NAME

readline − get a line from a user with editing

SYNOPSIS

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

typedef int Function ();

char *readline (prompt)
char *prompt;

int rl_add_defun (name, function, key)
char *name;
Function *function;
int key;

int rl_bind_key (key, function)
int key;
Function *function;

int rl_unbind_key (key)
int key;

int rl_bind_key_in_map (key, function, keymap)
int key;
Function *function;
Keymap keymap;

int rl_unbind_key_in_map (key, keymap)
int key;
Keymap keymap;

int rl_macro_bind (keyseq, macro, keymap)
char *keyseq, *macro;
Keymap keymap;

int rl_variable_bind (variable, value)
char *variable, *value;

int rl_parse_and_bind (line)
char *line;

int rl_translate_keyseq (keyseq, array, len)
char *keyseq, *array;
int *len;

Function *rl_named_function (command)
char *command;

Function *rl_function_of_keyseq (keyseq, keymap, type)
char *keyseq;
Keymap keymap;
int *type;

char **rl_invoking_keyseqs (function)
Function *function;

char **rl_invoking_keyseqs_in_map (function, keymap)
Function *function;
Keymap keymap;

void rl_function_dumper (readable)
int readable;

char **rl_funmap_names ()

COPYRIGHT

DESCRIPTION

readline will read a line from the terminal and return it, using **prompt** as a prompt. If **prompt** is null, no prompt is issued. The line returned is allocated with *malloc*(3), so 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.

In the following descriptions, **keymap** can be one of *emacs_keymap, emacs_meta_keymap, emacs_ctlx_keymap, vi_insertion_keymap, or vi_movement_keymap*.

rl_add_defun makes **name** appear as a bindable readline command, and makes **function** be the function called when that command is invoked. If **key** is not −1, it is bound to **function** in the current keymap.

rl_bind_key causes **key** to invoke **function**. The binding is made in the current keymap.

rl_unbind_key removes the binding for **key** in the current keymap.

rl_bind_key_in_map makes the **key** entry in **keymap** invoke **function**.

rl_unbind_key_in_map removes the binding for **key** in keymap **keymap**.

rl_macro_bind makes **keyseq** insert the string **macro**. The binding is performed in **keymap**.

rl_variable_bind sets the value of the readline variable **variable** to **value**.

rl_parse_and_bind takes as an argument a line of the same form as the readline startup file (see **INITIALIZATION FILE** below) and executes the commands therein.

rl_translate_keyseq converts **keyseq** into a new string, storing the result in **array**. This translates control and meta prefixes and the readline character escape sequences (see **Key Bindings** below). The length of the translated sequence is returned in ***len**.

rl_named_function returns the function that is executed when the readline command **command** is invoked.

rl_function_of_keyseq returns the function that is executed when **keyseq** is read and **keymap** is the current keymap. **type** is set to indicate whether the return value corresponds to a function, macro, or auxiliary keymap.

rl_invoking_keyseqs returns all of the key sequences in the current keymap that invoke **function**.

rl_invoking_keyseqs_in_map returns all of the key sequences in **keymap** that invoke **function**.

rl_function_dumper prints all of the readline functions and their bindings to the readline output stream. If **readable** is non−zero, the output is formattted so that it can be read back in to restore the bindings.

rl_funmap_names returns an array of all known readline bindable function names. The array is sorted.

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.

Unless otherwise stated, the other functions return 0 on success and non−zero on failure.

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 name of this file is taken from the value of the **INPUTRC** variable. If that variable is unset, the default is *~/.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: *RUBOUT*, *DEL*, *ESC*, *LFD*, *NEWLINE*, *RET*, *RETURN*, *SPC*, *SPACE*, 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. 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.

```
"\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 escape sequences is

\C- control prefix

\M- meta prefix

\e an escape character

**** backslash

\" literal "

\' literal '

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. Backslash will quote any 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**. The variables and their default values are:

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.

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**.

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

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.

comment−begin ("#")
The string that is inserted in **vi** mode when the **vi−comment** command is executed.

meta−flag (Off)
If set to **On**, readline will enable eight-bit input (that is, it will not strip the high bit from the characters it reads), regardless of what the terminal claims it can support.

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 prepending an escape character (in effect, using escape as the *meta prefix*).

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.

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.

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.

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.

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

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 three 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

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

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

EDITING COMMANDS

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

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 this, or the 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. By default, this is unbound.

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 the line is a modified history line, then 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.

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 point. This is a non-incremental search. By default, this command is unbound.

history−search−backward
Search backward through the history for the string of characters between the start of the current line and the current point. This is a non-incremental search. By default, this command is unbound.

yank−nth−arg (M−C−y)
Insert the first argument to the previous command (usually the second word on the previous line) at point (the current cursor position). 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.

yank−last−arg (M−., M−_)
Insert the last argument to the previous command (the last word on the previous line). With an argument, behave exactly like @code{yank-nth-arg}.

Commands for Changing Text

delete−char (C−d)
Delete the character under the cursor. If point is at the beginning of the line, 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. When given a numeric argument, save the deleted text on the kill−ring.

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. Point moves forward as well. If point is at the end of the line, then transpose the two characters before point. Negative arguments 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 the current (or following) word. With a negative argument, do the previous word, but do not move point.

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

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

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−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.

kill−whole−line
Kill all characters on the current line, no matter where the cursor is. By default, this is 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. Word boundaries are the same as those used by **forward−word**.

backward−kill−word (M−Rubout)
Kill the word behind the cursor. Word boundaries are the same as those used by **backward−word**.

unix−word−rubout (C−w)
Kill the word behind the cursor, using white space as a word boundary. The word boundaries are different from **backward−kill−word**.

delete−horizontal−space
Delete all spaces and tabs around point. By default, this is unbound.

yank (C−y)
Yank the top of the kill ring into the buffer at the cursor.

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
Each time this is executed, the argument count is multiplied by four. The argument count is initially one, so executing this function the first time makes the argument count four. By default, this

is not bound to a key.

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
Insert all completions of the text before point that would have been generated by **possible−completions**. By default, this is not bound to a key.

Keyboard Macros

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

end−kbd−macro (C-x))
Stop saving the characters typed into the current keyboard macro and save the definition.

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

Miscellaneous

re-read-init-file (C−x C−r)
Read in the contents of your init 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, ...)
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 typing the **undo** command enough times to return the line to its initial state.

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

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.

emacs−editing−mode (C−e)
When in **vi** editing 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 8th 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-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-_"                    -> undo
" " to "/"               -> self-insert
"0" to "9"               -> self-insert
":" to "~"               -> self-insert
"C-?"                    -> backward-delete-char
```

Emacs Meta bindings

```
"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-&"                    -> tilde-expand
"M--"                    -> digit-argument
"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->"	-> 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-O"	-> arrow-key-prefix
"M-P"	-> non-incremental-reverse-search-history
"M-R"	-> revert-line
"M-T"	-> transpose-words
"M-U"	-> upcase-word
"M-Y"	-> yank-pop
"M-C-Y"	-> yank-nth-arg
"M-C-?"	-> backward-delete-word

Emacs Control-X bindings

"C-XC-G"	-> abort
"C-XC-R"	-> re-read-init-file
"C-XC-U"	-> undo
"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-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-movement-mode
" " 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-["	-> abort
" "	-> forward-char
"#"	-> vi-comment
"$"	-> end-of-line
"%"	-> vi-match
"&"	-> vi-tilde-expand
"*"	-> vi-complete
"+"	-> down-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
"@"	-> is undefined
"A"	-> vi-append-eol
"B"	-> vi-prev-word
"C"	-> vi-change-to
"D"	-> vi-delete-to
"E"	-> vi-end-word
"F"	-> vi-char-search
"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

"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
"n"	-> vi-search-again
"r"	-> vi-change-char
"s"	-> vi-subst
"t"	-> vi-char-search
"u"	-> 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 (primary author)
bfox@ai.MIT.Edu

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 *bash−maintainers@prep.ai.MIT.Edu*. If you have a fix, you are welcome to mail that as well! Suggestions and 'philosophical' bug reports may be mailed to *bug-bash@prep.ai.MIT.Edu* 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.