click-shell Documentation

Release 1.1.dev20200217043951

Clark Perkins

Feb 17, 2020

Contents

click-shell is an extension to click that easily turns your click app into a shell utility. It is built on top of the built in python cmd module, with modifications to make it work with click.

click-shell is compatible with python versions 2.6, 2.7, 3.3, 3.4, and 3.5.

CHAPTER 1

Features

- Adds a "shell" mode **with command completion** to any click app
- Just a one line change for most click apps

Note: It should be noted that click-shell **only** alters functionality if no arguments are passed on the command line. Previously if no arguments were passed, the help was displayed.

1.1 Installation

The easiest way to install is with pip:

```
pip install click-shell
```

If you'd rather, you can clone the github repo and install manually:

```
git clone https://github.com/clarkperkins/click-shell.git
python setup.py install
```

1.2 Usage

There are 2 main ways to utilize click-shell: the decorator and the factory method.

1.2.1 Decorator

The easiest way to get going with click-shell is with the click style decorator. `@click_shell.shell` is meant to replace click's `@click.group` decorator for the root level of your app. In fact, the object generated by `@click_shell.shell` is a `click_shell.core.Shell` object, which is a subclass of `click.core.Group`.

```
from click_shell import shell

# @click.group()  # no longer
@shell(prompt='my-app > ', intro='Starting my app...')
def my_app():
    pass

@my_app.command()
def the_command():
    print 'the_command is running'
```

When run with the above arguments, you should expect an output like so:

```
$ python my_app.py
Starting my app...
my-app >
```

`@shell` takes 3 arguments:

- `prompt` - this will get printed as the beginning of each line in the shell Defaults to `'(Cmd) '`
- `intro` - this will get printed once when the shell first starts Defaults to `None`, meaning nothing gets printed
- `hist_file` - this is the location of the history file used by the shell. Defaults to `'~/.click-history'`

`@shell` also takes arbitrary keyword arguments, and they are passed on directly to the constructor for the *click_shell.Shell‘* class.

1.2.2 Factory Method

If you’d rather not use decorators (or can’t for some reason), you can manually create a shell object and start it up:

```
import click
from click_shell import make_click_shell

@click.group()
@click.pass_context
def my_app(ctx):
    pass



# Somewhere else in your code (as long as you have access to the root level Context␣
↪object)

shell = make_click_shell(ctx, prompt='my-app > ', intro='Starting my app...')
shell.cmdloop()
```

The first argument passed to `make_click_shell` must be the root level context object for your click application. The other 3 args (prompt, intro, hist_file) are the same as described above under the Decorator section.

1.3 Troubleshooting

1.3.1 Autocomplete

If autocomplete isn't working after installation, you may be missing the `readline` module. Try one of the following depending on your platform:

For macOS / linux (the `readline` extra):

```
pip install click-shell[readline]
```

For Windows / cygwin (the `windows` extra):

```
pip install click-shell[windows]
```