CVS – Basics for decision

René Scholz – Voland@jena.thur.de

December 8, 2000

Abstract

This document describes basics and features of the control revision system CVS (Concurrent Versions System).

1 In general: How CVS works

Taken from the page www.cs.mu.oz.au/~trd/www-free/revision.html:

CVS works by keeping a single copy of all your source code (and data files, test cases – anything you want, really) along with enough information to re-create any version that you have told CVS to keep. Essentially CVS creates a database of the change history of your code.
CVS is built on top of the older revision control system "RCS" (Revision Control System). RCS is good for keeping a few files under revision control in a single directory, but CVS is much better at handling many files and multiple directories. CVS still isn't perfect, but it's a very useful tool.
CVS lets every developer create a workspace where they can do anything they want to. You can work on the changes in that workspace in isolation from other changes. When you feel ready, you can update the source in your own workspace to integrate any changes that may have happened while you were isolated. When the changes in your workspace are complete, you can add them to the database. If you don't like them, you can just delete them. At any time, you can ask CVS to tell you what changes you have made.
If you're working on two different changes at once, you can put each of them in a different workspace. If you're adding a new feature, and a bug needs to be fixed, it's easy to just create a new workspace just to fix the bug.
It will let developers work on some changes in this workspace, update the source in the workspace with any changes other people have made in the meantime, and even create workspaces with older versions of the code.

The ability to create older versions means that you can apply other people's patches to the correct version, then update your workspace to the current version, letting CVS do most of the work for integrating changes (CVS will merge changes together - most of the time it is completely automatic).
You can also set CVS up as a client/server system over the Internet (or any TCP/IP network), allowing known users to upload and download changes directly, or even allow anonymous CVS access so anyone can download completely up-to-date source.

2 CVS Overview

- Basic version control functionality. That is, CVS maintains a history of all changes made to each directory tree it manages. Using this history, CVS can recreate past states of the tree, or show a developer when, why, and by whom a given change was made. If you are familiar with RCS or SCCS, the difference is that CVS operates on entire directory trees, not just single files.
- CVS supports branches, which allow several lines of development to occur in parallel, and provides mechanisms for merging branches back together when desired. CVS can tag the state of the directory tree at a given point and recreate that state. CVS can display the differences between tags or revisions in the standard diff formats.
- Can run scripts which you supply to log CVS operations or enforce site-specific policies.
- Client/server CVS enables developers scattered by geography or slow modems to function as a single team. The version history is stored on a single central server and the client machines have a copy of all the files that the developers are working on. Therefore, the network between the client and the server must be up to perform CVS operations (such as checkins or updates) but need not be up to edit or manipulate the current versions of the files. Clients can perform all the same operations which are available locally.
- **Unreserved checkouts, allowing more than one developer to work on the same files at the same time.**
- CVS provides a flexible modules database that provides a symbolic mapping of names to components of a larger software distribution. It applies names to collections of directories and files. A single command can manipulate the entire collection.

3 Properties & Features of CVS

- CVS is Open Source software.
- CVS is widely used.

- CVS-Clients run under Unix systems and also WindowsNT/95 (and other platforms), the CVS-Server runs under Unix.
- Can be run local or as client/server with a network.
- There are a number of different clients/GUI's available:

  jCVS, tkCVS, WinCVS, XEmacs (basic and even better with the package `PCL-CVS`), gCVS, Pharmacy, MacCVS etc.

  I suggest: jCVS (because it's Java based) or WinCVS for Windows platforms.
- Some Development Environments are also supported:

  Developer Studio, CodeWarrior, Visual Cafe, WipeOut.
- There is also a web interface (`cvsweb`) to browse the actual sources of a project.
- Security: CVS can use `rsh`, `ssh` or `kerberos` for transport mechanismen.
- There is an apache modul `mod_cvs`. With this module, when apache goes to fetch a page, it will first check whether it needs to update that page from CVS.

4 CVS Datasheet

Here is the complete datasheet from `www.cvshome.org/cyclic/cyclic-pages/CVS-sheet.html`, which describes CVS in detail:

The Concurrent Versions System (CVS) provides network-transparent source control for groups of developers.

- CVS maintains a history of all changes made to each directory tree it manages. Using this history, CVS can recreate past states of the tree, or show a developer when, why, and by whom a given change was made. CVS supports branches, which help manage long-term changes and bug-fix releases.
- CVS provides hooks to support process control and change control.
- CVS provides reliable access to its directory trees from remote hosts, using Internet protocols. Developers at remote sites can perform all the same operations available locally. Access can be authenticated using the Kerberos network security system.

- CVS supports parallel development, allowing more than one developer to work on the same sources at the same time.

Version Control

- Each developer uses cvs checkout to create her own copy of the source tree from the CVS repository. The command can operate on a directory tree, on a single file, or on a module; a module groups several files or directories into one entity, which can be operated on as a unit. One defines modules by editing the 'modules' file.
- The developer modifies, compiles, and tests the code in her copy of the source tree (called a working directory) with whatever editors and tools she chooses – Emacs, make, etags, etc. She uses cvs add and cvs remove to add and remove files.
- When the changes are complete, the developer uses cvs commit to merge her changes back into the repository. This makes her changes available to other developers.
- At any point, the developer may use cvs update to merge changes committed by others into her working directory. If there are uncommitted changes to files in her working directory, CVS prints a message and attempts to merge the changes from the repository with her changes in the working directory. If the merge fails, CVS indicates a conflict, which she resolves manually with a text editor.
- The developer can show the differences between two revisions with `cvs diff`; show the log of changes to a particular file with `cvs log`; show the history of each line of a file with `cvs annotate`; and show who has used cvs checkout, cvs tag, and several other CVS commands, using `cvs history`.
- CVS supports watches, allowing developers to request notification when someone begins editing a file, or obtain a list of developers currently working on a file.
- The user can record the state of the repository at a particular point with the `cvs tag` command, and can then use that tag as an argument to most CVS commands, for example to retrieve the files as of the tagged point.
- The developer can create a new development branch with `cvs tag -b`, and manipulate branches with `cvs update -r` and `cvs checkout -r`. Subsequent operations in that working directory apply to that branch. To return to the main branch, the developer can use `cvs update -A`. The `cvs update -j` command merges changes made on another branch into the working directory.
- The existence or nonexistence of a file is itself version controlled, so that files can exist on some but not all branches and users can reproduce the state of the files at any given point in time.

- The developer can mark a file as binary, which prohibits merging and line terminator conversions, using `cvs admin -kb`.

Process Control

CVS gives project managers fine control over the development process.

- The '`commitinfo`' configuration file tells CVS how to screen changes. When a developer tries to commit a change, CVS will run a script to determine whether the change is acceptable.
- The '`loginfo`' configuration file can tell CVS to run a script after a developer commits a change. For example, CVS can send E-mail to other developers, notifying them of the change.
- The '`rcsinfo`' configuration file can specify a template for log messages, presented to developers when they commit a change. For example, the template could prompt for a bug number.
- The '`editinfo`' configuration file tells CVS how to screen developers' log messages. For example, the script could ensure that a bug number mentioned in the log message is valid.

Change Control

The '`rcsinfo`' and '`editinfo`' configuration files (described in the Process Control section above) allow CVS to interface to external bug tracking systems.

Network Transparency

CVS provides reliable repository access to remote hosts using Internet protocols, facilitating collaboration with distant employees and contractors.

- CVS operations can be performed over the network. A developer on a remote host can check out a local copy of the sources, make changes, update her local copy with changes made by others, and commit her changes back into the repository.
- Remote operation is efficient, transmitting only those files which have changed. When appropriate, CVS transmits patches to files and verifies the results, rather than sending entire files. CVS can compress the text it transmits.
- Remote operation is reliable. CVS holds no internal locks while waiting for communications to complete, so network troubles will not disrupt others' access to the repository. It uses reliable transport mechanisms, not NFS, making it well-adapted for use over wide-area networks.

- Remote operation is authenticated. CVS can use the industry standard Kerberos protocols to verify the identity of the remote user. Kerberos is much more secure than the source-address authentication provided by the ordinary rlogin and rsh protocols. CVS can also work with ssh, a secure replacement for rsh, or use straight password authentication.
- Remote operation supports portable computers. While a developer's portable computer is connected to the network, she can update her working copy of the source code. While disconnected, she can develop her working copy. The next time she connects to the network, she can commit her changes.

Supported Platforms

The CVS client and server run on most Unix variants (although the more exotic ones might require minor porting effort). The CVS client also runs on Windows NT, OS/2 and VMS. Other ports are in progress, but the progress of those ports may depend on funding and/or volunteer effort.
CPU and memory requirements are modest – any client with enough capacity to run the operating system in question should have little trouble and a server with 32M of memory or even less can handle a fairly large source tree with a fair amount of activity. To estimate disk space requirements, if you are importing RCS files from another system, the size of those files is the approximate initial size of your repository, or if you are starting without any version history, a rule of thumb is to allow for the server approximately three times the size of the code to be under CVS for the repository (you will eventually outgrow this, but not for a while). On the machines on which the developers will be working, you'll want disk space for approximately one working directory for each developer (either the entire tree or a portion of it, depending on what each developer uses).

5 Misc

- The local (Jena) guru Lutz Donnerhacke suggests using `ssh` instead of `kerberos`, because `kerberos` is too old and uses symmetric keys, which are valid up to 8 hours.
  `ssh` uses asymmetric keypairs.
- There are some ways to handle reserved checkouts (Microsofts VSS works this way):
  - One can use the optional *watch* features in CVS. The watch features allow developers to request notification when someone begins editing a file, or obtain a list of developers currently working on a file.

  - Getting reserved checkouts via the `cvs admin -l` command (consult the manual; there is a trick or two involved in setting it up)

  As consequence the developers should do:

  - Update their workspace more often to reduce conflicts.
  - Check if the modified source is compileable before doing the update.

  Of course conflicts can be resolved (by hand).

- There is a `cvsadmin` user for the CVS, which has access to some meta data (`CVSROOT`).
- With a CVS on a single server machine backups are easy to be made.
- A compression can be used, but is often useless because the transported changes (diffs) are small.
- There is (was?) commercial support for CVS from Cyclic Software, see `http://www.cvshome.org/cyclic/cyclic-pages/support.html`.

6 Negatives

Here are some negative points about using CVS (particularly compared to VSS):

- Users/Developers must perhaps switch over from other source control systems like VSS.
- Former VSS Users/Developers must change their point of view: They are now not the only one editing a file!
- To convert a VSS-History would be a lot of work to do.
- There may be problems with very big files and binary files.

7 Experiences

The following text describes some experiences with CVS from Jim Blandy (a former emacs developer and founder of Cyclic Software):

Unlike RCS, CVS uses a copy-modify-merge development model. Under this model, each developer has her own working copy of each source file, which she may edit at any time. When her changes are complete, she performs an update operation, which merges into her working files any changes made by other developers. She then performs a commit operation, to publish the merged sources to the group. CVS flags any textual conflicts between the developer's own changes and those made by others, and requires her to resolve the conflicts before committing her changes.

In contrast with RCS, CVS performs no locking on source files; any developer can edit any file at any time. Instead, synchronization occurs via the merge/commit process. The question now becomes, how often do conflicts occur, and how difficult are they to resolve? In our experience, conflicts occur rarely. During the period Karl and I used CVS to manage the gene editor sources, we found one conflict roughly every two months. All our conflicts were straightforward to resolve. I believe conflict frequency depends partially on how cleanly the team has divided the project.

The rarity of serious conflicts may be surprising, until one realizes that they occur only when two developers disagree on the proper design for a given section of code; such a disagreement suggests that the team has not been communicating properly in the first place. In order to collaborate under any source management regimen, developers must agree on the general design of the system; given this agreement, overlapping changes are usually straightforward to merge.

Of course, CVS can only detect textual conflicts between changes, not semantic conflicts. If one developer changes the semantics of a function, and another developer adds a new call to that function expecting the old semantics, CVS alone will not warn them of this situation. However, one can configure CVS to require the program under development to pass a test suite before committing any changes. This warns the developer of any semantic conflicts that are visible to the test suite. Since our gene editor project was relatively small, Karl and I simply agreed to test changes manually before committing them.

Our experience with CVS has generally been quite positive. The ancestors of CVS have been in widespread use in the Unix community for several years now. CVS itself performs reliably for us; the internal design is relatively clean, so the bugs we have encountered have been straightforward to fix. Because Karl and I collaborated across a wide-area network, we relied on CVS's network transparency. It is indeed transparent; merging worked just as well remotely as locally. In our view, its most serious shortcoming was the lack of serious support and development; we intend Cyclic Software to fill that need.

8 Some big projects using CVS

Here are some bigger projects which are using CVS:

- SunOS 4.0 kernel source
- Mozilla
- GNU Emacs, XEmacs
- OpenBSD, FreeBSD, NetBSD
- EGCS
- The GIMP
- GNOME and Enlightenment
- Wine
- Apache
- PHP
- Python

9 Referencies

- CVShome.org:
  `www.cvshome.org/`
- Cyclic Software (old):
  `www.cvshome.org/cyclic/`
- *Version Management with CVS* by Per Cederqvist:
  `www.loria.fr/cgi-bin/molli/wilma.cgi/doc.847210383.html`

- *Introduction to CVS* by Jim Blandy (a tutorial):
  `www.cvshome.org/docs/blandy.html`
- ***The CVS Book*** by Karl Fogel:
  `cvsbook.red-bean.com/`