VERSION CONTROL

Version Control

* System for managing changes to (especially plain text) files
* Associates changes made to a group of files with a “revision”
* Applies timestamps, other information, to revision
* Supports retrieving any previous (or current) version
* Supports many users making changes to files simultaneously

Basics

* Software manages a database of changes
* User “commits” changes to the database
* User can “revert” changes to a past version, instead
* Can generally “diff” versions to see what changed
* Authorized users can “check out” a complete copy of a revision

History: SCCS

* Source Code Control System
* Written at Bell Labs in 1972
* Uses a technique called “interleaved deltas”
    * Store only changed portions of the file
* File format still used internally by systems like BitKeeper

History: RCS

* Revision Control System
* Written in the ‘80s at Purdue
* Designed as a free, more evolved alternative to SCCS
* Only supports working with single files
* Does not use a central repository
* Still occasionally used for server admin scripts

CVS

* Concurrent Versions System
* Developed as a series of scripts to add project support to RCS
* Scripts published in ’86, Version 1.0 submitted to FSF in 1990
* Introduced a formal notion of “branching” to version control
* Maintains a central repository to which changes are committed
* Very, very widely used, though this is changing

Branching

* Duplicate an object (file, directory, etc) so modifications can happen in parallel
* Copies called “child branches”, copied objects called “parents”
* Upper-most branch called “trunk”
* Changes are generally “merged” back into parent later
* Branches not intended for merging called forks

Subversion

* Begun in 2000 to be a mostly-compatible successor to CVS
* Supports renaming files, etc with full revision history
* Native, efficient support for binary files
* Multiple access protocols: filesystem, WebDAV, and svn
* Built in support in Eclipse, XCode, Visual Studio
* Available on lab machines

Multi-User Version Control

* Allows programmers to collaborate without emailing files
* Traditionally, a single server maintains the “official” copy
* Developers can check out copies, make changes, commit them
* Revisions are tagged with information on who did the commit
* Changes propagated to other users via update mechanism
* Support conflict resolution between merges (what happens if two users change the same thing)

Distributed Version Control

* All users maintain repositories with version history
* Central repository not needed, though often used
* No need for network access & operations are faster
* Can commit to local repo without committing to central repo
* Access control, security can be harder to attain
* Bazaar, Mercurial, and Git are popular examples