CVS Tutorial

This file is designed to give you a bird's eye view of the workings of the Concurrent Versions System (CVS). It is based (loosely) around CVS version 1.3. It is recommended that you make sure the CVS your are working with this version before continuing.

Please send me comments/additions/questions about this tutorial. It needs more input from folks. Novices can help by pointing out confusing sections or by sending me questions not covered by this file. Masters can help by first verifying that the information here is sane, by sending their own tutorials, and by generally providing intelligent feedback.

Thanks, Gray Watson `<gray.watson@antaire.com>`

0.1 Basic description of the CVS system.

CVS is a system that lets groups of people work simultaneously on groups of files (for instance program sources).

It works by holding a central 'repository' of the most recent version of the files. You may at any time create a personal copy of these files by 'checking out' the files from the repository into one of your directories. If at a later date newer versions of the files are put in the repository, you can 'update' your copy.

You may edit your copy of the files freely. If new versions of the files have been put in the repository in the meantime, doing an update merges the changes in the central copy into your copy.

When you are satisfied with the changes you have made in your copy of the files, you can 'commit' them into the central repository.

When you are finally done with your personal copy of the files, you can 'release' them and then remove them.

0.2 Some basic words and descriptions.

`Repository`
: The directory storing the master copies of the files. The main or master repository is a tree of directories.

`Module`
: A specific directory (or mini-tree of directories) in the main repository. Modules are defined in the CVS modules file.

`RCS`
: Revision Control System. A lower-level set of utilities on which CVS is layered.

`Check out`
: To make a copy of a file from its repository that can be worked on or examined.

`Revision`
: A numerical or alpha-numerical tag identifying the version of a file.

0.3 Look at the CVS basic command set.

Most of the below commands should be executing while in the directory you checked out. If you did a `cvs checkout malloc` then you should be in the malloc sub-directory to execute most of these commands. `cvs release` is different and must be executed from the directory above.

`cvs checkout (or cvs co)`

To make a local copy of a module's files from the repository execute `cvs checkout module` where module is an entry in your modules file (see below). This will create a sub-directory module and check-out the files from the repository into the sub-directory for you to work on.

`cvs update`

To update your copy of a module with any changes from the central repository, execute `cvs update`. This will tell you which files have been updated (their names are displayed with a U before them), and which have been modified by you and not yet committed (preceded by an M).

It can be that when you do an update, the changes in the central copy clash with changes you have made in your own copy. You will be warned of any files that contain clashes by a preceding C. Inside the files the clashes will be marked in the file surrounded by lines of the form `<<<<` and `>>>>`. You have to resolve the clashes in your copy by hand. After an update where there have been clashes, your original version of the file is saved as '`.#file.version`'.

If you feel you have messed up a file and wish to have CVS forget about your changes and go back to the version from the repository, delete the file and do an `cvs update`. CVS will announce that the file has been "lost" and will give you a fresh copy.

`cvs commit`

When you think your files are ready to be merged back into the repository for the rest of your developers to see, execute `cvs commit`. You will be put in an editor to make a message that describes the changes that you have made (for future reference). Your changes will then be added to the central copy.

When you do a commit, if you haven't updated to the most recent version of the files, CVS tells you this; then you have to first update, resolve any possible clashes, and then redo the commit.

`cvs add` and `cvs remove`

It can be that the changes you want to make involve a completely new file, or removing an existing one. The commands to use here are:

```
cvs add 'filename'
cvs remove 'filename'
```

You still have to do a commit after these commands to make the additions and removes actually take affect. You may make any number of new files in your copy of the repository, but they will not be committed to the central copy unless you do a `cvs add`.

CVS remove does not actually remove the files from the repository. It only removes them from the "current list" and puts the files in the CVS Attic. When another person checks out the module in the future they will not get the files that were removed. But if you ask for older versions that had the file before it was removed, the file will be checked out of the Attic.

`cvs release`

When you are done with your local copy of the files for the time being and want to remove your local copy use `cvs release module`. This must be done in the directory above the module sub-directory you which to release. It safely cancels the effects of `cvs checkout`. Usually you should do a commit first.

If you wish to have CVS also remove the module sub-directory and your local copy of the files then your `cvs release -d module`.

NOTE: Take your time here. CVS will inform you of files that may have changed or it does not know about (watch for the ? lines) and then with ask you to confirm this action. Make sure you want to do this.

`cvs log` To see the commit messages for files, and who made them, use:

```
cvs log ['filename(s)']
```

`cvs diff` To see the differences between your version of the files and the version in the repository do:

```
cvs diff ['filename(s)']
```

`cvs tag`

One of the exciting features of CVS is its ability to mark all the files in a module at once with a symbolic name. You can say 'this copy of my files is version 3'. And then later say 'this file I am working on looked better in version 3 so check out the copy that I marked as version 3.'

Use `cvs tag` to tag the version of the files that you have checked out. You can then at a later date retrieve this version of the files with the tag.

```
cvs tag tag-name [filenames]
```

Later you can do:

```
cvs co -r tag-name module
```

`cvs rtag` Like tag, rtag marks the current versions of files but it does not work on your local copies but on the files in the repository. To tag all my libraries with a version name I can do:

```
cvs rtag LIBRARY_2_0 lib
```

This will recursively go through all the repository directories below lib and add the LIBRARY_2_0 tag to each file. This is one of the most useful features of CVS (IMHO). Use this feature if you about to release a copy of the files to the outside world or just want to mark a point in the developmental progression of the files.

`cvs history`

To find out information about your CVS repositories use the `cvs history` command. By default history will show you all the entries that correspond to you. Use the -a option to show information about everyone.

```
cvs history -a -o   shows you (a)ll the checked (o)ut modules
cvs history -a -T   reports (a)ll the r(T)ags for the modules
cvs history -a -e   reports (a)ll the information about (e)verything
```

0.4 The things to do before using CVS.

Make sure all your developers have the CVSROOT environment variable set to the directory that is to hold your main file repository (mine is set to '`/usr/src/master`'). The following commands can be placed in a '`.cshrc`' or '`.profile`' files.

```
setenv CVSROOT /src/master
```

for tcsh/csh users, and

```
CVSROOT=/src/master; export CVSROOT
```

for bash/sh users.

Run the cvsinit script that comes with CVS to initialize the repository tree.

Encourage all your developers to make a working directory where they will be working on the files (mine is '~/src/work').

Edit the modules file to add the local *modules*. You can either do this by cd'ing to '${CVSROOT}/CVSROOT' and saying `co -l modules` and then editing 'modules', or, better, cd to your working directory and do a `cvs co modules` (`co` is an alias for `checkout`).

Next add your modules to the file. I added the following lines to my file:

```
# libraries
lib             antaire/lib

db              antaire/lib/db
dt              antaire/lib/dt
inc             antaire/lib/inc
lwp             antaire/lib/lwp
malloc          antaire/lib/malloc
inter           antaire/lib/inter
prt             antaire/lib/inter/prt
```

The above entries now allow me to `cvs co malloc` to create a directory 'malloc' where I am and check out the files from '${CVSROOT}/antaire/lib/malloc' into that directory.

`cvs co lib` will check out all my libraries and make a whole tree under lib: 'lib/db/*', 'lib/dt/*', 'lib/inc/*', etc.

Next, create a 'cvsignore' file in '${CVSROOT}/CVSROOT'. This file contains the local files that you want CVS to ignore. If you have standard temporary files, or log files, etc. that you would never want CVS to notice then you need to create this file.

The first time you should go into the CVSROOT directory, edit the file and `ci -u cvsignore` to check it in.

You should apply the mkmodules.patch then recompile and install the mkmodules file See ⟨undefined⟩ [Mkmodules Patch], page ⟨undefined⟩. Finally add the following line to your modules file (see above) so you can use CVS to edit the file in the future.

```
cvsignore       -i mkmodules CVSROOT cvsignore
```

I have in my file:

```
*.t
*.zip
MAKE.LOG
Makefile.dep
a.out
logfile
...
```

CVS ignores a number of common temp files (`*~`, `#*`, `RCS`, `SCCS`, etc..) automatically. (see the manual entry for cvs(5)).

warning: CVS is good at this. Any files in the cvsignore file will be ignored completely without a single warning.

Now, add your files into their respective module directories:

cd into your current directory which holds the files.

Build clean/clobber and make sure that only the files you want to be checked into the repository are in the current directory. Execute:

```
cvs import -m 'comment' repository vendortag releasetag
```

The comment is for you to document the module

The `repository` should be a path under ${CVSROOT}. My malloc library is checked into `antaire/lib/malloc`.

`vendortag` is a *release tag* that the vendor assigned to the files. If you are the vendor then put whatever you want there: (PRT_INITIAL, MALLOC_1_01, etc);

`Releasetag` is your local tag for this copy of the files. (PRT_1, malloc_1_01, etc);

0.5 A "real-life" example of the usage of CVS.

cd to your work directory:

```
I do cd ~/src/work
```

Execute `cvs co module` where module is an entry from the modules file (see above):

Say `cvs co malloc` to get my malloc library. It will create the sub-directory `malloc` and will load the files into this new directory.

Edit the files to your heart's content.

If you add any new files to the directory that you want the repository to know about you need to do a

```
cvs add file1 [file2 ...]
```

If you remove any files you need to do a

```
cvs remove file1 [file2 ...]
```

If you rename you need to do a combination remove and then add.

Execute `cvs update` to pull in the changes from the repository that others made. It will resolve conflicts semi-automatically. It will tell you about the files it updates. U means updated, C means there was a conflict that it could not automatically resolve. You need to edit the file by hand, look for the `<<<<<` and `>>>>>` lines and figure out how the file should look.

Execute `cvs commit` inside the directory you checked out to papply your changes to the repository so others can use them (if they have the module in question checked out already, they need to do a `cvs update` to see your changes).

When you are done with the files (for the time being) you `cd ..` to the above directory and do a `cvs release [-d] module-name` which will *check-in* the files. The optional `-d` will remove the directory and files from your work directory when it is done releasing them.

The release command will inform you if your made modifications to the files and if there are files it doesn't know about that you may have forgotten to add. watch for `? file` lines printed. You may have to stop the release and commit or `cvs add/remove` the files.

WARNING: `release -d` is unrecoverable. Make sure that you take your time here. Fortunately, `cvs release` asks whether you really want to do this before doing anything.

0.6 How to get more information about CVS commands.

All CVS commands take a `-H` option to give help:

`cvs -H` shows you the options and commands in cvs.

`cvs history -H`
shows you the options for CVS history.

All the CVS commands mentioned also accept a flag `-n`, that doesn't do the action, but lets you see what would happen. For instance, you can use `cvs -n update` to see which files would be updated.

To get more information, see the manual page `man cvs` for full (and much more complicated) details.

A basic knowledge of the Revision Control System (RCS) on which CVS is layered may also be of some assistance. see `man co` or `man ci` for more details.

0.7 What's the difference between the two.

0.7.1 Modules as Collections of files

One of the strong points about CVS is that it not only lets you retrieve old versions of specific *files*, you can collect files (or directories of files) into "modules" and operate on an entire module at once. The RCS history files of all modules are kept at a central place in the file system hierarchy. When someone wants to work an a certain module he just types `cvs checkout malloc` which causes the directory '`malloc`' to be created and populated with the files that make up the malloc project.

With `cvs tag malloc-1.0` you can give the symbolic tag `malloc-1.0` to all the versions of the file in the malloc module. Later on, you can do `cvs checkout -r malloc-1.0 malloc` to retrieve the files that make up the 1.0 release of `malloc`. You can even do things like `cvs diff -c -r malloc-1.0 -r malloc-1.5` to get a context diff of all files that have changed between release 1.0 and release 1.5!

0.7.2 No locking

If you work in a group of programmers you have probably often wanted to edit the function realloc() in '`alloc.c`', but Joe had locked '`alloc.c`' because he is editing free().

CVS does not lock files. Instead, both you and Joe can edit '`alloc.c`'. The first one to check in it won't realize that the other have been editing it. (So if you are quicker than Joe you wont have any trouble at all). Poor Joe just have to do `cvs update alloc.c` to merge in your changes in his copy of the file. As long as you changing different sections of the file the merge is totally automatic. If you change the same lines you will have to resolve the conflicts manually.

0.7.3 Friendlier user interface

If you don't remember the syntax of `cvs diff` you just type `cvs -H diff` and you will get a short description of all the flags. Just `cvs -H` lists all the sub-commands. I find the commands less cryptic than the RCS equivalents. Compare `cvs checkout module` (which can be abbreviated to `cvs co module`) with `co -l RCS/*,v` (or whatever it is you are supposed to say – it's a year since I used RCS seriously).

0.8 Some questions about CVS and some (often weak) answers.

When I say `cvs checkout module/sub-directory` and then `cvs release module/sub-directory` it says unknown module name. why?

- because module/sub-directory is not a module in the modules file.
- `cvs release module` *should* work with this.

Because of incorrect releasing of directories, I noticed that `cvs history` reports that modules are still checked out. how do I correct this?

- by editing $CVSROOT/CVSROOT/history VERY carefully
- I do not know the correct way of doing this but the O lines are for checking Out modules. Removing the last O line that corresponds to the module in question may work.

If I just made a typo and started to check out a tree I did not want to. Can I hit control-c? What are the ramifications?

- probably some files are checked out while others are not
- it is *not* a good idea to do this. Unless you are pushing the limit on your disk space or something, just wait till it is done and then release the directory.
- don't know exactly

I screw up and removed the tree that I was about to start working on. How do I tell CVS that I want to release it if I don't have it anymore?

- maybe you can't
- you need to edit $CVSROOT/CVSROOT/history **very** carefully to fix this problem (see above)
-

What is the proper way to configure CVS for multi-user operations? What sort of file directory modes are appropriate aside from 770 modes everywhere. Any setgid support?

- first, CVS should NOT be installed as setuid or setgid!!! It is not designed to support these functions and will thus be a security problem. (the same applied to RCS)
- set all your $CVSROOT repository directories to a common group (src would make sense). Setgid on the directories would also be good if your system supports 'chmod g+s dir'
- normal 770 or ug+rwx directory modes in your repository directories should work. Have your system administrator execute something to the affect of:

```
find $CVSROOT -type d -exec chgrp src {} \;
find $CVSROOT -type d -exec chmod 770 {} \;

and (if available)

find $CVSROOT -type d -exec chmod g+s {} \;
```

- your default umask should be set to 027
- make sure all your developers are in the src group.

I am constantly running into different library modules to fix problems or add need features. This may not be a good practice but how does CVS fit in this scenario. Should I checkout the modules I need and once and a while commit them?

– yes. as long as you realize the commit will make your changes seen by everyone. make sure to run the appropriate tests on them, etc.

I have 4 releases of my debug malloc subsystem. They are 1.01, 1.02, 1.03, 1.05. Should I:

```
cvs import -b 1.1.1 -m 'Malloc 1.01' MALLOC_1_01 malloc_1_01
cvs import -b 1.1.2 -m 'Malloc 1.02' MALLOC_1_02 malloc_1_02
cvs import -b 1.1.3 -m 'Malloc 1.03' MALLOC_1_03 malloc_1_03
cvs import -b 1.1.5 -m 'Malloc 1.05' MALLOC_1_05 malloc_1_05
```

for each set of files? Are these sane incantations?

– not really. `cvs import` may not be the right may to do this. you should `cvs import` the *first* set of files then for each release you should:
  - copy in the new version
  - `cvs add/remove` the files that have been added/deleted
  - `cvs commit` the new version
  - `cvs tag` the files with the appropriate release-tag
  - repeat

When I use the above multiple-import method and I say `cvs checkout malloc` I do not just get the files that correspond to version 1.05 . I get all of the files. I have to say `cvs checkout -r malloc_1_05 malloc` to do this. Is this correct or do I need to `cvs remove` the files that have been removed between the different versions?

– yes, you need to `cvs add/remove` files to have the repository know about them. the files that are removed are placed in the attic so that old-revisions can find them.

Is there a CVS feature to tell me what files have changed, what are new what have been removed from the current directory?

– when you do a `cvs update`, CVS will inform you of the changed/modified files with a M, updated files with a U, and removed files with an R. It will show you files that *might* need to be added with a ?.

– in the contrib directory under the CVS release there exists a cvscheck script which gives you some of this functionality.

If I had a file at one point, but I did a `cvs remove` on it, and now I need to recreate it. How can I re-add it? It complains that it is in the Attic. Can it live there for old versions but also exist the in normal repository for newer versions?

– No it cannot live in both the Attic and the repository.

– I believe you can move the file out of the Attic directory by hand. this seems to work fine.

– **WARNING**: playing with the repository files is a potentially dangerous situation. You may wish to let your system administrator take care of this. If you are doing it yourself, make sure to use `cp -i` and `mv -i` or just watch your back and take your time.

0.9 Some questions for people more experiences with CVS.

CVS handled binary files but it seems to corrupt them once and while?

- CVS uses RCS and RCS looks for certain sequences of characters like \$Id\$ and replaces them with version information. So what if a keyword sequence just happens to occur in a binary file since any sequence of ASCII characters is possible? CVS will update any keyword sequences it finds and will corrupt any binary files in which these sequences occur.

  So keyword substitution must be prevented in binary files by added info to the rcs files for each binary file. I also prevent it in any files in which I don't plan to insert rcs keywords just to reduce checkout time. This can be done in the repository using the rcs command:

  ```
  rcs -ko <filenames,v>
  ```

- It might also be because someone is using RCS version 4.x which can't handle binary files. It will corrupt the history files! For this reason it may be necessary to install CVS with a full path to the RCS and diff programs so that it doesn't depend on $PATH which of them are run.

How can I use the \$Log\$ keyword with CVS with Objective-C and // comments?

- RCS (and therefore CVS) doesn't know about Objective-C "//" comments by default. I must inform RCS so that when it expands its \$Log\$ keyword, it will prepend the proper comment characters.

  ```
  rcs -c"// " <filenames,v>
  ```

0.10 List of some of the folks that have provided feedback.

- Scott Michel <scottm@intime.intime.COM>
- Robert Lupton the Good <rhl@astro.Princeton.EDU> (texinfo hacker)
- Steven Pemberton, CWI, Amsterdam <Steven.Pemberton@cwi.nl>
- Per Cederqvist <ceder@lysator.liu.se>
- Art Isbell <isbell@cats.UCSC.EDU>
- Paul Eggert <eggert@twinsun.com> (RCS god)
- Brian Berliner <Brian.Berliner@Central.Sun.COM> (CVS god)

0.11 A patch for mkmodules to make it know about cvsignore

```
*** mkmodules.c~        Tue Mar 31 16:56:20 1992
--- mkmodules.c Sat Sep 19 14:30:19 1992
***************
*** 183,188 ****
--- 183,202 ----
            "a %s file can be used to configure 'cvs commit' checking",
            CVSROOTADM_COMMITINFO);
      (void) unlink_file (temp);
+
+     /*
+      * Now, check out the "cvsignore" file, so that it is always up-to-date
+      * in the CVSROOT directory.
+      */
```

```
+     make_tempfile (temp);
+     if (checkout_file (CVSROOTADM_IGNORE, temp) == 0)
+       rename_rcsfile (temp, CVSROOTADM_IGNORE);
+     else
+       error (0, 0,
+           "a %s file can be used to list files that cvs should ignore",
+           CVSROOTADM_IGNORE);
+     (void) unlink_file (temp);
+
      return (0);
  }
```

Table of Contents