kmd Documentation

Release 2.5.dev0

Stefan H. Holek

Sep 22, 2023

CONTENTS

A framework for building command interpreters and shells.

CHAPTER

ONE

OVERVIEW

A framework for building command interpreters and shells.

1.1 Interpreters

The *kmd.Kmd* class provides a simple framework for writing line-oriented command interpreters, also known as *shells*. These are often useful for test harnesses, prototypes, and administrative tools.

A *kmd.Kmd* instance is a line-oriented command interpreter. There is no good reason to instantiate *kmd.Kmd* itself; rather, it is used as base class for interpreter classes you define.

1.2 Completions

The *kmd.completions* package defines the *custom completion protocol* and implements a set of ready-to-use completions for *kmd.Kmd*. Applications may use the provided completions and/or add their own, domain-specific completions based on code in this package.

1.3 Quoting

The *kmd.quoting* module defines constants and functions for writing custom completions.

1.4 Upstream Documentation

The standard library documentation for cmd.Cmd.

The rl GNU Readline Bindings.

CHAPTER

TWO

INTERPRETERS

A base class for custom command interpreters.

2.1 Kmd Class

class kmd.**Kmd**(*completekey='TAB'*, *stdin=None*, *stdout=None*, *stderr=None*)

Interpreter base class.

This is a subclass of the standard library's `cmd.Cmd` class, using the new `rl` bindings for GNU Readline. The standard library documentation applies unless noted otherwise. Changes include:

1. The *Kmd* constructor accepts an additional `stderr` argument.
2. *preloop()* and *postloop()* are not stubs but contain important code bits. Subclasses must make sure to call their parents' implementations.
3. New methods: *input()*, *word_break_hook()*, *comment()*, *help()*, and *run()*.
4. Incomplete command names are automatically expanded if they are unique.
5. Command aliases can be defined by extending the `aliases` dictionary.
6. `help_*()` methods optionally receive the help topic as argument.
7. `complete_*()` methods may return any kind of iterable, not just lists.

Example:

```
import kmd

class MyShell(kmd.Kmd):
    prompt = 'myshell> '

    def do_quit(self, args):
        return True

MyShell().run()
```

Kmd.**alias_header = 'Command aliases (type help <topic>):'**

Header for the aliases section of the default help screen. If set to the empty string, the aliases section is omitted.

Kmd.**shell_escape_chars = '!'**

Special, single-character aliases for `do_shell()`.

Kmd.**history_file = ''**

If a history filename is set, Kmd loads and saves the history in *preloop()* and *postloop()*.

Kmd.**history_max_entries = -1**

A non-negative value limits the history size.

Kmd.**cmdloop**(*intro=None*)

Repeatedly issue a prompt, accept input, parse an initial prefix off the received input, and dispatch to action methods, passing them the remainder of the line as argument.

Kmd.**preloop**()

Called when the *cmdloop()* method is entered. Configures the readline completer and loads the history file.

Kmd.**postloop**()

Called when the *cmdloop()* method is exited. Resets the readline completer and saves the history file. Note that *postloop()* is called even if *cmdloop()* exits with an exception!

Kmd.**input**(*prompt*)

Read a line from the keyboard using `input()` (or `raw_input()` in Python 2). When the user presses the TAB key, invoke the readline completer.

Kmd.**word_break_hook**(*begidx*, *endidx*)

When completing `?<topic>` make sure `?` is a word break character.

Ditto for `!<command>` and `!`. Installed as `rl.completer.word_break_hook`.

Kmd.**complete**(*text*, *state*)

Return the next possible completion for `text`.

If a command has not been entered, complete against the command list. Otherwise try to call `complete_<command>()` to get a list of completions. Installed as `rl.completer.completer`.

Kmd.**onecmd**(*line*)

Interpret a command line.

This may be overridden, but should not normally need to be; see the `precmd()` and `postcmd()` methods for useful execution hooks. The return value is a flag indicating whether interpretation of commands by the interpreter should stop.

If there is a `do_<command>()` method for the command prefix, that method is called, with the remainder of the line as argument, and its return value is returned. Otherwise the return value of the *default()* method is returned.

Kmd.**parseline**(*line*)

Parse the line into a command name and a string containing the arguments. Returns a tuple containing (command, args, line); command and args may be None if the line could not be parsed.

Kmd.**emptyline**()

Called when the input line is empty. By default repeats the `lastcmd`.

Kmd.**comment**(*line*)

Called when the input line starts with a `#`. By default clears the `lastcmd`.

Kmd.**default**(*line*)

Called when the command prefix is not recognized. By default prints an error message.

Kmd.**do_help**(*topic=''*)

Print the help screen for `topic`.

If there is a `help_<topic>()` method, that method is called, with the unexpanded topic as argument. Otherwise, and if `topic` is a command, the docstring of the `do_<command>()` method is printed to stdout. If `topic` is empty the *help()* method is invoked.

Kmd.**help**()

Print the default help screen. Empty sections and sections with empty headers are omitted.

Kmd.**run**(*args=None*)

Run the Kmd.

If `args` is None it defaults to `sys.argv[1:]`. If arguments are present they are executed via *onecmd()*. Without arguments, enters the *cmdloop()*.

CHAPTER

THREE

COMPLETIONS

A set of completions for use with kmd.Kmd.

3.1 Completion Protocol

A *custom completion* is a class that implements at least two methods:

`Completion.__init__()`

Initializes the completion and configures the readline completer for the type of completion instantiated. May accept additional arguments.

`Completion.__call__(text)`

Returns an iterable of matches for `text`.

3.2 Filename Completion

class `kmd.completions.FilenameCompletion(quote_char='\\')`

Complete file and directory names. The `quote_char` argument specifies the preferred quoting style. Available styles are single-quote, double-quote, and backslash (the default).

To ensure proper configuration of readline, *FilenameCompletion* should always be instantiated before other completions.

`FilenameCompletion.__call__(text)`

Return filenames matching `text`. Starts at the current working directory.

`FilenameCompletion.char_is_quoted(text, index)`

Return True if the character at `index` is quoted. Installed as `rl.completer.char_is_quoted_function`.

`FilenameCompletion.quote_filename(text, single_match, quote_char)`

Return a quoted version of `text`. Installed as `rl.completer.filename_quoting_function`.

`FilenameCompletion.dequote_filename(text, quote_char)`

Return a dequoted version of `text`. Installed as `rl.completer.filename_dequoting_function`.

`FilenameCompletion.rewrite_dirname(text)`

Convert a directory name the user typed to a format suitable for passing to `opendir()`. Installed as `rl.completer.directory_rewrite_hook`.

FilenameCompletion.**rewrite_filename**(*text*)

Convert a filename read from the filesystem to a format suitable for comparing against the completion word. Installed as `rl.completer.filename_rewrite_hook`.

FilenameCompletion.**stat_filename**(*text*)

Convert a filename the user typed to a format suitable for passing to `stat()`. Installed as `rl.completer.filename_stat_hook`.

3.3 Username Completion

class kmd.completions.**UsernameCompletion**

Complete user names.

UsernameCompletion.**__call__**(*text*)

Return user names matching `text`.

User names are returned without decoration. The search string may start with a ~ character, in which case the users' home directories are returned instead. Home directories start with a ~ and end with a / character.

3.4 Hostname Completion

class kmd.completions.**HostnameCompletion**(*hostsfile='/etc/hosts'*)

Complete host names found in the system's hosts file.

HostnameCompletion.**__call__**(*text*)

Return host names matching `text`.

Host names are returned with a leading `@` character. The search string may start with an `@` character which is stripped before matching.

3.5 Environment Completion

class kmd.completions.**EnvironmentCompletion**

Complete names of variables in the process environment.

EnvironmentCompletion.**__call__**(*text*)

Return environment variables matching `text`.

Variable names are returned with a leading $ character. The search string may start with a $ character which is stripped before matching.

3.6 Command Completion

class kmd.completions.**CommandCompletion**

Complete names of commands on the system PATH.

CommandCompletion.**__call__**(*text*)

Return executables matching `text`. Does not include shell built-ins or aliases.

CHAPTER

FOUR

QUOTING

String and filename quoting support.

4.1 Constants

kmd.quoting.**QUOTE_CHARACTERS = '"\''**

These characters may be used in pairs to quote substrings of the line.

kmd.quoting.**WORD_BREAK_CHARACTERS = ' \t\n"\'><;|&=(:'**

These characters define word boundaries.

kmd.quoting.**FILENAME_QUOTE_CHARACTERS = '\\ \t\n"\'@><;|&=()#$`?*[!:{'**

These characters are quoted when they occur in filenames.

kmd.quoting.**SLASHIFY_IN_QUOTES = '\\"$`\n'**

These characters are backslash-quoted even between double quotes.

4.1.1 Extracted from Bash

kmd.quoting.**BASH_QUOTE_CHARACTERS = '\'"'**

Quote characters used by Bash.

kmd.quoting.**BASH_COMPLETER_WORD_BREAK_CHARACTERS = ' \t\n"\'@><;|&=(:'**

Word break characters used by Bash.

kmd.quoting.**BASH_NOHOSTNAME_WORD_BREAK_CHARACTERS = ' \t\n"\'><;|&=(:'**

Word break characters used by Bash when hostname completion is off.

kmd.quoting.**BASH_FILENAME_QUOTE_CHARACTERS = '\\ \t\n"\'@><;|&=()#$`?*[!:{~'**

Filename quote characters used by Bash.

kmd.quoting.**BASH_COMMAND_SEPARATORS = ';|&{(`'**

Command separators used by Bash.

kmd.quoting.**BASH_WHITESPACE_CHARACTERS = ' \t\n'**

Whitespace characters used by Bash.

kmd.quoting.**BASH_SLASHIFY_IN_QUOTES = '\\"$`\n'**

Slashify characters used by Bash.

4.2 Functions

kmd.quoting.**backslash_quote**(*text*, *chars=''*)

Backslash-quote all `rl.completer.filename_quote_characters` in `text`. If `chars` is given, only characters in `chars` are quoted.

kmd.quoting.**backslash_dequote**(*text*, *chars=''*)

Backslash-dequote all `rl.completer.filename_quote_characters` in `text`. If `chars` is given, only characters in `chars` are dequoted.

kmd.quoting.**is_fully_quoted**(*text*, *chars=''*)

Return True if all `rl.completer.filename_quote_characters` in `text` are backslash-quoted. If `chars` is given, only characters in `chars` are checked.

kmd.quoting.**char_is_quoted**(*text*, *index*)

Return True if the character at `index` is quoted.

kmd.quoting.**quote_string**(*text*, *single_match=True*, *quote_char=''*)

Return a `quote_char`-quoted version of `text`. If `single_match` is False, the quotes are not closed. The default `quote_char` is the first character in `rl.completer.quote_characters`.

kmd.quoting.**backslash_quote_string**(*text*, *single_match=True*, *quote_char=''*)

Return a backslash-quoted version of `text`. If a `quote_char` is given, behave like *quote_string()*.

kmd.quoting.**backslash_dequote_string**(*text*, *quote_char=''*)

Return a backslash-dequoted version of `text`. If `quote_char` is the single-quote, backslash-dequoting is limited to single-quotes.

kmd.quoting.**quote_filename**(*text*, *single_match=True*, *quote_char=''*)

Return a `quote_char`-quoted version of `text`. If `single_match` is False or `text` is a directory, the quotes are not closed. The default `quote_char` is the first character in `rl.completer.quote_characters`.

kmd.quoting.**backslash_quote_filename**(*text*, *single_match=True*, *quote_char=''*)

Return a backslash-quoted version of `text`. If a `quote_char` is given, behave like *quote_filename()*.

kmd.quoting.**backslash_dequote_filename**(*text*, *quote_char=''*)

Return a backslash-dequoted version of `text`. If `quote_char` is the single-quote, backslash-dequoting is limited to single-quotes.

CHAPTER

FIVE

EXAMPLES

Example code.

5.1 Simple App

A bare-bones application demonstrating filename and environment variable completion. You can run this example with `python -m kmd.examples.myshell`.

```
# TAB-complete command names, help topics, filenames, and environment variables

import os
import kmd

from kmd.completions import FilenameCompletion
from kmd.completions import EnvironmentCompletion


class MyShell(kmd.Kmd):

    intro = 'myshell 1.0 (type help for help)\n'
    prompt = 'myshell> '

    def preloop(self):
        super().preloop()
        self.completefilename = FilenameCompletion()
        self.completeenviron = EnvironmentCompletion()

    def do_cat(self, args):
        """Usage: cat <filename>"""
        os.system('cat ' + args)

    def do_echo(self, args):
        """Usage: echo $<varname>"""
        os.system('echo ' + args)

    def do_quit(self, args):
        """Usage: quit"""
        return True

    def do_EOF(self, args):
```

(continues on next page)

(continued from previous page)

```
        return True

    def complete_cat(self, text, *ignored):
        return self.completefilename(text)

    def complete_echo(self, text, *ignored):
        return self.completeenviron(text)

    def help_help(self):
        self.stdout.write('Usage: help [<topic>]\n')

    def emptyline(self):
        pass


def main():
    return MyShell().run()


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

5.2 Custom Completion

An implementation of environment variable completion.

```
# Environment variable completion

import os

from rl import completer


class EnvironmentCompletion(object):

    def __init__(self):
        """Configure the readline completer
        """
        if '$' not in completer.word_break_characters:
            completer.word_break_characters += '$'
        if '$' not in completer.special_prefixes:
            completer.special_prefixes += '$'

    def __call__(self, text):
        """Return environment variables matching 'text'

        Variable names are returned with a leading '$' character.
        The search string may start with a '$' character which is
        stripped before matching.
        """
```

(continues on next page)

(continued from previous page)

```
        if text.startswith('$'):
            text = text[1:]
        return ['$'+x for x in os.environ if x.startswith(text)]
```

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

k

INDEX

Symbols

A

B

C

D

E

F

H

I

K

M

O

P

Q

R

S

U

W