Os Command Py Documentation

Samuel Murail

May 02, 2020

Table of Contents:

Os_Command_py is a python library allowing a simplified use of the OS commands.

- **Online Documentation:** https://os-command-py.readthedocs.io/en/latest/
- **Source code repository:** https://github.com/samuelmurail/os_command_py

CHAPTER 1

Main features:

- **Basic OS operation**
  - directory creation
  - change working path
  - delete files and directories
- Command simplied use with subprocess

CHAPTER 2

Installation

2.1 Using Pypi

```
pip3 install os_command_py
```

2.2 Using Conda

```
conda install os_command_py -c conda-forge
```

2.3 From source code

Get the os_command_py library from github.

```
git clone https://github.com/samuelmurail/os_command_py.git
./setup.py install --user
```

CHAPTER 3

Author

- Samuel Murail, Associate Professor - Université Paris Diderot, CMPLI.

See also the list of contributors who participated in this project.

CHAPTER 4

License

This project is licensed under the GNU General Public License v2.0 - see the `LICENSE` file for details.

CHAPTER 5

Installation

5.1 1. Using conda

Use the conda forge channel to install *os_command_py*.

```
$ conda install os_command_py -c conda-forge
```

5.2 2. Using Pypi

Use pip to install *os_command_py*.

```
$ pip install os_command_py
```

5.3 3. Install os_command_py from source

The sources for Docking Python can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/samuelmurail/os_command_py
```

Or download the tarball:

```
$ curl -OJL https://github.com/samuelmurail/os_command_py/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

5.4 4. Test Installation

To test the installation, simply use `pytest`:

```
$ pytest

============================== test session starts ==============================
platform darwin -- Python 3.8.2, pytest-5.4.1, py-1.8.1, pluggy-0.13.1
rootdir: /Users/smurail/Documents/Code/os_command_py, inifile: pytest.ini
plugins: cov-2.8.1
collected 8 items

os_command_py/os_command.py ........                                    [100%]

=============================== 8 passed in 1.18s ===============================
```

CHAPTER 6

Usage

6.1 Simple Command

You can use the `os_command` module to create a `Command` object, which takes a list as input, here an example with a simple `ls` command:

```
>>> cmd_list = ['ls', '-a', '.']
>>> cmd_test = Command(list_cmd=cmd_list)
```

You can use the `display()` function to print the command:

```
>>> cmd_test.display()
ls -a .
```

An the the `run()` function to excectute it, the return code will be return in a dictionnary:

```
>>> return_code = cmd_test.run(out_data=True)
>>> print(return_code['stdout'])
.
..
1y0m.pdb
volume.xvg
```

Here is another example with the command `wc`:

```
>>> cmd_list = ['wc', './1y0m.pdb']
>>> cmd_test_2 = Command(list_cmd=cmd_list)
>>> cmd_test_2.display()
wc ./1y0m.pdb
>>> return_code = cmd_test_2.run(out_data=True)
>>> print('Number of line = {}  word = {} char = {}'.format(
... *return_code['stdout'].split()[:3]))
Number of line = 1627  word = 18466 char = 13...
```

6.2 Defining environment

The `define_env` can also be used on the `Command` object to change an environmnent variable:

```
>>> print(os.getenv('USELESS'))
None
>>> cmd_list = ['ls', '-lsh']
>>> cmd_test = Command(list_cmd=cmd_list)
>>> cmd_test.display()
ls -lsh
>>> # Here we define the $USELESS env. variable:
>>> cmd_test.define_env(os.environ.update({'USELESS': 'Changed'}))
>>> return_code = cmd_test.run(out_data=True)
>>> print(os.getenv('USELESS'))
Changed
```

6.3 Run a job in background

A command can also be ran in background using `run_background()`. It is permitting to run a parallel function, to *eg.* survey the command output. The function working in parallel has to be define in the `monitor` dictionnary with `'function'` as key, everyother keys of `monitor` can be used for the `monitor` function.

Here we run a bash command `sleep` for 1 second, and use a python function `function_iter()` that print out `"X"` every tenth of a seconds:

```
>>> import time
>>>
>>> # Create the function that will run while the command is running
>>>
>>> def function_iter(proc, dict):
...     while proc.poll() is None:
...         time.sleep(dict['refresh_time'])
...         print('X', end='')
>>>
>>> monitor = {'function': function_iter,
...            'refresh_time': 0.1}
>>>
>>> # Create the command
>>>
>>> cmd_list = ['sleep', '1']
>>> background_test = Command(list_cmd=cmd_list)
>>> background_test.display()
sleep 1
>>> return_code = background_test.run_background(monitor,
... display=True)
XXXXXXXXX
None
```

CHAPTER 7

Os Command class

class os_command_py.os_command.**Command**(*list_cmd*, *my_env=None*, *prefix='-'*, ***kwargs*)

The Command class is a way to launch bash command and mainly gromacs

Parameters

- **cmd** (*list*) – command list
- **env** (*dict*) – environment variable
- **prefix** (*str, opional, default "-"*) – argument prefix

Example

```
>>> # Simple ls command
>>>
>>> cmd_list = ['ls', '-a', TEST_PATH]
>>> cmd_test = Command(list_cmd=cmd_list)
>>> cmd_test.display() #doctest: +ELLIPSIS
ls -a ...input
>>> return_code = cmd_test.run(out_data=True)
>>> print(return_code['stdout']) #doctest: +ELLIPSIS
.
..
1y0m.pdb
volume.xvg
<BLANKLINE>
>>> # Not Working command:
>>>
>>> cmd_list = ['ls', '-a', '/NON_EXISTING_FILE']
>>> cmd_test = Command(list_cmd=cmd_list)
>>> cmd_test.display() #doctest: +ELLIPSIS
ls -a ...NON_EXISTING_FILE
>>> try:
...     cmd_test.run()
... except RuntimeError:
...     print('Command failed') #doctest: +ELLIPSIS
```

(continues on next page)

(continued from previous page)

```
The following command could not be executed correctly :
ls -a ...NON_EXISTING_FILE
Command failed
>>> # Word Count
>>> cmd_list = ['wc', os.path.join(TEST_PATH,'1y0m.pdb')]
>>> cmd_test_2 = Command(list_cmd=cmd_list)
>>> cmd_test_2.display() #doctest: +ELLIPSIS
wc ...1y0m.pdb
>>> return_code = cmd_test_2.run(out_data=True)
>>> print('Number of line = {}  word = {} char = {}'.format(    *return_code[
↪'stdout'].split()[:3])) #doctest: +ELLIPSIS
Number of line = 1627  word = 18466 char = 13...
```

define_env(*my_env*)

Define the environment of the Command object.

Example

```
>>> print(os.getenv('USELESS'))
None
>>> cmd_list = ['ls', '-lsh']
>>> cmd_test = Command(list_cmd=cmd_list)
>>> cmd_test.display() #doctest: +ELLIPSIS
ls -lsh
>>> cmd_test.define_env(os.environ.update({'USELESS': 'Changed'}))
>>> return_code = cmd_test.run(out_data=True)
>>> print(os.getenv('USELESS'))
Changed
```

display()

Show Command object that will be launch. Show only the name of the command (*eg. gmx*) instead of the full path. Show relative path for files in the command.

display_raw()

Show Command object that will be launch. Show the full path of the command as well as the full path for files in the command.

run(*com_input=''*, *display=False*, *out_data=False*)

Launch Command object that will be launch. return programm output is *out_data* is set to *True*

Parameters

- **com_input** (*str*) – input for the command
- **display** (*bool*) – option to display output
- **out_data** (*bool*) – option to return output data

Returns Return Code or output dict

Return type bool/dict

run_background(*func_input_dict*, *com_input=''*, *display=False*, *out_data=False*)

Launch Command object that will be launch. Will the command is running launch the *function* using *func_input_dict* as argument. return programm output is *out_data* is set to *True*

Parameters

- **function** (*function*) – function to be launch
- **func_input_dict** (*dict*) – input for the function

- **com_input** (*str*) – input for the command
- **display** (*bool*) – option to display output
- **out_data** (*bool*) – option to return output data

Returns Return Code or output dict

Return type bool/dict

Example:

```
>>> import time
>>>
>>> # Create the function that will run while the command is running
>>> def function_iter(proc, dict):
...     while proc.poll() is None:
...         time.sleep(dict['refresh_time'])
...         print('X', end='')
>>>
>>> monitor = {'function': function_iter,
...            'refresh_time': 0.1}
>>>
>>> # Create the command
>>> cmd_list = ['sleep', '1']
>>> background_test = Command(list_cmd=cmd_list)
>>> background_test.display() #doctest: +ELLIPSIS
sleep 1
>>> return_code = background_test.run_background(monitor,
...                                              display=True)
↪#doctest: +ELLIPSIS
XXXXXX...sleep 1
None
<BLANKLINE>
<BLANKLINE>
```

CHAPTER 8

Os Command shorcuts

Collection of function related to os and sys operations.

os_command_py.os_command.**check_directory_exist**(*directory*)

Check is a directory exist.

Parameters **directory** (*str*) – directory path

Returns if the file exist

Return type bool

Example

```
>>> test_exist = check_directory_exist(TEST_PATH)
>>> print("Directory {} exist: {}".format(TEST_PATH, test_exist))    #doctest:␣
↪+ELLIPSIS
Directory ...input exist: True
>>> test_exist = check_directory_exist(os.path.join(TEST_PATH,'no_way'))
>>> print("Directory {} exist: {}".format(
... TEST_PATH+'/no_way', test_exist)) #doctest: +ELLIPSIS
Directory ...no_way exist: False
```

os_command_py.os_command.**check_file_and_create_path**(*file*)

Check if file exist and create path if not available

Parameters **file** (*str*) – file path

Returns File existance

Return type bool

os_command_py.os_command.**check_file_exist**(*file*)

Check is a file exist.

Parameters **file** (*str*) – file path

Returns if the file exist

Return type bool

Example

```
>>> test_exist = check_file_exist(os.path.join(TEST_PATH, "1y0m.pdb"))
>>> print("1y0m.pdb exist: {}".format(test_exist))
1y0m.pdb exist: True
```

os_command_py.os_command.**create_and_go_dir**(*dir_name*)
Create the path to a directory and change path in it.

Parameters **dir_name** – directorie name

Example

```
>>> TEST_OUT = str(getfixture('tmpdir'))
>>> start_dir = os.getcwd()
>>> create_and_go_dir(os.path.join(TEST_OUT, "tmp"))
>>> print("Path: {}".format(os.getcwd())) #doctest: +ELLIPSIS
Path: ...tmp
>>> os.chdir(start_dir)
```

os_command_py.os_command.**create_dir**(*dir_name*)
Create the path to a directory.

Parameters **dir_name** – directorie name

os_command_py.os_command.**delete_directory**(*directory*)
Delete a file.

Parameters **directory** (*str*) – directory path

Returns operation sucess

Return type bool

os_command_py.os_command.**delete_file**(*file*)
Delete a file.

Parameters **file** (*str*) – file path

Returns operation sucess

Return type bool

os_command_py.os_command.**full_path_and_check**(*file*)
Return the full path of a file

Parameters **file** (*str*) – file path

Returns File path

Return type str

os_command_py.os_command.**get_directory**(*file*)
Return the path of a file directory

Parameters **file** (*str*) – file path

Returns File path

Return type str

os_command_py.os_command.**get_gmx_version**()
Get gromacs version of mdrun.

Example

```
print('Version is {}'.format(get_gmx_version()))
Version is 2016.4
```

os_command_py.os_command.**is_exe**(*fpath*)

Check is a file path exist and if user has access to it

Parameters **fpath** (*str*) – file path

Returns if the file is an executable

Return type bool

Example

```
>>> ls_path = which('ls')
>>> print(is_exe(ls_path))
True
```

os_command_py.os_command.**which**(**program_list*)

find and return the first path of a program if find. Look for all combination within the *$PATH* env variable

Parameters **program_list** (*list of str*) – list of program name

Returns path of the program

Return type str

Example

```
>>> ls_path = which('dontexist', 'ls')
>>> print(ls_path.find('ls') != -1) #doctest: +ELLIPSIS
True
```

..note:

Consider using shutil.which(program) for windows.

CHAPTER 9

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

9.1 Types of Contributions

9.1.1 Report Bugs

Report bugs at https://github.com/samuelmurail/os_command_py/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

9.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” and “help wanted” is open to whoever wants to implement it.

9.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with “enhancement” and “help wanted” is open to whoever wants to implement it.

9.1.4 Write Documentation

Docking Python could always use more documentation, whether as part of the official Docking Python docs, in docstrings, or even on the web in blog posts, articles, and such.

9.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/samuelmurail/os_command_py/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

9.2 Get Started!

Ready to contribute? Here's how to set up *os_command_py* for local development.

1. Fork the *os_command_py* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/os_command_py.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv os_command_py
$ cd os_command_py/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 os_command_py tests
$ python setup.py test or pytest
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

9.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.5, 3.6, 3.7 and 3.8, and for PyPy. Check https://travis-ci.com/samuelmurail/os_command_py/pull_requests and make sure that the tests pass for all supported Python versions.

9.4 Tips

To run a subset of tests:

```
$ pytest tests.test_os_command_py
```

9.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 10

Credits

10.1 Development Lead

- Samuel Murail, Université de Paris <samuel.murail@u-paris.fr>

We are open to any contribution.

CHAPTER 11

Indices and tables

- genindex
- modindex
- search

Python Module Index

o

Index