argser

Release 0.0.16

Bachynin Ivan

Mar 20, 2021

CONTENT

GitHub | PyPI | Docs | Examples | Installation | Changelog

Arguments parsing without boilerplate.

CHAPTER

ONE

FEATURES:

- arguments and type hints in IDE
- easy nested sub-commands
- sane defaults for arguments' params (ie if default of arg is 3 then type should be int, or when annotation/type/default is `bool` then generate 2 arguments: for true value `--arg` and for false `--no-arg`, ...)
-
- support for argparse actions
- common options/arguments reusability
- auto shortcuts generation: `--verbose -> -v, --foo_bar -> --fb`
- auto completion in shell (tnx to argcomplete)

CHAPTER

TWO

INSTALLATION

```
pip install argser
pip install argser[tabulate]  # for fancy tables support
pip install argser[argcomplete]  # for shell auto completion
pip install argser[all]
```

CHAPTER

THREE

NOTES FOR EXAMPLES

If second parameter of `parse_args` is string (as in almost all examples) then it will be parsed, otherwise arguments to parse will be taken from command line.

CHAPTER

FOUR

SIMPLE EXAMPLE

```python
from argser import parse_args

class Args:
    a = 'a'
    foo = 1
    bar: bool
    bar_baz = 42, "bar_baz help"

args = parse_args(Args, show=True)
```

```
 python playground.py -a "aaa bbb" -f 100500 --no-b
>>> Args(bar=False, a='aaa bbb', foo=100500, bar_baz=42)
```

```
 python playground.py -h
usage: playground.py [-h] [--bar] [--no-bar] [-a A] [--foo F] [--bar-baz B]

optional arguments:
    -h, --help            show this help message and exit
    --bar, -b             bool, default: None
    --no-bar, --no-b
    -a A                  str, default: 'a'
    --foo F, -f F         int, default: 1
    --bar-baz B, --bb B   int, default: 42. bar_baz help
```

CHAPTER

FIVE

GET ARGUMENTS FROM FUNCTION

```
import argser

def foo(a, b: int, c=1.2):
    return [a, b, c]

assert argser.call(foo, '1 2 -c 3.4') == ['1', 2, 3.4]
```

CHAPTER

SIX

SUB-COMMANDS

```python
from argser import parse_args, sub_command

class Args:
    a: bool
    b = []
    c = 5

    class SubArgs:
        d = 1
        e = '2'
    sub = sub_command(SubArgs, help='help message for sub-command')

args = parse_args(Args, '-a -b a b -c 10', parser_help='help message for root parser')
assert args.a is True
assert args.b == ['a', 'b']
assert args.c == 10
assert args.sub is None

args = parse_args(Args, '--no-a -c 10 sub -d 5 -e "foo bar"')
assert args.a is False
assert args.sub.d == 5
assert args.sub.e == 'foo bar'
```

```
 python playground.py -h
usage: playground.py [-h] [-a] [--no-a] [-b [B [B ...]]] [-c C] {sub} ...

positional arguments:
    {sub}

optional arguments:
    -h, --help      show this help message and exit
    -a              bool, default: None
    --no-a
    -b [B [B ...]]  List[str], default: []
    -c C            int, default: 5
```

```
 python playground.py sub1 -h
usage: playground.py sub [-h] [-d D] [-e E]

help message for sub-command

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

(continues on next page)

(continued from previous page)

```
    -d D        int, default: 1
    -e E        str, default: '2'
```

Can be deep nested:

```
from argser import parse_args, sub_command

class Args:
    a = 1
    class Sub1:
        b = 1
        class Sub2:
            c = 1
            class Sub3:
                d = 1
            sub3 = sub_command(Sub3)
        sub2 = sub_command(Sub2)
    sub1 = sub_command(Sub1)

args = parse_args(Args, '-a 1 sub1 -b 2 sub2 -c 3 sub3 -d 4')
```

6.1 Sub-commands from functions

```
import argser
subs = argser.SubCommands()

@subs.add
def foo():
    return 'foo'

@subs.add(description="foo bar")  # with additional arguments for sub-parser
def bar(a, b=1):
    return [a, b]

assert subs.parse('foo') == 'foo'
assert subs.parse('bar 1 -b 2') == ['1', 2]
```

6.1.1 Installation

```
pip install argser
pip install argser[tabulate]  # for fancy table support
pip install argser[argcomplete]  # for shell auto completion
pip install argser[all]
```

6.1.2 Examples

Sub-commands

```
>>> from argser import parse_args, sub_command

>>> class Args:
...     a: bool
...     b = []
...     c = 5
...     class Sub1:
...         d = 1
...         e = '2'
...         class Sub11:
...             a = 5
...         sub11 = sub_command(Sub11)
...     sub1 = sub_command(Sub1)
...     class Sub2:
...         f = 1
...         g = '2'
...     sub2 = sub_command(Sub2)

>>> args = parse_args(Args, '-a -c 10')
>>> assert args.a is True
>>> assert args.c == 10
>>> assert args.sub1 is None
>>> assert args.sub2 is None

>>> args = parse_args(Args, '-a -c 10 sub1 -d 5 sub11 -a 6')
>>> assert args.sub1.d == 5
>>> assert args.sub1.sub11.a == 6
>>> assert args.sub2 is None

>>> args = parse_args(Args, '-a -c 10 sub2 -g "foo bar"')
>>> assert args.sub1 is None
>>> assert args.sub2.g == "foo bar"
```

Arguments

str / int / float

```
>>> from argser import Opt

>>> class Args:
...     a: str  # default is None
...     b = 2  # default is 2
...     c: float = Opt(default=3.0, help="a3")  # default is 3.0, with additional␣
↪help text

>>> args = parse_args(Args, '-a "foo bar" -b 5 -c 4.2')
>>> assert args.a == 'foo bar'
>>> assert args.b == 5
>>> assert args.c == 4.2
```

booleans

```
>>> class Args:
...     a: bool  # default is None, to change use flags: -a or --no-a
...     b = True  # default is True, to change to False: ./script.py --no-b
...     c = False  # default is False, to change to True: ./script.py -c
...     d: bool = Opt(bool_flag=False)  # to change - specify value after flag: `-d␣
↪1` or `-d false` or ...

>>> args = parse_args(Args, '-d 0')
>>> assert args.a is None
>>> assert args.b is True
>>> assert args.c is False
>>> assert args.d is False

>>> args = parse_args(Args, '-a --no-b -c -d 1')
>>> assert args.a is True
>>> assert args.b is False
>>> assert args.c is True
>>> assert args.d is True
```

lists

```
>>> from typing import List

>>> class Args:
...     a = []  # default = [], type = str, nargs = *
...     b: List[int] = []  # default = [], type = int, nargs = *
...     c = [1.0]  # default = [], type = float, nargs = +
...     d: List[int] = Opt(default=[], nargs='+')  # default = [], type = int, nargs␣
↪= +

>>> args = parse_args(Args, '-a "foo bar" "baz"')
>>> assert args.a == ["foo bar", "baz"]
>>> args = parse_args(Args, '-b 1 2 3')
>>> assert args.b == [1, 2, 3]
>>> args = parse_args(Args, '-c 1.1 2.2')
>>> assert args.c == [1.1, 2.2]
>>> try:
...     args = parse_args(Args, '-d')  # error, -d should have more then one element
...     assert 0
... except SystemExit:
...     assert 1
```

positional arguments

```
>>> from argser import Arg

>>> class Args:
...     a: float = Arg()
...     b: str = Arg()

>>> args = parse_args(Args, '5 "foo bar"')
```

(continues on next page)

(continued from previous page)

```
>>> assert args.a == 5
>>> assert args.b == 'foo bar'
```

different prefixes

```
>>> from argser import Opt

>>> class Args:
...     aaa: int = Opt(prefix='-')
...     bbb: int = Opt(prefix='++')

>>> args = parse_args(Args, '-aaa 42 ++bbb 42')
>>> assert args.aaa == 42
>>> assert args.bbb == 42
```

argparse params

```
>>> from typing import List
>>> from argser import Opt

>>> class Args:
...     a = Opt(help="foo bar")  # with additional help message
...     b = Opt(action='count')
...     c: List[int] = Opt(action='append')

>>> args = parse_args(Args, '-a foo -bbb -c 1 -c 2')
>>> assert args.a == 'foo'
>>> assert args.b == 3
>>> assert args.c == [1, 2]
```

Actions

```
>>> from argser import Opt

>>> class Args:
...     a = Opt(action='store_const', default='42', const=42)

>>> args = parse_args(Args, '')
>>> assert args.a == '42'
>>> args = parse_args(Args, '-a')
>>> assert args.a == 42
```

```
>>> from typing import List
>>> from argser import Opt

>>> class Args:
...     a: List[int] = Opt(action='append', default=[])

>>> args = parse_args(Args, '-a 1')
>>> assert args.a == [1]
```

(continues on next page)

(continued from previous page)

```
>>> args = parse_args(Args, '-a 1 -a 2')
>>> assert args.a == [1, 2]
```

```
>>> from argser import Opt
>>> class Args:
...     verbose: int = Opt(action='count', default=0)

>>> args = parse_args(Args, '')
>>> assert args.verbose == 0

>>> args = parse_args(Args, '-vvv')
>>> assert args.verbose == 3
```

Reusability

```
>>> class CommonArgs:
...     value: int
...     verbose = Opt(action='count', default=0)
...     model_path = 'foo.pkl'

>>> class Args1(CommonArgs):
...     value: str  # redefine
...     epoch = 10

>>> class Args2(CommonArgs):
...     type = 'bert'

>>> args = parse_args(Args1, '--value "foo bar" --epoch 5')
>>> assert args.epoch == 5
>>> args = parse_args(Args2, '--value 10 --type albert')
>>> assert args.type == 'albert'
```

Call function with parsed arguments

```
>>> import argser

>>> def main(a, b: int, c=1.2, d: List[bool]=None):
...     return [a, b, c, d]

>>> assert argser.call(main, '1 2 -c 3.3 -d 1 0 1 1') == [
...     '1',
...     2,
...     3.3,
...     [True, False, True, True],
... ]
```

Or as decorator:

```
>>> import argser

>>> @argser.call('1 2')
... def foo(a, b: int):
...     assert a == '1' and b == 2
```

In examples above `a` (implicit string) and `b` (int) are positional argument because they don't have default values.

Multiple sub-commands:

```
>>> from argser import SubCommands
>>> subs = SubCommands()

>>> @subs.add(description="foo bar")
... def foo(): return 'foo'

>>> @subs.add
... def bar(a, b: int): return [a, b]

>>> subs.parse('foo')
'foo'
>>> subs.parse('bar 1 2')
['1', 2]
```

Override options globally

```
>>> import argser
>>> class Args:
...     a = 1
...     b = True
...     ccc_ddd = 'foo'

>>> args = argser.parse_args(
...     Args,
...     '+a 42 +b false +ccc+ddd "foo bar"',  # read from command if None
...     make_shortcuts=False,  # +ccc+ddd will not generate cd now
...     bool_flag=False,  # bool arg will require bool value near flag
...     prefix='+',  # change default prefix
...     repl=('_', '+'),  # change auto-replacer options (from, to)
...     override=True,  # only required if you need to override args defined with Opt/
↪Arg
... )

>>> assert args.a == 42
>>> assert args.b is False
>>> assert args.ccc_ddd == 'foo bar'
```

Display arguments

```
>>> from argser import sub_command, parse_args
>>> class Args:
...     a = 1
```

(continues on next page)

(continued from previous page)

```
...     b = 'foo'
...     class Sub:
...         a = 'foo bar'
...     sub = sub_command(Sub)
>>> args = parse_args(
...     Args,
...     '-a 42 sub -a "fooooooooo baaaaaaaaaaaaaaar baaaaaaaaaaaaaar"',
...     show='table',
... )
arg    value     arg     value
-----  -------   ------  ----------------------------
a      42        sub__a  'fooooooooo baaaaaaaaaaaaaar
b      'foo'             baaaaaaaaaaaaaar'
```

Or as tree:

```
>>> args = parse_args(
...     Args,
...     '-a 42 sub -a "fooooooooo baaaaaaaaaaaaaar baaaaaaaaaaaaaaz"',
...     show='tree',
... )
Args
├ a = 42
├ b = 'foo'
└ sub = Sub
  ├ a = 'fooooooooo baaaaaaaaaaaaaar
  └     baaaaaaaaaaaaaaz'
```

Or in one line:

```
>>> args = parse_args(
...     Args,
...     '-a 42 sub -a "fooooooooo baaaaaaaaaaaaaar baaaaaaaaaaaaaar"',
...     show=True,
... )
Args(a=42, b='foo', sub=Sub(a='fooooooooo baaaaaaaaaaaaaar baaaaaaaaaaaaaar'))
```

Or after parsing:

```
>>> from argser import print_args
>>> print_args(args, 'table')
arg    value     arg     value
-----  -------   ------  ----------------------------
a      42        sub__a  'fooooooooo baaaaaaaaaaaaaar
b      'foo'             baaaaaaaaaaaaaar'
```

Using existing parser

```
>>> from argparse import ArgumentParser, Namespace
>>> parser = ArgumentParser(prog='prog')
>>> action = parser.add_argument('--foo', default=42, type=int)

>>> class Args:
...     __namespace__: Namespace  # just for hints in IDE
...     a = 1
```

(continues on next page)

(continued from previous page)

```
...     b = True

>>> args = parse_args(Args, '--foo 100 -a 5 --no-b', parser=parser, parser_prog='WILL␣
↪NOT BE USED')
>>> args.a, args.b
(5, False)
>>> args.__namespace__.foo
100
```

Inspection

After parsing each attribute of parsed class will be replaced with populated instance of `argser.fields.Opt`.

```
>>> class Args:
...     a: bool
...     b = 1, "help for a"

>>> args = parse_args(Args, '--no-a -b 2')

>>> assert isinstance(Args.a, Opt)
>>> assert Args.a.type is bool
>>> assert args.a is False

>>> assert isinstance(Args.b, Opt)
>>> assert Args.b.type is int
>>> assert Args.b.help == "help for a"
>>> assert args.b == 2
```

Arguments factory

From callable:

```
>>> def read_a(value: str):
...     return int(value) + 1

>>> class Args:
...     a = Opt(default=1, factory=read_a)

>>> parse_args(Args, '-a 2').a
3
```

From method:

```
>>> class Args:
...     a = 1
...     def read_a(self, value: str):
...         return int(value) + 1

>>> parse_args(Args, '-a 2').a
3
```

From method with different name:

```
>>> class Args:
...     a = Opt(default=1, factory='get_a')
...     def get_a(self, value: str):
...         return int(value) + 1
```

```
>>> parse_args(Args, '-a 2').a
3
```

Auto completion

Check out argcomplete for setup guide.

Add autocompletes:

```
# using argcomplete's script
eval "$(register-python-argcomplete foo.py)"

# using argser
eval "$(argser auto)"  # for all scripts with PYTHON_ARGCOMPLETE_OK (in current dir)
eval "$(argser auto foo.py)"  # specific file
eval "$(argser auto /path/to/dir)"  # for all scripts (with PYTHON_ARGCOMPLETE_OK) in␣
↪/path/to/dir
eval "$(argser auto /path/to/dir foo.py)"  # combine
eval "$(argser auto --no-mark)"  # add autocomplete to every script
```

6.1.3 argser package

Submodules

argser.display module

argser.docstring module

argser.exceptions module

argser.fields module

argser.formatters module

argser.parse_func module

argser.parser module

argser.utils module

Module contents