rl Documentation

Release 3.1

Stefan H. Holek

Oct 31, 2022

CONTENTS

Alternative Python bindings for GNU Readline.

CHAPTER

ONE

OVERVIEW

Alternative Python bindings for GNU Readline.

1.1 Package Contents

rl exports the following components:

rl.completer
: Interface to the readline completer. Used to configure the completion aspects of readline.

rl.completion
: Interface to the active readline completion. Used to interact with readline when a completion is in progress.

rl.history
: Interface to the readline history. Used to read and write history files and to manipulate history entries.

rl.readline
: The readline bindings module. Contains everything known from the standard library plus extensions specific to the rl package. The *completer*, *completion*, and *history* interfaces make use of this module, and you should rarely need to interact with it directly.

rl.generator()
: A decorator turning any callable into a *completion entry function* that can be handed to readline.

rl.print_exc()
: A decorator printing exceptions to stderr. Useful when writing Python completions and hooks, as exceptions occurring there are usually swallowed by the in-between C code.

1.2 Readline Completion

Completion is the process initiated when the user presses the TAB key. It has three phases: Word breaking, match generation, and match insertion/display.

For each phase, readline provides configuration settings and hooks that allow applications to control the way the library behaves. See the *completer* and *completion* objects for detailed descriptions of available properties.

1.2.1 Call Graph

A calling sequence for filename completion may look like this:

- `complete_internal()`
  - `find_completion_word()`
    - *word_break_hook*
    - *char_is_quoted_function*
  - `gen_completion_matches()`
    - *completer*
      - *complete_filename()*
      - *directory_rewrite_hook* or
      - *directory_completion_hook*
      - *filename_dequoting_function*
      - *filename_rewrite_hook*
    - *ignore_some_completions_function*
  - `insert_match()`
    - *filename_quoting_function*
    - *filename_stat_hook*
  - `display_matches()`
    - *display_matches_hook*
      - *display_match_list()*
      - *directory_completion_hook* or
      - *filename_stat_hook*

1.3 Readline History

History allows readline to save and later recall lines the user has entered. The *history* object provides a list-like interface to the history buffer as well as functions to persist the history between sessions.

1.4 Upstream Documentation

The GNU Readline Library and the GNU History Library.

CHAPTER

TWO

COMPLETION SUPPORT

Readline completion support.

2.1 Completer Interface

class rl.**Completer**

Interface to the readline completer. Used to configure the completion aspects of readline.

This class is not intended for instantiation beyond the one *completer* object in this module. Typically, applications will import the *completer* object and use its properties and methods to configure readline:

```
from rl import completer

completer.quote_characters = '"\''
completer.query_items = 100
completer.parse_and_bind('TAB: complete')
```

Settings made through the *completer* object are global and permanent. If you want them restored you have to take care of it yourself.

Completer.**quote_characters**

Characters that may be used in pairs to quote substrings of the line.

Completer.**word_break_characters**

Characters that define word boundaries (a.k.a. delimiters).

Completer.**special_prefixes**

Characters that are word break characters but should be left in the word passed to the completion entry function.

Completer.**filename_quote_characters**

Characters that must be quoted when they occur in filenames.

Completer.**inhibit_completion**

If True, completion is disabled and the completion character is inserted as any other character. Defaults to False.

Completer.**query_items**

Threshold above which the user is prompted if they really want to see all matches. Defaults to 100. A negative value means never prompt.

Completer.**completer**

The completion entry function. The function is called as `function(text, state)` for `state` in 0, 1, 2, ... until it returns None. It should return the next possible completion for `text`. See the *generator()* factory for a simple way to support this protocol.

Completer.**startup_hook**

The startup hook function. The function is called with no arguments just before readline prints the first prompt.

Completer.**pre_input_hook**

The pre-input hook function. The function is called with no arguments after the first prompt has been printed and just before readline starts reading input characters.

Completer.**word_break_hook**

The word break hook function. The function is called as `function(begidx, endidx)` once per completion and should return a string of word break characters for the current completion or None to indicate no change. The passed-in `begidx` and `endidx` are what readline would use if the hook did not exist.

Completer.**char_is_quoted_function**

The char-is-quoted function. The function is called as `function(text, index)` and should return True if the character at `index` is quoted, False otherwise.

Completer.**filename_quoting_function**

The filename quoting function. The function is called as `function(text, single_match, quote_char)` and should return a quoted version of `text` or None to indicate no change. The `single_match` argument is True if the completion has generated only one match.

Completer.**filename_dequoting_function**

The filename dequoting function. The function is called as `function(text, quote_char)` and should return a dequoted version of `text` or None to indicate no change.

Completer.**directory_completion_hook**

The directory completion hook function. This function is allowed to modify the directory portion of filenames readline completes. The function is called as `function(dirname)` and should return a new directory name or None to indicate no change. At the least, the function must perform all necessary dequoting.

Completer.**ignore_some_completions_function**

The filename filter function. The function is called as `function(substitution, matches)` after all filenames have been generated and should return a filtered subset of `matches` or None to indicate no change.

Completer.**display_matches_hook**

The display matches hook function. The function is called as `function(substitution, matches, longest_match_length)` once each time matches need to be displayed. It typically calls *display_match_list()* to do the actual work. Note that `longest_match_length` is not a character count but the "printed length" of the longest string in `matches`.

Completer.**read_init_file**(*filename=None*)

Parse a readline initialization file. The default filename is the last filename used.

Completer.**parse_and_bind**(*line*)

Parse one line of a readline initialization file.

2.1.1 Additional hooks for when the filesystem representation differs from the representation in the terminal

Completer.**directory_rewrite_hook**

The directory rewrite hook function. This hook is used to prepare the directory name passed to `opendir()` during filename completion. The function is called as `function(dirname)` and should return a new directory name or None to indicate no change. At the least, the function must perform all necessary dequoting. New in readline 6.2.

Under Python 3 this hook returns filesystem encoding to readline.

Completer.**filename_rewrite_hook**

The filename rewrite hook function. This hook is called for every filename before it is compared against the completion word. The function is called as `function(filename)` and should return a new filename or None to indicate no change. New in readline 6.1.

Under Python 3 this hook returns preferred encoding to readline.

Completer.**filename_stat_hook**

The filename stat hook function. This hook is used to prepare the filename passed to `stat()` during match display. The function is called as `function(filename)` and should return a new filename or None to indicate no change. New in readline 6.3.

Under Python 3 this hook returns filesystem encoding to readline.

Note: If *directory_rewrite_hook* and/or *filename_stat_hook* are set, the *directory_completion_hook* must be None, and vice versa.

2.2 Completion Interface

class rl.**Completion**

Interface to the active readline completion. Used to interact with readline when a completion is in progress.

This class is not intended for instantiation beyond the one *completion* object in this module. Typically, applications will import the *completion* object and use its properties and methods when implementing custom completions:

```
from rl import completion

def complete(text):
    completion.append_character = '@'
    return completion.complete_username(text)
```

Settings made through the *completion* object are only valid for the duration of the current completion. They are reset to their defaults when a new completion starts.

Completion.**line_buffer**

The line buffer readline uses. This property may be assigned to to change the contents of the line.

Completion.**completion_type**

The type of completion readline performs.

Completion.**begidx**

The start index of the word in the line.

Completion.**endidx**

The end index of the word in the line.

Completion.**found_quote**

True if the word contains or is delimited by any quote character, including backslashes.

Completion.**quote_character**

The quote character found (not including backslashes).

Completion.**suppress_quote**

Do not append a matching quote character when completing a quoted string. Defaults to False.

Completion.**append_character**

The character appended when the completion returns a single match. Defaults to the space character.

Completion.**suppress_append**

Suppress the append character for this completion. Defaults to False.

Completion.**filename_completion_desired**

Treat matches as filenames. Directory names will have a slash appended, for example. Defaults to False. Set to True by *complete_filename()*.

Completion.**filename_quoting_desired**

If matches are filenames, quote them. Defaults to True. Has no effect if *filename_completion_desired* is False.

Completion.**complete_filename**(*text*)

Built-in filename completion. May be called from a completion entry function to initiate readline's filename completion. Returns a list of matches.

Completion.**complete_username**(*text*)

Built-in username completion. May be called from a completion entry function to initiate readline's username completion. Returns a list of matches.

Completion.**expand_tilde**(*text*)

Built-in tilde expansion. May be called from anywhere to tilde-expand a filename.

Completion.**display_match_list**(*substitution*, *matches*, *longest_match_length*)

Built-in matches display. May be called from a custom *display_matches_hook* to perform the default action: columnar display of matches.

Completion.**redisplay**(*force=False*)

Update the screen to reflect the current contents of *line_buffer*. If `force` is True, readline redisplays the prompt area as well as the line.

2.3 Functions

rl.**generator**(*func*)

Generator function factory.

Takes a function returning a list of matches and returns an object implementing the generator protocol readline requires. The function is called as `function(text)` and should return an iterable of matches for `text`.

rl.**print_exc**(*func*)

Decorator printing exceptions to stderr.

Useful when debugging completions and hooks, as exceptions occurring there are usually swallowed by the in-between C code.

CHAPTER

THREE

HISTORY SUPPORT

Readline history support.

3.1 History Interface

class rl.**History**

Interface to the readline history. Used to read and write history files and to manipulate history entries.

This class is not intended for instantiation beyond the one *history* object in this module. Typically, applications will import the *history* object and use its properties and methods to work with readline history:

```
from rl import history

history.max_entries = 300
history.read_file(histfile)
```

History entries can be accessed like elements in a Python list. The item at index 0 is the oldest, the item at -1 the most recent history entry.

History.**auto**

Controls whether readline automatically adds lines to the history. Defaults to True. Set to False if you want to call *append()* yourself.

History.**max_entries**

The maximum number of history entries kept. Beyond this point the history list is truncated by removing the oldest entry. A negative value means no limit. Defaults to -1.

History.**max_file**

The maximum size of a readline history file, in entries. Beyond this point the history file is truncated by removing the oldest entries. A negative value means no limit. Defaults to -1.

History.**append**(*line*)

Append a line to the history.

History.**__getitem__**(*index*)

Return the history item at index.

History.**__setitem__**(*index*, *line*)

Replace the history item at index.

History.**__delitem__**(*index*)

Remove the history item at index.

History.__len__()

The current history length.

History.__iter__()

Iterate over history items (old to new).

History.__reversed__()

Reverse-iterate over history items (new to old).

History.clear()

Clear the history.

History.read_file(*filename=None*, *raise_exc=False*)

Load a readline history file. The default filename is ~/.history. If raise_exc is True, IOErrors will be allowed to propagate.

History.write_file(*filename=None*, *raise_exc=False*)

Save a readline history file. The default filename is ~/.history. If raise_exc is True, IOErrors will be allowed to propagate.

History.append_file(*numitems*, *filename=None*, *raise_exc=False*)

Append the last numitems history entries to a readline history file. The default filename is ~/.history. If raise_exc is True, IOErrors will be allowed to propagate.

CHAPTER

FOUR

READLINE BINDINGS

Importing this module enables command line editing using GNU Readline.

4.1 Readline Interface

The *rl.readline* module is an API-compatible replacement for the standard library's `readline` bindings. The standard library documentation applies, with the following exceptions:

1. *get_completion_type()* returns a string.
2. *get_completion_append_character()* defaults to the space character.
3. *get_history_item()* is zero-based.
4. *redisplay()* accepts an optional `force` argument.

Beyond that, *rl.readline* adds a plethora of new functionality which is typically accessed through the high-level interfaces *rl.completer*, *rl.completion*, and *rl.history*. Functions not exposed through a high-level interface:

- *readline_version()* returns the readline library version as an integer.
- *read_key()* reads a character from the keyboard.
- *stuff_char()* stuffs a character into the input stream.
- *complete_internal()* executes the completer. Used in tests.

Note: It is possible to use `rl.readline` without the high-level APIs. To switch an existing application to `rl`, change occurrences of `import readline` to `from rl import readline`.

Note: Applications must not use the standard library `readline` and `rl.readline` simultaneously. This is because only one module can own the `PyOS_ReadlineFunctionPointer`.

4.2 Functions

rl.readline.**add_history**(*string*) → None

Add a line to the readline history.

rl.readline.**append_history_file**(*nelements*[, *filename*]) → None

Append the last nelements of the history to a readline history file. The default filename is ~/.history.

rl.readline.**clear_history**() → None

Clear the current readline history.

rl.readline.**complete_internal**(*what_to_do*) → int

Complete the word at or before the cursor position.

rl.readline.**display_match_list**(*substitution*, *matches*, *longest_match_length*) → None

Display a list of matches in columnar format on readline's output stream.

rl.readline.**filename_completion_function**(*text*, *state*) → string

A built-in generator function for filename completion.

rl.readline.**get_auto_history**() → bool

True if automatic history is enabled.

rl.readline.**get_begidx**() → int

Get the beginning index of the readline tab-completion scope.

rl.readline.**get_char_is_quoted_function**() → function

Get the function that determines whether or not a specific character in the line buffer is quoted.

rl.readline.**get_completer**() → function

Get the current completion entry function.

rl.readline.**get_completer_delims**() → string

Get the readline word delimiters for tab-completion.

rl.readline.**get_completer_quote_characters**() → string

Get list of characters that may be used to quote a substring of the line.

rl.readline.**get_completion_append_character**() → string

Get the character appended after the current completion.

rl.readline.**get_completion_display_matches_hook**() → function

Get the current completion display function.

rl.readline.**get_completion_found_quote**() → bool

When readline is completing quoted text, it sets this variable to True if the word being completed contains any quoting character (including backslashes).

rl.readline.**get_completion_query_items**() → int

Up to this many items will be displayed in response to a possible-completions call.

rl.readline.**get_completion_quote_character**() → string

When readline is completing quoted text, it sets this variable to the quoting character found.

rl.readline.**get_completion_suppress_append**() → bool

Do not append the completion_append_character after the current completion.

rl.readline.**get_completion_suppress_quote**() → bool
Do not append a matching quote character when performing completion on a quoted string.

rl.readline.**get_completion_type**() → string
Get the type of completion being attempted.

rl.readline.**get_completion_word_break_hook**() → function
A function to call when readline is deciding where to separate words for word completion.

rl.readline.**get_current_history_length**() → int
Return the current (not the maximum) length of history.

rl.readline.**get_directory_completion_hook**() → function
Get the current directory completion hook function.

rl.readline.**get_directory_rewrite_hook**() → function
Get the current directory rewrite hook function.

rl.readline.**get_endidx**() → int
Get the ending index of the readline tab-completion scope.

rl.readline.**get_filename_completion_desired**() → bool
If True, treat the results of matches as filenames.

rl.readline.**get_filename_dequoting_function**() → function
Get the current filename dequoting function.

rl.readline.**get_filename_quote_characters**() → string
Get list of characters that cause a filename to be quoted by the completer.

rl.readline.**get_filename_quoting_desired**() → bool
If True, filenames will be quoted.

rl.readline.**get_filename_quoting_function**() → function
Get the current filename quoting function.

rl.readline.**get_filename_rewrite_hook**() → function
Get the current filename rewrite hook function.

rl.readline.**get_filename_stat_hook**() → function
Get the current filename stat hook function.

rl.readline.**get_history_item**(*pos*) → string
Return the current contents of history item at pos.

rl.readline.**get_history_iter**() → iterator
Return a forward iterator over the history (oldest to newest).

rl.readline.**get_history_length**() → int
Return the maximum number of items written to the history file.

rl.readline.**get_history_list**() → list
Return the entire history as a Python list. Element 0 of the list is the beginning of time.

rl.readline.**get_history_max_entries**() → int
Return the current history size limit.

rl.readline.**get_history_reverse_iter**() → iterator

Return a reverse iterator over the history (newest to oldest).

rl.readline.**get_ignore_some_completions_function**() → function

This function may filter the results of filename completion.

rl.readline.**get_inhibit_completion**() → bool

If True, completion is disabled.

rl.readline.**get_line_buffer**() → string

Return the current contents of the line buffer.

rl.readline.**get_pre_input_hook**() → function

Get the current pre_input_hook function.

rl.readline.**get_rl_end**() → int

Return rl_end.

rl.readline.**get_rl_point**() → int

Return rl_point.

rl.readline.**get_special_prefixes**() → string

Characters that are word break characters, but should be left in text when it is passed to the completion function.

rl.readline.**get_startup_hook**() → function

Get the current startup_hook function.

rl.readline.**history_is_stifled**() → bool

True if a history size limit is set.

rl.readline.**insert_text**(*string*) → None

Insert text into the command line.

rl.readline.**parse_and_bind**(*string*) → None

Parse and execute single line of a readline init file.

rl.readline.**read_history_file**([*filename*]) → None

Load a readline history file. The default filename is ~/.history.

rl.readline.**read_init_file**([*filename*]) → None

Parse a readline initialization file. The default filename is the last filename used.

rl.readline.**read_key**() → string

Read a key from readline's input stream, typically the keyboard. Returns characters inserted with *stuff_char()* before starting to read from the stream.

rl.readline.**readline_version**() → int

Return the readline library version encoded in an integer. The format is `0xMMmm`, where `MM` is the major and `mm` the minor version number.

rl.readline.**redisplay**([*force*]) → None

Update the screen to reflect the current contents of the line buffer. If `force` is True, readline redisplays the prompt area as well as the line.

rl.readline.**remove_history_item**(*pos*) → None

Remove history item given by its position.

rl.readline.**replace_history_item**(*pos*, *string*) → None

Replace history item given by its position with string.

rl.readline.**replace_line**(*string*) → None

Replace the line buffer contents with string.

rl.readline.**set_auto_history**(*bool*) → None

Enable or disable automatic history.

rl.readline.**set_begidx**(*int*) → None

Set the beginning index of the readline tab-completion scope.

rl.readline.**set_char_is_quoted_function**([*function*]) → None

Set or remove the function that determines whether or not a specific character in the line buffer is quoted. The function is called as `function(text, index)` and should return True if the character at `index` is quoted, and False otherwise.

rl.readline.**set_completer**([*function*]) → None

Set or remove the completion entry function. The function is called as `function(text, state)`, for `state` in 0, 1, 2, ..., until it returns None. It should return the next possible completion starting with `text`.

rl.readline.**set_completer_delims**(*string*) → None

Set the readline word delimiters for tab-completion.

rl.readline.**set_completer_quote_characters**(*string*) → None

Set list of characters that may be used to quote a substring of the line.

rl.readline.**set_completion_append_character**(*string*) → None

Set the character appended after the current completion. May only be called from within custom completers.

rl.readline.**set_completion_display_matches_hook**([*function*]) → None

Set or remove the completion display function. The function is called as `function(substitution, matches, longest_match_length)` once each time matches need to be displayed.

rl.readline.**set_completion_found_quote**(*bool*) → None

When readline is completing quoted text, it sets this variable to True if the word being completed contains any quoting character (including backslashes).

rl.readline.**set_completion_query_items**(*int*) → None

Up to this many items will be displayed in response to a possible-completions call.

rl.readline.**set_completion_quote_character**(*string*) → None

When readline is completing quoted text, it sets this variable to the quoting character found.

rl.readline.**set_completion_suppress_append**(*bool*) → None

Do not append the completion_append_character after the current completion. May only be called from within custom completers.

rl.readline.**set_completion_suppress_quote**(*bool*) → None

Do not append a matching quote character when performing completion on a quoted string. May only be called from within custom completers.

rl.readline.**set_completion_type**(*string*) → None

Set the type of completion being attempted.

rl.readline.**set_completion_word_break_hook**([*function*]) → None

A function to call when readline is deciding where to separate words for word completion. The function is called as `function(begidx, endidx)` once for every completion, and should return a string of word break characters for the current completion, or None to indicate no change.

rl.readline.**set_directory_completion_hook**([*function*]) → None

This function is allowed to modify the directory portion of filenames readline completes. The function is called as `function(dirname)` and should return a new directory name or None to indicate no change. At the least, the function must perform all necessary dequoting.

rl.readline.**set_directory_rewrite_hook**([*function*]) → None

This function is used to prepare the director name passed to `opendir()` during filename completion. The function is called as `function(dirname)` and should return a new directory name or None to indicate no change. At the least, the function must perform all necessary dequoting.

rl.readline.**set_endidx**(*int*) → None

Set the ending index of the readline tab-completion scope.

rl.readline.**set_filename_completion_desired**(*bool*) → None

If True, treat the results of matches as filenames. May only be called from within custom completers.

rl.readline.**set_filename_dequoting_function**([*function*]) → None

Set or remove the filename dequoting function. The function is called as `function(text, quote_char)` and should return a string representing a dequoted version of `text`, or None to indicate no change.

rl.readline.**set_filename_quote_characters**(*string*) → None

Set list of characters that cause a filename to be quoted by the completer.

rl.readline.**set_filename_quoting_desired**(*bool*) → None

If True, filenames will be quoted. May only be called from within custom completers.

rl.readline.**set_filename_quoting_function**([*function*]) → None

Set or remove the filename quoting function. The function is called as `function(text, single_match, quote_char)` and should return a string representing a quoted version of `text`, or None to indicate no change. The `single_match` argument is True if the completion has generated only one match.

rl.readline.**set_filename_rewrite_hook**([*function*]) → None

This function is called for every filename before it is compared against the completion word. The function is called as `function(filename)` and should return a new filename or None to indicate no change.

rl.readline.**set_filename_stat_hook**([*function*]) → None

This function is used to prepare the filename passed to `stat()` during match display. The function is called as `function(filename)` and should return a new filename name or None to indicate no change.

rl.readline.**set_history_length**(*int*) → None

Set the maximum number of items written to the history file. A negative value inhibits history file truncation.

rl.readline.**set_ignore_some_completions_function**([*function*]) → None

This function may filter the results of filename completion. The function is called as `function(substitution, matches)` and should return a filtered subset of matches or None to indicate no change.

rl.readline.**set_inhibit_completion**(*bool*) → None

If True, completion is disabled and the completion character is inserted as any other character.

rl.readline.**set_pre_input_hook**([*function*]) → None

Set or remove the pre_input_hook function. The function is called with no arguments after the first prompt has been printed and just before readline starts reading input characters.

rl.readline.**set_special_prefixes**(*string*) → None

Characters that are word break characters, but should be left in text when it is passed to the completion function.

rl.readline.**set_startup_hook**([*function*]) → None

Set or remove the startup_hook function. The function is called with no arguments just before readline prints the first prompt.

rl.readline.**stifle_history**(*max_entries*) → None

Limit the history size to `max_entries` entries.

rl.readline.**stuff_char**(*string*) → bool

Insert a character into readline's input stream. Returns True if the insert was successful.

rl.readline.**tilde_expand**(*string*) → string

Return a new string which is the result of tilde expanding string.

rl.readline.**unstifle_history**() → int

Remove the history size limit.

rl.readline.**username_completion_function**(*text*, *state*) → string

A built-in generator function for username completion.

rl.readline.**write_history_file**([*filename*]) → None

Save a readline history file. The default filename is ~/.history.

CHAPTER

FIVE

EXAMPLES

Example code.

5.1 Completion Entry Function

The completion entry function is called as `function(text, state)` for `state` in 0, 1, 2, ... until it returns None. It should return the next possible completion for `text`. You can run this example with `python -m rl.examples.raw_input`.

```
# Complete system commands

import os

from rl import completer


class CommandCompleter:
    # A completion entry function implementing readline's
    # generator protocol

    def __call__(self, text, state):
        if state == 0:
            self.matches = list(self.complete_command(text))
        try:
            return self.matches[state]
        except IndexError:
            return None

    def complete_command(self, text):
        # Return executables matching 'text'
        for dir in os.environ.get('PATH').split(':'):
            if os.path.isdir(dir):
                for name in os.listdir(dir):
                    if name.startswith(text):
                        if os.access(os.path.join(dir, name), os.R_OK|os.X_OK):
                            yield name


def main():
    # Set the completion entry function
```

(continues on next page)

(continued from previous page)

```
    completer.completer = CommandCompleter()

    # Enable TAB completion
    completer.parse_and_bind('TAB: complete')

    command = input('command> ')
    print('You typed:', command)


if __name__ == '__main__':
    main()
```

5.2 Generator Factory

The *generator()* factory provides a simple way to support this protocol. It is typically used as a decorator but can be passed any callable to create a completion entry function. You can run this example with `python -m rl.examples.factory`.

```
# Complete system commands

import os

from rl import completer
from rl import generator
from rl import print_exc


@print_exc
@generator
def complete_command(text):
    # Return executables matching 'text'
    for dir in os.environ.get('PATH').split(':'):
        if os.path.isdir(dir):
            for name in os.listdir(dir):
                if name.startswith(text):
                    if os.access(os.path.join(dir, name), os.R_OK|os.X_OK):
                        yield name


def main():
    # Set the completion entry function
    completer.completer = complete_command

    # Enable TAB completion
    completer.parse_and_bind('TAB: complete')

    command = input('command> ')
    print('You typed:', command)
```

(continues on next page)

(continued from previous page)

```
if __name__ == '__main__':
    main()
```

5.3 Multiple Completions

The completion entry function is often a dispatcher, forwarding calls to more specific completion functions depending on position and format of the completion word. You can run this example with `python -m rl.examples.email`.

```
# Complete email addresses

from rl import completer
from rl import completion
from rl import generator
from rl import print_exc

from rl.utils import DEFAULT_DELIMS


def complete_hostname(text):
    # Search /etc/hosts for matching hostnames
    with open('/etc/hosts', 'rt') as f:
        lines = f.readlines()
    for line in lines:
        line = line.split()
        if line and not line[0].startswith('#'):
            for hostname in line[1:]:
                if hostname.startswith(text[1:]):
                    yield '@' + hostname


@print_exc
@generator
def complete_email(text):
    # Dispatch to username or hostname completion
    if text.startswith('@'):
        return complete_hostname(text)
    else:
        completion.append_character = '@'
        return completion.complete_username(text)


def main():
    # Configure word break characters
    completer.word_break_characters = DEFAULT_DELIMS.replace('-', '')

    # Configure special prefixes
    completer.special_prefixes = '@'

    # Set the completion entry function
    completer.completer = complete_email
```

(continues on next page)

(continued from previous page)

```
    # Enable TAB completion
    completer.parse_and_bind('TAB: complete')

    email = input('email> ')
    print('You typed:', email)


if __name__ == '__main__':
   main()
```

5.4 Filename Completion

Filename completion is readline's party trick. It is also the most complex feature, requiring various parts of readline to be set up. You can run this example with `python -m rl.examples.filename`.

```
# Complete filenames

import sys
import unicodedata

from rl import completer
from rl import completion
from rl import generator
from rl import print_exc


@print_exc
def char_is_quoted(text, index):
    # Return True if the character at index is quoted
    return index > 0 and text[index-1] == '\\'


@print_exc
def quote_filename(text, single_match, quote_char):
    # Backslash-quote characters in text
    if quote_char == "'":
        pass
    elif quote_char == '"':
        for c in '\\"$`':
            text = text.replace(c, '\\'+c)
    else:
        for c in completer.filename_quote_characters:
            text = text.replace(c, '\\'+c)
    return text


@print_exc
def dequote_filename(text, quote_char):
    # Backslash-dequote characters in text
```

(continues on next page)

(continued from previous page)

```
    if quote_char == "'":
        pass
    elif quote_char == '"':
        for c in '\\"$`':
            text = text.replace('\\'+c, c)
    else:
        for c in completer.filename_quote_characters:
            text = text.replace('\\'+c, c)
    return text


@print_exc
def rewrite_filename(text):
    # Normalize decomposed UTF-8 received from HFS Plus
    return unicodedata.normalize('NFC', text)


@print_exc
@generator
def complete_filename(text):
    matches = []
    # Complete usernames
    if text.startswith('~') and '/' not in text:
        matches = completion.complete_username(text)
    # Complete filenames
    if not matches:
        matches = completion.complete_filename(text)
    return matches


def main():
    # Configure quote characters
    completer.quote_characters = '\'"'
    completer.word_break_characters = ' \t\n"\'><;|&=(:'
    completer.filename_quote_characters = '\\ \t\n"\'@><;|&=()#$`?*[!:{'

    # Configure quoting functions
    completer.char_is_quoted_function = char_is_quoted
    completer.filename_quoting_function = quote_filename
    completer.filename_dequoting_function = dequote_filename

    # Configure Unicode converter on Mac OS X
    if sys.platform == "darwin":
        completer.filename_rewrite_hook = rewrite_filename

    # Set the completion entry function
    completer.completer = complete_filename

    # Enable TAB completion
    completer.parse_and_bind('TAB: complete')

    filename = input('file> ')
```

(continues on next page)

(continued from previous page)

```
    print('You typed:', filename)


if __name__ == '__main__':
    main()
```

5.5 Display Matches Hook

The *display_matches_hook* is called whenever matches need to be displayed.

```
# Python implementation of the default display_matches_hook

import sys

from rl import completer
from rl import completion
from rl import readline
from rl import print_exc


@print_exc
def display_matches_hook(substitution, matches, longest_match_length):
    num_matches = len(matches)
    if num_matches >= completer.query_items > 0:
        sys.stdout.write('\nDisplay all %d possibilities? (y or n)' % num_matches)
        sys.stdout.flush()
        while True:
            c = readline.read_key()
            if c in 'yY\x20': # SPACEBAR
                break
            if c in 'nN\x7f': # RUBOUT
                sys.stdout.write('\n')
                completion.redisplay(force=True)
                return
    completion.display_match_list(substitution, matches, longest_match_length)
    completion.redisplay(force=True)
```

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

r

INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

M

P

Q

R

S

T

U

W