

CSE 390 Lecture 9

Version control and Subversion (svn)

slides created by Marty Stepp, modified by Jessica Miller and Ruth Anderson
<http://www.cs.washington.edu/390a/>

Working Alone

- Ever done one of the following?
 - Had code that worked, made a bunch of changes and saved it, which broke the code, and now you just want the working version back...
 - Accidentally deleted a critical file, hundreds of lines of code gone...
 - Somehow messed up the structure/contents of your code base, and want to just "undo" the crazy action you just did
 - Hard drive crash!!!! Everything's gone, the day before deadline.
- Possible options:
 - Save as (MyClass-old.java)
 - Ugh. Just ugh. And now a single line change results in duplicating the entire file...
 - RAID to protect your files
 - That's one pricey laptop

Working in teams

- Whose computer stores the "official" copy of the project?
 - Can we store the project files in a neutral "official" location?
- Will we be able to read/write each other's changes?
 - Do we have the right file permissions?
 - Lets just email changed files back and forth! Yay!
- What happens if we both try to edit the same file?
 - Bill just overwrote a file I worked on for 6 hours!
- What happens if we make a mistake and corrupt an important file?
 - Is there a way to keep backups of our project files?
- How do I know what code each teammate is working on?

Solution: Version Control

- **version control system:** Software that tracks and manages changes to a set of files and resources.
- You use version control all the time
 - Built into word processors/spreadsheets/presentation software
 - The magical "undo" button takes you back to "the version before my last action"
 - Wiki's
 - Wiki's are all about version control, managing updates, and allowing rollbacks to previous versions

Software Version control

- Many version control systems are designed and used especially for software engineering projects
 - examples: CVS, **Subversion (SVN)**, Git, Monotone, BitKeeper, Perforce
- helps teams to work together on code projects
 - a shared copy of all code files that all users can access
 - keeps current versions of all files, and backups of past versions
 - can see what files others have modified and view the changes
 - manages conflicts when multiple users modify the same file
- not particular to source code; can be used for papers, photos, etc.
 - but often works best with plain text/code files

Repositories

- **repository:** Central location storing a copy of all files.



- **add:** adding a new file to the repository
- **check out:** downloading a file from the repo to edit it
 - you don't edit files directly in the repo; you edit a local **working copy**
 - once finished, the user checks in a new version of the file
- **commit:** checking in a new version of a file(s) that were checked out
- **revert:** undoing any changes to a file(s) that were checked out
- **update:** downloading the latest versions of all files that have been recently committed by other users

Repository Location

- Can create the repository anywhere
 - Can be on the same computer that you're going to work on, which might be ok for a personal project where you just want rollback protection
- But, usually you want the repository to be robust:
 - On a computer that's up and running 24/7
 - Everyone always has access to the project
 - On a computer that has a redundant file system (ie RAID)
 - No more worries about that hard disk crash wiping away your project!
- Hint: attu satisfies both of these

Subversion

command	description
svnadmin	make administrative changes to an SVN repository
svn	interact with an SVN repository

- **Subversion**: created to repair problems with older CVS system
 - supports directories, better renaming, atomic commits, good branching
 - currently the most popular free open-source version control system
- installing in Ubuntu:


```
$ sudo apt-get install subversion
```
- installing in Fedora:


```
System->Administration->Add/Remove Software
```

 Search for "subversion"



SVN commands

command	description
svn add <i>files</i>	schedule files to be added at next commit
svn ci [<i>files</i>]	commit / check in changed files
svn co <i>repo</i>	check out
svn help [<i>command</i>]	get help info about a particular command
svn import <i>directory repo</i>	adds a directory into repo as a project
svn merge <i>source1 source2</i>	merge changes
svn revert <i>files</i>	restore local copy to repo's version
svn resolve <i>files</i>	resolve merging conflicts
svn update [<i>files</i>]	update local copy to latest version
others: blame, changelist, cleanup, diff, export, ls/mv/rm/mkdir, lock/unlock, log, propset	

Setting up a repo

- on attu, create the overall repository:
 - \$ svnadmin create *repopath*
- from attu, add initial files into the repo (optional):
 - \$ svn import *directory repo*
- give the repo read/write permissions to your project group
 - \$ chgrp -R *myprojectgroup repopath*
 - \$ chmod -R g+rwX,o-rwx *repopath*
- *Exercise*: Create a repository on attu

Adding files to a repo

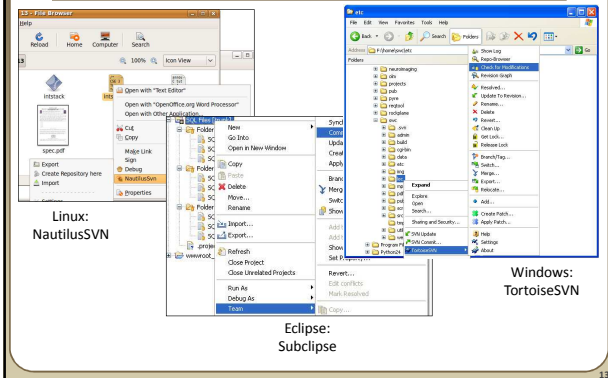
- on your computer, set up a local copy of the repo
 - \$ svn co svn+ssh://attu.cs.washington.edu/*foldername*
 - or, if you're setting up your local copy on attu as well:


```
$ svn co file:///homes/iws/username/foldername
```
 - after checkout, your local copy "remembers" where the repo is
- now copy your own files into the repo's folder and add them:
 - \$ svn add *filename*
 - *common error*: people forget to add files (won't compile for others)
- added files are not really sent to server until commit
 - \$ svn ci *filename* -m "*checkin message*"
 - put source code and resources into repo (no .o files, executables)

Committing changes

- updating (to retrieve any changes others have made):
 - \$ svn update
- examining your changes before commit:
 - \$ svn status
 - \$ svn diff *filename*
 - \$ svn revert *filename*
- committing your changes to the server:
 - \$ svn ci -m "*added 0(1) sorting feature*"
 - Version control tip: use good commit messages!
- *Exercise*: check out the repository, add some files, and commit them

Shell/IDE integration

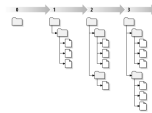


TortoiseSVN

- Available at <http://tortoisesvn.net/>
- Nice graphical interface for windows users
- To use on a repository located on attu:
 - Need to use the svn+ssh syntax:
 - `svn+ssh://username@attu.cs.washington.edu/repopath`
- *Exercise:* Check out our repository, modify a file, add a file, and commit our changes

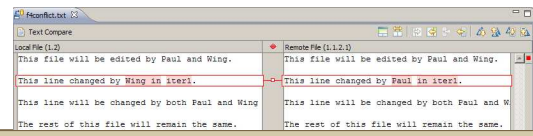
What's actually going on?

- Take a look inside the svn project folder...
 - Where the heck are our committed files?
 - Take a look at the readme...
- Everything is stored in SVN's database structure
 - So, even though you might have 100 versions of a file, there's not 100 copies of that file
 - Database stores the diff from version to version
 - Helps more efficiently store a large codebase across hundreds of versions
 - Don't worry about the details. Just don't mess with the repository directly!



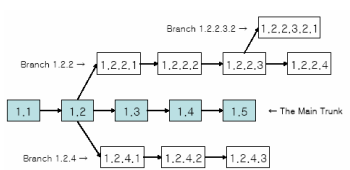
Merging and conflicts

- **merge:** Two sets of changes applied at same time to same files
 - happens when two users check out same file(s), both change it, and:
 - both commit, or
 - one changes it and commits; the other changes it and does an *update*
- **conflict:** when the system is unable to reconcile merged changes
 - **resolve:** user intervention to repair a conflict. Possible ways:
 - combining the changes manually in some way
 - selecting one change in favor of the other
 - reverting both changes (less likely)



Branches

- **branch (fork):** A second copy of the files in a repository
 - the two copies may be developed in different ways independently
 - given its own version number in the version control system
 - eventually be merged
- **trunk (mainline, baseline):** the main code copy, not part of any fork



A Day in the Life of SVN

- At the beginning of the day/work session, update working copy
 - `svn update`
- Make changes
 - `svn add`, `svn delete`, `svn copy`, `svn move`
- Review changes
 - `svn status`, `svn diff`
- Fix mistakes
 - may need to start from scratch: `svn revert`
- Get ready to commit changes
 - `svn update`, `svn resolve`
- Commit changes
 - `svn commit`
- Repeat many, many times
 - best practice: commit as soon as changes make a logical unit; commit often

Learn what you need

- Creating branches and using merge tools are usually more than you need for any curriculum projects
 - Conflict resolution tools can be confusing
 - May be easier to back up my conflicted file, update so I now have the current version, then manually merge my changes with the updated files
 - You probably won't have a good reason to create a branch in a department project
- But, they are definitely used in industry, and you should at least know about them

19

Another view: Git

- Git is another popular version control system.
- Main difference:
 - SVN:
 - central repository approach – the main repository is the only “true” source, only the main repository has the complete file history
 - Users check out local copies of the current version
 - Git:
 - Distributed repository approach – every checkout of the repository is a full fledged repository, complete with history
 - Greater redundancy and speed
 - Branching and merging repositories is more heavily used as a result
- Takeaway: There are differences beyond just differently named commands, learn about a tool before using it on a critical project!

20

Wrap-up

- You ***will*** use version control software when working on projects, both here and in industry
 - Rather foolish not to
 - Advice: just set up a repository, even for small projects, it will save you time and hassle
- Lots of online options for free open source code hosting
 - Google code, Git hub, JavaForge, SourceForge...
 - All use version control to manage the code database
- Any experiences with version control, positive/negative?

21