NAME

readline − get a line from a user with editing

SYNOPSIS

#include <stdio.h>
#include <readline/readline.h>
#include <readline/history.h>

*char **
readline (*const char *prompt*);

COPYRIGHT

DESCRIPTION

readline reads a line from the terminal and return it, using **prompt** as a prompt. If **prompt** is **NULL** or the empty string, **readline** does not issue a prompt. The line returned is allocated with *malloc*(3); the caller must free it when finished. The line returned has the final newline removed, so only the text of the line remains. Since it's possible to enter characters into the line while quoting them to disable any **readline** editing function they might normally have, this line may include embedded newlines and other special characters.

readline offers editing capabilities while the user is entering the line. By default, the line editing commands are similar to those of emacs. A vi−style line editing interface is also available.

This manual page describes only the most basic use of **readline**. Much more functionality is available; see *The GNU Readline Library* and *The GNU History Library* for additional information.

RETURN VALUE

readline returns the text of the line read. A blank line returns the empty string. If **EOF** is encountered while reading a line, and the line is empty, **readline** returns **NULL**. If an **EOF** is read with a non−empty line, it is treated as a newline.

NOTATION

This section uses Emacs-style editing concepts and uses its notation for keystrokes. Control keys are denoted by C−*key*, e.g., C−n means Control−N. Similarly, *meta* keys are denoted by M−*key*, so M−x means Meta−X. The Meta key is often labeled "Alt" or "Option".

On keyboards without a *Meta* key, M−*x* means ESC *x*, i.e., press and release the Escape key, then press and release the *x* key, in sequence. This makes ESC the *meta prefix*. The combination M−C−*x* means ESC Control−*x*: press and release the Escape key, then press and hold the Control key while pressing the *x* key, then release both.

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 preceding paragraph.

If your *Meta* key produces a key sequence with the ESC meta prefix, you can make M-*key* key bindings you specify (see **Readline Key Bindings** below) do the same thing by setting the **force−meta−prefix** variable.

Readline commands may be given numeric *arguments*, which normally act as a repeat count. Sometimes, however, it is the sign of the argument that is significant. Passing a negative argument to a command that acts in the forward direction (e.g., **kill−line**) makes that command act in a backward direction. Commands whose behavior with arguments deviates from this are noted below.

The *point* is the current cursor position, and *mark* refers to a saved cursor position. The text between the point and mark is referred to as the *region*.

When a command is described as *killing* text, the text deleted is saved for possible future retrieval (*yanking*). The killed text is saved in a *kill ring*. Consecutive kills accumulate the deleted text into one unit, which can be yanked all at once. Commands which do not kill text separate the chunks of text on the kill ring.

INITIALIZATION FILE

Readline is customized by putting commands in an initialization file (the *inputrc* file). The name of this file is taken from the value of the **INPUTRC** environment variable. 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 initialization file and sets the key bindings and variables found there, before reading any user input.

There are only a few basic constructs allowed in the inputrc file. Blank lines are ignored. Lines beginning with a **#** are comments. Lines beginning with a **$** indicate conditional constructs. Other lines denote key bindings and variable settings.

The default key-bindings in this document may be changed using key binding commands in the *inputrc* file. Programs that use this library may add their own commands and bindings.

For example, placing

M−Control−u: universal−argument

or

C−Meta−u: universal−argument

into the *inputrc* would make M−C−u execute the **readline** command *universal−argument*.

Key bindings may contain the following symbolic character names: *DEL*, *ESC*, *ESCAPE*, *LFD*, *NEWLINE*, *RET*, *RETURN*, *RUBOUT* (a destructive backspace), *SPACE*, *SPC*, and *TAB*.

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.

Key Bindings

The syntax for controlling key bindings in the *inputrc* file is simple. All that is required is the name of the command or the text of a macro and a key sequence to which it should be bound. The key sequence may be specified in one of two ways: as a symbolic key name, possibly with *Meta−* or *Control−* prefixes, or as a key sequence composed of one or more characters enclosed in double quotes. The key sequence and name are separated by a colon. There can be no whitespace between the name and the colon.

When using the form **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**, *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).

In the second form, **"keyseq"**:*function−name* or *macro*, **keyseq** differs from **keyname** above in that strings denoting an entire key sequence may be specified by placing the sequence within double quotes. Some GNU Emacs style key escapes can be used, as in the following example, but none of the symbolic character names are recognized.

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

In this example, *C−u* is again bound to the function **universal−argument**. *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 full set of GNU Emacs style escape sequences available when specifying key sequences is

\C− A control prefix.

\M− Adding the meta prefix or converting the following character to a meta character, as described below under **force-meta-prefix**.
\e An escape character.
**** Backslash.
\" Literal ", a double quote.
\' Literal ', a single quote.

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. The backslash escapes described above are expanded in the macro body. Backslash quotes any other character in the macro text, including " and '.

Bash will display or modify the current **readline** key bindings with the **bind** builtin command. The **−o emacs** or **−o vi** options to the **set** builtin change the editing mode during interactive use. Other programs using this library provide similar mechanisms. A user may always edit the *inputrc* file and have **readline** re-read it if a program does not provide any other means to incorporate new bindings.

Variables

Readline has variables that can be used to further customize its behavior. A variable may be set in the *inputrc* file with a statement of the form

set *variable−name value*

Except where noted, **readline** variables can take the values **On** or **Off** (without regard to case). Unrecognized variable names are ignored. When **readline** reads a variable value, empty or null values, "on" (case-insensitive), and "1" are equivalent to **On**. All other values are equivalent to **Off**.

The variables and their default values are:

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 (audible)
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, **readline** attempts to ring the terminal's bell.

bind−tty−special−chars (On)
If set to **On**, **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 (Off)
If set to **On**, **readline** attempts to briefly move the cursor to an opening parenthesis when a closing parenthesis is inserted.

colored−completion−prefix (Off)
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.

colored−stats (Off)
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.

comment−begin ("**#**")
The string that the **readline insert−comment** command inserts. This command is bound to **M−#** in emacs mode and to **#** in vi command mode.

completion−display−width (−1)
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 (Off)
If set to **On**, **readline** performs filename matching and completion in a case−insensitive fashion.

completion−map−case (Off)
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.

completion−prefix−display−length (0)
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. If a completion begins with a period, and **eadline** is completing filenames, it uses three underscores instead of an ellipsis.

completion−query−items (100)
This determines when the user is queried about viewing the number of possible completions generated by the **possible−completions** command. It may be set to any integer value greater than or equal to zero. If the number of possible completions is greater than or equal to the value of this variable, **readline** asks whether or not the user wishes to view them; otherwise **readline** simply lists them on the terminal. A zero value means **readline** should never ask; negative values are treated as zero.

convert−meta (On)
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 it with an escape character (converting the character to have the meta prefix). The default 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 (Off)
If set to **On**, **readline** inhibits word completion. Completion characters are inserted into the line as if they had been mapped to **self-insert**.

echo−control−characters (On)
When set to **On**, on operating systems that indicate they support it, **readline** echoes a character corresponding to a signal generated from the keyboard.

editing−mode (emacs)
Controls whether **readline** uses a set of key bindings similar to *Emacs* or *vi*. **editing−mode** 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.

enable−active−region (On)
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** variable, 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.

enable−bracketed−paste (On)
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 *bracketed−paste mode*; it prevents **readline** from executing any editing commands bound to key sequences appearing in the pasted text.

enable−keypad (Off)
When set to **On**, **readline** tries to enable the application keypad when it is called. Some systems need this to enable the arrow keys.

enable−meta−key (On)
When set to **On**, **readline** tries to enable any meta modifier key the terminal claims to support. 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).

expand−tilde (Off)
If set to **On**, **readline** performs tilde expansion when it attempts word completion.

force−meta−prefix (Off)
If set to **On**, **readline** modifies its behavior when binding key sequences containing \M- or Meta- (see **Key Bindings** above) 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).

history−preserve−point (Off)
If set to **On**, the history code attempts to place point at the same location on each history line retrieved with **previous-history** or **next-history**.

history−size (unset)
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. Setting *history−size* to a non-numeric value will set the maximum number of history entries to 500.

horizontal−scroll−mode (Off)
Setting this variable to **On** makes **readline** use a single line for display, scrolling the input horizontally on a single screen line when it becomes longer than the screen width rather than wrapping to a new line. This setting is automatically enabled for terminals of height 1.

input−meta (Off)
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 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 (“**C−[C−j**”)
The string of characters that should terminate an incremental search without subsequently executing the character as a command. If this variable has not been given a value, the characters *ESC* and **C−j** terminate an incremental search.

keymap (emacs)
Set the current **readline** keymap. The set of valid keymap names is *emacs, emacs−standard, emacs−meta, emacs−ctlx, vi, vi−command*, and *vi−insert*. *vi* is equivalent to *vi−command*; *emacs* is equivalent to *emacs−standard*. The default value is *emacs*; the value of **editing−mode** also affects the default keymap.

keyseq−timeout (500)
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** does not receive any input within the timeout, it uses the shorter but complete key sequence. 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.

mark−directories (On)
If set to **On**, completed directory names have a slash appended.

mark−modified−lines (Off)
If set to **On**, **readline** displays history lines that have been modified with a preceding asterisk (*****).

mark−symlinked−directories (Off)
If set to **On**, completed names which are symbolic links to directories have a slash appended, subject to the value of **mark−directories**.

match−hidden−files (On)
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.

menu−complete−display−prefix (Off)
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.

output−meta (Off)
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 (On)
If set to **On**, **readline** uses an internal pager resembling *more*(1) to display a screenful of possible completions at a time.

prefer−visible−bell
See **bell−style**.

print−completions−horizontally (Off)
If set to **On**, **readline** displays completions with matches sorted horizontally in alphabetical order, rather than down the screen.

revert−all−at−newline (Off)
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**().

search−ignore−case (Off)
If set to **On**, **readline** performs incremental and non-incremental history list searches in a case−insensitive fashion.

show−all−if−ambiguous (Off)
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.

show−all−if−unmodified (Off)
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.

show−mode−in−prompt (Off)
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*).

skip−completed−text (Off)
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.

vi−cmd−mode−string ((cmd))
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.

vi−ins−mode−string ((ins))
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.

visible−stats (Off)
If set to **On**, a character denoting a file's type as reported by *stat*(2) is appended to the filename when listing possible completions.

Conditional 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.

application

The *application* construct is used to include application-specific settings. Each program using the **readline** library sets the *application name*, and an initialization file 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*.

$else Commands in this branch of the **$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 would read */etc/inputrc*:

$include */etc/inputrc*

SEARCHING

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, *ESC* and **C−j** 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 newline 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.

EDITING COMMANDS

The following is a list of the names of the commands and the default key sequences to which they are bound. 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.

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.

forward−word (M−f)
Move forward to the end of the next word. Words are composed of alphanumeric characters (letters and digits).

backward−word (M−b)
Move back to the start of the current or previous word. Words are composed of alphanumeric characters (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. With a numeric argument, refresh the current line without clearing the screen.

redraw−current−line
Refresh the current line.

Commands for Manipulating the History

accept−line (Newline, Return)
Accept the line regardless of where the cursor is. If this line is non-empty, it may be added to the history list for future recall with **add_history()**. If the line is a modified history line, restore the history line to its original state.

previous−history (C−p)
Fetch the previous command from the history list, moving back in the list. This may also be bound to the up arrow key on some keyboards.

next−history (C−n)
Fetch the next command from the history list, moving forward in the list. 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.

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.

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 through the history starting at the current line 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 through the history 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. This may be bound to the Page Up 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. This may be bound to the Page Down 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.

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.

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−., M−_)
Insert the 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.

Commands for Changing Text

end−of−file **(usually C−d)**
: The character indicating end-of-file as set, for example, by *stty*(1). 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. When given a numeric argument, save the deleted text on the kill ring.

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.

quoted−insert (C−q, C−v)
: Add the next character typed to the line verbatim. This is how to insert characters 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 pasted characters are inserted as if each one was bound to **self−insert** instead of executing any editing commands.
: Bracketed paste sets the region to the inserted text and activates the region.

transpose−chars (C−t)
: Drag the character before point forward over the character at point, moving point forward as well. If point is at the end of the line, then this transposes the two characters before point. 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 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 point.

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

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

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.

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 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 point to the beginning of the line, saving the killed text on the kill-ring.

kill−whole−line
Kill all characters on the current line, no matter where point is.

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 those used by **forward−word**.

backward−kill−word (M−Rubout)
Kill the word behind point. Word boundaries are the same as those used by **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 (M−\)
Delete all spaces and tabs around point.

kill−region
Kill the text in the current region.

copy−region−as−kill
Copy the text in the region to the kill buffer, so it can be yanked immediately.

copy−backward−word
Copy the word before point to the kill buffer. The word boundaries are the same as **backward−word**.

copy−forward−word
Copy the word following point to the kill buffer. The word boundaries are the same as **forward−word**.

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. Only works following **yank** or **yank−pop**.

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.

Completing

complete (TAB)

Attempt to perform completion on the text before point. The actual completion performed is application-specific. **Bash**, for instance, attempts programmable completion first, otherwise treating the text as a variable (if the text begins with **$**), username (if the text begins with **~**), hostname (if the text begins with **@**), or command (including aliases, functions, and builtins) in turn. If none of these produces a match, it falls back to filename completion. **Gdb**, on the other hand, allows completion of program functions and variables, and only attempts filename completion under certain circumstances. The default **readline** completion 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.

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 store 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.

Miscellaneous

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 uppercase, run the command that is bound to the corresponding metafied lowercase character. The behavior is undefined if *x* is already lowercase.

prefix−meta (ESC)
Metafy the next character typed. **ESC f** is equivalent to **Meta−f**.

undo (C−_, 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 return the line to its initial state.

tilde−expand (M−˜)
Perform tilde expansion on the current word.

set−mark (C−@, M−<space>)
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 **readline 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. The default value of **comment−begin** causes this command to make the current line a shell comment. If a numeric argument causes the comment character to be removed, the line will be executed by the shell.

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.

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.

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.

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 switches **readline** to **emacs** editing mode.

vi−editing−mode (M−C−j)
When in **emacs** editing mode, this switches to **vi** editing mode.

DEFAULT KEY BINDINGS

The following is a list of the default emacs and vi bindings. Characters with the eighth bit set are written as M−<character>, and are referred to as *metafied* characters. The printable ASCII characters not mentioned in the list of emacs standard bindings are bound to the **self−insert** function, which just inserts the given character into the input line. In vi insertion mode, all characters not specifically mentioned are bound to **self−insert**. Characters assigned to signal generation by *stty*(1) or the terminal driver, such as C-Z or C-C, retain that function. Upper and lower case metafied characters are bound to the same function in the emacs mode meta keymap. The remaining characters are unbound, which causes **readline** to ring the bell (subject to the setting of the **bell−style** variable).

Emacs Mode

Emacs Standard bindings

```
"C-@"  set-mark
"C-A"  beginning-of-line
"C-B"  backward-char
"C-D"  delete-char
"C-E"  end-of-line
"C-F"  forward-char
"C-G"  abort
"C-H"  backward-delete-char
"C-I"  complete
"C-J"  accept-line
"C-K"  kill-line
"C-L"  clear-screen
"C-M"  accept-line
"C-N"  next-history
"C-P"  previous-history
"C-Q"  quoted-insert
"C-R"  reverse-search-history
"C-S"  forward-search-history
"C-T"  transpose-chars
"C-U"  unix-line-discard
"C-V"  quoted-insert
"C-W"  unix-word-rubout
"C-Y"  yank
"C-]"  character-search
"C-_"  undo
" " to "/"  self-insert
"0"  to "9"  self-insert
```

":" to "~" self-insert
"C-?" backward-delete-char

Emacs Meta bindings

"M-C-G" abort
"M-C-H" backward-kill-word
"M-C-I" tab-insert
"M-C-J" vi-editing-mode
"M-C-L" clear-display
"M-C-M" vi-editing-mode
"M-C-R" revert-line
"M-C-Y" yank-nth-arg
"M-C-[" complete
"M-C-]" character-search-backward
"M-space" set-mark
"M-#" insert-comment
"M-&" tilde-expand
"M-*" insert-completions
"M--" digit-argument
"M-." yank-last-arg
"M-0" digit-argument
"M-1" digit-argument
"M-2" digit-argument
"M-3" digit-argument
"M-4" digit-argument
"M-5" digit-argument
"M-6" digit-argument
"M-7" digit-argument
"M-8" digit-argument
"M-9" digit-argument
"M-<" beginning-of-history
"M-=" possible-completions
"M->" end-of-history
"M-?" possible-completions
"M-B" backward-word
"M-C" capitalize-word
"M-D" kill-word
"M-F" forward-word
"M-L" downcase-word
"M-N" non-incremental-forward-search-history
"M-P" non-incremental-reverse-search-history
"M-R" revert-line
"M-T" transpose-words
"M-U" upcase-word
"M-X" execute-named-command
"M-Y" yank-pop
"M-\" delete-horizontal-space
"M-~" tilde-expand
"M-C-?" backward-kill-word
"M-_" yank-last-arg

Emacs Control-X bindings

"C-XC-G" abort
"C-XC-R" re-read-init-file
"C-XC-U" undo

"C-XC-X" exchange-point-and-mark
"C-X(" start-kbd-macro
"C-X)" end-kbd-macro
"C-XE" call-last-kbd-macro
"C-XC-?" backward-kill-line

VI Mode bindings

VI Insert Mode functions

"C-D" vi-eof-maybe
"C-H" backward-delete-char
"C-I" complete
"C-J" accept-line
"C-M" accept-line
"C-N" menu-complete
"C-P" menu-complete-backward
"C-R" reverse-search-history
"C-S" forward-search-history
"C-T" transpose-chars
"C-U" unix-line-discard
"C-V" quoted-insert
"C-W" vi-unix-word-rubout
"C-Y" yank
"C-[" vi-movement-mode
"C-_" vi-undo
" " to "~" self-insert
"C-?" backward-delete-char

VI Command Mode functions

"C-D" vi-eof-maybe
"C-E" emacs-editing-mode
"C-G" abort
"C-H" backward-char
"C-J" accept-line
"C-K" kill-line
"C-L" clear-screen
"C-M" accept-line
"C-N" next-history
"C-P" previous-history
"C-Q" quoted-insert
"C-R" reverse-search-history
"C-S" forward-search-history
"C-T" transpose-chars
"C-U" unix-line-discard
"C-V" quoted-insert
"C-W" vi-unix-word-rubout
"C-Y" yank
"C-_" vi-undo
" " forward-char
"#" insert-comment
"$" end-of-line
"%" vi-match
"&" vi-tilde-expand
"*" vi-complete
"+" next-history
"," vi-char-search

"-" previous-history
"." vi-redo
"/" vi-search
"0" beginning-of-line
"1" to "9" vi-arg-digit
";" vi-char-search
"=" vi-complete
"?" vi-search
"A" vi-append-eol
"B" vi-prev-word
"C" vi-change-to
"D" vi-delete-to
"E" vi-end-word
"F" vi-char-search
"G" vi-fetch-history
"I" vi-insert-beg
"N" vi-search-again
"P" vi-put
"R" vi-replace
"S" vi-subst
"T" vi-char-search
"U" revert-line
"W" vi-next-word
"X" vi-rubout
"Y" vi-yank-to
"\" vi-complete
"^" vi-first-print
"_" vi-yank-arg
"‘" vi-goto-mark
"a" vi-append-mode
"b" vi-prev-word
"c" vi-change-to
"d" vi-delete-to
"e" vi-end-word
"f" vi-char-search
"h" backward-char
"i" vi-insertion-mode
"j" next-history
"k" previous-history
"l" forward-char
"m" vi-set-mark
"n" vi-search-again
"p" vi-put
"r" vi-change-char
"s" vi-subst
"t" vi-char-search
"u" vi-undo
"w" vi-next-word
"x" vi-delete
"y" vi-yank-to
"|" vi-column
"~" vi-change-case

SEE ALSO

The Gnu Readline Library, Brian Fox and Chet Ramey
The Gnu History Library, Brian Fox and Chet Ramey
bash(1)

FILES

~/.inputrc
Individual **readline** initialization file

AUTHORS

Brian Fox, Free Software Foundation
bfox@gnu.org

Chet Ramey, Case Western Reserve University
chet.ramey@case.edu

BUG REPORTS

If you find a bug in **readline**, you should report it. But first, you should make sure that it really is a bug, and that it appears in the latest version of the **readline** library that you have.

Once you have determined that a bug actually exists, mail a bug report to *bug−readline@gnu.org*. If you have a fix, you are welcome to mail that as well! Suggestions and “philosophical” bug reports may be mailed to *bug-readline@gnu.org* or posted to the Usenet newsgroup **gnu.bash.bug**.

Comments and bug reports concerning this manual page should be directed to *chet.ramey@case.edu*.

BUGS

It’s too big and too slow.