Using CVS

James Gallagher*

2004/06/23, Revision: 1.8

Contents

1 Introduction

This is a summary of CVS tricks I've found useful. Some of this information is hard to find in the (few) books on CVS and some is specific to the OPeNDAP effort. This is *not* a replacement for reading the CVS documentation. There's also some information on using CVS as part of the release process in "OPeNDAP Software Release Process."

*jgallagher@opendap.org

2 Using CVS to get the OPeNDAP Software

To start the ball rolling, check out the OPeNDAP software module. Unless you are working on dodsdev.gso.uri.edu you will be using CVS to access a remote repository. We used to use the pserver method of login, but since June 2004 we've been using ssh. Here's how to access the repository:

- Set CVS_RSH to ssh
- Set CVSROOT to :ext:<user>@dodsdev.gso.uri.edu:/usr/local/cvs-repository where <user> is your user name (or anoncvs if you want anonymous/read-only access. Instead of setting CVSROOT, you can use the CVS option -d :ext:<user>@dodsdev.gso ... Once you have checked out directories, you can run CVS commands within them without setting CVSROOT or using -d.

Make sure not to add a trailing / to the repository name; some versions of CVS gag if you do.

If you want to get the entire OPeNDAP source tree, use the command:.

```
cvs -d :ext:<user>@dodsdev.gso.uri.edu:/usr/local/cvs-repository co DODS
```

It should ask for your password. Unlike the pserver access method, using ssh will prompt you for your password *every* time you run a CVS command that affects the remote repository. This can get old quickly; see Section 3.

This will check out the entire DODS *module* from CVS. If this is what you want, you're all set. To build the software you may need autoconf, GNU make, bison and flex in addition to a decent C++ compiler (we're building with gcc 3.2.1 and VC++ 6.0).

In the past the DODS CVS module was much larger. To help people cope with that we used to advise that they check out only the part of the module in which they were interested. There's little need for this now, but in case someone finds the information useful, here's how to do it:

> From now on I'm going to leave off the -d option and repository string. You'll need it unless you set CVSROOT or are working inside already checked out directories.

```
cvs co -l DODS
```

This will checkout the root directory without recurring through all the subdirectories (the -l option suppresses recurring on all the subdirectories). You will have to select the subdirectories you want by hand. Unfortunately, there is no way to tell it to checkout everything except $< foo >$ (co is an alias for checkout).

Once you have checked out the root directory you no longer need to feed CVS the -d <big long string> thing; CVS remembers which repository to use.

Use:

```
cvs co DODS/etc

cvs co DODS/src/dap
cvs co DODS/src/nc-dods
```

to get only the source directories for the DAP library and netCDF server, for example, or

```
cvs co DODS/src
```

to get all the source directories.

Because it is not possible to get CVS to create the subdirectories of DODS without checking out the subdirectories contents (which would amount to the entire DODS module - it is very large) it is hard to know which subdirectories you might want. Here is a list of the subdirectories under DODS:

```
src     The source code.
etc     Destination for servers and also some support scripts.
bin     You must make this by hand.
lib     You must make this by hand.
include The include files installed by src/dap.
```

As with all software, there are dependencies that exist between these directories. If you get the `src` directory, you will also need `etc` and include before you can build any of the software. In addition, you must make the `bin` and `lib` directories.

3 Write Access and Automatic Logins

In the past write access to CVS used `pserver`. We stopped supporting that and switched to `ssh` in June of 2004. Using `ssh` is more secure than `pserver` since passwords are not sent over the network as plain text. However, to get write access to the repository, you'll now need a full-fledged account on `dodsdev.gso.uri.edu`. Contact Mark Schneider `<mschneider@gso.uri.edu>` or James Gallagher `<jgallagher@opendap.org>` for a login.

Once you have a login, you will likely soon tire of re-typing your password. Here's how to set up automatic `ssh` login. This is particularly useful if you're access CVS code from within an IDE.

[Thanks to Rob Morris and Mark Schneider for the following. jhrg]

3.1 Automatic login from Unix

Here's Unix instructions to `ssh` to `dodsdev.gso.uri.edu` without requiring logging in (tested under Red Hat and Mandrake Linux, Solaris, OSF and OS X).

On the host machine you want to login from:

1. Generate a `ssh` key
   ```
   \% ssh-keygen -t dsa
   ```
   Just hit return as the pass phrase.

2. Copy the key generated to account on dodsdev
   ```
   \% scp .ssh/id_dsa.pub <user>@dodsdev.gso.uri.edu:.ssh/<unique filename>
   ```
   Enter your dodsdev password when prompted.
3. Repeat step one and two for each host. Use a different unique filename for each iteration.
4. Use ssh to login to dodsdev.gso.uri.edu. For each filename that was copied in step two, use an editor to paste the keys in those files into your $(HOME)/.ssh/authorized_keys file. One key per line.

Your should now be able to ssh to dodsdev.gso.uri.edu without entering a password (use the ssh option -l <user> when your user name on a host differs from that on dodsdev).

To get CVS assess to dodsdev without a password from series of UNIX hosts:

1. Do the above steps to setup ssh for each unix host you desire.
2. Add the following variables to your environment for that host

   ```
   export CVS\_RSH=ssh
   export CVSROOT=:ext:<user>@dodsdev.gso.uri.edu:/usr/local/cvs-repository
   ```

3.2 Automatic login from Win32

On the win32 host machine you want to login from:

1. Download putty.zip from http://www.puttyssh.org/ and unzip it into "C:\Program Files" (or a directory of your choice).
2. Launch puttygen and generate a “1024 bit DSA key”.
3. Paste the public key genared into your $(HOME)/.ssh/authorized_keys file on dodsdev.
4. Save the private key to disk (in your putty install directory probably) to some file such as dodsdev_private.ppk or a name of your choice.
5. Create a short cut to pagent and modify that shortcuts’ properties “Target” field so that pagent takes a 1st arg of the file saved in step number four.
6. Put that shortcut in your startup folder and reboot (or just run it). Your private key should have been loaded.
7. You can now configure and run puttySSH in a rather straight-forward manner.

Your should now be able to use ssh to access dodsdev.gso.uri.edu without entering a password.

To get cvs assess to dodsdev from a win32 host:

1. Do the above steps to setup `ssh` if necessary.
2. Download and install to `tortoiseCVS` from `http://tortoisecvs.sourceforge.net/`
3. Add the following variables to your environment for that host.

   ```
   CVS_RSH      TortoisePlink.exe                                          Or
   CVSROOT      :ext:<user>@dodsdev.gso.uri.edu:/usr/local/cvs-repository

   C:\> cd <some empty dir>
   C:\> cvs co DODS -r release-3-4
   ```

   just right click in windows explorer and select “cvs checkout” (very slick).

To make it so that CVS doesn’t require a password:

1. Create a short cut to `pagent` and modify that shortcuts’ properties “Target” field so that pagent takes a 1st arg of the file saved in step number four of the ssh install instructions above.
2. Put that shortcut in your startup folder and reboot (or just run it). Your private key should always load at login time (transparently).

4 Branching

If you have checked out code a want to check changes in on a branch instead of the main trunk, do the following:

1. Tag the branch point.
2. Create the branch.
3. Update your copy of the code so that it is associated with that branch.
4. Check in the code.

   If you want to make a branch off of a branch, these same steps will work. However, it is best to stick to the trunk-and-branch model and not bifurcate branches themselves since it can be very hard for other people to figure out just where a particular revision fits into the CVS module.

Perform the following steps in the directory that contains the code you want to move to the new branch (which I’ll call `test-branch`):

```
cvs tag root-of-test-branch
cvs tag -b test-branch
```

The first command marks the place just before the branch point. This is important if you later need to get the code as it was just before you branched. The `test-branch` tag *does not* provide a way to do that. In fact

accessing that tag will get you the most recent code *on the branch*. So, you need to create a plain tag to be able to refer to the code as it existed right before the branch was created.

The second command creates the branch. The `-b` option makes this tag a branch. The symbolic name `test-branch` can be used to refer to the branch. Without a symbolic name, you'll be hosed.[1]

```
cvs update -r test-branch
```

This updates the tag of the code in the current working directory to `test-branch`. Since the tag is on a branch, now so is this code.

```
cvs ci -m "<some message>"
```

The code is now checked in on the branch.

> Note that there's a test CVS module that can be checked out and played with. You cannot do anything too terrible to this since the 'code' is just a couple of test files. Before doing anything that might cause the DODS repository to get hosed, try it out there. The name of the test module is `cvs-test`.

5 Merging

Suppose you create a branch and make several releases from it. All the while, other developers are adding changes to the trunk and changes/fixes to the released code. As each successive release is made, the changes associated with that release should be merged back into the trunk software. However, if the entire branch is merged back onto the trunk some changes/fixes will be lost — they will be undone.[2] Here's how to work around that.

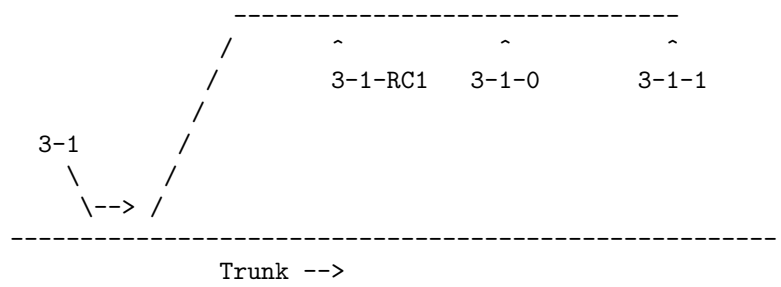


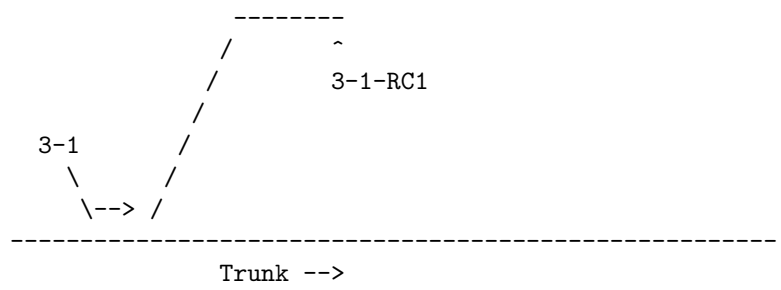
Figure: The CVS trunk and several tags. The 3-1 tag defines the code that forms the basis for a release, 3-1-RC1 is the first release candidate for version 3.1, the tag 3-1 is the first code sent to users and 3-1-1 is first

[1] In general, never use the CVS revision numbers unless you have to. Symbolic names are the most practical way to deal with groups of files. However, CVS does support date-based access including dates like `YESTERDAY`. This is a very useful feature!

[2] This is an odd quirk of CVS. It lets you remove changes made on a segment of a branch. It's important to *not* do that by mistake since it's rarely what you want to do.

set of fixes for that code. (The tags in the figure don't include the word `release` because that made it too crowded.)

First, tag every release point. Second only merge revisions with tags — this will keep things reasonably sane. Referring to the figure above, once the 3-1-RC1 tag is created (but before any further work is done) the software could be diagrammed as:

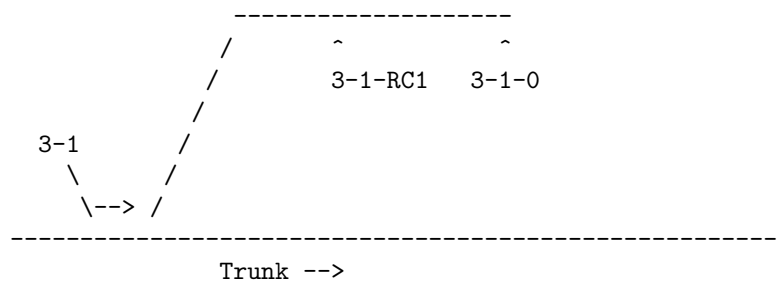


The command to merge the '3-1-RC1 changes' back onto the trunk is:

```
cvs update -j 3-1
```

This command is run in a directory that contains the trunk software.

After a while more changes are made and the 3-1-0 tag is created. A diagram of the software now looks like:



To merge the changes made between the 3-1-RC1 and 3-1-0 tags, use:

```
cvs update -j 3-1-RC1 -j 3-1
```

this command merges the changes between the two tags without removing the changes between the start of the branch and the tag 3-1-RC1 (which have already been merged). Note that the 3-1-0 tag is not used in this command. It is, however, a good idea to make it before doing the merge to make sure that the tag is created since subsequent merges will depend on it.

6 Useful commands

6.1 Shell macros

Here are some (Bourne) shell aliases I find useful:

`cvss` Show the status of any file that is not up to date in CVS. This is useful when you want to know which files have been changed.

```
alias cvss='cvs status 2>&1 | grep "'"^File\|Examining"'" | grep -v Up-to-date'
```

`cvsl` This does the same thing as cvss, but for the current directory only.

```
alias cvsl='cvs status -l 2>&1 | grep "'"^File\|Examining"'" | grep -v Up-to-date'
```

`cvsu` Show which files in the CWD are not in CVS. This is useful when you're adding files and you want to make sure that something has not been left out.

```
alias cvsu='cvs status -l 2>&1 | grep Unknown'
```

`cvsm` Get the filename only of every file in the CWD that's changed since the last check in. You can use this with an editor to great advantage. For example:

```
emacs `cvsm`
```

Opens all the changed files. You can then use Emacs' CVS tools to look at the diffs, etc.

```
alias cvsm='cvsl | cut -f 2 -d " "'
```

`cvsc` Which files in the CWD had conflicts on during the last merge operation? Note that this macro is also set up for use with an editor.

```
alias cvsc='cvsl | grep conflicts | cut -f 2 -d " "'
```

6.2 CVS logs

Looking at logs (the stuff you write into `cvs commit`):

```
cvs log -N -l -d ">1999-05-19" | more
```

Lists all the logs changed on or after 5/19/1999 (use $<$ for only after). The `-l` option suppresses recursive behavior and `-N` suppresses listing the symbolic tag names (which can be a long list for each file). When you're making a source release this can be an easy way to update the `ChangeLog` file.

6.3 Adding files to a module

Adding files:

```
cvs add <file>|<pattern>
```

Adds the file or files. You must run `cvs commit` to actually add them to the repository.

Binary file:

```
cvs add -kb <file>|<pattern>
```

The `-kb` flag suppresses keyword expansion; in case binary files (data files, GIF images, ...) contain a sequence of bytes that just happens to be `$Id$` or any of the other keywords recognized by CVS, using the `-kb` flag will keep those bytes from being changed! If you already added a binary file you can use the `cvs admin` command to turn of keyword expansion:

```
cvs admin -kb <file>|<pattern>
```

6.4 Checking out code

Checking out everything before a certain date:

```
cvs checkout -D 1999-03-25 DODS/src/nc-dods
```

checks out all the sources in `DODS/src/nc-dods` as they were *on or before* 25 March 1999.

Checking out tagged sources. See the CVS manual or info pages about tags. I use tags for *all* the releases, so you need to look at this:

```
cvs checkout -r release-2-10 DODS/src/nc-dods
```

checks out all the sources tagged `release-2-10`. Note that CVS does not allow dots (.) in tag names. Also note that if the tag is a branch point you now have a branch of the source tree to work in. Changes you make will be checked in on the branch, not the trunk (that's good). To move the changes over to the trunk see Section 5.

The checkout command does a complete checkout of a module or part of a module. If you want new sources inside an existing project tree, use `update`. If the file or directory does not exist, use `update -d`. In fact, it is best to use `update -d` always unless you *know* that no new files or directories have been added.[3] Here's examples of the two above operations using update instead of checkout:

```
cvs update -d -D 1999-03-25 nc-dods

cvs update -d -r release-2-10 nc-dods
```

From within `DODS/src/` will get the `nc-dods` sources from on or before 25 march 1999 or tagged `release-2-10`. The difference is that these commands are run from within an existing tree; the checkout is run above the tree.

6.5 Using the status command

Looking at information about the revisions of a source file:

[3] Also useful with `update` is the `-P` option which will prune any empty directories. CVS keeps the directory when files are removed because other revisions might need that directory. The `-P` option removes that clutter when it's not necessary.

```
    cvs status -v ChangeLog
```

Here's the output for that file:

```
[dcz:/usr/local/ferret-DODS/src/nc-dods-2.19.2] cvs status -v ChangeLog
===================================================================
File: ChangeLog         Status: Up-to-date

   Working revision:    1.3.4.2 Wed Apr 14 03:33:45 1999
   Repository revision: 1.3.4.2 /usr/local/cvs-repository/DODS/src/nc-dods/ChangeLog,v
   Sticky Tag:          ferret (branch: 1.3.4)
   Sticky Date:         (none)
   Sticky Options:      (none)

   Existing Tags:
ferret                          (branch: 1.3.4)
release-2-18                    (branch: 1.3.2)
no-gnu                          (branch: 1.2.4)
release-2-17                    (branch: 1.2.2)
```

You can see that there are four tagged revisions (the tags are ferret, `release-2-18`, `no-gnu` and `release-2-17`). The `Sticky Tag` indicates that I'm using the revision tagged `ferret`. Note that it is a tag on a branch and that I've made some changes. Here's the status of the same file in the development directory (i.e., on the trunk):

```
[dcz:/usr/local/DODS/src/nc-dods] cvs status -v ChangeLog
===================================================================
File: ChangeLog         Status: Up-to-date

   Working revision:    1.3     Thu Jan 21 22:56:55 1999
   Repository revision: 1.3     /usr/local/cvs-repository/DODS/src/nc-dods/ChangeLog,v
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)

   Existing Tags:
ferret                          (branch: 1.3.4)
release-2-18                    (branch: 1.3.2)
no-gnu                          (branch: 1.2.4)
release-2-17                    (branch: 1.2.2)
```

Note that the same tags are listed but that the Stick Tag is none and the revision number of the working file is 1.3, not 1.3.4.2 as in the previous example.

Aside: Don't confuse CVS' revision numbers with our version numbers.

A ChangeLog

```
$Log: using-cvs.tex,v $
Revision 1.8  2004/06/23 19:52:28  jimg
Added new CVS info for win32 from Rob.

Revision 1.7  2004/06/22 23:30:27  jimg
Fixed/replace Sectionref usage

Revision 1.6  2004/06/22 20:45:18  jimg
Updated with information about the new CVS access method (using ssh).

Revision 1.5  2004/01/09 20:01:02  jimg
Fixed some errors and made it less of a collection of old ASCII files and
more of a single paper.

Revision 1.4  2004/01/08 21:08:10  jimg
Formatted to the end and removed some material that was better covered in
"OPeNDAP Software Release Process."

Revision 1.3  2004/01/08 19:15:16  jimg
Started editing and formatting.

Revision 1.2  2003/06/19 00:36:16  jimg
Start at formatting...

Revision 1.1  2003/06/19 00:12:33  jimg
Initial version.
```