CVS: Concurrent Versions System

CVS
Concurrent Versions System

J.C. Gonzalez

1st MAGIC Telescope Software Meeting
The Eng
February 11th-13th, 1999
Outline of the talk

- What is and what is not CVS?
- Basics of CVS
- Sample session
- Managing projects with CVS
What is CVS?

- Concurrent Versions System
- Record history of files in a clever way
- Group developing

What is NOT CVS?

- Not a build system
- Not a substitute for management
- Not a substitute for developers communication
- Not a do-it-all utility
Basics of CVS

- Repository
- Modules
- Revision numbers / Branches
- Versions, revisions, releases
- Tags to release
Repository

- CVS Repository → complete copy of all the files and directories with under version control.

- Use CVS commands to access the files:
  1. Check-out module
  2. Modify, update, create new files
  3. Check-in back (commit) module

- Can be either local or remote

- Repository = CVSROOT + modules
Modules

- **Modules**: Groups of files close-related each other.
  - A library
  - An application
  - A small utility
  - A sub-program for a big application

- CVS is **module-oriented** (projects)

- Modules allow several developers working in different section of the same application, at the same time, without critical conflicts.
Revision numbers and Branches

- **Revision Numbers**: “Versions” of the files.
  - Example: 1.1, 1.2, 1.3.2.5, 1.3.2.5.1.2
  - Updated when you “check-in” (commit)
  - Sequential, with possible steps into a mayor revision number: 1.3 - 1.4 - 2.1 - …

- CVS is not limited to “linear development”
  Revision Numbers → Revision Tree → Branches
Revision numbers and Branches (II)

- Each branch → a self-maintained line of development
- Modifications in branches → easily transported back to the main trunk
- What are branches good for?
  - Develop new algorithms to substitute old ones.
  - Expand performance without disturbing main development (adding new options, ...)
  - Correct bugs in earlier versions
 Versions 

- Versions[files] \neq Versions[applications]

- Notation:
  
  \textbf{Revision} \equiv \text{Versions[files]}

  \textbf{Release} \equiv \text{Versions[applications]}
Tags to release

- Diferent files have different revisions history
- **Tags:** Symbolic revisions → **Releases**
CVS: Concurrent Versions System

file1
1.1
1.2
1.3
1.4

file2
1.1
1.2
1.3

file3
1.1
1.2
1.3

file4
1.1
1.2
1.3

file5
1.1

TAG
CVS: Concurrent Versions System

file1  file2  file3  file4  file5

1.1
1.2
1.1
1.3
1.1
1.2
1.3
1.4
1.1
1.2
1.3
1.6
1.3
1.4

TAG
Sample session

Context

We are working on a new algorithm to be implemented in the software trigger layer for MAGIC. The source code consists of some C files and a Makefile. The program is called “ta” (trigger algorithm) and this is also the name of your module in the repository.
Getting the code

HAL10M:~/> cvs checkout ta
HAL10M:~/> cd ta
HAL10M:~/ta> ls
CVS    Makefile   algor.c   main.c   param.c
HAL10M:~/ta> _

Modifications + hacking away

HAL10M:~/ta> emacs algor.c
...
HAL10M:~/ta> ls
CVS    Makefile   algor.c   algor.c~   main.c
param.c
HAL10M:~/ta> _
Did it work?

HAL10M:\~/ta> make
  compiling algor.c ...
  compiling main.c ...
  compiling param.c ...
  linking...
  done.
HAL10M:\~/ta> ls
CVS  Makefile  algor.c  algor.c~  algor.o
     main.c  main.o  param.c  param.o  ta*

Committing your changes

HAL10M:\~/ta> cvs commit -m "Added search of local islands" algor.c
Checking in algor.c;
/usr/local/cvsroot/ta/algor.c,v  <--  algor.c
new revision: 1.5; previous revision: 1.4
  done
HAL10M:\~/ta> _
Cleaning up

HAL10M:~/ta> cd ..
HAL10M:~/> cvs release -d ta
M param.c
? ta
You have [1] altered files in this repository.
Are you sure you want to release (and delete) module ‘ta’: n
** ‘release’ aborted by user choice.
HAL10M:~/> cd ta
HAL10M:~/ta> cvs diff param.c
...
HAL10M:~/ta> cvs commit -m "Added search of local islands" param.c
Checking in param.c;
/usr/local/cvsroot/ta/param.c,v <-- param.c
new revision: 1.3; previous revision: 1.2
done
HAL10M:~/ta> cd ..
HAL10M:~/> cvs release -d ta
? ta
You have [0] altered files in this repository.
Are you sure you want to release (and delete) module ‘ta’: y
HAL10M:~/> _
Managing projects with CVS

- Defining projects
- History
- Revision management
- Release management
- Multiple developers
Defining projects

- Creation of projects using old files
- Creation of projects from scratch
- Definition of the module in the repository
- CVS → recursive behaviour
History

- Log messages (cvs log)
- History database (log of CVS actions — cvs history)
- User-defined logging: customization for logging commits, check-outs, check-ins, tags, …
- Annotate command: what revision modified each line of a file?
Revision management

- Decide which policy to use regarding commits
- Several policies are possible
  - Too quickly $\rightarrow$ files may not even compile
  - Too slowly $\rightarrow$ improvements are not available
- Common policy: only to commit files when they compile
- Another policy: Force to pass a test suite
- Too controlled $\Rightarrow$ too regimented $\Rightarrow$
  $\Rightarrow$ counter-productive $\rightarrow$ get the software written
Release management

- Decide which policy to use regarding releases
- Suggestion: Frequent releases with a test suite and a test phase
Multiple developers

- CVS model: unreserved checkouts $\Rightarrow$ developers have their own “working copy”

- When committing $\Rightarrow$ possible conflicts. BUT with CVS one can bring his working copy up to date with the repository revision

- Several states for the files: Up-to-date, Locally Modified, Locally Added, Unresolved Conflict, ... 

- Informing others

- Watching files / Getting notified