LCI workshop
on high performance clustered computing

Linux Clusters Institute: Lmod: A Modern Environment Module System

Kyle Hutson, System Administrator, Kansas State University

KANSAS STATE UNIVERSITY | Computer Science

Overview

- What are environment modules?
  - Around since early 1990's
  - Sets up environment
    - PATH
    - LD_LIBRARY_PATH
    - Other Environment variables
  - Can be loaded an unloaded
  - Why?
- Lmod
  - Lua-based
  - Developed at TACC

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 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/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 to use Lmod?

- Same module commands as original modulefiles
- 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, ..

Safety features

- Essentials
  - 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!
- Advanced
  - 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?
- A standard set of modules?
  - e.g. GPU-enabled nodes
  - License-locked nodes
- Keep software for life of cluster or not?

Example

```
help([==[

Description
===========
The Open MPI Project is an open source MPI-2 implementation.


More information
================
 - Homepage: http://www.open-mpi.org/
]==])

whatis([==[Description: The Open MPI Project is an open source MPI-2 implementation.]==])
whatis([==[Homepage: http://www.open-mpi.org/]==])

local root = "/opt/software/software/OpenMPI/2.1.1-GCC-7.2.0-2.29"

conflict("OpenMPI")

if not isloaded("GCC/7.2.0-2.29") or mode() == "unload" then
   load("GCC/7.2.0-2.29")
end

if not isloaded("hwloc/1.11.7-GCCcore-7.2.0") or mode() == "unload" then
   load("hwloc/1.11.7-GCCcore-7.2.0")
end

prepend_path("CPATH", pathJoin(root, "include"))
prepend_path("LD_LIBRARY_PATH", pathJoin(root, "lib"))
prepend_path("LIBRARY_PATH", pathJoin(root, "lib"))
prepend_path("MANPATH", pathJoin(root, "share/man"))
prepend_path("PATH", pathJoin(root, "bin"))
prepend_path("PKG_CONFIG_PATH", pathJoin(root, "lib/pkgconfig"))
setenv("EBROOTOPENMPI", root)
setenv("EBVERSIONOPENMPI", "2.1.1")
setenv("EBDEVELOPENMPI", pathJoin(root, "easybuild/OpenMPI-2.1.1-GCC-7.2.0-2.29-easybuild-devel"))

-- Built with EasyBuild version 3.5.1
```

Shared Disk vs Local Install

- Local install: Fast, small
- Shared Disk: Big, slower
- Shared Disk: Keep software for life of system?

Modulefile Layout Choices

- Flat or Hierarchical
- Flat:
  - 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 would 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

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 Scheme

- 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
- Define consistent variables in each module:
- TACC uses <SITE_NAME>_<PKG_NAME>_{LIB,INC,BIN}
  - TACC_HDF5_BIN
  - TACC_HDF5_INC
- K-State uses <PKG_NAME>/<Version>-<toolchain>[-localupdate_version]

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)
- Hints:
  - 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.
- Want module command to work in all shells
- Want stacksize unlimited for MPI jobs
- How to fix:
  - 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 ™

Hiding 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
  - 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     user4
/apps/mfiles/settarg/5.9.1     user5
```

Most Used Commands

- module avail
- module load
- module list
- module unload
- module purge
- 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.
- 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

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.

Related: Easybuild

- Can be used for original modulefiles, as well as Lmod
- Software at https://easybuilders.github.io/easybuild/
- Allows you to quickly compile different versions (e.g. different architectures, toolchains, MPI, etc.)
- Example (from their website):

```
$ module load EasyBuild
$ export EASYBUILD_PREFIX=/tmp/$USER  # example prefix
$ eb HPL-2.0-goalf-1.1.0-no-OFED.eb –robot
```

Then you can use the module

```
$ export MODULEPATH=$EASYBUILD_PREFIX/modules/all:$MODULEPATH
$ module load HPL
```

Questions?