# wilderness

G.J.J. van den Burg

Apr 09, 2023

# CONTENTS


**1 Installation** **3**

**2 Usage** **5**

**3 Examples** **7**

**4 Notes** **9**
- 4.1 Wilderness . . . 9
- 4.2 Changelog . . . 10
- 4.3 Wilderness API Documentation . . . 11

**5 Indices and tables** **27**

**Python Module Index** **29**

**Index** **31**

Wilderness is a light wrapper around argparse for creating command line applications with multiple subcommands, in the style of Git. Wilderness also makes it easy to generate man pages for your application.

Wilderness is heavily inspired by Cleo and argparse-manpage, but aims to stick as closely as possible to argparse to avoid users having to learn a new API.

CHAPTER

ONE

# INSTALLATION

Wilderness is available on PyPI:

```
$ pip install wilderness
```

CHAPTER

TWO

# USAGE

Building command line applications with Wilderness is straightforward, but it does expect a certain structure of the application. You can create applications with or without subcommands, as illustrated with the fakegit and fakedf examples, respectively.

Creating wilderness applications consist of the following steps:

1. Subclassing the `wilderness.Application` class to hold the main application.
2. Adding one or more `wilderness.Command` objects for each of the subcommands, optionally organized into `wilderness.Group`s.
3. Minor changes to `setup.py` to build the manpages.

CHAPTER

THREE

# EXAMPLES

Here are some examples that use Wilderness to build command line applications:

| Repository | Description |
|---|---|
| fakegit | A multi-level command line application similar to Git |
| fakedf | An application without subcommands similar to df |
| CleverCSV | CleverCSV is a package for handling messy CSV files |
| Veld | Easy command line analytics |

Add your example here by opening a pull request!

CHAPTER

FOUR

# NOTES

License: See the LICENSE file.

Author: Gertjan van den Burg

## 4.1 Wilderness

Wilderness is a light wrapper around argparse for creating command line applications with multiple subcommands, in the style of Git. Wilderness also makes it easy to generate man pages for your application.

Wilderness is heavily inspired by Cleo and argparse-manpage, but aims to stick as closely as possible to argparse to avoid users having to learn a new API.

### 4.1.1 Installation

Wilderness is available on PyPI:

```
$ pip install wilderness
```

### 4.1.2 Usage

Building command line applications with Wilderness is straightforward, but it does expect a certain structure of the application. You can create applications with or without subcommands, as illustrated with the fakegit and fakedf examples, respectively.

Creating wilderness applications consist of the following steps:

1. Subclassing the `wilderness.Application` class to hold the main application.
2. Adding one or more `wilderness.Command` objects for each of the subcommands, optionally organized into `wilderness.Group`s.
3. Minor changes to `setup.py` to build the manpages.

### 4.1.3 Examples

Here are some examples that use Wilderness to build command line applications:

| Repository | Description |
|---|---|
| fakegit | A multi-level command line application similar to Git |
| fakedf | An application without subcommands similar to df |
| CleverCSV | CleverCSV is a package for handling messy CSV files |
| Veld | Easy command line analytics |

Add your example here by opening a pull request!

### 4.1.4 Notes

License: See the LICENSE file.

Author: Gertjan van den Burg

## 4.2 Changelog

### 4.2.1 Version 0.1.9

- Add MANIFEST.in file to package for more complete packaging (thanks to @martin-kokos)

### 4.2.2 Version 0.1.8

- Fix for running unit tests without man-db (GH-5)

### 4.2.3 Version 0.1.7

- Add option to automatically generate list of commands in Application manpage
- Fix support for tab indentation in man pages
- Add helpful errors when description is not of type str
- Some minor fixes

### 4.2.4 Version 0.1.6

- Bugfix for application testing
- Minor fixes for documentation of single-command applications
- Improve test discovery (GH-2)

### 4.2.5 Version 0.1.5

- Bugfix for synopsis of commands with mutually exclusive argument groups

### 4.2.6 Version 0.1.4

- Add support for mutually exclusive argument groups
- Typos and fixes

### 4.2.7 Version 0.1.3

- Add Tester class to test applications and commands
- Subclass ArgumentParser to handle exit on error
- Removed `application.get_argument` method
- Several smaller fixes and design improvements

### 4.2.8 Version 0.1.2

- Redesign manpage building api
- Use application help instead of manpage with `<app> help`

### 4.2.9 Version 0.1.1

- Add py.typed file for PEP 561.
- Update help action for application
- Minor fixes

### 4.2.10 Version 0.1.0

- Initial release

## 4.3 Wilderness API Documentation

### 4.3.1 wilderness package

#### Submodules

#### wilderness.application module

Application class

This module contains the Application class.

Author: G.J.J. van den Burg License: See the LICENSE file. Copyright: 2021, G.J.J. van den Burg

This file is part of Wilderness.

class wilderness.application.**Application**(*name: str*, *version: str*, *author: Optional[str] = None*, *title: Optional[str] = None*, *description: Optional[str] = None*, *default_command: Optional[str] = None*, *add_help: bool = True*, *extra_sections: Optional[Dict[str, str]] = None*, *prolog: Optional[str] = None*, *epilog: Optional[str] = None*, *options_prolog: Optional[str] = None*, *options_epilog: Optional[str] = None*, *add_commands_section: bool = False*)

Bases: *DocumentableMixin*

Base class for applications

This is the main Application object that Wilderness applications are expected to inherit from. All text that is supplied to the man pages, such as the description, can use basic formatting constructs documented in the *ManPage.groffify()* method.

**Parameters**

- **name** (*str*) – The name of the application.
- **version** (*str*) – The version of the application, to be used in creating the man pages.
- **author** (*Optional[str]*) – The author(s) of the application. This is used in the man pages, but is not actually visible in the output (it is recorded in the metadata header of the man pages).
- **title** (*Optional[str]*) – The title of the application is used as a short description. It shows up in the man pages as the text after the application name in the first section.
- **description** (*Optional[str]*) – Long description of the application. This is used in the man pages in the DESCRIPTION section after the synopsis.
- **default_command** (*Optional[str]*) – The default command to run when none is supplied on the command line. By default this is omitted and the help text is shown instead, but some applications may want to run a particular command as default instead.
- **add_help** (*bool*) – Whether to add help commands or not. This adds support for the traditional help flags `-h` or `--help` for the short help text on the command line, as well as the `help` command that opens the man pages for the subcommands of the application. Note that the short help text on the command line typically provides a list of available commands.

  See the FakeDF example for an application where this is not enabled.
- **extra_sections** (*Optional[Dict[str, str]]*) – Additional sections of documentation for the man page. This is expected to be provided as a dictionary where the keys are the section headers and the values are the section text. Basic formatting constructs such as lists and enumerations are understood by the text processor (see *ManPage.groffify()* for further details).
- **prolog** (*Optional[str]*) – Text to be shown in the short command line help text, before the (grouped) list of available commands. Newline characters are preserved.
- **epilog** (*Optional[str]*) – Text to be shown in the short command line help text, after the list of available commands. Newline characters are preserved.
- **options_prolog** (*Optional[str]*) – Text to be shown in the man page before the list of options. See the FakeDF application for an example.
- **options_epilog** (*Optional[str]*) – Text to be shown in the man page after the list of options. See the FakeDF application for an example.
- **add_commands_section** (*bool*) – Whether to automatically generate a section in the application man page that lists the available commands.

**add**(*command:* Command)

Add a command to the application

Note that the `register` method of the command is called when it is added to the application.

**Parameters**
**command** (*wilderness.command.Command*) – The command to add to the application.

**add_argument**(**args*, ***kwargs*) → Action

Add an argument to the application

This wraps the argparse.ArgumentParser.add_argument method, with the minor difference that it supports a "description" keyword argument, which will be used to provide a long help message for the argument in the man page.

**add_group**(*title: str*) → *Group*

Create a group of commands

**Parameters**
**title** (*str*) – The title for the group.

**Returns**
The created command group.

**Return type**
*wilderness.group.Group*

**property author: str**

The author(s) of the application

**property commands: List[*Command*]**

List the commands registered to the application

**Returns**
**commands** – The list of commands registered to the application.

**Return type**
List[*wilderness.command.Command*]

**create_manpage**() → *ManPage*

Create the Manpage for the application

**Returns**
**man_page** – The generated ManPage object.

**Return type**
*wilderness.manpages.ManPage*

**format_help**() → str

Format the command line help for the application

This method creates the help text for the command line, which is typically printed when the -h / –help / help command line arguments are used. The *print_help()* method calls this method to format the help text.

**Returns**
**help_text** – The help text as a single string.

**Return type**
str

**get_command**(*command_name: str*) → *Command*

Get a command by name

**Parameters**
**command_name** (*str*) – The name of the command to find

**Returns**
**command** – The instance of the Command to be returned.

**Return type**
*wilderness.command.Command*

**Raises**
**KeyError** – If no command with the provided name can be found, a KeyError is raised.

**get_commands_text**() → str

**property groups: List[*Group*]**

List the groups registered to the application

If no groups have been added to the application but commands have been added, this property will contain a single group, the root group.

**Returns**
**groups** – The list of groups registered to the application

**Return type**
List[*wilderness.group.Group*]

**handle**() → int

Main method to override for single-command applications.

When creating a single-command application (such as the FakeDF example), this method must be overridden with the actual functionality. For multi-command applications, this method is not used.

**Returns**
**return_code** – The return code of the application, to be used as the return code on the command line.

**Return type**
int

**property name: str**

The name of the application

**print_help**(*file: Optional[TextIO] = None*)

Print the command line help text for the application

**Parameters**
**file** (*Optional[TextIO]*) – The file to which to write the help text. If omitted, the help text will be written to sys.stdout.

**register**()

Register arguments to the application

Override this method to add command line arguments to the application (using self.add_argument, etc). For single-command applications, this should be used to add all command line arguments. For multi-command applications, this method can be used to add arguments that apply to all commands, or arguments such as –version.

This method is called upon initialization of the Application object.

**run**(*args: Optional[List[str]] = None*, *namespace: Optional[Namespace] = None*, *exit_on_error: bool = True*) → int

Main method to run the application

**Parameters**

- **args** (*Optional[List[str]]*) – List of arguments to the application. This is typically only used for testing, as by default the arguments will be read from the command line.
- **namespace** (*Optional[argparse.Namespace]*) – Namespace object to save the arguments to. By default a new argparse.Namespace object is created.
- **exit_on_error** (*bool*) – Whether or not to exit when argparse encounters an error.

**Returns**
**return_code** – The return code of the application, to be used as the return code at the command line.

**Return type**
int

**run_command**(*command: Command*) → int

Run a particular command directy

**Parameters**
**command** (*wilderness.command.Command*) – The command to execute

**Returns**
**return_code** – The return code of the handle() method of the command.

**Return type**
int

**set_epilog**(*epilog: str*) → None

Set the epilog of the command line help text

**Parameters**
**epilog** (*str*) – Text to include at the end of the command line help text.

**set_prolog**(*prolog: str*) → None

Set the prolog of the command line help text

**Parameters**
**prolog** (*str*) – Text to include before the list of commands in the command line help text. The prolog is printed after the synopsis of the application.

**property version:** str

The version of the package or application

### wilderness.argparse_wrappers module

ArgumentParser override

Author: G.J.J. van den Burg License: See the LICENSE file. Copyright: 2021, G.J.J. van den Burg

This file is part of Wilderness.

**class** wilderness.argparse_wrappers.**ArgumentGroup**(*group: _ArgumentGroup*)

Bases: object

**add_argument**(**args*, ***kwargs*)

**property command:** Optional[*wilderness.command.Command*]

class wilderness.argparse_wrappers.**ArgumentParser**(**args*, *exit_on_error=True*, ***kwargs*)

Bases: ArgumentParser

**exit**(*status: Optional[int] = 0*, *message: Optional[str] = None*)

class wilderness.argparse_wrappers.**MutuallyExclusiveGroup**(*meg: _MutuallyExclusiveGroup*)

Bases: object

**add_argument**(**args*, ***kwargs*)

**property command:** Optional[*wilderness.command.Command*]

### wilderness.command module

Command definition

This module contains the definitions for the Command class.

Author: G.J.J. van den Burg License: See the LICENSE file. Copyright: 2021, G.J.J. van den Burg

This file is part of Wilderness.

class wilderness.command.**Command**(*name: str*, *title: Optional[str] = None*, *description: Optional[str] = None*, *add_help: bool = True*, *extra_sections: Optional[Dict[str, str]] = None*, *options_prolog: Optional[str] = None*, *options_epilog: Optional[str] = None*)

Bases: *DocumentableMixin*

**add_argument**(**args*, ***kwargs*)

**add_argument_group**(**args*, ***kwargs*) → *ArgumentGroup*

**add_mutually_exclusive_group**(**args*, ***kwargs*) → *MutuallyExclusiveGroup*

**property application:** Optional[*wilderness.application.Application*]

**create_manpage**() → *ManPage*

**abstract handle**() → int

**property name:** str

**register**()

Register a virtual subclass of an ABC.

Returns the subclass, to allow usage as a class decorator.

**property title:** Optional[str]

### wilderness.documentable module

DocumentableMixin definitions

A documentable is either an application or command, for which we can generate a manpage.

Author: G.J.J. van den Burg License: See the LICENSE file. Copyright: 2021, G.J.J. van den Burg

This file is part of Wilderness.

**class** wilderness.documentable.**DocumentableMixin**(*description: Optional[str] = None*, *extra_sections: Optional[Dict[str, str]] = None*, *options_prolog: Optional[str] = None*, *options_epilog: Optional[str] = None*)

Bases: object

**property args: Namespace**

The parsed command line arguments

**property argument_help: Dict[str, Optional[str]]**

**abstract create_manpage**() → *ManPage*

**property description: Optional[str]**

**get_options_text**() → str

**get_synopsis**(*width: int = 80*) → str

**property parser: ArgumentParser**

### wilderness.formatter module

HelpFormatter

We have a slightly adjusted HelpFormatter that we use for the manpages.

Author: G.J.J. van den Burg License: See the LICENSE file. Copyright: 2021, G.J.J. van den Burg

This file is part of Wilderness.

**class** wilderness.formatter.**HelpFormatter**(*prog*, *indent_increment=2*, *max_help_position=24*, *width=None*)

Bases: HelpFormatter

### wilderness.group module

Group definitions

This module contains the definitions for the Group class, which is used to collect distinct Command objects for the application.

Author: G.J.J. van den Burg License: See the LICENSE file. Copyright: 2021, G.J.J. van den Burg

This file is part of Wilderness.

**class** wilderness.group.**Group**(*title: Optional[str] = None*, *is_root: bool = False*)

Bases: object

**add**(*command:* wilderness.command.Command) → None

**property application:** Optional[*wilderness.application.Application*]

**property commands:** List[*wilderness.command.Command*]

**commands_as_actions**() → List[Action]

**property is_root:** bool

Return whether the groups is its Application's root group

**set_app**(*app:* wilderness.application.Application) → None

**property title:** Optional[str]

### wilderness.help module

Help command definitions

This module contains the definitions for our HelpCommand and our HelpAction. The HelpCommand takes care of opening the manpage when the "help" subcommand is called, and the HelpAction is slightly modified to use our help text formatter (see the Application class).

Author: G.J.J. van den Burg License: See the LICENSE file. Copyright: 2021, G.J.J. van den Burg

This file is part of Wilderness.

**class** wilderness.help.**HelpCommand**

Bases: *Command*

**handle**() → int

**register**()

Register a virtual subclass of an ABC.

Returns the subclass, to allow usage as a class decorator.

wilderness.help.**have_man_command**() → bool

wilderness.help.**help_action_factory**(*app:* wilderness.application.Application)

### wilderness.manpages module

Code to generate manpages

Author: G.J.J. van den Burg License: See the LICENSE file. Copyright: 2021, G.J.J. van den Burg

This file is part of Wilderness.

**class** wilderness.manpages.**ManPage**(*application_name: str*, *author: Optional[str] = ''*, *command_name: Optional[str] = None*, *date: Optional[str] = None*, *title: Optional[str] = None*, *version: Optional[str] = ''*)

Bases: object

**add_section**(*label: str*, *text: str*) → None

**add_section_synopsis**(*synopsis: str*) → None

**export**(*output_dir: str*) → str

**groffify**(*text: str*) → str

Format a text line for use in manpages

This function supports several basic formatting constructs. First, newlines in the text are preserved. Next, lists can be created by starting lines with `*` , as long as each list entry starts on its own line (that is, separated by `n`). Indented text can be created by prefixing a line with one or more `t` characters. Numbered lists are also recognized, as long as each item starts with `1.` (that is, a digit followed by a period, followed by a space).

**Parameters**
**text** (*str*) – The text to convert

**Returns**
**formatted_text** – The formatted text ready for use in manpage documents.

**Return type**
str

**groffify_line**(*line: str*) → str

**header**() → str

**metadata**() → List[str]

**property name: str**

**preamble**() → List[str]

**section_name**() → str

wilderness.manpages.**build_manpages**(*app:* wilderness.application.Application, *output_directory: str* = *'man'*) → None

Write manpages to the output directory

**Parameters**

- **app** (*wilderness.Application*) – The application for which to generate manpages.
- **output_directory** (*str*) – The output directory to which to write the manpages.

### wilderness.tester module

Tester class

This module contains the CommandTester class.

Author: G.J.J. van den Burg License: See the LICENSE file. Copyright: 2021, G.J.J. van den Burg

This file is part of Wilderness.

**class** wilderness.tester.**Tester**(*app:* Application)

Bases: object

**property application: *Application***

**clear**()

**get_return_code**() → Optional[int]

**get_stderr**() → Optional[str]

**get_stdout**() → Optional[str]

**test_application**(*args: Optional[List[str]] = None*) → None

**test_command**(*cmd_name: str*, *args: List[str]*) → None

## Module contents

**class** wilderness.**Application**(*name: str*, *version: str*, *author: Optional[str] = None*, *title: Optional[str] = None*, *description: Optional[str] = None*, *default_command: Optional[str] = None*, *add_help: bool = True*, *extra_sections: Optional[Dict[str, str]] = None*, *prolog: Optional[str] = None*, *epilog: Optional[str] = None*, *options_prolog: Optional[str] = None*, *options_epilog: Optional[str] = None*, *add_commands_section: bool = False*)

Bases: *DocumentableMixin*

Base class for applications

This is the main Application object that Wilderness applications are expected to inherit from. All text that is supplied to the man pages, such as the description, can use basic formatting constructs documented in the *ManPage.groffify()* method.

**Parameters**

- **name** (*str*) – The name of the application.
- **version** (*str*) – The version of the application, to be used in creating the man pages.
- **author** (*Optional[str]*) – The author(s) of the application. This is used in the man pages, but is not actually visible in the output (it is recorded in the metadata header of the man pages).
- **title** (*Optional[str]*) – The title of the application is used as a short description. It shows up in the man pages as the text after the application name in the first section.
- **description** (*Optional[str]*) – Long description of the application. This is used in the man pages in the DESCRIPTION section after the synopsis.
- **default_command** (*Optional[str]*) – The default command to run when none is supplied on the command line. By default this is omitted and the help text is shown instead, but some applications may want to run a particular command as default instead.
- **add_help** (*bool*) – Whether to add help commands or not. This adds support for the traditional help flags `-h` or `--help` for the short help text on the command line, as well as the `help` command that opens the man pages for the subcommands of the application. Note that the short help text on the command line typically provides a list of available commands.

  See the FakeDF example for an application where this is not enabled.
- **extra_sections** (*Optional[Dict[str, str]]*) – Additional sections of documentation for the man page. This is expected to be provided as a dictionary where the keys are the section headers and the values are the section text. Basic formatting constructs such as lists and enumerations are understood by the text processor (see *ManPage.groffify()* for further details).
- **prolog** (*Optional[str]*) – Text to be shown in the short command line help text, before the (grouped) list of available commands. Newline characters are preserved.

- **epilog** (*Optional[str]*) – Text to be shown in the short command line help text, after the list of available commands. Newline characters are preserved.
- **options_prolog** (*Optional[str]*) – Text to be shown in the man page before the list of options. See the FakeDF application for an example.
- **options_epilog** (*Optional[str]*) – Text to be shown in the man page after the list of options. See the FakeDF application for an example.
- **add_commands_section** (*bool*) – Whether to automatically generate a section in the application man page that lists the available commands.

**add**(*command:* Command)

Add a command to the application

Note that the `register` method of the command is called when it is added to the application.

**Parameters**
**command** (*wilderness.command.Command*) – The command to add to the application.

**add_argument**(**args*, ***kwargs*) → Action

Add an argument to the application

This wraps the argparse.ArgumentParser.add_argument method, with the minor difference that it supports a "description" keyword argument, which will be used to provide a long help message for the argument in the man page.

**add_group**(*title: str*) → *Group*

Create a group of commands

**Parameters**
**title** (*str*) – The title for the group.

**Returns**
The created command group.

**Return type**
*wilderness.group.Group*

**property author: str**

The author(s) of the application

**property commands: List[Command]**

List the commands registered to the application

**Returns**
**commands** – The list of commands registered to the application.

**Return type**
List[*wilderness.command.Command*]

**create_manpage**() → *ManPage*

Create the Manpage for the application

**Returns**
**man_page** – The generated ManPage object.

**Return type**
*wilderness.manpages.ManPage*

format_help() → str

Format the command line help for the application

This method creates the help text for the command line, which is typically printed when the -h / –help / help command line arguments are used. The *print_help()* method calls this method to format the help text.

**Returns**
**help_text** – The help text as a single string.

**Return type**
str

get_command(*command_name: str*) → *Command*

Get a command by name

**Parameters**
**command_name** (*str*) – The name of the command to find

**Returns**
**command** – The instance of the Command to be returned.

**Return type**
*wilderness.command.Command*

**Raises**
**KeyError** – If no command with the provided name can be found, a KeyError is raised.

get_commands_text() → str

property groups: List[*Group*]

List the groups registered to the application

If no groups have been added to the application but commands have been added, this property will contain a single group, the root group.

**Returns**
**groups** – The list of groups registered to the application

**Return type**
List[*wilderness.group.Group*]

handle() → int

Main method to override for single-command applications.

When creating a single-command application (such as the FakeDF example), this method must be overridden with the actual functionality. For multi-command applications, this method is not used.

**Returns**
**return_code** – The return code of the application, to be used as the return code on the command line.

**Return type**
int

property name: str

The name of the application

print_help(*file: Optional[TextIO] = None*)

Print the command line help text for the application

**Parameters**
**file** (*Optional[TextIO]*) – The file to which to write the help text. If omitted, the help text will be written to sys.stdout.

**register**()

Register arguments to the application

Override this method to add command line arguments to the application (using self.add_argument, etc). For single-command applications, this should be used to add all command line arguments. For multi-command applications, this method can be used to add arguments that apply to all commands, or arguments such as –version.

This method is called upon initialization of the Application object.

**run**(*args: Optional[List[str]] = None*, *namespace: Optional[Namespace] = None*, *exit_on_error: bool = True*) → int

Main method to run the application

**Parameters**

- **args** (*Optional[List[str]]*) – List of arguments to the application. This is typically only used for testing, as by default the arguments will be read from the command line.
- **namespace** (*Optional[argparse.Namespace]*) – Namespace object to save the arguments to. By default a new argparse.Namespace object is created.
- **exit_on_error** (*bool*) – Whether or not to exit when argparse encounters an error.

**Returns**
**return_code** – The return code of the application, to be used as the return code at the command line.

**Return type**
int

**run_command**(*command: Command*) → int

Run a particular command directy

**Parameters**
**command** (*wilderness.command.Command*) – The command to execute

**Returns**
**return_code** – The return code of the handle() method of the command.

**Return type**
int

**set_epilog**(*epilog: str*) → None

Set the epilog of the command line help text

**Parameters**
**epilog** (*str*) – Text to include at the end of the command line help text.

**set_prolog**(*prolog: str*) → None

Set the prolog of the command line help text

**Parameters**
**prolog** (*str*) – Text to include before the list of commands in the command line help text. The prolog is printed after the synopsis of the application.

**property version:** *str*

The version of the package or application

**class** wilderness.**Command**(*name: str*, *title: Optional[str] = None*, *description: Optional[str] = None*, *add_help: bool = True*, *extra_sections: Optional[Dict[str, str]] = None*, *options_prolog: Optional[str] = None*, *options_epilog: Optional[str] = None*)

Bases: *DocumentableMixin*

**add_argument**(**args*, ***kwargs*)

**add_argument_group**(**args*, ***kwargs*) → *ArgumentGroup*

**add_mutually_exclusive_group**(**args*, ***kwargs*) → *MutuallyExclusiveGroup*

**property application:** *Optional[wilderness.application.Application]*

**create_manpage**() → *ManPage*

**abstract handle**() → int

**property name:** *str*

**register**()

Register a virtual subclass of an ABC.

Returns the subclass, to allow usage as a class decorator.

**property title:** *Optional[str]*

**class** wilderness.**Group**(*title: Optional[str] = None*, *is_root: bool = False*)

Bases: object

**add**(*command:* wilderness.command.Command) → None

**property application:** *Optional[wilderness.application.Application]*

**property commands:** *List[wilderness.command.Command]*

**commands_as_actions**() → List[Action]

**property is_root:** *bool*

Return whether the groups is its Application's root group

**set_app**(*app:* wilderness.application.Application) → None

**property title:** *Optional[str]*

**class** wilderness.**Tester**(*app:* Application)

Bases: object

**property application:** *Application*

**clear**()

**get_return_code**() → Optional[int]

**get_stderr**() → Optional[str]

**get_stdout**() → Optional[str]

**test_application**(*args: Optional[List[str]] = None*) → None

**test_command**(*cmd_name: str*, *args: List[str]*) → None

wilderness.**build_manpages**(*app:* wilderness.application.Application, *output_directory: str = 'man'*) → None

Write manpages to the output directory

**Parameters**

- **app** (*wilderness.Application*) – The application for which to generate manpages.
- **output_directory** (*str*) – The output directory to which to write the manpages.

CHAPTER

FIVE

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## W

wilderness, 20
wilderness.application, 11
wilderness.argparse_wrappers, 15
wilderness.command, 16
wilderness.documentable, 17
wilderness.formatter, 17
wilderness.group, 17
wilderness.help, 18
wilderness.manpages, 18
wilderness.tester, 19

# INDEX


## A

add() (*wilderness.Application method*), 21
add() (*wilderness.application.Application method*), 12
add() (*wilderness.Group method*), 24
add() (*wilderness.group.Group method*), 17
add_argument() (*wilderness.Application method*), 21
add_argument() (*wilderness.application.Application method*), 13
add_argument() (*wilderness.argparse_wrappers.ArgumentGroup method*), 15
add_argument() (*wilderness.argparse_wrappers.MutuallyExclusiveGroup method*), 16
add_argument() (*wilderness.Command method*), 24
add_argument() (*wilderness.command.Command method*), 16
add_argument_group() (*wilderness.Command method*), 24
add_argument_group() (*wilderness.command.Command method*), 16
add_group() (*wilderness.Application method*), 21
add_group() (*wilderness.application.Application method*), 13
add_mutually_exclusive_group() (*wilderness.Command method*), 24
add_mutually_exclusive_group() (*wilderness.command.Command method*), 16
add_section() (*wilderness.manpages.ManPage method*), 18
add_section_synopsis() (*wilderness.manpages.ManPage method*), 18
Application (*class in wilderness*), 20
Application (*class in wilderness.application*), 11
application (*wilderness.Command property*), 24
application (*wilderness.command.Command property*), 16
application (*wilderness.Group property*), 24
application (*wilderness.group.Group property*), 18
application (*wilderness.Tester property*), 24
application (*wilderness.tester.Tester property*), 19
args (*wilderness.documentable.DocumentableMixin property*), 17
argument_help (*wilderness.documentable.DocumentableMixin property*), 17
ArgumentGroup (*class in wilderness.argparse_wrappers*), 15
ArgumentParser (*class in wilderness.argparse_wrappers*), 16
author (*wilderness.Application property*), 21
author (*wilderness.application.Application property*), 13

## B

build_manpages() (*in module wilderness*), 25
build_manpages() (*in module wilderness.manpages*), 19

## C

clear() (*wilderness.Tester method*), 24
clear() (*wilderness.tester.Tester method*), 19
Command (*class in wilderness*), 24
Command (*class in wilderness.command*), 16
command (*wilderness.argparse_wrappers.ArgumentGroup property*), 16
command (*wilderness.argparse_wrappers.MutuallyExclusiveGroup property*), 16
commands (*wilderness.Application property*), 21
commands (*wilderness.application.Application property*), 13
commands (*wilderness.Group property*), 24
commands (*wilderness.group.Group property*), 18
commands_as_actions() (*wilderness.Group method*), 24
commands_as_actions() (*wilderness.group.Group method*), 18
create_manpage() (*wilderness.Application method*), 21
create_manpage() (*wilderness.application.Application method*), 13
create_manpage() (*wilderness.Command method*), 24
create_manpage() (*wilderness.command.Command method*), 16

create_manpage() (*wilderness.documentable.DocumentableMixin method*), 17

## D

description (*wilderness.documentable.DocumentableMixin property*), 17
DocumentableMixin (*class in wilderness.documentable*), 17

## E

exit() (*wilderness.argparse_wrappers.ArgumentParser method*), 16
export() (*wilderness.manpages.ManPage method*), 18

## F

format_help() (*wilderness.Application method*), 21
format_help() (*wilderness.application.Application method*), 13

## G

get_command() (*wilderness.Application method*), 22
get_command() (*wilderness.application.Application method*), 13
get_commands_text() (*wilderness.Application method*), 22
get_commands_text() (*wilderness.application.Application method*), 14
get_options_text() (*wilderness.documentable.DocumentableMixin method*), 17
get_return_code() (*wilderness.Tester method*), 24
get_return_code() (*wilderness.tester.Tester method*), 19
get_stderr() (*wilderness.Tester method*), 24
get_stderr() (*wilderness.tester.Tester method*), 19
get_stdout() (*wilderness.Tester method*), 24
get_stdout() (*wilderness.tester.Tester method*), 20
get_synopsis() (*wilderness.documentable.DocumentableMixin method*), 17
groffify() (*wilderness.manpages.ManPage method*), 19
groffify_line() (*wilderness.manpages.ManPage method*), 19
Group (*class in wilderness*), 24
Group (*class in wilderness.group*), 17
groups (*wilderness.Application property*), 22
groups (*wilderness.application.Application property*), 14

## H

handle() (*wilderness.Application method*), 22
handle() (*wilderness.application.Application method*), 14
handle() (*wilderness.Command method*), 24
handle() (*wilderness.command.Command method*), 16
handle() (*wilderness.help.HelpCommand method*), 18
have_man_command() (*in module wilderness.help*), 18
header() (*wilderness.manpages.ManPage method*), 19
help_action_factory() (*in module wilderness.help*), 18
HelpCommand (*class in wilderness.help*), 18
HelpFormatter (*class in wilderness.formatter*), 17

## I

is_root (*wilderness.Group property*), 24
is_root (*wilderness.group.Group property*), 18

## M

ManPage (*class in wilderness.manpages*), 18
metadata() (*wilderness.manpages.ManPage method*), 19
module
    wilderness, 20
    wilderness.application, 11
    wilderness.argparse_wrappers, 15
    wilderness.command, 16
    wilderness.documentable, 17
    wilderness.formatter, 17
    wilderness.group, 17
    wilderness.help, 18
    wilderness.manpages, 18
    wilderness.tester, 19
MutuallyExclusiveGroup (*class in wilderness.argparse_wrappers*), 16

## N

name (*wilderness.Application property*), 22
name (*wilderness.application.Application property*), 14
name (*wilderness.Command property*), 24
name (*wilderness.command.Command property*), 16
name (*wilderness.manpages.ManPage property*), 19

## P

parser (*wilderness.documentable.DocumentableMixin property*), 17
preamble() (*wilderness.manpages.ManPage method*), 19
print_help() (*wilderness.Application method*), 22
print_help() (*wilderness.application.Application method*), 14

## R

register() (*wilderness.Application method*), 23

register() (*wilderness.application.Application method*), 14
register() (*wilderness.Command method*), 24
register() (*wilderness.command.Command method*), 16
register() (*wilderness.help.HelpCommand method*), 18
run() (*wilderness.Application method*), 23
run() (*wilderness.application.Application method*), 14
run_command() (*wilderness.Application method*), 23
run_command() (*wilderness.application.Application method*), 15

## S

section_name() (*wilderness.manpages.ManPage method*), 19
set_app() (*wilderness.Group method*), 24
set_app() (*wilderness.group.Group method*), 18
set_epilog() (*wilderness.Application method*), 23
set_epilog() (*wilderness.application.Application method*), 15
set_prolog() (*wilderness.Application method*), 23
set_prolog() (*wilderness.application.Application method*), 15

## T

test_application() (*wilderness.Tester method*), 24
test_application() (*wilderness.tester.Tester method*), 20
test_command() (*wilderness.Tester method*), 25
test_command() (*wilderness.tester.Tester method*), 20
Tester (*class in wilderness*), 24
Tester (*class in wilderness.tester*), 19
title (*wilderness.Command property*), 24
title (*wilderness.command.Command property*), 16
title (*wilderness.Group property*), 24
title (*wilderness.group.Group property*), 18

## V

version (*wilderness.Application property*), 23
version (*wilderness.application.Application property*), 15

## W

wilderness
    module, 20
wilderness.application
    module, 11
wilderness.argparse_wrappers
    module, 15
wilderness.command
    module, 16
wilderness.documentable
    module, 17
wilderness.formatter
    module, 17
wilderness.group
    module, 17
wilderness.help
    module, 18
wilderness.manpages
    module, 18
wilderness.tester
    module, 19