Source Control

Comp-206 : Introduction to Software Systems
Lecture 21

Alexandre Denault
Computer Science
McGill University
Fall 2006

Source Revision / Control

- Source Control is about the management of revision.
  - Through its development, many components of an application experiences changes.
  - Changes to these components are noted with a revision number, kind of like with paper documents.
- Source Control allows you to collect these revisions and compare them.
  - For tracking/auditing purposes.
  - For debugging purposes.

Central Code Location

- The code is located in one central location (i.e. one server) called a code repository.
- Each developer acquires his copy of the code for his machine.
  - He does all the development locally, on his machine.
- When he wants to confirm a change and propagate this change to all the other developers, he commits his code to the repository.
  - He can update his local copy with changes other people commit to the central location.
- Each committed change is assigned a revision number.

Team Tool

- Source Control has also become an invaluable tool for team work in software development.
- It allows large groups of developers to work on the same project, and minimizes the risks of overlapping changes.
- Each developer can work on his local copy, without affecting other developers.
- Once he is sure his changes are stable, he simply commits it to the repository.

CVS

- CVS is the Concurrent Versions System, was created in the mid 1980's.
- It was recreated as a follow up to an earlier versioning system called Revision Control System (RCS).
  - RCS was great for individual files, bad for large projects.
- Although very popular in the 1990's, CVS had severe limitations.
  - You cannot move or rename files in CVS. You have to delete them and re-add them.
  - CVS has difficulty properly retaining permissions.
  - Directories are not versioned.

SVN

- Subversion (a.k.a. SVN) was developed as a modern day replacement to CVS.
    - Many of the developers working on CVS work on SVN.
- Subversion has many key features:
    - Commits are truly atomic (can't have problem with 2 people committing at the same time).
    - You can now move or rename files.
    - Directory are versioned.
    - Strong integration with Apache.
    - Python, Ruby, Perl, and Java language bindings.
    - Branching and tagging are faster.

Trunk / Branch / Tags

- Source Control Systems are usually separated into modules.
- The modules are further separated into three categories: the trunk, branches and tags (tree analogy).
  - The trunk is the main copy of your code.
  - Branches are separate copies of your main code.
  - Tags are snapshots of the trunk or branches.

Why use branches?

- On a project, most people work on the trunk.
- If a large change needs to be implemented and it might affect other people, then a branch is create for them.
  - Developers working on the special change work on the branch.
  - Other developers continue working on the trunk.
- When the large change is completed, the branch can be merge backed with the trunk.
  - Merging a branch back is a very difficult operation, especially if a lot of development has been done in the trunk.
- With this strategy, main developers are not affected with the big change.

Why use tags?

- As previously mentioned, tags are like snapshots for the trunk or branches.
- When developing a large application, companies will often release both major and minor releases.
  - Major release : Eclipse 3.0!
  - Minor release : Eclipse 3.2
  - Bugfix release : Eclipse 3.2.2
- Before you release software, you usually tag the branch with the version number.
  - Thus, you associate the version number with the revision number at that time.
- This allows you to do 2 things latter:
  - Find which file revision where used for that release.
  - Checkout a copy of the branch using those previous revision.

Another Strategy

- The trunk is used for the main development of the application.
- Before major release, you create a branch.
  - People working on that release continue working on that branch, making it more stable.
  - They are not allowed to add new features.
  - Regular developers continue working on the trunk.
- Before releasing, you tag that branch with the minor version number.
- Development on the branch is continued as long a the major release is supported..
  - People can go back to the branch if a minor version is ever needed.

Creating a repository

- To create a repository, you simply need to use the svnadmin command.

  ```
  svnadmin create  /home/bob/subversion
  ```

  - This would create an svn directory in /home/bob/subversion
- The next step would be to set up a trunk/branch/tag structure.
  - Unless you are working on a large project with regular releases, you don't need a trunk/branches/tags setup
- Note: You cannot create an SVN repository on your CS account. However, if you need one, the Socs Help people will be happy to give you special space to do so.

svn

- The svn command is an all purposes tool. It contains all the necessary functionality to
  - checkout code
  - update a repository
  - merge two revisions
  - commit code
  - Etc.

svn help

- You type in the *svn help* command to see

```
usage: svn <subcommand> [options] [args]
Subversion command-line client, version 1.2.3.
Type 'svn help <subcommand>' for help on a specific subcommand.

Most subcommands take file and/or directory arguments, recursing
on the directories.  If no arguments are supplied to such a
command, it recurses on the current directory (inclusive) by
   default.

Available subcommands:
   add
   blame (praise, annotate, ann)
   cat
   checkout (co)
   cleanup
   commit (ci)
   copy (cp)
...
```

URL of repository

- To use a repository, you need to knows it's location.
- In subversion, the location of the repository is known as the URL.
- The URL depends on which access method you want to use to contact the repository.
- For example, if you are using the same machine that the repository is located, you can use a *file* URL

  file:///home/bob/subversion
- Alternatively, you can tunnel through SSH to reach the repository.

  svn+ssh://username@server/home/bob/subversion
- Some repositories can be accessed through the web using apache.

  http://server/home/bob/subversion

Checking Out

```
svn checkout URL [PATH]
```

- The first step in using an SVN directory is checking out the code. This can be done using the svn checkout command.

```
svn checkout
  svn+ssh://adenau@svn.cs.mcgill.ca/xtra/cs206/t
  runk cs206-trunk
```

- This will checkout the main branch (trunk) of cs206 in the cs206-trunk directory.
- You can use the -r option to checkout a specific revision.

Adding

```
svn add FILES
```

- To add a file to a repository, you need to first place it in your checkout directory (in the correct location).
- Then call the svn add command.
- The fill will be added next time you commit your changes.

Status

```
svn status [PATH]
```

- For a given path, svn status will give the svn state of each file.
  - 'A' Added
  - 'C' Conflicted
  - 'D' Deleted
  - 'G' Merged
  - 'I' Ignored
  - 'M' Modified
  - 'R' Replaced
  - '?' item is not under version control
  - '!' item is missing
- More information about the output can be found by using svn help status.

Committing

```
svn commit [PATH]
```

- Once you've tested your changes, you can commit them to the repository.
- When committing, you will be asked to supply a short message.
- This short message should explain what you are committing:
  - Changes you did
  - Reasons for the change
  - Bugs you fixed (including bug id if available)

Updating

```
svn update [PATH]
```

- Other people are continuously contributing to the svn repository.
- To update your code with their latest changes, just use the svn update command.
- If somebody changed lines in a file that you also changed, a conflict occurs.
  - The file is going to be tagged as in a conflicted state.
  - Before you can commit your changes, you need to resolve the conflict.

Deleting

```
svn delete [FILES]
```

- This command will delete a file from the repository.
- Note that the file is only delete from the current revision.
    - The file will still exist in past revisions.

Conflict

- When a conflicted file is found, is it modified as so:

```
<<<<<<< .mine
if ( (i > 0) && (j > 0) ) {
   j++
>>>>>>>>>> .r314
if ( (i > 0) && (h < 0) ) {
   h--
>>>>>>>>>>
```

- By comparing the two code, you must merge them and resolve the conflict.
- In addition, two additional files will be created, one with a .r314 extension and one with a .mine extension.

Resolving

```
svn resolved FILE
```

- Once both piece of code have been merge, the svn resolve command must be used to indicate the new state of the file.

```
if ( (i > 0) && (j > 0) && (h < 0) ) {
   j++
   h--
```

File Locking

- To avoid conflict, some source control scheme offer locking mechanisms
  - Before working a file, you must acquire a lock on a file.
  - Only one lock may be granted per file.
  - After committing your changes, you must release your lock.
- Although no conflict occur, file locking slows down development, especially on popular files.

Conflict Avoidances

- To minimize the risk of conflicts, some companies have established "manual" locking scheme.
- One of the most memorable is the stuffed toy locking system.
  - Only the person with the stuffed toy on his desk can commit his code to repository.
  - A programmer can "acquire" the toy by getting it from its designated storage.
  - Once he is finished committing his code, he must return the toy to its designated storage.
- Although this solution solves some problems of simultaneous commits, it
  - shares a lot of problems with file locking.
  - does not prevent conflicts from occurring, just reduces the chances.

SourceSafe

- SourceSafe is the version control package solution from Microsoft, distributed with Visual Studio.
- It uses a purely file locking mechanism.
- SourceSafe provides tight integration with the Visual Studio tools.
- However, no clients for MacOS X or Unix exist.
- SourceSafe works well for small teams (5 or less), but does not scale well.

Perforce

- Perforce is the industry solution for revision control.
- It has an impressive client list
  - Activision, ATI, Cisco, EA, Ericsons, IBM, SCEA, etc
- Perforce supports several operating system and can integrate itself with several application.
  - Visual Studio / Eclipse / Xcode
  - Photoshop
  - 3DS Max, Maya
  - MS Office