

Topics in CVS

- Basic Setup and Environment
- Review of Basic Commands
- Tags and Branches
- Diffs and Merging
- Usage/Development Guidelines
- Advanced Usage and Examples

Basic Setup and Environment

- Environment Variables:

 - CVSROOT - where to find the repository

 - CVS_RSH - how to access it over the network

 - CVSEDITOR - specific editor to use for commit log messages

- Example Settings:

 - CVSROOT=user@cvs.example.com:/home/CVSROOT

 - CVSROOT=cvs.example.com:/home/CVSROOT

 - CVSROOT=/home/CVSROOT

 - CVS_RSH=ssh

- CLI Defaults can be specified in your `~/.cvsrc`

file. To set defaults you simply list the subcommands and their default flags on the same

line. e.g.

```
cvsc -q
diff -u
status -v
```

Creating a Repository and Projects

Making a repository:

```
mkdir /path/to/repository
export CVSROOT=/path/to/repository
cvs init
```

Creating a New Project:

```
mkdir my_project
cd my_project
cvs import my_project VENDOR START
cd ..
rmdir my_project
cvs checkout my_project
cd my_project
# just add water...
```

Note that "my_project" in the import command is not a package name technically. It is actually the name of the subdirectory in the repository where the project will be stored. It is recommended that subdirectories within the repository be used for third party software distributions. c.f. fsf/gcc, mozilla/webtools, etc.

Adding and Removing Files

Adding a file:

```
vi newfile.c  
cvs add newfile.c  
cvs commit newfile.c
```

Removing a file:

```
rm oldfile.c  
cvs rm oldfile.c  
cvs commit oldfile.c
```

Examining the Status of a File

The status-subcommand tells you about tags and the condition of the file. In particular, pay attention to tags and sticky tags.

```
=====  
cv$ status -v Makefile  
=====  
File: Makefile                               Status: Needs Patch  
  
Working revision: 1.3.2.4  
Repository revision: 1.3.2.4 /home/CVSROOT/eSolstice/Makefile,v  
Sticky Tag: v1_db3_src2 (branch: 1.3.2)  
Sticky Date: (none)  
Sticky Options: (none)  
  
Existing Tags:  
v1_db3_src2 (branch: 1.3.2)  
v1_db3_src1_stable (branch: 1.2.2)
```

Here we see that the 'Makefile' needs to be patch from the repository. This file was also checked out on a branch and has the branch tag set as a sticky tag.

The best way to check status of all files without changing your sources is the following command.

```
cv$ -n -q up
```

File Status Types

Up-to-date

The local copy is the same as the repository copy.

Needs Patch

The repository copy has been updated since the last update or checkout.

Needs Merge

Same as the above and the local copy has been modified.

Needs Checkout

The local copy was lost or deleted.

Locally Added

The file is new and not yet committed to the repository.

Locally Removed

The file is scheduled for removal with the next commit. File had conflicts on merge

The file could not be completely merged automatically and the programmer must manually resolve the conflicts before commit complete successfully.

Examining a File's Commit Log

The "log" subcommand can be used to see what the developer was thinking about or trying to accomplish with certain changes. It takes many useful options.

```
cvs log -N -b -d '2000-10-30<2000-11-22' Makefile
RCS file: /home/CVSRROOT/mithril/openbsd-pkgs/Makefile,v
Working file: Makefile
head: 1.34
branch:
locks: strict
access list:
keyword substitution: kv
total revisions: 65;   selected revisions: 2
description:
-----
revision 1.5
date: 2000/11/06 01:40:04;   author: adrian;   state: Exp;   lines: +1 -1
* Removed the redundant PKGNAME macro setting from the Makefiles.
* Updated the distribution version numbers to 0.9.6.
* Updated the checksum files.
* Temporarily disabled impossible in the top level Makefile since it's broken.
-----
revision 1.4
date: 2000/10/30 07:18:14;   author: adrian;   state: Exp;   lines: +1 -1
Everything works to build packages under OpenBSD. This is the _first_
pass at the post-S/MIME conversion code. IT IS NOT FUNCTIONAL OR TESTED!
=====
```

Updating and Reosolving Conflicts

The possibility for conflicts arise when two copies of the same revision of a source file are checked out independently and both copies are modified independently. The first modified file to be committed will commit easily and with out a conflict. The status of the uncommitted source file changes from 'locally Modified' to 'Needs Merge'.

CVS will refuse to commit any changes to this file until the file has been merged with the recently committed version. To merge the file, we use the update-subcommand.

```
cv$ update myfile.c  
M myfile.c
```

In this above case, the 'M' signifies that CVS was able to successfully merged the working copy with the changed version in the repository. In this case the file status reverts back to "Locally Modified". This is possible if the changes to the file did not overlap in the source code. The changes can then be committed.

Using CVS Diff

The "diff" subcommand lets you see the differences between any two revisions of a file. If no revisions are specified, by default you get a diff between your working copy and its base version. You can pass most normal diff(1) flags to it, and the ones that cvs does not recognize will be passed to diff.

```
: adrian@playpen; cvs diff -r 1.4 -r 1.5 Makefile
Index: Makefile
=====
RCS file: /home/CVSROOT/mithril/openbsd-pkgs/Makefile,v
retrieving revision 1.4
retrieving revision 1.5
diff -u -r1.4 -r1.5
--- Makefile      2000/10/30 07:18:14      1.4
+++ Makefile      2000/11/06 01:40:04      1.5
@@ -4,7 +4,7 @@
# SUBDIR += cgihtml
# SUBDIR += doc
SUBDIR += imaproxy
SUBDIR += impossible
SUBDIR += impossible
SUBDIR += key_front_end
SUBDIR += libcrypt-client
SUBDIR += libdbd
```

Updating and Reosolving Conflicts (cont)

If both checked out copies had changes that overlapped, the probable result is a conflict. Assuming this case, we would instead see the following when updating the sources.

```
cv$ update myfile.c  
C myfile.c
```

The 'C' indicates the conflict was found. At this point the file's status has switched to 'File had conflicts on merge'. The two conflicting edits are **both** versions are inserted into the file using '<<<<<', '=====' and '>>>>>' as delimiters for the two changes. You use the editor of your choice to choose one implementation or manually merge the two alternatives. Once you do this the status will revert to "Locally Modified" or "Up-to-date" again.

Merging Across Branches (Joining)

Joining is done by using the '-j' option with the "update" subcommand instead of the '-r' option. "-r" updates your working copy to the given revision, where as '-j' merges the differences between the working copy and the version specified by the '-j'. You can use multiple '-j' flags and the order is important.

'-j' is functionally the same as "cvs diff -r X.Y source | patch source".

Tracking Third Party Sources

The import subcommand is used repeated over the life of the project with each new release to track the vendor sources on their own branch.

```
cv$ import -m 'release 1.0' samples MAGICPOINT RELEASE_1_0
```

check out, hack and commit changes...

```
cv$ import -m 'release 2.0' samples MAGICPOINT RELEASE_2_0  
cv$ checkout -jMAGICPOINT:yesterday -jMAGICPOINT samples
```

hack to resolve conflicts and then commit...

```
cv$ commit -m 'Mergred new vendor distribution.'
```

A Complex Example

Joe fixed a simple bug in `index.html` with how it submitted the CGI form. His fix was committed to the main development trunk, and I needed the patch on the stable branch that is being tested on the production servers.

The first step was to figure out which commit by Joe was the one that fixed the problem. To find it, I used the `log`-subcommand.

```
cvcs log -N index.html |more
[...]  
-----  
revision 1.23  
date: 2000/05/18 15:04:08; author: adrian; state: Exp; lines: +22 -26  
Fixed non-releatice paths for /about stuff.  
-----  
revision 1.22  
date: 2000/05/09 18:27:35; author: joe; state: Exp; lines: +1 -1  
log in method from get to post  
-----  
revision 1.21  
date: 2000/04/25 17:56:00; author: joe; state: Exp; lines: +128 -159  
new page w/ tutorial  
-----  
[...]
```

From this we learn that Joe fixed the bug and committed revision 1.22. To check out exactly what he did, I ran the following:

A Complex Example (cont)

```
cvs diff -r 1.21 -r 1.22 index.html
Index: index.html
=====
RCS file: /home/CVSROOT/eSolstice/html/index.html,v
retrieving revision 1.21
retrieving revision 1.22
diff -u -r1.21 -r1.22
--- index.html 2000/04/25 17:56:00 1.21
+++ index.html 2000/05/09 18:27:35 1.22
@@ -90,7 +90,7 @@
 height="41" BORDER=0></A></td>
<td rowspan="2">
- <form name="log" method="get" action='cgi-bin/eSol/logineRemote.pl'>
+ <form name="log" method="post" action='cgi-bin/eSol/logineRemote.pl'>
    <INPUT TYPE=hidden NAME=script VALUE=FrontPage.pl>
    <!--<A HREF="https://www.eSolstice.com">go Secure<!--<img
```

Ok, this is what we were looking for. Now to merge it into the stable branch.

```
cvs update -j 1.21 -j 1.22 index.html
M index.html
RCS file: /home/CVSROOT/eSolstice/html/index.html,v
retrieving revision 1.21
retrieving revision 1.22
Merging differences between 1.21 and 1.22 into index.html
cvs ci -m 'merged POST/GET fix from HEAD.' index.html
```

With the patch for the changes between 1.21 and 1.22 applied to the head of the stable branch, all three were left to commit the change and we are done.

Gotchas and shortcuts when removing multiple files:

```
rm *.C
cvs rm *.C
cvs server: nothing known about '*.C'
```

The problem is that you need to list all of the removed files by name, because the second wildcard doesn't have anything left to match.

Command expansion work-around (1):

```
rm *.C^X*
rm foo.c bar.c
cvs!!
```

The control sequence causes the shell to expand the wildcard immediately. The '!' is a history expansion that copies down the entire last command.

Gotchas and shortcuts when removing multiple files (cont):

Command expansion work-around (2):

```
for i in *.c; do
  rm $i
  cvs rm $i
done
```

In this variant, we simply iterate over the list of files the wildcard matches and remove the files one by one.

Primary Command Flags

- H Displays usage information for command.
- Q Cause CVS to be really quiet.
- q Cause CVS to be somewhat quiet.
- r Make checked-out files read-only.
- n Do not do anything that will change the disk.
- d dir Overrides `{CVSROOT}` as the root of CVS tree.

What's in the CVS/* Files?

Root

- where the root of the repository can be found

Repository

- path to project on CVS server. Possibly relative to the root of the repository.

Entries

- per file state, e.g. checked out version, date, locally added, locally removed, etc.
- Occasionally you may need to kill lines from this file.

Tag

- sticky tag, if applicable

others

- *don't worry about them...*

Other Resources

- CLI help:

```
cv$ [--help-commands | --help-options ]
cv$ subcommand -H
```

- The on-line texinfo pages:

```
info cvs
```

- The manpages:

```
cv$ ( 1 ) , cv$ ( 5 )
```

- SourceGear (official maintainers of CVS):

<http://www.sourcegear.com/CVS>

Watch This Space

The Subversion Project:

"The goal of the Subversion project is to build a version control system that is a compelling replacement for CVS in the open source community."

<http://subversion.tigris.org/>