CVS @ Quios

Basic source control concepts and usage at Quios

Produced for
Quios.com

Version 1.1
June 26, 2XXX

Table of Contents:

Release History

Version	**Author**	**Date**
1.1	Bryan Scholz	June 26, 2XXX

Location and Dependencies

Document Location	
Code Location	
Dependencies	

1. The Full Documentation

http://www.loria.fr/~molli/cvs/doc/cvs_toc.html

If you really want to know the down and dirty guts of using CVS, or have an obscure question about its operation, just go to the above URL for the full manual by Per Cederqvist et al. This document is just an extremely brief introduction to CVS and how to get started with it at Quios.

2. CVS: Concurrent Versioning System

2.1 What is CVS?

CVS is a version control software system. It keeps track of different versions of files without using a huge amount of redundant storage to do so. Most importantly, it allows multiple people to work on the same project and even the same files without "stepping on" each other's files accidentally. It keeps a complete history of every change to every file, and comments on those changes. For text files, it can even provide a certain amount of automated merging.

This may seem complex, especially if you are not trained in software engineering or are unfamiliar with *source control*, a fancy term meaning any process allowing multiple people to work on a bunch of files in the same big project with (a) some history of what they have done, and (b) without literally stepping on each other's files by overwriting them. Do not be concerned; CVS is simple to use and has many GUI's that will meet your needs on just about any platform.

2.2 What is CVS Not?

- I will quote Per Cederqvist et al to describe what CVS is not:

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

> *When faced with conflicts within a single file, most developers manage to resolve them without too much effort. But a more general definition of "conflict" includes problems too difficult to solve without communication between developers. CVS cannot determine when simultaneous changes within a single file, or across a whole collection of files, will logically conflict with one another. Its concept of a* ***conflict*** *is purely textual, arising when two changes to the same base file are near enough to spook the merge (i.e. `diff3`) command. CVS does not claim to help at all in figuring out non-textual or distributed conflicts in program logic. For example: Say you change the arguments to function `X` defined in file `` `A' ``. At the same time, someone edits file `` `B' ``, adding new calls to function `X` using the old arguments. You are outside the realm of CVS's competence. Acquire the habit of reading specs and talking to your peers.*

- **CVS does not have change control**
- **CVS is not an automated testing program**
- **CVS does not have a built in process model**

2.3 How does CVS fit into my work at Quios?

Everything we are doing with the Quios Web Site is already under CVS source control. Right now it is on a physical Linux box called “linux2” which I will use for our example machine name. CVS is designed so that everyone can use it to keep track of their different versions of files and coordinate with each other. All you need to begin taking advantage of CVS is to get an appropriate CVS Client (next section) that will allow you to perform the basic CVS operations.

CVS stores all files in a central *repository*; so the first basic operation is to get or *check out* a copy of the files you will be working on. You then modify your local file in your private copy, and when you are finished, you *commit* or *check in* the modified file. CVS takes care of keeping track of the change history of your file, and of warning you if someone else edited your file while you were working on it. You can also *add* a file for the first time ever, which initiates a history of that file and creates a place for it in the repository.

3. Setting Up CVS On My Platform

3.1 Linux

On Linux, setup is trivial: CVS is already installed, and you simply run “cvs [whatever]” at the command line. You just have to make sure you are either on linux2 or have the appropriate repository NFS mounted, and set your CVSROOT environment variable to point to the repository. Currently, the correct setting in your **.bash_profile** is

```
export CVSROOT=/usr/local/src/cvsroot
```

3.2 Mac and Windows NT

On a Mac or Windows NT-based platform, you will be using the client-server properties of CVS to remotely login using *password authentication*. The process for getting set up is as follows:

1. **Download an appropriate client, such as WinCVS for NT or check out www.maccvs.org for a Mac-based client. Tom Mornini seems to have a good Mac based client as well.**
2. **Tell the CVS administrator what you want your CVS login name and password to be, and he will create it for you.**
3. **Set up your CVS GUI to point to the proper server. If the GUI asks you for the server name, that is** linux2**. Your username and password will be whatever you told the Administrator to make them. If you use a command-line interface on Windows NT, or a GUI that shows you terminal output, you may see a string like this: `“pserver:bscholz@linux2:/usr/local/src/cvsroot”.` If things do not seem to be working, see if the string looks like it has the correct form:**

"`pserver:username@server:/path/to/cvsroot`". You may need to find where in the setup dialogs you can put the correct information. If none of this works, ask someone who has already set up CVS or contact the CVS Administrator.

4. **See how those funny strings begin with "pserver"? That means Password Authentication Server, so make sure you select Password Authentication whenever your client GUI asks about Authentication Type.**
5. **Test your login by logging in however your GUI tells you to.**
6. **Get your first copy of the source by performing a *Checkout* of the "apache-common" module. The Checkout command should be pretty obvious if you look through your GUI's menus.**
7. **Once the Checkout is complete, you can just start editing files.**

3.3 Commonly Used CVS Command Line Operations

GUI's will allow you to either perform all of the following commands, or they will use the commands "behind the scenes" to present relevant information to you in a graphical manner. Just check through your menus to find the relevant commands. For you command line users, CVS always operates with respect to the current working directory, and most commands tend to be recursive when not pointed at specific files.

3.3.1 The Basics:

```
cvs checkout apache-common
```

Checks out the apache-common directory and all its subdirectories to your working folder. You execute this command when you begin working on a set of source files, and never again unless you need to move your local working directory or the source directory structure radically changes.

```
cvs -n update
cvs update myfile.mtpl
cvs update .
```

This is how you get the most recent versions of files in the Repository after the initial checkout. This command is so important that I have quoted the entire section of Update Output from the Per Cederqvist manual (after this summary). The "-n" option allows you to "pretend" to update and see what would happen if you did update at the moment. Using a specific filename will update that particular file, and using dot or passing no arguments will update everything in your current directory and everything below it, recursively, so be careful!

```
cvs commit -m "updated comments and fixed bug 123" myfile.mtpl
cvs commit myfile.mtpl
```

This is how you put changes you have made to a file into the Repository. Once this is done, you will always be able to go back to that version of the file and can safely mess up your local copy or even delete it. Henceforth, you can always get your old revision using the "update" command. Other people will now be able to "see" your file changes and will get them the next time they do a checkout or update—so make sure your file is ready before committing it. If you do not use "-m" for "message", CVS will try to pull up your favorite editor and let you enter a longer message. Always use a descriptive and succinct message.

```
cvs add -m "The best perl program ever" myfile.pl
```

When you need to add a new file, simply create it, edit it, and when satisfied, use the "add" command. The file will not actually exist in the Repository until you do a commit, but the filename will be reserved for you. You can add directories this way too, but you should talk to a manager or the CVS administrator before you start creating directories in CVS—you need to design your directory structure carefully as it is somewhat difficult to change once created.

ARTISTS AND CONTENT DEVELOPERS TAKE NOTE: Everybody needs to read this. When you add .GIF or .JPG or other graphics files, these are known as binaries and use a separate syntax for addition:

```
cvs add –kb –m "Quios main ad banner" newad.gif
```

If you accidentally forget to specify these as binary, do not panic. Simply contact your CVS administrator with the name and location of the file and he will fix it for you.

That's it for the basics. You can go a long way using these few commands. You can run

```
cvs update –h
```

or more generally

```
cvs <command> -h
```

for many more interesting options to use with commands, such as –l for mandating non-recursive behavior.

3.3.2 More Advanced Commands

To really leverage CVS's source tracking ability to help you keep track of the source, use the following commands, some of which work only on text files:

```
cvs status myfile
cvs status –v myfile
```

The Status command will give you very specific information on the particular version of a file you have, and its state, the most common States being Up-to-Date and Locally Modified (needs a commit), or Unknown (needs to be Added).

```
cvs diff myfile.txt
cvs diff -D 2000-02-15 startup.mpl
```

This will produce Unix diff-style output comparing your local revision with the one in the Repository, very useful for finding out what you have done since your last commit. This command will not work on binaries. With the "-D" argument, this compares your version with the version nearest to (in this case) February 15, 2000. Without the "-D" it compares with the most recently committed version, which will show all changes you have made since you edited the file, plus any changes anybody else has made and committed.

```
cvs log myfile
```

This extremely useful command will work on binaries and text files. It will give you the details on all versions of a particular file, and allow you to read all the messages other people have written about each committed revision. It is, in essence, a detailed history of the file.

3.4 Update Output Summary

Update output is so important that I have gone ahead and copied an excerpt explaining it. The excerpt comes from the Per Cederqvist et all manual mentioned in the first section of this document. The rest of this section is the quote from the manual:

`update` and `checkout` keep you informed of their progress by printing a line for each file, preceded by one character indicating the status of the file:

`U` *`file`*

The file was brought up to date with respect to the repository. This is done for any file that exists in the repository but not in your source, and for files that you haven't changed but are not the most recent versions available in the repository.

`P` *`file`*

Like ``U'`, but the CVS server sends a patch instead of an entire file. These two things accomplish the same thing.

`A` *`file`*

The file has been added to your private copy of the sources, and will be added to the source repository when you run `commit` on the file. This is a reminder to you that the file needs to be committed.

`R` *`file`*

The file has been removed from your private copy of the sources, and will be removed from the source repository when you run `commit` on the file. This is a reminder to you that the file needs to be committed.

`M` *`file`*

The file is modified in your working directory. ``M'` can indicate one of two states for a file you're working on: either there were no modifications to the same file in the repository, so that your file remains as you last saw it; or there were modifications in the repository as well as in your copy, but they were merged successfully, without conflict, in your working directory. CVS will print some messages if it merges your work, and a backup copy of your working file (as it looked before you ran `update`) will be made. The exact name of that file is printed while `update` runs.

`C` *`file`*

A conflict was detected while trying to merge your changes to *file* with changes from the source repository. *file* (the copy in your working directory) is now the result of attempting to merge the two revisions; an unmodified copy of your file is also in your working directory, with the name ``.#`*`file`*`.`*`revision`*`'` where *revision* is the revision that your modified file started from. Resolve the conflict as described in section Conflicts example. (Note that some systems automatically purge files that begin with ``.#'` if they have not been accessed for a few days. If you intend to keep a copy of your original file, it is a very good idea to rename it.) Under VMS, the file name starts with ``__'` rather than ``.#'`.

`?` *`file`*

file is in your working directory, but does not correspond to anything in the source repository, and is not in the list of files for CVS to ignore (see the description of the ``-I'` option, and see section Ignoring files via cvsignore).

3.5 Going Further

The details of such things as tagged revisions and administration of CVS are beyond the scope of this document and will generally be handled by the CVS Administrator. You should never attempt to do any sort of CVS administration, and you should never so much as “cd” into the central Repository (not to be confused with your working repository, which is your local copy of files). However, if you get deeply into CVS and want to know a little about tags and branching and such, simply go read the *Per Cederqvist et al* manual, which is so good I will end this document with it, as I began:

http://www.loria.fr/~molli/cvs/doc/cvs_toc.html