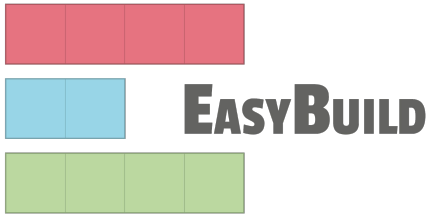

How EasyBuild Talks to Lmod

6 June 2023

Lmod monthly meeting

What is EasyBuild?

- **EasyBuild is a software build and installation framework**
- Strong focus on scientific software, performance, HPC systems
- Open source (GPLv2), implemented in Python (2.7, 3.5+)
- Brief history:
  - Created in-house at HPC-UGent in 2008
  - First released publicly in Apr'12 (version 0.5)
  - EasyBuild 1.0.0 released in Nov'12 (during SC12)
  - Worldwide community has grown around it since then!

https://easybuild.io

https://docs.easybuild.io

https://github.com/easybuilders

https://easybuild.io/join-slack

Twitter: @easy_build

Fediverse: @easybuild@mast.hpc.social

EasyBuild in a nutshell

- **Tool to provide a *consistent and well performing* scientific software stack**
- Uniform interface for installing scientific software on HPC systems
- Saves time by *automating* tedious, boring and repetitive tasks
- A platform for collaboration among HPC sites worldwide
- **Requires an environment modules tool like Lmod**
- **Automatically generates environment module files** (in Tcl or Lua syntax)

Group picture @ 5th EasyBuild User Meeting in Barcelona (Jan'20)

History on Lmod support in EasyBuild

- [Jul'13 - EasyBuild v1.6.0] Support for using Lmod as env. modules tool
- [Jul'14 - EasyBuild v1.14.0] Support for using a hierarchical module naming scheme
- [Apr'15 - EasyBuild v2.1.0] Support for generating env. module files in Lua syntax
- [Nov'16 - EasyBuild v3.0.0] **Lmod as default modules tool** + Lua as default module syntax
- [Dec'19 - EasyBuild v4.1.0] Deprecate support for Lmod 6 (yet still works today)
- [Apr'21 - EasyBuild v4.3.4] Manipulate `$MODULEPATH` directly rather than using `module use`

Lmod Unites

Todd Gamblin (Spack), Robert McLay (Lmod), Kenneth Hoste (EasyBuild)

Picture taken at HPC Knowledge Portal meeting 2017 (San Sebastián, Spain - June 2017)

Lmod's Dirty Little Secret: all it does is generate text!

- Lmod produces statements in a particular syntax to update current environment
- When the `module` or `ml` ommand is run, you are basically executing code generated by Lmod:

```
$ echo 'setenv("EXAMPLE", "test123")' > /tmp/Lmod-demo/test/1.2.3.lua
$ module use /tmp/Lmod-demo  # add location of test module to $MODULEPATH
$ lmod bash load test/1.2.3  # show statements that Lmod generates for bash shell
EXAMPLE=test123;
export EXAMPLE;
... # statements that update Lmod's internal bookkeeping ($_LMFILES_, ...) are omitted
$ eval $($LMOD_CMD bash load test/1.2.3)  # equivalent of "ml test/1.2.3"
$ echo $EXAMPLE
test123
```

Note: use `$LMOD_CMD` to find `lmod` command

Lmod speaks bash, Python, Perl, Lisp, CMake, R, Ruby, ...

see https://github.com/TACC/Lmod/tree/main/shells

```
$ lmod python load test/1.2.3
import os
os.environ["EXAMPLE"] = "test123";
```

```
$ lmod perl load test/1.2.3
$ENV{EXAMPLE}="test123";
```

```
$ lmod lisp load test/1.2.3
(setenv "EXAMPLE" test123)
```

```
$ lmod cmake load test/1.2.3
set(ENV{EXAMPLE} "test123")
```

```
$ lmod r load test/1.2.3
Sys.setenv("EXAMPLE"="test123");
```

```
$ lmod ruby load test/1.2.3
ENV["EXAMPLE"] = "test123"
```

Note: use `$LMOD_CMD` to find `lmod` command

Lmod (ab)uses both `stdout` and `stderr` output channels

- Statements to change environment are sent to `stdout` - not visible when running `module` or `ml`
- Human-oriented output is sent to `stderr` - escapes `eval` in `module` and `ml` functions
- Controlling exit code of `module` and `ml` is done by emitting `false` to create non-zero exit code

```
$ lmod bash no-such-command 2> /dev/null
false
$ echo $?
1
$ eval $(lmod bash no-such-command)
# stderr output omitted
$ echo $?
1
```

Note: use `$LMOD_CMD` to find `lmod` command

How EasyBuild Talks to Lmod

(see `Lmod` class in easybuild/tools/modules.py)

- EasyBuild uses "`lmod python`" to talk to Lmod
- Output produced to stdout/stderr is captured separately
- Exit code of `lmod` command is checked: non-zero exit code => failed command
- Python statements that change environment are executed with `exec` statement
- `--terse` option is used when running `available` or `list` subcommands (machine-readable)

```
$ lmod python --terse avail > /dev/null
/tmp/Lmod-demo:
test/
test/1.2.3
```

How EasyBuild Talks to Lmod
(loading a module)

(see `Lmod` class in easybuild/tools/modules.py)


```
$ python3
>>> import os, subprocess
>>> print(os.getenv('EXAMPLE'))
None
>>> lmod = os.getenv('LMOD_CMD')
>>> cmd = [lmod, 'python', 'load', 'test/1.2.3']
>>> proc = subprocess.Popen(cmd, stdout=subprocess.PIPE, stderr=subprocess.PIPE, universal_newlines=True)
>>> (stdout, stderr) = proc.communicate()
>>> exit_code = proc.returncode
>>> print(exit_code)
0
>>> print(stdout)
import os
os.environ["EXAMPLE"] = "test123";
...
>>> exec(stdout)
>>> print(os.getenv('EXAMPLE'))
test123
```

How EasyBuild Talks to Lmod
(failing to load a module)

(see `Lmod` class in easybuild/tools/modules.py)


```
$ python3
>>> import os, subprocess
>>> lmod = os.getenv('LMOD_CMD')
>>> cmd = [lmod, 'python', 'load', 'nosuchmodule/1.2.3']
>>> proc = subprocess.Popen(cmd, stdout=subprocess.PIPE, stderr=subprocess.PIPE, universal_newlines=True)
>>> (stdout, stderr) = proc.communicate()
>>> exit_code = proc.returncode
>>> print(exit_code)
1
>>> print(stderr)
Lmod has detected the following error: The following module(s) are unknown: "nosuchmodule/1.2.3"
...
>>> print(stdout)
_mlstatus = False
```

How EasyBuild Talks to Lmod *(module avail)*

(see `Lmod` class in easybuild/tools/modules.py)


```
$ python3
>>> import os, re, subprocess
>>> cmd = [lmod, 'python', '--terse', 'avail']
>>> proc = subprocess.Popen(cmd, stdout=subprocess.PIPE, stderr=subprocess.PIPE,
                            universal_newlines=True)
>>> (stdout, stderr) = proc.communicate()

>>> avail_regex = re.compile(r"^(?!-*\s)(?P<mod_name>[^\s\(]*[^:/])(?P<default>\(default\))?(\([^()]+\))?\s*$")

>>> modules = []
>>> for line in stderr.split('\n'):

...   matches = avail_regex.finditer(line)

...   for match in matches:

...   modules.append(match.group('mod_name'))

>>> modules
['test/1.2.3']
```

Specifics on EasyBuild and Lmod

- EasyBuild 4.x supports Lmod v6.x, v7.x, v8.x - but we plan to stop supporting Lmod 6.x ...
- EasyBuild supports organising module in a hierarchical way,
  by using the `HierarchalMNS` module naming scheme
  see https://tutorial.easybuild.io/2023-eb-eessi-uk-workshop/easybuild-module-naming-schemes
- EasyBuild instructs Lmod to ignore the spider cache by setting **`$LMOD_IGNORE_CACHE`**,
  because it needs to see the *current* set of available modules, even if they have only been just installed
- We always use **`--show-hidden`** when running `module avail`, so hidden modules are always visible
- We set **`$LMOD_REDIRECT`** to `'no'` so Lmod always sends output messages to `stderr`
  (requires because Lmod can be configured to send user-facing output to stdout)
- We set **`$LMOD_QUIET`** to `'1'` to avoid that Lmod is too chatty (and breaks our fancy regex's)
- We set **`$LMOD_EXTENDED_DEFAULT`** to `'no'` so only exact module versions are used