Introduction to Version Control

Lecture 1
Revision (or version) control

What is it?

- A method for tracking changes to files
- A way to work collaboratively
- A way to maintain a centralized or distributed shared copy of projects
Why is it useful?

**Individuals:**
- Backups
- Incremental versions (can *always revert*)
- Point-in-time marking (a.k.a. tagging)
- Branching (e.g., development and release versions)
- Complete change *history*

**Teams:**
- Same as individuals, plus:
- Allow multiple developers to work on the same set of files
- Merge changes, handle collisions (conflicts)
- Accurately assign blame
Types of version control systems

- Centralized
- Decentralized (distributed)
Centralized version control

Recipe +eggs

Recipe +flour

Recipe +sugar

Main Trunk

r1

Sue

r2

Joe

r3

Sam

r4

Recipe eggs flour sugar
Examples of centralized VCS

- CVS -- Concurrent Version System
  - http://www.nognu.org/cvs/

- SVN -- Subversion
  - http://subversion.tigris.org/
Distributed version control

Recipe
+eggs
+flour
+sugar

Branch
+eggs
+flour
+sugar

Sue
+eggs

Jill
+eggs
+flour

Sam
+flour
+sugar

Recipe
eggs
flour
sugar
Examples of distributed VCS

- Git
  - http://git-scm.com/

- Mercurial
  - http://mercurial.selenic.com/

- Bazaar
Centralized vs Distributed VCS

- Advantages of distributed VCS over centralized
  - More powerful and detailed change tracking => fewer conflicts
  - No centralized server necessary, all actions except sharing repositories are local
  - Branching and merging more reliable
  - Fast

- Drawbacks of distributed VCS:
  - Harder to understand
  - Revisions are not easy-to-understand incremental numbers
  - Easier for novices to make mistakes
Git

- **Readings**: [Git book, chapters 1 - 3]
  - General overview
  - Creating repositories
  - Sharing repositories
  - Typical workflow

- Git is available for most systems, also installed on ix

- First-time Git setup (Sec. 1.5 in Git Getting Started) -- do on your machine or ix if you haven’t used git before
Basic Git workflow

First:
- Create or clone a repository

Then:
- You modify files in your working directory
- You *stage* the files, adding snapshots of them to your staging area
- You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory
- You pull (or rebase), then push your changes
Exercises

1. Connect to a Unix shell, e.g., ssh to ix, create a top-level working directory, e.g., name it `topdir`
2. In the `topdir` directory, create a Git repository called `myrepo` (use `git init --bare myrepo`)
3. In the `topdir` directory, create a new clone of `myrepo`, called `myrepo1`
4. In the `myrepo1` local repository directory
   a. Add a file `test.txt`
   b. Commit and push your changes
5. In the `topdir` directory or on a different machine, create a new clone of `myrepo`, called `myrepo2`
6. In the `myrepo2` local repository directory
   a. Change the file you created in step 4, commit and push the changes.
7. Go back to `myrepo1` and pull all changes