
Software Design: Version Control

a.k.a. “Source control” or “Revision control”

Slides by: Greg Wilson
gvwilson@cs.utoronto.ca

Problem: Synchronizing Files

- Want to work on one set of files on four different machines
 - Home, this office, other office, friend's
- Option 1: single file system
 - Difficult to set up (and make secure)
 - Inflexible: what if I'm in someone else's office?
- Option 2: Carry around a floppy
 - Have to remember to copy files onto it
 - My project might not fit onto just one

Synchronizing Files (cont.)

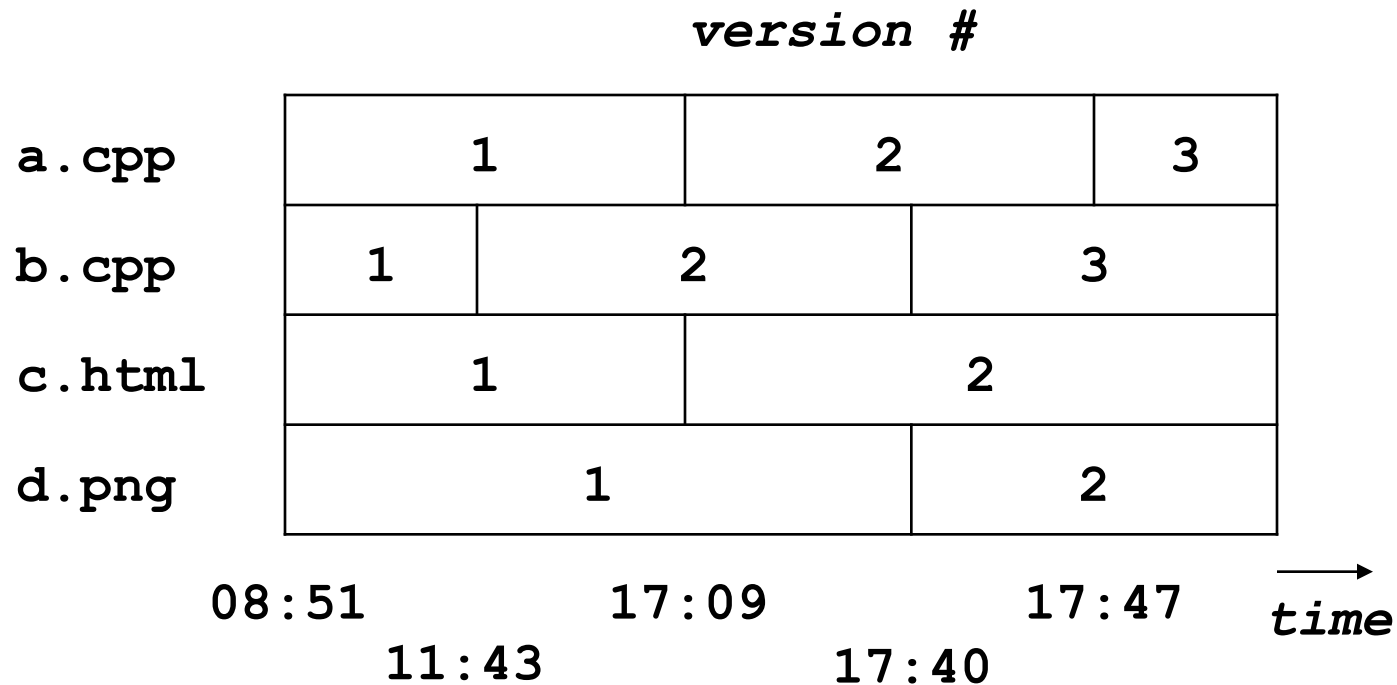
- Option 3: mail, ftp, scp, etc.
 - Still have to remember to push and pull exactly the right files at exactly the right time
- Option 4: get the computer to do the work
 - Keep a master copy in one place
 - Write a program to push and pull changes
- *If it's worth repeating, it's worth automating*

Problem: Undoing Changes

- Sometimes want to *revert* changes to a file
 - Start work, realize it's the wrong approach, want to get back to starting point
 - Like "undo" in an editor
- Solution: keep copies of old files in repository
 - Instead of overwriting `xyz.cpp`, rename it to `xyz.cpp.1`
 - Next time it changes, create `xyz.cpp.2`, etc.
 - Also record times: "What did these files look like at noon last Friday?"

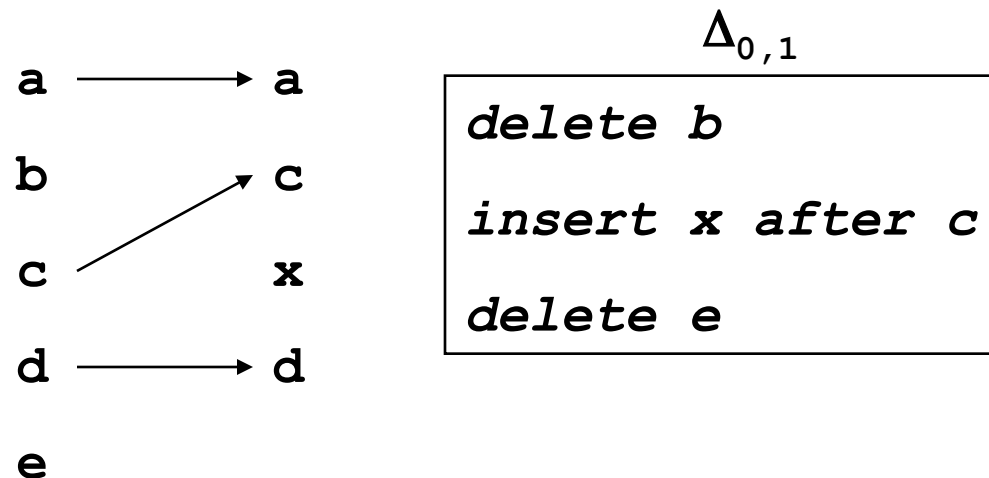
File Histories

- Two ways to describe a file
 - Version #3
 - At time 15:30



Storing Differences

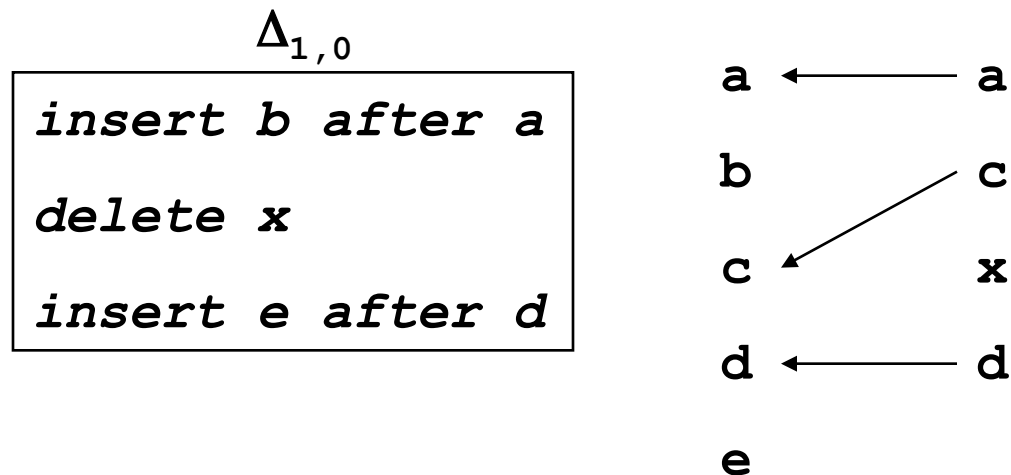
- Storing whole files very expensive
 - So just store differences



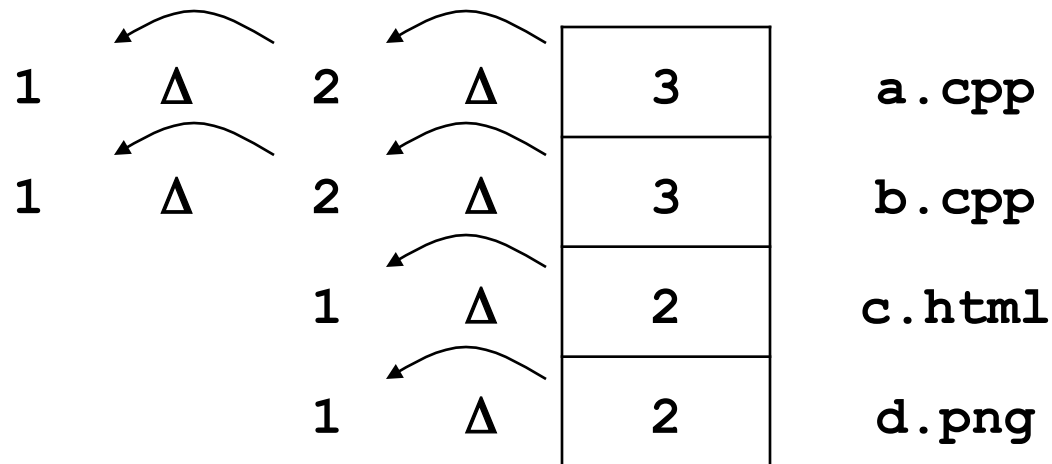
- $\text{file \# 0} + \Delta_{0,1} = \text{file \# 1}$
 - By induction, create any file from original

Backward Differences

- In practice, want recent versions of files much more often than old versions
 - So store latest version of file plus *backward differences*



Final Storage Scheme



Problem: Sharing Files

- How can Angela and Hassan work on the same files at the same time?
 - ✗ Edit the same physical files simultaneously
 - ✗ Copy files to and from a shared directory by hand
 - ✓ Use the version control system
- Record user ID in diff
 - Who changed what? When?

Managing Concurrency

- What happens if two (or more) people want to edit the same file at the same time?
- Option 1: prevent
 - Only allow one person to have a writeable copy of the file at once
 - Microsoft Visual SourceSafe
- Option 2: patch up afterwards
 - "Easier to get forgiveness than permission"
 - CVS, Perforce, and most other modern systems

Sharing Example

- Angela and Hassan both have copies of **a.cpp#5**
- Angela *commits* her changes
 - Creates **a.cpp#6** as usual
- Then Hassan wants to check in his changes
 - Conflict!
 - Must *resolve* differences by hand
 - Then commit the change to create **a.cpp#7**

Backing Out

- Angela decides there's something wrong with Hassan's work
- Can *revert* his change by:
 - Putting master copy of a.cpp back into state #6
 - Deleting Hassan's *delta*
- What if several files changed at once?
 - CVS doesn't record change sets explicitly
 - Better systems (e.g. Perforce) do
 - Revert entire set of changes in one step

CVS

- Concurrent Version System

- Command-line tool on Unix and Windows
- Built on top of older system called RCS
- Several graphical and web interfaces

- General model

- Repository (archive) – centralized storage area
- Sandbox (working directory) – local copy

CVS

- Typical interaction:
 - Check out the most current version of a module from the repository
 - Edit, test, debug
 - Commit the changes back to the repository

- Manual at <http://www.cvshome.org/docs/manual/>

Setting up CVS: Creating A Repository

- Create an empty repository (just a directory where you want CVS info to go)
- Set your CVSROOT environment variable
 - `setenv CVSROOT <directory>`
- Initialize CVS (from anywhere)
 - `cvs init`

Adding a Module

- Go to directory of files you want to add
 - `cvs import -m "Message" <module> <vend> <release>`
- If we wanted to add a module for hw7:
 - `cvs import -m "Our hw7 module" hw7 mturk start`

↑
comment

↑
name of
module

↑ ↑
anything

Common CVS Commands

<code>cv</code> s checkout root	Get initial copy
<code>cv</code> s add ...filenames...	Add new files
<code>cv</code> s update [...filenames...]	Compare state
<code>cv</code> s commit [...filenames...]	Commit changes
<code>cv</code> s remove ...filenames...	Remove files [1]
<code>cv</code> s diff ...filenames...	Look at differences
<code>cv</code> s log ...filenames...	Show history

[1] There is no explicit command for removing directories

Checking out a Module

- Check out a module to work on the code and be able to store it in CVS
 - Go to a directory where you want the module (folder) to be
 - `cvs checkout <module name>`

Creates a new subdirectory called <modular name>

Updating Files

- Get newest files from repository
 - `cvs update -d`
 - `-d` checks for new sub directories too, which are left out otherwise

Committing Your Changes

- After you make changes update the repository
 - `cvcs commit`
 - Prompts for comments on each file that was changed
 - Default editor for comments on csil is `vi` – to save and quit in `vi` type `ESC : w q`
 - `cvcs commit -m "Message"`
 - Skips the editing for comments

Adding / Removing Files

- `cv`s add <fileName>
 - Adds file next time commit is done
- `cv`s rm <fileName>
 - Deletes file next time commit is done (remove from local directory first)
 - Then: `cv`s commit -m "comment..."

Checking differences

- To see differences between two versions
 - `cvs diff -c -r <ver. 1> -r <ver. 2> <fileName>`
 - Prints differences between version 1 and 2 in nicely readable format

Seeing change log

- See log of messages people have written about changes they have made to a file
 - `cvs log <fileName>`

CVS Example

```
$ cvs checkout ex_1
cvs checkout: Updating ex_1
U ex_1/startingPoint.cpp
```

```
$ cd ex_1
$ ls
CVS/ startingPoint.cpp
```

```
$ edit startingPoint.cpp myWork.cpp
```

```
$ cvs add myWork.cpp
cvs add:
  scheduling file 'myWork.cpp' for addition
```

CVS Example (cont.)

```
$ cvs update
```

```
cvs update: Updating .
```

```
A myWork.cpp
```

```
$ cvs commit -m "Started assignment 1"
```

```
cvs commit: Examining .
```

```
RCS file: /repo/ex_1/myWork.cpp,v
```

```
done
```

```
Checking in myWork.cpp;
```

```
/repo/ex_1/myWork.cpp,v <-- myWork.cpp
```

```
initial revision: 1.1
```

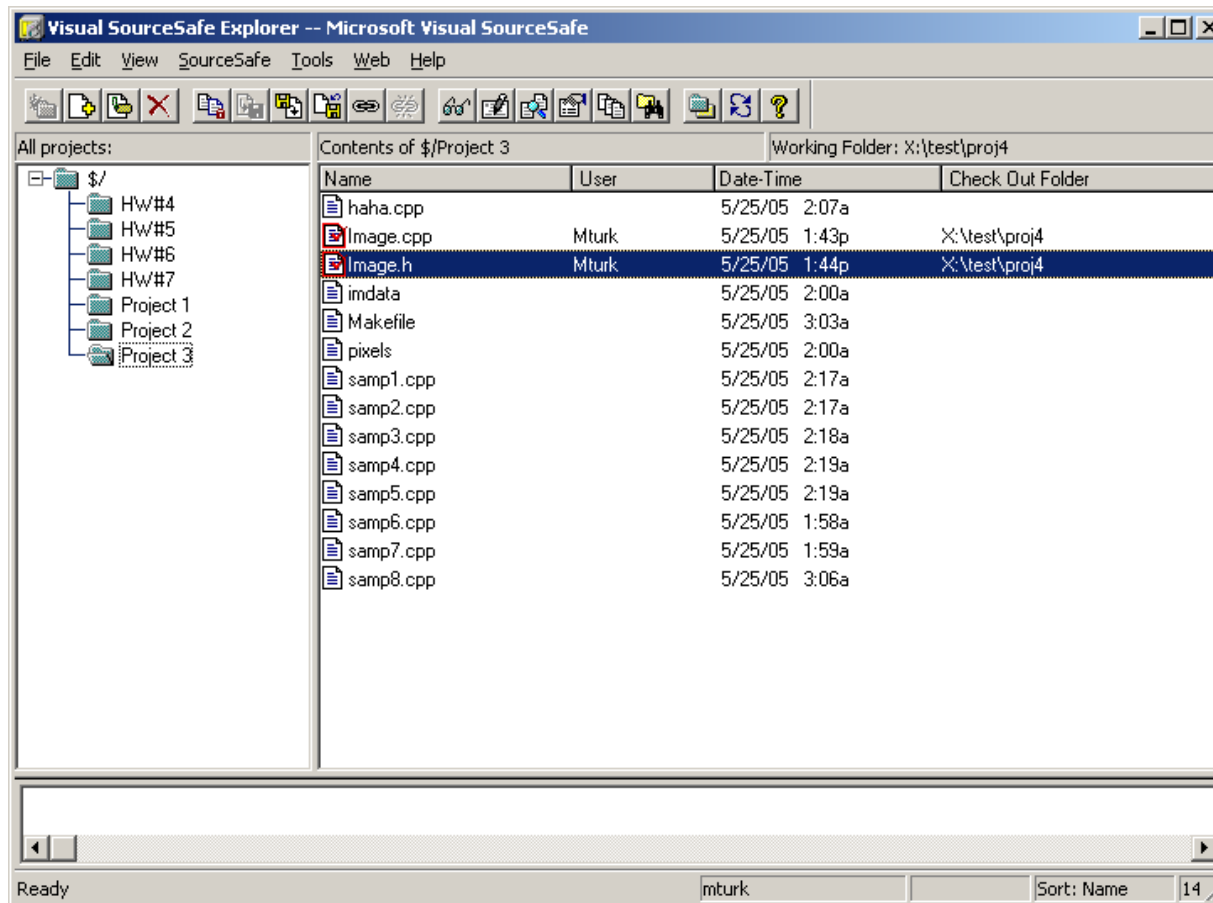
```
done
```

Good (simple) CVS Reference

- <http://www.cs.hmc.edu/qref/cvs.html>

Other popular revision control software

- Microsoft Visual SourceSafe



Other popular revision control software

- IBM (Rational) ClearCase
- Perforce

The Bottom Line

- Here's what will happen to you eventually if you don't use revision control regularly

