24.1. cmd — Support for line-oriented command interpreters¶

The Cmd class provides a simple framework for writing line-oriented command interpreters. These are often useful for test harnesses, administrative tools, and prototypes that will later be wrapped in a more sophisticated interface.

class `cmd.Cmd`([*completekey*[, *stdin*[, *stdout*]]])¶

A Cmd instance or subclass instance is a line-oriented interpreter framework. There is no good reason to instantiate Cmd itself; rather, it's useful as a superclass of an interpreter class you define yourself in order to inherit Cmd's methods and encapsulate action methods.

The optional argument *completekey* is the readline name of a completion key; it defaults to `Tab`. If *completekey* is not None and readline is available, command completion is done automatically.

The optional arguments *stdin* and *stdout* specify the input and output file objects that the Cmd instance or subclass instance will use for input and output. If not specified, they will default to sys.stdin and sys.stdout.

If you want a given *stdin* to be used, make sure to set the instance's use_rawinput attribute to `False`, otherwise *stdin* will be ignored.

Changed in version 2.3: The *stdin* and *stdout* parameters were added.

24.1.1. Cmd Objects¶

A Cmd instance has the following methods:

`Cmd.cmdloop`([*intro*])¶

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.

The optional argument is a banner or intro string to be issued before the first prompt (this overrides the intro class member).

If the readline module is loaded, input will automatically inherit **bash**-like history-list editing (e.g. `Control-P` scrolls back to the last command, `Control-N` forward to the next one, `Control-F` moves the cursor to the right non-destructively, `Control-B` moves the cursor to the left non-destructively, etc.).

An end-of-file on input is passed back as the string `'EOF'`.

An interpreter instance will recognize a command name `foo` if and only if it has a method `do_foo()`. As a special case, a line beginning with the character `'?'` is dispatched to the method `do_help()`. As another special case, a line beginning with the character `'!'` is dispatched to the method `do_shell()` (if such a method is defined).

This method will return when the `postcmd()` method returns a true value. The *stop* argument to `postcmd()` is the return value from the command's corresponding `do_*()` method.

If completion is enabled, completing commands will be done automatically, and completing of commands args is done by calling `complete_foo()` with arguments *text*, *line*, *begidx*, and *endidx*. *text* is the string prefix we are attempting to match: all returned matches must begin with it. *line* is the current input line with leading whitespace removed, *begidx* and *endidx* are the beginning and ending indexes of the prefix text, which could be used to provide different completion depending upon which position the argument is in.

All subclasses of Cmd inherit a predefined `do_help()`. This method, called with an argument `'bar'`, invokes the corresponding method `help_bar()`. With no argument, `do_help()` lists all available help topics (that is, all commands with corresponding `help_*()` methods), and also lists any undocumented commands.

`Cmd.onecmd`(*str*)¶
Interpret the argument as though it had been typed in response to the prompt. 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_*()` method for the command *str*, the return value of that method is returned, otherwise the return value from the `default()` method is returned.

`Cmd.emptyline`()¶
Method called when an empty line is entered in response to the prompt. If this method is not overridden, it repeats the last nonempty command entered.

`Cmd.default`(*line*)¶

Method called on an input line when the command prefix is not recognized. If this method is not overridden, it prints an error message and returns.

`Cmd.completedefault`(*text*, *line*, *begidx*, *endidx*)¶
Method called to complete an input line when no command-specific `complete_*()` method is available. By default, it returns an empty list.

`Cmd.precmd`(*line*)¶
Hook method executed just before the command line *line* is interpreted, but after the input prompt is generated and issued. This method is a stub in `Cmd`; it exists to be overridden by subclasses. The return value is used as the command which will be executed by the `onecmd()` method; the `precmd()` implementation may re-write the command or simply return *line* unchanged.

`Cmd.postcmd`(*stop*, *line*)¶
Hook method executed just after a command dispatch is finished. This method is a stub in `Cmd`; it exists to be overridden by subclasses. *line* is the command line which was executed, and *stop* is a flag which indicates whether execution will be terminated after the call to `postcmd()`; this will be the return value of the `onecmd()` method. The return value of this method will be used as the new value for the internal flag which corresponds to *stop*; returning false will cause interpretation to continue.

`Cmd.preloop`()¶
Hook method executed once when `cmdloop()` is called. This method is a stub in `Cmd`; it exists to be overridden by subclasses.

`Cmd.postloop`()¶
Hook method executed once when `cmdloop()` is about to return. This method is a stub in `Cmd`; it exists to be overridden by subclasses.

Instances of `Cmd` subclasses have some public instance variables:

`Cmd.prompt`¶
The prompt issued to solicit input.

`Cmd.identchars`¶
The string of characters accepted for the command prefix.

`Cmd.lastcmd`¶
The last nonempty command prefix seen.

`Cmd.intro`¶
A string to issue as an intro or banner. May be overridden by giving the `cmdloop()` method an argument.

`Cmd.doc_header`¶
The header to issue if the help output has a section for documented commands.

`Cmd.misc_header`¶
The header to issue if the help output has a section for miscellaneous help topics (that is, there are `help_*()` methods without corresponding `do_*()` methods).

`Cmd.undoc_header`¶
The header to issue if the help output has a section for undocumented commands (that is, there are `do_*()` methods without corresponding `help_*()` methods).

`Cmd.ruler`¶
The character used to draw separator lines under the help-message headers. If empty, no ruler line is drawn. It defaults to `'='`.

`Cmd.use_rawinput`¶
A flag, defaulting to true. If true, `cmdloop()` uses `raw_input()` to display a prompt and read the next command; if false, `sys.stdout.write()` and `sys.stdin.readline()` are used. (This means that by importing `readline`, on systems that support it, the interpreter will automatically support **Emacs**-like line editing and command-history keystrokes.)

Previous topic

24. Program Frameworks

Next topic

24.2. shlex — Simple lexical analysis

This Page

- Show Source

Navigation

- index
- modules |
- next |
- previous |