

# GNU Readline Library

Brian Fox

Free Software Foundation

Version 1.1

April 1991

This document describes the end user interface of the GNU Readline Library, a utility which aids in the consistency of user interface across discrete programs that need to provide a command line interface.

Published by the Free Software Foundation  
675 Massachusetts Avenue,  
Cambridge, MA 02139 USA

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be stated in a translation approved by the Foundation.

## Appendix A Command Line Editing

This text describes GNU's command line editing interface.

### A.1 Introduction to Line Editing

The following paragraphs describe the notation we use to represent keystrokes.

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

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 struck. If you do not have a meta key, the identical keystroke can be generated by typing **ESC** *first*, and then typing K. Either process is known as *metafying* the K key.

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 A.3 [Readline Init File], page 4, for more info).

### A.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 **RETURN**. You do not have to be at the end of the line to press **RETURN**; the entire line is accepted regardless of the location of the cursor within the line.

### A.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 `DEL` to back up, and delete the mistyped character.

Sometimes you may miss typing a character that you wanted to type, and not notice your 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 get ‘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 get ‘pulled back’ to fill in the blank space created by the removal of the text. A list of the basic bare essentials for editing the text of an input line follows.

<code>C-B</code>	Move back one character.
<code>C-F</code>	Move forward one character.
<code>DEL</code>	Delete the character to the left of the cursor.
<code>C-D</code>	Delete the character underneath the cursor.
Printing characters	
	Insert itself into the line at the cursor.
<code>C-_</code>	Undo the last thing that you did. You can undo all the way back to an empty line.

### A.2.2 Readline Movement Commands

The above table describes the most basic possible keystrokes that you need in order to do editing of the input line. For your convenience, many other commands have been added in addition to `C-B`, `C-F`, `C-D`, and `DEL`. Here are some commands for moving more rapidly about the line.

<code>C-A</code>	Move to the start of the line.
<code>C-E</code>	Move to the end of the line.
<code>M-F</code>	Move forward a word.
<code>M-B</code>	Move backward a word.
<code>C-L</code>	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.

### A.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* it back into the line. 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.

Here is the list of commands for killing text.

<b>C-K</b>	Kill the text from the current cursor position to the end of the line.
<b>M-D</b>	Kill from the cursor to the end of the current word, or if between words, to the end of the next word.
<b>M-DEL</b>	Kill from the cursor to the start of the previous word, or if between words, to the start of the previous word.
<b>C-W</b>	Kill from the cursor to the previous whitespace. This is different than <b>M-DEL</b> because the word boundaries differ.

And, here is how to *yank* the text back into the line. Yanking is

<b>C-Y</b>	Yank the most recently killed text back into the buffer at the cursor.
<b>M-Y</b>	Rotate the kill-ring, and yank the new top. You can only do this if the prior command is <b>C-Y</b> or <b>M-Y</b> .

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 in one clean sweep. 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.

### A.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’ you type 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**.

## A.3 Readline Init File

Although the Readline library comes with a set of Emacs-like keybindings, it is possible that you would like to use a different set of keybindings. You can customize programs that use Readline by putting commands in an *init* file in your home directory. The name of this file is ‘`~/.inputrc`’.

When a program which uses the Readline library starts up, the ‘`~/.inputrc`’ file is read, and the keybindings are set.

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

### A.3.1 Readline Init Syntax

There are only four constructs allowed in the ‘`~/.inputrc`’ file:

#### Variable Settings

You can change the state of a few variables in Readline. You do this by using the **set** command within the init file. Here is how you would specify that you wish to use Vi line editing commands:

```
set editing-mode vi
```

Right now, there are only a few variables which can be set; so few in fact, that we just iterate them here:

```
editing-mode
```

The **editing-mode** variable controls which editing mode you are using. By default, GNU Readline starts up in Emacs editing mode, where the keystrokes are most similar to Emacs. This variable can either be set to **emacs** or **vi**.

```
horizontal-scroll-mode
```

This variable can either be set to **On** or **Off**. Setting it to **On** means that

the text of the lines that you edit will scroll horizontally on a single screen line when they are larger than the width of the screen, instead of wrapping onto a new screen line. By default, this variable is set to `Off`.

#### `mark-modified-lines`

This variable when set to `On`, says to display an asterisk (\*) at the starts of history lines which have been modified. This variable is off by default.

#### `prefer-visible-bell`

If this variable is set to `On` it means to use a visible bell if one is available, rather than simply ringing the terminal bell. By default, the value is `Off`.

### Key Bindings

The syntax for controlling keybindings in the `~/.inputrc` file is simple. First you have to know the *name* of the command that you want to change. The following pages contain tables of the command name, the default keybinding, and a short description of what the command does.

Once you know the name of the command, simply place the name of the key you wish to bind the command to, a colon, and then the name of the command on a line in the `~/.inputrc` file. The name of the key can be expressed in different ways, depending on which is most comfortable for you.

*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 above example, `C-U` is bound to the function `universal-argument`, 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).

*"keyseq"*: *function-name* or *macro*

*keyseq* differs from *keyname* above in that strings denoting an entire key sequence can be specified. Simply place the key sequence in double quotes. GNU Emacs style key escapes can be used, as in the following example:

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

In the above example, `C-U` is 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'`.

### A.3.1.1 Commands For Moving

**beginning-of-line (C-A)**

Move to the start of the current line.

**end-of-line (C-E)**

Move to the end of the line.

**forward-char (C-F)**

Move forward a character.

**backward-char (C-B)**

Move back a character.

**forward-word (M-F)**

Move forward to the end of the next word.

**backward-word (M-B)**

Move back to the start of this, or the previous, word.

**clear-screen (C-L)**

Clear the screen leaving the current line at the top of the screen.

### **A.3.1.2 Commands For Manipulating The History**

**accept-line (Newline, Return)**

Accept the line regardless of where the cursor is. If this line is non-empty, add it to the history list. If this line was a history line, then restore the history line to its original state.

**previous-history (C-P)**

Move ‘up’ through the history list.

**next-history (C-N)**

Move ‘down’ through the history list.

**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 you are entering!

**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.

**forward-search-history (C-S)**

Search forward starting at the current line and moving ‘down’ through the the history as necessary.



### A.3.1.3 Commands For Changing Text

#### `delete-char` (C-D)

Delete the character under the cursor. If the cursor is at the beginning of the line, and there are no characters in the line, and the last character typed was not C-D, then return EOF.

#### `backward-delete-char` (Rubout)

Delete the character behind the cursor. A numeric arg says to kill the characters instead of deleting them.

#### `quoted-insert` (C-Q, C-V)

Add the next character that you type to the line verbatim. This is how to insert things like C-Q for example.

#### `tab-insert` (M-TAB)

Insert a tab character.

#### `self-insert` (a, b, A, 1, !, ...)

Insert yourself.

#### `transpose-chars` (C-T)

Drag the character before point forward over the character at point. Point moves forward as well. If point is at the end of the line, then transpose the two characters before point. Negative args don't work.

#### `transpose-words` (M-T)

Drag the word behind the cursor past the word in front of the cursor moving the cursor over that word as well.

#### `upcase-word` (M-U)

Uppercase all letters in the current (or following) word. With a negative argument, do the previous word, but do not move point.

#### `downcase-word` (M-L)

Lowercase all letters in the current (or following) word. With a negative argument, do the previous word, but do not move point.

#### `capitalize-word` (M-C)

Uppercase the first letter in the current (or following) word. With a negative argument, do the previous word, but do not move point.

### A.3.1.4 Killing And Yanking

#### `kill-line` (C-K)

Kill the text from the current cursor position to the end of the line.

**backward-kill-line** (**C**)

Kill backward to the beginning of the line. This is normally unbound.

**kill-word** (**M-D**)

Kill from the cursor to the end of the current word, or if between words, to the end of the next word.

**backward-kill-word** (**M-DEL**)

Kill the word behind the cursor.

**unix-line-discard** (**C-U**)

Do what **C-U** used to do in Unix line input. We save the killed text on the kill-ring, though.

**unix-word-rubout** (**C-W**)

Do what **C-W** used to do in Unix line input. The killed text is saved on the kill-ring. This is different than **backward-kill-word** because the word boundaries differ.

**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**.

### A.3.1.5 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** (**C**)

Do what **C-U** does in emacs. By default, this is not bound.

### A.3.1.6 Letting Readline Type For You

**complete** (**TAB**)

Attempt to do completion on the text before point. This is implementation defined. Generally, if you are typing a filename argument, you can do filename completion; if you are typing a command, you can do command completion, if you are typing in a symbol to GDB, you can do symbol name completion, if you are typing in a variable to Bash, you can do variable name completion...

**possible-completions** (**M-?**)

List the possible completions of the text before point.

### A.3.1.7 Some Miscellaneous Commands

**re-read-init-file (C-X C-R)**

Read in the contents of your ‘~/inputrc’ file, and incorporate any bindings found there.

**abort (C-G)**

Stop running the current editing command.

**prefix-meta (ESC)**

Make the next character that you type be metafied. This is for people without a meta key. Typing ESC F is equivalent to typing M-F.

**undo (C-\_)**

Incremental undo, separately remembered for each line.

**revert-line (M-R)**

Undo all changes made to this line. This is like typing the ‘undo’ command enough times to get back to the beginning.

### A.3.2 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.

In order to switch interactively between Emacs and Vi editing modes, use the command M-C-J (toggle-editing-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 ‘edit’ mode, where you can edit the text of the line with the standard Vi movement keys, move to previous history lines with ‘k’, and following lines with ‘j’, and so forth.



# Table of Contents

<b>Appendix A</b>	<b>Command Line Editing.....</b>	<b>1</b>
A.1	Introduction to Line Editing.....	1
A.2	Readline Interaction.....	1
A.2.1	Readline Bare Essentials.....	2
A.2.2	Readline Movement Commands.....	2
A.2.3	Readline Killing Commands.....	3
A.2.4	Readline Arguments.....	3
A.3	Readline Init File.....	4
A.3.1	Readline Init Syntax.....	4
A.3.1.1	Commands For Moving.....	5
A.3.1.2	Commands For Manipulating The History.....	6
A.3.1.3	Commands For Changing Text.....	7
A.3.1.4	Killing And Yanking.....	7
A.3.1.5	Specifying Numeric Arguments.....	8
A.3.1.6	Letting Readline Type For You.....	8
A.3.1.7	Some Miscellaneous Commands.....	9
A.3.2	Readline Vi Mode.....	9

