

---

# Argparse 教學

發<sup>📖</sup> 3.10.11

Guido van Rossum  
and the Python development team

5 月 24, 2023

Python Software Foundation  
Email: docs@python.org

## Contents

1	概念	2
2	基本用法	2
3	介紹位置參數	3
4	介紹選項參數	4
4.1	短选项 . . . . .	6
5	現在結合位置與選項參數	6
6	进行一些小小的改进	10
6.1	矛盾的选项 . . . . .	11
7	結論	12

---

作者 Tshepang Lekhonkhobe

這個教學傾向簡介 Python 官方標準含式庫中推薦的命令列剖析模組 `argparse`。

---

**備註:** 另外兩個具有同樣功能的模組 `getopt` (一個相等於 C 語言中的 `getopt()`) 以及被<sup>📖</sup>用的 `optparse`。而 `argparse` 也是根據 `optparse` <sup>📖</sup>基礎發展而來，因此有非常近似的使用方式。

---

# 1 概念

藉由命令 **ls** 的使用開始這些功能的介紹：

```
$ ls
cpython  devguide  prog.py  pypy  rm-unused-function.patch
$ ls pypy
ctypes_configure  demo  dotviewer  include  lib_pypy  lib-python ...
$ ls -l
total 20
drwxr-xr-x 19 wena wena 4096 Feb 18 18:51 cpython
drwxr-xr-x  4 wena wena 4096 Feb  8 12:04 devguide
-rwxr-xr-x  1 wena wena  535 Feb 19 00:05 prog.py
drwxr-xr-x 14 wena wena 4096 Feb  7 00:59 pypy
-rw-r--r--  1 wena wena  741 Feb 18 01:01 rm-unused-function.patch
$ ls --help
Usage: ls [OPTION]... [FILE]...
List information about the FILES (the current directory by default).
Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.
...
```

我們可以從四個命令中可以學到的幾個概念：

- 命令 **ls** 在執行時不用其他參數就可以顯示出當前目錄下的內容。
- 根據這樣的概念延伸後來舉個例子，如果我們想秀出一個不在目錄的資料夾 **pypy** 的內容。我們可以在命令後加上一個位置參數。會用位置參數這樣的名稱是因爲程式會知道輸入的參數該做的事情。這樣的概念很像另一個命令 **cp**，基本的使用方式是 **cp SRC DEST**。第一個位置參數代表的是想要複製的目標，第二個位置的參數代表的則是想要複製到的地方。
- 現在我們想再增加一些，要顯示除了檔名之外更多的資訊。在這就可以選擇加上 **-l** 這個參數。
- 這是 **help** 文件的片段。對於以前從未使用過的程序來講非常有用，可以透過這些 **help** 文件來了解這些該怎使用。

## 2 基本用法

我們以一個很簡單的例子開始下面的介紹：

```
import argparse
parser = argparse.ArgumentParser()
parser.parse_args()
```

下面是運行這些代碼的結果：

```
$ python3 prog.py
$ python3 prog.py --help
usage: prog.py [-h]

options:
  -h, --help  show this help message and exit
$ python3 prog.py --verbose
usage: prog.py [-h]
prog.py: error: unrecognized arguments: --verbose
$ python3 prog.py foo
usage: prog.py [-h]
prog.py: error: unrecognized arguments: foo
```

接著是發生的情況：

- 運行這個❶本而❷有給與任何參數時就不會顯示任何東西至標準輸出畫面上。這❸❹不是這❺的有用。
- 第二個我們呈現出了 argparse 模組的用處。我們幾乎❶有做什❷事情，但已經得到一個很好的幫助信息。
- 這個 --help 選項可以簡短的表示成 -h，這是唯一一個選項我們不用去指明的（意即，❶有必要在這個參數後加上任何數值）。如果指定其他參數給他會造成錯誤。也因❷這樣，我們得到了一個免費的信息。

### 3 介紹位置參數

例如：

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("echo")
args = parser.parse_args()
print(args.echo)
```

運行這段代碼：

```
$ python3 prog.py
usage: prog.py [-h] echo
prog.py: error: the following arguments are required: echo
$ python3 prog.py --help
usage: prog.py [-h] echo

positional arguments:
  echo

options:
  -h, --help  show this help message and exit
$ python3 prog.py foo
foo
```

接者是發生的情❶：

- 我們增加了 add\_argument()，利用這個方法可以指名讓我們的程式接受哪些命令列參數。
- 現在呼叫我們的程序時需要指定一個參數選項。
- 在這個例子中，parse\_args() 這個方法確實根據了 echo 這個選項回傳了資料。
- 这一变量是 argparse 免费施放的某种“魔法”（即是说，不需要指定哪个变量是存储哪个值的）。你也可以注意到，这一名称与传递给方法的字符串参数一致，都是 echo。

注意，雖然 help 秀出了看起來不錯的信息，但現在❶❷有給予到實質幫助。像剛剛增加的 echo 這個位置參數，除了猜測和讀原始碼之外，我們根本不曉得該怎❸使用他。因此我們來做一點事讓他變得更有用：

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("echo", help="echo the string you use here")
args = parser.parse_args()
print(args.echo)
```

然後我們得到：

```
$ python3 prog.py -h
usage: prog.py [-h] echo

positional arguments:
```

(下页继续)

```

echo          echo the string you use here

options:
  -h, --help  show this help message and exit

```

現在來做一些更有用處的事情：

```

import argparse
parser = argparse.ArgumentParser()
parser.add_argument("square", help="display a square of a given number")
args = parser.parse_args()
print(args.square**2)

```

下面是運行這些代碼的結果：

```

$ python3 prog.py 4
Traceback (most recent call last):
  File "prog.py", line 5, in <module>
    print(args.square**2)
TypeError: unsupported operand type(s) for ** or pow(): 'str' and 'int'

```

那裏沒有如預期這樣。這是因為 `argparse` 將我們給予選項的值當成字串，除非我們告訴他要怎做。所以我們來告訴 `argparse` 將這個輸入值當成整數來使用：

```

import argparse
parser = argparse.ArgumentParser()
parser.add_argument("square", help="display a square of a given number",
                    type=int)
args = parser.parse_args()
print(args.square**2)

```

下面是運行這些代碼的結果：

```

$ python3 prog.py 4
16
$ python3 prog.py four
usage: prog.py [-h] square
prog.py: error: argument square: invalid int value: 'four'

```

這樣很順利。現在程序在開始之前會因錯誤的輸入而回報有用的訊息結束掉。

## 4 介紹選項參數

到目前为止，我们一直在研究位置参数。让我们看看如何添加可选的：

```

import argparse
parser = argparse.ArgumentParser()
parser.add_argument("--verbosity", help="increase output verbosity")
args = parser.parse_args()
if args.verbosity:
    print("verbosity turned on")

```

接者是結果：

```

$ python3 prog.py --verbosity 1
verbosity turned on
$ python3 prog.py
$ python3 prog.py --help

```

```
usage: prog.py [-h] [--verbosity VERBOSITY]

options:
  -h, --help            show this help message and exit
  --verbosity VERBOSITY
                        increase output verbosity
$ python3 prog.py --verbosity
usage: prog.py [-h] [--verbosity VERBOSITY]
prog.py: error: argument --verbosity: expected one argument
```

接者是發生的情況：

- 這個程式是寫成如果有指名 `--verbosity` 這個參數選項那才顯示些資訊，反之亦然。
- 不添加这一选项时程序没有提示任何错误而退出，表明这一选项确实是可选的。注意，如果一个可选参数没有被使用时，相关变量被赋值为 `None`，在此例中是 `args.verbosity`，这也就是为什么它在 `if` 语句中被当作逻辑假。
- **Help** 訊息稍微有些不一樣。
- 當使用 `--verbosity` 參數選項時必須要指定一個數值。

在上面的例子中 `--verbosity`，接受任意的整數，但對我們的程式來只接受兩個輸入值，`True` 或 `False`。所以我們來修改一下程式碼使其符合：

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("--verbose", help="increase output verbosity",
                    action="store_true")
args = parser.parse_args()
if args.verbose:
    print("verbosity turned on")
```

接者是結果：

```
$ python3 prog.py --verbose
verbosity turned on
$ python3 prog.py --verbose 1
usage: prog.py [-h] [--verbose]
prog.py: error: unrecognized arguments: 1
$ python3 prog.py --help
usage: prog.py [-h] [--verbose]

options:
  -h, --help  show this help message and exit
  --verbose   increase output verbosity
```

接者是發生的情況：

- 现在，这一选项更多地是一个标志，而非需要接受一个值的什么东西。我们甚至改变了选项的名字来符合这一思路。注意我们现在指定了一个新的关键词 `action`，并赋值为 `"store_true"`。这意味着，当这一选项存在时，为 `args.verbose` 赋值为 `True`。没有指定时则隐含地赋值为 `False`。
- 当你为其指定一个值时，它会报错，符合作为标志的真正的精神。
- 注意不同的 **help** 文件。

## 4.1 短选项

如果你很熟悉命令列的使用的话，你将会发现我还讲到关于短参数。其实这很简单：

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("-v", "--verbose", help="increase output verbosity",
                    action="store_true")
args = parser.parse_args()
if args.verbose:
    print("verbosity turned on")
```

效果就像这样：

```
$ python3 prog.py -v
verbosity turned on
$ python3 prog.py --help
usage: prog.py [-h] [-v]

options:
  -h, --help            show this help message and exit
  -v, --verbose         increase output verbosity
```

注意新的表示对于帮助文件也是一样的

## 5 现在结合位置与选项参数

我们的程式成长的越来越复杂：

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("square", type=int,
                    help="display a square of a given number")
parser.add_argument("-v", "--verbose", action="store_true",
                    help="increase output verbosity")
args = parser.parse_args()
answer = args.square**2
if args.verbose:
    print(f"the square of {args.square} equals {answer}")
else:
    print(answer)
```

然后现在的输出结果：

```
$ python3 prog.py
usage: prog.py [-h] [-v] square
prog.py: error: the following arguments are required: square
$ python3 prog.py 4
16
$ python3 prog.py 4 --verbose
the square of 4 equals 16
$ python3 prog.py --verbose 4
the square of 4 equals 16
```

- 我们带回了一个位置参数，结果发生了报错。
- 注意现在的顺序对于程式来已经不再重要了。

给我们的程序加上接受多个冗长度的值，然后实际来用用：

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("square", type=int,
                    help="display a square of a given number")
parser.add_argument("-v", "--verbosity", type=int,
                    help="increase output verbosity")
args = parser.parse_args()
answer = args.square**2
if args.verbosity == 2:
    print(f"the square of {args.square} equals {answer}")
elif args.verbosity == 1:
    print(f"{args.square}^2 == {answer}")
else:
    print(answer)
```

接者是結果：

```
$ python3 prog.py 4
16
$ python3 prog.py 4 -v
usage: prog.py [-h] [-v VERBOSITY] square
prog.py: error: argument -v/--verbosity: expected one argument
$ python3 prog.py 4 -v 1
4^2 == 16
$ python3 prog.py 4 -v 2
the square of 4 equals 16
$ python3 prog.py 4 -v 3
16
```

除了最后一个，看上去都不错。最后一个暴露了我们的程序中有一个 bug。我们可以通过限制 `--verbosity` 选项可以接受的值来修复它：

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("square", type=int,
                    help="display a square of a given number")
parser.add_argument("-v", "--verbosity", type=int, choices=[0, 1, 2],
                    help="increase output verbosity")
args = parser.parse_args()
answer = args.square**2
if args.verbosity == 2:
    print(f"the square of {args.square} equals {answer}")
elif args.verbosity == 1:
    print(f"{args.square}^2 == {answer}")
else:
    print(answer)
```

接者是結果：

```
$ python3 prog.py 4 -v 3
usage: prog.py [-h] [-v {0,1,2}] square
prog.py: error: argument -v/--verbosity: invalid choice: 3 (choose from 0, 1, 2)
$ python3 prog.py 4 -h
usage: prog.py [-h] [-v {0,1,2}] square

positional arguments:
  square                display a square of a given number

options:
  -h, --help            show this help message and exit
  -v {0,1,2}, --verbosity {0,1,2}
                        increase output verbosity
```

注意这一改变同时反应在错误信息和帮助信息里。

现在，让我们使用另一种的方式来改变冗长度。这种方式更常见，也和 CPython 的可执行文件处理它自己的冗长度参数的方式一致（参考 `python --help` 的输出）：

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("square", type=int,
                    help="display the square of a given number")
parser.add_argument("-v", "--verbosity", action="count",
                    help="increase output verbosity")
args = parser.parse_args()
answer = args.square**2
if args.verbosity == 2:
    print(f"the square of {args.square} equals {answer}")
elif args.verbosity == 1:
    print(f"{args.square}^2 == {answer}")
else:
    print(answer)
```

我們已經介紹過另一個操作“count”用來計算指定的選項出現的次數。

```
$ python3 prog.py 4
16
$ python3 prog.py 4 -v
4^2 == 16
$ python3 prog.py 4 -vv
the square of 4 equals 16
$ python3 prog.py 4 --verbosity --verbosity
the square of 4 equals 16
$ python3 prog.py 4 -v 1
usage: prog.py [-h] [-v] square
prog.py: error: unrecognized arguments: 1
$ python3 prog.py 4 -h
usage: prog.py [-h] [-v] square

positional arguments:
  square                display a square of a given number

options:
  -h, --help            show this help message and exit
  -v, --verbosity       increase output verbosity
$ python3 prog.py 4 -vvv
16
```

- 是的，它现在比前一版本更像是一个标志（和 `action="store_true"` 相似）。这能解释它为什么报错。
- 它也表现得与“store\_true”的行为相似。
- 現在來秀一下“count”這個動作會給予什麼。你可能之前就有見過這種用法。
- 如果你不添加 `-v` 标志，这一标志的值会是 `None`。
- 應該要如預期那樣，就算給予長選項我們也要獲得一樣的輸出結果。
- 可惜的是，对于我们的脚本获得的新能力，我们的帮助输出并没有提供很多信息，但我们总是可以通过改善文档来修复这一问题（比如通过 `help` 关键字参数）。
- 最后一个输出暴露了我们程序中的一个 bug。

讓我們來解問題

```
import argparse
parser = argparse.ArgumentParser()
```

(下页继续)

```

parser.add_argument("square", type=int,
                    help="display a square of a given number")
parser.add_argument("-v", "--verbosity", action="count",
                    help="increase output verbosity")
args = parser.parse_args()
answer = args.square**2

# bugfix: replace == with >=
if args.verbosity >= 2:
    print(f"the square of {args.square} equals {answer}")
elif args.verbosity >= 1:
    print(f"{args.square}^2 == {answer}")
else:
    print(answer)

```

而這也正是它給的：

```

$ python3 prog.py 4 -vvv
the square of 4 equals 16
$ python3 prog.py 4 -vvvv
the square of 4 equals 16
$ python3 prog.py 4
Traceback (most recent call last):
  File "prog.py", line 11, in <module>
    if args.verbosity >= 2:
TypeError: '>=' not supported between instances of 'NoneType' and 'int'

```

- 第一組輸出很好，修復了之前的 bug。也就是說，我們希望任何  $\geq 2$  的值尽可能详尽。
- 第三個輸出不是這圖的好。

我們來修復這個錯誤：

```

import argparse
parser = argparse.ArgumentParser()
parser.add_argument("square", type=int,
                    help="display a square of a given number")
parser.add_argument("-v", "--verbosity", action="count", default=0,
                    help="increase output verbosity")
args = parser.parse_args()
answer = args.square**2
if args.verbosity >= 2:
    print(f"the square of {args.square} equals {answer}")
elif args.verbosity >= 1:
    print(f"{args.square}^2 == {answer}")
else:
    print(answer)

```

我們剛剛引入了又一個新的關鍵字 `default`。我們把它設置為 0 來讓它可以與其他整數值相互比較。記住，默認情況下如果一個可選參數沒有被指定，它的值會是 `None`，並且它不能和整數值相比較（所以產生了 `TypeError` 異常）。

而且

```

$ python3 prog.py 4
16

```

憑借我們目前已學的東西你就可以做到許多事情，而我們還僅僅學了一些皮毛而已。`argparse` 模塊是非常強大的，在結束篇教程之前我們將再探索更多一些內容。

## 6 进行一些小小的改进

如果我们想要扩展我们的小程式做比範例更多的事：

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("x", type=int, help="the base")
parser.add_argument("y", type=int, help="the exponent")
parser.add_argument("-v", "--verbosity", action="count", default=0)
args = parser.parse_args()
answer = args.x**args.y
if args.verbosity >= 2:
    print(f"{args.x} to the power {args.y} equals {answer}")
elif args.verbosity >= 1:
    print(f"{args.x}^{args.y} == {answer}")
else:
    print(answer)
```

結果：

```
$ python3 prog.py
usage: prog.py [-h] [-v] x y
prog.py: error: the following arguments are required: x, y
$ python3 prog.py -h
usage: prog.py [-h] [-v] x y

positional arguments:
  x                the base
  y                the exponent

options:
  -h, --help            show this help message and exit
  -v, --verbosity        show verbosity count (0-2)

$ python3 prog.py 4 2 -v
4^2 == 16
```

请注意到目前为止我们一直在使用详细级别来 更改所显示的文本。以下示例则使用详细级别来显示 更多的文本：

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("x", type=int, help="the base")
parser.add_argument("y", type=int, help="the exponent")
parser.add_argument("-v", "--verbosity", action="count", default=0)
args = parser.parse_args()
answer = args.x**args.y
if args.verbosity >= 2:
    print(f"Running '{__file__}'")
if args.verbosity >= 1:
    print(f"{args.x}^{args.y} == ", end="")
print(answer)
```

結果：

```
$ python3 prog.py 4 2
16
$ python3 prog.py 4 2 -v
4^2 == 16
$ python3 prog.py 4 2 -vv
Running 'prog.py'
4^2 == 16
```

## 6.1 矛盾的选项

到目前为止，我们一直在使用 `argparse.ArgumentParser` 实例的两个方法。让我们再介绍第三个方法 `add_mutually_exclusive_group()`。它允许我们指定彼此相互冲突的选项。让我们再更改程序的其余部分以便使用新功能更有意义：我们将引入 `--quiet` 选项，它将与 `--verbose` 正好相反：

```
import argparse

parser = argparse.ArgumentParser()
group = parser.add_mutually_exclusive_group()
group.add_argument("-v", "--verbose", action="store_true")
group.add_argument("-q", "--quiet", action="store_true")
parser.add_argument("x", type=int, help="the base")
parser.add_argument("y", type=int, help="the exponent")
args = parser.parse_args()
answer = args.x**args.y

if args.quiet:
    print(answer)
elif args.verbose:
    print(f"{args.x} to the power {args.y} equals {answer}")
else:
    print(f"{args.x}^{args.y} == {answer}")
```

我们的程序现在变得更简洁了，我们出于演示需要略去了一些功能。无论如何，输出是这样的：

```
$ python3 prog.py 4 2
4^2 == 16
$ python3 prog.py 4 2 -q
16
$ python3 prog.py 4 2 -v
4 to the power 2 equals 16
$ python3 prog.py 4 2 -vq
usage: prog.py [-h] [-v | -q] x y
prog.py: error: argument -q/--quiet: not allowed with argument -v/--verbose
$ python3 prog.py 4 2 -v --quiet
usage: prog.py [-h] [-v | -q] x y
prog.py: error: argument -q/--quiet: not allowed with argument -v/--verbose
```

这应该很容易理解。我添加了末尾的输出这样你就可以看到其所达到的灵活性，即混合使用长和短两种形式的选项。

在我們結論之前，你可能想告訴你的用戶這個程式的主要目的，以防萬一他們不知道：

```
import argparse

parser = argparse.ArgumentParser(description="calculate X to the power of Y")
group = parser.add_mutually_exclusive_group()
group.add_argument("-v", "--verbose", action="store_true")
group.add_argument("-q", "--quiet", action="store_true")
parser.add_argument("x", type=int, help="the base")
parser.add_argument("y", type=int, help="the exponent")
args = parser.parse_args()
answer = args.x**args.y

if args.quiet:
    print(answer)
elif args.verbose:
    print(f"{args.x} to the power {args.y} equals {answer}")
else:
    print(f"{args.x}^{args.y} == {answer}")
```

请注意用法文本中有细微的差异。注意 `[-v | -q]`，它的意思是说我们可以使用 `-v` 或 `-q`，但不能同

时使用两者：

```
$ python3 prog.py --help
usage: prog.py [-h] [-v | -q] x y

calculate X to the power of Y

positional arguments:
  x                the base
  y                the exponent

options:
  -h, --help      show this help message and exit
  -v, --verbose
  -q, --quiet
```

## 7 結論

`argparse` 模組提供了比這☞展示更多的功能。它的文件是非常全面詳細且充滿了例子。通過本教學，你應該比較容易消化它們了。