



Rensselaer

**SCOREC**

SCIENTIFIC  
COMPUTATION  
RESEARCH  
CENTER

# Source Code Revision Control with Subversion

---

Christophe Dupré

May 13, 2005

Scientific Computation Research Center  
Rensselaer Polytechnic Institute, Troy, NY



# Why Revision Control?

---

- Provides a place to store your code
  - *Reduce clutter*
  - *Independent of individual accounts*
- Historical record of what was done over time
  - *Safety net*
- Synchronization between developers

# Why Use Subversion?

---

- Functional superset of CVS
- Directory versioning (rename and moves)
- Atomic Commits
- File meta-data
- True client-server model
- Cross-platform, open-source

# Subversion Architecture

---

- Each working copy has a `.svn` directory
  - *Similar to the CVS's CVS directory*
  - *Stores metadata about each file*
- Local or Network Repository
- Network access over HTTP or SSH
- Encrypted authentication
  - *Cleartext password stored in `~/.subversion`*
- Fine-grained authorization
- Command line client is `svn`

# CVS vs SVN

---

- Most CVS commands exist in SVN
  - *Checkout, commit, update, status...*
- Arguments position matters in CVS

```
$ cvs -d /cvsroot update -d
```
- Not so in SVN

```
$ svn log -r 123 foo.c
$ svn log foo.c -r 123
```

# Revisions (1)

---

- Revision numbers are applied to an object to identify a unique version of that object.
- CVS
  - *Revision numbers are per file.*
  - *No connection between two files with the same revision number.*
  - *A commit only modifies the version number of the files that were modified.*
    - foo.c rev 1.2 and bar.c rev 1.10.
  - *After commit of bar.c:*
    - foo.c rev 1.2 and bar.c rev 1.11.

## Revisions (2)

---

- Revision numbers are applied to an object to identify a unique version of that object.
- SVN
  - *Revision numbers are global across the whole repository.*
  - *Snapshot in time of the whole repository.*
  - *A commit modifies the version number of all the files.*
    - *foo.c rev 25 and bar.c rev 25.*
  - *After commit of bar.c:*
    - *foo.c rev 26 and bar.c rev 26.*
  - *foo.c rev 25 and 26 are identical.*

# Basic Work Cycle (1)

- Checkout a working copy
- Update a working copy
- Make changes
- Examine your changes
- Commit your changes

# Basic Work Cycle (2)

## ➤ Checkout a working copy

```
$ svn checkout \  
> https://svn.scorec.rpi.edu/svn/TEST/foo  
$ cd foo
```

## ➤ Update a working copy

### – *Update all files*

```
$ svn update
```

### – *Update to an older revision*

```
$ svn update -r 45
```

### – *Update an older revision of bar.c*

```
$ svn update -r 23 bar.c
```

```
$ svn update -r 1
```

```
A  changepwd.form
```

```
D  trunk
```

```
D  branches
```

```
Updated to revision 1.
```

# Basic Work Cycle (3)

## ➤ Update output

- U *foo*
  - File *foo* was (U)pdated (pulled from repository)
- A *foo*
  - File *foo* was (A)dded to your working copy
- D *foo*
  - File *foo* was (D)eleted from your working copy
- R *foo*
  - File *foo* was (R)eplaced, that is it was deleted and a new file with the same name was added.
- G *foo*
  - File *foo* received new changes and was also changed in your working copy. The changes did not collide and so were mer(G)ed.
- C *foo*
  - File *foo* received (C)onflicting changes from the server. The overlap needs to be resolved by you.

# Basic Work Cycle (4)

## ➤ Make changes

### – *Add new files and directories*

```
$ touch README.txt
```

```
$ svn mkdir Presentations
```

```
$ touch Presentations/simple.txt
```

```
$ svn add Presentations/simple.txt README.txt
```

### – *Delete files*

```
$ svn rm foo
```

### – *Rename files*

```
$ svn mv README.txt README_OLD.txt
```

### – *Copy files and directories*

```
$ svn cp Presentations Presentation_new
```

# Basic Work Cycle (5)

## ➤ Examine your changes

– *svn status: list of changed files*

```
?  arcanum.pdf | File is not managed by svn
M  howto.tex   | File has local content modifications
A  howto.toc   | File is scheduled for addition
M  arcanum.tex | File has properties by not content modification
D  Makefile    | File scheduled for deletion
```

## ➤ Even more details with `-v`

– *Revision numbers*

– *Who made last modification*

## ➤ Status of repository with `-u`

– *Shows changes in repository as well*

## Basic Work Cycle (6)

### ➤ Examine your changes

– *svn diff*: shows your modifications

– *In your local working copy*

```
$ svn diff
```

– *Between a repository revision and your local copy*

```
$ svn diff -r 34 foo.c
```

– *Between two repository revisions*

```
$ svn diff -r 2:5 foo.c
```

### ➤ Revert your changes

```
$ svn revert foo.c
```

# Basic Work Cycle (7)

- Commit your changes

  - `$ svn commit`

- Will open an editor to type in change log

- Alternatively, short logs can be input inline

  - `$ svn commit -m "my short log"`

- *Logs can be retrieved for a file or a tree*

  - `$ svn log foo.c`

# Conflict Resolution

- Look for "C" when you update
- Better than CVS:
  - *Conflict markers are placed in the file to display the overlap (just like CVS).*
  - *Three tmp files are created. these are the original three files that could not be merged.*
  - *SVN will not allow you to commit files with conflicts.*
- Solutions to resolve
  - *Hand-merge the files*
  - *copy one of the tmp files on top of your working copy*
  - *svn revert to toss out your changes*
- Once resolved, you need to tell svn about it  
*\$ svn resolve foo.c*

# File & Directory Properties (1)

- Each file and directory has a list of properties associated with it
- Arbitrary properties & values
- Subversion defines some properties:
  - `svn:ignore`
  - `svn:eol-style`
  - `svn:mime-type`
  - `svn:executable`
  - `svn:keywords`

- Listing properties

```
$ svn proplist README.txt
Properties on 'README.txt':
  svn:mime-type
  svn:eol-style
```

# File & Directory Properties (2)

## ➤ Getting a property value

```
$ svn propget svn:mime-type README.txt
```

## ➤ Setting a property

```
$ svn propset svn:eol-style native README.txt
```

# Dealing with binary files

- Subversion is optimized for dealing with text files (source code, LaTeX documents, etc)
- But, it can deal with binary files
  - *Will not diff nor merge*
  - *Will not change EOL nor apply keywords*
- SVN has a binary detection algorithm, but it sometimes fails (PDF have a text header)
  - *Need to set svn:mime-type property manually to application/octet-stream*

# Repository Organization

- Per-project directories
- Three subdirectories per project:
  - *trunk, tags, branches*
- Trunk is for main development
- Tags is for read-only snapshots
- Branches is a work area

# Working with Branches

## ➤ Create a new branch

```
$ svn cp http://svn.scorec.rpi.edu/svn/TEST/trunk \  
> http://svn.scorec.rpi.edu/svn/TEST/branches/duprec-work  
Committed revision 6  
$ svn co http://svn.scorec.rpi.edu/svn/TEST/branches/duprec-work
```

## ➤ Make Changes...

## ➤ Merge trunk changes into branch

```
$ svn merge -r 6:HEAD \  
> http://svn.scorec.rpi.edu/svn/TEST/trunk .
```

## ➤ Test merged branch

## ➤ Merge branch into trunk

```
$ cd trunk  
$ svn merge -r 6:HEAD \  
> http://svn.scorec.rpi.edu/svn/TEST/branches/duprec-work .
```

# Best Practices

---

- Commit early, commit often
- Commit logical changesets
- Track merges manually
  - *When committing the result of a merge, write a descriptive log*

Merged revisions 3490:4120 of /branches/foobranh to /trunk

- Be patient with large files and repositories
- Know when to create branches
- Trunk should be *stable*

# Subversion at SCOREC

- CVS migration to SVN
  - *Will be done project by project*
- SVN repositories URL:
  - <https://svn.scorec.rpi.edu/svn/SCOREC>
  - <https://svn.scorec.rpi.edu/svn/TSTT>
  - <https://svn.scorec.rpi.edu/svn/TEST>
  - *More can be created*
- Web-based access
  - <https://svn.scorec.rpi.edu/wsvn>

# For More Information

- Subversion project home
  - <http://subversion.tigris.org>
- Subversion online book
  - <http://svnbook.red-bean.com>
- Subversion QuickRef
  - <http://subversion.tigris.org/files/documents/15/177/svn-ref.ps>