

---

# Seashore

*Release [seashore, version 20.6.0]*

**Jun 07, 2020**



---

## Contents

---

<b>1 Quick start</b>	<b>1</b>
<b>2 API</b>	<b>3</b>
2.1 Executor . . . . .	3
2.2 Shell . . . . .	5
<b>3 Release Process</b>	<b>9</b>
<b>Python Module Index</b>	<b>11</b>
<b>Index</b>	<b>13</b>



# CHAPTER 1

---

## Quick start

---

The Seashore library enables Pythonic command-based automation.

Creating an executor is easy:

```
from seashore import Executor, Shell, NO_VALUE
xctr = seashore.Executor(seashore.Shell())
```

Running commands looks like calling Python functions. In batch mode, commands will return their standard output and error.

```
base, dummy = xctr.git.rev_parse(show_toplevel=seashore.NO_VALUE,
                                ).batch(cwd=git_dir)
```

If an error occurs, an exception will be raised. If we just want to exit if any error is raised, but not leave a traceback,

```
def main():
    with seashore.autocode_exit():
        call_functions()
        run_executors()
```

The context will auto translate process errors to system exit.

There are also nice helpers, like `in_docker_machine`, which will return an executor where the docker commands are all pointed at a given docker machine.

```
dock_xctr = xctr.in_docker_machine('default')
dock_xctr.docker.run('ubuntu:latest', net='none',
                    rm=seashore.NO_VALUE,
                    interactive=seashore.NO_VALUE,
                    terminal=seashore.NO_VALUE,
                    volume='/myvolume',
                    env=dict(AWESOME='TRUE')).interactive()
```



## 2.1 Executor

Construct command-line lists.

`NO_VALUE` – indicate an option with no value (a boolean option)

**class** `seashore.executor.Command` (*name*)

A command is something that can be bound to an executor. Commands get automatically bound if defined as members of an executor.

**Parameters** `name` – the name of a ‘Modern UNIX’ command (i.e., something with subcommands).

**bind** (*executor*, *\_dummy=None*)

Bind a command to an executor.

**Parameters** `executor` – the executor to bind to

**Returns** something that has methods `batch`, `interactive` and `popen` methods.

**class** `seashore.executor.Eq` (*content*)

Wrap a string to indicate = option

Wrap a string to indicate that the option *has* to be given as ‘–name=value’ rather than the usually equivalent and more automation-friendly ‘–name value’

`git show --format, I’m looking at you.`

**class** `seashore.executor.Executor` (*shell*, *pypi=None*, *commands=NOTHING*)

Executes commands.

Init parameters:

**Parameters**

- **shell** – something that actually runs subprocesses. Should match the interface of `Shell`.
- **pypi** – optional. An extra index URL.
- **commands** – optional. An iterable of strings which are commands to support.

The default commands that are supported are `git`, `pip`, `conda`, `docker`, `docker_machine`.

**add\_command** (*name*)

Add a new command.

**Parameters** **name** – name of command

**chdir** (*path*)

Return a new executor where the working directory is different.

**Parameters** **path** – new path

**Returns** new executor with a different working directory

**command** (*args*)

Prepare a command from a raw argument list.

**Parameters** **args** – argument list

**Returns** something that supports batch/interactive/popen

**conda\_install** (*pkg\_ids*, *channels=None*)

Use conda to install packages

**Parameters**

- **pkg\_ids** – an list of package names
- **channels** – (optional) a list of channels to install from

**Raises** `ProcessError` if the installation fails

**in\_docker\_machine** (*machine*)

Return an executor where all docker commands would point at a specific Docker machine.

**Parameters** **machine** – name of machine

**Returns** a new executor

**in\_virtualenv** (*envpath*)

Return an executor where all Python commands would point at a specific virtual environment.

**Parameters** **envpath** – path to virtual environment

**Returns** a new executor

**patch\_env** (*\*\*kwargs*)

Return a new executor where the environment is patched with the given attributes

**Parameters** **kwargs** – new environment variables

**Returns** new executor with a shell with a patched environment.

**pip\_install** (*pkg\_ids*, *index\_url=None*)

Use pip to install packages

**Parameters**

- **pkg\_ids** – an list of package names
- **index\_url** – (optional) an extra PyPI-compatible index

**Raises** `ProcessError` if the installation fails

**prepare** (*command*, *subcommand*, *\*args*, *\*\*kwargs*)

Prepare a command (inspired by SQL statement preparation).

**Parameters**

- **command** – name of command (e.g., `apt-get`)
- **subcommand** – name of sub-command (e.g., `install`)
- **args** – positional arguments
- **kwargs** – option arguments

**Returns** something that supports batch/interactive/popen

`seashore.executor.cmd(binary, subcommand, *args, **kwargs)`

Construct a command line for a “modern UNIX” command.

Modern UNIX command do a closely-related-set-of-things and do it well. Examples include `apt-get` or `git`.

#### Parameters

- **binary** – the name of the command
- **subcommand** – the subcommand used
- **args** – positional arguments (put last)
- **kwargs** – options

**Returns** list of arguments that is suitable to be passed to `subprocess.Popen` and friends.

When specifying options, the following assumptions are made:

- Option names begin with `--` and any `_` is assumed to be a `-`
- If the value is `NO_VALUE`, this is a “naked” option.
- If the value is a string or an int, these are presented as the value of the option.
- If the value is a list, the option will be repeated multiple times.
- If the value is a dict, the option will be repeated multiple times, and its values will be `<KEY>=<VALUE>`.

## 2.2 Shell

Running subprocesses with a shell-like interface.

**exception** `seashore.shell.ProcessError(*args)`

A process has exited with non-zero status.

**class** `seashore.shell.Shell`

Run subprocesses.

Init arguments:

#### Parameters

- **cwd** – current working directory (default is process’s current working directory)
- **env** – environment variables dict (default is a copy of the process’s environment)

**batch** (`command`, `cwd=None`)

Run a process, wait until it ends and return the output and error

#### Parameters

- **command** – list of arguments
- **cwd** – current working directory (default is to use the internal working directory)

**Returns** pair of standard output, standard error

**Raises** `ProcessError` with (return code, standard output, standard error)

**chdir** (*path*)

Change internal current working directory.

Changes internal directory in which subprocesses will be run. Does not change the process's own current working directory.

**Parameters** **path** – new working directory

**clone** ()

Clone the shell object.

**Returns** a new Shell object with a copy of the environment dictionary

**getenv** (*key*)

Get internal environment variable.

Return value of variable in internal environment in which subprocesses will be run.

**Parameters** **key** – name of variable

**Returns** value of variable

**Raises** `KeyError` if key is not in environment

**interactive** (*command*, *cwd=None*)

Run a process, while its standard output and error go directly to ours.

**Parameters**

- **command** – list of arguments
- **cwd** – current working directory (default is to use the internal working directory)

**Raises** `ProcessError` with (return code, standard output, standard error)

**popen** (*command*, *\*\*kwargs*)

Run a process, giving direct access to the `subprocess.Popen` arguments.

**Parameters**

- **command** – list of arguments
- **kwargs** – keyword arguments passed to `subprocess.Popen`

**Returns** a `Process`

**reap\_all** ()

Kill, as gently as possible, all processes.

Loop through all processes and try to kill them with a sequence of `SIGINT`, `SIGTERM` and `SIGKILL`.

**redirect** (*command*, *outfp*, *errfp*, *cwd=None*)

Run a process, while its standard error and output go to pre-existing files

**Parameters**

- **command** – list of arguments
- **outfp** – output file object
- **errfp** – error file object
- **cwd** – current working directory (default is to use the internal working directory)

**Raises** `ProcessError` with return code

**setenv** (*key*, *val*)

Set internal environment variable.

Changes internal environment in which subprocesses will be run. Does not change the process's own environment.

**Parameters**

- **key** – name of variable
- **value** – value of variable

`seashore.shell.autoexit_code` (*\*args*, *\*\*kws*)

Context manager that translates `ProcessError` to immediate process exit.



## CHAPTER 3

---

### Release Process

---

In a virtual environment:

```
$ pip install incremental twisted click twine
$ git checkout master
$ git pull --rebase
$ git checkout -b new-release
$ python -m incremental.update --patch
$ git commit -a -m 'update to new version'
$ git push
```

On GitHub, create Pull Request, review and merge.

Then, back in the virtual environment:

```
$ git checkout master
$ git pull --rebase
$ pip wheel .
$ python setup.py sdist
$ twine upload seashore*.whl dist/seashore*.tar.gz
$ git tag v<version number>
$ git push --tags
```

On GitHub, create a release. Names for next few releases:

- *Dimorphodon macronyx*
- *Squaloraja polyspondyla*
- *Coprolite*

We base releases on the discoveries of [Mary Anning](#) who is the heroine of the tongue twister “she sells seashells by the seashore”.

After releasing, make sure to avoid accidental releases:

```
$ git checkout master
$ git pull --rebase
$ git checkout -b make-dev
$ python -m incremental.update seashore --dev
$ git commit -a -m 'prevent accidental releases'
$ git push
```

On GitHub, review and merge.

**S**

`seashore.executor`, 3

`seashore.shell`, 5



**A**

`add_command()` (*seashore.executor.Executor method*), 4

`autoexit_code()` (*in module seashore.shell*), 7

**B**

`batch()` (*seashore.shell.Shell method*), 5

`bind()` (*seashore.executor.Command method*), 3

**C**

`chdir()` (*seashore.executor.Executor method*), 4

`chdir()` (*seashore.shell.Shell method*), 6

`clone()` (*seashore.shell.Shell method*), 6

`cmd()` (*in module seashore.executor*), 5

`Command` (*class in seashore.executor*), 3

`command()` (*seashore.executor.Executor method*), 4

`conda_install()` (*seashore.executor.Executor method*), 4

**E**

`Eq` (*class in seashore.executor*), 3

`Executor` (*class in seashore.executor*), 3

**G**

`getenv()` (*seashore.shell.Shell method*), 6

**I**

`in_docker_machine()` (*seashore.executor.Executor method*), 4

`in_virtualenv()` (*seashore.executor.Executor method*), 4

`interactive()` (*seashore.shell.Shell method*), 6

**P**

`patch_env()` (*seashore.executor.Executor method*), 4

`pip_install()` (*seashore.executor.Executor method*), 4

`popen()` (*seashore.shell.Shell method*), 6

`prepare()` (*seashore.executor.Executor method*), 4

`ProcessError`, 5

**R**

`reap_all()` (*seashore.shell.Shell method*), 6

`redirect()` (*seashore.shell.Shell method*), 6

**S**

`seashore.executor` (*module*), 3

`seashore.shell` (*module*), 5

`setenv()` (*seashore.shell.Shell method*), 6

`Shell` (*class in seashore.shell*), 5