Motivating Example

- J. Doe is working on a project at home, at work
- several versions in both places:

  `project, project2, newproject, new2project, finalversion, reallyfinal, another`

- different versions of same files, with different bugs
- there is a unique combination of files, each from a different machine/directory, that have the combination of features you want...

What is a Source Code Management System?

- manage what you <u>create</u>, <u>develop</u> or <u>fix</u>
- several degrees:
  - total disorder: no source code management "send me that zip tomorrow"
  - total order: the manager is peeking over your shoulder while you type
  - most code management systems are somewhere in between

Source Code Management: Concepts

- **source tree**
- file **revisions**: numeric, symbolic
- source tree **versions**: collection of revisions
- **checkout** a certain revision
- **commit** or **checkin** changes (new revisions)
- **lock/release**
- **conflict** resolution
- **watching** files
- **branches**

Source Code Management Software

- The mother of them all: SCCS
  - copyrighted Unix software/cloned by others:
  - HPUX, Solaris. supported by AT&T
  - non Y2K compliant! numerous fixes exist

`http://www.cyclic.com/cyclic-pages/sccs.html`

More Source Code Management Software

- Revision Control System (RCS)
  - Free Software Foundation
    `http://www.gnu.org/software/rcs`
  - runs on many, many platforms, including DOS
  - Functionality very similar to sccs

CVS Intro

- Concurrent Versions System
  - a central place to store your code
  - manages several versions of the code
  - manages the directory structure of the code
  - allows several people to work together
  - allows mobile users to work from several places
  - works for documents and source code

What CVS Does **<u>Not</u>** Do:

- not a build system
  - doesn’t take care of your makefiles
- not a substitute for management
  - the pointy-haired boss lives
- not a substitute for communication
- no change control
  - if it’ s buggy it’s buggy
- no process model
  - nobody tells you when to commit

More Info On CVS

- general: Pascal Molli's page

  `www.loria.fr/~molli/cvs-index.html`
- linux

  `/usr/doc/cvs*/cvs.ps`
- win32/Mac: wincvs home page

  `www.wincvs.org`
- Peter Parnes' CVS tutorial:

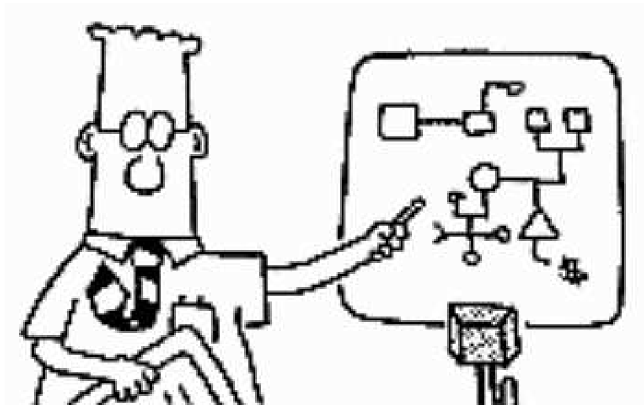
  `www.cdt.luth.se/~peppar/presentation/cvs/`

Netscape Hypertext Document

CVS Introduction Lecture

Welcome...

Mates CVS Introduction



(Thanks to Brian J. Swetland)

Peter Parnes

What is CVS?

Concurrent Versions System

CVS is a version control system. It allows you to record the history of your source files.

What does it do?

- CVS stores previous versions of files efficiently
- CVS allows a group of people to work on the same code
- CVS allows easy retrieval of old versions
- CVS runs on UNIX, Windows (95 or NT), Macintosh, or VMS

CVS is not ...

- CVS is not a build system
- CVS is not a substitute for management
- CVS is not a substitute for developer communication

Peter Parnes

CVS versus RCS and SCCS

How is it different from RCS or SCCS?

- RCS or SCCS use a lock-modify-unlock model
- CVS allows several developers to work on sources at once
- CVS is layered on top of RCS
- CVS can work as a client-server system

More Information

CVS is a large and complex system that is very powerful.

To be able to use all of its capabilities well, you should take time to read the manual. This workshop will provide a basic overview of CVS and how to use it. Consult the manual (and the man pages) for more extensive information. A postscript version of the CVS 1.9 manual is available here.

Peter Parnes

Setting up CVS

Where can I get CVS?

Any mirror of GNU software will have CVS. I usually use ftp://ftp.luth.se/pub/gnu/ Look for a file named `cvs-X.Y.tar.gz` The latest version was cvs-1.9.tar.gz as of November 1996. Pre compiled Mac and Windows versions are available elsewhere.

What else do I need?

You need a recent, non-broken diff. GNU diffutils-2.7 work well. You should have a recent version of rcs (since CVS is layered on top of it). GNU rcs-5.7 is available from your local GNU mirror.

- diffutils-2.7.tar.gz
- rcs-5.7.tar.gz

Peter Parnes

Building CVS

You'll need a C compiler, Yacc or Bison, make, etc. CVS uses GNU Autoconf. You should be able to ungzip and untar the archive and simply type:

```
$ ./configure
$ make
$ make install
```

Detailed information is in the `INSTALL` file in the distribution. Consult your local UNIX Guru if you have difficulties. Build instructions for Mac, Windows, and VMS are included in the package as well.

Peter Parnes

The Repository

CVS uses a central repository to store all the packages (aka modules) that are under it's control. While multiple repositories are possible, it is usually easiest to have just one. The environment variable `CVSROOT` should point to the repository. If your CVSROOT is /work/cvsroot, you can set this in a csh variant shell with:

```
% setenv CVSROOT /work/cvsroot
```

or if you use a ksh variant:

```
$ export CVSROOT=/work/cvsroot
```

Creating the Repository

The first time you use CVS, you should create the repository with:

```
$ cvs init
```

If there is already an existing repository in `CVSROOT` it will not be modified.

Peter Parnes

CVS Directory Structure

```
/work
 |
 +--cvsroot
 |    |
 |    +--CVSROOT
      |       (administrative files)
      |
      +--xmosaic
      |
      |
      +--fmcc
      |
```

Important Notes

It is very important that you do not modify files in the repository unless you know what you're doing -- you can break CVS by messing up the repository.

CVS will create a directory in the repository for each module you create. CVS also creates the CVSROOT directory for its own use.

Peter Parnes

Versions and Revisions

CVS differentiates Revisions (which are a per-file concept) from Versions (which are a per-package concept).

Revisions

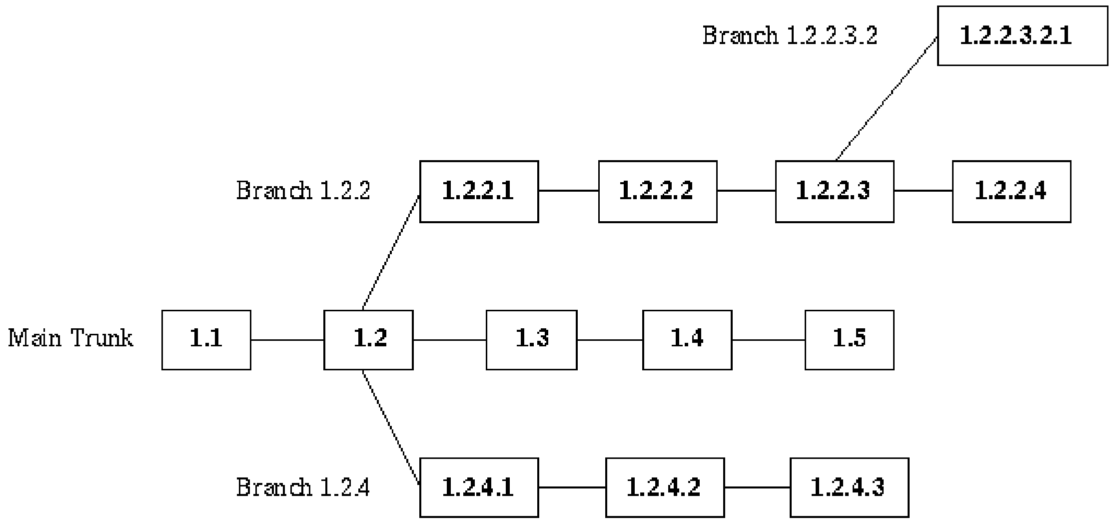
- Each version of a file has a unique *revision number*
- A revision number always has an even number of dot-separated integers
- 1.1 is the default first revision of a file
- Successive revision numbers increase the rightmost number by one

Branches

- Each branch has a unique *branch number*
- A branch number is created by appending a number to the revision number of the revision the branch is forked from
- More than one branch may fork from on revision

Peter Parnes

Revision Example



Peter Parnes

Creating a New Module

In CVS parlance, a "module" is a directory hierarchy...

Putting an existing project into CVS

Let's say you've been working on the Frobozz Magic C Compiler (fmcc) and you want to put it under version control. If it lives in a directory called `fmcc`, you could:

```
$ cd fmcc
$ cvs import -m "imported sources" fmcc fro
```

CVS will create a `fmcc` directory in the CVSROOT and import the current directory and it's children into revision control. The last two arguments are the vendor and release tags. They don't have to have any special meaning.

You can make sure that this worked by saving the old directory and checking the project out of the CVS repository:

```
$ cd ..
$ mv fwcc fwcc.orig
$ cvs checkout fwcc
$ cd fwcc ; ls
```

Peter Parnes

Starting a fresh project

```
$ mkdir hello
$ cd hello
$ cvs import -m "new project started" hello
```

You now have a module called "hello" with no files in it.

Notes about checked out projects

There will be a CVS directory in each directory of a checked out project. CVS uses files in the CVS directory to maintain state.

Do not modify or delete these files or you will no longer be able to use that checked out copy.

Peter Parnes

Adding / Editing / Committing

Getting a copy of a module out of the repository:

$ **cvs checkout** *module* (checkout = co)

This will create a directory called *module* and check the module of the same name out into it. If you want to check out into a different directory use (note that the -d must be **between** the command and the module name):

$ **cvs checkout** -d *directory module*

Adding a file to a checked out module:

$ **cvs add** *file*

file will be added at the next commit.

Removing a file from a module:

$ **cvs remove** *file*

file will be removed at the next commit. `cvs remove` will not work unless you delete *file* first.

Peter Parnes

Updating

Bring your copy of a module up to date:

```
$ cvs update
```

Update will compare files in a checked out module with their counterparts in the repository. All files examined will be shown with a one letter code preceding them.

- **U** - update - the file changed in the repository but not in your checkout. Your checkout has been updated
- **M** - merge - your checked out file has changed, but it can be merged into the repository copy with no errors
- **C** - conflict - your checked out file has changed and CANNOT be merged into the repository.
- **?** - unknown - this file is not under the control of CVS and was ignored.

Committing your changes back into the repository:

```
$ cvs commit -m "log message"
```

CVS will not let you commit if you have non-up-to-date files. If you omit the log message (-m option), an editor will be invoked to allow you to enter a log message.

Peter Parnes

Dealing with Conflicts

If two developers change a file in such a way that the changes cannot be merged, a conflict will occur during update. Let's say there's a file `hello.c`:

```
main() {
        puts("Hello World");
}
```

And one developer changes the puts() to a printf() and commits it:

```
main() {
        printf("Hello World");
}
```

```
$ cvs update
cvs update: Updating .
M hello.c
$ cvs commit -m "printf is better"
cvs commit: Examining .
cvs commit: Committing .
Checking in hello.c;
/work/cvsroot/hello/hello.c,v  <--  hello.c
new revision: 1.2; previous revision: 1.1
done
```

Peter Parnes

More Dealing with Conflicts

Meanwhile, elsewhere, another developer adds some punctuation:

```
main() {
        puts("Hello, World!");
}
```

There will be a problem during update:

```
$ cvs update
cvs update: Updating .
RCS file: /work/cvsroot/hello/hello.c,v
retrieving revision 1.1.1.1
retrieving revision 1.2
Merging differences between 1.1.1.1 and 1.2
rcsmerge: warning: conflicts during merge
cvs update: conflicts found in hello.c
C hello.c
```

Peter Parnes

Even More Dealing with Conflicts

```
main() {
<<<<<<< hello.c
        puts("Hello, World!");
=======
        printf("Hello World");
>>>>>>> 1.2
}
```

The <<<<<<< and >>>>>>>'s mark the area where changes overlap. This must be resolved by hand. Conflicts will only occur if two (or more) developers are changing the same section of the same file. Usually this is a result of poor communication and can be easily avoided.

Peter Parnes

Remote Access

CVS (1.7 or greater) has a built-in client server mode. Supported modes in version 1.9 include rsh, pserver, or kerberos authenticated. We'll cover pserver and rsh here.

Setting up the server

Create a CVS repository if there is not already one.

Add this to `/etc/services`:

```
cvs     2401/tcp         # remote cvs server
```

Add this to `/etc/inetd.conf`:

```
cvs     stream  tcp   nowait  root   /path/t
```

Restarted inetd. Optionally, you may create a `passwd` file in the CVSROOT that contains `name:cryptedpasswd` pairs, one per line. This allows users to avoid using their normal login passwords with CVS.

Peter Parnes

More Remote Access with PServer

concerned

Note for the paranoid:

You'll probably want to run CVS under tcpwrappers. The CVS server needs to run as root in order to change to the appropriate UID after authentication. The server accepts a connection, checks the authentication, and either exits or changes the UID and continues.

Setting up the client

Change your CVSROOT to something like:

```
:pserver:joeuser@foo.bar.com:/path/to/cvsro
```

Before you can issues any CVS commands, you must:

```
$ cvs login
```

and enter your password. CVS stores your crypt()'d password in .cvspass in your home directory for use in future connections. The paranoid may wish to delete this between sessions. Once you have logged in, you may issue cvs commands just the same as you would with a local repository.

Peter Parnes

Remote Access using RSH

In some environments it is more convenient to use 'rsh' for remote access instead. This can be done by changing CVSROOT to something like:

```
:ext:basil.cdt.luth.se:/proj/mates/mates/CV
```

and then running the commands as usual.

The security might want to add set CVS_RSH to

```
ssh
```

which makes CVS use SSH and encryption for remote access.

Peter Parnes

Logging / Annotations / Other Useful Stuff

Getting information about a file:

`$` **`cvs status`** *`file`*

EXAMPLE

Examining the logs on a file:

`$` **`cvs log`** *`file`*

EXAMPLE

Annotations - who did what

`$` **`cvs annotate`** *`file`*

EXAMPLE

Peter Parnes

Keyword expansion fun

Certain keywords will be automatically expanded in source files:

- **Id** -- filename, revision, date, author, state
- **$Date$** -- date and time the file was checked in
- **$Author$** -- author of the file
- **$Revision$** -- revision of the file
- **Log** -- causes log entries to be appended

An example:

```
/* $Id: hello.c,v 1.3 1996/11/15 22:14:05 s
 * $Date: 1996/11/15 22:14:05 $
 * $Author: swetland $
 * $Revision: 1.3 $
 */

main() {
        printf("Hello World");
}
```

Peter Parnes

More Keyword Expansion Fun

If you embed this strings in a string constant, the rcs `ident` command can be used to get the version of an executable. Example, hello.c:

```
static char *id =
        "$Id: hello.c,v 1.3 1996/11/15 22:1
main() {
        printf("Hello, World!");
}
```

Try typing:

```
$ ident hello
hello:
     $Id: hello.c,v 1.3 1996/11/15 22:14:05
```

Peter Parnes

Tagging / Releases

Each file in cvs has a version number.
However, it is often useful to give a symbolic name to all the files at once.

```
$ cvs tag version_1_0
```

will attach the tag "version_1_0" to all the files in the current directory.

```
file1   file2   file3   file4
                1.1
                1.2
                1.3     1.1
        1.1     1.4     1.2
1.1 --- 1.2 --- 1.5 --- 1.3 <-- version_1_0
1.2     1.3     1.6
1.3
```

Checking out a specific version

```
$ cvs checkout -r version_1_0 hello
```

will checkout a copy of the 1.0 version of the hello module.

Peter Parnes

Branches and Merging

WARNING! Merging is complicated and potentially messy. Proceed with manual in hand.

Creating a new branch

$ **cvs rtag -b -r** *branchname module*

That branch may now be checked out with

$ **cvs checkout -r** *branchname module*

Any changes in the working copy of a branch will not affect the main tree.

Merging a branch into the current trunk

$ **cvs update -j** *branchname*

Will join *branchname* into the current checkout

Merging differences

$ **cvs update -j** *revision* **-j** *revision*

will merge the **difference** between the two revisions into the current working copy.

Peter Parnes

Access Methods

- command-line
- Emacs PCL-CVS
- TkCVS

Peter Parnes

How to set up PCL-CVS

Add the following to your .emacs:

```
(autoload 'cvs-update "pcl-cvs"
              "Run a 'cvs update' in the curren
output to a *cvs* buffer and run cvs-mode o
If optional prefix argument LOCAL is non-ni
              t)

(autoload 'cvs-update-other-window "pcl-cvs
             "Run a 'cvs update' in the current
output to a *cvs* buffer, display it in the
cvs-mode on it.

If optional prefix argument LOCAL is non-ni
              t)
```

And bind it to some convenient key such as F4:

```
(global-set-key [f4] 'cvs-update)
```

or start it manually using:

```
M-x cvs-update
```

Peter Parnes