NAME

Term::ReadLine::Gnu − Perl extension for the GNU Readline/History Library

SYNOPSIS

```
use Term::ReadLine;   # Do not "use Term::ReadLine::Gnu;"
$term = new Term::ReadLine 'ProgramName';
while ( defined ($_ = $term->readline('prompt>')) ) {
  ...
}
```

DESCRIPTION

Overview

This is an implementation of Term::ReadLine <http://search.cpan.org/dist/Term-ReadLine/> using the GNU Readline/History Library <https://tiswww.cwru.edu/php/chet/readline/rltop.html>.

For basic functions object oriented interface is provided. These are described in the section ‘‘Standard Methods’’ and "`Term::ReadLine::Gnu` Functions".

This package also has the interface with the almost all functions and variables which are documented in the GNU Readline/History Library Manual. They are documented in the section "`Term::ReadLine::Gnu` Functions" and "`Term::ReadLine::Gnu` Variables" briefly. For further details of the GNU Readline/History Library, see GNU Readline Library Manual <https://tiswww.cwru.edu/php/chet/readline/readline.html> and GNU History Library Manual <https://tiswww.cwru.edu/php/chet/readline/history.html>.

There are some `Term::ReadLine::Gnu` original features. They are described in the section "`Term::ReadLine::Gnu` Specific Features"

The sample programs under *eg/* directory and test programs under *t/* directory in the `Term::ReadLine::Gnu` distribution <http://search.cpan.org/dist/Term-ReadLine-Gnu/> include many examples of this module.

Standard Methods

These are standard methods defined by Term::ReadLine <http://search.cpan.org/dist/Term-ReadLine/>.

`ReadLine`
: returns the actual package that executes the commands. If this package is being used, `Term::ReadLine::Gnu` is returned.

`new(NAME,[IN,OUT])`
: returns the handle for subsequent calls to following functions. Argument is the name of the application. Optionally can be followed by two arguments for `IN` and `OUT` file handles. These arguments should be globs.

`readline(PROMPT[,PREPUT])`
: gets an input line, with actual `GNU Readline` support. Trailing newline is removed. Returns `undef` on `EOF`. `PREPUT` is an optional argument meaning the initial value of input.

  The optional argument `PREPUT` is granted only if the value `preput` is in `Features`.

  `PROMPT` may include some escape sequences. Use `RL_PROMPT_START_IGNORE` to begin a sequence of non-printing characters, and `RL_PROMPT_END_IGNORE` to end the sequence.

`AddHistory(LINE1, LINE2, ...)`
: adds the lines to the history of input, from where it can be used if the actual `readline` is present.

`IN`, `OUT`
: return the file handles for input and output or `undef` if `readline` input and output cannot be used for Perl.

`MinLine([MAX])`
: If argument `MAX` is specified, it is an advice on minimal size of line to be included into history. `undef` means do not include anything into history. Returns the old value.

`findConsole`
: returns an array with two strings that give most appropriate names for files for input and output using conventions `"<$in"`, `">$out"`.

`Attribs`
: returns a reference to a hash which describes internal configuration (variables) of the package. Names of keys in this hash conform to standard conventions with the leading `rl_` stripped.

  See section "`Term::ReadLine::Gnu` Variables" for supported variables.

`Features`
: Returns a reference to a hash with keys being features present in current implementation. Several optional features are used in the minimal interface: `appname` should be present if the first argument to `new` is recognized, and `minline` should be present if `MinLine` method is not dummy. `autohistory` should be present if lines are put into history automatically (maybe subject to `MinLine`), and `addHistory` if `AddHistory` method is not dummy. `preput` means the second argument to `readline` method is processed. `getHistory` and `setHistory` denote that the corresponding methods are present. `tkRunning` denotes that a Tk application may run while ReadLine is getting input.

`tkRunning`
: makes Tk event loop run when waiting for user input (i.e., during `readline` method).

`event_loop`
: See the description of `event_loop` on Term::ReadLine <http://search.cpan.org/dist/Term-ReadLine/>.

`ornaments`
: makes the command line stand out by using termcap data. The argument to `ornaments` should be 0, 1, or a string of a form `"aa,bb,cc,dd"`. Four components of this string should be names of *terminal capacities*, first two will be issued to make the prompt standout, last two to make the input line standout.

`newTTY`
: takes two arguments which are input filehandle and output filehandle. Switches to use these filehandles.

`enableUTF8`
: Enables UTF-8 support.

  If STDIN is in UTF-8 by the `-C` command-line switch or `PERL_UNICODE` environment variable, or `IN` file handle has `utf8` IO layer, then UTF-8 support is also enabled. In other cases you need this `enableUTF8` method.

  This is an original method of `Term::ReadLine:Gnu`.

`Term::ReadLine::Gnu` Functions

All these GNU Readline/History Library functions supported are callable via method interface and have names which conform to standard conventions with the leading `rl_` stripped. For example `rl_foo()` function is called as `$term->foo()`.

The titles of the following sections are same as the titles of the corresponding sections in the "Programming with GNU Readline" section in the GNU Readline Library Manual <https://tiswww.cwru.edu/php/chet/readline/readline.html>. Refer them for further details.

Although it is preferred to use method interface, most methods have lower level functions in `Term::ReadLine::Gnu::XS` package. To use them a full qualified name is required.

Basic Behavior

The function `readline()` prints a prompt and then reads and returns a single line of text from the user.

```
$_ = $term->readline('Enter a line: ');
```

You can change key-bindings using `bind_key(KEY, FUNCTION [,MAP])` function. The first argument, `KEY`, is the character that you want bind. The second argument, `FUNCTION`, is the function to call when `KEY` is pressed. The `FUNCTION` can be a reference to a Perl function (see ‘‘Custom Functions’’) or a ‘‘named function’’ named by `add_defun()` function or commands described in the ‘‘Bindable Readline Commands’’ section in the GNU Readline Library Manual <https://tiswww.cwru.edu/php/chet/readline/readline.html>.

```
$term->bind_key(ord "\ci, 'tab-insert');
```

The above example binds Control-I to the ’tab−insert’ command.

Custom Functions

You can write new functions using Perl. The calling sequence for a command foo looks like

```
sub foo ($count, $key) { ... }
```

where `$count` is the numeric argument (or 1 if defaulted) and `$key` is the key that invoked this function.

Here is an example;

```
sub reverse_line {              # reverse a whole line
    my($count, $key) = @_;      # ignored in this sample function

    $t->modifying(0, $a->{end}); # save undo information
    $a->{line_buffer} = reverse $a->{line_buffer};
}
```

See the ‘‘Writing a New Function’’ section in the GNU Readline Library Manual <https://tiswww.cwru.edu/php/chet/readline/readline.html> for further details.

Readline Convenience Functions

Naming a Function

`add_defun(NAME, FUNCTION [,KEY=-1])`

Add name to a Perl function `FUNCTION`. If optional argument `KEY` is specified, bind it to the `FUNCTION`. Returns reference to`FunctionPtr`.

Example:

```
# name `reverse-line' to a function reverse_line(),
# and bind it to "\C-t"
$term->add_defun('reverse-line', \&reverse_line, ord "\ct");
```

Selecting a Keymap

```
make_bare_keymap
           Keymap  rl_make_bare_keymap()

copy_keymap(MAP)
           Keymap  rl_copy_keymap(Keymap|str map)

make_keymap
           Keymap  rl_make_keymap()

discard_keymap(MAP)
           Keymap  rl_discard_keymap(Keymap|str map)

free_keymap(MAP)
           void    rl_free_keymap(Keymap|str map)

empty_keymap(MAP)
           int     rl_empty_keymap(Keymap|str map)                 # GRL 8.0

get_keymap
           Keymap  rl_get_keymap()
```

```
set_keymap(MAP)
            Keymap  rl_set_keymap(Keymap|str map)

get_keymap_by_name(NAME)
            Keymap  rl_get_keymap_by_name(str name)

get_keymap_name(MAP)
            str     rl_get_keymap_name(Keymap map)

set_keymap_name(NAME, MAP)
            int     rl_set_keymap_name(str name, Keymap|str map)    # GRL 8.0
```

Binding Keys

```
bind_key(KEY, FUNCTION [,MAP])
            int     rl_bind_key(int key, FunctionPtr|str function,
                                Keymap|str map = rl_get_keymap())
```

Bind KEY to the FUNCTION. FUNCTION is the name added by the add_defun method. If optional argument MAP is specified, binds in MAP. Returns non-zero in case of error.

```
bind_key_if_unbound(KEY, FUNCTION [,MAP])
            int     rl_bind_key_if_unbound(int key, FunctionPtr|str function,
                                           Keymap|str map = rl_get_keymap()) # GRL

unbind_key(KEY [,MAP])
            int     rl_unbind_key(int key, Keymap|str map = rl_get_keymap())
```

Bind KEY to the null function. Returns non-zero in case of error.

```
unbind_function(FUNCTION [,MAP])
            int     rl_unbind_function(FunctionPtr|str function,
                                       Keymap|str map = rl_get_keymap())

unbind_command(COMMAND [,MAP])
            int     rl_unbind_command(str command,
                                      Keymap|str map = rl_get_keymap())

bind_keyseq(KEYSEQ, FUNCTION [,MAP])
            int     rl_bind_keyseq(str keyseq, FunctionPtr|str function,
                                   Keymap|str map = rl_get_keymap()) # GRL 5.0

set_key(KEYSEQ, FUNCTION [,MAP])
            int     rl_set_key(str keyseq, FunctionPtr|str function,
                               Keymap|str map = rl_get_keymap())    # GRL 4.2

bind_keyseq_if_unbound(KEYSEQ, FUNCTION [,MAP])
            int     rl_bind_keyseq_if_unbound(str keyseq, FunctionPtr|str function
                                              Keymap|str map = rl_get_keymap()) #

generic_bind(TYPE, KEYSEQ, DATA, [,MAP])
            int     rl_generic_bind(int type, str keyseq,
                                    FunctionPtr|Keymap|str data,
                                    Keymap|str map = rl_get_keymap())

parse_and_bind(LINE)
            void    rl_parse_and_bind(str line)
```

Parse LINE as if it had been read from the *~/.inputrc* file and perform any key bindings and variable assignments found. For further detail see GNU Readline Library Manual <https://tiswww.cwru.edu/php/chet/readline/readline.html>.

```
read_init_file([FILENAME])
            int     rl_read_init_file(str filename = '~/.inputrc')
```

Associating Function Names and Bindings

```
named_function(NAME)
            FunctionPtr rl_named_function(str name)

get_function_name(FUNCTION)
            str     rl_get_function_name(FunctionPtr function)      # TRG original

function_of_keyseq(KEYSEQ [,MAP])
            (FunctionPtr|Keymap|str data, int type)
                    rl_function_of_keyseq(str keyseq,
                                          Keymap|str map = rl_get_keymap())

invoking_keyseqs(FUNCTION [,MAP])
            (@str)  rl_invoking_keyseqs(FunctionPtr|str function,
                                        Keymap|str map = rl_get_keymap())

function_dumper([READABLE])
            void    rl_function_dumper(int readable = 0)

list_funmap_names
            void    rl_list_funmap_names()

funmap_names
            (@str)  rl_funmap_names()

add_funmap_entry(NAME, FUNCTION)
            int     rl_add_funmap_entry(char *name, FunctionPtr|str function)
```

Allowing Undoing

```
begin_undo_group
            int     rl_begin_undo_group()

end_undo_group
            int     rl_end_undo_group()

add_undo(WHAT, START, END, TEXT)
            int     rl_add_undo(int what, int start, int end, str text)

free_undo_list
            void    rl_free_undo_list()

do_undo
            int     rl_do_undo()

modifying([START [,END]])
            int     rl_modifying(int start = 0, int end = rl_end)
```

Redisplay

```
redisplay
            void    rl_redisplay()

forced_update_display
            int     rl_forced_update_display()

on_new_line
            int     rl_on_new_line()

on_new_line_with_prompt
            int     rl_on_new_line_with_prompt()                    # GRL 4.1

clear_visible_line()
            int     rl_clear_visible_line()                         # GRL 7.0

reset_line_state
            int     rl_reset_line_state()
```

```
crlf
            int     rl_crlf()

show_char(C)
            int     rl_show_char(int c)

message(FMT[, ...])
            int     rl_message(str fmt, ...)

clear_message
            int     rl_clear_message()

save_prompt
            void    rl_save_prompt()

restore_prompt
            void    rl_restore_prompt()

expand_prompt(PROMPT)
            int     rl_expand_prompt(str prompt)

set_prompt(PROMPT)
            int     rl_set_prompt(const str prompt)                 # GRL 4.2
```

Modifying Text

```
insert_text(TEXT)
            int     rl_insert_text(str text)

delete_text([START [,END]])
            int     rl_delete_text(int start = 0, int end = rl_end)

copy_text([START [,END]])
            str     rl_copy_text(int start = 0, int end = rl_end)

kill_text([START [,END]])
            int     rl_kill_text(int start = 0, int end = rl_end)

push_macro_input(MACRO)
            int     rl_push_macro_input(str macro)
```

Character Input

```
read_key
            int     rl_read_key()

getc(STREAM)
            int     rl_getc(FILE *STREAM)

stuff_char(C)
            int     rl_stuff_char(int c)

execute_next(C)
            int     rl_execute_next(int c)

clear_pending_input()
            int     rl_clear_pending_input()                        # GRL 4.2

set_keyboard_input_timeout(uSEC)
            int     rl_set_keyboard_input_timeout(int usec)         # GRL 4.2
```

Terminal Management

```
prep_terminal(META_FLAG)
            void    rl_prep_terminal(int META_FLAG)
```

```
deprep_terminal()
            void    rl_deprep_terminal()

tty_set_default_bindings([MAP])
            void    rl_tty_set_default_bindings([Keymap|str map = rl_get_keymap()]

tty_unset_default_bindings([MAP])
            void    rl_tty_unset_default_bindings([Keymap|str map = rl_get_keymap(

tty_set_echoing(VALUE)
            int     rl_tty_set_echoing(int value)                   # GRL 7.0

reset_terminal([TERMINAL_NAME])
            int     rl_reset_terminal(str terminal_name = getenv($TERM))
```

Utility Functions

```
save_state(READLINE_STATE)
            READLINE_STATE  rl_save_state()                         # GRL 6.0

restore_state(READLINE_STATE)
            int     rl_restore_state(READLINE_STATE)                # GRL 6.0

free(MEM)
            Not implemented since not required for Perl.
            int     rl_free(void *mem)                              # GRL 6.0

replace_line(TEXT [,CLEAR_UNDO])
            int     rl_replace_line(str text, int clear_undo = 0)   # GRL 4.3

extend_line_buffer(LEN)
            Not implemented since not required for Perl.
            int     rl_extend_line_buffer(int len)

initialize
            int     rl_initialize()

ding
            int     rl_ding()

alphabetic(C)
            int     rl_alphabetic(int C)                            # GRL 4.2

display_match_list(MATCHES [,LEN [,MAX]])
            void    rl_display_match_list(\@matches, len = $#maches, max) # GRL 4.
```

Since the first element of an array `@matches` as treated as a possible completion, it is not displayed. See the descriptions of `completion_matches()`. When`MAX` is omitted, the max length of an item in `@matches` is used.

Miscellaneous Functions

```
macro_bind(KEYSEQ, MACRO [,MAP])
            int     rl_macro_bind(const str keyseq, const str macro, Keymap map)

macro_dumper(READABLE)
            int     rl_macro_dumper(int readline)

variable_bind(VARIABLE, VALUE)
            int     rl_variable_bind(const str variable, const str value)

variable_value(VARIABLE)
            str     rl_variable_value(const str variable)           # GRL 5.1

variable_dumper(READABLE)
            int     rl_variable_dumper(int readline)
```

```
set_paren_blink_timeout(uSEC)
            int     rl_set_paren_blink_timeout(usec)                # GRL 4.2
get_termcap(cap)
            str     rl_get_termcap(cap)
clear_history
            void    rl_clear_history()                              # GRL 6.3
activate_mark
            void    rl_activate_mark()                              # GRL 8.1
deactivate_mark
            void    rl_deactivate_mark()                            # GRL 8.1
keep_mark_active
            void    rl_keep_mark_active()                           # GRL 8.1
mark_active_p
            int     rl_mark_active_p()                              # GRL 8.1
```

Alternate Interface

```
callback_handler_install(PROMPT, LHANDLER)
            void    rl_callback_handler_install(str prompt, pfunc lhandler)
callback_read_char
            void    rl_callback_read_char()
callback_sigcleanup                                 # GRL 7.0
            void    rl_callback_sigcleanup()
callback_handler_remove
            void    rl_callback_handler_remove()
```

Readline Signal Handling

```
pending_signal()
            int     rl_pending_signal()                             # GRL 7.0
cleanup_after_signal
            void    rl_cleanup_after_signal()                       # GRL 4.0
free_line_state
            void    rl_free_line_state()                            # GRL 4.0
reset_after_signal
            void    rl_reset_after_signal()                         # GRL 4.0
check_signals
            void    rl_check_signals()                              # GRL 8.0
echo_signal_char
            void    rl_echo_signal_char(int sig)                    # GRL 6.0
resize_terminal
            void    rl_resize_terminal()                            # GRL 4.0
set_screen_size(ROWS, COLS)
            void    rl_set_screen_size(int ROWS, int COLS)          # GRL 4.2
get_screen_size()
            (int rows, int cols)    rl_get_screen_size()            # GRL 4.2
reset_screen_size()
            void    rl_reset_screen_size()                          # GRL 5.1
```

```
set_signals
            int     rl_set_signals()                                  # GRL 4.0
clear_signals
            int     rl_clear_signals()                                # GRL 4.0
```

Completion Functions

```
complete_internal([WHAT_TO_DO])
            int     rl_complete_internal(int what_to_do = TAB)
completion_mode(FUNCTION)
            int     rl_completion_mode(FunctionPtr|str function)    # GRL 4.3
completion_matches(TEXT [,FUNC])
            (@str)  rl_completion_matches(str text,
                                          pfunc func = filename_completion_functio
filename_completion_function(TEXT, STATE)
            str     rl_filename_completion_function(str text, int state)
username_completion_function(TEXT, STATE)
            str     rl_username_completion_function(str text, int state)
list_completion_function(TEXT, STATE)
            str     list_completion_function(str text, int state)   # TRG original
```

History Functions

Initializing History and State Management

```
using_history
            void    using_history()
history_get_history_state
            HISTORY_STATE   history_get_hitory_state()              # GRL 6.3
history_set_history_state
            void    history_set_hitory_state(HISTORY_STATE)         # GRL 6.3
```

History List Management

```
add_history(STRING)
            void    add_history(str string)
add_history_time(STRING)
            void    add_history_time(str string)                    # GRL 5.0
remove_history(WHICH)
            str     remove_history(int which)
free_history(HISTENT)
            Not implemented since Term::ReadLine::Gnu does not support the
            member 'data' of HIST_ENTRY structure. remove_history() frees
            the memory.
            histdata_t      free_history_entry(HIST_ENTRY *histent) # GRL 5.0
replace_history_entry(WHICH, STRING)
            str     replace_history_entry(int which, str string)
clear_history
            void    clear_history()
StifleHistory(MAX)
            int     stifle_history(int max|undef)
```

stifles the history list, remembering only the last `MAX` entries. If `MAX` is undef, remembers all entries. This is a replacement of `unstifle_history()`.

```
unstifle_history
            int     unstifle_history()
```

This is equivalent with `stifle_history(undef)`.

```
history_is_stifled
            int     history_is_stifled()
```

`SetHistory(LINE1 [, LINE2, ...])`
sets the history of input, from where it can be used if the actual `readline` is present.

Information About the History List

```
history_list
            Not implemented since not required for Perl.
            HIST_ENTRY **history_list()
```

```
where_history
            int     where_history()
```

```
current_history
            str     current_history()
```

```
history_get(OFFSET)
            str     history_get(offset)
```

```
history_get_time(OFFSET)
            time_t  history_get_time(offset)                        # GRL 5.0
```

```
history_total_bytes
            int     history_total_bytes()
```

`GetHistory`
returns the history of input as a list, if actual `readline` is present.

Moving Around the History List

```
history_set_pos(POS)
            int     history_set_pos(int pos)
```

```
previous_history
            str     previous_history()
```

```
next_history
            str     next_history()
```

Searching the History List

```
history_search(STRING [,DIRECTION])
            int     history_search(str string, int direction = -1)
```

```
history_search_prefix(STRING [,DIRECTION])
            int     history_search_prefix(str string, int direction = -1)
```

```
history_search_pos(STRING [,DIRECTION [,POS]])
            int     history_search_pos(str string,
                                       int direction = -1,
                                       int pos = where_history())
```

Managing the History File

```
ReadHistory([FILENAME [,FROM [,TO]]])
            int     read_history(str filename = '~/.history',
                                 int from = 0, int to = -1)

            int     read_history_range(str filename = '~/.history',
                                       int from = 0, int to = -1)
```

adds the contents of `FILENAME` to the history list, a line at a time. If `FILENAME` is false, then read from *~/.history*. Start reading at line`FROM` and end at `TO`. If`FROM` is omitted or zero, start at the beginning. If`TO` is omitted or less than `FROM`, then read until the end of the file. Returns true if successful, or false if not. `read_history()` is an alias of `read_history_range()`.

```
WriteHistory([FILENAME])
            int     write_history(str filename = '~/.history')
```

writes the current history to `FILENAME`, overwriting `FILENAME` if necessary. If`FILENAME` is f alse, then write the history list to *~/.history*. Returns true if successful, or false if not.

```
append_history(NELEMENTS [,FILENAME])
            int     append_history(int nelements, str filename = '~/.history')

history_truncate_file([FILENAME [,NLINES]])
            int     history_truncate_file(str filename = '~/.history',
                                          int nlines = 0)
```

History Expansion

```
history_expand(STRING)
            (int result, str expansion) history_expand(str string)
```

Note that this function returns `expansion` in the scalar context.

```
get_history_event(STRING, CINDEX [,QCHAR])
            (str text, int cindex) = get_history_event(str  string,
                                                       int  cindex,
                                                       char qchar = '\0')

history_tokenize(STRING)
            (@str)  history_tokenize(str string)

history_arg_extract(STRING, [FIRST [,LAST]])
            str history_arg_extract(str string, int first = 0, int last = '$')
```

`Term::ReadLine::Gnu` Variables

Following GNU Readline/History Library variables can be accessed by a Perl program. See GNU Readline Library Manual <https://tiswww.cwru.edu/php/chet/readline/readline.html> and GNU History Library Manual <https://tiswww.cwru.edu/php/chet/readline/history.html> for details of each variable. You can access them by using `Attribs` methods. Names of keys in this hash conform to standard conventions with the leading `rl_` stripped.

Examples:

```
$attribs = $term->Attribs;
$v = $attribs->{library_version};   # rl_library_version
$v = $attribs->{history_base};      # history_base
```

Readline Variables

```
str rl_line_buffer
int rl_point
int rl_end
int rl_mark
int rl_done
int rl_num_chars_to_read (GRL 4.1)
int rl_pending_input
int rl_dispatching
int rl_erase_empty_line (GRL 4.0)
str rl_prompt (read only)
str rl_display_prompt
int rl_already_prompted (GRL 4.1)
```

```
str rl_library_version (read only)
int rl_readline_version (read only)
int rl_gnu_readline_p (GRL 4.1, read only)
str rl_terminal_name
str rl_readline_name
filehandle rl_instream
filehandle rl_outstream
int rl_prefer_env_winsize (GRL 5.1)
pfunc rl_last_func (GRL 4.2, read only)
pfunc rl_startup_hook
pfunc rl_pre_input_hook (GRL 4.0)
pfunc rl_event_hook
pfunc rl_getc_function
pfunc rl_signal_event_hook (GRL 6.3)
pfunc rl_input_available_hook (GRL 6.3)
pfunc rl_redisplay_function
pfunc rl_prep_term_function (GRL 2.1)
pfunc rl_deprep_term_function (GRL 2.1)
Keymap rl_executing_keymap (read only)
Keymap rl_binding_keymap (read only)
str rl_executing_macro (GRL 4.2, read only)
int rl_executing_key (GRL 6.3, read only)
str rl_executing_keyseq (GRL 6.3, read only)
int rl_key_sequence_length (read only)
int rl_readline_state (GRL 4.2)
int rl_explicit_arg (read only)
int rl_numeric_arg (read only)
int rl_editing_mode (read only)
```

Signal Handling Variables

```
int rl_catch_signals (GRL 4.0)
int rl_catch_sigwinch (GRL 4.0)
int rl_persistent_signal_handlers (GRL 7.0)
int rl_change_environment (GRL 6.3)
```

Completion Variables

```
pfunc rl_completion_entry_function
pfunc rl_attempted_completion_function
pfunc rl_filename_quoting_function
pfunc rl_filename_dequoting_function
pfunc rl_char_is_quoted_p
pfunc rl_ignore_some_completions_function
pfunc rl_directory_completion_hook
pfunc rl_directory_rewrite_hook (GRL 4.2)
pfunc rl_filename_stat_hook (GRL 6.3)
pfunc rl_filename_rewrite_hook (GRL 6.1)
pfunc rl_completion_display_matches_hook (GRL 4.0)
str rl_basic_word_break_characters
str rl_basic_quote_characters
str rl_completer_word_break_characters
pfunc rl_completion_word_break_hook (GRL 5.0)
str rl_completer_quote_characters
str rl_filename_quote_characters
str rl_special_prefixes
int rl_completion_query_items
```

```
int rl_completion_append_character
int rl_completion_suppress_append (GRL 4.3)
int rl_completion_quote_character (GRL 5.0, read only)
int rl_completion_suppress_quote (GRL 5.0)
int rl_completion_found_quote (GRL 5.0, read only)
int rl_completion_mark_symlink_dirs (GRL 4.3)
int rl_ignore_completion_duplicates
int rl_filename_completion_desired
int rl_filename_quoting_desired
int rl_attempted_completion_over
int rl_sort_completion_matches (GRL 6.0)
int rl_completion_type (read only)
int rl_completion_invoking_key (GRL 6.0, read only)
int rl_inhibit_completion
```

History Variables

```
int history_base
int history_length
int history_max_entries (called `max_input_history', read only)
int history_write_timestamps (GRL 5.0)
char history_expansion_char
char history_subst_char
char history_comment_char
str history_word_delimiters (GRL 4.2)
str history_search_delimiter_chars
str history_no_expand_chars
int history_quotes_inhibit_expansion
int history_quoting_state
pfunc history_inhibit_expansion_function
```

Function References

```
rl_getc
rl_redisplay
rl_callback_read_char
rl_display_match_list
rl_filename_completion_function
rl_username_completion_function
list_completion_function
shadow_redisplay
Tk_getc
```

Custom Completion

In this section variables and functions for custom completion are described along with examples.

Most of descriptions in this section came from GNU Readline Library Manual <https://tiswww.cwru.edu/php/chet/readline/readline.html>.

`completion_entry_function`

This variable holds reference refers to a generator function for `completion_matches()`.

A generator function is called repeatedly from `completion_matches()`, returning a string each time. The arguments to the generator function are `TEXT` and `STATE`. `TEXT` is the partial word to be completed. `STATE` is zero the first time the function is called, allowing the generator to perform any necessary initialization, and a positive non-zero integer for each subsequent call. When the generator function returns `undef` this signals `completion_matches()` that there are no more possibilities left.

If this variable set to undef, built-in `filename_completion_function` is used.

A sample generator function, `list_completion_function`, is defined in Gnu.pm. You can use it as follows;

```
use Term::ReadLine;
...
my $term = new Term::ReadLine 'sample';
my $attribs = $term->Attribs;
...
$attribs->{completion_entry_function} =
    $attribs->{list_completion_function};
...
$attribs->{completion_word} =
    [qw(reference to a list of words which you want to use for completion)
$term->readline("custom completion>");
```

See also `completion_matches`.

`attempted_completion_function`

A reference to an alternative function to create matches.

The function is called with `TEXT`, `LINE_BUFFER`, `START`, and `END`. `LINE_BUFFER` is a current input buffer string. `START` and `END` are indices in `LINE_BUFFER` saying what the boundaries of `TEXT` are.

If this function exists and returns null list or `undef`, or if this variable is set to `undef`, then an internal function `rl_complete()` will call the value of `completion_entry_function` to generate matches, otherwise the array of strings returned will be used.

The default value of this variable is `undef`. You can use it as follows;

```
use Term::ReadLine;
...
my $term = new Term::ReadLine 'sample';
my $attribs = $term->Attribs;
...
sub sample_completion {
    my ($text, $line, $start, $end) = @_;
    # If first word then username completion, else filename completion
    if (substr($line, 0, $start) =~ /^\s*$/) {
        return $term->completion_matches($text,
                                         $attribs->{'username_completion_f
    } else {
        return ();
    }
}
...
$attribs->{attempted_completion_function} = \&sample_completion;
```

`completion_matches(TEXT, ENTRY_FUNC)`

Returns an array of strings which is a list of completions for `TEXT`. If there are no completions, returns `undef`. The first entry in the returned array is the substitution for`TEXT`. The remaining entries are the possible completions.

`ENTRY_FUNC` is a generator function which has two arguments, and returns a string. The first argument is `TEXT`. The second is a state argument; it is zero on the first call, and non-zero on subsequent calls. `ENTRY_FUNC` returns `undef` to the caller when there are no more matches.

If the value of `ENTRY_FUNC` is undef, built-in `filename_completion_function` is used.

`completion_matches` is a Perl wrapper function of an internal function

`completion_matches()`. See also`completion_entry_function`.

`completion_function`
A variable whose content is a reference to a function which returns a list of candidates to complete.

This variable is compatible with Term::ReadLine::Perl <http://search.cpan.org/dist/Term-ReadLine-Perl/> and very easy to use.

```
use Term::ReadLine;
...
my $term = new Term::ReadLine 'sample';
my $attribs = $term->Attribs;
...
$attribs->{completion_function} = sub {
    my ($text, $line, $start) = @_;
    return qw(a list of candidates to complete);
};
```

`list_completion_function(TEXT, STATE)`
A sample generator function defined by `Term::ReadLine::Gnu`. Example code at `completion_entry_function` shows how to use this function.

`Term::ReadLine::Gnu` **Specific Features**

`Term::ReadLine::Gnu` Specific Functions

`CallbackHandlerInstall(PROMPT, LHANDLER)`
This method provides the function `rl_callback_handler_install()` with the following additional feature compatible with `readline` method; ornament feature, Term::ReadLine::Perl <http://search.cpan.org/dist/Term-ReadLine-Perl/> compatible completion function, history expansion, and addition to history buffer.

`call_function(FUNCTION, [COUNT [,KEY]])`

```
int     rl_call_function(FunctionPtr|str function, count = 1, key = -1
```

`get_all_function_names`
Returns a list of all function names.

`shadow_redisplay`
A redisplay function for password input. You can use it as follows;

```
$attribs->{redisplay_function} = $attribs->{shadow_redisplay};
$line = $term->readline("password> ");
```

`filename_list`
Returns candidates of filenames to complete. This function can be used with `completion_function` and is implemented for the compatibility with Term::ReadLine::Perl <http://search.cpan.org/dist/Term-ReadLine-Perl/>.

`list_completion_function`
See the description of section ‘‘Custom Completion’’.

`Term::ReadLine::Gnu` Specific Variables

`do_expand`
When true, the history expansion is enabled. By default false.

`completion_function`
See the description of section ‘‘Custom Completion’’.

`completion_word`
A reference to a list of candidates to complete for `list_completion_function`.

`Term::ReadLine::Gnu` Specific Commands

`history-expand-line`
The equivalent of the Bash `history-expand-line` editing command.

`operate-and-get-next`
The equivalent of the Korn shell `operate-and-get-next-history-line` editing command and the Bash `operate-and-get-next`.

This command is bound to `\C-o` by default for the compatibility with the Bash and Term::ReadLine::Perl <http://search.cpan.org/dist/Term-ReadLine-Perl/>.

`display-readline-version`
Shows the version of `Term::ReadLine::Gnu` and the one of the GNU Readline Library.

`change-ornaments`
Change ornaments interactively.

FILES

~/.inputrc
Readline init file. Using this file it is possible that you would like to use a different set of key bindings. When a program which uses the GNU Readline library starts up, the init file is read, and the key bindings are set.

The conditional init constructs is supported. The program name which is specified by the first argument of `new` method is used as the application construct.

For example, when your program calls `new` method as follows;

```
...
$term = new Term::ReadLine 'PerlSh';
...
```

your *~/.inputrc* can define key bindings only for the program as follows;

```
...
$if PerlSh
Meta-Rubout: backward-kill-word
"\C-x\C-r": re-read-init-file
"\e[11~": "Function Key 1"
$endif
...
```

For further details, see the section ‘‘Readline Init File’’ in the GNU Readline Library Manual <https://tiswww.cwru.edu/php/chet/readline/readline.html>

EXPORTS

No symbols are exported by default. The following tags are defined and their symbols can be exported.

prompt
RL_PROMPT_START_IGNORE RL_PROMPT_END_IGNORE

match_type
NO_MATCH SINGLE_MATCH MULT_MATCH

keymap_type
ISFUNC ISKMAP ISMACR

undo_code
UNDO_DELETE UNDO_INSERT UNDO_BEGIN UNDO_END

rl_state
RL_STATE_NONE RL_STATE_INITIALIZING RL_STATE_INITIALIZED RL_STATE_TERMPREPPED RL_STATE_READCMD RL_STATE_METANEXT RL_STATE_DISPATCHING RL_STATE_MOREINPUT RL_STATE_ISEARCH RL_STATE_NSEARCH RL_STATE_SEARCH RL_STATE_NUMERICARG RL_STATE_MACROINPUT RL_STATE_MACRODEF RL_STATE_OVERWRITE RL_STATE_COMPLETING

RL_STATE_SIGHANDLER RL_STATE_UNDOING RL_STATE_INPUTPENDING RL_STATE_TTYCSAVED RL_STATE_CALLBACK RL_STATE_VIMOTION RL_STATE_MULTIKEY RL_STATE_VICMDONCE RL_STATE_CHARSEARCH RL_STATE_REDISPLAYING RL_STATE_DONE

They can be exported as follows;

```
use Term::ReadLine;
BEGIN {
    import Term::ReadLine::Gnu qw(:keymap_type RL_STATE_INITIALIZED);
}
```

ENVIRONMENT

The environment variable `PERL_RL` governs which ReadLine clone is loaded. See the ENVIRONMENT section on Term::ReadLine <http://search.cpan.org/dist/Term-ReadLine/> for further details.

SEE ALSO

Term::ReadLine::Gnu Project Home Page <https://github.com/hirooih/perl-trg>

GNU Readline Library Manual <https://tiswww.cwru.edu/php/chet/readline/readline.html>

GNU History Library Manual <https://tiswww.cwru.edu/php/chet/readline/history.html>

Sample and test programs (*eg/** and *t/**) in the `Term::ReadLine::Gnu` distribution <http://search.cpan.org/dist/Term-ReadLine-Gnu/>

Term::ReadLine <http://search.cpan.org/dist/Term-ReadLine/>

Works which use Term::ReadLine::Gnu

Distributions which depend on Term::ReadLine::Gnu on CPAN <http://www.cpan.org/>
<https://metacpan.org/requires/distribution/Term−ReadLine−Gnu>

Perl Debugger <http://perldoc.perl.org/perldebug.html>

```
perl −d
```

Perl Shell (psh) <http://gnp.github.io/psh/>

The Perl Shell is a shell that combines the interactive nature of a Unix shell with the power of Perl.

A programmable completion feature compatible with bash is implemented.

SPP (Synopsys Plus Perl) <http://vlsiweb.stanford.edu/~jsolomon/SPP/>

SPP (Synopsys Plus Perl) is a Perl module that wraps around Synopsys' shell programs. SPP is inspired by the original dc_perl written by Steve Golson, but it's an entirely new implementation. Why is it called SPP and not dc_perl? Well, SPP was written to wrap around any of Synopsys' shells.

PFM (Personal File Manager for Unix/Linux) <http://p-f-m.sourceforge.net/>

Pfm is a terminal-based file manager written in Perl, based on PFM.COM for MS-DOS (originally by Paul Culley and Henk de Heer).

The soundgrab <https://sourceforge.net/projects/soundgrab/>

soundgrab is designed to help you slice up a big long raw audio file (by default 44.1 kHz 2 channel signed sixteen bit little endian) and save your favorite sections to other files. It does this by providing you with a cassette player like command line interface.

PDL (The Perl Data Language) <http://pdl.perl.org/>

PDL (''Perl Data Language'') gives standard Perl the ability to compactly store and speedily manipulate the large N−dimensional data arrays which are the bread and butter of scientific computing.

PIQT (Perl Interactive DBI Query Tool) <http://piqt.sourceforge.net/>

PIQT is an interactive query tool using the Perl DBI database interface. It supports ReadLine, provides a built in scripting language with a Lisp like syntax, an online help system, and uses wrappers to interface to the DBD modules.

vshnu (the New Visual Shell) <http://www.cs.indiana.edu/~kinzler/vshnu/>
A visual shell and CLI shell supplement.

If you know any other works you recommend, please let me know.

AUTHOR

Hiroo Hayashi `<hiroo.hayashi@computer.org>`

<http://search.cpan.org/~hayashi/>

TODO

GTK+ support in addition to Tk.

BUGS

Submit a bug report to the bug tracker on GitHub <https://github.com/hirooih/perl-trg/issues>.
`add_defun()` can define up to 16 functions.
Some functions and variables do not have test code yet. Your contribution is welcome. See *t/readline.t* for details.
If the pager command (| or ||) in Perl debugger causes segmentation fault, you need to fix *perl5db.pl*. See <https://rt.perl.org/Public/Bug/Display.html?id=121456> for details.

LICENSE