A Brief Introduction to CVS

John Baldwin

A Brief Introduction to CVS
by John Baldwin

$Revision: 1.5 $ Edition
Published $Date: 1999/10/16 00:01:02 $

This document is intended to serve as a brief overview of CVS for the members of the OddBall compiler programming group. It will walk the reader through an example checkout, commit, and update. For more information on CVS in general, please read the cvs(1) man and info pages on server.

Table of Contents

List of Examples

Chapter 1. Introduction and Overview

1.1. What is CVS?

CVS is a source code control system, much like RCS, SCCS, or PVCS. CVS is also an acronym. It stands for *C*oncurrent *V*ersions *S*ystem. Unlike SCCS, RCS, and PVCS, however, CVS assumes that the database of source code it is maintaining is not just used in one place. Instead, the work area and the database area are completely seperate. This easily allows for multiple work areas. Other version control systems require cutomized scripts and ugly hacks to enable multiple workspaces. This means that every developer can check out their own copy of the tree in their home directory.

In addition to seperating the work area from the database area, CVS is designed to handle multiple developers working on the same project at the same time. Thus, with CVS, developers do not need to obtain a lock on a source file before they can modify it. Instead, CVS keeps track of what versions a developer checked out. If another developer then makes changes to that file and commits those changes to the *repository* (the database that CVS maintains) then the first developer will have to update their local source tree before then can commit their changes. Fortunately, CVS automates the actual merging process, making this process very simple.

1.2. Getting Started

Before you start working with CVS, you will need to do two things:

1. Create a work area to do your work in
2. Tell CVS where the repository that you will be using is.

1.2.1. Creating a Work Area

A work area is simply a subdirectory to store your checked-out copy of the source tree. I usually use a `work/` subdirectory under my home directory to work in. You can create one for yourself like so:

Example 1-1. Creating a Work Area

Make sure you are in your home directory.

```
> cd
```

Create the actual directory

```
> mkdir work
```

1.2.2. CVSROOT: Telling CVS Where Your Repository Is

To tell CVS where the repository that you will be using is, simply set the CVSROOT environment variable to the directory containing the repository. For this project, the CVSROOT on `server` is `/home/compiler/cvs`. The following example shows how to setup CVSROOT.

Example 1-2. Setting CVSROOT

By default, your accoung will be using `tcsh` for your shell. In that case, you would use the following command to set CVSROOT:

```
> setenv CVSROOT /home/compiler/cvs
```

If you have changed your shell to a Bourne shell derivative such as `sh`, `ksh`, or `bash`, then you would use the following command:

```
# CVSROOT=/home/compiler/cvs
# export CVSROOT
```

> **Note:** The CVSROOT variable is only used by the `checkout` command. If you have already checked a source tree out, then CVS will know where you got it from automatically.

1.3. Invoking CVS

All of the functionality of CVS is stored in one binary. CVS commands are selected via a command line parameter when cvs is invoked. For example, to run the `commit` command, you would type in the following:

```
> cvs commit
[ output not shown ]
```

Chapter 2. `checkout`: How To Get The Source

To be able to work on the code, you need to have a source tree of files that you can edit. To get a copy of the source tree from CVS, you use the `checkout` command. This command takes one required argument and several optional switches. The mandatory argument specifies which part of the source tree you would like to checkout. This argument must be the name of a directory inside of the repository. For example, to check out all of the files in the `src/` subdirectory, you would use `src` for the mandatory argument. To check out everything, use the directory `.`.

Example 2-1. Checking Out The Entire Tree

Change into the work area

```
> cd ~/work
```

Checkout a copy of the source tree

```
> cvs checkout .
cvs checkout: Updating .
U Makefile
cvs checkout: Updating CVSROOT
U CVSROOT/checkoutlist
U CVSROOT/commitinfo
U CVSROOT/config
U CVSROOT/cvswrappers
U CVSROOT/editinfo
U CVSROOT/loginfo
U CVSROOT/modules
U CVSROOT/notify
U CVSROOT/rcsinfo
U CVSROOT/taginfo
U CVSROOT/verifymsg
cvs checkout: Updating doc
U doc/Makefile
cvs checkout: Updating doc/bnf
U doc/bnf/bnf.txt
cvs checkout: Updating doc/cvsguide
U doc/cvsguide/Makefile
U doc/cvsguide/book.sgml
cvs checkout: Updating doc/manual
U doc/manual/Makefile
U doc/manual/book.sgml
U doc/manual/contract.txt
U doc/manual/sampleProgram.txt
cvs checkout: Updating doc/misc
U doc/misc/slides.doc
```

```
U doc/misc/source.odd
cvs checkout: Updating mk
U mk/doc.mk
U mk/subdir.mk
cvs checkout: Updating src
U src/Makefile
U src/README
cvs checkout: Updating src/lex
U src/lex/lexer.c
cvs checkout: Updating src/prep
cvs checkout: Updating src/symtab
U src/symtab/dll.c
U src/symtab/dll.h
U src/symtab/dll_lib.c
U src/symtab/dll_lib.h
U src/symtab/symtab.c
U src/symtab/symtab.h
cvs checkout: Updating src/syntax
```

Chapter 3. `commit`: Making Changes

With CVS, you do not have to lock files. Instead, you are free to modify the files in your work directory as much as you want. Once you have tested your changes, you need to commit them to the repository so that other developers can get them. This is done with the `commit` command.

By default, the `commit` command will commit all changes in the current directory and below. You can limit the files whose changes are committed by explicitly listing the files and/or subdirectories that you want to commit on the command line. For example, this command would commit all the changes in the `doc/cvsguide` directory and any subdirectories:

```
> pwd
/v/home/john/work/compiler/doc/cvsguide
> cvs commit
[ output not shown ]
```

If I wanted to only commit the changes in `doc/cvsguide/book.sgml`, however, then I would use the following command instead:

Example 3-1. Sample Commit of `doc/cvsguide/book.sgml`

```
> pwd
/v/home/john/work/compiler/doc/cvsguide
> cvs commit book.sgml
```

The screen will then clear and you will be thrown into vi(1) to compose the log message. For example, I used the following log message for this commit:

```
Add in the first half of the commit section.

This commit is actually used inside of the CVS guide to demonstrate the
commit command.
CVS: ----------------------------------------------------------
CVS: Enter Log.  Lines beginning with `CVS:' are removed automatically
CVS:
CVS: Committing in .
CVS:
CVS: Modified Files:
CVS:    book.sgml
CVS: ----------------------------------------------------------
~
~
```

```
~
~
~
~
~
~
~
~
~
```

I then exited vi(1) to save the log message. CVS then output the following before finishing.

```
Checking in book.sgml;
/home/compiler/cvs/doc/cvsguide/book.sgml,v  <--  book.sgml
new revision: 1.2; previous revision: 1.1
done
```

Voila, revision 1.2 of `doc/cvsguide/book.sgml`has just been checked in

> **Note:** In order to commit changes to a file, you have to have the latest version checked out. If you do not have the latest version checked out, then you can use the `update` command to update your sources to the latest version. Once you have updated your sources and resolved any conflicts, you may commit your changes.

Chapter 4. `update`: Keeping Your Sources Up to Date

When other people change files in the repository, you need to be able to merge those changes into your work area. CVS can do that for you via the `update` command. When you run `cvs update`, CVS checks each file and directory that you have checked out. If any of the files are out of date (that is, a newer version is in the repository than in your work area) then it updates your copy of the file to the latest version. If you have modified one of the files in your work area, then CVS will preserve your changes. It will, however, merge the changes between tbe version of the file that you last checked out or updated to and the version in the repository into your file. For example, suppose you check out version 1.2 of a file `lextest.c`. Suppose that another developer also checks this file out and makes some changes to it. This other developer makes some changes to a function named `GenerateTest` and then commits these changes to the repository with the `commit` command. You also make some changes to `lextest.c` but to a different function named `OutputResults`. When you run `cvs update`, CVS will update your copy of `lextest.c` to contain the new version of `GenerateTest` written by the other developer. However, CVS will preserve your newer version of the `OutputResults` function even though you haven't committed your changes yet.

4.1. Conflicts

As mentioned above, the `update` command will attempt to merge in all changes automatically. However, if you and someone else both modify the same lines of the source, then CVS will not be able to merge that change in. Instead it will inform you that there were conflicts and will store the conflicts in the file that was updated. The conflicts are delimited by lines that contain seven less than and greater than symbols. They are seperated by a line containing the seven equal signs.

Example 4-1. Simple Conflict

What CVS tells you when it detects a conflict:

```
> cvs update -A lexer.c
RCS file: /home/compiler/cvs/src/lex/lexer.c,v
retrieving revision 1.5
retrieving revision 1.6
Merging differences between 1.5 and 1.6 into lexer.c
rcsmerge: warning: conflicts during merge
cvs server: conflicts found in lexer.c
C lexer.c
```

Here's the portion of the file that contains the conflict:

```
        // double quotes likely indicate the start or end of quote mode
case '"':
  if (!quoteMode)
    quoteMode = 1;
«««< lexer.c
  else if (quoteMode) a conflict demo
=======
  else if ((quoteMode) && (lexer_oneLine[lexer_nextPosition+1] != '"'))
»»»> 1.6
            quoteMode = 0;
  else
    doubleQuote = 1;
  break;
```

If you get a conflict, then you have to manually edit the file to determine what the final version of code in conflict should be. Once you have done that, you should delete the lines containing the greater than, less than, and equal signs. Then you may commit your changes. CVS will not let you commit your changes until you have resolved all of the conflicts.

4.2. Common Options

There are a couple of flags that are usually used with the `update` command. The two most often used are `-d` and `-P`. Note that multiple options can be combined into one option. In other words, instead of typing **`cvs update -d -P`**, you can type **`cvs update -dP`** to get the same effect.

`-d`

This option allows the `update` command to check out subdirectories that are not currently checked out. Thus, if someone else adds a new directory, this will get it for you. By default, cvs will only update files in directories that you have already checked out.

`-P`

This option allows CVS to “prune” any empty directories that are left after updating the files. This can happen when all of the files in a directory are removed via the remove command.

Example 4-2. A Sample Update

```
> cvs update
? src/lex/lexer
```

```
cvs server: Updating .
cvs server: Updating CVSROOT
cvs server: Updating doc
M doc/Makefile
cvs server: Updating doc/bnf
M doc/bnf/bnf.txt
cvs server: Updating doc/cvsguide
M doc/cvsguide/book.sgml
cvs server: Updating doc/manual
M doc/manual/Makefile
cvs server: Updating doc/misc
cvs server: Updating doc/share
cvs server: Updating mk
M mk/doc.mk
cvs server: Updating src
cvs server: Updating src/lex
P src/lex/lexer.c
U src/lex/lexer.h
cvs server: Updating src/prep
cvs server: Updating src/symtab
cvs server: Updating src/syntax
```

The letters prefixing the filenames indicate the status of any noteworthy files. The cvs(1) man page lists these letters and their meanings. In this example, the `src/lex/lexer` file is a file in my work area that is not in the repository. The `doc/Makefile`, `doc/bnf/bnf.txt`, `doc/cvsguide/book.sgml`, `doc/manual/Makefile`, and `mk/doc.mk` files in my work area have all been modified by me but not had those modifications committed to the repository. The `src/lex/lexer.c` and `src/lex/lexer.h` were both updated to newer versions that had been checked into the repository by someone else.