click-option-group

Release 0.5.6

Eugene Prilepin

Jun 09, 2023

CONTENTS

click-option-group is a Click-extension package that adds option groups missing in Click.

CHAPTER

ONE

AIM AND MOTIVATION

Click is a package for creating powerful and beautiful command line interfaces (CLI) in Python, but it has no the functionality for creating option groups.

Option groups are convenient mechanism for logical structuring CLI, also it allows you to set the specific behavior and set the relationship among grouped options (mutually exclusive options for example). Moreover, argparse stdlib package contains this functionality out of the box.

At the same time, many Click users need this functionality. You can read interesting discussions about it in the following issues:

- issue 257
- issue 373
- issue 509
- issue 1137

The aim of this package is to provide group options with extensible functionality using canonical and clean API (Click-like API as far as possible).

CHAPTER

TWO

INSTALLING

You can install and update click-option-group using pip:

```
pip install -U click-option-group
```

CHAPTER

THREE

QUICKSTART

Here is a simple example how to use option groups in your Click-based CLI.

```
import click
from click_option_group import optgroup

@click.command()
@optgroup.group('Server configuration',
                help='The configuration of some server connection')
@optgroup.option('-h', '--host', default='localhost', help='Server host name')
@optgroup.option('-p', '--port', type=int, default=8888, help='Server port')
@click.option('--debug/--no-debug', default=False, help='Debug flag')
def cli(host, port, debug):
     print(params)

if __name__ == '__main__':
    cli()
```

CHAPTER

FOUR

CONTENTS

4.1 Tutorial

4.1.1 A Simple Example

Let's start with a simple example. Just use *optgroup* for declaring option groups by decorating your command function in Click-like API style.

```
# app.py

import click
from click_option_group import optgroup, RequiredMutuallyExclusiveOptionGroup

@click.command()
@optgroup.group('Server configuration',
                help='The configuration of some server connection')
@optgroup.option('-h', '--host', default='localhost', help='Server host name')
@optgroup.option('-p', '--port', type=int, default=8888, help='Server port')
@optgroup.option('-n', '--attempts', type=int, default=3, help='The number of connection␣
↪attempts')
@optgroup.option('-t', '--timeout', type=int, default=30, help='The server response␣
↪timeout')
@optgroup.group('Input data sources', cls=RequiredMutuallyExclusiveOptionGroup,
                help='The sources of the input data')
@optgroup.option('--tsv-file', type=click.File(), help='CSV/TSV input data file')
@optgroup.option('--json-file', type=click.File(), help='JSON input data file')
@click.option('--debug/--no-debug', default=False, help='Debug flag')
def cli(**params):
    print(params)

if __name__ == '__main__':
    cli()
```

Now we can see help for our app:

```
$ python app.py --help
Usage: app.py [OPTIONS]

Options:
  Server configuration:           The configuration of some server connection
```

(continues on next page)

(continued from previous page)

```
  -h, --host TEXT                 Server host name
  -p, --port INTEGER              Server port
  -n, --attempts INTEGER          The number of connection attempts
  -t, --timeout INTEGER           The server response timeout
Input data sources: [mutually_exclusive, required]
                                  The sources of the input data
  --tsv-file FILENAME             CSV/TSV input data file
  --json-file FILENAME            JSON input data file
--debug / --no-debug              Debug flag
--help                            Show this message and exit.
```

4.1.2 How It Works

Firstly, we declare the group using *optgroup.group()* decorator:

```
@optgroup.group('Server configuration', help='The configuration of some server connection
↪')
```

Note: Also we can declare groups just using `optgroup()`:

```
@optgroup('Server configuration', help='The configuration of some server connection')
```

Secondly, we declare the grouped options below using *optgroup.option()* decorator:

```
@optgroup.option('-h', '--host', default='localhost', help='Server host name')
@optgroup.option('-p', '--port', type=int, default=8888, help='Server port')
```

And that is all!

4.1.3 Checking Declarations

Attention: The important point: do not mix *optgroup.option()* and `click.option()` decorators!

click-option-group checks the decorators order and raises the exception if *optgroup.option()* and `click.option()` decorators are mixed.

The following code is incorrect:

```
@optgroup.group('My group')
@click.option('--hello')  # ERROR
@optgroup.option('--foo')
@click.option('--spam')  # ERROR
@optgroup.option('--bar')
```

The correct code looks like:

```
@click.option('--hello')
@optgroup.group('My group')
@optgroup.option('--foo')
@optgroup.option('--bar')
@click.option('--spam')
```

If we try to use `optgroup.option` without `optgroup.grpup()`/`optgroup()` declaration it also will raise the exception.

The following code is incorrect:

```
@click.command()
@click.option('--hello')
@optgroup.option('--foo')  # ERROR: Missing declaration of the option group
@optgroup.option('--bar')  # ERROR: Missing declaration of the option group
@click.option('--spam')
def cli(**params):
    pass
```

If we declare only option group without the options it will raise warning.

```
@click.command()
@click.option('--hello')
@optgroup.group('My group')  # WARN: The empty option group
@click.option('--spam')
def cli(**params):
    pass
```

4.1.4 API Features

Besides *optgroup* based decorators the package offers another way to declare grouped options using *OptionGroup* based class objects directly. We can use the instances of these classes and use its *OptionGroup.option()* method as decorator for declaring and adding options to the group.

Here is an example how it looks:

```
import click
from click_option_group import OptionGroup, RequiredMutuallyExclusiveOptionGroup

server_config = OptionGroup('Server configuration', help='The configuration of some␣
↪server connection')
input_sources = RequiredMutuallyExclusiveOptionGroup('Input data sources', help='The␣
↪sources of the input data')

@click.command()
@server_config.option('-h', '--host', default='localhost', help='Server host name')
@server_config.option('-p', '--port', type=int, default=8888, help='Server port')
@input_sources.option('--tsv-file', type=click.File(), help='CSV/TSV input data file')
@input_sources.option('--json-file', type=click.File(), help='JSON input data file')
@click.option('--debug/--no-debug', default=False, help='Debug flag')
def cli(**params):
    print(params)
```

(continues on next page)

(continued from previous page)

```
if __name__ == '__main__':
    cli()
```

In this case initially we create group objects and then we use *OptionGroup.option()* method for declaring options.

As well as in above example we cannot mix `option` and `click.option` decorators. The following code is incorrect and will raise the exception:

```
@server_config.option('-h', '--host', default='localhost', help='Server host name')
@click.option('--foo')  # ERROR
@server_config.option('-p', '--port', type=int, default=8888, help='Server port')
@input_sources.option('--tsv-file', type=click.File(), help='CSV/TSV input data file')
@click.option('--bar')  # ERROR
@input_sources.option('--json-file', type=click.File(), help='JSON input data file')
```

4.1.5 Behavior and Relationship among Options

The groups are useful to define the specific behavior and relationship among grouped options.

click-option-groups provides two main classes: *OptionGroup* and *GroupedOption*.

- *OptionGroup* class is a new entity for Click that provides the abstraction for grouping options and manage it.
- *GroupedOption* class is inherited from `click.Option` and provides the functionality for grouped options.

OptionGroup and *GroupedOption* classes contain the basic functionality for support option groups. Both these classes do not contain the specific behavior or relationship among grouped options.

The specific behavior can be implemented by using the inheritance, mainly, in *OptionGroup* sub classes. **click-option-groups** provides some useful *OptionGroup* based classes out of the box:

- *RequiredAnyOptionGroup* – At least one option from the group must be set
- *AllOptionGroup* – All options from the group must be set or none must be set
- *RequiredAllOptionGroup* – All options from the group must be set
- *MutuallyExclusiveOptionGroup* – Only one or none option from the group must be set
- *RequiredMutuallyExclusiveOptionGroup* – Only one required option from the group must be set

OptionGroup based class can be specified via `cls` argument in `optgroup()`/`optgroup.group()` decorator or can be used directly when the second API way is used.

If you want to implement some complex behavior you can create a sub class of *GroupedOption* class and use your *GroupedOption* based class via `cls` argument in `optgroup.option`/`OptionGroup.option` decorator method:

```
@click.command()
@optgroup('My group', cls=MyCustomOptionGroup)
@optgroup.option('--foo', cls=MyCustomGroupedOption)
...
```

4.1.6 Limitations

The package does not support nested option groups (option subgroups). This is intentional. Nested option groups complicate the implementation, API and CLI and most often it is not necessary.

If you think you need to nested option groups try redesign your CLI and doing it with nesting commands.

4.2 API Reference

optgroup	A helper class to manage creating groups and group options via decorators
GroupedOption	Represents grouped (related) optional values
OptionGroup	Option group manages grouped (related) options
RequiredAnyOptionGroup	Option group with required any options of this group
AllOptionGroup	Option group with required all/none options of this group
RequiredAllOptionGroup	Option group with required all options of this group
MutuallyExclusiveOptionGroup	Option group with mutually exclusive behavior for grouped options
RequiredMutuallyExclusiveOptionGroup	Option group with required and mutually exclusive behavior for grouped options

class click_option_group.**optgroup**

A global instance of the helper class to manage creating groups and group options via decorators

The class provides two decorator-methods: `group`/`__call__` and `option`. These decorators should be used for adding grouped options. The class have single global instance `optgroup` that should be used in most cases.

The example of usage:

```
from click_option_group import optgroup

...
@optgroup('Group 1', help='option group 1')
@optgroup.option('--foo')
@optgroup.option('--bar')
@optgroup.group('Group 2', help='option group 2')
@optgroup.option('--spam')
...
```

group(*name*, *, *cls*, *help*, **attrs*)

The decorator creates a new group and collects its options

Creates the option group and registers all grouped options which were added by *option()* decorator.

Parameters

- **name** – Group name or None for deault name
- **cls** – Option group class that should be inherited from *OptionGroup* class

- **help** – Group help or None for empty help
- **attrs** – Additional parameters of option group class

option(**param_decls*, ***attrs*)

The decorator adds a new option to the group

The decorator is lazy. It adds option decls and attrs. All options will be registered by `group()` decorator.

Parameters

- **param_decls** – option declaration tuple
- **attrs** – additional option attributes and parameters

---

class click_option_group.**GroupedOption**(*param_decls: Optional[Sequence[str]] = None*, **, group:* OptionGroup, ***attrs: Any*)

Represents grouped (related) optional values

The class should be used only with *OptionGroup* class for creating grouped options.

Parameters

- **param_decls** – option declaration tuple
- **group** – *OptionGroup* instance (the group for this option)
- **attrs** – additional option attributes

property group: *OptionGroup*

Returns the reference to the group for this option

Returns
OptionGroup the group instance for this option

---

class click_option_group.**OptionGroup**(*name: Optional[str] = None*, **, hidden: bool = False*, *help: Optional[str] = None*)

Option group manages grouped (related) options

The class is used for creating the groups of options. The class can de used as based class to implement specific behavior for grouped options.

Parameters

- **name** – the group name. If it is not set the default group name will be used
- **help** – the group help text or None

property name: **str**

Returns the group name or empty string if it was not set

Returns
group name

property help: **str**

Returns the group help or empty string if it was not set

Returns
group help

property name_extra: List[str]

Returns extra name attributes for the group

property forbidden_option_attrs: List[str]

Returns the list of forbidden option attributes for the group

get_help_record(*ctx: Context*) → Optional[Tuple[str, str]]

Returns the help record for the group

Parameters
ctx – Click Context object

Returns
the tuple of two fileds: *(name, help)*

option(**param_decls: str, **attrs: Any*) → Callable

Decorator attaches a grouped option to the command

The decorator is used for adding options to the group and to the Click-command

get_options(*ctx: Context*) → Dict[str, *GroupedOption*]

Returns the dictionary with group options

get_option_names(*ctx: Context*) → List[str]

Returns the list with option names ordered by addition in the group

handle_parse_result(*option:* GroupedOption, *ctx: Context*, *opts: Mapping[str, Any]*) → None

The method should be used for adding specific behavior and relation for options in the group

---

class click_option_group.**RequiredAnyOptionGroup**(*name: Optional[str] = None, *, hidden: bool = False, help: Optional[str] = None*)

Option group with required any options of this group

RequiredAnyOptionGroup defines the behavior: At least one option from the group must be set.

property forbidden_option_attrs: List[str]

Returns the list of forbidden option attributes for the group

property name_extra: List[str]

Returns extra name attributes for the group

handle_parse_result(*option:* GroupedOption, *ctx: Context*, *opts: Mapping[str, Any]*) → None

The method should be used for adding specific behavior and relation for options in the group

---

class click_option_group.**AllOptionGroup**(*name: Optional[str] = None, *, hidden: bool = False, help: Optional[str] = None*)

Option group with required all/none options of this group

***AllOptionGroup* defines the behavior:**

- All options from the group must be set or None must be set

property forbidden_option_attrs: List[str]

Returns the list of forbidden option attributes for the group

property name_extra: List[str]

Returns extra name attributes for the group

handle_parse_result(*option: GroupedOption, ctx: Context, opts: Mapping[str, Any]*) → None

The method should be used for adding specific behavior and relation for options in the group

---

class click_option_group.**RequiredAllOptionGroup**(*name: Optional[str] = None, *, hidden: bool = False, help: Optional[str] = None*)

Option group with required all options of this group

RequiredAllOptionGroup defines the behavior: All options from the group must be set.

property forbidden_option_attrs: List[str]

Returns the list of forbidden option attributes for the group

property name_extra: List[str]

Returns extra name attributes for the group

handle_parse_result(*option: GroupedOption, ctx: Context, opts: Mapping[str, Any]*) → None

The method should be used for adding specific behavior and relation for options in the group

---

class click_option_group.**MutuallyExclusiveOptionGroup**(*name: Optional[str] = None, *, hidden: bool = False, help: Optional[str] = None*)

Option group with mutually exclusive behavior for grouped options

***MutuallyExclusiveOptionGroup* defines the behavior:**

- Only one or none option from the group must be set

property forbidden_option_attrs: List[str]

Returns the list of forbidden option attributes for the group

property name_extra: List[str]

Returns extra name attributes for the group

handle_parse_result(*option: GroupedOption, ctx: Context, opts: Mapping[str, Any]*) → None

The method should be used for adding specific behavior and relation for options in the group

---

class click_option_group.**RequiredMutuallyExclusiveOptionGroup**(*name: Optional[str] = None, *, hidden: bool = False, help: Optional[str] = None*)

Option group with required and mutually exclusive behavior for grouped options

***RequiredMutuallyExclusiveOptionGroup* defines the behavior:**

- Only one required option from the group must be set

property name_extra: List[str]

Returns extra name attributes for the group

handle_parse_result(*option: GroupedOption, ctx: Context, opts: Mapping[str, Any]*) → None

The method should be used for adding specific behavior and relation for options in the group

4.3 Changelog

4.3.1 v0.5.6 (09.06.2023)

- Add `optgroup.help_option` decorator to add help option to the group (PR #50)
- Use GitHub Actions instead of Travis CI for CI
- Delete tox runner
- Add Python 3.11 to the setup classifiers

4.3.2 v0.5.5 (12.10.2022)

- Add `tests/` directory to tarball
- Add `tests_cov` extra dependencies for testing with coverage

4.3.3 v0.5.4 (12.10.2022)

- Move frame gathering into error code path (PR #34)
- Fix typos (PR #37)
- PEP 561 support (PR #42)
- Update docs dependencies and Travis CI Python version matrix (PR #43)

4.3.4 v0.5.3 (14.05.2021)

- Update Click dependency version to <9 (Issue #33)

4.3.5 v0.5.2 (28.11.2020)

- Do not use default option group name. An empty group name will not be displayed
- Slightly edited error messages
- All arguments except `name` in `optgroup` decorator must be keyword-only

4.3.6 v0.5.1 (14.06.2020)

- Fix incompatibility with autocomplete: out of the box Click completion and click-repl (Issue #14)

4.3.7 v0.5.0 (10.06.2020)

- Add `AllOptionGroup` class: all options from the group must be set or none must be set (PR #13)
- Fix type hints
- Update docs

4.3.8 v0.4.0 (18.05.2020)

- Support multi-layer wrapped functions (PR #10)
- Fix flake8 issues

4.3.9 v0.3.1

- Add `hidden=True` to `_GroupTitleFakeOption` as a temporary workaroud for issue #4

4.3.10 v0.3.0

- Add support for hidden options inside groups (PR #2)

4.3.11 v0.2.3

- Transfer the repo to click-contrib organisation

4.3.12 v0.2.2

- Add true lineno in warning when declaring empty option group
- Update readme

4.3.13 v0.2.1

- Use RuntimeWarning and stacklevel 2 when declaring empty option group
- Update readme

4.3.14 v0.2.0

- Implement `RequiredMutuallyExclusiveOptionGroup` class instead of `required` argument for `MutuallyExclusiveOptionGroup`
- Add tests with 100% coverage
- Update readme

4.3.15 v0.1.0

- First public release

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX