
$ svn commit
EXP-MyBranch.rev1481.2009-03-24 tag
Adding tags/EXP-MyBranch.rev1481.2009-03-24
Committed revision 1482.
```

Change directory so you are in the parent directory of the branch you want to tag.
Tagging

- When you want to take a snapshot of your current development (in trunk or branch), or create a release, you tag the code in question.

- Tag the experimental development with a date. Let’s use the previous convention and call it `EXP-MyBranch.rev1481.2009-03-24`

  ```
  $ pwd
  branches/EXP-MyBranch
  $ cd ..
  $ svn copy EXP-MyBranch tags/EXP-MyBranch.rev1481.2009-03-24
  $ svn commit
  EXP-MyBranch.rev1481.2009-03-24 tag
  Adding tags/EXP-MyBranch.rev1481.2009-03-24
  Committed revision 1482.
  ```

Tagging a snapshot or release is also done with the copy command. Again, we consider this a tag because we say it is.
Tagging

- When you want to take a snapshot of your current development (in trunk or branch), or create a release, you tag the code in question.

- Tag the experimental development with a date. Let’s use the previous convention and call it `EXP-MyBranch.rev1481.2009-03-24`

  ```
  $ pwd
  branches/EXP-MyBranch
  $ cd ..
  $ svn copy EXP-MyBranch tags/EXP-MyBranch.rev1481.2009-03-24
  $ svn commit
  EXP-MyBranch.rev1481.2009-03-24 tag
  Adding tags/EXP-MyBranch.rev1481.2009-03-24
  Committed revision 1482.
  ```

Put the tag name in the log message when you commit.
Tagging

- When you want to take a snapshot of your current development (in trunk or branch), or create a release, you tag the code in question.

- Tag the experimental development with a date. Let’s use the previous convention and call it `EXP-MyBranch.rev1481.2009-03-24`

  ```bash
  $ pwd
  branches/EXP-MyBranch
  $ cd ..
  $ svn copy EXP-MyBranch tags/EXP-MyBranch.rev1481.2009-03-24
  $ svn commit
  EXP-MyBranch.rev1481.2009-03-24 tag
  Adding tags/EXP-MyBranch.rev1481.2009-03-24
  Committed revision 1482.
  ```

- There is no further development on the tagged snapshot.
User Setup of Subversion (Unix only)

• The first time you use subversion (e.g. to checkout code) you will need to type your password.

• A configuration directory, .subversion, will be created in your $HOME directory.

• Password may be stored as clear text so ensure configuration directory is readable only by you:
  
  chmod go-rwx .subversion

  or

  chmod 700 .subversion

• Default client side behaviour of subversion can be modified via environment variables or by editing the configuration file, config.
Configuring default behaviour

• Not an exhaustive treatment of `.subversion/config` changes.

• `[helpers]` section.
  – Uncomment the `editor-cmd` entry line and set to your editor of choice,
    ```
    editor-cmd = vi
    ```
  Emacs is typically the default on Linux (IBM may be different). Can also use an environment variable
  ```
  export SVN_EDITOR=vi (sh) or setenv SVN_EDITOR vi (csh)
  ```

• `[miscellany]` section.
  – Uncomment the `global-ignores` entry line and modify accordingly,
    ```
    global-ignores = *.o *.mod <add others as needed>
    ```
  – Uncomment the `enable-auto-props` entry line and set it.
    ```
    enable-auto-props = yes
    ```

• `[auto-props]` section. Only valid if enabled.
  – Use to automatically set properties for files when they are committed. For example,
    I want keywords to be expanded in my F90/95 source,
    ```
    *.f90 = svn:keywords=Id Revision
    ```
    and I want my shell scripts to be executable,
    ```
    *.sh = svn:executable
    ```
Final Comments

• Commit early, and often.

• Branches and tags are cheap, so use them liberally.

• Use Subversion, not operating system, commands to manipulate files in your working copy. Don’t try to subvert subversion.

• Consider a regular purge of your working copy.

• Subversion can greatly ease the task of managing code development for a team.

• But, always remember, it does not obviate the need for development team members to talk to each other.
  – Who is working on what?
  – What branches are under development?
  – When will branches be merged with the trunk? What criteria?

Adopt some sort of convention so no-one is in the dark.
Where to get more information

- EMC Subversion Forum. Post questions, and/or discoveries. It’s not too helpful for non-NCEP folks without VPN so maybe setup similar on library website?

Forum for tips that one discovers via usage; or how-to questions

Forum for more conceptual questions about version control
Where to get more information

- The Subversion forum itself, http://svnforum.org, is a great resource.
Where to get more information

  - Make sure the version you are looking at corresponds to the subversion client version on your machine!

- “Pragmatic Version Control Using Subversion” is available from The Pragmatic Programmers or Amazon. Not too complicated and has some good ideas regarding practices.
Dealing with change

Satir Change Model

1. The group is at a familiar place, but there may be imbalance between the group and its environment.

2. The group confronts a foreign element that requires a response.

3. The group enters the unknown.

4. The members discover a transforming idea that shows how the foreign element can benefit them.

5. If the change is well conceived and assimilated, the group and its environment are in better accord and performance stabilises at a higher level.

The impact on group performance of a well assimilated change during the five stages of the Satir Change Model.

© stevenmsmith.com
The End

Questions?
Extra Slides
Commit Log Message Format

• May seem pedantic, but you can use the subversion logs to create ChangeLog files. CRTM use GNU ChangeLog format.
• Example log message:

  src:Coefficients/EmisCoeff subdirectory.
  * EmisCoeff_Dindle.f90: Added svn:keywords property.
    (Clear_EmisCoeff): Removed initialisation of structure dimensions.
    (Destroy_EmisCoeff): Added initialisation of structure dimensions.
    (Equal_EmisCoeff): Replaced local logical arrays for all-array value checking with loop over array elements.

  src:Coefficients/CloudCoeff subdirectory.
  * CloudCoeff_Dindle.f90 (Info_CloudCoeff): Cosmetic changes only.
Commit Log Message Format

- May seem pedantic, but you can use the subversion logs to create ChangeLog files. CRTM use GNU ChangeLog format.
- Example log message:

  src:Coefficients/EmisCoeff subdirectory.
  * EmisCoeff_Define.f90: Added svn:keywords property.
    (Clear_EmisCoeff): Removed initialisation of structure dimensions.
    (Destroy_EmisCoeff): Added initialisation of structure dimensions.
    (Equal_EmisCoeff): Replaced local logical arrays for all-array value checking with loop over array elements

  src:Coefficients/CloudCoeff subdirectory.
  * CloudCoeff_Define.f90 (Info_CloudCoeff): Cosmetic changes only.

Single line detailing the CRTM category (src, fix, scripts, external or test) and its directory location. Each subdirectory gets its own entry.
Commit Log Message Format

- May seem pedantic, but you can use the subversion logs to create ChangeLog files. CRTM use GNU ChangeLog format.
- Example log message:

  src:Coefficients/EmisCoeff subdirectory.
  * EmisCoeff_Define.f90: Added svn:keywords property.
    (Clear_EmisCoeff): Removed initialisation of structure dimensions.
    (Destroy_EmisCoeff): Added initialisation of structure dimensions.
    (Equal_EmisCoeff): Replaced local logical arrays for all-array value checking with loop over array elements

  src:Coefficients/CloudCoeff subdirectory.
  * CloudCoeff_Define.f90 (Info_CloudCoeff): Cosmetic changes only.

The log entry should mention every file that has changed.
Commit Log Message Format

- May seem pedantic, but you can use the subversion logs to create ChangeLog files. CRTM use GNU ChangeLog format.
- Example log message:

  src:Coefficients/EmisCoeff subdirectory.
  * EmisCoeff_Define.f90: Added svn:keywords property.
    (Clear_EmisCoeff): Removed initialisation of structure dimensions.
    (Destroy_EmisCoeff): Added initialisation of structure dimensions.
    (Equal_EmisCoeff): Replaced local logical arrays for all-array value checking with loop over array elements.

  src:Coefficients/CloudCoeff subdirectory.
  * CloudCoeff_Define.f90 (Info_CloudCoeff): Cosmetic changes only.

Name all the changed procedures in full. Do not abbreviate
  (*_EmisCoeff)
or combine
  ({Clear,Destroy,Equal}_EmisCoeff)
since then a search for a particular procedure would not find the entry.