

"Code management in multi-programmer environments."

Author: D. Hatton (DESY/Hamburg)

Table of Contents:

1. [Overview](#)
 2. [Version control. What is it and why?](#)
 3. [CVS \(Concurrent Versions System.\)](#)
 4. [Seperate development paths.](#)
 - ◆ Branching and merging...
 5. [Some communication issues...](#)
 - ◆ About source conflicts.
 - ◆ Automatic notifications.
 6. [cvsweb...](#)
 - ◆ A tour of the CVS web interface.
-

[table of contents](#)[next topic](#)

1. Overview.

- Software project version control.

Without some reliable mechanism, the job of maintaining and tracking the source versions underlying a release becomes unnecessarily error prone and convoluted, even on a small one person project. Version tracking on a project involving several developers becomes practically impossible.

- What version control tools are out there?

Microsoft Visual Source Safe and PVCS both of which, both of which are commercial.

For this discussion, version management issues will be looked at in terms of CVS (Concurrent Versions System).

It's certainly 'good enough' for most purposes and almost ubiquitous. As well as being free.

- Issues of multi-programmer source management using CVS.
-

- Some additional tools for source navigation, bug tracking and automatic build maintainance.
-

[table of contents](#)[next topic](#)



2. Version control. What is it and why?

What is it?

Version control, essentially this is about having some means of being able to reproduce a set of source files that satisfied some set of conditions in the past. e.g. Conditions such as, an executable could be built from them which passed certain test conditions... or perhaps they exhibit some problem feature and they can be used to reproduce it. Of course we could apply this to other contexts such as document control, the principal is the same.

Why use a tool for version control?

Why use it? This should be clear, but very often source versions are maintained in a manual way often in an ad-hoc manner e.g. Someone develops some library, each week starting a new directory in order to 'backup' the last changes from the previous week...

```
lrwxrwxrwx 1 fred users 9 Feb 16 13:12 current -> lib-week3
drwxr-xr-x 2 fred users 4096 Feb 16 13:11 lib-week1
drwxr-xr-x 2 fred users 4096 Feb 16 13:12 lib-week2
drwxr-xr-x 2 fred users 4096 Feb 16 13:12 lib-week3
```

For large project, this can waste a lot of disk space for one thing, this is really is nothing more than a manual backup mechanism. User fred will have to create a directory for each build that he is satisfied with. There will be unnecessary trouble in tracking changes in case of a problem later. If indeed, it is possible to track the changes at all... the backups are mostly arbitrarily based on some date. If several developers are involved and there are several modules, it would be hard if not

impossible to avoid frequent errors ...

Another approach is often to keep executable versions, these tend to become dissociated from the source code meaning that if there's a problem, very often the source code simply cannot be determined.

- So clearly even at the most basic level, some automated version control tool will save a great deal of unnecessary effort and trouble.



[table of contents](#)



[next topic](#)

[table of contents](#)[next topic](#)

3. CVS (Concurrent Versions System.)

What is CVS?

CVS is an open source version control tool with several features that we'll be exploring. It stores project files as a set of sub-directories beneath the repository root. Basic features:

- Maintains a central repository holding file revision changes.
 - Allowing symbolic tags to be associated with any collection of file revisions.
 - Parallel development using branches.
-

Applying cvs ...

We'll introduce it's basic deployment by running through the steps necessary, to get the previous 'weekly backup' example under cvs...

- Creating the repository.

CVS relies on a central directory where it stores it's administrative files as well as the target revision file information, this is known as the repository.

The user fred would create say, /home/fred/project/CVS, setting the environment variable CVSROOT=/home/fred/project/CVS, this then tells CVS where the repository is for all subsequent operations. Next, the command 'cvs init' sets up the repository control files and it is ready for use.

- Importing files.

In the directory containing the files, running : cvs import PROJECT vendor-tag initial_release_tag; This stores the initial revisions in the repository. The files can then be checked out using cvs checkout PROJECT, cvs creates a working directory PROJECT/ containing the source files. These source files can be worked on without affecting the anybody else or the central repository. A set of changes can be 'commit'ed to the repository as follows : cvs commit files...

- Tracking versions using tags.

In cvs there is the facility to give symbolic names to sets of files with possibly several different revision numbers. It's called tagging... It's used to tie together a set of file revisions that constitute some meaningful release or condition... In order to issue such a tag 'cvs tag SYMBOLIC_TAG module' is used. Where SYMBOLIC_TAG is an arbitrary name, and module is the target repository directory.

[table of contents](#)[next topic](#)

[table of contents](#)[next topic](#)

4. Seperate development paths.

Branching

An important feature that CVS has is the ability to allow parallel development paths on the same module directory. Very often a user will want to do some developmental work on a module that will introduce significant changes, while needing to maintain some earlier stable release. The user creates a 'branch' at the point in question... `cvstag -b BRANCH_TAG SYMBOLIC_TAG module_name`. The user can then maintain the changes on the branch by '`cvsc checkout -rBRANCH_TAG`'. At a later stage a branch may be merged back on to the main trunk by '`cvsup update -jBRANCH_TAG`' from the working directory containing the latest revisions of the main trunk. However a branch may be maintained without merging permanently if required.

[table of contents](#)[next topic](#)

[table of contents](#)[next topic](#)

5. Some communication issues.

Conflicts

Source conflicts occur when two or more users attempt to change the same line of code. Conflicts can also occur as the result of a single users work. How does CVS handle this? An example...

Automatic notifications

Support for receiving notifications based on cvs operations such as tagging...

[table of contents](#)[next topic](#)

[table of contents](#)[next topic](#)

6. CVSWEB.

 **Introduction to the cvsweb interface...**

[cvsweb](#)

[table of contents](#)[next topic](#)



[table of contents](#)

7. Summary and concluding remarks.



Closing remarks.

Bug tracking, source navigation.



[table of contents](#)



[next topic](#)
