clik-shell

Release 0.90.1

Joe Joyce and contributors

May 23, 2019

Contents

clik-shell is a tiny glue library between clik[1] and cmd[2]:

```
from clik import app
from clik_shell import DefaultShell


@app
def myapp():
    yield


# ... subcommands for myapp ...


@myapp
def shell():
    yield
    DefaultShell(myapp).cmdloop()
```

See *the quickstart* (page 3) for more documentation on what clik-shell can do.

[1] https://clik.readthedocs.io
[2] https://docs.python.org/3/library/cmd.html

CHAPTER 1

Quickstart

clik-shell makes it easy to add an interactive command shell to your clik[3] application.

1.1 Example Program

Here's the program we'll be working with:

```
from clik import app

@app
def myapp():
    """Example application for clik-shell."""
    yield
    print('myapp')

@myapp
def foo():
    """Print foo."""
    yield
    print('foo')

@myapp
def bar():
    """Print bar."""
    yield
    print('bar')

@myapp
def baz():
    """A subcommand with subcommands."""
    yield
```

(continues on next page)

[3] https://clik.readthedocs.io

(continued from previous page)

```
    print('baz')

@baz
def spam():
    """Print spam."""
    yield
    print('spam')

@baz
def ham():
    """Print ham."""
    yield
    print('ham')

@baz
def eggs():
    """Print eggs."""
    yield
    print('eggs')

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

1.2 Add Shell Subcommand

Add a new subcommand that makes use of *clik_shell.DefaultShell* (page 10):

```
from clik_shell import DefaultShell

@myapp
def shell():
    """Interactive command shell for my application."""
    yield
    DefaultShell(myapp).cmdloop()
```

That's it! The example application now has an interactive command shell:

```
$ ./example.py shell
myapp
myapp> help

Documented commands (type help <topic>):
========================================
EOF  bar  baz  exit  foo  help  quit  shell

myapp> help foo
usage: foo [-h]

Print foo.

optional arguments:
  -h, --help  show this help message and exit

myapp> help baz
```

(continues on next page)

(continued from previous page)

```
usage: baz [-h] {spam,ham,eggs} ...

A subcommand with subcommands.

optional arguments:
  -h, --help       show this help message and exit

subcommands:
  {spam,ham,eggs}
    spam           Print spam.
    ham            Print ham.
    eggs           Print eggs.

myapp> foo
foo
myapp> baz
usage: baz [-h] {spam,ham,eggs} ...
baz: error: the following arguments are required: {spam,ham,eggs}

myapp> qux
error: unregonized command: qux (enter ? for help)

myapp> baz spam
baz
spam
myapp> exit

$
```

1.3 Intended Usage

In practice, the base shell is designed to be subclassed:

```
class Shell(DefaultShell):
    def __init__(self):
        super(Shell, self).__init__(myapp)

@myapp
def shell():
    """Interactive command shell for my application."""
    yield
    Shell().cmdloop()
```

DefaultShell (page 10) is a subclass of Cmd[4], so subclasses of DefaultShell (page 10) can make use of everything in Cmd[5]. This is useful for things like customizing the prompt and adding introductory text:

```
class Shell(DefaultShell):
    intro = 'Welcome to the myapp shell. Enter ? for a list of commands.\n\n'
    prompt = '(myapp)% '
```

With those updates:

[4] https://docs.python.org/3/library/cmd.html#cmd.Cmd
[5] https://docs.python.org/3/library/cmd.html#cmd.Cmd

```
$ ./example.py shell
myapp
Welcome to the myapp shell. Enter ? for a list of commands.


(myapp)%
```

1.4 Excluding Commands from the Shell

As implemented, the `shell` command is available from within the shell:

```
$ ./example.py shell
myapp
myapp> ?

Documented commands (type help <topic>):
========================================
EOF  bar  baz  exit  foo  help  quit  shell

myapp> shell
myapp> exit

myapp> exit

$
```

This works, but isn't the desired behavior. There's no reason for users to start a "subshell." For this case, *clik_shell.exclude_from_shell()* (page 9) is available:

```
from clik_shell import DefaultShell, exclude_from_shell

@exclude_from_shell
@myapp
def shell():
    """Interactive command shell for my application."""
    yield
    Shell().cmdloop()
```

Now users cannot call `shell` from within the shell:

```
$ ./example.py shell
myapp
myapp> ?

Documented commands (type help <topic>):
========================================
EOF  bar  baz  exit  foo  help  quit

myapp> shell
error: unregonized command: shell (enter ? for help)

myapp> exit

$
```

Note that *exclude_from_shell* (page 9) is not limited to the shell command itself – it may be used on any subcommand to exclude that subcommand from the shell interface.

1.5 Shell-Only Commands

To create a command that is available only in the shell, define a new `do_*` method as outlined in the `cmd`[6] documentation:

```
import subprocess

class Shell(DefaultShell):
    def do_clear(self, _):
        """Clear the terminal screen."""
        yield
        subprocess.call('clear')
```

1.6 Base Shell Classes

DefaultShell (page 10) adds a few commonly desired facilities to the default command loop:

- `exit` and `quit` commands to exit the shell
- `EOF` handler, which exits the shell on `Ctl-D`
- `KeyboardInterrupt` handler, which exits the shell on `Ctl-C`
- `cmd.Cmd.emptyline()`[7] override to a no-op (by default it runs the last command entered)

If you want to implement these facilities yourself, subclass *clik_shell.BaseShell* (page 9) instead of the default shell. The base shell defines only three methods on top of `cmd.Cmd`[8]:

- *__init__* (page 9), which dynamically generates the `do_*` and `help_*` methods
- *default* (page 9), which overrides the default `cmd.Cmd.default()`[9] implementation in order to hack in support for hyphenated command names (see below)
- *error* (page 10), which is called when a command exits with a non-zero code

1.7 Hyphenated Commands

`cmd`[10] does not natively support commands with hyphenated names – commands are defined by creating a `do_*` method and methods may not have hyphens in them. Due to this constraint, there's not much clik-shell can do but work around it as best as possible:

- For the purpose of defining methods, all hyphens are converted to underscores – so `my-subcommand` becomes `my_subcommand`
- A hook is added to `cmd.Cmd.default()`[11] to recognize `my-subcommand` and redirect it to `my_subcommand`

[6] https://docs.python.org/3/library/cmd.html#module-cmd
[7] https://docs.python.org/3/library/cmd.html#cmd.Cmd.emptyline
[8] https://docs.python.org/3/library/cmd.html#cmd.Cmd
[9] https://docs.python.org/3/library/cmd.html#cmd.Cmd.default
[10] https://docs.python.org/3/library/cmd.html#module-cmd
[11] https://docs.python.org/3/library/cmd.html#cmd.Cmd.default

Le sigh. This sucks because:

- The underscore names aren't the "real" command names
- The hyphen names don't show up in the help documentation
- In theory someone could define `my-subcommand` **and** `my_subcommand`, which totally breaks this scheme (in practice, anyone who designs a CLI where those two commands do different things deserves to have their app broken)

But, I mean, at least `my-subcommand` doesn't bail out. And that's the *only* reason the workaround was implemented. Otherwise it's a pretty ugly wart on an otherwise reasonably-designed API.

CHAPTER 2

API

`clik_shell.`**`exclude_from_shell`**`(`*command_or_fn*`)`

Exclude command from the shell interface.

This decorator can be applied before or after the command decorator:

```
@exclude_from_shell
@myapp
def mycommand():

# is the same as

@myapp
@exclude_from_shell
def mycommand():
```

Parameters **`command_or_fn`** (`clik.command.Command`[12] or function) – Command instance or function

Returns Whatever was passed in

`class` `clik_shell.`**`BaseShell`**`(`*command*`)`

Bases: `cmd.Cmd`[13]

Minimal implementation to integrate clik and cmd.

`__init__``(`*command*`)`

Instantiate the command loop.

Parameters **`command`** (`clik.command.Command`[14]) – "Root" command object (usually the application object created by `clik.app.app()`[15])

[12] https://clik.readthedocs.io/en/0.92.4/development/internals.html#clik.command.Command
[13] https://docs.python.org/3/library/cmd.html#cmd.Cmd
[14] https://clik.readthedocs.io/en/0.92.4/development/internals.html#clik.command.Command
[15] https://clik.readthedocs.io/en/0.92.4/development/internals.html#clik.app.app

default(*line*)

Override that hackily supports commands with hyphens.

See the quickstart in the documentation for further explanation.

Parameters **line** (`str`[16]) – Line whose command is unrecognized

Return type `None`

error(*exit_code*)

Handle non-zero subcommand exit code.

By default, this prints a generic error message letting the user know the exit code.

Parameters **exit_code** (`int`[17]) – Exit code from the subcommand

Return type `None`

prompt = None

Prompt for the command loop. If `None`, the prompt is set to `"name> "`, where `name` is the name of the root command object.

Type `str`[18] or `None`

class clik_shell.**DefaultShell**(*command*)

Bases: *clik_shell.BaseShell* (page 9)

Command loop subclass that implements commonly desire facilities.

cmdloop()

Override that supports graceful handling of keyboard interrupts.

do_EOF(_)

Exit the shell.

do_exit(_)

Exit the shell.

do_quit(_)

Exit the shell.

emptyline()

Override that turns an empty line into a no-op.

By default, the command loop runs the previous command when an empty line is received. This is bad default behavior because it's not what users expect.

If "run the last command" is the desired behavior, you should extend `BaseClass` rather than this class.

[16] https://docs.python.org/3/library/stdtypes.html#str
[17] https://docs.python.org/3/library/functions.html#int
[18] https://docs.python.org/3/library/stdtypes.html#str

CHAPTER 3

Internals

Clik extension for adding an interactive command shell to an application.

author Joe Joyce <joe@decafjoe.com>

copyright Copyright (c) Joe Joyce and contributors, 2017-2019.

license BSD

`clik_shell.`**`EXCLUDE = <object object>`**

Unique object used to indicate that a command should not be present in the shell.

Type `object`

`clik_shell.`**`get_shell_subcommands_for`**`(`*parent_command*`)`

Return list of command objects that should be present in the shell.

This excludes the commands that have been marked with *`exclude_from_shell()`* (page 9).

Parameters **`command`** (`clik.command.Command`[19]) – Command for which to get shell subcommands

Returns List of commands that should be present in the shell

Return type `list`[20] of `clik.command.Command`[21] instances

`clik_shell.`**`parser_for`**`(`**args*, ***kwds*`)`

Context manager that creates a root parser object for `command`.

See *`make_action_method()`* (page 11) and *`make_help_method()`* (page 12) for usage.

Parameters **`command`** (`clik.command.Command`[22]) – Command for which to create a parser

Returns Argument parser for the command

Return type `argparse.ArgumentParser`[23]

[19] https://clik.readthedocs.io/en/0.92.4/development/internals.html#clik.command.Command
[20] https://docs.python.org/3/library/stdtypes.html#list
[21] https://clik.readthedocs.io/en/0.92.4/development/internals.html#clik.command.Command
[22] https://clik.readthedocs.io/en/0.92.4/development/internals.html#clik.command.Command
[23] https://docs.python.org/3/library/argparse.html#argparse.ArgumentParser

`clik_shell.`**`make_action_method`**`(`*command*`)`

Dynamically generate the `do_` method for `command`.

Parameters **`command`** (`clik.command.Command`[24]) – Command for which to generate `do_` method

Returns Method that calls the given command

Return type `fn(self, line)`

`clik_shell.`**`make_help_method`**`(`*command*`)`

Dynamically generate the `help_` method for `command`.

Parameters **`command`** (`clik.command.Command`[25]) – Command for which to generate `help_` method

Returns Method that prints the help for the given command

Return type `fn(self)`

[24] https://clik.readthedocs.io/en/0.92.4/development/internals.html#clik.command.Command

[25] https://clik.readthedocs.io/en/0.92.4/development/internals.html#clik.command.Command

CHAPTER 4

Changelog

4.1 0.90.0 – 2017-10-22

- Initial public release.

Python Module Index

Index