Watson - Console

Release 2.0.0

September 30, 2014

Contents

Create console commands with ease.

CHAPTER 1

Build Status

CHAPTER 2

Installation

```
pip install watson-console
```

CHAPTER 3

Testing

Watson can be tested with py.test. Simply activate your virtualenv and run `python setup.py test`.

CHAPTER 4

Contributing

If you would like to contribute to Watson, please feel free to issue a pull request via Github with the associated tests for your code. Your name will be added to the AUTHORS file under contributors.

CHAPTER 5

Table of Contents

5.1 Usage

Commands in Watson are broken down into three parts, SCRIPT NAMESPACE ARGUMENTS. The script refers to the file that the user needs to call to execute the command (this usually refers to *console.py*). The namespace is the location that the command resides in, allowing console commands to be split into different functional areas. Arguments are what the user can pass to the command to modify it's behaviour.

An example command from watson.framework is *new*, which resides in the *project* namespace. It also contains several arguments, *dir* and *override*. The command itself looks like:

```
console.py project new [--dir DIR] [--override] name app_name
```

5.1.1 The anatomy of a command

```
# can be executed by 'script.py my_namespace method'
from watson.console import command
from watson.console.decorators import arg

class MyNamespace(command.Base):
    help = 'Displayed when script.py my_namespace -h is called'
    name = 'my_namespace'  # if not defined, a snake_case version of the class name will be used

    @arg()
    def method(self):
        """Command specific help, printed when -h is called

        More information about the command.
        """
        print('Run!')

    @arg()
    def another(self, positional):
        """Help...

        Args:
            positional: The positional argument help string
        """
```

5.1.2 Defining arguments

Whenever a command is executed, any arguments that are passed to it will also be passed to the associated method.

The @arg decorator also takes any kwargs that the `add_argument` method from argparse has (see https://docs.python.org/3/library/argparse.html).

Positional

Positional arguments can be defined in two ways, either using the @arg decorator, or by just adding the name as argument to the method.

```
# imports...

class MyCommand(command.Base):
    # help, name etc...

    @arg()
    def method(self, positional):
        """The command help

        Args:
            positional: The positional argument help string
        """

    @arg('positional')
    def another(self, **kwargs)
        """Help

        Args:
            positional: The help string
        """
```

Optional

Optional arguments are also created in the same way that positional arguments are, except that they take an additional optional=True argument. The name of the argument must also be defined in the @arg decorator.

```
# imports...

class MyCommand(command.Base):
    # help, name etc...

    @arg('optional', optional=True)
    def method(self, optional):
        """The command help

        Args:
            optional: The optional argument help string
        """
```

5.1.3 Using the command in your app

Within your application config, simply create a new definition named `commands`. Assuming the above command is within the `myapp.commands` module, the definition would look like this:

```
# within config.py
from watson.console.command import find_commands_in_module
from myapp import commands

commands = find_commands_in_module(commands)
```

5.2 Reference Library

5.2.1 watson.console.colors

watson.console.colors.**fail**(*string*, *terminate=True*)
Wraps a string in the terminal colors for fail.

Example:

```
fail('some text')  # colored text in terminal
```

Parameters

- **string** (*string*) – The string to wrap
- **terminate** (*boolean*) – Whether or not to terminate the color

watson.console.colors.**header**(*string*, *terminate=True*)
Wraps a string in the terminal colors for headers.

Example:

```
header('some text')  # colored text in terminal
```

Parameters

- **string** (*string*) – The string to wrap
- **terminate** (*boolean*) – Whether or not to terminate the color

watson.console.colors.**ok_blue**(*string*, *terminate=True*)
Wraps a string in the terminal colors for ok blue.

Example:

```
ok_blue('some text')  # colored text in terminal
```

Parameters

- **string** (*string*) – The string to wrap
- **terminate** (*boolean*) – Whether or not to terminate the color

watson.console.colors.**ok_green**(*string*, *terminate=True*)
Wraps a string in the terminal colors for ok green.

Example:

```
ok_green('some text')  # colored text in terminal
```

Parameters

- **string** (*string*) – The string to wrap
- **terminate** (*boolean*) – Whether or not to terminate the color

watson.console.colors.**warning**(*string*, *terminate=True*)

Wraps a string in the terminal colors for warning.

Example:

```
warning('some text')  # colored text in terminal
```

Parameters

- **string** (*string*) – The string to wrap
- **terminate** (*boolean*) – Whether or not to terminate the color

5.2.2 watson.console.command

class watson.console.command.**Base**

The base command that outlines the required structure for a console command.

Help is automatically invoked when the *-h* or *–help* option is used or when the command or namespace is not specified.

If a name attribute is not specified on the class then a snake_cased version of the name will be used in its place.

http://docs.python.org/dev/library/argparse.html#the-add-argument-method

Example:

```
# can be executed by `script.py my_namespace command`
class MyNamespace(command.Base, ContainerAware):
    help = 'Top level namespace message'

    @arg()
    def command(self):
        '''Command specific help.
        '''
        print('Run!')

# with arguments from the function
# can be executed by `script.py my_namespace command somevalue
class MyNamespace(command.Base, ContainerAware):
    help = 'Top level namespace message'

    @arg()
    def command(self, value):
        '''Command specific help.

        Args:
            value: A value to pass
        '''
        print('Run', value)

# with options
class MyNamespace(command.Base, ContainerAware):
    help = 'Top level namespace message'

    @arg('value', optional=True)
```

```
    def command(self, value):
        '''Command specific help.
        '''
        print('Run!')
```

watson.console.command.**find_commands_in_module**(*module*)
Retrieves a list of all commands within a module.

Returns A list of commands from the module.

5.2.3 watson.console.runner

exception watson.console.runner.**ConsoleError**
An error that should be raised from within the command.

class watson.console.runner.**Runner**(*commands=None*)
A command line runner that allows new commands to be added and run on demand.

Commands can be added either as a fully qualified name, or imported.

Example:

```
runner = Runner(commands=['module.commands.ACommand'])
runner()
```

__init__(*commands=None*)

add_command(*command*)
Convenience method to add new commands after the runner has been initialized.

Parameters **command** (*string|class*) – the command to add

add_commands(*commands*)
Convenience method to add multiple commands.

Parameters **commands** (*list|tuple*) – the commands to add

attach_commands(*parser*, *namespace*)
Register the commands against the parser.

Parameters

- **parser** – The parser to add commands to
- **namespace** – The namespace the commands should sit within

commands
A list of all commands added to the runner.

Returns OrderedDict containing all the commands.

execute(*args*)
Execute the runner and any commands the user has specified.

get_command(*command_name*)
Returns an initialized command from the attached commands.

Parameters **command_name** – The command name to retrieve

name
Returns the name of the script that runner was executed from.

update_usage(*parser*, *namespace*, *is_subparser=False*)
Updates the usage for the relevant parser.

Forces the usage message to include the namespace and color.

5.2.4 watson.console.styles

watson.console.styles.**bold**(*string*, *terminate=True*)
Bolds text within the terminal.

Example:

Parameters

- **string** (*string*) – The string to wrap
- **terminate** (*boolean*) – Whether or not to terminate the styling

watson.console.styles.**format_style**(*string*, *start*, *end='\x1b[0m'*)
Formats text for usage within the terminal.

Example:

Parameters

- **string** (*string*) – The string to wrap
- **terminate** (*boolean*) – Whether or not to terminate the styling

watson.console.styles.**underline**(*string*, *terminate=True*)
Underlines text within the terminal.

Example:

Parameters

- **string** (*string*) – The string to wrap
- **terminate** (*boolean*) – Whether or not to terminate the styling

W