Subprocess Library All Versions

Examples

More flexibility with Popen

Using `subprocess.Popen` give more fine-grained control over launched processes than `subprocess.call` .

Launching a subprocess

```
process = subprocess.Popen([r'C:\path\to\app.exe', 'arg1', '--flag', 'arg'])
```

The signature for `Popen` is very similar to the `call` function; however, `Popen` will return immediately instead of waiting for the subprocess to complete like `call` does.

Waiting on a subprocess to complete

```
process = subprocess.Popen([r'C:\path\to\app.exe', 'arg1', '--flag', 'arg'])
process.wait()
```

Reading output from a subprocess

```
process = subprocess.Popen([r'C:\path\to\app.exe'], stdout=subprocess.PIPE, stderr=subprocess.PI

# This will block until process completes
stdout, stderr = process.communicate()
print stdout
print stderr
```

Interactive access to running subprocesses

You can read and write on `stdin` and `stdout` even while the subprocess hasn't completed. This could be useful when automating functionality in another program.

Writing to a subprocess

```
process = subprocess.Popen([r'C:\path\to\app.exe'], stdout = subprocess.PIPE, stdin = subprocess


process.stdin.write('line of input\n') # Write input

line  = process.stdout.readline() # Read a line from stdout

# Do logic on line read.
```

However, if you only need one set of input and output, rather than dynamic interaction, you should use `communicate()` rather than directly accessing `stdin` and `stdout` .

Reading a stream from a subprocess

In case you want to see the output of a subprocess line by line, you can use the following snippet:

```
process = subprocess.Popen(<your_command>, stdout=subprocess.PIPE)
while process.poll() is None:
    output_line = process.stdout.readline()
```

in the case the subcommand output do not have EOL character, the above snippet does not work. You can then read the output character by character as follows:

```
process = subprocess.Popen(<your_command>, stdout=subprocess.PIPE)
while process.poll() is None:
    output_line = process.stdout.read(1)
```

The `1` specified as argument to the `read` method tells read to read 1 character at time. You can specify to read as many characters you want using a different number. Negative number or 0 tells to `read` to read as a single string until the EOF is encountered (see here).

In both the above snippets, the `process.poll()` is `None` until the subprocess finishes. This is used to exit the loop once there is no more output to read.

The same procedure could be applied to the `stderr` of the subprocess.

Calling External Commands

The simplest use case is using the subprocess.call function. It accepts a list as the first argument. The first item in the list should be the external application you want to call. The other items in the list are arguments that will be passed to that application.

```
subprocess.call([r'C:\path\to\app.exe', 'arg1', '--flag', 'arg'])
```

For shell commands, set shell=True and provide the command as a string instead of a list.

```
subprocess.call('echo "Hello, world"', shell=True)
```

Note that the two command above return only the exit status of the subprocess. Moreover, pay attention when using shell=True since it provides security issues (see here).

If you want to be able to get the standard output of the subprocess, then substitute the subprocess.call with subprocess.check_output . For more advanced use, refer to this .

How to create the command list argument

The subprocess method that allows running commands needs the command in form of a list (at least using shell_mode=True).

The rules to create the list are not always straightforward to follow, especially with complex commands. Fortunately, there is a very helpful tool that allows doing that: shlex . The easiest way of creating the list to be used as command is the following:

```
import shlex
cmd_to_subprocess = shlex.split(command_used_in_the_shell)
```

A simple example:

```
import shlex
shlex.split('ls --color -l -t -r')

out: ['ls', '--color', '-l', '-t', '-r']
```

Syntax

```
subprocess.call(args, *, stdin=None, stdout=None, stderr=None, shell=False, timeout=None)
```

```
subprocess.Popen(args, bufsize=-1, executable=None, stdin=None, stdout=None, stderr=None,
preexec_fn=None, close_fds=True, shell=False, cwd=None, env=None,
universal_newlines=False, startupinfo=None, creationflags=0, restore_signals=True,
start_new_session=False, pass_fds=())
```

Parameters

Parameter	Details
args	A single executable, or sequence of executable and arguments - 'ls' , ['ls', '-la']
shell	Run under a shell? The default shell to /bin/sh on POSIX
cwd	Working directory of the child process.

Remarks