CISC 275: Introduction to Software Engineering

Lab 7:
Doing More with

SUBVERSION

Charlie Greenbacker
University of Delaware
Fall 2011

Overview

- Additional collaboration problems
  - Locking vs Merging
- Merging & resolving conflicts
- Recommended repository layout
- More commands
  - Exploring revision history
  - Checking out previous revisions
- Repository web access
- Instructions for Lab Exercise

Additional collaboration problems

- Preventing problems caused by simultaneous edits
  - Lock-Modify-Unlock
    - Pros: no overwrites, no conflicts
    - Cons: forget to unlock, no concurrency = slow
  - Copy-Modify-Merge
    - Pros: work in parallel, concurrency
    - Cons: complicated, have to deal with conflicts
- SVN & CVS mostly follow Copy-Modify-Merge
  - Locking may be needed in special situations

Merging & resolving conflicts

- What if `svn update` reports conflict with foo.c?
- SVN detected changes intersecting local edits
  - 3 new temp files created: foo.mine, foo.r1, foo.r2 (your version, previous version, updated version)
  - Can't commit until you remove them
  - Conflict markers inserted in working copy (foo.c)
- 3 ways to fix: by hand, pick one, or punt with `revert`
  - Once resolution is implemented:

```
> svn resolved foo.c
> svn commit -m "commit message"
```

Merging & resolving conflicts

- Fix conflicts by hand in working copy file (foo.c)
- Examine conflict markers & take proper action

```
> cat sandwich.txt
Slice of bread
Lettuce
Tomato
<<<<<<< .mine
Provolone
Roast Beef
=======
Sauerkraut
Spicy brown mustard
Pastrami
>>>>>>> .r2
Slice of bread
```

- Need to communicate with teammates
  - Use all, one, or subset
  - Might not want both pastrami & roast beef on same sandwich!

Locking files for modification

- Merging is for text-based files, like source code
- What if we need to modify a non-text file?
- Example: editing an image
- Concurrent editing & merging changes = bad idea
  - Instead, use locks to prevent simultaneous work
  - First: `svn lock foo.png -m "message"`
  - Later: `svn commit foo.png -m "message"`
    - `commit` **removes lock, but** `delete` **does not!**
  - Or: `svn unlock foo.png`

Repository layout

- Subversion doesn't force any specific design
- However, “best practices” offer suggestions
- Inside root directory of repository
  - Trunk: main line of development
  - Branches: special features, major tweaks, etc.
  - Tags: “static” revisions for release packages, etc.
- Checkout/modify/commit mostly to/from trunk

Repository layout

- Branches: like a shared sandbox
  - Can try big changes w/ multiple authors without involving everyone or affecting main trunk
  - Or, create version of package for new platform or with a unique extended feature
  - Copy or merge files between trunk & branches
    - Likely merge individual files & specific revision numbers to maintain compatibility
    - Can eventually merge back with trunk, fork into entirely new project, or "crawl into hole"

```
svn copy trunk branches/mac_osx
```

Repository layout

- Tags: “snapshot” of project frozen at certain point
  - Enables packaging of releases
    - Easier than checking out specific revision
    - Remembering “release 1.0” vs “revision 4822”
  - Future commits won't affect tags
  - Note: this is a convention, not enforced by SVN
    - Nothing to stop modification (effectively becomes branch)
    - So, once created, leave tags alone!

```
svn copy trunk tags/release1.0
```

More commands

- Exploring revision history

`svn log [path]`

```
> svn log sandwich.txt
------------------------------------------------------------------------
r3 | sally | Mon, 15 Jul 2002 18:03:46 -0500 | 1 line

Changed type of cheese to provolone.
------------------------------------------------------------------------
r2 | harry | Mon, 15 Jul 2002 17:47:57 -0500 | 2 lines

Added lettuce & tomato.
------------------------------------------------------------------------
r1 | sally | Mon, 15 Jul 2002 17:40:08 -0500 | 4 lines

Initial import
------------------------------------------------------------------------
```

More commands

- Checking out previous revision
  `svn checkout -r 1729`
  - Creates working copy of previous revision for inspection or modification
- Updating to later revision (since last checkout)
  `svn update -r 1729`
  - Updates working copy with changes committed between last checkout and given revision

Repository web access

- Browse files & directories via web browser
  - Quickly view contents of text files
  - Conveniently download individual files
- But, use command-line for non-trivial interaction
- You can use URL for checkout, etc. instead of path

```
> svn checkout http://svn.apache.org/repos/asf/
  subversion/trunk/ svn
```

- Example: http://svn.apache.org/viewvc/subversion/

Lab Exercise (in pairs)

- First step: getting started...
- Both partners will checkout the repo
  `svn co https://shuebox.nss.udel.edu/cisc275/shared`
  - Then cd into local copy of shared dir (`cd shared`)
- One partner will create & add a new file named "[username].txt" with his/her name in the text
  `svn add [username].txt` (after creating file locally)
  `svn commit [username].txt -m "initial check-in"`
- Other partner will grab a copy, edit it by adding his/her name to the list & commit it back
  `svn up [then edit the file locally]`
  `svn commit [username].txt -m "added my name"`

Lab Exercise (in pairs)

- Second step: dealing with conflicts & merges...
- Ensure both partners have latest revision (`svn up`)
- Now, simultaneously edit the same line of code
  - Then, one partner commits & the other updates
  - Update will report conflict, re: intersecting edits
  - File now contains conflict markers like in Slide 5
    - Copy snippet of file showing conflict markers into submission email
  - Resolve as you see fit & commit with message

Lab Exercise (in pairs)

- Email partner names, filename, & snippet showing conflict markers to me by Tuesday, Oct 18
  - I'll inspect your code in the svn repo, making sure each partner made commits w/ good messages
- You're certainly welcome to use the subclipse SVN plug-in for Eclipse to access the repository
  - However, I want everyone to be comfortable using the command-line svn too
  - If you choose to do this, try to install the subclipse plug-in on your own... email me if you run into any problems

Notes on Authentication

- If you receive a validation warning when checking-out for the first time, you should permanently accept the validation
- Depending on where & how you're logged-in, svn may initially try to use your local username to authenticate with the repository
- Might not be same as your udel username
- If svn prompts for a password for this incorrect local username, hit enter & you'll then be asked to specify the correct udel username & password
- svn will cache the correct info for future use

Reminder: 275 Repos

- https://shuebox.nss.udel.edu/cisc275/shared/
- https://shuebox.nss.udel.edu/cisc275/group0/
  ...
  https://shuebox.nss.udel.edu/cisc275/group7/