LCI workshop
on high performance clustered computing

Linux Clusters Institute: Lmod: A Modern Environment Module System

Robert McLay, Manager Software Tools, TACC

Outline

- What are Environment Modules?
- Kinds of Module Tools: Tmod, Cmod, Lmod, …
- What is Lmod?
- Modulefile Architecture
- Advanced Topics
- Where to go for help

Conclusions: Lmod

- Latest version: https://github.com:TACC/Lmod.git
- Stable version: http://lmod.sf.net
- Documentation: http://lmod.readthedocs.org
- Mailing List: lmod-users@lists.sourceforge.net
- Join here: https://lists.sourceforge.net/lists/listinfo/lmod-users

What are Modules?

- Modules are the way sites provide optional software: MPI, Boost, ...
- Modules add to PATH and set other env. vars for each package
- Modules can be unloaded.
- Modulefiles are in one file not one for each shell. (e.g. Compiler init scripts)

What is Lmod?

- A modern replacement for a tried and true concept.
- The guiding principal: ``Make life easier w/o getting in the way.''
- Reads both TCL and Lua modulefiles

Fundamental Issues

- Software Packages are created and updated all the time.
- Some Users need new versions for new features and bug fixes.
- Other Users need older versions for stability and continuity.
- User programs using pre-built C++ \& Fortran libraries must link with the same compiler.
- Similarly, MPI Applications must build and link with same MPI/Compiler pairing when using pre-built MPI libraries.

Example of Lmod: Environment Modules (I)

$ module avail

```
------------------ /opt/apps/modulefiles/MPI/intel/12.0/mpich2/1.4 ---------
 petsc/3.1 (default)    petsc/3.1-debug    pmetis/4.0    tau/2.20.3
------------------- /opt/apps/modulefiles/Compiler/intel/12.0 --------------
boost/1.45.0              gotoblas2/1.13          openmpi/1.4.3
boost/1.46.0              mpich2/1.3.2            openmpi/1.5.1
boost/1.46.1 (default)    mpich2/1.4 (default)    openmpi/1.5.3 (default)
------------------------- /opt/apps/modulefiles/Core ----------------------
StdEnv                intel/11.1               papi/4.1.4
admin/admin-1.0       intel/12.0 (default)     scite/2.28
ddt/ddt               lmod/lmod                tex/2010
dmalloc/dmalloc       local/local (default)    unix/unix (default)
fdepend/1.2           mkl/mkl                  visit/visit
gcc/4.4               noweb/2.11b
gcc/4.5 (default)
```

Example of Lmod: Environment Modules (II)

```
$ module list
Currently Loaded Modules:
  1) StdEnv  2) gcc/4.5  3) mpich2/1.4  4) petsc/3.1
$ module unload gcc
Inactive Modules:
  1) mpich2  2) petsc
$ module load intel
Activating Modules:
  1) mpich2  2) petsc
$ module load gcc
Due to MODULEPATH changes the follow modules have been reloaded:
  1) mpich2  2) petsc
```

Why You Might Want To Use Lmod?

- Same module commands as in Tmod
- Active Development; Frequent Releases; Bug fixes.
- Vibrant Community
- Used all over the world
- Enjoy many capabilities w/o changing a single module file
- Many more advantages when you're ready

Lmod Features

- Reads for TCL and Lua modulefiles
- One name rule.
- Support a Software Hierarchy
- Fast module avail via optional spider cache
- Properties (gpu, mic)
- Semantic Versioning: 5.6 is older than 5.10
- family(``compiler''), family(``MPI'') support
- Optional Tracking: What modules are used?
- Many other features: ml, collections, hooks, nag, ...

Tmod vs. Lmod

- Tmod (TCL/C) is in maintenance mode, Lmod active
- Some work on Tmod (TCL) recently.
- Lmod has many more features
- Tmod: module load gcc/5.3 gcc/6.0 loads both modules
- Lmod has the ``One Name Rule''
- Lmod is close to Tmod, but not the same.

Safety Features of Lmod (I)

- Users can only load one version of a package
- module load xyz/2.1 -> load xyz version 2.1
- module load xyz/2.2 -> unload 2.1, loads 2.2
- This can not be overridden!

Safety Features of Lmod (II)

- Lmod added a new command: family(”name”)
- All of our compiler modules have family(“compiler”)
- All of our MPI modules have family(“MPI”)
- Users can only load one compiler or MPI stack at a time
- Power users can override this restriction at their own peril

Module Architecture Design

- Lua or TCL modulefiles?
- Optional software: shared or local?
- Flat or Hierarchical Module Layout?
- Naming conventions? (N/V, C/N/V, N/V/V?)
- A standard set of modules?
- Keep software for life of cluster or not?
- Problems with Bash

Shared Disk versus Local Install

- Local install: Fast, small
- Shared Disk: Big, slower
- TACC: local install (mostly)
- Shared Disk: Keep software for life of system?

Modulefile Layout Choices

- Flat Naming Scheme?
- Hierarchical Naming Scheme?

Flat Naming Scheme: PETSc

- PETSc is a parallel iterative solver package:
  - PETSc/4.1-mvapich2-2.1-gcc-6.3
  - PETSc/4.1-mvapich2-2.1-intel-17.0
  - PETSc/4.1-openmpi-1.8-gcc-6.3
  - PETSc/4.1-openmpi-1.8-intel-17.0

Problems w/ Flat naming scheme

- User have to load:
  - module load intel/17.0
  - module load mvapich2/2.1-intel-17.0
  - module load PETSc/4.1-mvapich2-2.1-intel-17.0
- Changing compilers means unloading all three modules
- Reloading new compiler, MPI, PETSc modules
- Not loading correct modules -> Mysterious Failures!
- Onus on package compatibility on users!
- Or extremely complicated modulefiles

Extremely complicated modulefiles

- Protect users via conflicts and/or prereqs
- The problem is that they are fragile
- What happens with a new compiler or MPI stack?

Hierarchical Naming Schemes

- Store modules under one tree: opt/apps/modulefiles
- One strategy is to use sub-directories:
  - Core: Regular packages: apps, compilers, git
  - Compiler: Packages that depend on compiler: boost, MPI
  - MPI: Packages that depend on MPI/Compiler: PETSc, FFTW3

MODULEPATH

- MODULEPATH is a colon separated list of directories: containing directories and module files.
- No modulefiles loaded ->users can only load core modules.
- Loading a compiler module adds to MODULEPATH
  - Users can load compiler dependent modules.
  - This includes MPI implementations modules.
- Loading an MPI module adds to MODULEPATH
  - Users can load MPI libraries that match the MPI/compiler pairing
- See https://lmod.readthedocs.io/en/latest/080_hierarchy.html for details

Modulefile contents

- Be consistent! Find a convention and stick with it
- We define consistent variables in each module:
- <SITE_NAME>_<PKG_NAME>_{LIB,INC,BIN}
  - TACC_HDF5_BIN
  - TACC_HDF5_INC

Providing a standard set of modules

- Define a compiler and mpi stack
- /opt/apps/modulefiles/Core/StdEnv.lua:
  - load("gcc","mvapich2")

A Standard Set of Modules (II)

- In /etc/profile.d/z99_StdEnv.sh:

```
if [ -z "$__Init_Default_Modules" ]; then
   export __Init_Default_Modules=1;
   export LMOD_SYSTEM_DEFAULT_MODULES="StdEnv”
   module --initial_load --no_redirect restore
 else
   module refresh
 fi
```

- Why is the module refresh there?

User Collections with Save/Restore

- Users can setup their own initially loaded modules:
  - Users simply load, unload and/or swap until happy.
  - module save saves state into “default”
  - Shell startup scripts do “module restore” which load user’s default if it exists
- Users can create other collections:
  - $ module save name to save it
  - $ module restore name to restore it
- Note that a collection does a module purge before restoring collection.

Module reset, restore

- module reset -> module purge; module load $LMOD_SYSTEM_DEFAULT_MODULES
- module restore -> module purge; load default collection or module reset.

Module Naming Conventions

- N/V: Name/Version (e.g. bowtie/2.3)
- C/N/V: Category/Name/Version (e.g. bio/bowtie/2.3)
- N/V/V: Name/Version/Version (e.g. bowtie/64/2.3)

Module Naming Conventions (II)

- Try to stick with N/V if possible
- It's less typing
- C/N/V might be helpful to novice users
- But your obvious categories may not be obvious to your users
- Avoid N/V/V unless your users are experts
- Or if you really need 64/32 bit libraries

Bash issues

- Bash Startup is typically ``broken'' for non-login interactive shells
- Redhat, Centos, MacOS typically DO NOT source /etc/bashrc on interactive shells
- MPI jobs start an interactive shell.

Bash Issues (II)

- Want module command to work in all shells
- Want stacksize unlimited for MPI jobs
- We patched bash to force it to source /etc/tacc/bashrc

Bash Repair Choices

- Switch users to Z shell?
- patch bash (see Lmod docs)
- Expect all users to source /etc/bashrc in ~/.bashrc
- Expect all users to start jobs with #!/bin/bash -l
- I don’t trust all users to do the right thing ™

Keeping Software for Life of Cluster or Not

- It is possible with a shared disk approach
- You might want to hide older modules

Hidden modules

- Sites have always hide modules by adding a leading dot
- For example gcc/.6.3
- Lmod 7 also allows for hidden module via MODULERC
- system MODULERC or ~/.modulerc

Using System MODULERC to hide modules

- In $MODULERCFILE or /app/lmod/etc/rc:

  ```
  #%Module
  hide-version foo/3.2
  ```

Tracking Module Usage

- Lmod makes it easy to track module usage.
- Lmod can be setup to send a tagged message to syslog
- Rsyslog can send tags to a separate file.
- See lmod/contrib/tracking_module_usage/* for details

Usage counts

```
$ analyzeLmodDB --sqlPattern '%fftw%' --start '2015-01-01’ --end
'2015-02-01'  counts
Module path                                          Distinct Users
-----------                                          --------------
/apps/intel13/mpich_3_2/mfiles/fftw3/3.3.2                      151
/apps/intel13/mpich_3_2/mfiles/fftw2/2.1.5                       62
/apps/intel13/impi_4_1/mfiles/fftw3/3.3.2                        45
/apps/intel13/impi_4_1/mfiles/fftw2/2.1.5                        19
```

Distinct Users

```
$ ./analyzeLmodDB --sqlPattern '%/settarg/%' username
Module path                    User Name
-----------                    ---------
/apps/mfiles/settarg/5.8       user1
/apps/mfiles/settarg/5.8       user2
/apps/mfiles/settarg/5.8       user3
/apps/mfiles/settarg/5.8.1     mclay
/apps/mfiles/settarg/5.9.1     user5
```

Why does Lmod work at all?

- The Environment is inherited from the parent process
- Changes in the child's environment DOES NOT affect the parent's
- So how could Lmod work at all?

The trick is:

- The lmod program generates text.
- The module command eval's that text.
- module () { eval $($LMOD_CMD bash “$@”) ;}

Why is this important?

- It's a useful trick to know
- Debugging Modulefiles:
- $LMOD_CMD bash load module 2> /dev/null > stdout.txt

Tracing Lmod

- A new feature of Lmod 7.4.4+
- module -T ...
- export LMOD_TRACING=yes
- Can trace loads and how restores work.

How to trace Lmod startup behavior

- How do you trace module commands in /etc/profile.d/*.sh?
- Could modify /etc/profile.d/z99_StdEnv.sh to turn on LMOD_TRACING for a particular user. UGH!
- Install SHELL STARTUP DEBUG package instead

SHELL STARTUP DEBUG Package

- Install from shellstartupdebug.sf.net
- Tracks startup behavior of /etc/profile.d/*.sh
- Allows for the setting of env. vars before /etc/profile.d/*.sh is sourced.
- Requires modifying /etc/bashrc ...

SHELL STARTUP DEBUG (II)

- export SHELL_STARTUP_DEBUG=1 (in ~/.init.sh)
- Example output:

```
/etc/profile{
    /etc/profile.d/z00_lmod.sh{
    } Time = 0.0770
    /etc/profile.d/z99_StdEnv.sh{
    } Time = 0.2067
    /etc/bash.bashrc{
    } Time = 0.2338
} Time = 0.3156
```

SHELL STARTUP DEBUG and LMOD_TRACING

- Put `export LMOD_TRACING=yes` in ~/.init.sh
- Track Lmod startup issues:

```
running: module --initial\_load restore
Using collection:      /home/user/.lmod.d/default
  Setting MODULEPATH to: /apps/mfiles/Darwin:/apps/mfiles/Core
  Loading: unix (fn: /apps/mfiles/Core/unix/unix.lua)
  Loading: gcc (fn: /apps/mfiles/Darwin/gcc/5.2.lua)
  Loading: StdEnv (fn: /apps/mfiles/Core/StdEnv.lua)
```

Advanced Topics

- ml
- Spider Cache
- Modulefiles can have properties
- New functions pushenv()
- SitePackage.lua and hooks

ml

- I have trouble typing ``module''
- Needed a shortcut program not an alias.
- ml was born
- ml -> module list
- ml name -> module load name
- ml –a b –c d -> module unload a c; module load b d
- ml av -> module avail
- ml spider -> module spider

Spider Cache

- Lmod has to know what modulefiles are in MODULEPATH
- It walks the directories in MODULEPATH every time!
- Or you can have system spider cache to speed things up
- https://lmod.readthedocs.io/en/latest/130_spider_cache.html for details.

Spider Cache Advantages

- The spider cache speeds up avail and spider greatly
- All system modulefiles have been read, properties determined
- Lua is quite fast and reading and interpreting a single file
- This is preferable to walking the directory tree and reading every module
- Why is every module file read: properties

Spider Cache Disadvantages

- There is only one: Keeping it up-to-date!
- If Lmod sees a valid cache file it assumes it is correct
- Otherwise what's the point.
- Currently loads bypass cache but avail and spider depend on it.
- Personal modules are not effected by system cache foo.

Module Properties

- TACC deployed Stampede with MIC accelerators
- Some modules will be ``MIC'' aware: mkl, fftw3, phdf5, ...
- Lmod will decorate these modules:

```
1) unix/unix      3) ddt/ddt       5) mpich2/1.5    7) phdf5/1.8.9 (m)
2) intel/13.0     4) mkl/mkl (*)   6) petsc/3.2     8) StdEnv
Where
   (m):  module is build natively for MIC
   (*):  module is build natively for MIC and offload to the MIC.


add_property("arch","mic")                -- > phdf5
add_property("arch","mic:offload")        -- > mkl
```

Module Properties (II): Sticky

- A module can be sticky.
- It requires ``--force'' to unload or purge.

```
add_property("lmod","sticky")
```

pushenv()

- Suppose you'd like to set CC in the environment.
- setenv() won't work
- pushenv() will!

```
                          #     setenv()    pushenv()
$ module load   gcc;      # -> CC=gcc       CC=gcc
$ module load   mpich;    # -> CC=mpicc     CC=mpicc
$ module unload mpich;    # -> CC is unset  CC=gcc
$ module unload gcc;      # -> CC is unset  CC is unset
```

SitePackage.lua

- Extra Functions for all modulefiles for a site needs
- Support for hooks like load_hook()
- See contrib/SitePackage for examples

Remote Debugging

- No software over ten lines is bug free.
- Lmod is no exception.
- Bug reports are as easy as:
  - module --config 2> config.log
  - module -D avail 2> avail.log

Conclusions: Lmod

- Latest version: https://github.com:TACC/Lmod.git
- Stable version: http://lmod.sf.net
- Documentation: http://lmod.readthedocs.org
- Shell Startup Debug: http://shellstartupdebug.sf.net
- Mailing List: lmod-users@lists.sourceforge.net
- Join here: https://lists.sourceforge.net/lists/listinfo/lmod-users