User's Guide to PCL-CVS — The Emacs Front-End to CVS

Per Cederqvist
Stefan Monnier

1 About PCL-CVS

PCL-CVS is a front-end to CVS versions 1.9 and later. It concisely shows the present status of a checked out module in an Emacs buffer and provides single-key access to the most frequently used CVS commands. For Emacs users accustomed to VC, PCL-CVS can be thought of as a replacement for VC-dired (see section "Dired under VC" in *The GNU Emacs Manual*) specifically designed for CVS.

PCL-CVS was originally written many years ago by Per Cederqvist who proudly maintained it until January 1996, at which point he released the beta version 2.0b2 and passed on the maintainership to Greg A Woods. Development stayed mostly dormant for a few years during which version 2.0 never seemed to be able to leave the "beta" stage while a separate XEmacs version was slowly splitting away. In late 1998, Stefan Monnier picked up development again, adding some major new functionality and taking over the maintenance.

As of Emacs 21, PCL-CVS is part of the standard Emacs distribution.

1.1 Contributors to PCL-CVS

Contributions to the package are welcome. I have limited time to work on this project, but I will gladly add any code that you contribute to me to this package (see Chapter 9 [Bugs], page 18).

The following persons have made contributions to PCL-CVS.

- Brian Berliner wrote CVS, together with some other contributors. Without his work on CVS this package would be useless...
- Per Cederqvist wrote most of the otherwise unattributed functions in PCL-CVS as well as all the documentation.
- Inge Wallin (`inge@lysator.liu.se`) wrote the skeleton of '`pcl-cvs.texi`', and gave useful comments on it. He also wrote the files '`elib-node.el`' and '`compile-all.el`'. The file '`cookie.el`' was inspired by Inge.
- Linus Tolke (`linus@lysator.liu.se`) contributed useful comments on both the functionality and the documentation.
- Jamie Zawinski (`jwz@jwz.com`) contributed '`pcl-cvs-lucid.el`', which was later renamed to '`pcl-cvs-xemacs.el`'.
- Leif Lonnblad contributed RCVS support (since superceded by the new remote CVS support).
- Jim Blandy (`jimb@cyclic.com`) contributed hooks to automatically guess CVS log entries from '`ChangeLog`' contents, and initial support of the new Cygnus / Cyclic remote CVS, as well as various sundry bug fixes and cleanups.
- Jim Kingdon (`kingdon@cyclic.com`) contributed lots of fixes to the build and installation procedure.
- Greg A. Woods (`woods@weird.com`) contributed code to implement the use of per-file diff buffers, and vendor join diffs with emerge and ediff, as well as various and sundry bug fixes and cleanups.
- Greg Klanderman (`greg.klanderman@alum.mit.edu`) implemented toggling of marked files, setting of CVS command flags via prefix arguments, updated the XEmacs support, updated the manual, and fixed numerous bugs.

- Stefan Monnier (`monnier@cs.yale.edu`) added a slew of other features and introduced even more new bugs. If there's any bug left, you can be sure it's his.
- Masatake YAMATO (`masata-y@is.aist-nara.ac.jp`) made a gracious contribution of his cvstree code to display a tree of tags which was later superseded by the new `cvs-status-mode`.

Apart from these, a lot of people have sent us suggestions, ideas, requests, bug reports and encouragement. Thanks a lot! Without you there would be no new releases of PCL-CVS.

1.2 Installation

As mentioned above, PCL-CVS comes bundled with Emacs version 21.1 and later. If you're using Emacs 20, you can download an older version of PCL-CVS from `ftp://flint.cs.yale.edu/pub/monnier/pcl-cvs`. That version also works on XEmacs.

If you are running XEmacs 21.0 or later, PCL-CVS is available in pre-compiled package form. Please refer to the XEmacs manual for instructions regarding package selection and installation. Currently, that PCL-CVS package also requires you to have installed the '`xemacs-base`', '`elib`', and '`dired`' packages.

If you have TeX installed at your site, you can make a typeset manual from '`pcl-cvs.texi`'.

1. If PCL-CVS came with the Emacs distribution, type *`make pcl-cvs.dvi`* in the '`man`' subdirectory of the Emacs source tree.
2. Alternatively, run TeX by typing *`texi2dvi pcl-cvs.texi`*.
3. Convert the resulting device independent file '`pcl-cvs.dvi`' to a form which your printer can output and print it. If you have a PostScript printer, there is a program, `dvi2ps`, which does. There is also a program which comes together with TeX, `dvips`, which you can use.

2 Getting started

This document assumes that you know what CVS is, and that you at least know the fundamental concepts of CVS. If that is not the case, you should read the CVS documentation. Type *`info -f cvs`* or *`man cvs`*.

PCL-CVS is only useful once you have checked out a module. So before you invoke it, you must have a copy of a module somewhere in the file system.

You can invoke PCL-CVS by typing *`M-x cvs-examine`* ⟨RET⟩. You can also invoke it via the menu bar, under '`Tools`'. Or, if you prefer, you can also invoke PCL-CVS by simply visiting the CVS administrative subdirectory of your module, with a prefix argument. For example, to invoke PCL-CVS in a separate frame, type *`C-u C-x 5 f ~/my/project/CVS`* ⟨RET⟩.

The function `cvs-examine` will ask for a directory. The command '`cvs -n update`' will be run in that directory. (It should contain files that have been checked out from a CVS archive.) The output from `cvs` will be parsed and presented in a table in a buffer called '`*cvs*`'. It might look something like this:

```
Repository : /usr/CVSroot
Module     : test
Working dir: /users/ceder/FOO/test


In directory .:
           Need-Update            bar
           Need-Update            file.txt
           Modified               namechange
           Need-Update            newer
In directory sub:
           Modified               ChangeLog

--------------------- End ---------------------
-- last cmd: cvs -f -z6 -n update -d -P --
```

In this example, your repository is in '`/usr/CVSroot`' and CVS has been run in the directory '`/users/ceder/FOO/test`'. The three files ('`bar`', '`file.txt`' and '`newer`') that are marked with '`Need-Update`' have been changed by someone else in the CVS repository. Two files ('`namechange`' and '`sub/ChangeLog`') have been modified locally, and need to be checked in.

You can move the cursor up and down in the buffer with *`C-n`* and *`C-p`* or *`n`* and *`p`*. If you press *`c`* on one of the '`Modified`' files, that file will be checked in to the CVS repository. See Section 5.6 [Committing changes], page 9. You can also press *`O`* to update any of the files that are marked '`Need-Update`'. You can also run *`M-x cvs-update`* ⟨RET⟩ (bound to *`M-u`* in the '`*cvs*`' buffer) to update all the files.

You can then press `=` to easily get a '`diff`' between your modified file and the base version that you started from, or you can press *`l`* to get the output from '`cvs log`'. Many more such commands are available simply by pressing a key (see Section 5.8 [Getting info about files], page 10).

3 Buffer contents

The display contains several columns, some of which are optional. These columns are, from left to right:

- Optionally, the head revision of the file. This is the latest version found in the repository. It might also contain (instead of the head revision) a sub status which typically gives further information about how we got to the current state, for example '`patched`', '`merged`', . . .
- An asterisk when the file is *marked* (see Chapter 4 [Selected files], page 6).
- The actual status of the file wrt the repository. See below.
- Optionally, the base revision of the file. This is the version which the copy in your working directory is based upon.
- The file name.

The '`file status`' field can have the following values:

'`Modified`'
: The file is modified in your working directory, and there was no modification to the same file in the repository. This status can have the following substatus:

  '`merged`' The file was modified in your working directory, and there were modifications in the repository as well, but they were merged successfully, without conflict, in your working directory.

'`Conflict`'
: A conflict was detected while trying to merge your changes to *file* with changes from the repository. *file* (the copy in your working directory) is now the output of the `rcsmerge` command on the two versions; an unmodified copy of your file is also in your working directory, with the name '`.#`*file*`.`*version*', where *version* is the RCS revision that your modified file started from. See Section 5.13 [Viewing differences], page 12, for more details.

  A conflict can also come from a disagreement on the existence of the file rather than on its content. This case is indicated by the following possible substatus:

  '`removed`' The file is locally removed but a new revision has been committed to the repository by someone else.

  '`added`' The file is locally added and has also been added to the repository by someone else.

  '`modified`'
  The file is locally modified but someone else has removed it from the repository.

'`Added`' The file has been added by you, but it still needs to be checked in to the repository.

'`Removed`' The file has been removed by you, but it still needs to be checked in to the repository. You can resurrect it by typing `a` (see Section 5.9 [Adding and removing files], page 10).

‘`Unknown`’
: A file that was detected in your directory, but that neither appears in the repository, nor is present on the list of files that CVS should ignore.

‘`Up-to-date`’
: The file is up to date with respect to the version in the repository. This status can have a substatus of:

  ‘`added`’
  : You have just added the file to the repository.

  ‘`updated`’
  : The file was brought up to date with respect to the repository. This is done for any file that exists in the repository but not in your source, and for files that you haven’t changed but are not the most recent versions available in the repository.

  ‘`patched`’
  : The file was brought up to date with respect to the remote repository by way of fetching and applying a patch to the file in your source. This is equivalent to ‘`updated`’ except that CVS decided to use a hopefully more efficient method.

  ‘`committed`’
  : You just committed the file.

‘`Need-Update`’
: Either a newer version than the one in your source is available in the repository and you have not modified your checked out version, or the file exists in the repository but not in your source. Use ‘`cvs-mode-update`’ bound to *O* to update the file.

‘`Need-Merge`’
: You have modified the checked out version of the file, and a newer version is available in the repository. A merge will take place when you run a ‘`cvs-update`’.

‘`Missing`’
: The file has been unexpectedly removed from your working directory although it has not been ‘`cvs remove`’d.

4 Selected files

Many of the commands work on the current set of *selected* files which can be either the set of marked files (if any file is marked and marks are no ignored) or whichever file or directory the cursor is on.

If a directory is selected but the command cannot be applied to a directory, then it will be applied to the set of files under this directory which are in the '`*cvs*`' buffer.

Furthermore, each command only operates on a subset of the selected files, depending on whether or not the command is *applicable* to each file (based on the file's status). For example, `cvs-mode-commit` is not applicable to a file whose status is '`Need-Update`'. If it should happen that PCL-CVS guesses the applicability wrong, you can override it with the special prefix `cvs-mode-force-command` normally bound to *`M-f`* (and file a bug report). The applicability rule can be slightly changed with `cvs-allow-dir-commit` and `cvs-force-dir-tag`.

By default, marks are always in effect (you may change this, however, by setting the variable `cvs-default-ignore-marks`) except for the commands that '`tag`' or '`diff`' a file (which can be changed with the variable `cvs-invert-ignore-marks`).

In addition, you may use the special prefix `cvs-mode-toggle-marks` normally bound to ⟨T⟩ to toggle the use of marks for the following command.

This scheme might seem a little complicated, but once one gets used to it, it is quite powerful.

For commands to mark and unmark files, see Section 5.5 [Marking files], page 9.

5 Commands

This chapter describes all the commands that you can use in PCL-CVS.

5.1 Entering PCL-CVS

Most commands in PCL-CVS require that you have a '`*cvs*`' buffer. The commands that you use to get one are listed below. For each, a '`cvs`' process will be run, the output will be parsed by PCL-CVS, and the result will be printed in the '`*cvs*`' buffer (see Chapter 3 [Buffer contents], page 4, for a description of the buffer's contents).

`M-x cvs-update`
: Run a '`cvs update`' command. You will be asked for the directory in which the '`cvs update`' will be run.

`M-x cvs-examine`
: Run a '`cvs -n update`' command. This is identical to the previous command, except that it will only check what needs to be done but will not change anything. You will be asked for the directory in which the '`cvs -n update`' will be run.

`M-x cvs-status`
: Run a '`cvs status`' command. You will be asked for the directory in which the '`cvs status`' will be run.

`M-x cvs-checkout`
: Run a '`cvs checkout`' command. You will be asked for the directory in which the '`cvs update`' will be run and the module to be checked out.

`M-x cvs-quickdir`
: Populate the '`*cvs*`' buffer by just looking at the '`CVS/Entries`' files. This is very much like `cvs-examine` except that it does not access the CVS repository, which is a major advantage when the repository is far away. But of course, it will not be able to detect when a file needs to be updated or merged.

The first four of those commands are also reachable from the menu bar under '`Tools->PCL-CVS`'. Finally, an alternative way is to visit the CVS administrative subdirectory in your work area with a simple prefix argument. For example *`C-u C-x C-f ~/my/work/CVS`* ⟨RET⟩. This by default runs `cvs-quickdir` but the specific behavior can be changed with `cvs-dired-action` and `cvs-dired-use-hook`.

By default, the commands above will descend recursively into subdirectories. You can avoid that behavior by including '`-l`' in the flags for the command. These flags can be set by giving a prefix argument to the command (e.g., by typing *`C-u M-x cvs-update`* ⟨RET⟩ *`-l`* ⟨RET⟩).

5.2 Setting flags for CVS commands

This section describes the convention used by nearly all PCL-CVS commands for setting optional flags sent to CVS. A single *`C-u`* prefix argument is used to cause the command to prompt for flags to be used for the current invocation of the command only. Two `C-u` prefix arguments are used to prompt for flags which will be set permanently, for the current invocation and all that follow, until the flags are changed, or unless temporary flags are set which override them.

Perhaps an example or two is in order. Say you are about to add a binary file to the repository, and want to specify the flags '`-kb`' to '`cvs add`'. You can type `C-u a -kb` ⟨RET⟩, enter the description, and the file will be added. Subsequent '`cvs add`' commands will use the previously prevailing flags.

As a second example, say you are about to perform a diff and want to see the result in unified diff format, i.e. you'd like to pass the flag '`-u`' to both '`cvs diff`' and '`diff`'. You'd also like all subsequent diffs to use this flag. You can type `C-u C-u = -u` ⟨RET⟩ and the diff will be performed, and the default flags will be set to `("-u")`. You can of course override this flag for a single diff by using a single `C-u` prefix argument.

In addition to this, some commands can take *special prefix* arguments. These work as follows: When called with a `C-u` prefix, the user is prompted for a new value of the special prefix and the special prefix is activated for the next command. When called without the `C-u` prefix, the special prefix is re-activated (with the same value as last time) for the next command. Calling the prefix command again when it's already activated deactivates it. Calling it with the `C-u C-u` prefix activates it for all subsequent commands until you deactivate it explicitly. The special prefixes are:

`T`
: Toggles whether or not marks will be active in the next command.

`b`
: Provide the next command with a branch (can be any version specifier) to work on.

`B`
: Secondary branch argument. Only meaningful if `b` is also used. It can be used to provide a second branch argument to `cvs-mode-diff` or to `cvs-mode-update`.

`M-f`
: Forces the next command to apply to every selected file rather than only to the ones PCL-CVS thinks are relevant.

5.3 Updating the '`*cvs*`' buffer

The following commands can be used from within the '`*cvs*`' buffer to update the display:

`M-u`
: Runs the command '`cvs-update`'.

`M-e`
: Runs the command '`cvs-examine`'.

`M-s`
: Runs the command '`cvs-status`'.

In addition to the above commands which operate on the whole module, you can run the equivalent CVS command on just a subset of the files/directories with these keys:

`O`
: Runs `cvs-mode-update` on the selected files. When run on the top-level directory, this is equivalent to `M-u`.

`e`
: Runs `cvs-mode-examine` on the selected files. When run on the top-level directory, this is equivalent to `M-e`.

s Runs `cvs-mode-status` on the selected files. When run on the top-level directory, this is equivalent to *M-s*, except that CVS output will be shown in a '`*cvs-info*`' buffer that will be put in '`cvs-status-mode`'.

5.4 Movement Commands

You can use most normal Emacs commands to move forward and backward in the buffer. Some keys are rebound to functions that take advantage of the fact that the buffer is a PCL-CVS buffer:

⟨SPC⟩
n These keys move the cursor one file forward, towards the end of the buffer (`cvs-mode-next-line`).

p This key moves one file backward, towards the beginning of the buffer (`cvs-mode-previous-line`).

5.5 Marking files

PCL-CVS works on a set of *selected files* (see Chapter 4 [Selected files], page 6). You can mark and unmark files with these commands:

m This marks the file that the cursor is positioned on. If the cursor is positioned on a directory all files in that directory are marked. (`cvs-mode-mark`).

u Unmark the file that the cursor is positioned on. If the cursor is on a directory, all files in that directory are unmarked. (`cvs-mode-unmark`).

M Mark *all* files in the buffer (`cvs-mode-mark-all-files`).

M-⟨DEL⟩ Unmark *all* files (`cvs-mode-unmark-all-files`).

⟨DEL⟩ Unmark the file on the previous line, and move point to that line (`cvs-mode-unmark-up`).

⟨%⟩ Mark all files matching a regular expression (`cvs-mode-mark-matching-files`).

⟨T⟩ Toggle use of marks for the next command (`cvs-mode-toggle-marks`).

5.6 Committing changes

Committing changes basically works as follows:

1. After having selected the files you want to commit, you type either *c* or *C* which brings up a special buffer '`*cvs-commit*`'.
2. You type in the log message describing the changes you're about to commit (see Chapter 6 [Log Edit Mode], page 14).
3. When you're happy with it, you type *C-c C-c* to do the actual commit.

There's no hidden state, so you can abort the process or pick it up again at any time.

The set of files actually committed is really decided only during the very last step, which is a mixed blessing. It allows you to go back and change your mind about which files to commit, but it also means that you might inadvertently change the set of selected files. To reduce the risk of error, `C-c C-c` will ask for confirmation if the set of selected files has changed between the first step and the last. You can change this last detail with `log-edit-confirm`.

As for the difference between `c` (i.e. `cvs-mode-commit`) and `C` (i.e. `cvs-mode-commit-setup`) is that the first gets you straight to '`*cvs-commit*`' without erasing it or changing anything to its content, while the second first erases '`*cvs-commit*`' and tries to initialize it with a sane default (it does that by either using a template provided by the CVS administrator or by extracting a relevant log message from a '`ChangeLog`' file).

If you are editing the files in your Emacs, an automatic '`revert-buffer`' will be performed. (If the file contains '`$Id$`' keywords, '`cvs commit`' will write a new file with the new values substituted. The auto-revert makes sure that you get them into your buffer). The revert will not occur if you have modified your buffer, or if '`cvs-auto-revert`' is set to '`nil`'.

5.7 Editing files

There are currently three commands that can be used to find a file (that is, load it into a buffer and start editing it there). These commands work on the line that the cursor is situated at. They always ignore any marked files.

`f` — Find the file that the cursor points to (`cvs-mode-find-file`). If the cursor points to a directory, run `dired` on that directory; See Info file '`emacs`', node '`Dired`'.

`o` — Like `f`, but use another window (`cvs-mode-find-file-other-window`).

`A` — Invoke '`add-change-log-entry-other-window`' to edit a '`ChangeLog`' file. The '`ChangeLog`' file will be found in the directory of the file the cursor points to, or in a parent of that directory. (`cvs-mode-add-change-log-entry-other-window`).

5.8 Getting info about files

`l` — Call the command `cvs-mode-log` which runs '`cvs log`' on all selected files, and show the result in a temporary buffer '`*cvs-info*`' (see Chapter 7 [Log View Mode], page 15).

`s` — Call the command `cvs-mode-status` which runs '`cvs status`' on all selected files, and show the result in a temporary buffer '`*cvs-info*`'.

5.9 Adding and removing files

The following commands are available to make it easy to add files to and remove them from the CVS repository.

`a` — Add all selected files. This command can be used on '`Unknown`' files (see Chapter 3 [Buffer contents], page 4). The status of the file will change to '`Added`', and you will have to use *c* ('`cvs-mode-commit`' see Section 5.6 [Committing changes], page 9), to really add the file to the repository.

This command can also be used on '`Removed`' files (before you commit them) to resurrect them.

The command that is run is `cvs-mode-add`.

r This command removes the selected files (after prompting for confirmation). The files are deleted from your directory and (unless the status was '`Unknown`'; see Chapter 3 [Buffer contents], page 4) they will also be '`cvs remove`'d. If the files' status was '`Unknown`' they will disappear from the buffer. Otherwise their status will change to '`Removed`', and you must use *c* ('`cvs-mode-commit`', see Section 5.6 [Committing changes], page 9) to commit the removal.

The command that is run is `cvs-mode-remove-file`.

5.10 Undoing changes

U If you have modified a file, and for some reason decide that you don't want to keep the changes, you can undo them with this command. It works by removing your working copy of the file and then getting the latest version from the repository (`cvs-mode-undo-local-changes`.

5.11 Removing handled entries

x This command allows you to remove all entries that you have processed. More specifically, the lines for '`Up-to-date`' files (see Chapter 3 [Buffer contents], page 4) are removed from the buffer. If a directory becomes empty the heading for that directory is also removed. This makes it easier to get an overview of what needs to be done.

x invokes `cvs-mode-remove-handled`. If '`cvs-auto-remove-handled`' is set to non-`nil`, this will automatically be performed after every commit.

C-k This command can be used for lines that '`cvs-mode-remove-handled`' would not delete, but that you want to delete (`cvs-mode-acknowledge`).

5.12 Ignoring files

i Arrange so that CVS will ignore the selected files. The file names are added to the '`.cvsignore`' file in the corresponding directory. If the '`.cvsignore`' file doesn't exist, it will be created.

The '`.cvsignore`' file should normally be added to the repository, but you could ignore it as well, if you like it better that way.

This runs `cvs-mode-ignore`.

5.13 Viewing differences

`=`
`d =`
: Display a '`cvs diff`' between the selected files and the version that they are based on. (`cvs-mode-diff`).

`d b`
: If CVS finds a conflict while merging two versions of a file (during a '`cvs update`', see Section 5.3 [Updating the buffer], page 8) it will save the original file in a file called '`.#`*file*`.`*version*' where *file* is the name of the file, and *version* is the revision number that *file* was based on.

  With the `d b` command you can run a '`diff`' on the files '`.#`*file*`.`*version*' and '*file*'.

`d h`
: Display a '`cvs diff`' between the selected files and the head revision in the repository (the most recent version on the current branch) (`cvs-mode-diff-head`).

`d v`
: Display a '`cvs diff`' between the selected files and the head revision of the vendor branch in the repository. (`cvs-mode-diff-vendor`).

By default, '`diff`' commands ignore the marks. This can be changed with `cvs-invert-ignore-marks`.

5.14 Running ediff

`d e`
: This uses `ediff` (or `emerge`, depending on '`cvs-idiff-imerge-handlers`') to allow you to view diffs. If a prefix argument is given, PCL-CVS will prompt for a revision against which the diff should be made, else the default will be to use the BASE revision.

`d E`
: This command use `ediff` (or `emerge`, see above) to allow you to do an interactive 3-way merge.

  Note: When the file status is '`Conflict`', CVS has already performed a merge. The resulting file is not used in any way if you use this command. If you use the `q` command inside '`ediff`' (to successfully terminate a merge) the file that CVS created will be overwritten.

5.15 Updating files

`O`
: Update all selected files with status '`Need-update`' by running '`cvs update`' on them. (`cvs-mode-update`).

5.16 Tagging files

`t`
: Tag all selected files by running '`cvs tag`' on them (`cvs-mode-tag`). It's usually preferable to tag a directory at a time. Rather than selecting all files (which too often doesn't select all files but only the few that are displayed), clear the selection with `M-DEL` (`cvs-mode-unmark-all-files`), position the cursor on the directory you want to tag and hit `t`.

By default, '`tag`' commands ignore the marks. This can be changed with `cvs-invert-ignore-marks`. Also, by default '`tag`' can only be applied to directories, see `cvs-force-dir-tag` if you want to change this behavior.

5.17 Miscellaneous commands

`M-x cvs-mode-byte-compile-files`
: Byte compile all selected files that end in '`.el`'.

`M-x cvs-mode-delete-lock`
: This command deletes the lock files that the '`*cvs*`' buffer informs you about. You should normally never have to use this command, since CVS tries very carefully to always remove the lock files itself.

  You can only use this command when a message in the '`*cvs*`' buffer tells you so. You should wait a while before using this command in case someone else is running a `cvs` command.

  Also note that this only works if the repository is local.

`?`
`h`
: Show a summary of common command key bindings in the echo area (`cvs-help`).

`q`
: Quit PCL-CVS, killing the '`*cvs*`' buffer (`cvs-mode-quit`).

6 Editing a Log Message

Buffers for entering/editing log messages for changes which are about to be committed are put into Log Edit mode.

Sometimes the log buffer contains default text when you enter it, typically the last log message entered. If it does, mark and point are set around the entire contents of the buffer so that it is easy to kill the contents of the buffer with *C-w*.

If you work by writing entries in the '`ChangeLog`' (see ⟨undefined⟩ [(emacs)Change Log], page ⟨undefined⟩) and then commit the change under revision control, you can generate the Log Edit text from the ChangeLog using *C-a C-a* (*log-edit-insert-changelog*). This looks for entries for the file(s) concerned in the top entry in the ChangeLog and uses those paragraphs as the log text. This text is only inserted if the top entry was made under your user name on the current date. See ⟨undefined⟩ [(emacs)Change Logs and VC], page ⟨undefined⟩, for the opposite way of working—generating ChangeLog entries from the revision control log.

In the Log Edit buffer, *C-c C-f* (*M-x log-edit-show-files*) shows the list of files to be committed in case you need to check that.

When you have finished editing the log message, type *C-c C-c* to exit the buffer and commit the change.

7 Browsing a Log of Changes

Log View mode provides a few useful commands for navigating revision control log output. It is used for the output buffers of both `cvs-mode-log` and `vc-print-log`.

In this mode, *`n`* goes to the next message and *`p`* goes to the previous message and *`N`* and *`P`* go to the next and previous files, respectively, in multi-file output. With a numeric prefix argument, these commands move that many messages of files.

8 Customization

If you have an idea about any customization that would be handy but isn't present in this list, please tell me! For info on how to reach me, see Chapter 9 [Bugs], page 18.

'`cvs-auto-remove-handled`'
: If this variable is set to any non-`nil` value, '`cvs-mode-remove-handled`' will be called every time you check in files, after the check-in is ready. See Section 5.11 [Removing handled entries], page 11.

'`cvs-auto-remove-directories`'
: If this variable is set to any non-`nil` value, directories that do not contain any files to be checked in will not be listed in the '`*cvs*`' buffer.

'`cvs-auto-revert`'
: If this variable is set to any non-'`nil`' value any buffers you have that visit a file that is committed will be automatically reverted. This variable defaults to '`t`'. See Section 5.6 [Committing changes], page 9.

'`cvs-update-prog-output-skip-regexp`'
: The '`-u`' flag in the '`modules`' file can be used to run a command whenever a '`cvs update`' is performed (see `cvs(5)`). This regexp is used to search for the last line in that output. It is normally set to '`$`'. That setting is only correct if the command outputs nothing. Note that PCL-CVS will get very confused if the command outputs *anything* to `stderr`.

'`cvs-cvsroot`'
: This variable can be set to override '`CVSROOT`'. It should be a string. If it is set, then every time a `cvs` command is run, it will be called as '`cvs -d` *cvs-cvsroot*`...`'. This can be useful if your site has several repositories.

'`log-edit-require-final-newline`'
: When you enter a log message by typing into the '`*cvs-commit-message*`' buffer, PCL-CVS normally automatically inserts a trailing newline, unless there already is one. This behavior can be controlled via '`cvs-commit-buffer-require-final-newline`'. If it is '`t`' (the default behavior), a newline will always be appended. If it is '`nil`', newlines will never be appended. Any other value causes PCL-CVS to ask the user whenever there is no trailing newline in the commit message buffer.

'`log-edit-changelog-full-paragraphs`'
: If this variable is non-`nil`, include full '`ChangeLog`' paragraphs in the CVS log created by '`cvs-mode-changelog-commit`'. This may be set in the local variables section of a '`ChangeLog`' file, to indicate the policy for that '`ChangeLog`'.

  A *'`ChangeLog`' paragraph* is a bunch of log text containing no blank lines; a paragraph usually describes a set of changes with a single purpose, but perhaps spanning several functions in several files. Changes in different paragraphs are unrelated.

  You could argue that the CVS log entry for a file should contain the full '`ChangeLog`' paragraph mentioning the change to the file, even though it may mention other files, because that gives you the full context you need to understand the change. This is the behavior you get when this variable is set to `t`, the default.

On the other hand, you could argue that the CVS log entry for a change should contain only the text for the changes which occurred in that file, because the CVS log is per-file. This is the behavior you get when this variable is set to `nil`.

‘`cvs-sort-ignore-file`’
: If this variable is set to any non-‘`nil`’ value, the ‘`.cvsignore`’ file will always be sorted whenever you use ‘`cvs-mode-ignore`’ to add a file to it. This option is on by default.

8.1 Customizing Faces

PCL-CVS adds a few extra features, including menus, mouse bindings, and fontification the ‘`*cvs*`’ buffer. The faces defined for fontification are listed below:

‘`cvs-header-face`’
: used to highlight directory changes.

‘`cvs-filename-face`’
: used to highlight file names.

‘`cvs-unknown-face`’
: used to highlight the status of files which are ‘`Unknown`’.

‘`cvs-handled-face`’
: used to highlight the status of files which are handled and need no further action.

‘`cvs-need-action-face`’
: used to highlight the status of files which still need action.

‘`cvs-marked-face`’
: used to highlight the marked file indicator (‘`*`’).

9 Bugs (known and unknown)

If you find a bug or misfeature, don't hesitate to tell us! Send email to `bug-gnu-emacs@gnu.org` which is gatewayed to the newsgroup '`gnu.emacs.bugs`'. Feature requests should also be sent there. We prefer discussing one thing at a time. If you find several unrelated bugs, please report them separately. If you are running PCL-CVS under XEmacs, you should also send a copy of bug reports to `xemacs-beta@xemacs.org`.

If you have problems using PCL-CVS or other questions, send them to `help-gnu-emacs@gnu.org`, which is gatewayed to the '`gnu.emacs.help`' newsgroup. This is a good place to get help, as is `cvs-info@gnu.org`, gatewayed to '`gnu.cvs.help`'.

If you have ideas for improvements, or if you have written some extensions to this package, we would like to hear from you. We hope that you find this package useful!

Below is a partial list of currently known problems with PCL-CVS version 2.0.

Unexpected output from CVS
: Unexpected output from CVS may confuse PCL-CVS. It will create warning messages in the '`*cvs*`' buffer alerting you to any parse errors. If you get these messages, please send a bug report to the email addresses listed above. Include the contents of the '`*cvs*`' buffer, the output of the CVS process (which should be found in the '`*cvs-tmp*`' buffer), and the versions of Emacs, PCL-CVS and CVS you are using.

Function and Variable Index

This is an index of all the functions and variables documented in this manual.

Concept Index

This is an index of concepts discussed in this manual.

Key Index

This index includes an entry for each PCL-CVS key sequence documented in this manual.

Short Contents

Table of Contents