User's Guide
to
pcl-cvs - the Emacs Front-End to CVS

release 1.05-CVS-$Name: $

Per Cederqvist

last updated 20 Nov 1995

1 Installation

This section describes the installation of pcl-cvs, the GNU Emacs CVS front-end. You should install not only the elisp files themselves, but also the on-line documentation so that your users will know how to use it. You can create typeset documentation from the file '`pcl-cvs.texinfo`' as well as an on-line info file. The following steps are also described in the file '`INSTALL`' in the source directory.

1.1 Installation of the pcl-cvs program

1. Possibly edit the file '`Makefile`' to reflect the situation at your site. We say "possibly" because the version of pcl-cvs included with CVS uses a configuration mechanism integrated with the overall mechanisms used by the CVS build and install procedures. Thus the file `Makefile` will be generated automatically from the file `Makefile.in`, and it should not be necessary to edit it further.

   If you do have to edit the '`Makefile`', the only things you have to change is the definition of `lispdir` and `infodir`. The elisp files will be copied to `lispdir`, and the info file(s) to `infodir`.
2. Configure pcl-cvs.el

   There are a couple of pathnames that you have to check to make sure that they match your system. They appear early in the file '`pcl-cvs.el`'.

   NOTE: If your system is running emacs 18.57 or earlier you MUST uncomment the line that says:

   ```
   (setq delete-exited-processes nil)
   ```

   Setting `delete-exited-processes` to `nil` works around a bug in emacs that causes it to dump core. The bug was fixed in emacs 18.58.
3. Release 1.05 and later of pcl-cvs requires parts of the Elib library, version 1.0 or later. Elib is available via anonymous ftp from prep.ai.mit.edu in '`pub/gnu/elib-1.0.tar.gz`', and from a lot of other sites that mirror prep. Get Elib, and install it, before proceeding.

   NOTE: The version of pcl-cvs included with CVS includes a copy of Elib in the sub-directory '`elib`' under the '`contrib/pcl-cvs`' directory.
4. Type '`make install`' in the source directory. This will byte-compile all '`.el`' files and copy the '`*.elc`' files into the directory you specified in step 1.

   If you want to install the '`*.el`' files too, you can type '`make install-el`' to do so.

   If you only want to create the compiled elisp files, but don't want to install them, you can type '`make`' without parameters.

5. Edit the file 'default.el' in your emacs lisp directory (usually '/usr/gnu/lib/emacs/site-lisp' or something similar) and enter the contents of the file 'pcl-cvs-startup.el' into it. It contains a couple of auto-loads that facilitates the use of pcl-cvs.

1.2 Installation of the on-line manual.

1. Create the info file(s) 'pcl-cvs.info*' from 'pcl-cvs.texinfo' by typing 'make info'. If you don't have the program 'makeinfo' you can get it by anonymous ftp from e.g. 'prep.ai.mit.edu' as 'pub/gnu/texinfo-3.7.tar.gz' (there might be a newer version there when you read this).
2. Install the info file(s) 'pcl-cvs.info*' into your standard 'info' directory. You should be able to do this by typing 'make install-info'.
3. Edit the file 'dir' in the 'info' directory and enter one line to contain a pointer to the info file(s) 'pcl-cvs.info*'. The line can, for instance, look like this:

```
* Pcl-cvs: (pcl-cvs).           An Emacs front-end to CVS.
```

1.3 How to make typeset documentation from pcl-cvs.texinfo

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

1. Run TeX by typing "make pcl-cvs.dvi". You will not get the indices unless you have the texindex program.
2. 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 About pcl-cvs

Pcl-cvs is a front-end to CVS versions 1.5 through 1.7 and newer; and possibly verison 1.3 and 1.4A2. It integrates the most frequently used CVS commands into an emacs interface.

2.1 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 8 [Bugs], page 22).

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 documentation.
- Inge Wallin (`inge@lysator.liu.se`) wrote the skeleton to `pcl-cvs.texinfo`, 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@lucid.com`) contributed `pcl-cvs-lucid.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 install procedure.
- Greg A. Woods (`woods@planix.com`) contributed code to implement the use of per-file diff buffers; and vendor join diffs with emerge and ediff; as well as various an sundry bug fixes and cleanups.

Apart from these, a lot of people have send me suggestions, ideas, requests, bug reports and encouragement. Thanks a lot! Without your there would be no new releases of pcl-cvs.

2.2 Where can I get pcl-cvs?

The current release of pcl-cvs is included in CVS-1.7.

The author's release of pcl-cvs can be fetched via anonymous ftp from `ftp.lysator.liu.se`, (IP no. 130.236.254.1) in the directory `pub/emacs`. If you don't live in Scandinavia you should probably check with archie to see if there is a site closer to you that archives pcl-cvs.

New releases will be announced to appropriate newsgroups. If you send your email address to me I will add you to my list of people to mail when I make a new release.

3 Getting started

This document assumes that you know what CVS is, and that you at least knows the fundamental concepts of CVS. If that is not the case you should read the man page for 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 invoke pcl-cvs by typing `M-x cvs-update RET`. If your emacs responds with '`[No match]`' your system administrator has not installed pcl-cvs properly. Try `M-x load-library RET pcl-cvs RET`. If that also fails - talk to your root. If it succeeds you might put this line in your '`.emacs`' file so that you don't have to type the '`load-library`' command every time you wish to use pcl-cvs:

```
(autoload 'cvs-update "pcl-cvs" nil t)
```

The function `cvs-update` will ask for a directory. The command '`cvs 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:

```
PCL-CVS release 1.05-CVS-$Name:  $.

In directory /users/ceder/FOO/test:
  Updated     bar
  Updated     file.txt
  Modified ci namechange
  Updated     newer

In directory /users/ceder/FOO/test/sub:
  Modified ci ChangeLog
---------- End -----
```

In this example the two files ('`bar`', '`file.txt`', and '`newer`') that are marked with '`Updated`' have been copied from the CVS repository to '`/users/ceder/FOO/test/`' since someone else have checked in newer versions of them. Two files ('`namechange`' and '`sub/ChangeLog`') have been modified locally, and needs 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 5.4

[Committing changes], page 11. You can press `x` to get rid of the "uninteresting" files that have only been '`Updated`' (and don't require any further action from you).

You can also easily get a '`diff`' between your modified file and the base version that you started from, and you can get the output from '`cvs log`' and '`cvs status`' on the listed files simply by pressing a key (see Section 5.6 [Getting info about files], page 12).

4 Buffer contents

The display contains four columns. They contain, from left to right:

- An asterisk when the file is *marked* (see Section 4.2 [Selected files], page 8).
- The status of the file. See See Section 4.1 [File status], page 7, for more information.
- A "need to be checked in"-marker (`ci`).
- The file name.

4.1 File status

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

`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 a remote repository by way of fetching and applying a patch to the file in your source. This is done for any file that exists in a remote repository and in your source; of which you haven't changed locally but is not the most recent version available in the remote repository.

`Modified`

The file is modified in your working directory, and there was no modification to the same file in the repository.

`Merged` The file is modified in your working directory, and there were modifications in the repository as well as in your copy, 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 source 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 5.11 [Viewing differences], page 14, for more details.

`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 needs to be checked in to the repository. You can resurrect it by typing `a` (see Section 5.7 [Adding and removing files], page 13).

'`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.

There are also a few special cases, that rarely occur, which have longer strings in the fields:

'`Removed from repository`'
: The file has been removed from your directory since someone has removed it from the repository. (It is still present in the Attic directory, so no permanent loss has occurred). This, unlike the other entries in this table, is not an error condition.

'`Removed from repository, changed by you`'
: You have modified a file that someone have removed from the repository. You can correct this situation by removing the file manually (see see Section 5.7 [Adding and removing files], page 13).

'`Removed by you, changed in repository`'
: You have removed a file, and before you committed the removal someone committed a change to that file. You could use `a` to resurrect the file (see see Section 5.7 [Adding and removing files], page 13).

'`Move away` *file* `- it is in the way`'
: For some reason CVS does not like the file *file*. Rename or remove it.

'`This repository is missing! Remove this dir manually.`'
: It is impossible to remove a directory in the CVS repository in a clean way. Someone have tried to remove one, and CVS gets confused. Remove your copy of the directory.

4.2 Selected files

Many of the commands works on the current set of *selected* files.

- If there are any files that are marked they constitute the set of selected files.
- Otherwise, if the cursor points to a file, that file is the selected file.
- Otherwise, if the cursor points to a directory, all the files in that directory that appears in the buffer are the selected files.

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

See Section 5.3 [Marking files], page 11 tells how you mark and unmark files.

5 Commands

This chapter describes all the commands that you can use in pcl-cvs.

5.1 Updating the directory

`M-x cvs-update`
: Run a '`cvs update`' command. You will be asked for the directory in which the '`cvs update`' will be run. The output will be parsed by pcl-cvs, and the result printed in the '`*cvs*`' buffer (see see Chapter 4 [Buffer contents], page 7 for a description of the contents).

  By default, '`cvs-update`' will descend recursively into subdirectories. You can avoid that behavior by giving a prefix argument to it (e.g., by typing `C-u M-x cvs-update RET`).

  All other commands in pcl-cvs requires that you have a '`*cvs*`' buffer. This is the command that you use to get one.

  CVS uses lock files in the repository to ensure the integrity of the data files in the repository. They might be left behind i.e. if a workstation crashes in the middle of a CVS operation. CVS outputs a message when it is waiting for a lock file to go away. Pcl-cvs will show the same message in the *cvs* buffer, together with instructions for deleting the lock files. You should normally not have to delete them manually — just wait a little while and the problem should fix itself. But if the lock files doesn't disappear you can delete them with `M-x cvs-delete-lock RET`.

`g`
: This will run '`cvs update`' again. It will always use the same buffer that was used with the previous '`cvs update`'. Give a prefix argument to avoid descending into subdirectories. This runs the command '`cvs-mode-update-no-prompt`'.

`G`
: This will run '`cvs update`' and prompt for a new directory to update. This runs the command '`cvs-update`'.

5.2 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`
`C-n`
`n` These keys move the cursor one file forward, towards the end of the buffer (`cookie-next-cookie`).

`C-p`
`p` These keys move one file backward, towards the beginning of the buffer (`cookie-previous-cookie`).

5.3 Marking files

Pcl-cvs works on a set of *selected files* (see Section 4.2 [Selected files], page 8). 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 will be 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 will be unmarked. (`cvs-mode-unmark`).

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

`ESC 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`).

5.4 Committing changes

`c` All files that have a "need to be checked in"-marker (see Chapter 4 [Buffer contents], page 7) can be checked in with the `c` command. It checks in all selected files (see Section 4.2 [Selected files], page 8) (except those who lack the "ci"-marker - they are ignored). Pressing `c` causes `cvs-mode-commit` to be run.

When you press `c` you will get a buffer called '`*cvs-commit-message*`'. Enter the log message for the file(s) in it. When you are ready you should press `C-c C-c` to actually commit the files (using `cvs-edit-done`).

Normally the '`*cvs-commit-message*`' buffer will retain the log message from the previous commit, but if the variable `cvs-erase-input-buffer` is set to a non-`nil` value the buffer will be erased. Point and mark will always be located around the entire buffer so that you can easily erase it with `C-w` ('`kill-region`').

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-after-commit`' is set to '`nil`'.

`C` This is just like '`cvs-mode-commit`', except that it tries to provide appropriate default log messages by looking at the '`ChangeLog`'s in the current directory. The idea is to write your ChangeLog entries first, and then use this command to commit your changes. Pressing `C` causes `cvs-mode-changelog-commit` to be run.

To select default log text, pcl-cvs:

- finds the ChangeLogs for the files to be checked in;
- verifies that the top entry in the ChangeLog is on the current date and by the current user; if not, no default text is provided;
- search the ChangeLog entry for paragraphs containing the names of the files we're checking in; and finally
- uses those paragraphs as the default log text in the '`*cvs-commit-message*`' buffer.

You can then commit the '`ChangeLog`' file once per day without any log message.

5.5 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 ignore any marked files.

`f` Find the file that the cursor points to. Run '`dired`' if the cursor points to a directory (`cvs-mode-find-file`).

`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`' will be found in the directory of the file the cursor points to. (`cvs-mode-add-change-log-entry-other-window`).

5.6 Getting info about files

Both of the following commands can be customized. See Chapter 6 [Customization], page 18.

`l` Run '`cvs log`' on all selected files, and show the result in a temporary buffer (`cvs-mode-log`).

`s` Run '`cvs status`' on all selected files, and show the result in a temporary buffer (`cvs-mode-status`).

5.7 Adding and removing files

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

`a` Add all selected files. This command can be used on '`Unknown`' files (see see Section 4.1 [File status], page 7). The status of the file will change to '`Added`', and you will have to use `c` ('`cvs-mode-commit`', see see Section 5.4 [Committing changes], page 11) to really add the file to the repository.

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

Selected files that are neither '`Unknown`' nor '`Removed`' will be ignored by this command.

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

`r` This command removes the selected files (after prompting for confirmation). The files are '`rm`'ed from your directory and (unless the status was '`Unknown`'; see Section 4.1 [File status], page 7) they will also be '`cvs remove`'d. If the files were '`Unknown`' they will disappear from the buffer. Otherwise their status will change to '`Removed`', and you must use `c` ('`cvs-mode-commit`', see Section 5.4 [Committing changes], page 11) to commit the removal.

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

5.8 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`.

5.9 Removing handled entries

`x` This command allows you to remove all entries that you have processed. More specifically, the lines for '`Updated`' files (see Section 4.1 [File status], page 7 and files that have been checked in (see Section 5.4 [Committing changes], page 11) 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.

The command is called `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`).

5.10 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`' doesn't exist it will be created.

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

This runs `cvs-mode-ignore`.

5.11 Viewing differences

`d` Display a '`cvs diff`' between the selected files and the RCS version that they are based on. See Chapter 6 [Customization], page 18 describes how you can send flags to '`cvs diff`'. If *cvs-diff-ignore-marks* is set to a non-`nil` value or if a prefix argument is given (but not both) any marked files will not be considered to be selected. (`cvs-mode-diff-cvs`).

`b` If CVS finds a conflict while merging two versions of a file (during a '`cvs update`', see Section 5.1 [Updating the directory], page 10) it will save the original file in a file called '`.#`*FILE*`.`*VERSION*' where *FILE* is the name of the file, and *VERSION* is the RCS version number that your file was based on.

With the `b` command you can run a '`diff`' on the files '`.#`*FILE*`.`*VERSION*' and '*FILE*'. You can get a context- or Unidiff by setting '`cvs-diff-flags`' - see Chapter 6 [Customization], page 18. This command only works on files that have status '`Conflict`' or '`Merged`'.

If *cvs-diff-ignore-marks* is set to a non-`nil` value or if a prefix argument is given (but not both) any marked files will not be considered to be selected. (`cvs-mode-diff-backup`).

5.12 Running ediff

`e` This command works slightly different depending on the version of '`ediff`' and the file status.

With modern versions of '`ediff`', this command invokes '`run-ediff-from-cvs-buffer`' on one file.

Note: When the file status is '`Merged`' or '`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.

Older versions of '`ediff`' use an interface similar to '`emerge`'. The function '`cvs-old-ediff-interface`' is invoked if the version of '`ediff`' you have doesn't support '`run-ediff-from-cvs-buffer`'. These older versions do not support merging of revisions.

'`Modified`'
: Run '`ediff-files`' with your working file as file A, and the latest revision in the repository as file B.

'`Merged`'
'`Conflict`'
: Run '`ediff-files3`' with your working file (as it was prior to your invocation of '`cvs-update`') as file A, the latest revision in the repository as file B, and the revision that you based your local modifications on as ancestor.

'`Updated`'
'`Patched`'
: Run '`ediff-files`' with your working file as file A, and a given revision in the repository as file B. You are prompted for the revision to ediff against, and you may specify either a tag name or a numerical revision number (see Section 5.6 [Getting info about files], page 12).

5.13 Running emerge

`E` Invoke '`emerge`' on one file. This command works slightly different depending on the file status.

'`Modified`'
: Run '`emerge-files`' with your working file as file A, and the latest revision in the repository as file B.

'`Merged`'
'`Conflict`'
: Run '`emerge-files-with-ancestor`' with your working file (as it was prior to your invocation of '`cvs-update`') as file A, the latest revision in the repository as file B, and the revision that you based your local modifications on as ancestor.

Note: When the file status is '`Merged`' or '`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 '`emerge`' (to successfully terminate the merge) the file that CVS created will be overwritten.

5.14 Reverting your buffers

`R`
: If you are editing (or just viewing) a file in a buffer, and that file is changed by CVS during a '`cvs-update`', all you have to do is type `R` in the *cvs* buffer to read in the new versions of the files.

  All files that are '`Updated`', '`Merged`' or in '`Conflict`' are reverted from the disk. Any other files are ignored. Only files that you were already editing are read.

  An error is signalled if you have modified the buffer since it was last changed. (`cvs-mode-revert-updated-buffers`).

5.15 Miscellaneous commands

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

`M-x cvs-delete-lock`
: This command can be used in any buffer, and 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.

`q` Bury the *cvs* buffer. (`bury-buffer`).

6 Customization

If you have an idea about any customization that would be handy but isn't present in this list, please tell me! See Chapter 8 [Bugs], page 22 for info on how to reach me.

'`cvs-erase-input-buffer`'

If set to anything else than `nil` the edit buffer will be erased before you write the log message (see Section 5.4 [Committing changes], page 11).

'`cvs-inhibit-copyright-message`'

The copyright message that is displayed on startup can be annoying after a while. Set this variable to '`t`' if you want to get rid of it. (But don't set this to '`t`' in the system defaults file - new users should see this message at least once).

'`cvs-diff-flags`'

A list of strings to pass as arguments to the '`cvs diff`' and '`diff`' programs. This is used by '`cvs-mode-diff-cvs`' and '`cvs-mode-diff-backup`' (key `b`, see Section 5.11 [Viewing differences], page 14). If you prefer the Unidiff format you could add this line to your '`.emacs`' file:

```
(setq cvs-diff-flags '("-u"))
```

'`cvs-diff-ignore-marks`'

If this variable is non-`nil` or if a prefix argument is given (but not both) to '`cvs-mode-diff-cvs`' or '`cvs-mode-diff-backup`' marked files are not considered selected.

'`cvs-log-flags`'

List of strings to send to '`cvs log`'. Used by '`cvs-mode-log`' (key `l`, see Section 5.6 [Getting info about files], page 12).

'`cvs-status-flags`'

List of strings to send to '`cvs status`'. Used by '`cvs-mode-status`' (key `s`, see Section 5.6 [Getting info about files], page 12).

'`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 5.9 [Removing handled entries], page 14.

'`cvs-auto-revert-after-commit`'

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 is default '`t`'. See Section 5.4 [Committing changes], page 11.

`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 everytime a cvs command is run it will be called as '`cvs -d` *cvs-cvsroot*...' This can be useful if your site has several repositories.

`TMPDIR`

Pcl-cvs uses this *environment variable* to decide where to put the temporary files it needs. It defaults to '`/tmp`' if it is not set.

`cvs-commit-buffer-require-final-newline`

When you enter a log message in the '`*cvs-commit-message*`' buffer pcl-cvs will normally automatically insert 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.

`cvs-sort-ignore-file`

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

7 Future enhancements

Pcl-cvs is still under development and needs a number of enhancements to be called complete. Below is my current wish-list for future releases of pcl-cvs. Please, let me know which of these features you want most. They are listed below in approximately the order that I currently think I will implement them in.

- Rewritten parser code. There are many situations where pcl-cvs will fail to recognize the output from CVS. The situation could be greatly increased.
- '`cvs-status`'. This will run '`cvs status`' in a directory and produce a buffer that looks pretty much like the current *cvs* buffer. That buffer will include information for all version-controlled files. (There will be a simple keystroke to remove all "uninteresting" files, that is, files that are "Up-to-date"). In this new buffer you will be able to update a file, commit a file, et c. The big win with this is that you will be able to watch the differences between your current working file and the head revision in the repository before you update the file, and you can then choose to update it or let it wait for a while longer.
- Log mode. When this mode is finished you will be able to move around (using `n` and `p`) between the revisions of a file, mark two of them, and run a diff between them. You will be able to hide branches (similar to the way you can hide sub-paragraphs in outline-mode) and do merges between revisions. Other ideas about this are welcome.
- The current model for marks in the *cvs* buffer seems to be confusing. I am considering to use the VM model instead, where marks are normally inactive. To activate the mark, you issue a command like '`cvs-mode-next-command-uses-marks`'. I might implement a flag so that you can use either version. Feedback on this before I start coding it is very welcome.
- It should be possible to run commands such as '`cvs log`', '`cvs status`' and '`cvs commit`' directly from a buffer containing a file, instead of having to '`cvs-update`'. If the directory contains many files the '`cvs-update`' can take quite some time, especially on a slow machine. I planed to put these kind of commands on the prefix `C-c C-v`, but that turned out to be used by for instance c++-mode. If you have any suggestions for a better prefix key, please let me know.
- Increased robustness. For instance, you can not currently press `C-g` when you are entering the description of a file that you are adding without confusing pcl-cvs.
- Support for multiple active *cvs* buffers.
- Dired support. I have an experimental '`dired-cvs.el`' that works together with CVS 1.2. Unfortunately I wrote it on top of a non-standard '`dired.el`', so it must be rewritten.
- An ability to send user-supplied options to all the cvs commands.
- Pcl-cvs is not at all clever about what it should do when '`cvs update`' runs a program (due to the '`-u`' option in the '`modules`' file — see '`cvs(5)`'). The current release uses a regexp to

search for the end. At the very least that regexp should be configured for different modules. Tell me if you have any idea about what is the right thing to do. In a perfect world the program should also be allowed to print to '`stderr`' without causing pcl-cvs to crash.

If you miss something in this wish-list, let me know! I don't promise that I will write it, but I will at least try to coordinate the efforts of making a good Emacs front end to CVS. See See Chapter 8 [Bugs], page 22 for information about how to reach me.

So far, I have written most of pcl-cvs in my all-to-rare spare time. If you want pcl-cvs to be developed faster you can write a contract with Signum Support to do the extension. You can reach Signum Support by email to '`info@signum.se`' or via mail to Signum Support AB, Box 2044, S-580 02 Linkoping, Sweden. Phone: +46 (0) 13 - 21 46 00. Fax: +46 (0) 13 - 21 47 00.

8 Bugs (known and unknown)

If you find a bug or misfeature, don't hesitate to tell me! Send email to '`ceder@lysator.liu.se`'.

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

Below is a partial list of currently known problems with pcl-cvs version 1.05.

Commit causes Emacs to hang
: Emacs waits for the '`cvs commit`' command to finish before you can do anything. If you start a background job from the loginfo file you must take care that it closes '`stdout`' and '`stderr`' if you do not want to wait for it. (You do that with '`background-command &>- 2&>- &`' if you are starting '`background-command`' from a '`/bin/sh`' shell script).

  Your emacs will also hang if there was a lock file in the repository. In this case you can type `C-g` to get control over your emacs again.

Name clash in Emacs 19
: This is really a bug in Elib or the Emacs 19 distribution. Both Elib and Emacs 19.6 through at least 19.10 contains a file named '`cookie.el`'. One of the files will have to be renamed, and we are currently negotiating about which of the files to rename.

Commands while cvs-update is running
: It is possible to type commands in the *cvs* buffer while the update is running, but error messages is all that you will get. The error messages should be better.

Unexpected output from CVS
: Unexpected output from CVS confuses pcl-cvs. It will currently create a bug report that you can mail to me. It should do something more civilized.

Appendix A GNU GENERAL PUBLIC LICENSE

Function and Variable Index

Concept Index

K

L

M

O

P

R

S

T

U

V

Key Index

Short Contents

Table of Contents