CS2720 Practical Software Development

Subversion Tutorial Spring 2011

Instructor: Rex Forsyth
Office: C-558
E-mail: forsyth@cs.uleth.ca
Tel: 329-2496

Tutorial Web Page: http://www.cs.uleth.ca/~forsyth/cs2720/lab/lab.html

Version Control

A typical client/server system consists of a repository on the server where data is maintained.

- clients may read data from the server or write data to the server.
- the server remembers every change made to data in the repository
- typically reads will only see the latest version
- possible to view earlier versions
- possible to find historical information
  - who made the last change
  - what changes were made on a specified date
  - what differences are there between versions

File Sharing

We want to be able to share data without stepping on each others toes.

eg Consider a file X that has an A in it

1. Joe edits X and changes the A to a B
2. Jill edits X and changes the A to a C
3. Joe saves X
4. Jill saves X

What is in the file?

Possible Solutions

1. Lock–Modify–Unlock
   - Joe locks the data and proceeds to edit it.
   - When finished he saves the data and then unlocks it.
   - While he is working on it, no one else can access it.

   Problems
   - someone forgets to unlock the data
   - waste of time and resources since different people may be working on different parts of the data.
   - false sense of security

     Joe locks A and modifies it
     Jill locks B and modifies it
     A depends on B

2. Copy–Modify–Merge
   - Joe checks out a copy of the data and begins editing.
   - Jill checks out a copy of the data and begins editing.
   - Jill saves her changes and writes to the repository
   - Joe saves his changes and attempts to write to the repository
   - The server informs Joe that his copy is out of date
   - Joe updates from the repository and tries to merge his changes
   - If merge can not be done automatically, Joe and Jill get together and, with the help of the server, resolve conflicts.
   - The merged version is written to the repository.

   The Lock–Modify–Unlock model is still required to be available. Why?

Subversion implements the Copy–Modify–Merge model and makes locking available.

We have a Subversion server

Its name is **https://svnhost**

Each group will have their own repository on the server. Only group members will be able to read data from the server and write data to the server.

All subversion commands are preceded by **svn**

To get the current data from the server, use the **checkout** command

- Syntax

  ```
  svn checkout serverName/repositoryName/pathname [workingCopy]
  ```

- whatever is in *pathname* will be copied into *workingCopy*
- If *workingCopy* is omitted, data will be copied into *pathname*
- Easiest method – change to the desired working copy directory and use `.` as the workingCopy name **eg**

  ```
  svn checkout https://svnhost/rexs/example1 .
  ```

- If there is nothing in the repository then, of course, there is no *pathname*, so to start, just use

  ```
  svn checkout https://svnhost/rexs .
  ```

- The working copy contains all the data in the *pathname* as well as a special **.svn** directory which is used by the server.

To place an existing project into the repository, use the **import** command.

- Any time you are writing data to the repository, you must include a message indicating what is being written.
- This is done by using the **-m** option followed by the desired message in quotations.
- If you forget to include the **-m** option, the server will pop open your prefered editor. You then type in the message. When you save and exit the editor, the contents will become the message.
- Suppose you have an existing project in the directory **XXXX** which you want to place into the repository. Use the following command :

  ```
  svn import XXXX https://svnhost/rexs/projName -m "my message"
  ```

- If you leave off *projName* then all files in **XXXX** will just be placed in the top level of the repository.

Once a member of the group has imported the project into the repository, then other group members can check it out. In fact, in order to get an **subversion** working copy, even the one who imported it, must check it out.

If you enter the command `ls -al` in this working copy directory, you will see the extra directory **.svn**

This directory holds extra info that subversion needs to keep track of changes made by you and others. It represents a snapshot of the repository at the time you checked out this working copy.

You may now make changes to the files in your working copy. Since subversion has the **.svn** directory, it will know which files you have changed and also if files in the repository have changed.

Changes you make in your working copy will not change the repository.

To submit the changes you have made in your working copy, to the repository, use the **commit** command.

- Since this is an operation that is writing data to the repository, you must use the **-m** option and include a message.

  ```
  svn commit filename -m "message"
  ```

- If your changes do not conflict with any new changes in the repository, then they are written to the repository and the **.svn** is updated.
- Now if someone else checks out the data from the repository, they will get your committed changes.
- If no filename is given, eg `svn commit -m "message"`, then all changes in the current directory will be committed
- Each time anyone does a commit, the version number changes for those files which have been modified.

To get the latest version of changes in the repository into your working copy, use the **update** command. eg

```
svn update [pathname]
```

For each updated item, a line will start with a character reporting the action taken. These characters have the following meaning:

- A – Added
- D – Deleted
- U – Updated
- C – Conflict
- G – Merged

Starting from scratch:

1. `svn checkout https://svnhost/repositoryName .`
2. create directories and files
3. mark these for addition
4. `svn commit -m "some informative message"`

Subversion uses information in the **.svn** directory to determine the current state of any file. Each file in a working copy, is in one of 4 possible states:

1. unchanged and current
   - the file matches the data in the **.svn** directory and the file in the **.svn** directory matches that in the repository.
   - committing this file will do nothing
   - an update will not affect this file
2. locally changed but current
   - the file does not match the data in the **.svn** directory but the data in the **.svn** directory does match that in the repository.
   - committing this file will publish the changed file to the repository
   - an update will not affect this file

3. unchanged but out of date
   - the file matches the data in the **.svn** directory but the data in the **.svn** directory does match that in the repository.
   - committing this file will do nothing
   - an update will copy the file from the repository into your working copy.
4. locally changed and out of date
   - the file does not match the data in the **.svn** directory and the data in the **.svn** directory does not match that in the repository.
   - committing this file will fail with an *out of date* error
   - an update will attempt to merge the file in the repository with your file
     – if merging can not be done automatically, the file is put in a **conflict** state and the users have to resolve the conflict.

To determine the status of files in your working copy, use the **status** command. eg

```
svn status [pathname]
```

- With no options, the status command only reports local changes, ie differences between your files and those in the **.svn** directory
- **-**u – reports out of date items, ie those items that are different in the repository and the **.svn** directory
- **-**v – reports full revision history on every item
- symbols have the following meanings:
  - ? – not under version control
  - A – marked for addition
  - C – in conflict
  - D – marked for deletion
  - M – file has local modifications
  - * – file is out of date

Note: **commit** and **update** commands are independent of each other.
You may commit without getting others changes.
You may get others changes without committing your own.

You may delete your working copy at any time. Use `rm -rf` to avoid confirming every file and directory.

To obtain a *clean* copy, without all the **.svn** directories, use the **export** command. eg

```
svn export https://svnhost/repositoryName .
```

More svn commands

- help [command]
- add – mark a file to be added
- delete – mark a file to be removed
- copy – copy a file and mark to be added
- move – copy file to new name and mark it for addition; mark original for removal
- mkdir – make a directory and mark it to be added
- diff – report the difference between two revisions
- revert – undo (most) local changes
- resolved filename – indicate resolution of conflicts by removing conficting files.
- list – list files in the repository
- log – display the subversion log file
- cleanup – recursively clean up the working copy, removing locks, resuming unfinished operations, etc.

Many of these commands have alternate versions eg *ls* instead of *list*

Many commands allow access directly to the repository by specifying a **URL** instead of a local pathname. Remember that if writing to the repository, you **MUST** use the **-m** option.

For a list of all the commands and alternates, type `svn help`

Typical subversion working cycle

1. **update** your working copy, thus obtaining all other committed changes
2. Make your changes
3. Examine status of files using the **status** command
4. Undo changes if required
5. Publish your changes using the **commit** command
6. If files are out of date, them try to merge using the **update** command
7. If files are in conflict
   (a) Get together with other group members and decide how to resolve conflicts
   (b) Make the changes
   (c) Indicate resolution using the **resolved** command
   (d) **commit** the resolved changes