Source code management systems

- **SVN, Git, Mercurial, Bazaar, ...**
- **for managing large projects with multiple people**
  - work locally or across a network
- **store and retrieve all versions of all directories and files in a project**
  - source code, documentation, tests, binaries, ...
- **support multiple concurrent users**
  - independent editing of files
  - merged into single version
- **highly recommended for COS 333 projects!**
  - save all previous versions of all files so you can back out of a bad change
  - log changes to files so you can see who changed what and why
  - mediate conflicting changes made by different users to maintain consistency

Basic sequence for SVN

- **create a repository**
  - where SVN stores its copies of your files
  - including all changes made by anyone
- **each person checks out a copy of the files**
  - "copy - modify - merge"
  - get files from repository to work on
    does not lock the repository
  - make changes in a local copy
  - when satisfied, check in (== commit) changes
- **if my changes don't conflict with your changes**
  - SVN updates its copies with the revised versions
  - automatically merges edits on different lines
  - keeps previous copies
- **if my changes conflict with your changes**
  - e.g., we both changed lines in the same part of file,
    SVN doesn't permit the checkin
  - we have to resolve the conflict manually

Basic sequence, continued

- **when changes are committed, SVN insists on a log message**
  - strong encouragement to record what change was made and why
  - can get a history of changes to one or more files
  - can run diff to see how versions of a file differ

- **can create multiple branches of a project**

- **can tag snapshots for, e.g., releases**

- **can be used as client-server over a network, so can do distributed development**
  - repository on one machine
  - users and their local copies can be anywhere

Getting started

- **to put code under SVN control, do this once:**

```
svnadmin create repository
[mkdir proj.dir & put files in it, or use existing directory ]
svn import proj.dir file:///repository -m 'initial repository'
svn checkout file:///repository working.dir
```

- **create, edit files in working.directory**

```
cd working.dir
ed x.c    # etc.
svn diff x.c
svn add newfile.c
```

- **update the repository from the working directory**

```
svn commit  # commit all the changes
```

- **for more info, read svn.help on web page, SVN book, etc.**

Alternatives

- **Git**

  http://git-scm.com/

- **Bazaar**

  http://bazaar-vcs.org

- **Mercurial**

  http://www.selenic.com/mercurial

- **comparison page**

  http://www.infoq.com/articles/dvcs-guide

Git

- **originally written by Linus Torvalds, 2005**
- **distributed**
  - no central server: every working directory is a complete repository
  - has complete history and revision tracking capabilities
- **originally for maintaining Linux kernel**
  - lots of patches
  - many contributors
  - very distributed
  - dispute with BitKeeper (commercial system)
  - dissatisfaction with CVS / SVN

Basic Git sequences (git-scm.com/documentation, gitref.org)

```
cd project
git init
```
makes .git repository
```
git add .
git commit
```
makes a snapshot of current state

[modify files]
```
git add …  [for new ones]
git rm …   [for dead ones]
git commit
git log --stat –summary
git clone [url]
```
makes a copy of a repository

CAS: Centralized Authentication Service

- **if your project requires users to log in with a Princeton netid**
  don't ask users to send you their passwords at all,
  and especially not in the clear

- **OIT provides a central authentication service**
  - the user visits your startup page
  - the user is asked to authenticate via OIT's service
  - the name and password are sent to an OIT site for validation
    (without passing through your code at all)
  - if OIT authenticates the user, your code is called

- **OIT web page about CAS:**

  `https://sp.princeton.edu/oit/sdp/CAS/`
  `Wiki%20Pages/Home.aspx`

- **sample code:**

  `www.cs.princeton.edu/~bwk/public_html/CAS`

Behind the scenes in the client libraries

- **your web page sends user to**

  `https://fed.princeton.edu/cas/login?`

  `service=`*`url-where-user-will-log-in`*

- ***CAS* sends user back to the service url to log in**

  with a parameter `ticket=`*`hash-of-something`*

- **your login code sends this back to**

  `https://fed.princeton.edu/cas/validate?`

  `ticket=`*`hash`*`&service=`*`url…log-in`*

- **result from this is either 1 line with "`no`"**

  or two lines with "`yes`" and *`netid`*