clom Documentation

Release 0.7

Mike Thornton

October 13, 2014

CHAPTER 1

About

Clom is a Python Command Line Object Mapper. It's intended to make generating commands and escaping arguments easier for command line interaction. It's particularly useful when used with Fabric or `subprocess`.

- Clom on GitHub
- Clom on Pypi

CHAPTER 2

Installation

Install with `pip` or `easy_install`.

```
pip install clom
```

CHAPTER 3

Usage Examples

Import:

```
>>> from clom import clom
```

Build a command:

```
>>> clom.echo("Don't test me")
"echo 'Don'\\''t test me'"
```

Augment with arguments:

```
>>> clom.ls.with_opts('-a', '-t', l=True).with_args('~/')
"ls -a -t -l '~/'"
>>> clom.curl('http://dev.host', X='POST', data='message=hello')
'curl -X POST --data message=hello http://dev.host'
```

Use sub commands:

```
>>> clom.git.checkout('mybranch')
'git checkout mybranch'
```

Execute with ease:

```
>>> clom.seq(5).shell.all()
['1', '2', '3', '4', '5']

>>> clom.seq.shell('5').first()
'1'
```

Iterate over results:

```
>>> for i in clom.seq(3).shell():
...     print(i)
...
1
2
3
```

Handle errors:

```
>>> clom.touch('/not/a/thing').shell()
Traceback (most recent call last):
  ...
CommandError: Error while executing "touch /not/a/thing" (1):
touch: cannot touch `/not/a/thing': No such file or directory
```

Group commands:

```
>>> from clom import AND, OR
>>> OR(clom.vagrant.up, clom.echo('Vagrant up failed'))
"( vagrant up || echo 'Vagrant up failed' )"
>>> OR(clom.vagrant.up, clom.echo('Vagrant up failed')).shell()
<CommandResult return_code=0, stdout=18 bytes, stderr=... bytes>
>>> print(OR(clom.false, clom.echo('Vagrant up failed')).shell())
Vagrant up failed
```

Re-use commands:

```
>>> echo = clom.echo
>>> echo.one.two
'echo one two'
>>> echo.three.four.shell.all()
['three four']
>>> echo.foo.bar.shell.all()
['foo bar']
```

Background tasks:

```
>>> clom.VBoxHeadless.with_opts(startvm="Windows Base").background()
"nohup VBoxHeadless --startvm 'Windows Base' &> /dev/null &"
>>> clom.VBoxHeadless.with_opts(startvm="Windows Base").background().shell()
<CommandResult return_code=0, stdout=0 bytes, stderr=0 bytes>

>>> vbox.list.runningvms.shell.all()
['"Windows Base" {949ec0af-92d0-4140-8a6c-36301ca6f695}']
```

Works great with fabric:

```
>>> from fabric.api import run, local
>>> local(clom.ls)
[localhost] local: ls
clom            clom.egg-info   docs            nohup.out       tests
''
```

Can even create fab commands:

```
>>> clom.fab.test('doctest', 'unit').deploy('dev')
'fab test:doctest,unit deploy:dev'
>>> clom.fab.with_opts('-a', hosts='dev.host').deploy.with_args('dev','test')
'fab -a --hosts dev.host deploy:dev,test'
```

CHAPTER 4

API Documentation

4.1 Clom API

4.1.1 clom

The main interface to clom is the clom object:

```
>>> from clom import clom
>>> clom.cat
'cat'
```

Each attribute of the clom object is a `clom.command.Command`.

`clom.`**`clom`**
    Manager for generating commands.

`clom.`**`NOTSET`**
    Represents an argument that is not set as opposed to `None` which is a valid value

`clom.`**`STDIN`**
    Standard In file descriptor

`clom.`**`STDOUT`**
    Standard Out file descriptor

`clom.`**`STDERR`**
    Standard Error file descriptor

class `clom.`**`AND`**(**commands*)
    Combine commands together that must execute together.

Parameters **commands** – List of Commands or Operations to combine

```
>>> from clom import clom
>>> AND(clom.echo('foo'), clom.echo('bar'))
'( echo foo && echo bar )'
```

class `clom.`**`OR`**(**commands*)
    Combine commands together that must not execute together.

Parameters **commands** – List of Commands or Operations to combine

```
>>> from clom import clom
>>> OR(clom.echo('foo'), clom.echo('bar'))
'( echo foo || echo bar )'
```

4.1.2 Commands

class `clom.command.`**`Command`**(*clom*, *name*, *parent=None*)

A command line command.

Don't use directly, instead use a clom object.

```
>>> from clom import clom
>>> type(clom.cat)
<class 'clom.command.Command'>
```

`as_string`(**args*, ***kwargs*)

Shortcut for *command.with_opts(**kwargs).with_args(*args)*

Returns str - Command suitable to pass to the command line

`shell`

Returns a *Shell* that will allow you to execute commands on the shell.

class `clom.command.`**`Operation`**

Base class for all command line operations, functions, commands, etc.

`as_string`()

Returns str - Command suitable to pass to the command line

`shell`

Returns a *Shell* that will allow you to execute commands on the shell.

4.1.3 Shell

class `clom.shell.`**`Shell`**(*cmd*)

Easily run 'Command's on the system's shell.

`all`(**args*, ***kwargs*)

Executes the command and returns a list of the lines of the result.

Alias for *shell(...).all()*

```
>>> str(clom.echo.shell.all('foo\nfoobar'))
"['foo', 'foobar']"
```

`execute`(**args*, ***kwargs*)

Execute the command on the shell without capturing output.

Use this if the result is very large or you do not care about the results.

Raises CommandError

Returns CommandResult

`first`(**args*, ***kwargs*)

Executes the command and returns the first line. Commands with no output return empty-string.

Alias for *shell(...).first()*

```
>>> clom.echo.shell.first('foo\nfoobar')
'foo'

>>> clom.true.shell.first()
''
```

iter (**args*, ***kwargs*)
Executes the command and returns an iterator of the results.

Alias for *shell(...).iter()*

last (**args*, ***kwargs*)
Executes the command and returns the last line.

Alias for *shell(...).last()*

```
>>> str(clom.echo.shell.last('foo\nfoobar'))
'foobar'
```

class clom.shell.**CommandError** (*return_code*, *stdout*, *stderr*, *message*)
An error returned from a shell command.

class clom.shell.**CommandResult** (*return_code*, *stdout=''*, *stderr=''*)
The result of a command execution.

all (*strip=True*)
Get all lines of the results as a list.

code
Alias to *return_code*

first (*strip=True*)
Get the first line of the results.

You can also get the return code:

```
>>> r = CommandResult(2)
>>> r.first().return_code
2
```

iter (*strip=True*)
Iterate over the command results split by lines with whitespace optionally stripped.

Parameters **strip** – bool - Strip whitespace for each line

last (*strip=True*)
Get the last line of the results.

You can also get the return code:

```
>>> r = CommandResult(2)
>>> r.last().return_code
2
```

return_code
Returns the status code returned from the command.

stderr
Returns the command's stderr as a string.

stdout
Returns the command's stdout as a string.

4.1.4 Arguments

class clom.arg.**RawArg** (*data*)
A command line argument that is not escaped at all.

class `clom.arg.`**`LiteralArg`**`(`*data*`)`

A command line argument that is fully escaped.

Use this if you want the value to be maintained and not interpolated.

Chars like * and $ are escaped and the value is wrapped in quotes.

See also:

http://www.gnu.org/software/bash/manual/bashref.html#Single-Quotes

Python Module Index

Index