CVS Cheat Sheet

Basic CVS

Specifying Repository

Using the global option `-d`

```
cvs -d :method:user@host:/cvs/dir
```

Setting the environment variable `CVSROOT`

```
export
CVSROOT=:method:user@host:/cvs/dir
```

Importing a Project

To import the project `pirogi` into CVS

```
cvs import -m "message" myproj vendortag
releasetag
```

Checking Out

To check out a working copy of `myproj`

```
cvs co myproj
```

Finding Out What You Did

To find out what modifications have been made

```
cvs diff -c
```

Updating

To update your local copy with changes made to the repository

```
cvs update
```

Committing

To send modifications in `file1` and `file2` to the repository

```
cvs commit -m "message" file1 file2
```

Implicit Arguments

In most CVS commands, if you do not specify an argument, the command acts recursively on all the files in the current directory.

Conflict Resolution

Conflicts occur when same portion of the code is modified and committed by different users. The code in conflict is shown delimited by markers. Keep the required code, remove the unwanted code and the makers.

```
<<<<<<< (filename)
  The code you wrote.
=======
  Code in the repository.
>>>>>>> (latest revision in repository)
```

Finding Out Who Did What

To find out what changes have been made on `myfile` and by whom

```
cvs log myfile
```

Examining Changes

To find out difference between revision `rev1` and current working copy of the file `myfile`

```
cvs diff -c -r rev1 myfile
```

To find out difference between revision `rev1` and `rev2` of the file `myfile`

```
cvs diff -c -r rev1 -r rev2 myfile
```

Reverting Changes

To revert `file` from `rev1` to `rev2`

```
cvs update -j rev1 -j rev2
```

File Operations

Adding Files

To add the file `myfile`, use the `add` command followed by a commit

```
cvs add myfile
cvs commit -m "Added myfile."
```

Adding Directories

To add the directory `mydir`

```
cvs add mydir
```

Removing Files

To remove the file `myfile`, remove the file first and then use the `remove` command

```
cvs remove myfile
```

Removing Directories

To remove a directory, first remove all the files in it using the `remove` command, and then prune the empty directory using

```
cvs update -P
```

Renaming Files/Directories

To rename a file/directory, remove the old file from CVS and add the new file. `:-)`

Tags

Creating Tag

To tag the working copy with `mytag`

```
cvs tag -c mytag
```

You must have checked-in all required modifications before running this command.

Deleting Tags

To delete the tag `mytag`

```
cvs tag -d mytag
```

Using Tags

Tags can be used wherever a revision no. can be used, especially with the `-r` command option.

Checking out by Tags

To check out the project `myproject` by tag `mytag`

```
cvs co -r mytag myproject
```

When You Forget to Tag

Most commands also work with dates with the `-D` command option. For example, to check out the project `myproject` as it was on 14th Sept, 2004

```
cvs co -D 2004-9-14 myproject
```

Date Formats

Accepted date formats include

```
2004-9-14
14 Sep 2004
13 Sep 2004 23:10
14/9/2004
5 hours ago
5 days ago
```

Keyword Substitution

Frequently Used Keywords

`$Id$` RCS filename, revision number, date, author, state and locker(if locked)

`$Name$` Tag name used to check out the file.

`$Log$` Accumulates commit messages for a source file.

Setting Substitution Modes

Each file has a stored default substitution mode, and each working copy of a file also has a substitution mode. The former is set by `-k` option to `add` and `import`. The latter is set by the `-k` option to `checkout` and `update`.

Available Modes

`-kkv` Default form `$Revision:  2.3$`

`-kkvl` Like above, except that a locker's name is always inserted

`-kk` Only names `$Revision:  $`

`-ko` Original keyword string at the time of checkin

`-kb` Like above, but also inhibit EOL conversion

`-kv` Only values `2.3`

Public Distribution

To create a package for distributing `1.2` version of `myproj`

```
cvs export -d myproj-1.2 -r MYPROJ_1_2
myproj
```

`myproj-1.2` is the directory into which the files will be checked out.

Branches

Creating Branches

To branch off into `mybranch` from the working copy

```
cvs tag -b mybranch
```

Accessing Branches

Branch names can appear wherever tag names can appear. The branch name always refers to the latest revisions at the head of the branch.

Merging Branches

To merge changes made on the branch `mybranch` into the working copy

```
cvs update -kk -j mybranch
```

The `-kk` option is used to avoid conflicts due to keyword expansion. Note that this should not be used with binary files.

Multiple Merges

If the branch `mybranch` grows after you have merged it at `mymerge`, you can merge the grown part using

```
cvs update -j mymerge -j mybranch
```

Tracking Third-Party Sources

Importing the Source

To import version 2.3 sources of the project `theirproj` provided by the vendor `Them`

```
cvs import -m "Import of TheirProj 2.3"
theirproj Them THEIRPROJ_2_3
```

If the third party also uses CVS, use the `-ko` option.

When You Receive New Versions

You must import again from the new sources. If the new version is 2.4 then

```
cvs import -m "Import of TheirProj 2.4"
theirproj Them THEIRPROJ_2_4
```

And then merge the changes in the new version into the main trunk

```
cvs co -j THEIRPROJ_2_3 -j THEIRPROJ_2_4
theirproj
```

`$Revision: 1.5 $ $Date: 2005/02/20 04:40:02 $.`