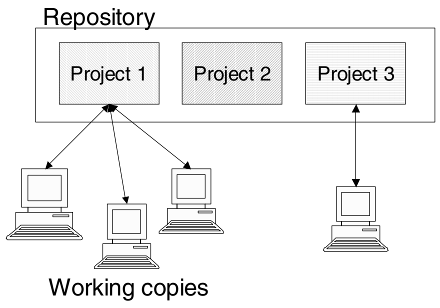
Why use version control?

- Saves all previous versions of all files so that you can undo (buggy) edits.
- Logs changes to files so you can trace how your sources have evolved over time.
- Mediates conflicting changes made by several different users---helps keep consistency.



Setting up CVS

- Create the repository:

```
% mkdir ~/cvsroot
% [set permissions for ~/cvsroot]
% setenv CVSROOT ~/cvsroot
% cvs init
```

- Set the CVSROOT environment variable in your `.cshrc` file.

Creating a Project

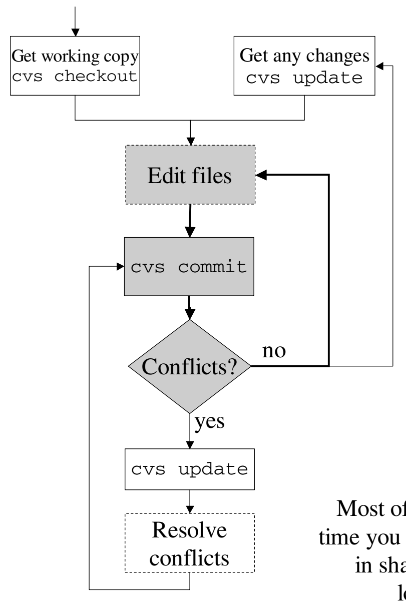
- Create a project from existing files:

```
% cd base64
% cvs import -m "base64" base64 carlk start
```

- Checkout the new project and delete the old files.

```
% cvs checkout base64
```

Development Process with CVS



Most of the time you stay in shaded loop.

Dealing with Conflicts

- If another user has committed changes to lines you have edited, CVS will report a conflict.

Your version

```
decode (infile, stdout);
<<<<<<< decode64.c
  if( fclose (infile) ) return 0;
=======
  if( fclose (infile) ) printf(" couldn't close file.");
>>>>>>> 1.5
```

Repository version

- Manually edit to remove the conflict.
- Edits on different lines are automatically merged.

Frequently Used Commands

- Get the latest revisions of files with:
```
% cvs update
```
- Write changes back to repository with:
```
% cvs commit -m"log msg"
```

If you omit –m the editor given in the EDITOR environment variable will be started.

- Add files to project with:
```
% cvs add -m"log msg" files…
```
Use `-kb` for binary files.

- Remove files from project with:
```
% rm file.c
% cvs remove -m"log msg" file.c
```
- Most commands default to work on the current directory and all its subdirectories.

Keywords

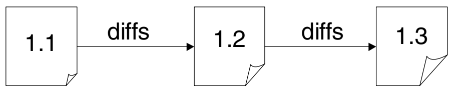
- Useful to put dynamic information in source file.
- When you get a new revision, keywords are replaced with information about the file.

Keyword	Replaced with
`$Author$`	Username who checked in this revision.
`$Date$`	Date the this revision was checked in.
`$Id$`	A string containing the filename, revision, author, date, and some status info. Useful at tops of files.
`$Revision$`	The revision number of this file.
`$Log$`	The complete history of this file.

```
/* $Id$
 * Author: $Author$
 * $Log$
 */
```

```
/* $Id: decode64.c,v 1.2 2001/04/01 17:32:45 carlk Exp $
 * Author: $Author: carlk $
 * $Log: decode64.c,v $
 * Revision 1.2  2001/04/01 17:32:45  carlk
 * added keywords
 */
```

Tracking Changes



- CVS keeps track of all revisions of your files.
- View changes between two revisions:
```
% cvs diff -r 1.2 -r 1.3 file
% cvs –D "Apr 2" –D "Apr 3" file
```
- View the log comments:
```
% cvs log file
```
- View when each line changed:
```
% cvs annotate file
```

Tracking Repository History

- View history of commits:
```
% cvs history -c
```
Can view other events:
```
% cvs history -x event_codes
```

Code	Event
O	Checkout command issued.
C	Conflict detected.
G	Merging was necessary (but no conflicts).
U	A working file was copied from the repository.
A	A file was added.
M	A file was modified.
R	A file was removed.

e.g.
```
% cvs history -x AR main.c
```

Getting Previous Versions

- Retrieve a given revision with:

```
% cvs update -r 1.1 files…
% cvs update -D "Apr 2" files…
```

- Date can be in just about any reasonable format.

- **All subsequent updates will get given revision.**

- Start getting the most recent version:

```
% cvs update -A files…
```

- Remembering version numbers and dates can be hard; use symbolic tags (next slide).

Tags

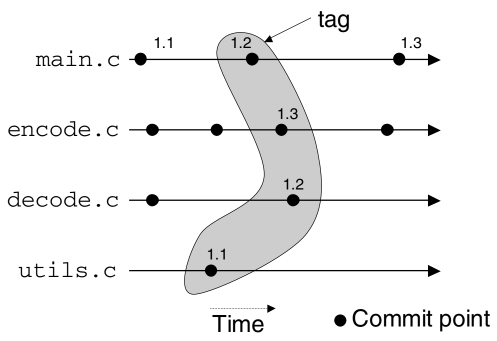
- Tags associate a name with a set of particular revisions of some files.
- Useful to mark a "release."

- Create a new tag with:

```
% cvs tag mytag main.c encode.c …
```

- Checkout a set of tagged files with:

```
% cvs checkout -r mytag
```



Ignoring Files

- Many types of files should not go into the repository (*.bak, *.o, core) --- CVS ignores most of these.

- Can make CVS ignore any file by putting file patterns into `~/.cvsignore`. Eg.

```
encode64
decode64
*.gif
```

- CVS won't include matching files in any commit or update operation.
- It won't generate warning messages if these files are not in the repository.

Running Scripts on Events

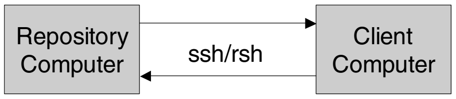
- Can force CVS to run shell scripts on events like commit.
- `cvsroot/CVSROOT` contains a file called `commitinfo` with lines of the following format:

$$\underbrace{\texttt{base64.*}}_{\text{Regexp matching directory name.}} \quad \underbrace{\texttt{myshellscript}}_{\text{Shell script to run.}}$$

- If regexp matches the directory of a committed file, *myshellscript* will be run.
- Regexp can also be `ALL` or `DEFAULT`.
- If *myshellscript* exits with non-zero status, commit will not be allowed.

- Similar mechanisms for other events.

CVS Over the Network



(e.g. hats)

- No server process required.
- Uses ssh/rsh to communicate. On client:

```
% setenv CVS_RSH ssh
```

- Setup ssh so that it doesn't ask for a password.
  1. generate keys with no password
  2. put the public key file on server, add to authorization file.
  3. add *hostname username* to .shosts.
- Client sets CVSROOT to:

:ext:*uid*@boater.princeton.edu:*dir*/cvsroot

CVS On Other Platforms

- **CVS home page** has clients for Windows XX, Linux, and other Unix flavors.

`http://www.cvshome.org/`

- On other platforms, setup and use is similar, but exact syntax may differ.
- CVS home page also has excellent manual.
- **SourceForge** hosts open source projects for free. They provide:
  - CVS repositories
  - web servers
  - compile farm
  - and more…

`http://sourceforge.net/`

CVS Quick Reference

File Manipulation	
`add`	Add a file or directory to be managed by CVS. `% cvs add -m "new feature" mult.c`

`remove`	Mark a file deleted in the repository; all previous versions will still be available. You should remove the working copy before issuing the this command.
	`% rm mult.c`
	`% cvs remove -m "deleted feature" mult.c`

`update`	Copy the most recent versions of the given files into the working directory. File defaults to ".".
`commit`	Merge any changes made back into the repository. `% cvs commit -m "fixed bug" mult.c`
`tag`	Create a tag or a branch.
File Information	
`status`	Show information about files. Use `-v` to see what tags are attached to the files.
`diff`	Show the changes made between any two versions of a file. Defaults to showing the changes between working copy and most recent version.
`log`	Show the history of given files.
Setup	
`init`	Create a new repository (probably only need to do this once). `% cvs init -d ~/cvsroot`
`import`	Create a new project from existing sources (in current directory). `% cvs import -m "prj" prjname vendtag reltag`

Most commands take `-m` to specify a log message and `-r` to operate on a given revision/tag.