Introduction to Revision Control

Ulrik Nyman

November 13th 2008

Last updated: November 14, 2008

Todays Agenda

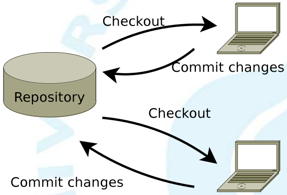
- Revision Control
  - What is it good for?
  - What is it?
- Exercises
  - I will show the basics
  - Fetch slides and work from them
- Will probably not take four hours

Why should I use revision control

- Provides a structured way for working together on a set of files
  - Also beneficial when working alone
- No lost changes
- Provides history of changes
  - To see what went wrong

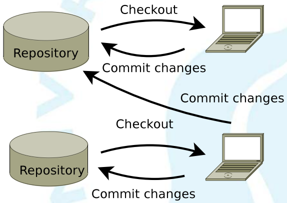
Revision Control 101

- Central concept is a repository
  - A database which stores the current content and all changes made to the content.
  - Old revisions can be brought out and differences viewed.
- Users "checkout" copies of the repository and commit back changes.
- Users must update their copy of the repository.



Distributed Revision Control

- Everybody
  has their own repository
- Users "checkout"
  copies of the repository
  and commit back changes.
- Changes can be moved
  between different repositories



Revision Control Systems

- Central Systems

cvs	The old system that many projects still use
Subversion (svn)	The most popular replacement for **cvs**
...	

- Distributed Systems

Bazaar (bzr)	I use this one
git	Linux Kernel, Ruby on Rails, WINE or X.org
Mercurial	
...	

Subversion

- An open source revision control system
  - Has mostly replaced CVS
- Works quite well
- Well documented
  - Has free online book (also available in paper version)
  - Available at: `http://svnbook.red-bean.com/`
  - Very well written
- Tutorial on `support.cs.aau.dk`
  - Build in help: `svn help` and `svn help` *`command`*
- Uses a centralized repository
  - Easier to grasp than distributed version control

Tortoise SVN

- Graphical client for Windows
- Can be used instead of the command line tool
- Can be downloaded from:

  `http://tortoisesvn.tigris.org/`

Exercises

- First: Have a terminal on the server available
    - This is needed to create the repository
- The command line tool is used in the exercises
    - A graphical client can of course also be used

Creating a Repository

- Creating a repository using the `svnadmin` tool.
- Do the following in a terminal:
  - `svnadmin create ~/SVN_test`
  - This creates a repository in a directory named `SVN_test` located in your home directory
- The URL to your repository is:
  - `svn+ssh://marge.cs.aau.dk/user/`*`username`*`/SVN_test`
- The directory should be considered a black box
  - Only in emergencies should you change anything manually in this directory.

Checking out a copy

- Before working, a copy of the repository must be created.
- This is done with the following command (one line):

```
svn checkout
svn+ssh://homer.cs.aau.dk/user/username/SVN_test
svncheckout
```

  - The copy will be in the directory `svncheckout` .
- Remember: You are working on a copy.
  - Changes will only be visible to others after you commit.

Putting data in the Repository

- Subversion only tracks files which you ask it to track.
- To add a file to the repository:
  - `svn add` *`filename`*
- The file is only added to your local copy. To update the repository, you must commit the changes:
  - `svn commit`
  - The commit may fail if a conflict is detected.
- Note: It is also possible to import entire directory trees directly into the repository using the `svn import` command.

Viewing changes

- Sometimes it is nice to see what has been changed.
  - `svn status` provides a high level view of changes.
  - Note: Only local changes are displayed.
- File diffs can be seen with:
  - `svn diff`
- If you are happy with the changes, you can commit.
- If not, you can revert changes:
  - `svn revert` *`filename`*
  - This will roll back the file to your latest checkout
  - Careful! This can delete your work

Moving, Deleting, and Copying

- To rename a file: `svn move` *`oldname newname`*
- To delete a file: `svn delete` *`filename`*
  - Note: History is still kept for the deleted file, so it can be restored later.
- To copy a file: `svn copy` *`filename filecopy`*
  - Copying may seem strange, but it is useful when splitting a file, as history will exist for both files, and the split itself is explicit.

Getting Changes

- Other people often change the content of the repository
- To get these changes, you must update your copy:
  - `svn update`
  - This will update your copy to the latest revision
- Creating a repository, adding files, committing changes and receiving changes are the most common and necessary operations.
  - Most persons only use these and get along fine.

Conflicts

- When updating, a merge conflict can occur
- Conflicts happen when Subversion cannot merge a change you have made to a file and a change that has been made to the same file in the repository.
- Conflicts are resolved manually by editing the file
  - After fixing the conflict, mark it resolved with `svn resolved` *`filename`*
- Conflicts can be prevented by having people work on different files.
- Avoid huge commits, they increase the chance of conflicts and are harder to merge.
- With a bit of thought, conflicts rarely occurs.

Working with revisions

- Subversion allows you to explore the history of a repository.
- Checking out an old revision:
  - `svn checkout` *repository_url* `-r` *revision*
  - Also possible to specify dates. See the build-in help or the book.
- Viewing changes between different versions:
  - `svn diff -r:REV1:REV2`
  - This will show the differences between two revisions
- You can specify filename to see changes made on a certain file (also works on subtrees).

Trunk, Tags, and Branches

- A Subversion repository is usually split into a trunk, tags and branches.
- Trunk is current development (or mainline).
- Tags are “marked” revisions e.g., releases, or known good versions.
- Branches are for development happening in parallel with trunk, but does not disturb trunk.
- Tags and branches are often not necessary in smaller projects, but can still be useful.
- The free Subversion book explains tags and branches very well.

Setting up the Repository - Again

- When using trunk, tags and branches, the repository must be setup in a slightly different way.
  1. Create a repository using `svnadmin` (as before)
  2. Checkout the repository and add the directories `trunk`, `tags`, `branches`, and commit.
  3. Re-checkout the trunk directory and work with that as main development.
- Tags and branches are explained in the following slides.

Tags

- In Subversion tagging is implemented using copying
  - Other revision control systems use other mechanisms
- A tag can be done like this (one command):

```
svn copy svn+ssh://homer.cs.aau.dk/RepoPath/trunk
svn+ssh://homer.cs.aau.dk/RepoPath/tags/MyTag -m
"Tagging trunk to MyTag"
```

- Then the current trunk will be available at `tags/MyTag` even after changes have been made.
- Note: The repository does not make a complete copy, as this is not needed. Tagging is very cheap.

Branches

- A branch is basically a copy of trunk, in which development can be done without disturbing trunk.
    - Good for longer development tasks such as replacing an entire subsystem or similar.
- Working with branches is too complicated to be covered in a few slides.
    - Check out the Subversion book, which covers the topic in a very understandable way.

A final note on working with revision control

- Avoid huge commits
  - Difficult to see what has actually changed
  - They make merging difficult
- Many editors also integrate with Subversion

- Yes, there is a learning curve, and you might have to change the way you work.
- But it is worth it.
- All serious software projects use it for tracking code.
- You should too (and for LaTeXas well).