NAME

cvshelp – advice on using the Concurrent Versions System

DESCRIPTION

This man page is based on experience using CVS. It is bound to change as we gain more experience. If you come up with better advice than is found here, contact the Software Technology Group and we will add it to this page.

Getting Started

Use the following steps to prepare to use CVS:

- Take a look at the CVS manual page to see what it can do for you, and if it fits your environment (or can possibly be made to fit your environment).

  man cvs

  If things look good, continue on...

- Setup the master source repository. Choose a directory with ample disk space available for source files. This is where the RCS ',v' files will be stored. Say you choose **/src/master** as the root of your source repository. Make the **CVSROOT.adm** directory in the root of the source repository:

  mkdir /src/master/CVSROOT.adm

- Populate this directory with the *loginfo* and *modules* files from the **/usr/doc/local/cvs** directory. Edit these files to reflect your local source repository environment – they may be quite small initially, but will grow as sources are added to your source repository. Turn these files into RCS controlled files:

  cd /src/master/CVSROOT.adm
  ci –m'Initial loginfo file' loginfo
  ci –m'Initial modules file' modules

- Run the command:

  mkmodules /src/master/CVSROOT.adm

  This will build the **ndbm**(3) file for the modules database.

- Remember to edit the *modules* file manually when sources are checked in with **checkin** or CVS **add**. A copy of the *modules* file for editing can be retrieved with the command:

  cvs checkout CVSROOT.adm

- Have all users of the CVS system set the CVSROOT environment variable appropriately to reflect the placement of your source repository. If the above example is used, the following commands can be placed in a *.login* or *.profile* file:

  setenv CVSROOT /src/master

  for csh users, and

  CVSROOT=/src/master; export CVSROOT

  for sh users.

Placing Locally Written Sources Under CVS Control

Say you want to place the 'whizbang' sources under CVS control. Say further that the sources have never been under revision control before.

- Move the source hierarchy (lock, stock, and barrel) into the master source repository:

  mv ~/whizbang $CVSROOT

- Clean out unwanted object files:

  cd $CVSROOT/whizbang
  make clean

- Turn every file in the hierarchy into an RCS controlled file:

  descend −f 'ci −t/dev/null −m"Placed under CVS control" −nV*x_y* ***'**

  In this example, the initial release tag is **V***x_y*, representing version *x.y*.

You can use CVS on sources that are already under RCS control. The following example shows how. In this example, the source package is called 'skunkworks'.

- Move the source hierarchy into the master source repository:

  mv ˜/skunkworks $CVSROOT

- Clean out unwanted object files:

  cd $CVSROOT/skunkworks
  make clean

- Clean out unwanted working files, leaving only the RCS ',v' files:

  descend −r rcsclean

  Note: If any working files have been checked out and changed, **rcsclean** will fail. Check in the modified working files and run the command again.

- Get rid of **RCS** subdirectories. CVS does not use them.

  descend −r −f 'mv RCS/*,v .'
  descend −r −f 'rmdir RCS'

- Delete any unwanted files that remain in the source hierarchy. Then make sure all files are under RCS control:

  descend −f 'ci −t/dev/null −m"Placed under CVS control" −n*tag* ***'**

  tag is the latest symbolic revision tag that you applied to your package (if any). Note: This command will probably generate lots of error messages (for directories and existing RCS files) that you can ignore.

Placing a Third-Party Source Distribution Under CVS Control

The **checkin** command checks third-party sources into CVS. The difference between third-party sources and locally written sources is that third-party sources must be checked into a separate branch (called the *vendor branch*) of the RCS tree. This makes it possible to merge local changes to the sources with later releases from the vendor.

- Save the original distribution kit somewhere. For example, if the master source repository is **/src/master** the distribution kit could be saved in **/src/dist**. Organize the distribution directory so that each release is clearly identifiable.
- Unpack the package in a scratch directory, for example **˜/scratch**.
- Create a repository for the package. In this example, the package is called 'Bugs-R-Us 4.3'.

  mkdir $CVSROOT/bugs

- Check in the unpacked files:

  cd ˜/scratch
  checkin −m 'Bugs-R-Us 4.3 distribution' bugs VENDOR V4_3

  There is nothing magic about the tag 'VENDOR', which is applied to the vendor branch. You can use whatever tag you want. 'VENDOR' is a useful convention.

- Never modify vendor files before checking them in. Check in the files *exactly* as you unpacked them. If you check in locally modified files, future vendor releases may wipe out your local changes.

Working With CVS-Controlled Sources

To use or edit the sources, you must check out a private copy. For the following examples, the master files are assumed to reside in **$CVSROOT/behemoth**. The working directory is **~/work**. See **cvs**(local) for more details on the commands mentioned below.

To Check Out Working Files

Use CVS **checkout**:

cd ~/work
cvs checkout behemoth

There is nothing magic about the working directory. CVS will check out sources anywhere you like. Once you have a working copy of the sources, you can compile or edit them as desired.

To Display Changes You Have Made

Use CVS **diff** to display detailed changes, equivalent to **rcsdiff**(local). You can also use **cvscheck**(local) to list files added, changed, and removed in the directory, but not yet **commit**ted. You must be in a directory containing working files.

To Display Revision Information

Use CVS **log**, which is equivalent to **rlog**(local). You must be in a directory containing working files.

To Update Working Files

Use CVS **update** in a directory containing working files. This command brings your working files up to date with changes checked into the master repository since you last checked out or updated your files.

To Check In Your Changes

Use CVS **commit** in a directory containing working files. This command checks your changes into the master repository. You can specify files by name or use

cvs commit −a

to **commit** all the files you have changed.

To Add a File

Add the file to the working directory. Use CVS **add** to mark the file as added. Use CVS **commit** to add the file to the master repository.

To Remove a File

Remove the file from the working directory. Use CVS **remove** to mark the file as removed. Use CVS **commit** to move the file from its current location in the master repository to the CVS *Attic* directory.

To Add a Directory

Add the directory to the working directory. Use CVS **add** to add the directory to the master repository.

To Remove a Directory

You shouldn't remove directories under CVS. You should instead remove their contents and then prune them (using the **−f** and **−p** options) when you **checkout** or **update** your working files.

To Tag a Release

Use CVS **tag** to apply a symbolic tag to the latest revision of each file in the master repository. For example:

cvs tag V2_1 behemoth

To Retrieve an Exact Copy of a Previous Release

During a CVS **checkout** or **update**, use the **−r** option to retrieve revisions associated with a symbolic tag. Use the **−f** option to ignore all RCS files that do not contain the tag. Use the **−p** option to prune directories that wind up empty because none of their files matched the tag. Example:

cd ˜/work
cvs checkout −r V2_1 −f −p behemoth

Logging Changes

It is a good idea to keep a change log together with the sources. As a minimum, the change log should name and describe each tagged release. The change log should also be under CVS control and should be tagged along with the sources.

cvslog(local) can help. This command logs changes reported during CVS **commit** operations. It automatically updates a change log file in your working directory. When you are finished making changes, you (optionally) edit the change log file and then commit it to the master repository.

Note: You must edit the change log to describe a new release and **commit** it to the master repository *before* **tag**ging the release using CVS. Otherwise, the release description will not be included in the tagged package.

See **cvslog**(local) for more information.

Merging a Subsequent Third-Party Distribution

The initial steps in this process are identical to placing a third-party distribution under CVS for the first time: save the distribution kit and unpack the package in a scratch directory. From that point the steps diverge. The following example considers release 5.0 of the Bugs-R-Us package.

- Check in the sources after unpacking them:

  cd ˜/scratch
  **checkin −m 'Bugs-R-Us 5.0 distribution' bugs VENDOR V5_0 **
  | tee ˜/WARNINGS

  It is important to save the output of **checkin** in a file because it lists the sources that have been locally modified. It is best to save the file in a different directory (for example, your home directory). Otherwise, **checkin** will try to check it into the master repository.

- In your usual working directory, check out a fresh copy of the distribution that you just checked in.

  cd ˜/work
  cvs checkout −r VENDOR bugs

  The **checkout** command shown above retrieves the latest revision on the vendor branch.

- See the 'WARNINGS' file for a list of all locally modified sources. For each locally modified source, look at the differences between the new distribution and the latest local revision:

  cvs diff −r *LocalRev file*

  In this command, *LocalRev* is the latest numeric or symbolic revision on the RCS trunk of *file*. You can use CVS **log** to get the revision history.

- If your local modifications to a file have been incorporated into the vendor's distribution, then you should reset the default RCS branch for that file to the vendor branch. CVS doesn't provide a mechanism to do this. You have to do it by hand in the master repository:

  rcs −bVENDOR *file*,**v**

- If your local modifications need to be merged with the new distribution, use CVS **join** to do it:

  cvs join −r VENDOR *file*

  The resulting file will be placed in your working directory. Edit it to resolve any overlaps.

- Test the merged package.
- Commit all modified files to the repository:

  cvs commit −a

- Tag the repository with a new local tag.

Applying Patches to Third-Party Sources

Patches are handled in a manner very similar to complete third-party distributions. This example considers patches applied to Bugs-R-Us release 5.0.

- Save the patch files together with the distribution kit to which they apply. The patch file names should clearly indicate the patch level.
- In a scratch directory, check out the last ‘clean’ vendor copy − the highest revision on the vendor branch with *no local changes*:

  cd ˜/scratch
  cvs checkout −r VENDOR bugs

- Use **patch**(local) to apply the patches. You should now have an image of the vendor’s software just as though you had received a complete, new release.
- Proceed with the steps described for merging a subsequent third-party distribution.
- Note: When you get to the step that requires you to check out the new distribution after you have checked it into the vendor branch, you should move to a different directory. Do not attempt to **checkout** files in the directory in which you applied the patches. If you do, CVS will try to merge the changes that you made during patching with the version being checked out and things will get very confusing. Instead, go to a different directory (like your working directory) and check out the files there.

Advice to Third-Party Source Hackers

As you can see from the preceding sections, merging local changes into third-party distributions remains difficult, and probably always will. This fact suggests some guidelines:

- Minimize local changes. *Never* make stylistic changes. Change makefiles only as much as needed for installation. Avoid overhauling anything. Pray that the vendor does the same.
- Avoid renaming files or moving them around.
- Put independent, locally written files like help documents, local tools, or man pages in a subdirectory called ‘local-additions’. Locally written files that are linked into an existing executable should be added right in with the vendor’s sources (not in a ‘local-additions’ directory). If, in the future, the vendor distributes something equivalent to your locally written files you can CVS **remove** the files from the ‘local-additions’ directory at that time.

SEE ALSO

cvs(local), **checkin**(local), **cvslog**(local), **cvscheck**(local)

AUTHOR

Lowell Skoog
Software Technology Group
Technical Computing