



# **Version Management Systems for Code Development: An Introduction to CVS**

Adrian Wander

Computational Science and Engineering Department,  
CLRC Daresbury Laboratory, Warrington, Cheshire, WA4 4AD.

*Revision: 1.0*

## **Abstract**

CVS is a code development tool that, at first site, appears to be a bit of a waste of time. However, it is probably the single most useful tool in software development. Once you start using it you will quickly realise how useful it is, and will wonder how you ever managed without it!

**This is a Technical Note of the UKHEC Collaboration.**

Report available from <http://www.ukhec.ac.uk/publications/notes/versions.pdf>

**© UKHEC 2002.**

Neither the UKHEC Collaboration nor its members separately accept any responsibility for loss or damage arising from the use of information contained in any of their reports or in any communication about their tests or investigations.

Version control systems are tools designed to aid program development. They meet an important need in development activities by automating many of the tasks involved. These tasks include maintaining all versions of a program in a recoverable format, merging parallel development tracks into a single version, maintaining version logs detailing changes, and enabling several developers to simultaneously work on different sections of the code.

The principal difference between CVS and RCS is the way in which access to the code by multiple programmers is handled. Within RCS a file being worked on by a programmer is locked preventing other programmers from accessing the same file. Within CVS multiple programmers can have access to the code at the same time!

In both systems, the code is maintained in a single repository. The repository is created ( `cvs import` ), and the original source code is committed. Subsequently, files are checked out of the repository ( `cvs checkout` ), modified and checked back in ( `cvs commit` ). Conflicts created by several programmers editing the same file are flagged at check in time. Each check in creates a new version of the code. (The repository does not store each version. It stores the information needed to generate each version from the original source code - this prevents the repository growing too large). As the code is checked in, the software prompts the users to enter a short description of the changes made to the file, which is stored as a log. Consequently, previous versions of the code are always accessible, and a history detailing the changes in the file is automatically created.

Although CVS is designed to allow multiple users to access the same code I find that the tools available within CVS are more powerful than those within RCS and hence I use it even for projects on which only I am working.

Typical examples of the problems that can easily be resolved using CVS are:

- A bug has been introduced by a recent edit - what changes were made? This can be checked easily by diffing the new version and old version of the code ( `cvs diff` ).
- I have just accidentally deleted an important file - I can recover a version from the archive ( `cvs update` ).
- I am working on a version of the code while someone else is modifying another section of the code to fix a bug. I can update my files easily, and potential conflicts between the two sets of editing will be flagged. If no conflicts are present both versions will be merged automatically ( `cvs update` ).
- I have worked on a number of files and can't remember which ones. A status command is available letting me know which files have been modified ( `cvs status` ).
- I modified a file some time ago and can't remember why? A history command is available showing all accesses to the repository together with a log command to examine the comments I made when checking in the file ( `cvs log / cvs history` ).
- I have released a version of the code. I am now generating new functionality while also supporting the released version. I need to fix bugs in the released version and merge those changes into the current version as well. This is an

obvious example of a branch. One branch of the development tree deals with new developments, the other deals with bug fixes. At any point the branches can be merged.

- I have reached a point in the development where I have a reasonably stable version of a code. This can be tagged for ease of later access (cvs rtag).

Another useful feature is that code being worked on can be stored on a secure, backed up computer system. Code is then checked out onto local workstations (which are often not backed up!), modified and checked back into the secure central location. CVS can also be used to check out code remotely. We access the CRYSTAL repository from the CRAY-T3E in Manchester during development via a secure shell server (ssh). The developers in Turin also access the code in the same manner.

Hence, the system has tools to aid distributed developers and is also an invaluable aid for the solo code developer.

The CVS code may be obtained from

<http://www.cvshome.org/>

This site contains a number of precompiled versions for a range of operating systems, as well as the source code in case you need to build it yourself.

Further introductory material is available from

[http://www.cvshome.org/new\\_users.html](http://www.cvshome.org/new_users.html)

<http://www.loria.fr/~molli/cvs-index.html>