CSC 195.01 2013S *Technologies for Mediascripting*

Class 02: Source Code Management Systems

Held: Thursday, 31 January 2013

Summary: In today's class, we do some basic exploration of source code management (SCM) systems.

Related Pages:

- EBoard.

Notes:

- Don't forget this Thursday's CS extras talk on summer research opportunities in CS at Grinnell.
- Please see the homework listed at the end of today's outline.

Overview:

- Review: Why we need source code management systems
- Contrasting philosophies of SCM
- Common tasks with source code management systems
- Behaving well
- Subversion and the Glimmer subversion repository
- Git
- Homework

Review: Why we need source code management systems

I've discussed this quickly once, but thought it would be useful to review.

- Large projects involve lots of files.
- Large projects involve lots of people.
- Large projects involve lots of updates.
- What are some implications of these issues?
  - At any one time, different modifications are likely to be in different states (just started; somewhere along the way; perhaps working but not completely; fully tested; etc.)
  - No one person really "owns" any piece of code - what you write today may be modified by someone tomorrow.
  - Changes you make to code someone else writes may have unintended consequences.
  - Different programmers may be in different places at different times; the only way to communicate may be through the code (rather than meetings or whatever).
  - ...
- Source code management systems are designed to help address these and other issues of collaborative software development.
  - But much of what they do for collaboration they also help with on individual projects.

- Main features of SCMs
  - A way to “check out” parts of a project to work on them
  - A way to “check in” the parts of your project after you’ve made the changes
  - A way to access all past versions of any file, so that you can retrieve code you deleted (or see what changes you made)
  - A way to help programmers deal with (well, at least identify) conflicting changes.
  - Encouragement to document changes and capabilities.

Contrasting philosophies of SCM

- While I have been referring to SCMs as a single type of entity, different source code management systems have very different approaches to dealing with the common issues.
- Early SCMs (at least the ones I used - rcs and sccs) tended to emphasize much more of a library model - the repository is a library; only one person can have any particular piece of the library checked out.
- Some more modern SCMs, such as the way I use CVS and Subversion, still emphasize the single repository model, although they do permit more than one person to work on the same file.
- More recently, git (and other systems) use more of what I’d call a parallel repository model. Rather than thinking of each programmer contributing back to the central repository, we think of each programmer as having her own copy of the project.
  - There is still one centralized copy of the project,
  - You can pull changes from the central copy.
  - You can request that the owner pull your changes back into the central repository.

Common tasks with source code management systems

Rare but important tasks

- Create a new repository / Upload an existing codebase into a new repository
- Grab a repository so that you can work on it on your computer.
  - In the distributed philosophy, this may involve making a copy of the repository first.
- Retrieve an older version of a file
- Rollback to an earlier version of the whole repository (useful for undoing multiple changes)

Typical workflow.

- Grab the changes other people made. (Typically done at the start of each session).
- Do your work.
- Note what changes you’ve made.
  - See what files have changed
  - See what details of those files have changed
- Commit your changes back to the repository (either your copy or the central) copy.
  - This typically involves documenting your changes.

Related issues

- Add files to the repository
- Remove files from the repository

Behaving well in a collaborative project

- **Never put broken code into the repository!**
  - Arguably, you should thoroughly test any code you put into the repository. (How thoroughly you test depends on the status of the project.)
  - Errors in your code should not affect the ability of your colleagues to work with the project.
- Take documentation seriously. Your documentation (both the notes you log and the notes that are there) may be the only way you communicate with the other people who work on your project.

Subversion and the Glimmer subversion repository

- For many years, we've been using Subversion (svn) as the primary repository for Glimmer projects.
  - We are moving many projects to git.
  - I'm still most comfortable with subversion - I know the model better and I know the commands better.
- I'll do a simple demo.
- List all the files in a repository
  `svn ls svn://svn.cs.grinnell.edu/glimmer`
- Make a copy of a repository (or subrepository)
  `svn co svn://svn.cs.grinnell.edu/glimmer/sandbox`
- Enter the copy
  `cd sandbox`
- See what's happened in the repository
  `svn log | less`
- Compare an old version of a file to the current version
  `svn diff -r 4615 test.c`
- [Play around]
- Determine what files I've changed.
  `svn status`
- Determine precise differences
  `svn diff test.c`
- Note that I've added a file
  `svn add newfile.c`
- Propagate changes back to the central repository
  `svn commit`
- Grab the changes someone else made `svn update`

Git

- These days, every year or two a new SCM arrives.
- git is a currently popular SCM system
  - It's also likely to live on for awhile, since I believe it's used for the main Linux development stream
- As I noted earlier, git takes a slightly different philosophy on source code management than I'm used to.
- Nonetheless, we are going to do our best to use git for future Glimmer projects.
- An important aspect of git is github, a popular hosting site for git projects.
  - github adds some social media aspects to SCM.
  - My sense is that potential employers now use github (or related sites) a lot when making hires.
  - You should have an account on github and be productively active.

Homework

Note: I would like you to do these tasks using command-line git, at least the first time you do them.

- Set up an account on github.
- Set up a project under your account. The project can be almost anything - code from 151 or 161, notes from a class, whatever you think is interesting and you're willing to make public.
- Make a change and incorporate it into your project.

The next steps require collaboration with one or more classmates.

- Fork one of your classmate's projects and make a copy on your computer.
- Make some change to the project.
- Send a pull request to the classmate.
- When you receive a pull request, incorporate the code from the request.
