subprocess.run Documentation

Release 0.0.8

Sebastian Pawluś

November 16, 2013

Contents

Python's standard **subprocess** module provides most of the capabilities you need to run external processes from Python, but the API is thoroughly misleading. It requires you to check documentation every time when you are trying to do really basic things related to creating external processes.

The **subprocess.run** extension was create to run processes in a polite way.

CHAPTER 1

Walkthrough

```
$ > pip install subprocess.run
```

To install the package.

```
>>> from subprocess import run
```

If installation went successful, import **run** from **subprocess** module.

```
>>> run('uname -v').stdout
'#19-Ubuntu SMP Wed Oct 9 16:20:46 UTC 2013'
```

The standard output (successful command output) is available thought the **stdout** attribute.

```
>>> run('uname -v').status
0
```

You can access execution status through **status** attribute.

```
>>> run('rm not_existing_directory').stderr
'rm: cannot remove `not_existing_directory': No such file or directory'
```

If something went not so well, the error output is available thought the **stderr** attribute.

```
>>> run('ls -la', 'wc -l').stdout
14
```

If pipe is needed, just add more one after another. Example above is the same as **ls -la | wc -l**.

```
>>> run('ls -la', 'wc -l', 'wc -c').stdout
3
```

And more pipe.

```
>>> run('wc -c', data="test").stdout
```

And even more pipe. But this time, the call will take data from python script. This would be roughly something like **echo test | wc -c**

```
>>> run('ls -la').stdout.lines
['total 20',
'drwxrwxr-x 3 user user 4096 Dec 20 22:55 .',
'drwxrwxr-x 5 user user 4096 Dec 20 22:57 ..',
```

```
'drwxrwxr-x 2 user user 4096 Dec 20 22:37 dir',
'-rw-rw-r-- 1 user user    0 Dec 20 22:52 file']
```

To help with output processing, both **stdout** and **stderr** outputs are equipped with **lines** attribute, it will help with slicing your output to a list of strings.

```
>>> run('ls -la').stdout.qlines
[
   ['total 20']
   ['drwxrwxr-x', '3', 'user', 'user', '4096', 'Dec', '20', '22:55', '.'],
   ['drwxrwxr-x', '5', 'user', 'user', '4096', 'Dec', '20', '22:57', '..'],
   ['drwxrwxr-x', '2', 'user', 'user', '4096', 'Dec', '20', '22:37', 'dir'],
   ['-rw-rw-r--', '1', 'user', 'user', '0', 'Dec', '20', '22:52', 'file']
]
```

And with **qlines**, to split lines to words.

```
>>> run('ls -la', 'wc -l', 'wc -c').chain
```

Will return list of executed calls. Then let's do something that actually make sens

```
>>> [call.status for call in run('ls -la', 'wc -l', 'wc -c').chain]
[0, 0, 0]
```

This will return list of statuses inside **chain**.

```
>>> [call.stdout for call in run('ls -la', 'wc -l', 'wc -c').chain]
['total 20\ndrwxrwxr-x 3 user user 4096 Dec 20 22:55 .\ndrwxrwxr-x 5 user user 4096 Dec 20 22:57 ..',
'95',
'2']
```

This will return list of stdouts for commands inside **chain**.

```
from subprocess import run

run('grep something', data=run.stdin)

$ ps aux | python script.py
```

To read from shell pipe.

CHAPTER 2

Status

The codebase is less than 100 LOC, feel free to look at it and explain to me why I should/shouldn't do things this way. Library seems to be pretty stable, feel free to use it as you want.

CHAPTER 3

Source Code

https://github.com/xando/subprocess.run

CHAPTER 4

Supported platforms

- Python2.6
- Python2.7
- Python3.3
- PyPy2.1

CHAPTER 5

Tests

```
>>> python setup.py test
```