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 will read a line from the terminal and return it, using **prompt** as a prompt. If **prompt** is **NULL** or the empty string, no prompt is issued. 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.

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, **NULL** is returned. If an **EOF** is read with a non−empty line, it is treated as a newline.

NOTATION

An emacs-style notation is used to denote 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. (On keyboards without a *meta* key, M−*x* means ESC *x*, i.e., press the Escape key then the *x* key. This makes ESC the *meta prefix*. The combination M−C−*x* means ESC−Control−*x*, or press the Escape key then hold the Control key while pressing the *x* key.)

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**) causes that command to act in a backward direction. Commands whose behavior with arguments deviates from this are noted.

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 cause the text to be accumulated 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, the ultimate default is */etc/inputrc*. When a program which uses the readline library starts up, the init file is read, and the key bindings and variables are set. 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. Other lines denote key bindings and variable settings. Each program using this library may add its 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*.

The following symbolic character names are recognized while processing key bindings: *DEL*, *ESC*, *ESCAPE*, *LFD*, *NEWLINE*, *RET*, *RETURN*, *RUBOUT*, *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*).

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 name may be specified in one of two ways: as a symbolic key name, possibly with *Meta−* or *Control−* prefixes, or as a key sequence. The name and key sequence 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 the symbolic character names are not 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−	control prefix
\M−	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 should be used to indicate a macro definition. Unquoted text is assumed to be a function name. In the macro body, the backslash escapes described above are expanded. Backslash will quote any other character in the macro text, including " and '.

Bash allows the current readline key bindings to be displayed or modified with the **bind** builtin command. The editing mode may be switched during interactive use by using the **−o** option to the **set** builtin command. Other programs using this library provide similar mechanisms. The *inputrc* file may be edited and re-read 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 a variable value is read, 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:

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 treated specially by the kernel's terminal driver to their readline equivalents.

comment−begin (“#”)
The string that is inserted in **vi** mode when the **insert−comment** command is executed. This command is bound to **M−#** in emacs mode and to **#** in vi command mode.

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

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, the user is asked whether or not he wishes to view them; otherwise they are simply listed on the terminal. A negative value causes readline to never ask.

convert−meta (On)
If set to **On**, readline will convert characters with the eighth bit set to an ASCII key sequence by stripping the eighth bit and prefixing it with an escape character (in effect, using escape as the *meta prefix*).

disable−completion (Off)
If set to **On**, readline will inhibit word completion. Completion characters will be inserted into the line as if they had been mapped to **self-insert**.

editing−mode (emacs)
Controls whether readline begins with a set of key bindings similar to emacs or vi. **editing−mode** can be set to either **emacs** or **vi**.

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

expand−tilde (Off)
If set to **on**, tilde expansion is performed when readline attempts word completion.

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**.

horizontal−scroll−mode (Off)
When set 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.

input−meta (Off)

If set to **On**, readline will enable eight-bit input (that is, it will not clear the eighth bit in the characters it reads), regardless of what the terminal claims it can support. The name **meta−flag** is a synonym for this variable.

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* will terminate an incremental search.

keymap (emacs)

Set the current readline keymap. The set of legal keymap names is *emacs, emacs-standard, emacs-meta, emacs-ctlx, vi, vi-move, 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.

mark−directories (On)

If set to **On**, completed directory names have a slash appended.

mark−modified−lines (Off)

If set to **On**, history lines that have been modified are displayed 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**, causes readline to match files whose names begin with a '.' (hidden files) when performing filename completion, unless the leading '.' is supplied by the user in the filename to be completed.

output−meta (Off)

If set to **On**, readline will display characters with the eighth bit set directly rather than as a meta-prefixed escape sequence.

page−completions (On)

If set to **On**, readline uses an internal *more*-like pager to display a screenful of possible completions at a time.

print−completions−horizontally (Off)

If set to **On**, readline will display completions with matches sorted horizontally in alphabetical order, rather than down the screen.

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.

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 used.

$if The **$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 extends to the end of the line; 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 the full name of the terminal and the portion of the terminal name before the first **−**. This allows *sun* to match both *sun* and *sun−cmd*, for instance.

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
```

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

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

$include
This directive takes a single filename as an argument and reads commands and 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. To search backward in the history for a particular string, type **C−r**. 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 *Escape* and **C−J** characters will terminate an incremental search. **C−G** will abort an incremental search and restore the original line. When the search is terminated, the history entry containing the search string becomes the current line.

To find other matching entries in the history list, type **C−s** or **C−r** as appropriate. This will search backward or forward in the history for the next line matching the search string typed so far. Any other key sequence bound to a readline command will terminate the search and execute that command. For instance, a newline will terminate the search and accept 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.

Non-incremental searches read the entire search string before starting to search for matching history lines. 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*.

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. 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).

clear−screen (C−l)
Clear the screen leaving the current line at the top of the screen. With an 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, the history line is restored to its original state.

previous−history (C−p)
Fetch the previous command from the history list, moving back in the list.

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

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.

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.

non−incremental−forward−search−history (M−n)
Search forward through the history using a non-incremental search for a string supplied by the user.

history−search−forward
Search forward through the history for the string of characters between the start of the current line and the current cursor position (the *point*). This is a non-incremental search.

history−search−backward
Search backward through the history for the string of characters between the start of the current line and the point. 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, the argument is extracted 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 an argument, behave exactly like **yank−nth−arg**. Successive calls to **yank−last−arg** move back through the history list, inserting the last argument of each line in turn. The history expansion facilities are used to extract the last argument, as if the "!$" history expansion had been specified.

Commands for Changing Text

delete−char (C−d)

Delete the character at point. If point is at the beginning of the line, there are no characters in the line, and the last character typed was not bound to **delete−char**, then return **EOF**.

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 that you type 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.

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 over 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.

Killing and Yanking

kill−line (C−k)

Kill the text from point to the end of the line.

backward−kill−line (C−x Rubout)
Kill backward to the beginning of the line.

unix−line−discard (C−u)
Kill backward from point to the beginning of the line. The killed text is saved 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 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. The killed text is saved on the kill-ring.

unix−filename−rubout
Kill the word behind point, using white space and the slash character as the word boundaries. The killed text is saved on the kill-ring.

delete−horizontal−space (M−\)
Delete all spaces and tabs around point.

kill−region
Kill the text between the point and *mark* (saved cursor position). This text is referred to as the *region*.

copy−region−as−kill
Copy the text in the region to the kill buffer.

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 or 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 completion 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 and functions) in turn. If none of these produces a match, filename completion is attempted. **Gdb**, on the other hand, allows completion of program functions and variables, and only attempts filename completion under certain circumstances.

possible−completions (M−?)
List the possible completions of the text before point.

insert−completions (M−*)
Insert all completions of the text before point that would have been generated by **possible−completions**.

menu−complete
Similar to **complete**, but replaces the word to be completed with a single match from the list of possible completions. Repeated execution of **menu−complete** steps through the list of possible completions, inserting each match in turn. At the end of the list of completions, the bell is rung (subject to the setting of **bell−style**) and the original text is restored. An argument of *n* moves *n* positions forward in the list of matches; a negative argument may be used to move backward through the list. This command is intended to be bound to **TAB**, but 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**). If at the end of the line, behaves identically to **possible-completions**.

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.

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−uppercase−version (M−a, M−b, M−*x*, ...)
If the metafied character *x* is lowercase, run the command that is bound to the corresponding uppercase character.

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, the mark is set to that position.

exchange−point−and−mark (C−x C−x)
Swap the point with the mark. The current cursor position is set to the saved position, and the old cursor position is saved as the mark.

character−search (C−])
A character is read and point is moved to the next occurrence of that character. A negative count searches for previous occurrences.

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

insert−comment (M−#)
Without a numeric argument, the value of the readline **comment−begin** variable is inserted 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**, the value is inserted, otherwise the characters in **comment-begin** are deleted 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** makes 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. If a numeric argument is supplied, the output is formatted in such a way that it can be made part of an *inputrc* file.

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.

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-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-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-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-[" vi-movement-mode
"C-_" 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" 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" backward-delete-char
"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" prev-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@ins.CWRU.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@ins.CWRU.Edu*.

BUGS

It’s too big and too slow.