ESD Software Engineering Group

CVS Cheat-sheet

SLAC Detailed
SLAC Computing
Software
Software Detailed
CVS

This page outlines some of the common CVS commands as they may be used in the SLAC ESD Software Group. See the CVS Manual for more, in particular Multiple Developers (chapter 10) and the Guide to CVS Commands (appendix A).

Contents: checkout, add, commit, update, release, import. Real-world Examples
See Also: How to Undelete a file (see section 6.2 at that URL).

Common CVS Command Summary

cvs checkout The way you reserve files in CVS that you want to edit.

To check out a subdirectory tree, cd to the directory in your working space into which you want cvs to place the directory it checks out. Use "cvs checkout" giving the name of the directory in the cvs repository you want to checkout, where the name you give is a directory under CVSROOT, presently $CD_SOFT/cvs (eg app/alh, script). The directory you give, and all subdirectories, will be placed in your working directory.

	`cvs checkout [options] modules...`	
Egs	`cvs checkout edu`	Checks out everything under edu, placing an edu directory, plus all subordinate subdirectories, in the working dir.
	`cvs checkout app/alh`	Checks out everything under app/alh placing an app directory in your working directory, but only the alh subdirectory of app.
	`cvs co package/aida/common/script/MakefileAida.sun4`	Checks out only the single file MakefileAida.sun4, creating package/aida/common/script/MakefileAida.sun4 in the working directory, containing that one file.

cvs add Add new files to an existing directory under CVS control.

The cvs add command tells CVS to add the given file to its list of files that should be tracked in the working directory. The file is not created in the repository until you CVS commit the file or the directory it's in. The file must be in the local directory when you cvs add it; that is, you can't cp and cvs add with one command by giving a full pathname of the file. To add a whole new directory hierarchy to the source repository (for example, files received from a third-party vendor), use the cvs import command instead. See CVS Manual section A.12 import--Import sources into CVS, using vendor branches.

	`cvs add [-k kflag] [-m message] files ...`	
Egs	`cvs add myfile.c`	Schedule *myfile.c* to be added to the repository

cvs commit Put your changes in CVS

The commit command is used to place the changes you made to files in your local working directory back into the CVS repository. Note that it is usually a good idea to run *cvs update* on your checked out files *before* running the cvs commit, so that cvs can alert you to possible conflicts between your changes and changes that may have been made to the repository since you did your cvs checkout. See CVS Manual “Bringing a file Up to Date”.

	`cvs commit [-lnRf] [-m 'log_message' | -F file] [-r revision] [files...]`	
Egs	`cvs commit`	Commit everything from the working directory down.
	`cvs commit -m "add test suite" package/aida/common/script/MakefileAida.sun4`	Commit only the file given, and give the comment in the command line rather than start an editor.

cvs update Bring a checkout up to date with the repository

The cvs update command is used to merge changes that have been made to a repository into files that have been checked out. Note that it is reverse operation from the one we normally do on VMS, we only ever merge changes made *from a checkout into* the CMS repository. Since in cvs the norm is to checkout whole directory trees, cvs update is the way you find out if anyone has checked stuff in on to of you. In particular its a good idea to run cvs update on your checked out files before running the cvs commit. See CVS Manual “Bringing a file Up to Date”.

	`cvs update [-ACdflPpR] [-I name] [-j rev [-j rev]] [-k kflag] [-r tag|-D date] [-W spec] files...`	
Egs	`cvs update -dA package/aida/common/script`	Bring working dir specified up to date with the repository (merge changes made to the repository into the local files).
	`cvs update -A package/aida/common/script/MakefileAida.sun4`	Bring just the named file up-to-date with the repository

cvs release To relinquish your interest in a branch of the repository

The release command is used only to tell cvs you are no longer interested in the part of the repository you checked out. CVS release will alert you to whether you have left any modified files in your local working directory, and then confirm the release. If you confirm, it will make a note in the history file. Note, the CVS release command is not used to put files into the repository like the phonetically similar CMS replace.

	`cvs release [-d] directories...`	
Egs	`cvs release package/aida/common/script`	Tell CVS that you’re no longer interested in package/aida/common/script.
	`cvs release -d package/aida/common/script`	Tell CVS that you’re no longer interested in package/aida/common/script.and tell cvs to delete your working copy of this directory tree.

cvs import The way you create a new directory or tree of directories in CVS.

You use a cvs import command when you want to add a whole directory to CVS. CVS import is not used to add a bunch of files to an existing directory - for that use "cvs add" (see above). Before getting into the command itself, first pick a place in the existing cvs tree where you want to add your stuff. For this example, let's say you wanted to add a directory of "tool" files to cvs at the new directory "common/tool", so its reference directory would be $CD_SOFT/ref/common/tool/. The argument you would have to give to the cvs import command will be "common/tool". The argument is always the full pathname, after the $CD_SOFT/cvs part, of the root of the directory you want to create, even if some of the intermediate directories already exist (in this case, "common/" already exists).
cvs import always imports all the files, and all subdirectories, in the working directory **from which it is being run**. That is, it imports a directory tree into the place specified by the argumetk. So, be careful not to do something like cd to a directory which

contains the **root** of a directory tree which you want to import and then issue cvs import giving as the argument the leaf-of-directory-tree you want to import, e.g. cd ~/work (containing common/tool) and then cvs import common/to.ol. That would create $CD_SOFT/cvs/common/tool/common/tool/!! If you only want to import *a single directory*, then the root and the leaf are the same directory, so you can use a sequence of commands as in example 1) below. But if you really want to import more than one directory, you have to use a sequence like that in example 2.

Also be careful not to import a directory system that contains a subdirectory that is itself the result of a CVS checkout, because that subdirectory will contain a CVS subdirectory. This is very messy to clean up. You shouldn't ever want to anyway, because cvs import must always be run from the directory whose files you want to import, and always takes the fully qualified cvs module name as the argument.

The two other arguments to cvs import are the "vendor tag", and the "release" tag:

- "vendor tag" is a free form text string you're supposed to use to identify the vendor of software. Since it's a CVS tag, it should be all upper case and not have any special charatcters save the "_" (like no "." or "-"). Our standard for this tag is "CD_SOFT", when we're the vendors.
- "release tag", is also a free form text string you're supposed to use to identify the release of the software you're putting in CVS. For EPICS software, we use a release tag like "R3_13_6", for all other software, for the initial release, we use "R1_0".

After you have done the cvs import, be sure to go to the corresponding reference area and do the initial cvs checkout.

Eg	`cvs import [options] directory-name vendor-tag release-tag`	

1	`cd ~/work/common/tool`
	
	`cvs import common/tool CD_SOFT R1_0`
	
	`cd $CD_SOFT/ref`
	`cvs checkout common/tool`
		Say ~/work/common/tool is the directory where all the tool files are. All the files in that directory will be imported (unless they're in the CVSIGNORE set).
		Imports all the files from your working directory, into cvs/common/tool.
		Creates the initial checkout of the directory you just created in cvs.

2	`cd ~/work`
	
	`cvs import app/myapp CD_SOFT R1_0`
	
	`cd $CD_SOFT/ref`
	`cvs checkout app/myapp`
		Say ~/work is the **root** directory of where all the files are of a new application are. All the files and all the subdirectories in that directory will be imported into cvs/app/myapp (unless they're in the CVSIGNORE set).
		
		Imports all the files from ~/work, into cvs/app/myapp.

3	`cvs import -m "initial import" ... app/myapp CD_SOFT R1_0`	As above, but gave a comment on the command line rather than making cvs start an editor and asking for the comment interactively.

Real-world examples

Modifying a single file in a single directory

In this example we modify a single makefile in the AIDA script area. It is checked out, modified, and checked back in.

`cd tmp`	Move to the directory in which you want to work
`cvs co package/aida/common/script/MakefileAida.sun4`	Cvs checkout the file you want to work with
`cd package/aida/common/script`	Change dir to dir of

`Emacs MakefileAida.sun4 &`	Edit file
`cd ../../../..`	Go back up to issue cvs update and commit, so the filename on which to act is the same as when it was checked out.
`cvs update -A package/aida/common/script/MakefileAida.sun4`	Verify that no-one has modified the file in the repository since you checked it out. CVS should reply simply “M” meaning you have modified the file.
`cvs commit package/aida/common/script/MakefileAida.sun4`	Update the repository with your modified file.
`cvs release package/aida/common/script`	Relinquish interest
`rm -r package`	Delete local package dir and all sub-dirs.

Modifying an entire “package”

In this example we checkout a whole sub-tree, add a file copied in from elsewhere, modify it and another file, and check in the whole directory tree. The sub-tree used for illustration is that containing Aida, but this may be any sub-tree in the CD_SOFT repository.

`cd tmp`	
`cvs co package/aida`	Checkout Aida (cvs/package/aida and all subordinate directories)
`cd package/aida/common/script`	Change dir to where you want to do some work
`cp /afs/slac/package/aida/common/script/aidapackagelist.txt .`	Copy a new file to be added to CVS into the checked out area.
`cvs add aidapackagelist.txt`	Tell CVS about the file you want to add
`emacs aidapackagelist.txt`	Modify the added file
`emacs MakefileAida.sun4`	Modify another file that was already in this directory
`cd ../../../..`	Go back up to the directory from which the checkout was made, in order to do the update verification and commit.
`cvs update -dA package/aida`	Merge in updates that other developers have done to your local copy of package/aida.
`cvs commit`	Update the CVS repository with your changes
`<gmake aida>`	Make aida. If it builds clean go on to the next step, if it doesn’t you can keep cvs committing modified versions from the local directory and re-updating the reference area until it does.
`cvs release package/aida`	Finally, when the reference area is rebuilt, relinquish package/aida.
`rm -r package`	Clean-up local workspace.

Owner: Greg White
Last modified: Monday 12-Jan, 2004. Ron MacKenzie. Removed directions that said to update the reference area. That is now done automatically for you. 28-Mar-2005, Greg White:Clarify cvs import command help, or at least give more help.