Python Command Line Arguments Examples

Posted By nixCraft <webmaster@cyberciti.biz> On March 13, 2013 @ 10:47 pm [7 Comments]

> I am a new Python programming user. I am working on user-friendly command-line interfaces, and I want to pass input via the shell args as follows:


[1]

```
./myscript.py filename
./myscript.py in.file output.file
./myscript.py -i in.file -o output.file
```

How do I pass and parse command-line options and arguments using Python under Unix like operating systems?

Python provides the following two options:

1. The **`getopt module`** is a parser for command line options whose API is designed to be familiar to users of the C getopt() function.
2. The **`argparse module`** makes it easy to write user-friendly command-line interfaces and I recommend this module for all users.

Contents

- Command line parameters [2]
- getopt module [3]
- Parsing command line arguments [4]
- argparse module [5]
- Examples [6]
- References [7]

Understanding command line parameters

Most Unix and Linux commands can take different actions depending on the command line arguments supplied to the command.

Tutorial details	
Difficulty	**Intermediate** [8] (**rss** [9])
Root privileges	No
Requirements	Python
Estimated completion time	10 minute

What is a command line argument?

A command line argument is nothing but an argument sent to a program being called. A program can take any number of command line arguments. For example, type the following command:

```
cat /etc/passwd
```

cat is the name of an actual command and shell executed this command when you type command at shell prompt. The first word on the command line is:

1. `cat` - name of the command to be executed.
2. Everything else on command line is taken as arguments to this command.
3. Total number of command line arguments passed to the ls command including command name itself are 2.

Consider the following example:

```
ls --sort=size foo.txt bar.txt
```

Where,

1. `ls` : Command name.
2. `--sort=size foo.txt bar.txt` : The arguments.
3. `4` : The total number of arguments.

Try the following command and note down its command line arguments:

Command	Command name	Total number of arguments	Arguments
`ls`	ls	1	None
`ls /etc/resolv.conf`	ls	2	/etc/resolv.conf

`grep --color root /etc/passwd`	grep	4	--color, root, /etc/passwd

getopt module

The syntax is:

```
import sys

# get argument list using sys module
sys.argv
```

Examples

Use the following code to display the total number of arguments passed to the demo.py:

```
#!/usr/bin/python
# demo.py - CMD Args Demo By nixCraft
import sys

# Get the total number of args passed to the demo.py
total = len(sys.argv)

# Get the arguments list
cmdargs = str(sys.argv)

# Print it
print ("The total numbers of args passed to the script: %d " % total)
print ("Args list: %s " % cmdargs)
```

Save and close the file. Run it as follows:

```
$ python demo.py input.txt output.txt
```

OR

```
$ chmod +x demo.py
$ ./demo.py input.txt output.txt
```

Sample outputs:

```
The total numbers of args passed to the script: 3
Args list: ['demo.py', 'input.txt', 'output.txt']
```

Parsing command line arguments (options and arguments)

Edit the file demo.py and update it as follows:

```
#!/usr/bin/python
__author__ = 'nixCraft'
import sys

total = len(sys.argv)
cmdargs = str(sys.argv)
print ("The total numbers of args passed to the script: %d " % total)
print ("Args list: %s " % cmdargs)
# Pharsing args one by one
print ("Script name: %s" % str(sys.argv[0]))
print ("First argument: %s" % str(sys.argv[1]))
print ("Second argument: %s" % str(sys.argv[2]))
```

Save and close the file. Run it as follows:

```
$ ./demo.py input.txt output.txt
```

Sample outputs:

```
The total numbers of args passed to the script: 3
Args list: ['./demo.py', 'input.txt', 'output.txt']
Script name: ./demo.py
First argument: input.txt
Second argument: output.txt
```

Try to run the same program as follows:

```
$ ./demo.py -i input.txt -o output.txt
```

Sample outputs:

```
The total numbers of args passed to the script: 5
Args list: ['./demo.py', '-i', 'input.txt', '-o', 'output.txt']
Script name: ./demo.py
First argument: -i
Second argument: input.txt
```

The script only printed first three arguments and skipped the rest. The output is not correct. Further, how do you separate the options such as `-i` or `-o` and arguments passed to each option such as `input.txt` or `output.txt`? You can fix this problem with the following code:

```
#!/usr/bin/python
__author__ = 'nixCraft'
import sys

total = len(sys.argv)
cmdargs = str(sys.argv)
print ("The total numbers of args passed to the script: %d " % total)
print ("Args list: %s " % cmdargs)
print ("Script name: %s" % str(sys.argv[0]))
for i in xrange(total):
    print ("Argument # %d : %s" % (i, str(sys.argv[i])))
```

Sample outputs:

```
$ ./demo.py -i input.txt -o output.txt
The total numbers of args passed to the script: 5
Args list: ['./demo.py', '-i', 'input.txt', '-o', 'output.txt']
Script name: ./demo.py
Argument # 0 : ./demo.py
Argument # 1 : -i
Argument # 2 : input.txt
Argument # 3 : -o
Argument # 4 : output.txt
```

However, I do not recommend to use the for loop and phrase the argument method. It was included here for demonstration and historical purpose only. The correct and recommend Python syntax is as follows:

```
#!/usr/bin/python
__author__ = 'nixCraft'
import sys, getopt
# Store input and output file names
ifile=''
ofile=''

# Read command line args
myopts, args = getopt.getopt(sys.argv[1:],"i:o:")

###############################
# o == option
# a == argument passed to the o
###############################
for o, a in myopts:
    if o == '-i':
        ifile=a
    elif o == '-o':
        ofile=a
    else:
        print("Usage: %s -i input -o output" % sys.argv[0])
```

```
# Display input and output file name passed as the args
print ("Input file : %s and output file: %s" % (ifile,ofile) )
```

Run it as follows:

```
$ ./demo.py -i foo.txt -o bar.txt
Input file : foo.txt and output file: bar.txt
$ ./demo.py -o bar.txt -i foo.txt
Input file : foo.txt and output file: bar.txt
$ ./demo.py -o bar.txt
Input file :  and output file: bar.txt
$ ./demo.py
Input file :  and output file:
```

Creating usage messages

The updated code is as follows:

```
#!/usr/bin/python
__author__ = 'vivek'
import sys, getopt

ifile=''
ofile=''

##############################
# o == option
# a == argument passed to the o
##############################
# Cache an error with try..except
# Note: options is the string of option letters that the script wants to recognize, with
# options that require an argument followed by a colon (':') i.e. -i fileName
#
try:
    myopts, args = getopt.getopt(sys.argv[1:],"i:o:")
except getopt.GetoptError as e:
    print (str(e))
    print("Usage: %s -i input -o output" % sys.argv[0])
    sys.exit(2)

for o, a in myopts:
    if o == '-i':
        ifile=a
    elif o == '-o':
        ofile=a

# Display input and output file name passed as the args
print ("Input file : %s and output file: %s" % (ifile,ofile) )
```

Run it as follows:

```
$ ./demo.py -i input.txt -o output.txt
Input file : input.txt and output file: output.txt
$ ./demo.py -i input.txt -o output.txt -z
option -z not recognized
$ ./demo.py -i input.txt -o
option -o requires argument
Usage: ./demo.py -i input -o output
```

Say hello to argparse module

The argparse module helps you to write the same logic with less code and more informative help and error messages. The syntax is:

WARNING! These examples requires Python version 2.7 and above.

```
import argparse
parser = argparse.ArgumentParser(description='ADD YOUR DESCRIPTION HERE
```

```
parser.add_argument('-i','--input', help='Input file name',required=True)
args = parser.parse_args()
## do something with args
```

Examples

Create a file called demo1.py:

```
#!/usr/bin/python
import argparse
__author__ = 'nixCraft'

parser = argparse.ArgumentParser(description='This is a demo script by nixCraft.')
parser.add_argument('-i','--input', help='Input file name',required=True)
parser.add_argument('-o','--output',help='Output file name', required=True)
args = parser.parse_args()

## show values ##
print ("Input file: %s" % args.input )
print ("Output file: %s" % args.output )
```

Save and close the file. Run it as follows:

```
$ chmod +x demo1.py
$ ./demo1.py
usage: demo1.py [-h] -i INPUT -o OUTPUT
demo1.py: error: argument -i/--input is required
$ ./demo1.py -h
usage: demo1.py [-h] -i INPUT -o OUTPUT

This is a demo script by nixCraft.

optional arguments:
  -h, --help            show this help message and exit
  -i INPUT, --input INPUT
                        Input file name
  -o OUTPUT, --output OUTPUT
                        Output file name
$ ./demo1.py -i input.txt -o output.txt
Input file: input.txt
Output file: output.txt
$ ./demo1.py -i input.txt
usage: demo1.py [-h] -i INPUT -o OUTPUT
demo1.py: error: argument -o/--output is required
$ ./demo1.py -i input.txt -o output.txt -z
usage: demo1.py [-h] -i INPUT -o OUTPUT
demo1.py: error: unrecognized arguments: -z
$ ./demo1.py --input input.txt --output output.txt
Input file: input.txt
Output file: output.txt
```

References:

1. Python argparse [10] module and getopt [11] module.

URLs in this post:

[1] Image: **http://www.cyberciti.biz/faq/category/python/**
[2] Command line parameters: **#s1**
[3] getopt module: **#s2**
[4] Parsing command line arguments: **#s3**
[5] argparse module: **#s4**
[6] Examples: **#s5**
[7] References: **#s6**

[8] Intermediate: **http://www.cyberciti.biz/faq/tutorial-difficulty-level/intermediate/**
[9] rss: **http://www.cyberciti.biz/faq/tutorial-difficulty-level/intermediate/feed/**
[10] argparse: **http://docs.python.org/2/library/argparse.html**
[11] getopt: **http://docs.python.org/2/library/getopt.html**