Executing BASH from Python

mervine.net/executing-bash-from-python

Here's a summary of the ways to call external programs and the advantages and disadvantages of each:

1. `os.system("some_command with args")` passes the command and arguments to your system's shell. This is nice because you can actually run multiple commands at once in this manner and set up pipes and input/output redirection. For example, `os.system("some_command < input_file | another_command > output_file")` However, while this is convenient, you have to manually handle the escaping of shell characters such as spaces, etc. On the other hand, this also lets you run commands which are simply shell commands and not actually external programs. http://docs.python.org/lib/os-process.html

2. `stream = os.popen("some_command with args")` will do the same thing as os.system except that it gives you a file-like object that you can use to access standard input/output for that process. There are 3 other variants of popen that all handle the i/o slightly differently. If you pass everything as a string, then your command is passed to the shell; if you pass them as a list then you don't need to worry about escaping anything. http://docs.python.org/lib/os-newstreams.html

3. The Popen class of the subprocess module. This is intended as a replacement for os.popen but has the downside of being slightly more complicated by virtue of being so comprehensive. For example, you'd say `print Popen("echo Hello World", stdout=PIPE, shell=True).stdout.read()` instead of `print os.popen("echo Hello World").read()` but it is nice to have all of the options there in one unified class instead of 4 different popen functions. http://docs.python.org/lib/node528.html

4. The call function from the subprocess module. This is basically just like the Popen class and takes all of the same arguments, but it simply wait until the command completes and gives you the return code. For example: `return_code = call("echo Hello World", shell=True)` http://docs.python.org/lib/node529.html

5. The os module also has all of the fork/exec/spawn functions that you'd have in a C program, but I don't recommend using them directly.

In addition the to five above, there's one more worth noting. pexpect, which is a python implementation of Expect. Here's a simple example:

```
import pexect
child = pexpect("echo \"foo\")
child.expect("foo", timeout=10)
child.sendline("echo \"bar\")
child.expect("bar", timeout=10)
child.interact()
```

I just found `sh`, which is awesome! It allows you to execute most shell commands as native python fuctions.

Example:

```
import sh
sh.cd('/path/to/Development')
print(sh.pwd())
```
