Python 3 Subprocess Examples

Last updated: 23 Nov 2019

Table of Contents

> *All examples use Python 3.5 or later (unless noted) and assume you're running Linux or a unix-based OS.*
>
> *All examples can be found **on this Jupyter notebook***

call() example

> *Use `run()` instead on Python v3.5+*

With suprocess.call() you pass an array of commands and parameters.

`subprocess.call()` returns the **return code** of the called process.

```
import subprocess

subprocess.call(["ls", "-lha"])
# >>> 0 (the return code)
```

» *subprocess.call() does **not raise an exception** if the underlying process errors!* «

```
import subprocess

# no Python Exception is thrown!
subprocess.call(["./bash-script-with-bad-syntax"])
# >>> 127
```

call() example using shell=True

» *Use `run()` instead on Python v3.5+* «

If `shell=True`, the command string is interpreted as a raw shell command.

Using `shell=True` may expose you to code injection if you use user input to build the command string.

```
subprocess.call("ls -lha", shell=True)
# returns 0 (the return code)
```

call() example, capture stdout and stderr

» *If you are on Python 3.5+, use `subprocess.run()` instead as it's safer.* «

```
import subprocess
import sys

# create two files to hold the output and errors, respective
with open('out.txt','w+') as fout:
    with open('err.txt','w+') as ferr:
        out=subprocess.call(["ls",'-lha'],stdout=fout,stderr
        # reset file to read from it
        fout.seek(0)
        # save output (if any) in variable
        output=fout.read())

        # reset file to read from it
        ferr.seek(0)
        # save errors (if any) in variable
        errors = ferr.read()

output
# total 20K
# drwxrwxr-x  3 felipe felipe 4,0K Nov  4 15:28 .
# drwxrwxr-x 39 felipe felipe 4,0K Nov  3 18:31 ..
# drwxrwxr-x  2 felipe felipe 4,0K Nov  3 19:32 .ipynb_check
# -rw-rw-r--  1 felipe felipe 5,5K Nov  4 15:28 main.ipynb

errors
# '' empty string
```

call() example, force exception if process causes error

Use `subprocess.check_call()`

```
import subprocess

# unlike subprocess.call, this throws a CalledProcessError
# if the underlying process errors out
subprocess.check_call(["./bash-script-with-bad-syntax"])
```

Run command and capture output

> *Using `universal_newlines=True` converts the output to a string instead of a byte array.*

- **Python version 2.7 -> 3.4**

```
import subprocess

# errors in the created process are raised here too
```

```
output = subprocess.check_output(["ls","-lha"],universal_n

output
# total 20K
# drwxrwxr-x  3 felipe felipe 4,0K Nov  4 15:28 .
# drwxrwxr-x 39 felipe felipe 4,0K Nov  3 18:31 ..
# drwxrwxr-x  2 felipe felipe 4,0K Nov  3 19:32 .ipynb_che
# -rw-rw-r--  1 felipe felipe 5,5K Nov  4 15:28 main.ipynb
```

- **Python version 3.5+**

```
import subprocess

# run() returns a CompletedProcess object if it was succes
# errors in the created process are raised here too
process = subprocess.run(['ls','-lha'], check=True, stdout
output = process.stdout

output
# total 20K
# drwxrwxr-x  3 felipe felipe 4,0K Nov  4 15:28 .
# drwxrwxr-x 39 felipe felipe 4,0K Nov  3 18:31 ..
# drwxrwxr-x  2 felipe felipe 4,0K Nov  3 19:32 .ipynb_che
# -rw-rw-r--  1 felipe felipe 5,5K Nov  4 15:28 main.ipynb
```

Run raw string as a shell command line

> *Don't do this if your string uses user input, as they may inject arbitrary code!*

This is similar to the example above, with `shell=True`

- **Python version 2.7 -> 3.4**

```
import subprocess

# errors in the created process are raised here too
output = subprocess.check_output("ls -lha", shell=True, un

output
# total 20K
# drwxrwxr-x  3 felipe felipe 4,0K Nov  4 15:28 .
# drwxrwxr-x 39 felipe felipe 4,0K Nov  3 18:31 ..
# drwxrwxr-x  2 felipe felipe 4,0K Nov  3 19:32 .ipynb_che
# -rw-rw-r--  1 felipe felipe 5,5K Nov  4 15:28 main.ipynb
```

- **Python version 3.5+**

```
import subprocess

# run() returns a CompletedProcess object if it was succes
# errors in the created process are raised here too
process = subprocess.run('ls -lha', shell=True, check=True
output = process.stdout

output
# total 20K
# drwxrwxr-x  3 felipe felipe 4,0K Nov  4 15:28 .
# drwxrwxr-x 39 felipe felipe 4,0K Nov  3 18:31 ..
# drwxrwxr-x  2 felipe felipe 4,0K Nov  3 19:32 .ipynb_che
# -rw-rw-r--  1 felipe felipe 5,5K Nov  4 15:28 main.ipynb
```

run() example: run command and get return code

`run()` behaves mostly the same way as `call()` and you should use it instead of call() for version 3.5 onwards.

> *subprocess.run() does **not raise an exception** if the underlying process errors!*

```
import subprocess

cp = subprocess.run(["ls","-lha"])

cp
# CompletedProcess(args=['ls', '-lha'], returncode=0)
```

run() example: run command, force exception if underlying process errors

Use `check=True` to force the Python method to throw an exception if the underlying process encounters errors:

```
import subprocess

subprocess.run(["ls","foo bar"], check=True)
# -------------------------------------------------------------
# CalledProcessError                        Traceback (most recent c
# ----> 1 subprocess.run(["ls","foo bar"], check=True)
# /usr/lib/python3.6/subprocess.py in run(input, timeout, ch
#     416         if check and retcode:
#     417             raise CalledProcessError(retcode, proc
# --> 418                                      output=stdout
#     419     return CompletedProcess(process.args, retcode,
#     420
# CalledProcessError: Command '['ls', 'foo bar']' returned n
```

run() example: using shell=True

As in the call() example, `shell=True`, the command string is interpreted as a raw shell command.

Again, Using `shell=True` may expose you to code injection if you use user input to build the command string.

```
import subprocess

cp = subprocess.run(["ls -lha"],shell=True)

cp
# CompletedProcess(args=['ls -lha'], returncode=0)
```

run() example: store output and error message in string

If the underlying process returns a nonzero exit code, you will **not get an exception**; the error message can be accessed via the `stderr` attribute in the `CompletedProcess` object.

- case 1: process return 0 exit code

```
import subprocess

cp = subprocess.run(["ls","-lha"], universal_newlines=True

cp.stdout
# total 20K
# drwxrwxr-x  3 felipe felipe 4,0K Nov  4 15:28 .
# drwxrwxr-x 39 felipe felipe 4,0K Nov  3 18:31 ..
# drwxrwxr-x  2 felipe felipe 4,0K Nov  3 19:32 .ipynb_che
# -rw-rw-r--  1 felipe felipe 5,5K Nov  4 15:28 main.ipynb
cp.stderr
# '' (empty string)
cp.returncode
# 0
```

- case 2: process returns nonzero exit code

```
import subprocess

cp = subprocess.run(["ls","foo bar"], universal_newlines=T

cp.output
# '' (empty string)
cp.stderr
# ls: cannot access 'foo bar': No such file or directory
cp.returncode
# 2
```

- case 3: other OS-level errors

  this case will throw an exception no matter what. For example, if you call an executable that doesn't exist. This throws an exception because it wasn't that the subprocess had an error - it never got created in the first place.

```
import subprocess

try:
    cp = subprocess.run(["xxxx","foo bar"], universal_newl
except FileNotFoundError as e:
    print(e)
    # [Errno 2] No such file or directory: 'xxxx'
```

Popen example: run command and get return code

subprocess.Popen() is used for more complex examples where you need. See Popen() vs call() vs run()

> *This causes the python program to block until the subprocess returns.*

```
from subprocess import Popen

p = Popen(["ls","-lha"])
p.wait()
# 0
```

Popen example: Store the output and error messages in a string

```
import subprocess
from subprocess import Popen

p = Popen(["ls","-lha"], stdout=subprocess.PIPE, stderr=subp

output, errors = p.communicate()

```

```
output
# total 20K
# drwxrwxr-x  3 felipe felipe 4,0K Nov  4 15:28 .
# drwxrwxr-x 39 felipe felipe 4,0K Nov  3 18:31 ..
# drwxrwxr-x  2 felipe felipe 4,0K Nov  3 19:32 .ipynb_check
# -rw-rw-r--  1 felipe felipe 5,5K Nov  4 15:28 main.ipynb

errors
# '' (empty string)
```

Popen example: Redirect output to file

```
import subprocess
from subprocess import Popen

path_to_output_file = '/tmp/myoutput.txt'

myoutput = open(path_to_output_file,'w+')

p = Popen(["ls","-lha"], stdout=myoutput, stderr=subprocess.
output, errors = p.communicate()

output
# there's nothing here because we didn't set stdout=subproce

errors
# '' empty string

# stdout has been written to this file
with open(path_to_output_file,"r") as f:
    print(f.read())

# total 20K
# drwxrwxr-x  3 felipe felipe 4,0K Nov  4 17:00 .
# drwxrwxr-x 39 felipe felipe 4,0K Nov  3 18:31 ..
# drwxrwxr-x  2 felipe felipe 4,0K Nov  3 19:32 .ipynb_check
# -rw-rw-r--  1 felipe felipe 7,7K Nov  4 17:00 main.ipynb
```

Popen example: Redirect output and errors to the same

file

```
import subprocess
from subprocess import Popen

path_to_output_file = '/tmp/myoutput.txt'

myoutput = open(path_to_output_file,'w+')

# file 'foo bar' doesn't exist
p = Popen(["ls","foo bar"], stdout=myoutput, stderr=myoutput

output, errors = p.communicate()

output
# there's nothing here because we didn't send stdout to subp

errors
# there's nothing here either

# stdout and stderr have been written to this file
with open(path_to_output_file,"r") as f:
    print(f.read())
```

```
# ls: cannot access 'foo bar': No such file or directory
```

Popen example: Run command in the background

By default, calls to `Popen()` spawn a subprocess in the background and don't wait for it to terminate (unless you use `wait()` on the Popen object).

Pipe commands together

Use `Popen`:

```
from subprocess import Popen,PIPE

# this is equivalent to ls -lha | grep "foo bar"
p1 = Popen(["ls","-lha"], stdout=PIPE)
p2 = Popen(["grep", "foo bar"], stdin=p1.stdout, stdout=PIPE
p1.stdout.close()

output = p2.communicate()[0]
```

Wait for command to terminate, asynchronously

Use asyncio and **await**.

Method `asyncio.create_subprocess_exec()` works much the same way as `Popen()` but calling `wait()` and `communicate()` on the returned objects **does not block the processor**, so the Python interpreter can be used in other things while the external subprocess doesn't return.

» *Python 3.6+ is needed here* «

```
import asyncio

proc = await asyncio.create_subprocess_exec(
    'ls','-lha',
    stdout=asyncio.subprocess.PIPE,
    stderr=asyncio.subprocess.PIPE)


# if proc takes very long to complete, the CPUs are free to
# other processes
stdout, stderr = await proc.communicate()

proc.returncode
# 0

# must call decode because stdout is a bytes object
stdout.decode()
# total 24K
# drwxrwxr-x  3 felipe felipe 4,0K Nov  4 17:52 .
# drwxrwxr-x 39 felipe felipe 4,0K Nov  3 18:31 ..
# drwxrwxr-x  2 felipe felipe 4,0K Nov  3 19:32 .ipynb_check
# -rw-rw-r--  1 felipe felipe  11K Nov  4 17:52 main.ipynb

stderr.decode()
# ''  empty string
```

---

call() vs run()

As of Python version 3.5, `run()` should be used instead of `call()`.

- `run()` returns a `CompletedProcess` object instead of the process return code.
  - A `CompletedProcess` object has attributes like args, returncode, etc. subprocess.CompletedProcess
- other functions like `check_call()` and `check_output()` can all be replaced with `run()`.

Popen vs run() and call()

`call()` and `run()` are **convenience functions** and should be used for simpler cases.

`Popen()` is much more powerful and handles all cases, not just simple ones.

Felipe 03 Nov 2018 23 Nov 2019 python3

« Project Review: Text Classification of Legal Documents

Archive

Thoughts on Michelangelo: Uber's Machine Learning Platform »

Related content

Dialogue & Discussion