Lmod: Building a Community around an Environment Modules Tool

Robert McLay

The Texas Advanced Computing Center

Jan. 30, 2015

TACC

THE UNIVERSITY OF TEXAS AT AUSTIN

Outline

- What are Environment Modules and Lmod?
- Using Lua to implement Lmod features
- My experience building a community
- Lmod tech. solutions to built trust
- Conclusions

What are Environment Modules?

- A tool to set (and unset!) environment variables.
- Useful for adding elements to $PATH, $LD_LIBRARY_PATH
- Also remove elements from $PATH, etc as well.
- 1st paper on Env. Modules in 1991 by Furlani.
- There is a TCL/C based tool (a.k.a Tmod)

Why is this useful?

- High Performance Computers have hundreds of users.
- Physicists, Chemists, Biologist, Engineers need different software
- Modules provide a convenient way to support them all.
- A software developers delight:
  - Switch compilers easily.
  - For both versions and programs.

An Example

```
$  ddt
command not found: ddt
$  module load ddt
$  ddt
$  module rm ddt
$  ddt
command not found: ddt
```

What is Lmod?

- A Lua based Enviroment Module Tool that supports TCL modulefiles
- Tmod doesn't support a software hierarchy
- The C++ Boost library needs to match Compiler and Version
- Switch compilers should swap boost to match
- Lmod does this automatically

Why is this possible?

- How can a command effect the current environment?
- In Unix, the child process inherits the environment
- Not the other way around

The trick

- All *lmod* does is produce text.
- `module () { eval $(lmod bash "$@")}`

A Simple Modulefile in Lua

```
help([[A help message for ddt]])
setenv("TACC_DDT_DIR", "/opt/apps/ddt/3.4.1")
prepend_path("PATH",   "/opt/apps/ddt/3.4.1/bin")
```

Results from `module load ddt`

```
# in Bash:
export TACC_DDT_DIR="/opt/apps/ddt/3.4.1"
export PATH="/opt/apps/ddt/3.4.1/bin:..."

# in C-shell:
setenv TACC_DDT_DIR "/opt/apps/ddt/3.4.1"
setenv PATH         "/opt/apps/ddt/3.4.1/bin:..."
```

Results from `module unload ddt`

```
# in Bash:
unset TACC_DDT_DIR
export PATH="..."

# in C-shell:
unsetenv TACC_DDT_DIR
setenv   PATH          "..."
```

What is Lmod doing?

- Sites/Users write actions required for load.
- Each function does different things depending on mode.
- It is usually either: action, reverse, quiet

How to handle the different modes?

1. Single functions with if tests
2. redefine `setenv`, `prepend_path`, ...
3. Class based solution

Module functions

```
main()
  local mode = "load"
  mcp = MasterControl.build(mode)
  ...
  sandbox_run(fn)
end
function setenv(...)
   mcp:setenv(...)
end
function prepend_path(...)
   mcp:prepend_path(...)
end
```

Factory to build Lmod evaluation modes

```
function MasterControl.build(name)
  local tbl = { load = require('MC_Load'),
                unload = require('MC_Unload'), }
  return tbl[name]:create()
end
```

MC_Load.lua, MC_Unload.lua

```
# MC_Load.lua
local M         = inheritsFrom(MasterControl)
M.help          = MasterControl.quiet
M.prepend_path = MasterControl.prepend_path
M.setenv        = MasterControl.setenv
return M

# MC_Unload.lua
local M         = inheritsFrom(MasterControl)
M.help          = MasterControl.quiet
M.prepend_path = MasterControl.remove_path
M.setenv        = MasterControl.unsetenv
return M
```

Sandboxing

- Lmod uses a "sandbox" to evaluate modulefiles
- This is `pcall(untrusted_function)` with a limited environment
- No stack traceback for broken modulefiles.
- Protect Users from calling Lmod internal functions.
- Sites can add their own functions.

Hooks & SitePackage.lua

- Tmod has been around for 20+ years.
- Each site does things differently.
- Sites must be able to control behavior.
- Sites can create a SitePackage.lua file
- Register function with the hooks.

Example Hook Functions

- Use the load hook to keep track of module usage
- Register the site's name.
- Use the message hook to add to the output of avail and spider.

Passing Lmod state between calls

- Lmod uses a table to keep its state
- This is base64 encoded and stored in the environment
- This encoding avoids having to deal with quotes.

Lmod beginning (I)

- Lmod was first released in 2009.
- I prototyped it in Lua
- Figuring that Tmod community might be interested.
- The prototype was fast enough!

Lmod beginning (II)

- It was deployed at TACC.
- Opt-in/Opt-out strategy.
- TACC is one of the largest Open Science HPC systems in US.
- With over 10K user accounts (not all active)
- Lmod was designed to "scratch the itch" of our problems.
- Announced Lmod on the Environment Modules Mailing list.

Early User Interaction

- A user will find Lmod as the "answer" to their needs
- Sometimes I'm their new best friend.
- Sometimes that means stretching Lmod in new directions
- Sometimes that means saying no.
- I refused to add A.I. or change core features for your site only.

Building Trust

- We are all busy people.
- Few will change to my tool just because it is new.
- It has to be way "better" somehow.
- They want to know that you'll be around to support it.

Lmod's way to Build Trust

- A mailing list where questions get answered.
- Presentations at important conferences: SC, XSEDE, ISC
- Good up-to-date documentation (I'm working on it!!)
- Try not to break compatibility with original.

Specific issues w.r.t. Lua

- Lua is a great language to work in.
- But the community of users is much smaller than Python.
- There has been some resistance to accepting Lmod
- "*I don't want to learn Lua!*" - a busy sys-admin
- "*When is this going to be ported to Python?*" - another sys-admin
- A fair amount of feature request and bug reports
- Not much contributed code.

Feature Requests from users

- User requests feature that I don't think I'll use
- But sometimes my site does!
- Sometime users ask the right questions
- They can sometimes solve tech problems.

Tech Solutions

- A Test Suite for Lmod
- Logging Capability
- Report configuration.

Test Suite

- Hermes framework test suite: tm program
- A Test tool manager. It runs shell scripts
- The script much report pass/fail/diff
- "Wrong" tests can easily be rerun.
- Available: https://github.com/rtmclay/Hermes

Test Suite (II)

- Lmod produces both stderr and stdout
- Each output is filter to make it generic.
- `/home/mclay` $\Rightarrow$ `HOME` etc.
- Both stderr and stdout must match exactly with gold copies.
- Suite take under 2mins to run all tests.

Logging Capability

- I have yet to find a GUI debugger I like for lua.
- I developed this simple logger with indentation and { }
- This logging is *always* available
- This way I can debug remote problems!

Logging Code

```
local dbg = require("Dbg"):dbg()
function M.Load(n)
    dbg.start{"Load(",n,")"}
    ...
    dbg.print{"var: ",var,"\n"}
    ...
    dbg.fini("Load")
end
```

Logging example

```
$ module -D load ddt 2> load.log

lmod(-D load ddt){
  Load(ddt){
    MT:_build_locationTbl(mpathA){
      Cache:cache(){
      } Cache:cache
      moduleT: false
    }
      ...
}
```

Lmod configuration report

- There are many version of Lmod in the wild
- Some a year or more old.
- A bug may have been fixed in a new version
- Lmod has many options to control its behavior
- The configuration report is always include with logging.

Lmod configuration report

```
Modules based on Lua: Version 6.0.25 2016-01-12 09:51
    by Robert McLay mclay@tacc.utexas.edu

Description                          Value
-----------                          -----
Allow TCL modulefiles                yes
Auto swapping                        yes
Case Independent Sorting             no
Colorize Lmod                        yes
...
```

Conclusions

- It has been an interesting ride!
- Lmod is available from github, sourceforge
- Available: brew, fedora, Debian, Ubuntu, ...
- Lmod is more reliable and much more capable than just in-house project
- It has been a great deal of fun
- But also a lot of work!
- And there is no way I'm going to keep them all happy.