# Argparse 教學

發🄵 *2.7.18*

**Guido van Rossum**
**and the Python development team**

**5 月 20, 2020**

**Python Software Foundation**
**Email: docs@python.org**

## Contents



**author** Tshepang Lekhonkhobe

This tutorial is intended to be a gentle introduction to `argparse`, the recommended command-line parsing module in the Python standard library. This was written for argparse in Python 3. A few details are different in 2.x, especially some exception messages, which were improved in 3.x.

**備🄵:** 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🄵用的 `optparse`。而 `argparse` 也是根據 `optparse` 🄵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F底下的F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F的資料夾 pypy 的F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F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FF的目標，第二個位置的參數代表的則是想要FF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F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F非常有用，可以透過這些 help 文件來了解這些該怎F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：

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

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

(下页继续)

(繼續上一頁)

```
usage: prog.py [-h]
prog.py: error: unrecognized arguments: foo
```

接者是發生的情🅵:

- 運行這個🅵本而🅵有給與任何參數時就不會顯示任何東西至標准輸出畫面上。這🅵🅵不是這🅵的有用。
- 第二個我們呈現出了 argparse 模組的用處。我們幾乎🅵有做什🅵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，🅵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🅵這樣，我們得到了一個免費的信息。

# 3 介紹位置參數

例子:

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

運行這段代碼:

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

positional arguments:
  echo

optional arguments:
  -h, --help  show this help message and exit
$ python prog.py foo
foo
```

接者是發生的情🅵:

- 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。
- The variable is some form of 『magic』that argparse performs for free (i.e. no need to specify which variable that value is stored in). You will also notice that its name matches the string argument given to the method, echo.

注意, 雖然 help 秀出了看起來不錯的信息, 但現在🅵🅵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🅵使用他。因此我們來做一點事讓他變得更有用:

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

(下页继续)

(繼續上一頁)

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

然後我們得到：

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

positional arguments:
  echo        echo the string you use here

optional arguments:
  -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：

```
$ python 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FF有如預期這樣。這是因F argparse 將我們給予選項的值當成字串，除然我們告訴他要怎F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：

```
$ python prog.py 4
16
$ python 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F錯誤的輸入而回報有用的訊息F結束掉。

# 4 介紹選項參數

So far we have been playing with positional arguments. Let us have a look on how to add optional ones:

```
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:

```
$ python prog.py --verbosity 1
verbosity turned on
$ python prog.py
$ python prog.py --help
usage: prog.py [-h] [--verbosity VERBOSITY]

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

接者是發生的情🄵:

- 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。
- To show that the option is actually optional, there is no error when running the program without it. Note that by default, if an optional argument isn't used, the relevant variable, in this case `args.verbosity`, is given `None` as a value, which is the reason it fails the truth test of the `if` statement.
- 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:

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

(下页继续)

(繼續上一頁)

```
usage: prog.py [-h] [--verbose]

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

接者是發生的情[F]:

- The option is now more of a flag than something that requires a value. We even changed the name of the option to match that idea. Note that we now specify a new keyword, `action`, and give it the value `"store_true"`. This means that, if the option is specified, assign the value `True` to `args.verbose`. Not specifying it implies `False`.
- It complains when you specify a value, in true spirit of what flags actually are.
- 注意不同的 help 文件。

## 4.1 Short options

如果你很熟悉命令列的使用的話，你將會發現我還[F]講到關於短參數。其實這很簡單:

```
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"
```

And here goes:

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

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

注意新的表示對於幫助文件也是一樣的

# 5 現在結合位置與選項參數

我們的程式成長的越來越[F]雜:

```
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()
```

(下页继续)

(繼續上一頁)

```
answer = args.square**2
if args.verbose:
    print "the square of {} equals {}".format(args.square, answer)
else:
    print answer
```

然後現在的輸出結果：

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

- We’ve brought back a positional argument, hence the complaint.
- 注意現在的順序對於程式來㠯已經不再重要了.

How about we give this program of ours back the ability to have multiple verbosity values, and actually get to use them:

```
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 "the square of {} equals {}".format(args.square, answer)
elif args.verbosity == 1:
    print "{}^2 == {}".format(args.square, answer)
else:
    print answer
```

接者是結果：

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

These all look good except the last one, which exposes a bug in our program. Let’s fix it by restricting the values the `--verbosity` option can accept:

```
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 "the square of {} equals {}".format(args.square, answer)
elif args.verbosity == 1:
    print "{}^2 == {}".format(args.square, answer)
else:
    print answer
```

接者是結果:

```
$ python 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)
$ python prog.py 4 -h
usage: prog.py [-h] [-v {0,1,2}] square

positional arguments:
  square                display a square of a given number

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

Note that the change also reflects both in the error message as well as the help string.

Now, let's use a different approach of playing with verbosity, which is pretty common. It also matches the way the CPython executable handles its own verbosity argument (check the output of `python --help`):

```
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 "the square of {} equals {}".format(args.square, answer)
elif args.verbosity == 1:
    print "{}^2 == {}".format(args.square, 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參數出現的次數。

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

(下页继续)

(繼續上一頁)

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

positional arguments:
  square           display a square of a given number

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

- Yes, it’s now more of a flag (similar to `action="store_true"`) in the previous version of our script. That should explain the complaint.
- It also behaves similar to 「store_true」action.
- 現在來秀一下「count」這個動作會給予什🄴。你可能之前就有見過這種用法。
- And, just like the 「store_true」action, if you don’t specify the `-v` flag, that flag is considered to have `None` value.
- 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。
- Sadly, our help output isn’t very informative on the new ability our script has acquired, but that can always be fixed by improving the documentation for our script (e.g. via the `help` keyword argument).
- That last output exposes a bug in our program.

讓我們來解🄴問題

```
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",
                    help="increase output verbosity")
args = parser.parse_args()
answer = args.square**2

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

而這也正是它給的：

```
$ python prog.py 4 -vvv
the square of 4 equals 16
```

(下页继续)

(繼續上一頁)

```
$ python prog.py 4 -vvvv
the square of 4 equals 16
$ python prog.py 4
Traceback (most recent call last):
  File "prog.py", line 11, in <module>
    if args.verbosity >= 2:
TypeError: unorderable types: NoneType() >= int()
```

- First output went well, and fixes the bug we had before. That is, we want any value >= 2 to be as verbose as possible.
- 第三個輸出不是這🅵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 "the square of {} equals {}".format(args.square, answer)
elif args.verbosity >= 1:
    print "{}^2 == {}".format(args.square, answer)
else:
    print answer
```

We’ ve just introduced yet another keyword, `default`. We’ ve set it to `0` in order to make it comparable to the other int values. Remember that by default, if an optional argument isn’ t specified, it gets the `None` value, and that cannot be compared to an int value (hence the `TypeError` exception).

而且

```
$ python prog.py 4
16
```

You can go quite far just with what we’ ve learned so far, and we have only scratched the surface. The `argparse` module is very powerful, and we’ ll explore a bit more of it before we end this tutorial.

# 6 Getting a little more advanced

如果我們想要擴展我們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 "{} to the power {} equals {}".format(args.x, args.y, answer)
elif args.verbosity >= 1:
```

(下页继续)

(繼續上一頁)

```
    print "{}^{} == {}".format(args.x, args.y, answer)
else:
    print answer
```

結果:

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

positional arguments:
  x                the base
  y                the exponent

optional arguments:
  -h, --help       show this help message and exit
  -v, --verbosity
$ python prog.py 4 2 -v
4^2 == 16
```

Notice that so far we've been using verbosity level to *change* the text that gets displayed. The following example instead uses verbosity level to display *more* text instead:

```
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 "Running '{}'".format(__file__)
if args.verbosity >= 1:
    print "{}^{} ==".format(args.x, args.y),
print answer
```

結果:

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

## 6.1 Conflicting options

So far, we have been working with two methods of an `argparse.ArgumentParser` instance. Let's introduce a third one, `add_mutually_exclusive_group()`. It allows for us to specify options that conflict with each other. Let's also change the rest of the program so that the new functionality makes more sense: we'll introduce the `--quiet` option, which will be the opposite of the `--verbose` one:

```
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 "{} to the power {} equals {}".format(args.x, args.y, answer)
else:
    print "{}^{} == {}".format(args.x, args.y, answer)
```

Our program is now simpler, and we've lost some functionality for the sake of demonstration. Anyways, here's the output:

```
$ python prog.py 4 2
4^2 == 16
$ python prog.py 4 2 -q
16
$ python prog.py 4 2 -v
4 to the power 2 equals 16
$ python 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
$ python 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
```

That should be easy to follow. I've added that last output so you can see the sort of flexibility you get, i.e. mixing long form options with short form ones.

在我們結論之前，你可能想告訴你的用🄴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:
```

(下页继续)

(繼續上一頁)

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

Note that slight difference in the usage text. Note the `[-v | -q]`, which tells us that we can either use `-v` or `-q`, but not both at the same time:

```
$ python 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

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

## 7 結論

`argparse` 模組提供了比這F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。