GNU Readline Library User Interface

Edition 8.3, for `Readline Library` Version 8.3.
November 2024

Chet Ramey, Case Western Reserve University
Brian Fox, Free Software Foundation

This manual describes the end user interface of the GNU Readline Library (version 8.3, 29 November 2024), a library which aids in the consistency of user interface across discrete programs which provide a command line interface.

Copyright © 1988–2024 Free Software Foundation, Inc.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Table of Contents

1 Command Line Editing 1
1.1 Introduction to Line Editing 1
1.2 Readline Interaction 1
1.2.1 Readline Bare Essentials 2
1.2.2 Readline Movement Commands 2
1.2.3 Readline Killing Commands 3
1.2.4 Readline Arguments 3
1.2.5 Searching for Commands in the History 3
1.3 Readline Init File 4
1.3.1 Readline Init File Syntax 4
1.3.2 Conditional Init Constructs 14
1.3.3 Sample Init File 15
1.4 Bindable Readline Commands 18
1.4.1 Commands For Moving 18
1.4.2 Commands For Manipulating The History 19
1.4.3 Commands For Changing Text 21
1.4.4 Killing And Yanking 22
1.4.5 Specifying Numeric Arguments 23
1.4.6 Letting Readline Type For You 24
1.4.7 Keyboard Macros 25
1.4.8 Some Miscellaneous Commands 25
1.5 Readline vi Mode 27

Appendix A GNU Free Documentation License .. 28

1 Command Line Editing

This chapter describes the basic features of the GNU command line editing interface.

1.1 Introduction to Line Editing

The following paragraphs use Emacs style to describe the notation used to represent keystrokes.

The text *C-k* is read as 'Control-K' and describes the character produced when the k key is pressed while the Control key is depressed.

The text *M-k* is read as 'Meta-K' and describes the character produced when the Meta key (if you have one) is depressed, and the k key is pressed (a *meta character*), then both are released. The Meta key is labeled ALT or Option on many keyboards. On keyboards with two keys labeled ALT (usually to either side of the space bar), the ALT on the left side is generally set to work as a Meta key. One of the ALT keys may also be configured as some other modifier, such as a Compose key for typing accented characters.

On some keyboards, the Meta key modifier produces characters with the eighth bit (0200) set. You can use the enable-meta-key variable to control whether or not it does this, if the keyboard allows it. On many others, the terminal or terminal emulator converts the metafied key to a key sequence beginning with ESC as described in the next paragraph.

If you do not have a Meta or ALT key, or another key working as a Meta key, you can generally achieve the latter effect by typing ESC *first*, and then typing k. The ESC character is known as the *meta prefix*).

Either process is known as *metafying* the k key.

If your Meta key produces a key sequence with the ESC meta prefix, you can make *M-key* key bindings you specify (see Key Bindings in Section 1.3.1 [Readline Init File Syntax], page 4) do the same thing by setting the force-meta-prefix variable.

The text *M-C-k* is read as 'Meta-Control-k' and describes the character produced by metafying *C-k*.

In addition, several keys have their own names. Specifically, DEL, ESC, LFD, SPC, RET, and TAB all stand for themselves when seen in this text, or in an init file (see Section 1.3 [Readline Init File], page 4). If your keyboard lacks a LFD key, typing C-j will output the appropriate character. The RET key may be labeled Return or Enter on some keyboards.

1.2 Readline Interaction

Often during an interactive session you type in a long line of text, only to notice that the first word on the line is misspelled. The Readline library gives you a set of commands for manipulating the text as you type it in, allowing you to just fix your typo, and not forcing you to retype the majority of the line. Using these editing commands, you move the cursor to the place that needs correction, and delete or insert the text of the corrections. Then, when you are satisfied with the line, you simply press RET. You do not have to be at the end of the line to press RET; the entire line is accepted regardless of the location of the cursor within the line.

1.2.1 Readline Bare Essentials

In order to enter characters into the line, simply type them. The typed character appears where the cursor was, and then the cursor moves one space to the right. If you mistype a character, you can use your erase character to back up and delete the mistyped character.

Sometimes you may mistype a character, and not notice the error until you have typed several other characters. In that case, you can type *C-b* to move the cursor to the left, and then correct your mistake. Afterwards, you can move the cursor to the right with *C-f*.

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. These are the bare essentials for editing the text of an input line:

C-b Move back one character.

C-f Move forward one character.

DEL or Backspace
Delete the character to the left of the cursor.

C-d Delete the character underneath the cursor.

Printing characters
Insert the character into the line at the cursor.

C-_ or *C-x C-u*
Undo the last editing command. You can undo all the way back to an empty line.

Depending on your configuration, the Backspace key might be set to delete the character to the left of the cursor and the DEL key set to delete the character underneath the cursor, like *C-d*, rather than the character to the left of the cursor.

1.2.2 Readline Movement Commands

The above table describes the most basic keystrokes that you need in order to do editing of the input line. For your convenience, many other commands are available in addition to *C-b*, *C-f*, *C-d*, and DEL. Here are some commands for moving more rapidly within the line.

C-a Move to the start of the line.

C-e Move to the end of the line.

M-f Move forward a word, where a word is composed of letters and digits.

M-b Move backward a word.

C-l Clear the screen, reprinting the current line at the top.

Notice how *C-f* moves forward a character, while *M-f* moves forward a word. It is a loose convention that control keystrokes operate on characters while meta keystrokes operate on words.

1.2.3 Readline Killing Commands

Killing text means to delete the text from the line, but to save it away for later use, usually by *yanking* (re-inserting) it back into the line. ('Cut' and 'paste' are more recent jargon for 'kill' and 'yank'.)

If the description for a command says that it 'kills' text, then you can be sure that you can get the text back in a different (or the same) place later.

When you use a kill command, the text is saved in a *kill-ring*. Any number of consecutive kills save all of the killed text together, so that when you yank it back, you get it all. The kill ring is not line specific; the text that you killed on a previously typed line is available to be yanked back later, when you are typing another line.

Here is the list of commands for killing text.

`C-k` Kill the text from the current cursor position to the end of the line.

`M-d` Kill from the cursor to the end of the current word, or, if between words, to the end of the next word. Word boundaries are the same as those used by `M-f`.

`M-DEL` Kill from the cursor to the start of the current word, or, if between words, to the start of the previous word. Word boundaries are the same as those used by `M-b`.

`C-w` Kill from the cursor to the previous whitespace. This is different than `M-DEL` because the word boundaries differ.

Here is how to *yank* the text back into the line. Yanking means to copy the most-recently-killed text from the kill buffer into the line at the current cursor position.

`C-y` Yank the most recently killed text back into the buffer at the cursor.

`M-y` Rotate the kill-ring, and yank the new top. You can only do this if the prior command is `C-y` or `M-y`.

1.2.4 Readline Arguments

You can pass numeric arguments to Readline commands. Sometimes the argument acts as a repeat count, other times it is the *sign* of the argument that is significant. If you pass a negative argument to a command which normally acts in a forward direction, that command will act in a backward direction. For example, to kill text back to the start of the line, you might type '`M-- C-k`'.

The general way to pass numeric arguments to a command is to type meta digits before the command. If the first 'digit' typed is a minus sign ('`-`'), then the sign of the argument will be negative. Once you have typed one meta digit to get the argument started, you can type the remainder of the digits, and then the command. For example, to give the `C-d` command an argument of 10, you could type '`M-1 0 C-d`', which will delete the next ten characters on the input line.

1.2.5 Searching for Commands in the History

Readline provides commands for searching through the command history for lines containing a specified string. There are two search modes: *incremental* and *non-incremental*.

Incremental searches begin before the user has finished typing the search string. As each character of the search string is typed, Readline displays the next entry from the history

matching the string typed so far. An incremental search requires only as many characters as needed to find the desired history entry. When using emacs editing mode, type *C-r* to search backward in the history for a particular string. Typing *C-s* searches forward through the history. The characters present in the value of the `isearch-terminators` variable are used to terminate an incremental search. If that variable has not been assigned a value, the `ESC` and *C-j* characters terminate an incremental search. *C-g* aborts an incremental search and restores 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 history list, type *C-r* or *C-s* as appropriate. This searches backward or forward in the history for the next entry matching the search string typed so far. Any other key sequence bound to a Readline command terminates the search and executes that command. For instance, a `RET` terminates the search and accepts the line, thereby executing the command from the history list. A movement command will terminate the search, make the last line found the current line, and begin editing.

Readline remembers the last incremental search string. If two *C-r*s are typed without any intervening characters defining a new search string, Readline uses any remembered search string.

Non-incremental searches read the entire search string before starting to search for matching history entries. The search string may be typed by the user or be part of the contents of the current line.

1.3 Readline Init File

Although the Readline library comes with a set of Emacs-like keybindings installed by default, it is possible to use a different set of keybindings. Any user can customize programs that use Readline by putting commands in an *inputrc* file, conventionally in their home directory. The name of this file is taken from the value of the environment variable `INPUTRC`. If that variable is unset, the default is `~/.inputrc`. If that file does not exist or cannot be read, Readline looks for `/etc/inputrc`.

When a program that uses the Readline library starts up, Readline reads the init file and sets any variables and key bindings it contains.

In addition, the `C-x C-r` command re-reads this init file, thus incorporating any changes that you might have made to it.

1.3.1 Readline Init File Syntax

There are only a few basic constructs allowed in the Readline init file. Blank lines are ignored. Lines beginning with a '`#`' are comments. Lines beginning with a '`$`' indicate conditional constructs (see Section 1.3.2 [Conditional Init Constructs], page 14). Other lines denote variable settings and key bindings.

Variable Settings

> You can modify the run-time behavior of Readline by altering the values of variables in Readline using the `set` command within the init file. The syntax is simple:
>
> ```
> set variable value
> ```

Here, for example, is how to change from the default Emacs-like key binding to use `vi` line editing commands:

```
set editing-mode vi
```

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

Boolean variables (those that can be set to on or off) are set to on if the value is null or empty, *on* (case-insensitive), or 1. Any other value results in the variable being set to off.

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

`active-region-start-color`
: A string variable that controls the text color and background when displaying the text in the active region (see the description of `enable-active-region` below). This string must not take up any physical character positions on the display, so it should consist only of terminal escape sequences. It is output to the terminal before displaying the text in the active region. This variable is reset to the default value whenever the terminal type changes. The default value is the string that puts the terminal in standout mode, as obtained from the terminal's terminfo description. A sample value might be '`\e[01;33m`'.

`active-region-end-color`
: A string variable that "undoes" the effects of `active-region-start-color` and restores "normal" terminal display appearance after displaying text in the active region. This string must not take up any physical character positions on the display, so it should consist only of terminal escape sequences. It is output to the terminal after displaying the text in the active region. This variable is reset to the default value whenever the terminal type changes. The default value is the string that restores the terminal from standout mode, as obtained from the terminal's terminfo description. A sample value might be '`\e[0m`'.

`bell-style`
: Controls what happens when Readline wants to ring the terminal bell. If set to '`none`', Readline never rings the bell. If set to '`visible`', Readline uses a visible bell if one is available. If set to '`audible`' (the default), Readline attempts to ring the terminal's bell.

`bind-tty-special-chars`
: If set to '`on`' (the default), Readline attempts to bind the control characters that are treated specially by the kernel's terminal driver to their Readline equivalents. These override the default Readline bindings described here. Type '`stty -a`' at a Bash prompt to see your current terminal settings, including the special control characters (usually `cchars`).

`blink-matching-paren`
: If set to '`on`', Readline attempts to briefly move the cursor to an opening parenthesis when a closing parenthesis is inserted. The default is '`off`'.

`colored-completion-prefix`
: If set to '`on`', when listing completions, Readline displays the common prefix of the set of possible completions using a different color. The color definitions are taken from the value of the `LS_COLORS` environment variable. If there is a color definition in `LS_COLORS` for the custom suffix '`readline-colored-completion-prefix`', Readline uses this color for the common prefix instead of its default. The default is '`off`'.

`colored-stats`
: If set to '`on`', Readline displays possible completions using different colors to indicate their file type. The color definitions are taken from the value of the `LS_COLORS` environment variable. The default is '`off`'.

`comment-begin`
: The string to insert at the beginning of the line by the `insert-comment` command. The default value is `"#"`.

`completion-display-width`
: The number of screen columns used to display possible matches when performing completion. The value is ignored if it is less than 0 or greater than the terminal screen width. A value of 0 causes matches to be displayed one per line. The default value is -1.

`completion-ignore-case`
: If set to '`on`', Readline performs filename matching and completion in a case-insensitive fashion. The default value is '`off`'.

`completion-map-case`
: If set to '`on`', and *completion-ignore-case* is enabled, Readline treats hyphens ('`-`') and underscores ('`_`') as equivalent when performing case-insensitive filename matching and completion. The default value is '`off`'.

`completion-prefix-display-length`
: The maximum length in characters of the common prefix of a list of possible completions that is displayed without modification. When set to a value greater than zero, Readline replaces common prefixes longer than this value with an ellipsis when displaying possible completions.

`completion-query-items`
: The number of possible completions that determines when the user is asked whether the list of possibilities should be displayed. If the number of possible completions is greater than or equal to

this value, Readline asks whether or not the user wishes to view them; otherwise, Readline simply lists the completions. This variable must be set to an integer value greater than or equal to zero. A zero value means Readline should never ask; negative values are treated as zero. The default limit is `100`.

`convert-meta`
: If set to '`on`', Readline converts characters it reads that have the eighth bit set to an ASCII key sequence by clearing the eighth bit and prefixing an `ESC` character, converting them to a meta-prefixed key sequence. The default value is '`on`', but Readline sets it to '`off`' if the locale contains characters whose encodings may include bytes with the eighth bit set. This variable is dependent on the `LC_CTYPE` locale category, and may change if the locale changes. This variable also affects key bindings; see the description of `force-meta-prefix` below.

`disable-completion`
: If set to '`On`', Readline inhibits word completion. Completion characters are inserted into the line as if they had been mapped to `self-insert`. The default is '`off`'.

`echo-control-characters`
: When set to '`on`', on operating systems that indicate they support it, Readline echoes a character corresponding to a signal generated from the keyboard. The default is '`on`'.

`editing-mode`
: The `editing-mode` variable controls the default set of key bindings. By default, Readline starts up in emacs editing mode, where the keystrokes are most similar to Emacs. This variable can be set to either '`emacs`' or '`vi`'.

`emacs-mode-string`
: If the *show-mode-in-prompt* variable is enabled, this string is displayed immediately before the last line of the primary prompt when emacs editing mode is active. The value is expanded like a key binding, so the standard set of meta- and control- prefixes and backslash escape sequences is available. The '`\1`' and '`\2`' escapes begin and end sequences of non-printing characters, which can be used to embed a terminal control sequence into the mode string. The default is '`@`'.

`enable-active-region`
: *point* is the current cursor position, and *mark* refers to a saved cursor position (see Section 1.4.1 [Commands For Moving], page 18). The text between the point and mark is referred to as the *region*. When this variable is set to '`On`', Readline allows certain commands to designate the region as *active*. When the region is active, Readline highlights the text in the region using the value

of the `active-region-start-color`, which defaults to the string that enables the terminal's standout mode. The active region shows the text inserted by bracketed-paste and any matching text found by incremental and non-incremental history searches. The default is '`On`'.

`enable-bracketed-paste`
When set to '`On`', Readline configures the terminal to insert each paste into the editing buffer as a single string of characters, instead of treating each character as if it had been read from the keyboard. This is called putting the terminal into *bracketed paste mode*; it prevents Readline from executing any editing commands bound to key sequences appearing in the pasted text. The default is '`On`'.

`enable-keypad`
When set to '`on`', Readline tries to enable the application keypad when it is called. Some systems need this to enable the arrow keys. The default is '`off`'.

`enable-meta-key`
When set to '`on`', Readline tries to enable any meta modifier key the terminal claims to support when it is called. On many terminals, the Meta key is used to send eight-bit characters; this variable checks for the terminal capability that indicates the terminal can enable and disable a mode that sets the eighth bit of a character (0200) if the Meta key is held down when the character is typed (a meta character). The default is '`on`'.

`expand-tilde`
If set to '`on`', Readline attempts tilde expansion when it attempts word completion. The default is '`off`'.

`force-meta-prefix`
If set to '`on`', Readline modifies its behavior when binding key sequences containing `\M-` or `Meta-` (see `Key Bindings` in Section 1.3.1 [Readline Init File Syntax], page 4) by converting a key sequence of the form `\M-`*C* or `Meta-`*C* to the two-character sequence *ESC* *C* (adding the meta prefix). If `force-meta-prefix` is set to '`off`' (the default), Readline uses the value of the `convert-meta` variable to determine whether to perform this conversion: if `convert-meta` is '`on`', Readline performs the conversion described above; if it is '`off`', Readline converts *C* to a meta character by setting the eighth bit (0200). The default is '`off`'.

`history-preserve-point`
If set to '`on`', the history code attempts to place the point (the current cursor position) at the same location on each history line retrieved with `previous-history` or `next-history`. The default is '`off`'.

`history-size`
: Set the maximum number of history entries saved in the history list. If set to zero, any existing history entries are deleted and no new entries are saved. If set to a value less than zero, the number of history entries is not limited. By default, the number of history entries is not limited. If you try to set *history-size* to a non-numeric value, the maximum number of history entries will be set to 500.

`horizontal-scroll-mode`
: Setting this variable to '`on`' means that the text of the lines being edited will scroll horizontally on a single screen line when the lines are longer than the width of the screen, instead of wrapping onto a new screen line. This variable is automatically set to '`on`' for terminals of height 1. By default, this variable is set to '`off`'.

`input-meta`
: If set to '`on`', Readline enables eight-bit input (that is, it does not clear the eighth bit in the characters it reads), regardless of what the terminal claims it can support. The default value is '`off`', but Readline sets it to '`on`' if the locale contains characters whose encodings may include bytes with the eighth bit set. This variable is dependent on the `LC_CTYPE` locale category, and its value may change if the locale changes. The name `meta-flag` is a synonym for `input-meta`.

`isearch-terminators`
: The string of characters that should terminate an incremental search without subsequently executing the character as a command (see Section 1.2.5 [Searching], page 3). If this variable has not been given a value, the characters `ESC` and *`C-j`* terminate an incremental search.

`keymap`
: Sets Readline's idea of the current keymap for key binding commands. Built-in `keymap` names are `emacs`, `emacs-standard`, `emacs-meta`, `emacs-ctlx`, `vi`, `vi-move`, `vi-command`, and `vi-insert`. `vi` is equivalent to `vi-command` (`vi-move` is also a synonym); `emacs` is equivalent to `emacs-standard`. Applications may add additional names. The default value is `emacs`; the value of the `editing-mode` variable also affects the default keymap.

`keyseq-timeout`
: Specifies the duration Readline will wait for a character when reading an ambiguous key sequence (one that can form a complete key sequence using the input read so far, or can take additional input to complete a longer key sequence). If Readline doesn't receive any input within the timeout, it uses the shorter but complete key sequence. Readline uses this value to determine whether or not input is available on the current input source (`rl_instream` by default). The value is specified in milliseconds, so a value of 1000 means that Readline will wait one second for additional input. If this variable

is set to a value less than or equal to zero, or to a non-numeric value, Readline waits until another key is pressed to decide which key sequence to complete. The default value is `500`.

`mark-directories`
: If set to '`on`', completed directory names have a slash appended. The default is '`on`'.

`mark-modified-lines`
: When this variable is set to '`on`', Readline displays an asterisk ('`*`') at the start of history lines which have been modified. This variable is '`off`' by default.

`mark-symlinked-directories`
: If set to '`on`', completed names which are symbolic links to directories have a slash appended, subject to the value of `mark-directories`. The default is '`off`'.

`match-hidden-files`
: This variable, when set to '`on`', forces Readline to match files whose names begin with a '`.`' (hidden files) when performing filename completion. If set to '`off`', the user must include the leading '`.`' in the filename to be completed. This variable is '`on`' by default.

`menu-complete-display-prefix`
: If set to '`on`', menu completion displays the common prefix of the list of possible completions (which may be empty) before cycling through the list. The default is '`off`'.

`output-meta`
: If set to '`on`', Readline displays characters with the eighth bit set directly rather than as a meta-prefixed escape sequence. The default is '`off`', but Readline sets it to '`on`' if the locale contains characters whose encodings may include bytes with the eighth bit set. This variable is dependent on the `LC_CTYPE` locale category, and its value may change if the locale changes.

`page-completions`
: If set to '`on`', Readline uses an internal pager resembling *more*(1) to display a screenful of possible completions at a time. This variable is '`on`' by default.

`prefer-visible-bell`
: See `bell-style`.

`print-completions-horizontally`
: If set to '`on`', Readline displays completions with matches sorted horizontally in alphabetical order, rather than down the screen. The default is '`off`'.

`revert-all-at-newline`
: If set to '`on`', Readline will undo all changes to history lines before returning when executing `accept-line`. By default, history lines

may be modified and retain individual undo lists across calls to `readline()`. The default is '`off`'.

`search-ignore-case`
: If set to '`on`', Readline performs incremental and non-incremental history list searches in a case-insensitive fashion. The default value is '`off`'.

`show-all-if-ambiguous`
: This alters the default behavior of the completion functions. If set to '`on`', words which have more than one possible completion cause the matches to be listed immediately instead of ringing the bell. The default value is '`off`'.

`show-all-if-unmodified`
: This alters the default behavior of the completion functions in a fashion similar to *show-all-if-ambiguous*. If set to '`on`', words which have more than one possible completion without any possible partial completion (the possible completions don't share a common prefix) cause the matches to be listed immediately instead of ringing the bell. The default value is '`off`'.

`show-mode-in-prompt`
: If set to '`on`', add a string to the beginning of the prompt indicating the editing mode: emacs, vi command, or vi insertion. The mode strings are user-settable (e.g., *emacs-mode-string*). The default value is '`off`'.

`skip-completed-text`
: If set to '`on`', this alters the default completion behavior when inserting a single match into the line. It's only active when performing completion in the middle of a word. If enabled, Readline does not insert characters from the completion that match characters after point in the word being completed, so portions of the word following the cursor are not duplicated. For instance, if this is enabled, attempting completion when the cursor is after the first '`e`' in '`Makefile`' will result in '`Makefile`' rather than '`Makefilefile`', assuming there is a single possible completion. The default value is '`off`'.

`vi-cmd-mode-string`
: If the *show-mode-in-prompt* variable is enabled, this string is displayed immediately before the last line of the primary prompt when vi editing mode is active and in command mode. The value is expanded like a key binding, so the standard set of meta- and control-prefixes and backslash escape sequences is available. The '`\1`' and '`\2`' escapes begin and end sequences of non-printing characters, which can be used to embed a terminal control sequence into the mode string. The default is '`(cmd)`'.

`vi-ins-mode-string`
: If the *show-mode-in-prompt* variable is enabled, this string is displayed immediately before the last line of the primary prompt when vi editing mode is active and in insertion mode. The value is expanded like a key binding, so the standard set of meta- and control-prefixes and backslash escape sequences is available. The '`\1`' and '`\2`' escapes begin and end sequences of non-printing characters, which can be used to embed a terminal control sequence into the mode string. The default is '`(ins)`'.

`visible-stats`
: If set to '`on`', a character denoting a file's type is appended to the filename when listing possible completions. The default is '`off`'.

Key Bindings
: The syntax for controlling key bindings in the init file is simple. First you need to find the name of the command that you want to change. The following sections contain tables of the command name, the default keybinding, if any, and a short description of what the command does.

  Once you know the name of the command, simply place on a line in the init file the name of the key you wish to bind the command to, a colon, and then the name of the command. There can be no space between the key name and the colon – that will be interpreted as part of the key name. The name of the key can be expressed in different ways, depending on what you find most comfortable.

  In addition to command names, Readline allows keys to be bound to a string that is inserted when the key is pressed (a *macro*). The difference between a macro and a command is that a macro is enclosed in single or double quotes.

  keyname: *function-name* or *macro*
  : *keyname* is the name of a key spelled out in English. For example:

    ```
    Control-u: universal-argument
    Meta-Rubout: backward-kill-word
    Control-o: "> output"
    ```

    In the example above, *C-u* is bound to the function `universal-argument`, *M-DEL* is bound to the function `backward-kill-word`, and *C-o* is bound to run the macro expressed on the right hand side (that is, to insert the text '`> output`' into the line).

    This key binding syntax recognizes a number of symbolic character names: *DEL*, *ESC*, *ESCAPE*, *LFD*, *NEWLINE*, *RET*, *RETURN*, *RUBOUT* (a destructive backspace), *SPACE*, *SPC*, and *TAB*.

  "*keyseq*": *function-name* or *macro*
  : *keyseq* differs from *keyname* above in that strings denoting an entire key sequence can be specified, by placing the key sequence in double quotes. Some GNU Emacs style key escapes can be used, as in the

following example, but none of the special character names are recognized.

```
"\C-u": universal-argument
"\C-x\C-r": re-read-init-file
"\e[11~": "Function Key 1"
```

In the above example, *C-u* is again bound to the function `universal-argument` (just as it was in the first example), '*C-x C-r*' is bound to the function `re-read-init-file`, and '`ESC [ 1 1 ~`' is bound to insert the text '`Function Key 1`'.

The following GNU Emacs style escape sequences are available when specifying key sequences:

`\C-` A control prefix.

`\M-` Adding the meta prefix or converting the following character to a meta character, as described above under `force-meta-prefix` (see `Variable Settings` in Section 1.3.1 [Readline Init File Syntax], page 4).

`\e` An escape character.

`\\` Backslash.

`\"` ", a double quotation mark.

`\'` ', a single quote or apostrophe.

In addition to the GNU Emacs style escape sequences, a second set of backslash escapes is available:

`\a` alert (bell)

`\b` backspace

`\d` delete

`\f` form feed

`\n` newline

`\r` carriage return

`\t` horizontal tab

`\v` vertical tab

`\`*nnn* The eight-bit character whose value is the octal value *nnn* (one to three digits).

`\x`*HH* The eight-bit character whose value is the hexadecimal value *HH* (one or two hex digits).

When entering the text of a macro, single or double quotes must be used to indicate a macro definition. Unquoted text is assumed to be a function name. Tthe backslash escapes described above are expanded in the macro body. Backslash will quote any other character in the macro text, including '"' and '''.

For example, the following binding will make '*C-x* \' insert a single '\' into the line:

```
"\C-x\\": "\\"
```

1.3.2 Conditional Init Constructs

Readline implements a facility similar in spirit to the conditional compilation features of the C preprocessor which allows key bindings and variable settings to be performed as the result of tests. There are four parser directives available.

`$if` The `$if` construct allows bindings to be made based on the editing mode, the terminal being used, or the application using Readline. The text of the test, after any comparison operator, extends to the end of the line; unless otherwise noted, no characters are required to isolate it.

`mode` The `mode=` form of the `$if` directive is used to test whether Readline is in `emacs` or `vi` mode. This may be used in conjunction with the '`set keymap`' command, for instance, to set bindings in the `emacs-standard` and `emacs-ctlx` keymaps only if Readline is starting out in `emacs` mode.

`term` The `term=` form may be used to include terminal-specific key bindings, perhaps to bind the key sequences output by the terminal's function keys. The word on the right side of the '`=`' is tested against both the full name of the terminal and the portion of the terminal name before the first '`-`'. This allows `xterm` to match both `xterm` and `xterm-256color`, for instance.

`version` The `version` test may be used to perform comparisons against specific Readline versions. The `version` expands to the current Readline version. The set of comparison operators includes '`=`' (and '`==`'), '`!=`', '`<=`', '`>=`', '`<`', and '`>`'. The version number supplied on the right side of the operator consists of a major version number, an optional decimal point, and an optional minor version (e.g., '`7.1`'). If the minor version is omitted, it defaults to '`0`'. The operator may be separated from the string `version` and from the version number argument by whitespace. The following example sets a variable if the Readline version being used is 7.0 or newer:

```
$if version >= 7.0
set show-mode-in-prompt on
$endif
```

`application`
The *application* construct is used to include application-specific settings. Each program using the Readline library sets the *application name*, and you can test for a particular value. This could be used to bind key sequences to functions useful for a specific program. For instance, the following command adds a key sequence that quotes the current or previous word in Bash:

```
$if Bash
```

```
# Quote the current or previous word
"\C-xq": "\eb\"\ef\""
$endif
```

`variable` The *variable* construct provides simple equality tests for Readline variables and values. The permitted comparison operators are '`=`', '`==`', and '`!=`'. The variable name must be separated from the comparison operator by whitespace; the operator may be separated from the value on the right hand side by whitespace. String and boolean variables may be tested. Boolean variables must be tested against the values *on* and *off*. The following example is equivalent to the `mode=emacs` test described above:

```
$if editing-mode == emacs
set show-mode-in-prompt on
$endif
```

`$else` Commands in this branch of the `$if` directive are executed if the test fails.

`$endif` This command, as seen in the previous example, terminates an `$if` command.

`$include` This directive takes a single filename as an argument and reads commands and key bindings from that file. For example, the following directive reads from `/etc/inputrc`:

```
$include /etc/inputrc
```

1.3.3 Sample Init File

Here is an example of an *inputrc* file. This illustrates key binding, variable assignment, and conditional syntax.

```
# This file controls the behaviour of line input editing for
# programs that use the GNU Readline library.  Existing
# programs include FTP, Bash, and GDB.
#
# You can re-read the inputrc file with C-x C-r.
# Lines beginning with '#' are comments.
#
# First, include any system-wide bindings and variable
# assignments from /etc/Inputrc
$include /etc/Inputrc

#
# Set various bindings for emacs mode.

set editing-mode emacs

$if mode=emacs

Meta-Control-h:  backward-kill-word Text after the function name is ignored

#
# Arrow keys in keypad mode
#
#"\M-OD":        backward-char
#"\M-OC":        forward-char
#"\M-OA":        previous-history
#"\M-OB":        next-history
#
# Arrow keys in ANSI mode
#
"\M-[D":        backward-char
"\M-[C":        forward-char
"\M-[A":        previous-history
"\M-[B":        next-history
#
# Arrow keys in 8 bit keypad mode
#
#"\M-\C-OD":       backward-char
#"\M-\C-OC":       forward-char
#"\M-\C-OA":       previous-history
#"\M-\C-OB":       next-history
#
# Arrow keys in 8 bit ANSI mode
#
#"\M-\C-[D":       backward-char
#"\M-\C-[C":       forward-char
```

```
#"\M-\C-[A":       previous-history
#"\M-\C-[B":       next-history

C-q: quoted-insert

$endif

# An old-style binding.  This happens to be the default.
TAB: complete

# Macros that are convenient for shell interaction
$if Bash
# edit the path
"\C-xp": "PATH=${PATH}\e\C-e\C-a\ef\C-f"
# prepare to type a quoted word --
# insert open and close double quotes
# and move to just after the open quote
"\C-x\"": "\"\"\C-b"
# insert a backslash (testing backslash escapes
# in sequences and macros)
"\C-x\\": "\\"
# Quote the current or previous word
"\C-xq": "\eb\"\ef\""
# Add a binding to refresh the line, which is unbound
"\C-xr": redraw-current-line
# Edit variable on current line.
"\M-\C-v": "\C-a\C-k$\C-y\M-\C-e\C-a\C-y="
$endif

# use a visible bell if one is available
set bell-style visible

# don't strip characters to 7 bits when reading
set input-meta on

# allow iso-latin1 characters to be inserted rather
# than converted to prefix-meta sequences
set convert-meta off

# display characters with the eighth bit set directly
# rather than as meta-prefixed characters
set output-meta on

# if there are 150 or more possible completions for a word,
# ask whether or not the user wants to see all of them
set completion-query-items 150
```

```
# For FTP
$if Ftp
"\C-xg": "get \M-?"
"\C-xt": "put \M-?"
"\M-.": yank-last-arg
$endif
```

1.4 Bindable Readline Commands

This section describes Readline commands that may be bound to key sequences. Command names without an accompanying key sequence are unbound by default.

In the following descriptions, *point* refers to the current cursor position, and *mark* refers to a cursor position saved by the `set-mark` command. The text between the point and mark is referred to as the *region*. Readline has the concept of an *active region*: when the region is active, Readline redisplay highlights the region using the value of the `active-region-start-color` variable. The `enable-active-region` variable turns this on and off. Several commands set the region to active; those are noted below.

1.4.1 Commands For Moving

`beginning-of-line (C-a)`
: Move to the start of the current line. This may also be bound to the Home key on some keyboards.

`end-of-line (C-e)`
: Move to the end of the line. This may also be bound to the End key on some keyboards.

`forward-char (C-f)`
: Move forward a character. This may also be bound to the right arrow key on some keyboards.

`backward-char (C-b)`
: Move back a character. This may also be bound to the left arrow key on some keyboards.

`forward-word (M-f)`
: Move forward to the end of the next word. Words are composed of letters and digits.

`backward-word (M-b)`
: Move back to the start of the current or previous word. Words are composed of letters and digits.

`previous-screen-line ()`
: Attempt to move point to the same physical screen column on the previous physical screen line. This will not have the desired effect if the current Readline line does not take up more than one physical line or if point is not greater than the length of the prompt plus the screen width.

`next-screen-line ()`
: Attempt to move point to the same physical screen column on the next physical screen line. This will not have the desired effect if the current Readline line does

not take up more than one physical line or if the length of the current Readline line is not greater than the length of the prompt plus the screen width.

`clear-display (M-C-l)`
Clear the screen and, if possible, the terminal's scrollback buffer, then redraw the current line, leaving the current line at the top of the screen.

`clear-screen (C-l)`
Clear the screen, then redraw the current line, leaving the current line at the top of the screen.

`redraw-current-line ()`
Refresh the current line. By default, this is unbound.

1.4.2 Commands For Manipulating The History

`accept-line (Newline or Return)`
Accept the line regardless of where the cursor is. If this line is non-empty, you can add it to the history list using `add_history()`. If this line is a modified history line, then restore the history line to its original state.

`previous-history (C-p)`
Move 'back' through the history list, fetching the previous command. This may also be bound to the up arrow key on some keyboards.

`next-history (C-n)`
Move 'forward' through the history list, fetching the next command. This may also be bound to the down arrow key on some keyboards.

`beginning-of-history (M-<)`
Move to the first line in the history.

`end-of-history (M->)`
Move to the end of the input history, i.e., the line currently being entered.

`reverse-search-history (C-r)`
Search backward starting at the current line and moving 'up' through the history as necessary. This is an incremental search. This command sets the region to the matched text and activates the region.

`forward-search-history (C-s)`
Search forward starting at the current line and moving 'down' through the history as necessary. This is an incremental search. This command sets the region to the matched text and activates the region.

`non-incremental-reverse-search-history (M-p)`
Search backward starting at the current line and moving 'up' through the history as necessary using a non-incremental search for a string supplied by the user. The search string may match anywhere in a history line.

`non-incremental-forward-search-history (M-n)`
Search forward starting at the current line and moving 'down' through the history as necessary using a non-incremental search for a string supplied by the user. The search string may match anywhere in a history line.

`history-search-backward ()`
: Search backward through the history for the string of characters between the start of the current line and the point. The search string must match at the beginning of a history line. This is a non-incremental search. By default, this command is unbound, but may be bound to the Page Down key on some keyboards.

`history-search-forward ()`
: Search forward through the history for the string of characters between the start of the current line and the point. The search string must match at the beginning of a history line. This is a non-incremental search. By default, this command is unbound, but may be bound to the Page Up key on some keyboards.

`history-substring-search-backward ()`
: Search backward through the history for the string of characters between the start of the current line and the point. The search string may match anywhere in a history line. This is a non-incremental search. By default, this command is unbound.

`history-substring-search-forward ()`
: Search forward through the history for the string of characters between the start of the current line and the point. The search string may match anywhere in a history line. This is a non-incremental search. By default, this command is unbound.

`yank-nth-arg (M-C-y)`
: Insert the first argument to the previous command (usually the second word on the previous line) at point. With an argument *n*, insert the *n*th word from the previous command (the words in the previous command begin with word 0). A negative argument inserts the *n*th word from the end of the previous command. Once the argument *n* is computed, this uses the history expansion facilities to extract the *n*th word, as if the '`!`*n*' history expansion had been specified.

`yank-last-arg (M-. or M-_)`
: Insert last argument to the previous command (the last word of the previous history entry). With a numeric argument, behave exactly like `yank-nth-arg`. Successive calls to `yank-last-arg` move back through the history list, inserting the last word (or the word specified by the argument to the first call) of each line in turn. Any numeric argument supplied to these successive calls determines the direction to move through the history. A negative argument switches the direction through the history (back or forward). This uses the history expansion facilities to extract the last word, as if the '`!$`' history expansion had been specified.

`operate-and-get-next (C-o)`
: Accept the current line for return to the calling application as if a newline had been entered, and fetch the next line relative to the current line from the history for editing. A numeric argument, if supplied, specifies the history entry to use instead of the current line.

`fetch-history ()`
: With a numeric argument, fetch that entry from the history list and make it the current line. Without an argument, move back to the first entry in the history list.

1.4.3 Commands For Changing Text

`end-of-file` (usually `C-d`)
: The character indicating end-of-file as set, for example, by `stty`. If this character is read when there are no characters on the line, and point is at the beginning of the line, Readline interprets it as the end of input and returns EOF.

`delete-char (C-d)`
: Delete the character at point. If this function is bound to the same character as the tty EOF character, as *`C-d`* commonly is, see above for the effects. This may also be bound to the Delete key on some keyboards.

`backward-delete-char (Rubout)`
: Delete the character behind the cursor. A numeric argument means to kill the characters, saving them on the kill ring, instead of deleting them.

`forward-backward-delete-char ()`
: Delete the character under the cursor, unless the cursor is at the end of the line, in which case the character behind the cursor is deleted. By default, this is not bound to a key.

`quoted-insert (C-q or C-v)`
: Add the next character typed to the line verbatim. This is how to insert key sequences like *`C-q`*, for example.

`tab-insert (M-TAB)`
: Insert a tab character.

`self-insert (a, b, A, 1, !, ...)`
: Insert the character typed.

`bracketed-paste-begin ()`
: This function is intended to be bound to the "bracketed paste" escape sequence sent by some terminals, and such a binding is assigned by default. It allows Readline to insert the pasted text as a single unit without treating each character as if it had been read from the keyboard. The characters are inserted as if each one was bound to `self-insert` instead of executing any editing commands.

  Bracketed paste sets the region (the characters between point and the mark) to the inserted text. It sets the *active region*.

`transpose-chars (C-t)`
: Drag the character before the cursor forward over the character at the cursor, moving the cursor forward as well. If the insertion point is at the end of the line, then this transposes the last two characters of the line. Negative arguments have no effect.

`transpose-words (M-t)`
: Drag the word before point past the word after point, moving point past that word as well. If the insertion point is at the end of the line, this transposes the last two words on the line.

`upcase-word (M-u)`
: Uppercase the current (or following) word. With a negative argument, uppercase the previous word, but do not move the cursor.

`downcase-word (M-l)`
: Lowercase the current (or following) word. With a negative argument, lowercase the previous word, but do not move the cursor.

`capitalize-word (M-c)`
: Capitalize the current (or following) word. With a negative argument, capitalize the previous word, but do not move the cursor.

`overwrite-mode ()`
: Toggle overwrite mode. With an explicit positive numeric argument, switches to overwrite mode. With an explicit non-positive numeric argument, switches to insert mode. This command affects only `emacs` mode; `vi` mode does overwrite differently. Each call to `readline()` starts in insert mode.

  In overwrite mode, characters bound to `self-insert` replace the text at point rather than pushing the text to the right. Characters bound to `backward-delete-char` replace the character before point with a space.

  By default, this command is unbound, but may be bound to the Insert key on some keyboards.

1.4.4 Killing And Yanking

`kill-line (C-k)`
: Kill the text from point to the end of the current line. With a negative numeric argument, kill backward from the cursor to the beginning of the line.

`backward-kill-line (C-x Rubout)`
: Kill backward from the cursor to the beginning of the current line. With a negative numeric argument, kill forward from the cursor to the end of the line.

`unix-line-discard (C-u)`
: Kill backward from the cursor to the beginning of the current line.

`kill-whole-line ()`
: Kill all characters on the current line, no matter where point is. By default, this is unbound.

`kill-word (M-d)`
: Kill from point to the end of the current word, or if between words, to the end of the next word. Word boundaries are the same as `forward-word`.

`backward-kill-word (M-DEL)`
: Kill the word behind point. Word boundaries are the same as `backward-word`.

`unix-word-rubout (C-w)`
: Kill the word behind point, using white space as a word boundary, saving the killed text on the kill-ring.

`unix-filename-rubout ()`
: Kill the word behind point, using white space and the slash character as the word boundaries, saving the killed text on the kill-ring.

`delete-horizontal-space ()`
: Delete all spaces and tabs around point. By default, this is unbound.

`kill-region ()`
: Kill the text in the current region. By default, this command is unbound.

`copy-region-as-kill ()`
: Copy the text in the region to the kill buffer, so it can be yanked right away. By default, this command is unbound.

`copy-backward-word ()`
: Copy the word before point to the kill buffer. The word boundaries are the same as `backward-word`. By default, this command is unbound.

`copy-forward-word ()`
: Copy the word following point to the kill buffer. The word boundaries are the same as `forward-word`. By default, this command is unbound.

`yank (C-y)`
: Yank the top of the kill ring into the buffer at point.

`yank-pop (M-y)`
: Rotate the kill-ring, and yank the new top. You can only do this if the prior command is `yank` or `yank-pop`.

1.4.5 Specifying Numeric Arguments

`digit-argument (`*`M-0`*`, `*`M-1`*`, ... `*`M--`*`)`
: Add this digit to the argument already accumulating, or start a new argument. *M--* starts a negative argument.

`universal-argument ()`
: This is another way to specify an argument. If this command is followed by one or more digits, optionally with a leading minus sign, those digits define the argument. If the command is followed by digits, executing `universal-argument` again ends the numeric argument, but is otherwise ignored. As a special case, if this command is immediately followed by a character that is neither a digit nor minus sign, the argument count for the next command is multiplied by four. The argument count is initially one, so executing this function the first time makes the argument count four, a second time makes the argument count sixteen, and so on. By default, this is not bound to a key.

1.4.6 Letting Readline Type For You

`complete (TAB)`
: Attempt to perform completion on the text before point. The actual completion performed is application-specific. The default is filename completion.

`possible-completions (M-?)`
: List the possible completions of the text before point. When displaying completions, Readline sets the number of columns used for display to the value of `completion-display-width`, the value of the environment variable `COLUMNS`, or the screen width, in that order.

`insert-completions (M-*)`
: Insert all completions of the text before point that would have been generated by `possible-completions`, separated by a space.

`menu-complete ()`
: Similar to `complete`, but replaces the word to be completed with a single match from the list of possible completions. Repeatedly executing `menu-complete` steps through the list of possible completions, inserting each match in turn. At the end of the list of completions, `menu-complete` rings the bell (subject to the setting of `bell-style`) and restores the original text. An argument of *n* moves *n* positions forward in the list of matches; a negative argument moves backward through the list. This command is intended to be bound to `TAB`, but is unbound by default.

`menu-complete-backward ()`
: Identical to `menu-complete`, but moves backward through the list of possible completions, as if `menu-complete` had been given a negative argument. This command is unbound by default.

`export-completions ()`
: Perform completion on the word before point as described above and write the list of possible completions to Readline's output stream using the following format, writing information on separate lines:

  - the number of matches *N*;
  - the word being completed;
  - *S*:*E*, where S and E are the start and end offsets of the word in the Readline line buffer; then
  - each match, one per line

  If there are no matches, the first line will be "0", and this command does not print any output after the *S*:*E*. If there is only a single match, this prints a single line containing it. If there is more than one match, this prints the common prefix of the matches, which may be empty, on the first line after the *S*:*E*, then the matches on subsequent lines. In this case, *N* will include the first line with the common prefix.

  The user or application should be able to accommodate the possibility of a blank line. The intent is that the user or application reads *N* lines after the line containing *S*:*E* to obtain the match list. This command is unbound by default.

`delete-char-or-list ()`
: Deletes the character under the cursor if not at the beginning or end of the line (like `delete-char`). At the end of the line, it behaves identically to `possible-completions`. This command is unbound by default.

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 definition.

`call-last-kbd-macro (C-x e)`
: Re-execute the last keyboard macro defined, by making the characters in the macro appear as if typed at the keyboard.

`print-last-kbd-macro ()`
: Print the last keyboard macro defined in a format suitable for the *inputrc* file.

1.4.8 Some Miscellaneous Commands

`re-read-init-file (C-x C-r)`
: Read in the contents of the *inputrc* file, and incorporate any bindings or variable assignments found there.

`abort (C-g)`
: Abort the current editing command and ring the terminal's bell (subject to the setting of `bell-style`).

`do-lowercase-version (M-A, M-B, M-x, ...)`
: If the metafied character *x* is upper case, run the command that is bound to the corresponding metafied lower case character. The behavior is undefined if *x* is already lower case.

`prefix-meta (ESC)`
: Metafy the next character typed. Typing '`ESC f`' is equivalent to typing *M-f*.

`undo (C-_ or C-x C-u)`
: Incremental undo, separately remembered for each line.

`revert-line (M-r)`
: Undo all changes made to this line. This is like executing the `undo` command enough times to get back to the initial state.

`tilde-expand (M-~)`
: Perform tilde expansion on the current word.

`set-mark (C-@)`
: Set the mark to the point. If a numeric argument is supplied, set the mark to that position.

`exchange-point-and-mark (C-x C-x)`
: Swap the point with the mark. Set the current cursor position to the saved position, then set the mark to the old cursor position.

`character-search (C-])`
: Read a character and move point to the next occurrence of that character. A negative argument searches for previous occurrences.

`character-search-backward (M-C-])`
: Read a character and move point to the previous occurrence of that character. A negative argument searches for subsequent occurrences.

`skip-csi-sequence ()`
: Read enough characters to consume a multi-key sequence such as those defined for keys like Home and End. CSI sequences begin with a Control Sequence Indicator (CSI), usually *ESC [*. If this sequence is bound to "\e[", keys producing CSI sequences have no effect unless explicitly bound to a Readline command, instead of inserting stray characters into the editing buffer. This is unbound by default, but usually bound to *ESC [*.

`insert-comment (M-#)`
: Without a numeric argument, insert the value of the `comment-begin` variable at the beginning of the current line. If a numeric argument is supplied, this command acts as a toggle: if the characters at the beginning of the line do not match the value of `comment-begin`, insert the value; otherwise delete the characters in `comment-begin` from the beginning of the line. In either case, the line is accepted as if a newline had been typed.

`dump-functions ()`
: Print all of the functions and their key bindings to the Readline output stream. If a numeric argument is supplied, the output is formatted in such a way that it can be made part of an *inputrc* file. This command is unbound by default.

`dump-variables ()`
: Print all of the settable variables and their values to the Readline output stream. If a numeric argument is supplied, the output is formatted in such a way that it can be made part of an *inputrc* file. This command is unbound by default.

`dump-macros ()`
: Print all of the Readline key sequences bound to macros and the strings they output to the Readline output stream. If a numeric argument is supplied, the output is formatted in such a way that it can be made part of an *inputrc* file. This command is unbound by default.

`execute-named-command (M-x)`
: Read a bindable Readline command name from the input and execute the function to which it's bound, as if the key sequence to which it was bound appeared in the input. If this function is supplied with a numeric argument, it passes that argument to the function it executes.

`emacs-editing-mode (C-e)`
: When in `vi` command mode, this causes a switch to `emacs` editing mode.

`vi-editing-mode (M-C-j)`
: When in `emacs` editing mode, this causes a switch to `vi` editing mode.

1.5 Readline vi Mode

While the Readline library does not have a full set of `vi` editing functions, it does contain enough to allow simple editing of the line. The Readline `vi` mode behaves as specified in the `sh` description in the POSIX standard.

In order to switch interactively between `emacs` and `vi` editing modes, use the command *M-C-j* (bound to emacs-editing-mode when in `vi` mode and to vi-editing-mode in `emacs` mode). The Readline default is `emacs` mode.

When you enter a line in `vi` mode, you are already placed in 'insertion' mode, as if you had typed an '`i`'. Pressing `ESC` switches you into 'command' mode, where you can edit the text of the line with the standard `vi` movement keys, move to previous history lines with '`k`' and subsequent lines with '`j`', and so forth.

Appendix A GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.
`http://fsf.org/`

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

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.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released

under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

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 notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

   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.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their

titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `http://www.gnu.org/copyleft/`.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

    "Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

    "CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

    "Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

    An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

    The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

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 document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with. . . Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.