GNU Readline Library

This manual describes the GNU Readline Library (version 5.0, 28 January 2004), a library which aids in the consistency of user interface across discrete programs which provide a command line interface.

Table of Contents

1 Command Line Editing

This chapter describes the basic features of the gnu

When you add text in the middle of a line, you will notice that characters to the right of the cursor are ‘pushed over’ to make room for the text that you have inserted. Likewise, when you delete text behind the cursor, characters to the right of the cursor are ‘pulled back’ to fill in the blank space created by the removal of the text. A list of the bare essentials for editing the text of an input line followsxt of anbac8.612(ediC-b34(an)813.94TD[57.5902(ediM)-248(oserted

search. If that variable has not been assigned a value, the *hESCi* and *C-J* characters will terminate an incremental search. *C-g* will abort an incremental search and restore the original line. When the search is terminated, the history entry containing the search string becomes the current line.

To find other matching entries in the termn

Variable names and values, where appropriate, are recognized without regard to case.

A great deal of run-time behavior is changeable with the following variables.

`bell-style`

Controls what happens when Readline wants to ring the terminal bell. If set to ‘none

If set to ‘on’, the history code attempts to place point at the same location on each history line retrieved with `previous-history` or `next-history`.

`horizontal-scroll-mode`
This variable can be set to either ‘on’ or ‘off’. Setting it to ‘on’ means that the text of the lines being edited will scroll horizontally on a single screen line when they are longer than the width of the screen, instead of wrapping onto a new screen line. By default, this variable to ‘ off’.

`input-meta`
If on’, Readlinet-bit inputit
theth bitharacters it

`output-meta`
: If set to ‘on’, Readline will display characters with the eighth bit set directly rather than as a meta-prefixed escape sequence. The default is ‘off’.

`page-completions`
: If set to ‘on’, Readline uses an internal `more`-like pager to display

insert mode. This command affects only emacs

1.4.7 Keyboard Macros

`start-kbd-macro (C-x ()`
: Begin saving the characters typed into the current keyboard macro.

`end-kbd-macro (C-x ))`
: Stop saving the characters typed into the current keyboard macro and save the

`character-search-backward (M-C-])`
: A character is read and point is moved to the previous occurrence of that character. A negative count searches for subsequent occurrences.

`insert-comment (M-#)`
: Without a numeric argument, the value of the `comment-begin` variable is in-

2 Programming with GNU Readline

This chapter describes the interface between the gnu Readline Library and other programs. If you are a programmer, and you wish to include the features found in gnu Readline

```
      free (line_read);
      line_read = (char *)NULL;
    }

  /* Get a line from the user. */
  line_read = readline ("");

  /* If the line has any text in it,
     save it on the history. */
  if (line_read && *line_read)
    add_history (line_read);

  return (line_read);
}
```

This function gives the user the default behaviour of *hTABi* completion: completion on file names. If you do not want Readline to complete on filenames, you can change the binding

2.2.1 Readline Typedefs

For readabilty, we declare a number of new object types, all pointers to functions.
F3abiltyyp33(d45f331wharsyns48*rl˙compentry˙20.0˙isyns.)(co-3isyns. F3abi4eyyp33(d45f334wharsyns48**

behavior (refreshing the current line as opposed to refreshing the screen, for example). Some choose to ignore it. In general, if a function uses the numeric argument as a repeat count, it should be able to do something useful with both negative and positive arguments. At the very least, it should be aware that it can be passed a negative argument.

A command function should return 0 if its action completes successfully, and a non-zero

char * rl_prompt [Variable]

The prompt Readline uses. This is set from the argument to `readline()`, and should not be assigned to directly. The

`rl_hook_func_t * rl` [Variable]

RL_STATE_TERMPREPPED

`int` rl_numeric_arg [Variable]

`int` **rl_read_init_file** (`const char *filenam` [Function]

`int` **rl_forced_update_display** (`void`) [Function]

by bracketing a sequence of such characters with the special markers RL_PROMPT_START_IGNORE and RL_PROMPT_END_IGNORE (declared in 'readline.h'. This may be used to embed terminal-specific escape sequences in prompts.

int rl_ [Function]

`int` **rl_initialize** (`void`) [Function]
Initialize or re-initialize Readline's internal state. It's not strictly necessary to call this; `readline()` calls it before reading any input.

`int` **rl_ding** (`void`) [Function]
Ring the terminal bell, obeying the setting of `bell-style`.

`int` **rl_alphabetic** (`int c`) [Function]
Return 1 if *c* is an alphabetic character.

`void` **rl_display** [Function]

`int` **rl_variable_bind** (`const char *variable, const char *value`) [Function]
Make the Readline variable *variable* have *value*. This behaves as if the readline command '`set` *`variable value`*' had been executed in an `inputrc` file (see Section 1.3.1 [Readline Init File Syntax], page 4).

`void` **rl_variable_dumper** (`int readable`) [Function]
Print the readline variable names and their current values to `rl_outstream`. If *readable* is non-zero, the list is formatted in such a way that it can be made part of an `inputrc` file and re-read.

`int` **rl_set_paren_blink_timeout** (`int u`) [Function]
Set the time interval (in microseconds) that Readline waits when showing a balancing character when `blink-matching-paren`

the function referred to by the value of `rl_deprep_term_function` should be called before the program exits to reset the terminal settings.

`int` **rl_set_signals** (`void`) [Function]

Install Readline's signal handler for `SIGINT`, `SIGQUIT`, `SIGTERM`, `SIGALRM`, `SIGTSTP`,

`int` **rl_complete** (`int ignore, int invoking_key`) [Function]
Complete the word at or before point. You have supplied the function that does the

matching the text against names in the filesystem. It is called with *text*, the text of the word to be dequoted, and *quote_char*, which is the quoting character that delimits the filename (usually ‘'’ or ‘"’). If *quote_char* is zero, the filename was not in an embedded string.

`rl_linebuf_func_t *` **rl_char_is_quoted_p** [Variable]

A pointer to a function to call that determines whether or not a specific character in the line buffer is quoted, according to whatever quoting mechanism the program calling Readline

`const char * rl_` [Variable]

`int` **rl_inhibit_completion** [Variable]

```
/* fileman.c -- A tiny application which demonstrates how to use the
   GNU Readline library.  This application interactively allows users
```

```
while (line[i] && whitespace (line[i]))
  i++;
```

```
com_rename (arg)
     char *arg;
{
  too_dangerous ("rename");
  return (1);
}

com_stat (arg)
     char *arg;
{
  struct stat finfo;

  if (!valid_argument ("stat", arg))
    return (1);

  if (stat (arg, &finfo) == -1)
    {
      perror (arg);
      return (1);
    }

  printf ("Statistics for `%s':\n", arg);

  printf ("%s has %d link%s, and is %d byte%s in length.\n", arg,
```

```
  if (!printed)
    {
      printf ("No commands match `%s'.  Possibilties are:\n", arg);

      for (i = 0; commands[i].name; i++)
        {
          /* Print in six columns. */
          if (printed == 6)
            {
              printed = 0;
              printf ("\n");
            }

          printf ("%s\t", commands[i].name);
          printed++;
        }

      if (printed)
        printf ("\n");
    }
  return (0);
}

/* Change to the directory ARG. */
com_cd (arg)
     char *arg;
{
  if (chdir (arg) == -1)
    {
      perror (arg);
      return 1;
    }

  com_pwd ("");
  return (0);
}

/* Print out the current working directory. */
com_pwd (ignore)
     char *ignore;
{
  char dir[1024], *s;

  s = getcwd (dir, sizeof(dir) - 1);
  if (s == 0)
    {
      printf ("Error getting pwd: %s\n", dir);
      return 1;
    }

  printf ("Current directory is %s\n", dir);
  return 0;
}

/* The user wishes to quit using this program.  Just set DONE
   non-zero. */
com_quit (arg)
     char *arg;
```

Appendix A Copying This Manual

A.1 GNU Free Documentation License

Version 1.2, November 2002

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others. 344priocip5(commer56ible4(Ln)27(t)]TJ0-w)253380(wy)27who-333(lic(The)-407(purp(lic

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

A.1.1 ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the