你真的会使用 Python 命令吗?

原创 小帅b 学习python的正确姿势 今天

来自专辑

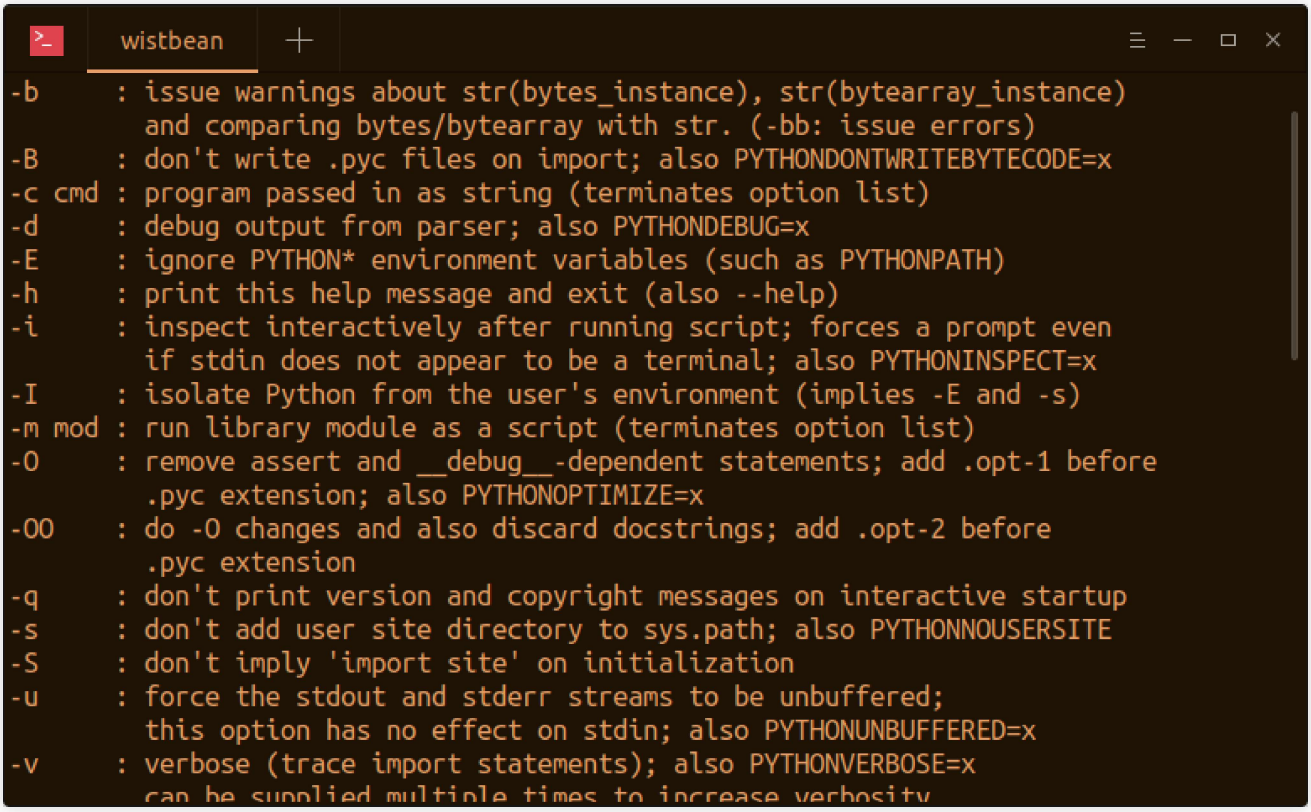
各种 Python 小技巧

我想你最常用到的 Python 命令就是运行 Python 脚本文件，像这样：

```
python xxx.py
```

不过你知道为什么这样就可以直接执行 Python 代码么?

还有， Python 还有一些常用的命令可玩，你试过吗?



接下来，就是学习 Python 的正确姿势：

我们来说说那些我们常用到的 Python 命令。

python xxx.py

当你通过 Python 执行脚本文件的时候，sys.argv[0] 会存储这个 py 文件名称：

```
xxx.py+
1 import sys
2
3 if __name__ == '__main__':
4     print('sys.argv[0]:', sys.argv[0])
```

```
wistbean@wistbean:~$ python xxx.py
sys.argv[0]: xxx.py
```

而当你在 python xxx.py 后面再添加一些参数的时候，sys.argv 也同样可以接收到相关的参数：

```
xxx.py+
1 import sys
2
3 if __name__ == '__main__':
4     print('sys.argv[0]:', sys.argv[0])
5     print('sys.argv[1]:', sys.argv[1])
6     print('sys.argv[2]:', sys.argv[2])
7     print('sys.argv[3]:', sys.argv[3])
```

```
wistbean@wistbean:~$ python xxx.py aaa bbb ccc
sys.argv[0]: xxx.py
sys.argv[1]: aaa
sys.argv[2]: bbb
sys.argv[3]: ccc
```

而且，它会把你的执行的 py 文件路径添加到 sys.path 中来，将它作为主模块来运行：

```
wistbean@wistbean:~$ python xxx.py
sys.argv[0]: xxx.py
sys.path: ['/home/wistbean', '/usr/local/lib/python38
', '/usr/local/lib/python3.8/lib-dynload', '/home/wis
packages', '/usr/local/lib/python3.8/site-packages']
```

除了直接使用 python 执行脚本文件之外，你也可以使用 Python 执行整个 Python 项目目录或者压缩文件，不过在这里面你需要定义一个 __main__.py ，要不然解释器无法识别运行：

```
wistbean@wistbean:~$ python todo/
/usr/local/bin/python3.8: can't find '__main__' module in 'todo/'
```

而当你的目录中有定义 __main__.py 的时候，它就可以将 main 添加到 sys.path 中来，作为 main 模块执行：

```
wistbean@wistbean:~$ python todo/
['todo/', '/usr/local/lib/python38.zip', '/usr/local
python3.8/lib-dynload', '/home/wistbean/.local/lib/p
cal/lib/python3.8/site-packages']
```

除此之外，你还可以直接使用脚本的文件执行 Python 代码，在你的脚本文件的第一行定义 Python 环境：

```
xxx.py+
#!/usr/bin/python3

import sys

if __name__ == '__main__':
    print('sys.argv[0]:', sys.argv[0])
    print('sys.path:', sys.path)
```

接着添加脚本文件的执行权限，然后就可以直接用脚本文件名称直接运行 Python 了：

```
wistbean@wistbean:~$ chmod +x xxx.py
wistbean@wistbean:~$ ./xxx.py
sys.argv[0]: ./xxx.py
sys.path: ['/home/wistbean', '/usr/lib/python36.zip',
/python3.6/lib-dynload', '/usr/local/lib/python3.6/di
thon3.6/dist-packages/setuptools-40.8.0-py3.6.egg', '
```

python -c

使用 python -c 可以让你在命令行中写 Python 代码执行，可以使用 ; 进行代码分行：

```
wistbean@wistbean:~$ python -c "foo = 'fxxkpython'; print(foo)"
fxxkpython
```

一种更好的方式是使用空行对代码进行分行：

```
wistbean@wistbean:~$ python -c "
> import calendar
> print(calendar.month(2020, 8))
> "
    August 2020
Mo Tu We Th Fr Sa Su
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
```

当你在 python 解释器中使用 -c 命令的时候，它会将 -c 添加到 sys.argv[0] 中，将当前的路径添加到 sys.path 中：

```
wistbean@wistbean:~$ python -c "import xxx"
sys.argv[0]: -c
sys.path: ['', '/usr/local/lib/python38.zip', '/usr/l
/lib/python3.8/lib-dynload', '/home/wistbean/.local/l
sr/local/lib/python3.8/site-packages']
```

python -m

使用 -m 模块可以以脚本的方式执行 Python 的模块或者包，因为执行的是模块，所以就不用像执行脚本文件那样把 .py 扩展名称写出来。

```
wistbean@wistbean:~$ python -m xxx.py
sys.argv[0]: -m
sys.path: ['/home/wistbean', '/usr/local/lib/python38.zip', '/usr/local/lib/python3.8
', '/usr/local/lib/python3.8/lib-dynload', '/home/wistbean/.local/lib/python3.8/site-
packages', '/usr/local/lib/python3.8/site-packages']
/usr/local/bin/python3.8: Error while finding module specification for 'xxx.py' (Modu
leNotFoundError: __path__ attribute not found on 'xxx' while trying to find 'xxx.py')
```

```
wistbean@wistbean:~$ python -m xxx
sys.argv[0]: /home/wistbean/xxx.py
sys.path: ['/home/wistbean', '/usr/local/lib/python38
', '/usr/local/lib/python3.8/lib-dynload', '/home/wis
packages', '/usr/local/lib/python3.8/site-packages']
```

可以看到，模块的完全路径会添加到 sys.argv[0]，将当前的路径添加到 sys.path 中，将模块名作为 main 执行。

你能体会到 python xxx.py 和 python -m xxx 之间的区别么？

当然，你也可以使用 -m 来执行包中的模块：

```
wistbean@wistbean:~$ python -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

通过 -m 的形式，解释器会通过搜索路径找到相应的模块执行，这样你就不需要想执行 .py 那样写入文件的绝对路径啦。

使用 python -m 的一个常用场景是在不同的版本环境中执行相应的模块，比如使用 pip 安装模块的时候：

```
wistbean@wistbean:~$ python3.6 -m pip install requests
Collecting requests
  Downloading http://pypi.doubanio.com/packages/45/1e/0c169c6a5381e241ba7404
d86ab872c9bed8bdcd4c423954103/requests-2.24.0-py2.py3-none-any.whl (61kB)
    100% |                                | 71kB 14.1MB/s
Collecting chardet<4,>=3.0.2 (from requests)
  Downloading http://pypi.doubanio.com/packages/bc/a9/01ffebfb562e4274b6487b
ca55ec7510b22e4c51f14098443b8/chardet-3.0.4-py2.py3-none-any.whl (133kB)
    100% |                                | 143kB 5.4MB/s
Collecting urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 (from requests)
```

```
wistbean@wistbean:~$ python3.8 -m pip install requests
Defaulting to user installation because normal site-packages is not writ
Looking in indexes: http://pypi.douban.com/simple
Requirement already satisfied: requests in /usr/local/lib/python3.8/site
22.0)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 i
```

```
wistbean@wistbean:~$ python3.9 -m pip install requests
Defaulting to user installation because normal site-packages is not writeab
Looking in indexes: http://pypi.douban.com/simple
Requirement already satisfied: requests in ./.local/lib/python3.9/site-pack
.0)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.9/si
 (from requests) (2.9)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in
```

python -i

当你使用 -c 命令的时候，还想执行完相关代码后进入 Python 交互模式就可以使用 -i ：

```
wistbean@wistbean:~$ python -i -c "import xxx"
sys.argv[0]: -c
sys.path: ['', '/usr/local/lib/python38.zip', '/usr/local/lib/p
/python3.8/lib-dynload', '/home/wistbean/.local/lib/python3.8/s
/lib/python3.8/site-packages']
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
```

还有其它的命令，你可以通过 python –h 找到相关的解释：

```
wistbean@wistbean:~$ python -h
usage: /usr/local/bin/python3.8 [option] ... [-c cmd | -m mod | file | -] [arg] ...
Options and arguments (and corresponding environment variables):
-b     : issue warnings about str(bytes_instance), str(bytearray_instance)
         and comparing bytes/bytearray with str. (-bb: issue errors)
-B     : don't write .pyc files on import; also PYTHONDONTWRITEBYTECODE=x
-c cmd : program passed in as string (terminates option list)
-d     : debug output from parser; also PYTHONDEBUG=x
-E     : ignore PYTHON* environment variables (such as PYTHONPATH)
-h     : print this help message and exit (also --help)
-i     : inspect interactively after running script; forces a prompt even
         if stdin does not appear to be a terminal; also PYTHONINSPECT=x
-I     : isolate Python from the user's environment (implies -E and -s)
-m mod : run library module as a script (terminates option list)
-O     : remove assert and __debug__-dependent statements; add .opt-1 before
         .pyc extension; also PYTHONOPTIMIZE=x
-OO    : do -O changes and also discard docstrings; add .opt-2 before
         .pyc extension
-q     : don't print version and copyright messages on interactive startup
-s     : don't add user site directory to sys.path; also PYTHONNOUSERSITE
-S     : don't imply 'import site' on initialization
-u     : force the stdout and stderr streams to be unbuffered;
         this option has no effect on stdin; also PYTHONUNBUFFERED=x
-v     : verbose (trace import statements); also PYTHONVERBOSE=x
```

可能有时候你也想要提供一些命令给别人使用，Python 内置了一个 argparse 的库，你可以使用它来创建你要提供的命令，比如这样：

```
wistbean@wistbean: ~
xxx.py                                                    buffers
1 #!/usr/bin/python3
2
3 import argparse
4 parser = argparse.ArgumentParser()
5 parser.add_argument("-f", "--fxxk", help="fxxxxxxxxxxxkpython",
6                     action="store_true")
7 args = parser.parse_args()
8 if args.fxxk:
9     print ("用户执行了 fxxk 命令")
NORMAL  xxx.py                    pyt…  11%  1/9 ln :  1
```

执行的时候就可以使用相关的命令了：

```
wistbean@wistbean:~$ python xxx.py -f
用户执行了 fxxk 命令
wistbean@wistbean:~$ python xxx.py -h
usage: xxx.py [-h] [-f]

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

除此之外，还有一个叫做 click 的第三方库，也是用来创建命令的，不同的是它可以用装饰器的方式实现，你可以直接使用 @click.option 来定义命令选项，用起来相对简单，这是一个官方的例子：

```
wistbean@wistbean: ~
xxx.py+                                                    buffers
1  #!/usr/bin/python3
2
3  import click
4
5  @click.command()
6  @click.option('--count', default=1, help='Number of greetings.')
7  @click.option('--name', prompt='Your name',
8                help='The person to greet.')
9  def hello(count, name):
10     """Simple program that greets NAME for a total of COUNT times."""
11     for x in range(count):
12         click.echo('Hello %s!' % name)
13
14 if __name__ == '__main__':
15     hello()
NORMAL  xxx.py[+]                    pyt…   100%  15/15 ln : 11
```

使用起来是这样的：

```
wistbean@wistbean:~$ python xxx.py --help
Usage: xxx.py [OPTIONS]

  Simple program that greets NAME for a total of COUNT times.

Options:
  --count INTEGER  Number of greetings.
  --name TEXT      The person to greet.
  --help           Show this message and exit.
```

```
wistbean@wistbean:~$ python xxx.py --count 6 --name xsb_pro
Hello xsb_pro!
Hello xsb_pro!
Hello xsb_pro!
Hello xsb_pro!
Hello xsb_pro!
Hello xsb_pro!
```

当然，它还提供了多种创建命令的方式，你可以在以下链接中找到：

https://click.palletsprojects.com/en/6.x/

ok，以上就是小帅b今天给你带来的分享，希望对你有帮助，那么我们下回见，peace！

扫一扫

学习 Python 没烦恼

学习python的正确姿势

微信扫描二维码，关注我的公众号

请拼命点赞

阅读原文