clom Documentation

Release 0.7

Mike Thornton

April 02, 2012

CONTENTS

CHAPTER

ONE

ABOUT

Clom is a Python Command Line Object Mapper. It's intended to make generating 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

TWO

INSTALLATION

Install with `pip` or `easy_install`.

```
pip install clom
```

CHAPTER

THREE

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.ls.shell.all()
['clom', 'clom.egg-info', 'docs', 'tests']

>>> clom.git.status.shell('.').first()
'# On branch master'
```

Iterate over results:

```
>>> for path in clom.ls.shell():
...     print path
...
clom
clom.egg-info
docs
tests
```

Handle errors:

```
>>> clom.vagrant.up.shell()
Traceback (most recent call last):
  File "<console>", line 1, in <module>
  File "/Users/mthornton/Dropbox/Projects/python-clom/src/clom/shell.py", line 164, in __call__
    raise CommandError(status, stdout, stderr, 'Error while executing "%s" (%s):\n%s' % (cmd, status,
```

```
CommandError: Error while executing "vagrant up" (3):
No Vagrant environment detected. Run `vagrant init` to set one up.
```

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()
<clom.shell.CommandResult object at 0x10c4a85d0>
>>> print OR(clom.vagrant.up, clom.echo('Vagrant up failed')).shell()
No Vagrant environment detected. Run `vagrant init` to set one up.
Vagrant up failed
```

Re-use commands:

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

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

FOUR

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'>
```

append_to_file(*filename*, *fd=1*)
Append this command's output to a file.

Parameters

- **filename** – Filename to append to
- **fd** – File descriptor to redirect to file

Returns Operation

See Also:

http://tldp.org/LDP/abs/html/io-redirection.html

```
>>> clom.ls.append_to_file('list.txt')
'ls >> list.txt'

>>> clom.ls.append_to_file('list.txt', arg.STDERR)
'ls 2>> list.txt'
```

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

Returns str - Command suitable to pass to the command line

background()
Run the command in the background and don't block for output.

See Also:

http://www.gnu.org/software/bash/manual/bashref.html#Lists

```
>>> clom.ls.background()
'nohup ls &> /dev/null &'
```

from_file(*filename*)
Read a file's contents with this command's stdin.

Parameters **filename** – The filename to read input from

Returns Command

See Also:

http://tldp.org/LDP/abs/html/io-redirection.html

```
>>> clom.cat.from_file('list.txt')
'cat < list.txt'
```

hide_output(*fd=1*)
Redirect a command's file descriptors to /dev/null.

Parameters **fd** – File descriptor to redirect to /dev/null

Returns Operation

See Also:

http://tldp.org/LDP/abs/html/io-redirection.html

```
>>> clom.cat.hide_output()
'cat > /dev/null'

>>> clom.cat.hide_output(STDERR)
'cat 2> /dev/null'
```

output_to_file(*filename*, *fd=1*)

Replace a file's contents with this command's output.

Parameters

- **filename** – Filename to append to
- **fd** – File descriptor to redirect to file

Returns Operation

See Also:

http://tldp.org/LDP/abs/html/io-redirection.html

```
>>> clom.ls.output_to_file('list.txt')
'ls > list.txt'

>>> clom.ls.output_to_file('list.txt', STDERR)
'ls 2> list.txt'
```

pipe_to(*to_cmd*)

Pipe this command to another.

Parameters **to_cmd** – Operation or Command to pipe to

Returns Operation

See Also:

http://tldp.org/LDP/abs/html/io-redirection.html

See Also:

http://www.gnu.org/software/bash/manual/bashref.html#Pipelines

```
>>> clom.ls.pipe_to(clom.grep)
'ls | grep'
```

redirect(*from_fd*, *to_fd*)

Redirect a command's file descriptors.

Parameters

- **from_fd** – File descriptor to redirect from
- **to_fd** – File descriptor to redirect to

Returns Operation

See Also:

http://tldp.org/LDP/abs/html/io-redirection.html

```
>>> clom.cat.redirect(STDERR, STDOUT)
'cat 2>&1'
```

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

with_args(**args*)
Arguments to call the command with.

Parameters **args** – A list of arguments to pass to the command. Arguments are by automatically escaped as a *clom.arg.LiteralArg* unless you pass in a *clom.arg.RawArg*.

Returns Command

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

with_env(***kwargs*)
Run the operation with environmental variables.

Parameters **kwargs** – dict - Environmental variables to run command with

with_opts(**args*, ***kwargs*)
Options to call the command with.

Parameters

- **kwargs** – A dictionary of options to pass to the command. Keys are generated as *–name value* or *-n value* depending on the length.
- **args** – A list of options to pass to the command. Args are only escaped, no other special processing is done.

Returns Command

```
>>> clom.curl.with_opts('--basic', f=True, header='X-Test: 1')
'curl --basic --header 'X-Test: 1' -f'
```

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

append_to_file(*filename*, *fd=1*)
Append this command's output to a file.

Parameters

- **filename** – Filename to append to
- **fd** – File descriptor to redirect to file

Returns Operation

See Also:

http://tldp.org/LDP/abs/html/io-redirection.html

```
>>> clom.ls.append_to_file('list.txt')
'ls >> list.txt'

>>> clom.ls.append_to_file('list.txt', arg.STDERR)
'ls 2>> list.txt'
```

as_string()

Returns str - Command suitable to pass to the command line

background()
Run the command in the background and don't block for output.

See Also:

http://www.gnu.org/software/bash/manual/bashref.html#Lists

```
>>> clom.ls.background()
'nohup ls &> /dev/null &'
```

hide_output(*fd=1*)
Redirect a command's file descriptors to /dev/null.

Parameters **fd** – File descriptor to redirect to /dev/null

Returns Operation

See Also:

http://tldp.org/LDP/abs/html/io-redirection.html

```
>>> clom.cat.hide_output()
'cat > /dev/null'

>>> clom.cat.hide_output(STDERR)
'cat 2> /dev/null'
```

output_to_file(*filename*, *fd=1*)
Replace a file's contents with this command's output.

Parameters

- **filename** – Filename to append to
- **fd** – File descriptor to redirect to file

Returns Operation

See Also:

http://tldp.org/LDP/abs/html/io-redirection.html

```
>>> clom.ls.output_to_file('list.txt')
'ls > list.txt'

>>> clom.ls.output_to_file('list.txt', STDERR)
'ls 2> list.txt'
```

pipe_to(*to_cmd*)
Pipe this command to another.

Parameters **to_cmd** – Operation or Command to pipe to

Returns Operation

See Also:

http://tldp.org/LDP/abs/html/io-redirection.html

See Also:

http://www.gnu.org/software/bash/manual/bashref.html#Pipelines

```
>>> clom.ls.pipe_to(clom.grep)
'ls | grep'
```

redirect (*from_fd*, *to_fd*)
Redirect a command's file descriptors.

Parameters

- **from_fd** – File descriptor to redirect from
- **to_fd** – File descriptor to redirect to

Returns Operation

See Also:

http://tldp.org/LDP/abs/html/io-redirection.html

```
>>> clom.cat.redirect(STDERR, STDOUT)
'cat 2>&1'
```

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

with_env (***kwargs*)
Run the operation with environmental variables.

Parameters **kwargs** – dict - Environmental variables to run command with

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.

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

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

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

C