Concurrent Versioning System

Introduction to CVS

Tzahi Fadida - TzahiFadida@MyRealBox.com

Special thanks for reviewing and coauthoring to Orna Agmon

Agenda

- CVS - why do I need it?
- CVS - what is it?
- CVS - what is it NOT?
- CVS - ok I am convinced, what now? - Internals
- CVS - Getting more into it. - Philosophy
- CVS - OK Show me the money!!!! - Examples
- CVS - where can I find out more?

CVS - why do I need it? - Ver Control

- Ever worked on a project on your local computer and the H.D crashed?
- Ever worked on a file, saved, then changed your mind and worked hours to restore the mistake?
- Ever tried to find out what change you did a month ago?
- Does your boss breathe down your neck to increase productivity?

CVS - why do I need it? - team work

- Ever needed to work with a team?
- Ever had to work on the same file with a team mate and needed to wait until he finished?
- Ever had to merge a whole branch of a project with changes that another team member had just made?
- Ever wanted to trace your progress or others'?

CVS - A solution

- Enters CVS!
- CVS will answer all of the above mentioned problems and more.
- You only need to know a small part of CVS’s features in order to get started.

CVS - what is it?

- CVS is a source code management system, based on a client- server model.
  - Retains projects in a centralized manner.
  - All changes in every file are traceable.
  - Allows to retrieve specific project revisions.
  - Allows access in a distributed manner.

CVS - what is it NOT?

- CVS is not really user friendly- this is what graphical front ends are for.
- CVS is not an excuse not to backup.
- CVS does not come instead of talking to fellow developers and deciding on policies and future directions.
- CVS doesn’t read minds, you have to tell it what to do.
- CVS is NOT proprietary! I.e. you don’t have to pay. In fact, its GPL’d.

A CVS Client

- A simple text client is usually installed on Linux.
- For Unix, Cervisia and many other clients, since CVS was originally written for unix.
- WinCVS Windows users.
- jCVS II for java users.

A CVS Server

- Existing internet servers, such as http://www.sourceforge.org or http://www.berlios.de.
- Existing servers inside your organization.
- Download a server for Linux from http://www.cvshome.org.
- Download a server ported for Windows http://www.cvsnt.org/wiki/.

The place where the server keeps its information is called a repository. It is recommended to allow at least 3 times the current size of the projects for the CVS repository.

CVS - How does it do that?

- CVS lower layer is actualy RCS - Revision Control System. CVS allows you to manage entire software projects ontop of RCS.
- RCS
  - RCS automates the storing, retrieval, logging, identification, and merging of revisions.
  - RCS is useful for text that is revised frequently, including source code, programs, documentation, graphics, papers, and form letters.
- CVS also supports storage of binary files.

Why shouldn’t I use RCS then?

- RCS has a strong locking, i.e. no one can work with it except the locker. CVS has weak locking: multi-checkout and concurrent editing is possible.
- CVS supports various types of actions taken when a reserved file is touched (for example, sending an email)
- CVS has the capability to work decentralized, while RCS is centralized and works locally.
- CVS handles complex projects better.
- RCS has a smaller command set and is simpler to use. CVS has a steep learning curve.

Graphic CVS Clients

For large projects it is sometimes better to use a graphical cvs client, for the following advantages:

- Graphical representation of a file increments.
- Quick access to macros.
- Visual repesntation of changes to files.
- Easier to remember rarely used features.

See documentation for the client of your choice, in order to set it up and connect it to the server.

CVS Basic Work Methods

Importing, checking out, and commiting.

- When you have a project outside CVS, you import your project (a directory with sources) to the cvs (using the cvs client or gui).
- When you have a project inside CVS, and you want a local copy, you check out a module (your project name) that was created on the cvs to a working directroy.
- When you finish editing something, (and you have at least verified that it compiles) you commit (check in) your changes to the server using your client, with a comment on what was changed.

CVS Phylosophy

- A good practice is to commit your changes when you can write a significant change in the comment of the commit action and that the change will not break what was previously working. This way, your team will be able to be more productive and release more versions as the projects develop. In addition, the files can be compiled and debugged independently by other parties.
- It might be beneficial to create a ChangeLog file, in addition to the comments, summarizing project-wise the changes that were made. The comments here will document the files that were actually changed. Please note that there are macros that can automatically generate a ChangeLog.

CVS - Simultanous Check Out

- NO exclusive checkouts like RCS's "co -l".
- More then one person can checkout, change a file and commit changes at the same time.
- This is accomplished by a conflict mechanism that allows you to merge your changes to previous changes entered, even if the changes were made against different versions.

CVS - Conflicts

- A conflict can occur if two or more cvs users checked out a file and changed it. Let’s designate this file with version 1.1
- The first person changed it and committed the changes to the server. Now the file is designated 1.2 at the server.
- The second person now also wants to commit his or her version 1.2 of the file, not knowing yet it was already committed.

CVS - Solving Conflicts

- If the changes are far from each other in the file, the second person gets a “Merge” message.
- If the changes were done on the same lines (or close lines) the second person gets a “Conflict” message from the server, indicating that the user version (1.2) of the file needs to be merged with the file on the server (also version 1.2).
- The second user uses some good merge software and (hopefully) in a few seconds commits version 1.3 of the file to the server.

CVS Using the Command line Client

- The command line tool can be used either locally or over a communication protocol.
- Note that CVS does not allow working as root.

CVS setting ENV variables

- Set the environment variable CVSROOT to indicate the repository.

```
[devGuy@LinuxRules devGuy]$ CVSROOT=:pserver:tzahi@127.0.0.1:/home/cvs/master
[devGuy@LinuxRules devGuy]$ export CVSROOT
If its not a csh descendant, use - setenv CVSROOT
:pserver:tzahi@127.0.0.1:/home/cvs/master
[devGuy@LinuxRules: /pop2sms]$ cvs login
Logging in to :pserver:tzahi@127.0.0.1:2401/home/cvs/master

CVS password:
```

- CVSROOT can also be a simple directory, for example “/cvs”
- Other environment variables will define, for example, the text editor used to insert comments when checking in.

Example: Importing a module

- A project can be built of several modules. A simple project will have only one module.
- Let’s work on a project directory called myProject containing a file named myFile.c
- cvs import -m "message" myProject vend1 release2
- The above must be run from within the myProject directory. This will add the project to the repository as a module.

Nothing has changed in the current directory. In order to work with CVS we must check out the module we just checked in.

Example: Checking out a module

- [devGuy@LinuxRules devGuy]$ cvs co myProject
- cvs server: Updating myProject
- U myProject/myFile.c
- [devGuy@LinuxRules devGuy]$ ls myProject
- myFile.c CVS

Now we have a new directory created inside the current directory, called myProject. Inside it there is a CVS directory, which holds CVS local version info. Now we can work on our project files.

Example: Exporting a module

Export is required to checkout the project without the CVS Sub-Directories, so it’s used mostly to release a working version.

- [devGuy@LinuxRules devGuy]$ cvs export -r HEAD myProject

Example: Merging Changes

After making some changes for whatever reasons, we now have to submit our changes to the repository.

```
[devGuy@LinuxRules myProject]$ cvs update
cvs server: Updating .
M myFile.c
[devGuy@LinuxRules myProject]$ cvs commit -m "what changes were made?"
cvs commit: Examining .
Checking in myFile.c;
/home/cvs/master/myProject/myFile.c,v <– myFile.c
new revision: 1.2; previous revision: 1.1

done
```

Example: Reviewing Changes

- [devGuy@LinuxRules myProject]$ cvs log myFile.c
  RCS file: /home/cvs/master/myProject/myFile.c,v
  Working file: myFile.c
  head: 1.2
  branch:
  locks: strict
  access list:
  symbolic names:
    arelease: 1.1.1.1
    avendor: 1.1.1

  keyword substitution: kv

Reviewing Changes- Cont

```
-total revisions: 3;     selected revisions: 3
-description:
----------------------------
-revision 1.2
-date:2003/08/08 22:20:53;author:tzahi;state:Exp;lines: +3 -2
-what changes were made?
----------------------------
-revision 1.1
-date: 2003/08/08 21:59:22;  author: tzahi;  state: Exp;
-branches:  1.1.1;
-Initial revision
----------------------------
-revision 1.1.1.1
-date:2003/08/08 21:59:22;author:tzahi;state:Exp; lines:+0 -0
-no message
-============================================================
```

Actually seeing the changes

```
-[devGuy@LinuxRules myProject]$cvs diff -c -r1.1 -r1.2 myFile.c
-Index: myFile.c
-===================================================================
-RCS file: /home/cvs/master/myProject/myFile.c,v
-retrieving revision 1.1
-retrieving revision 1.2
-diff -c -r1.1 -r1.2
-*** myFile.c    8 Aug 2003 21:59:22 -0000       1.1
---- myFile.c    8 Aug 2003 22:20:53 -0000       1.2
-***************
-*** 1,2 ****
-! sdf
-! sdf
-\ No newline at end of file
---- 1,3 ----
-! Hello World!
-!
-!
```

Actually seeing the changes- Cont.

- -c means human readable form
- -r 1.1 -r 1.2 means changes need to be made to turn version 1.1 to version 1.2
- myFile.c the file to inspect. Not specifying a file will list the whole directory matching the criteria.

Example- Adding and Deleting Files

- ”cvs add myNewFile.c” will schedule a file to be added.
- ”cvs commit” will actually submit it to the repository.
- cvs rm myNewFile.c will schedule the file for removal
- cvs commit will actually remove it from future check outs. You can still see it in the repository.
- In order to commit after removing a file, you must also delete the file from your current directory.

Example- A Conflict

```
-[devGuy@LinuxRules myProject]$ cvs update
-cvs server: Updating .
-RCS file: /home/cvs/master/myProject/myFile.c,v
-retrieving revision 1.2
-retrieving revision 1.3
-Merging differences between 1.2 and 1.3 into myFile.c
-rcsmerge: warning: conflicts during merge
-cvs server: conflicts found in myFile.c
-C myFile.c
```

Example- Working Off a Conflict

This is what the file myFile.c will now looks like:

«««< myFile.c

Hello World2!

=======

Hello Conflict World!

»»»> 1.3

You can either edit the file using a text editor, removing what you don’t like, or use a GUI merger:

- Xemacs’s emerge
- kdiff3 http://kdiff3.sourceforge.net/)
- winmerge http://winmerge.sourceforge.net/)

Example - Working Off a Conflict...

```
[devGuy@LinuxRules myProject]$ cvs commit -m "solved the conflict"
cvs commit: Examining .
Checking in myFile.c;
/home/cvs/master/myProject/myFile.c,v <– myFile.c
new revision: 1.4; previous revision: 1.3

Done
```

- WARNING:
- CVS cannot be relied upon to solve all conflicts. CVS may merge two files with logically contradicting changes, if they are phisically located at different points in the file. (example - if someone else has changed an API, and you just added a function which uses this API). In this case, you will only get a MERGE warning from CVS.

Example - Working Off a Conflict...

- WARNING:
- In addition, different files may have contradicting changes. In that case, not even a MERGE warning will be produced by CVS. This is why the ChangeLog file is important, and why communication between developers is irreplacable.
- One solution is to agree on an API and every developer will have his sperate area to make changes and a change to a common file will be sent outside cvs to the area owner developer.

EXAMPLE - Tags

- Tags are used to draw straight lines connecting various project files.
- A tag may contain version 1.3 for file a and version 1.4 of file b.
- A tag can be called by a name in order to retrieve the information.
- cvs tag -r “this-is-the-tag-name-without-spaces”
- Important: the Tagging is done on the files currently in the repository, not on the files in the current directory.

Branches

- Used mostly to release a version and then continue to the next.
- For example, assume you released a branch called ver1.0, and continued to develop version 2.0. A few months later a crucial bug-fix is needed for ver1.0. You can develop the patch on branch - ver1.0 and not disturb the work on ver2.0.
- Using tags, the code of ver1.0 is available.
- Using branches, development of the bug fix is still done under version control.
- The users of ver1.0 are not forced to move on to the next (possibly unstable) version ver2.0 in order to get a bug-fix.

Repository and Directories

- A repository is really just a directory with CVS admininstration files.
- The modules are just directories in the repository.
- Even a sub directory of a project/module can be a module.
- You can publish the module in the admin "modules" file in $repository/CVSROOT/
- A project may be comprised of several modules, using the modules file.
- If you want to delete a module you can do it manually (though its not recommended).

Renaming a file

- Removing the original file and add a new one. This means the editing history of the new file will not be available. This is the proper way to avoid messing up tags and branches information.
- Manually rename the file in the repository. Will keep the history, but will mess up old branch,tag, revision states, etc.. Information.
- Duplicate the original file's history on a new name and remove the original file. Thus, you reserve history and revision information, but add to the old revision information a virtually non-existing file at the time. You can however remove tags and rev state and branch information manually if necessary to alleviate this problem. Dangerous.

Where can I find out more?

- Documentation area at http://www.cvshome.org
- Comprehensive documentation http://www.gnu.org/manual/cvs/html_mono/cvs.html
- Comparison of various VCSs http://better-scm.berlios.de/comparison/comparison.html# by Shlomi Fish.

CVSLIBS - a Java Library

- CVSLIBS is a library that can be used to reach cvs servers from you own java application. actually its an interface with various fuctions to the cvsc library of the JCVS II project. Also its LGPL'd.
- CVSLIBS can be found here:
- http://www.technion.ac.il/tzahi/