EasyProcess Documentation

Release 0.0.7

ponty

April 10, 2011

CONTENTS

EasyProcess is an easy to use python `subprocess` interface.

Date: April 10, 2011

Contents:

EasyProcess is an easy to use python subprocess interface.

home: https://github.com/ponty/EasyProcess

html documentation: http://ponty.github.com/EasyProcess

pdf documentation: https://github.com/ponty/EasyProcess/raw/master/docs/_build/latex/EasyProcess.pdf

Features:

- layer on top of subprocess module
- easy to start, stop programs
- easy to get standard output/error, return code of programs
- command can be list or string
- logging
- timeout
- unittests
- crossplatform, development on linux

Known problems:

- shell is not supported
- large stdout/stderr was not tested, maybe not efficent
- stdout/stderr is set only after the subprocess has finished
- stop() does not kill whole subprocess tree

CHAPTER

ONE

BASIC USAGE

```
>>> from easyprocess import EasyProcess
>>> EasyProcess('echo hello').call().stdout
'hello'
```

CHAPTER

TWO

INSTALLATION

2.1 General

- install setuptools or pip
- install the program:

if you have setuptools installed:

```
# as root
easy_install EasyProcess
```

if you have pip installed:

```
# as root
pip install EasyProcess
```

2.2 Ubuntu

```
sudo apt-get install python-setuptools
sudo easy_install EasyProcess
```

2.3 Uninstall

```
# as root
pip uninstall EasyProcess
```

CHAPTER

THREE

USAGE

```
>>> from easyprocess import EasyProcess
>>> # Run program, wait for it to complete, get stdout (command is string):
>>> EasyProcess('echo hello').call().stdout
'hello'
>>> # Run program, wait for it to complete, get stdout (command is list):
>>> EasyProcess(['echo','hello']).call().stdout
'hello'
>>> # Run program, wait for it to complete, get stderr:
>>> EasyProcess('python --version').call().stderr
'Python 2.6.6'
>>> # Run program, wait for it to complete, get return code:
>>> EasyProcess('python --version').call().return_code
0
>>> # Run program, wait 1 second, stop it, get stdout:
>>> print EasyProcess('ping localhost').start().sleep(1).stop().stdout
PING localhost.localdomain (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost.localdomain (127.0.0.1): icmp_req=1 ttl=64 time=0.032 ms
64 bytes from localhost.localdomain (127.0.0.1): icmp_req=2 ttl=64 time=0.031 ms
>>> # Run program, wait for it to complete, check for errors:
>>> EasyProcess('ls').check()
<Proc cmd_param=ls alias=None cmd=['ls'] (ls) oserror=None returncode=0 stdout="build
dist
docs
easyprocess
EasyProcess.egg-info
LICENSE.txt
MANIFEST.in
nosetests.xml
pavement.py
paver-minilib.zip
README.rst
setup.py
temp
tests
TODO" stderr="">
```

Exceptions in check:

```
>>> EasyProcess('bad_command').check()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "easyprocess.py", line 84, in check
    raise EasyProcessCheckError(self)
easyprocess.EasyProcessCheckError: EasyProcess check failed!
```

```
 OSError:[Errno 2] No such file or directory
 cmd:['bad_command']
 return code:None
 stderr:None
>>> EasyProcess('sh -c bad_command').check()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "easyprocess.py", line 84, in check
    raise EasyProcessCheckError(self)
easyprocess.EasyProcessCheckError: EasyProcess check failed!
 OSError:None
 cmd:['sh', '-c', 'bad_command']
 return code:127
 stderr:sh: bad_command: not found
```

3.1 Timeout

```
>>> from easyprocess import EasyProcess
>>> # Run ping with  timeout
>>> print EasyProcess('ping localhost').call(timeout=1).stdout
PING localhost.localdomain (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost.localdomain (127.0.0.1): icmp_req=1 ttl=64 time=0.028 ms
64 bytes from localhost.localdomain (127.0.0.1): icmp_req=2 ttl=64 time=0.029 ms
```

CHAPTER

FOUR

LOGGING

Example program:

```
from easyprocess import EasyProcess
import logging

# turn on logging
logging.basicConfig(level=logging.DEBUG)

EasyProcess('echo hello').call()
EasyProcess('python --version').call()
EasyProcess('ping localhost').start().sleep(1).stop()
EasyProcess('python --version').check()
try:
    EasyProcess('bad_command').check()
except Exception, detail:
    print  detail

try:
    EasyProcess('sh -c bad_command').check()
except Exception, detail:
    print  detail
```

Output:

```
$ python -m easyprocess.examples.log
DEBUG:easyprocess:param: "echo hello" command: ['echo', 'hello'] ("echo hello")
DEBUG:easyprocess:reading config: /home/titi/.easyprocess.cfg
DEBUG:easyprocess:process was started (pid=27464)
DEBUG:easyprocess:Thread started
DEBUG:easyprocess:process has ended
DEBUG:easyprocess:return code=0
DEBUG:easyprocess:stdout=hello
DEBUG:easyprocess:stderr=
DEBUG:easyprocess:Thread finished
DEBUG:easyprocess:param: "python --version" command: ['python', '--version'] ("python --version")
DEBUG:easyprocess:process was started (pid=27466)
DEBUG:easyprocess:Thread started
DEBUG:easyprocess:process has ended
DEBUG:easyprocess:return code=0
DEBUG:easyprocess:stdout=
DEBUG:easyprocess:stderr=Python 2.6.6
DEBUG:easyprocess:Thread finished
DEBUG:easyprocess:param: "ping localhost" command: ['ping', 'localhost'] ("ping localhost")
DEBUG:easyprocess:process was started (pid=27468)
```

```
DEBUG:easyprocess:Thread started
DEBUG:easyprocess:stopping process (pid=27468 cmd="['ping', 'localhost']")
DEBUG:easyprocess:process is active -> sending SIGTERM
DEBUG:easyprocess:process has ended
DEBUG:easyprocess:return code=-15
DEBUG:easyprocess:stdout=PING localhost.localdomain (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost.localdomain (127.0.0.1): icmp_req=1 ttl=64 time=0.028 ms
64 bytes from localhost.localdomain (127.0.0.1): icmp_req=[rest of output was removed, max_bytes_to_
DEBUG:easyprocess:stderr=
DEBUG:easyprocess:Thread finished
DEBUG:easyprocess:param: "python --version" command: ['python', '--version'] ("python --version")
DEBUG:easyprocess:process was started (pid=27470)
DEBUG:easyprocess:Thread started
DEBUG:easyprocess:process has ended
DEBUG:easyprocess:return code=0
DEBUG:easyprocess:stdout=
DEBUG:easyprocess:stderr=Python 2.6.6
DEBUG:easyprocess:Thread finished
DEBUG:easyprocess:param: "bad_command" command: ['bad_command'] ("bad_command")
DEBUG:easyprocess:OSError exception:[Errno 2] No such file or directory
DEBUG:easyprocess:param: "sh -c bad_command" command: ['sh', '-c', 'bad_command'] ("sh -c bad_comman
DEBUG:easyprocess:process was started (pid=27473)
DEBUG:easyprocess:Thread started
DEBUG:easyprocess:process has ended
DEBUG:easyprocess:return code=127
DEBUG:easyprocess:stdout=
DEBUG:easyprocess:stderr=sh: bad_command: not found
DEBUG:easyprocess:Thread finished
start error <Proc cmd_param=bad_command alias=None cmd=['bad_command'] (bad_command) oserror=[Errno 2
check error, return code is not zero! <Proc cmd_param=sh -c bad_command alias=None cmd=['sh', '-c', '
```

CHAPTER

FIVE

REPLACING EXISTING FUNCTIONS

Replacing os.system:

```
retcode = os.system("ls -l")
==>
p = EasyProcess("ls -l").call()
retcode = p.return_code
print p.stdout
```

Replacing subprocess.call:

```
retcode = subprocess.call(["ls", "-l"])
==>
p = EasyProcess(["ls", "-l"]).call()
retcode = p.return_code
print p.stdout
```

CHAPTER

SIX

API

easyprocess.**EasyProcess**
alias of Proc

class easyprocess.**Proc**(*cmd*, *ubuntu_package=None*, *url=None*, *max_bytes_to_log=200*)
simple interface for subprocess

shell is not supported (shell=False)

call(*timeout=None*)
Run command with arguments. Wait for command to complete.

Return type self

check(*return_code=0*)
Run command with arguments. Wait for command to complete. If the exit code was as expected and there is no exception then return, otherwise raise EasyProcessError.

Parameters **return_code** – int, expected return code

Return type self

check_installed()
Used for testing if program is installed.

Run command with arguments. Wait for command to complete. If OSError raised, then raise EasyProcessCheckInstalledError with information about program installation

Parameters **return_code** – int, expected return code

Return type self

is_alive()
poll process (subprocess.Popen.poll())

Return type bool

pid
PID (subprocess.Popen.pid)

Return type int

return_code
returncode (subprocess.Popen.returncode)

Return type int

sendstop()
Kill process by sending SIGTERM. Do not wait for command to complete.

Return type self

sleep(*sec*)
sleeping (same as `time.sleep()`)

Return type self

start()
start command in background and does not wait for it

Return type self

stop()
Kill process by sending SIGTERM. and wait for command to complete.

same as `sendstop().wait()`

Return type self

wait(*timeout=None*)
Wait for command to complete.

Return type self

wrap(*callable*, *delay=0*)
returns a function which: 1. start process 2. call callable, save result 3. stop process 4. returns result

Return type

CHAPTER

SEVEN

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

INDEX