uboat

Release 0.1.1

unknown

Jul 02, 2019

CONTENTS

1 Installation **3**

2 Supports **5**

3 Docs & Source **7**

4 Usage **9**
- 4.1 Example 1: Sub-command Registration . . . 9
- 4.2 Example 2: Configuration and Command-line Flags . . . 9
- 4.3 Example 3: Sub-command Parameters and Flags . . . 10
- 4.4 Example Source . . . 12

5 Methods **13**

6 Indices and tables **15**

Python Module Index **17**

Index **19**

This library helps you write command-line python scripts to use sub commands (e.g. “git commit”, “git checkout”). It is a thin wrapper around the built-in library argparse.

This was written before the author discovered the vastly superior: click. You should use that instead.

CHAPTER

ONE

INSTALLATION

```
$ pip install uboat
```

CHAPTER

TWO

SUPPORTS

uboat has been tested with Python 3.6 and 3.7

CHAPTER

THREE

DOCS & SOURCE

Docs: http://uboat.readthedocs.io/en/latest/

Source: https://github.com/cltrudeau/uboat

Version: 0.1.1

CHAPTER

FOUR

USAGE

uboat is a thin wrapper for the sub-parser functionality of the `argparse` python library. Creating sub-commands is now as simple as using a decorator.

4.1 Example 1: Sub-command Registration

The simplest case, creating a single sub-command. Put the following in a file named "simple.py"

```
#!/usr/bin/env python

import uboat

# ========================================================================

@uboat.command
def show(args):
    print('This is the "show" sub-command running')

# ========================================================================

if __name__ == '__main__':
    uboat.process_arguments()
```

And execution:

```
$ python example.py show
This is some info

$ pyton example.py
usage: example.py [-h] {show} ...

positional arguments:
  {show}

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

4.2 Example 2: Configuration and Command-line Flags

uboat also supports passing configuration options to the `ArgumentParser` and associated `subparsers` objects, this is typically used to improve the help output. You can also add general flags that go before the sub-command.

Put the following in a file named "flags.py":

```
import uboat
from uboat import flag

# ================================================================

@uboat.command
def greet(args):
    if args.yell:
        print('HELLO!!!!')
    else:
        print('hello')

# ================================================================

if __name__ == '__main__':
    uboat.configure(description='A script that greets you')
    uboat.configure_subparser(title='sub-commands',
            description='valid sub-commands')
    uboat.add_flags(
        flag('--yell', action='store_true',
            help='Makes the greeting louder'),
    )

    uboat.process_arguments()
```

You can add one or more `flag()` objects to the `add_flags` call. The first parameter of the object is the name of the flag, the rest are keyword arguments passed through to the `argparse` `add_argument()` call.

The associated output:

```
$ ./flags.py greet
hello

$ ./flags.py --yell greet
HELOO!!!!!

$ ./flags.py --help
usage: flags.py [-h] [--yell] {hello} ...

A script that greets you

optional arguments:
  -h, --help  show this help message and exit
  --yell      Makes the greeting louder

sub-commands:
  valid sub-commands

  {hello}
```

4.3 Example 3: Sub-command Parameters and Flags

Sub-commands can also support arguments and flags. These are handled by adding information to the registration decorator.

Put the following in a file called "parms.py"

```
import uboat
from uboat import flag

# ================================================================

@uboat.command(
    flag('packages', nargs='+', help='Name of package to add'),
    name='install', help='Pretends to install something')
def install_cmd(args):
    print('Installing:', ','.join(args.packages))
    print('Done')

# ================================================================

if __name__ == '__main__':
    uboat.add_flags(
        flag('--suffix', action='store_true',
            help='Adds a sentence after script output'),
    )

    args = uboat.process_arguments()

    if args.suffix:
        print('\nThis is the suffix line')
```

The above registers a sub-command called "install", notice that the name of the function in this case isn't the same as the command. Passing the "name" parameter to the registration decorator overrides the use of the function name as the sub-command. The registration decorator uses the same `flag()` concept as the `add_flags()` call explained in *Example 2: Configuration and Command-line Flags*. The flags in the decorator are just passed through to the `argparse` `add_argument()` call and so can be either parameters or flags depending on the keyword arguments.

The `process_arguments()` call returns a reference to the `argparse` `Namespace` object and so can be checked after the sub-command has run.

The above example in usage:

```
$ ./parms.py
usage: parms.py [-h] [--suffix] {install} ...

positional arguments:
  {install}
    install   Pretends to install something

optional arguments:
  -h, --help  show this help message and exit
  --suffix    Adds a sentence after script output

$ ./parms.py install
usage: parms.py install [-h] packages [packages ...]
parms.py install: error: the following arguments are required: packages

$ ./parms.py install foo
Installing: foo
Done
```

4.4 Example Source

Source code for the above examples is available in the source code repository:

Source: https://github.com/cltrudeau/uboat/tree/master/extras

CHAPTER

FIVE

METHODS

uboat.**add_flags**(**args*)

Registers command line flags with the CommandManager. Takes one or more *flag()* calls which are wrappers to the argparse add_argument() method.

uboat.**command**(**decorator_args*, ***decorator_kwargs*)

Decorator for registering new sub-commands.

Simplest case is to be called without parameters, registering a sub-command with the name of the function wrapped.

```
@uboat.command
def show(args):
    print('This is the "show" sub-command running')
```

Alternatively, parameters can be passed to the decorator. Using a parameter name will override the method's name as the sub-command. One or more *flag()* methods can be passed in to configure flags or arguments for the sub-command. All parameters of the *flag()* call are passed through to the argparse add_argument() call. Parameters not wrapped in the *flag()* method are passed through to the creation of the subparser.

```
@uboat.command(
    flag('packages', nargs='+', help='List of packages to add'),
    name='install', help='Pretends to install something')
def install_cmd(args):
    print('Installing:', ','.join(args.packages))
    print('Done')
```

The above registers a sub-command called "install" (NB: *not* "install_cmd") which takes a parameter with one or more argument stored in "packages". This is the equivalent of the following argparse code:

```
import argparse

parser = argparse.ArgumentParser()
subparsers = parser.add_subparsers()

install_parser = subparsers.add_parser('install',
    help='Pretends to install something')
install_parser.ad_argument('packages', nargs='+',
    help='List of packages to add')
install_parser.set_defaults(func=install_cmd)

args = parser.parse_args()
args.func(args)
```

uboat.**configure**(***kwargs*)
: Registers configuration parameters for the `ArgumentParser`. All parameters passed to this method are sent through to the `ArgumentParser` constructor.

uboat.**configure_subparser**(***kwargs*)
: Registers configurtion parameters for the subparser created inside the `ArgumentParser`. All parameters passed to this method are sent through to the `add_suparsers()` method on the `ArgumentParser`.

uboat.**flag**(*name*, ***kwargs*)
: Function acts like a wrapper for flag parameters that would be used inside of a `argparse` `add_argument()` call.

uboat.**process_arguments**()
: Method to be called to have the program parse the command line arguments and execute any sub-commands found there.

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

u

uboat, 13

INDEX

A

add_flags() (*in module uboat*), 13

C

command() (*in module uboat*), 13
configure() (*in module uboat*), 13
configure_subparser() (*in module uboat*), 14

F

flag() (*in module uboat*), 14

P

process_arguments() (*in module uboat*), 14

U

uboat (*module*), 13