CVS For Fun and Profit

Mark Allman
mallman@grc.nasa.gov

Overview

- Why use version control?
  - track software changes for development
    - e.g., what did I actually change since yesterday when everything worked fine?
    - e.g., which version of the software did I use for my experiment last month?
  - track tools written for research
    - e.g., simulation and analysis scripts
  - track configs used in experiments
    - e.g., router config files

Overview (cont.)

- track changes to papers
  - who has seen what version?
  - who hacked on section 3 last and what did they do?
  - which version did I send to the conference and which was the tech report?

Overview (cont.)

- We all have our ad-hoc systems for doing this
  - keep a "working version" on a shared disk
  - tuck away a tarball of today’s code in case I mess everything up tomorrow
  - burn "finished" versions to CD with meaningful names
  - maybe keep a notes file
  - etc.

Overview (cont.)

- CVS provides a framework for version control that is more formal than our ad-hoc methods (yet not too formal as to be difficult to use)
- CVS also provides a way to share workspace with others and keep track of what everyone is doing

CVS Versions and Interfaces

- CVS interfaces and versions exist for unix, macos and windows
  - GUIs and text interfaces
  - Hooks for editors
  - Etc.
- All my examples are using the basic unix text interface
- Even if that doesn’t describe your situation the concepts and terminology of the talk apply

CVS Version Numbers

- CVS keeps its own, internal version number for each file
- These version numbers do not have to correlate to higher level version numbers that you wish to assign to a given snapshot of the code, paper, etc.
- The CVS version numbers are basically a function of the number of times a file has been updated in the repository
  - (and, also, of the "branch" a file is on)

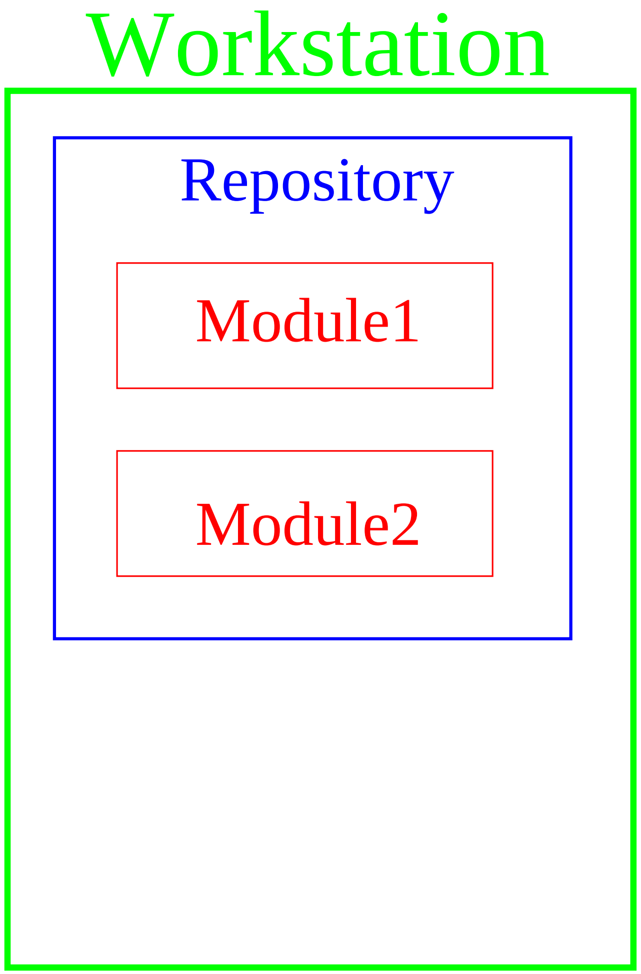
CVS Version Numbers (cont.)

- All files in the repository do not have to have the same version number to keep track of things
  - a .c file may have a CVS version number of 1.45 because it is editted often
  - meanwhile the corresponding .h file may have a CVS version number of 1.7 because it is not editted much

Repositories and Modules

- CVS uses repositories of modules to track things
- The respository is the authoritative store for whatever is under version control
  - we generally keep one repository (or, at least a small number)
- Within each repository we keep modules
  - e.g., one for each project
- Users do not edit things in the repository directly, but rather through a defined interface of commands

Repositories and Modules (cont.)



Creating a Repository

```
guns% mkdir /home/mallman/tester

guns% cvs -d /home/mallman/tester init

guns% cd /home/mallman/tester

guns% ls
CVSROOT/

guns% ls CVSROOT
Emptydir      [...]
checkoutlist   [...]
checkoutlist,v [...]
commitinfo     [...]
commitinfo,v   [...]
```

- "You can look, but you better not touch"

Locating a Repository

- The "-d" argument to CVS is used to indicate which repository you want to use
  - "-d /full/path/to/repository"
    - this indicates where the repository is in your local filesystem
  - "-d hostname:/full/path/to/repository"
    - this indicates the full path name of a repository on a remote machine

- Typing "-d foo" all the time is tiresome. So, you can set the "CVSROOT" environment variable to the argument you would give to the -d option
  - e.g., "export CVSROOT=/mycvsroot"

Accessing Remote Repositories

- Basically, use ssh
- There is a CVS server (a "pserver")
  - insecure
  - mostly used for anonymous CVS access

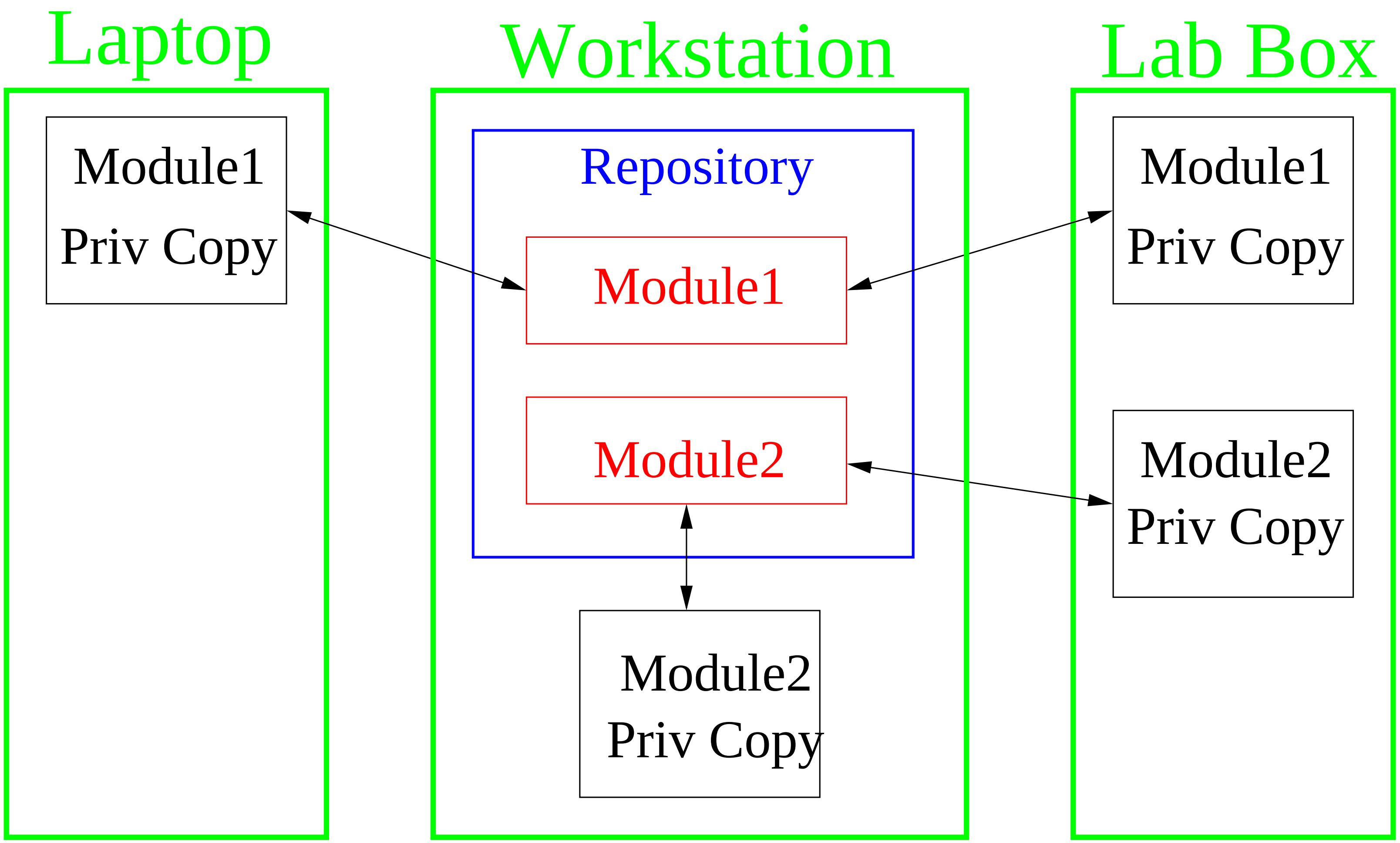
- To use ssh for CVS set the following environment variable:
  - export CVS_RSH=ssh

Private Copies

- Since you cannot edit files in the repository directly we need another copy of the files that you can touch (edit, add, delete, etc.).

- Making a private copy of a module:
  - cvs checkout module_name
  - cvs co module_name

Private Copies (cont.)



Private Copies (cont.)

```
guns% cvs co cvs-talk
U cvs-talk/Makefile
U cvs-talk/cvs.mm

guns% cd cvs-talk

guns% ls
CVS/          Makefile      cvs.mm

guns% ls CVS
Entries       Repository    Root
```

Private Copies (cont.)

- You can edit the files in a private copy as you please
- But, stay away from the "CVS" subdirectories
- Also, if you’re in the cvs-talk directory you no longer have to use "-d" because the information is stored in the "CVS" directory

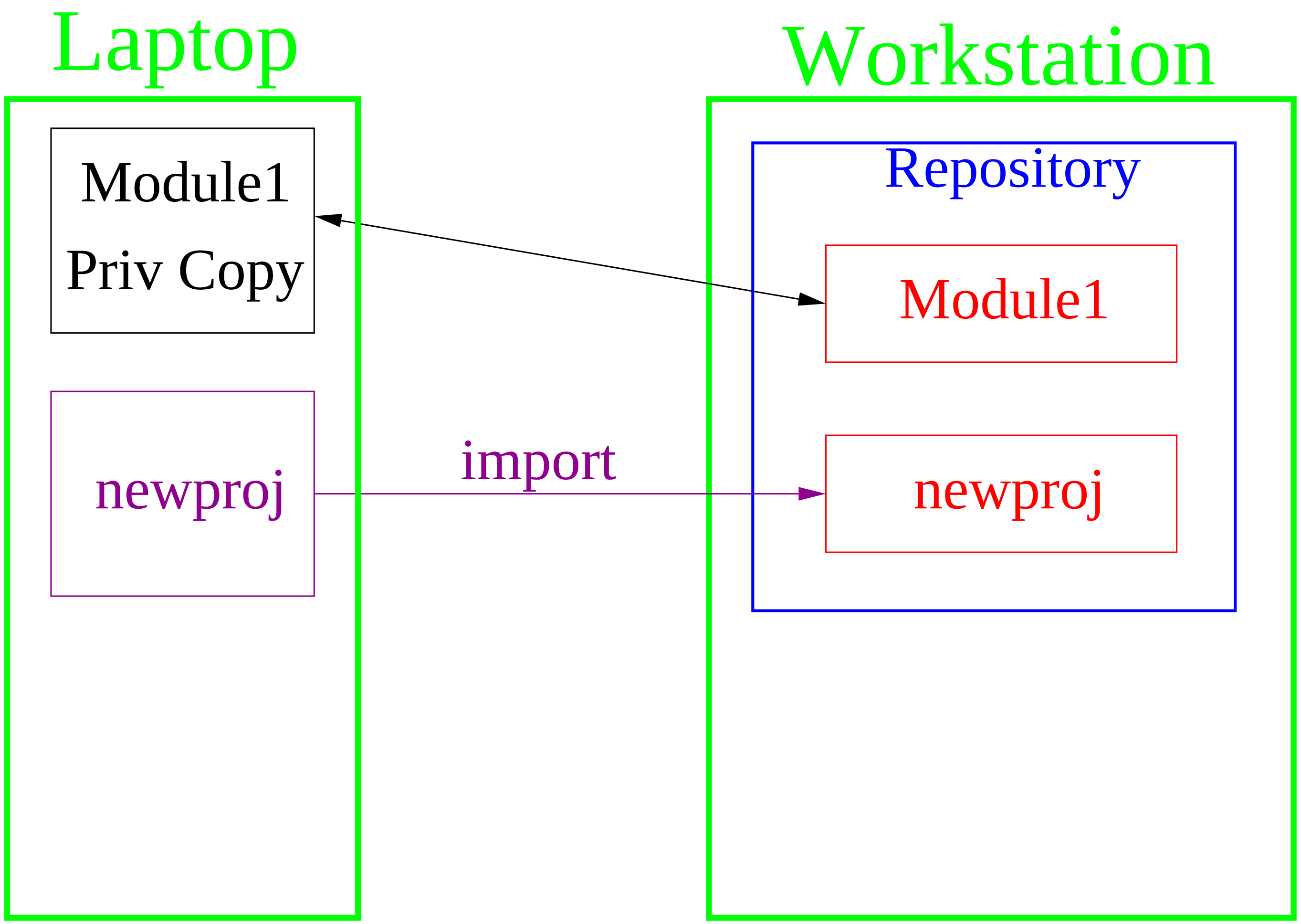
Starting a Module

- You start with an initial directory of stuff. From there you "import" a new module into the repository.

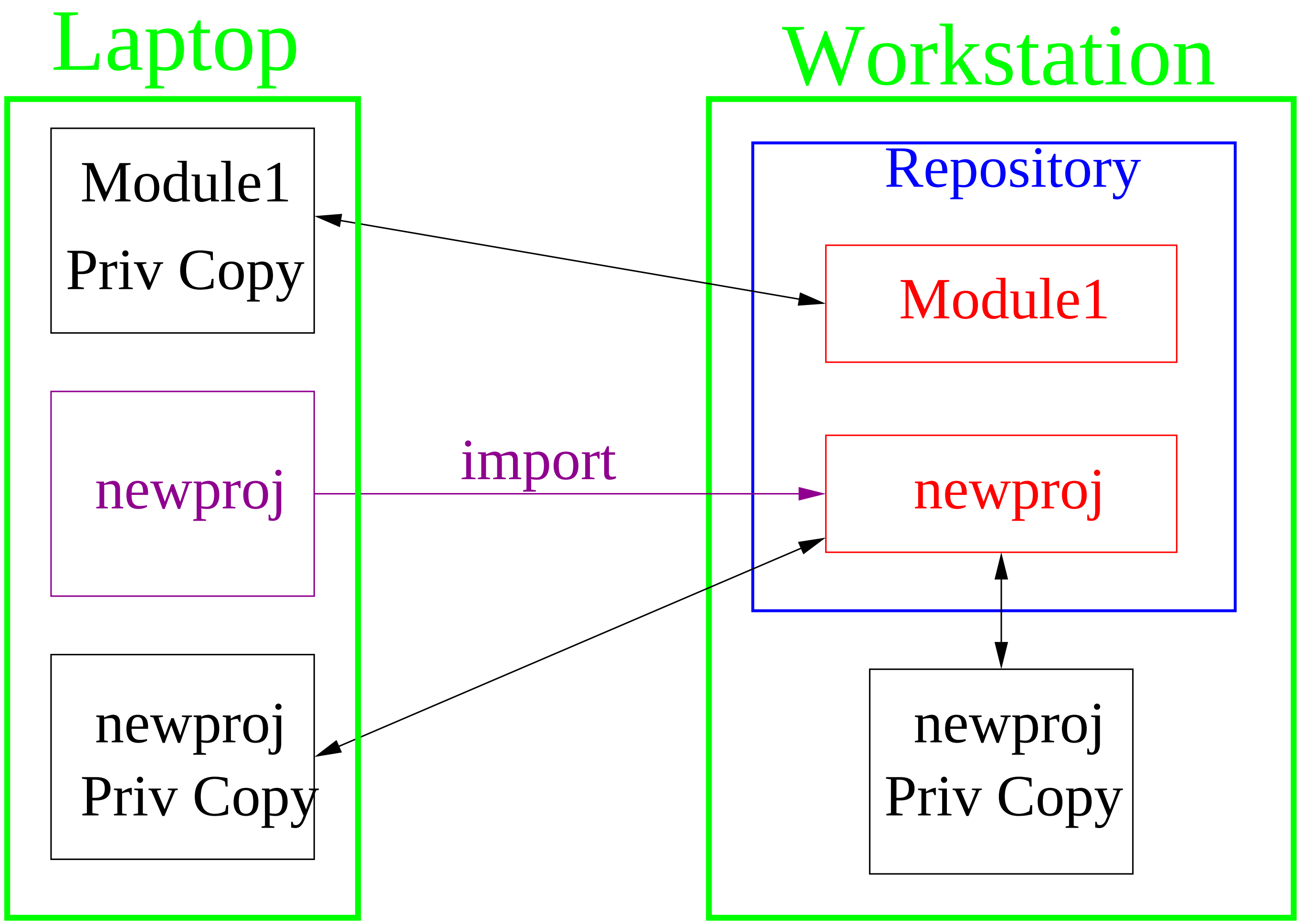
- Importing:
  - cvs import module_name vendor release_tags

- But, "new-module" is not a private copy of the CVS repository
  - You must checkout a copy from CVS to get all the meta-files, etc.

Starting a Module (cont.)



Starting a Module (cont.)



Starting a Module (cont.)

```
guns% cd newproj

guns% ls -1
Makefile
foo.c
analysis.py

guns% cvs import newproj mallman init
 [CALLS EDITOR]
N newproj/Makefile
N newproj/foo.c
N newproj/analysis.py

No conflicts created by this import
```

Starting a Module (cont.)

```
guns% cd ..

guns% rm -rf newproj

guns% cvs co newproj
cvs checkout: Updating newproj
U newproj/Makefile
U newproj/analysis.py
U newproj/foo.c
```

Committing Your Changes

- Once you have changed a file in your private copy and are satisfied that it is right, you need to commit the file to the repository

- Committing:
  - cvs commit filename
  - cvs commit

- You will be asked to enter a log message when committing

Committing Your Changes (cont.)

```
lawyers% cvs co cvs-talk
U cvs-talk/Makefile
U cvs-talk/cvs.mm

lawyers% cd cvs-talk

lawyers% emacs cvs.mm

lawyers% cvs commit
 [CALLS EDITOR]
Checking in cvs.mm;
/home/mallman/.cvsroot/cvs-talk/cvs.mm,v  <--  cvs.mm
new revision: 1.2; previous revision: 1.1
done
```

Committing Your Changes (cont.)

```
CVS: ----------------
CVS: Enter Log. [...]
CVS:
CVS: Committing in .
CVS:
CVS: Modified Files:
CVS:  cvs.mm
CVS: ----------------
```

Committing Your Changes (cont.)

- When do I commit files?
  - wwwweeeellllllll ....

Updating Your Copy

- After checking out a module you often want to update the copy with changes that have been made to the module in the repository:
  - by a colleague
  - by you from a different private copy of the module
    - e.g., the private copy that lives on your laptop
- Updating a private copy:
  - cvs update filename
  - cvs update
  - cvs up
  - cvs up -d -R

Updating Your Copy (cont.)

```
guns% cd cvs-talk
guns% cvs up
cvs server: Updating .
P cvs.mm
```

Adding Files

- Sometimes (!) it is handy to be able to add things to the repository

- Adding files:
  - cvs add filename
  - cvs commit [filename]

- Adding directories:
  - cvs add directory
  - That is, you do not have to commit to make a directory addition
  - Also, note that adding a directory does not add its contents

Adding Files (cont.)

```
guns% mkdir figs
guns% cvs add figs
Directory /home/mallman/cvs-talk/figs added to the repository
```

Adding Files (cont.)

```
guns% cd figs

guns% xfig 1.fig

guns% cvs add 1.fig
cvs add: scheduling file ‘1.fig’ for addition
cvs add: use ’cvs commit’ to add this file permanently

guns% cvs commit
 [CALLS EDITOR]
cvs commit: Examining .
RCS file: /home/mallman/.cvsroot/cvs-talk/figs/1.fig,v
done
Checking in 1.fig;
/home/mallman/.cvsroot/cvs-talk/figs/1.fig,v  <--  1.fig
initial revision: 1.1
done
```

Removing Files

- To remove a file from a repository you nuke the file in your private copy, tell the repository you want to remove the file and then commit.

- Removing file:
  - nuke the file ("rm file")
  - cvs remove file
  - cvs commit

Removing Files (cont.)

```
guns% cd cvs-talk/figs

guns% rm 1.fig

guns% cvs remove 1.fig
cvs remove: scheduling ‘1.fig’ for removal
cvs remove: use ’cvs commit’ to remove this file permanently

guns% cvs commit
 [CALLS EDITOR]
cvs commit: Examining .
Removing 1.fig;
/home/mallman/tester/foo/figs/1.fig,v  <--  1.fig
new revision: delete; previous revision: 1.1
done
```

Getting Status

- Sometimes you want to know the status of a file in your private copy with respect to the repository
  - Do I have an un-committed version of this file?
  - Do I need an update for this file?
  - Etc.
- Getting status:
  - cvs status filename
  - cvs status

Getting Status (cont.)

```
guns% cvs status cvs.mm
==============================================================
File: cvs.mm            Status: Locally Modified

   Working revision:    1.4
   Repository revision: 1.4
/home/mallman/.cvsroot/cvs-talk/cvs.mm,v
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)
```

Getting More Status

- You can grab all the commit entries that people have written about a particular file to get some version history.
- Version history:
  - cvs log filename
  - cvs log

Getting More Status (cont.)

```
guns% cvs log cvs.mm

RCS file: /home/mallman/.cvsroot/cvs-talk/cvs.mm,v
Working file: cvs.mm
head: 1.4
[...]

----------------------------
revision 1.4
date: 2003/01/16 14:28:26;  author: mallman;  state: Exp;  lines: +1 -0

Added slides on adding and removing file from CVS.
----------------------------
revision 1.3
date: 2003/01/16 14:26:49;  author: mallman;  state: Exp;  lines: +13
-3


Added slides on updating private copies from the respository.
```

Finding Changes

- Often what you want to know is:
  - what did I change that really hosed things up?
  - what changed between this experiment and the last experiment?
  - what did Bob change in the paper last night?

Finding Changes (cont.)

- Finding differences:
  - cvs diff filename
    - changes between your private copy and repository copy
  - cvs diff -r x.y filename
    - changes between your private copy and version x.y in the repository
  - cvs diff -r x.y -r a.b filename
    - changes between versions x.y and a.b in the repository (without taking your private copy into account)

Tagging

- It is often useful to tag all files in the repository under one name so that you can get back to some known point later
  - a particular release of software
  - all code and configs used for some demo

- To tag:
  - cvs tag tagname
    - tagname cannot contain periods

- E.g.:
  - cvs tag traffic_v_1-2-23

Tagging (cont.)

- Then you can:
  - cvs co -r traffic_v_1-2-23 traffic
  - cvs diff -r traffic_v_1-2-23

Binary Files

- A word about CVS and binary files...

Other Features

- There is lots more that CVS can do
  - branching and merging
  - watching, releasing, annotating
  - history
  - email notification for commits
  - lots of interesting riffs on the commands we talked about
  - etc.

Final Word

- Use CVS:
  - it's just good science
  - it will save you time in the long run
  - it will help collaboration
- If you don't particularly like CVS specifically but want to use version control there are lots of other packages that can help
  - RCS
  - BitKeeper
  - see SourceForge

Pointers

- CVS Homepage:
  - http://www.cvshome.org/
- Slides from this talk:
  - http://roland.grc.nasa.gov/~mallman/talks/cvs.ps
  - http://roland.grc.nasa.gov/~mallman/talks/cvs.pdf
- Principles of version control:
  - http://www.perforce.com/perforce/bestpractices.html