Modules

Efficient and Easy User Environment Configuration and Management

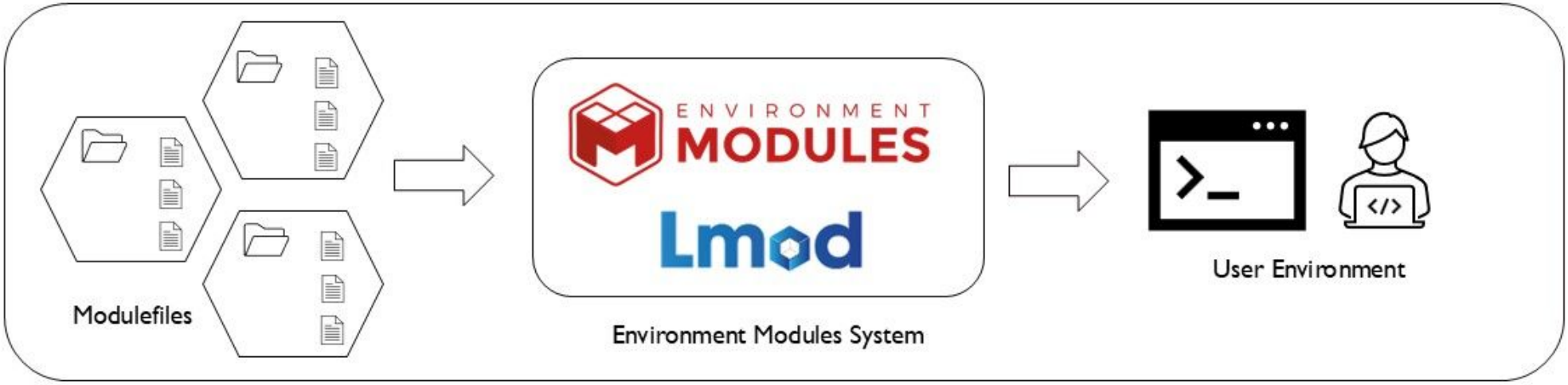
Santosh S Malagi
https://github.com/santoshsmalagi

Contents

Introduction To Environment Modules

What Are Environment Modules?

- Environment Modules allow dynamic modification of a user's environment by altering environment variables such as `$PATH` or performing other configuration tasks.
- A *modulefile* outlines the recipe needed to configure a user's shell for application startup, or for loading, unloading, and switching, between software packages.
- `'module'` is the command interface to an environment modules system. It reads and interprets the modulefile specified by the user and performs actions necessary to setup the required application startup environment.



Why Use Environment Modules?

- **#1 Compute environments need to satisfy a wide variety of users or groups of users**
  - The problem is often exacerbated by existence of multiple versions of software packages, libraries, runtimes etc. often compiled with different compilers, tweaks, or configuration settings.
- **#2 Managing multiple software versions is a complex undertaking**
  - A group of users want access to the latest and greatest version of software, whereas another group wants a specific back-level or bug-fix version
  - E.g. in a centralized CAD environment multiple versions of tool A, tool B, and tool C are installed
    - One group needs v11 of tool A, v9.3 of tool B, and v21.2 of tool C; whereas another group needs v14 of tool A, v9.4 of tool B and v22.0 of tool C
- **#3. End-users may not be knowledgeable in installing, managing, configuring the software tools**

- ***Environment modules provide a simple and systematic solution to user environment management!***
  - They are shell independent and support all major shells – `bash, csh, tclsh` etc.
  - Modulefiles are created per application per version basis, they can be dynamically loaded and unloaded
  - Modulefiles support advanced scripting capabilities
  - Application access or user policy can be enforced when a user loads a module
  - Meta-modules can load an entire suite of software applications
  - They are simple to use and avoid multiple copies of custom shell startup scripts

Tcl Modules (Tmod) vs Lua Modules (Lmod)

- Environment modules were first proposed in the 1990s as a solution to managing and configuring a myriad combination of compilers and libraries in HPC clusters. Since then, there have been multiple implementations of the environment modules system.

- **The "Environment Modules" or Tcl Environment Modules (Tmod)**
  - The more popular and widely used environment modules system implemented as a Tcl package - https://modules.sourceforge.net/

- **Lua Environment Modules (Lmod)**
  - An alternative and modern implementation of the environment modules system in Lua - https://tacc.utexas.edu/research/tacc-research/lmod/

- **Tmod vs Lmod**
  - Tmod is often considered a more mature environment module system, often available as a part of several Linux distributions. It supports modulefiles written in Tcl.
  - Lmod is a Lua based system which supports 'hierarchical modulefiles'. It can read modulefiles in both Lua and Tcl.

Installing and Configuring Lmod

- The installation of Lmod requires installing Lua as well, detailed instructions can be found here: https://lmod.readthedocs.io/en/latest/030_installing.html

- Once installed, ensure the following are appropriately set prior to using `'module'` command:

Environment Variable	Purpose	Example
`LMOD_CMD`	The path to the installed lmod command	`/usr/share/lmod/lmod/libexec/lmod`
`LMOD_DIR`	The directory that contains the installed lmod command	`/usr/share/lmod/lmod/libexec`
`LMOD_PKG`	This is the directory that contains the `libexec, init` directories etc.	`/usr/share/lmod/lmod`
`LMOD_ROOT`	parent directory of `LMOD_PKG`	`/usr/share/lmod`

https://lmod.readthedocs.io/en/latest/052_Environment_Variables.html

- Lmod internally sets the variables `LUA_PATH` and `LUA_CPATH` to ensure Lmod always uses the Lua version and libraries used to install it, instead of what is available in `PATH` to avoid any potential compatibility issues.

- **Locating modulefiles:**
  - The directory to search for modulefiles can be specified using `MODULEPATH`.
  - A directory in `MODULEPATH` can have an arbitrary number of sub-directories.

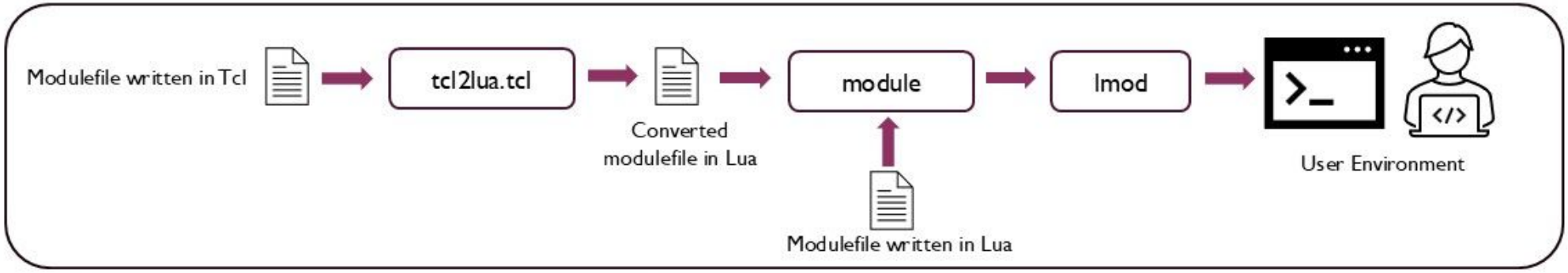
Using Lmod

How Lmod Works

- `'module'` command is a two-part process, the module shell function is as follows:

```
$ type module
module() { eval $($LMOD_CMD bash "$@") }
```

- `$LMOD_CMD` points to `lmod` command.
- `eval` reads the output from `stdout` and changes the current shell's environment. Any text written to `stderr` bypasses the `eval` and is written to the terminal.
- Further, the `module` command makes use of Lua feature known as 'ModuleTables' to remember its state, this is used when unloading a module.
- For modulefiles written in Tcl, Lmod uses a program called `tcl2lua.tcl` to read TCL modulefiles and converts them to Lua. The purpose of `tcl2lua.tcl` is to evaluate the regular TCL command but replace "module functions", such as `prepend-path` or `setenv`, and converts them to Lua functions.



https://lmod.readthedocs.io/en/latest/040_FAQ.html https://lmod.readthedocs.io/en/latest/095_tcl2lua.html

Modulefiles

- A modulefile is an ASCII file which contains the set of steps or recipe needed to configure a user's shell for a specific task or application startup.
- Usually, this recipe involves setting and unsetting various environment variables, checking for values in the user environment, running shell commands etc.
- Modulefiles are usually written in Tcl (or Lua for Lmod). They often support several advanced scripting capabilities, as well as user-defined extensions.
- Modulefiles can be installed in a central location for general use by multiple users, or in a user directory for personal use. The `MODULEPATH` variable specifies the directory search path for environment variables.

```
-- -----------------------------------
-- simple modulefile example in Lua
-- -----------------------------------

help("Load GCC v15.2")
whatis("Version: v15.2")
whatis("Host: x86_64-pc-linux-gnu")

local GCC_INSTALL_DIR = "/work/INSTALL/gcc_v15p2"

-- .. is the Lua string concatenation operator

-- setup PATH to GCC install dir
prepend_path("PATH", GCC_INSTALL_DIR .. "/bin")

-- list of dir path to search for header files
setenv("CPATH", GCC_INSTALL_DIR .. "/include/c++")

-- list of dirs to search for compile time static/shared libs
setenv("LIBRARY_PATH", GCC_INSTALL_DIR .. "/lib64")

-- list of dir paths to search for runtime shared libs
setenv("LD_LIBRARY_PATH", GCC_INSTALL_DIR .. "/lib64")
```

```
$ module help gcc/v15.2

---------- Module specific help for "gcc/v15.2" -----------

gcc/15.2     : Version: v15.2
gcc/15.2     : Host: x86_64-pc-linux-gnu
```

Lua Modulefile Functions

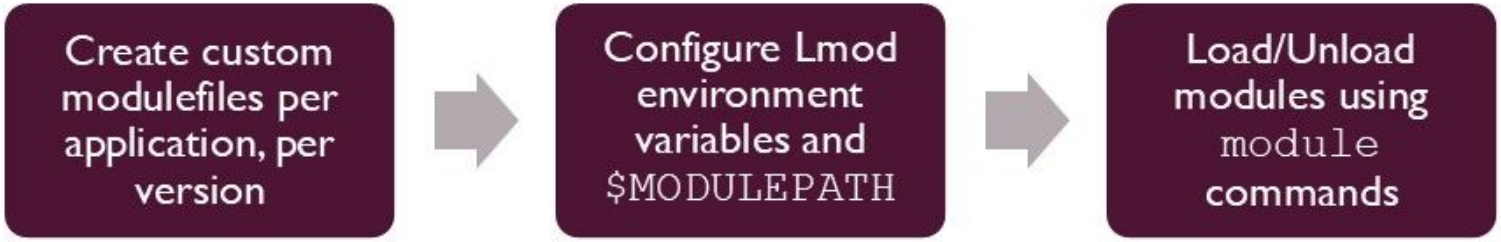
- Lmod provides several pre-defined functions to outline a set of actions to load a modulefile.
- These include functions for setting, unsetting or appending to environment variables, loading/unloading other module files, adding module help strings, sourcing shell scripts apart from additional functions to get values of shell environment variables, filesystem actions etc.
- The most common Lmod functions are as follows:

`prepend_path ("PATH", "/path/to/pkg/bin")`	prepend to a path-like variable the value
`prepend_path ("PATH", "/path/to/pkg/bin", "delim")`	prepend to a path-like variable the value. It is possible to add a third argument to be the delimiter
`append_path ("PATH", "/path/to/pkg/bin")`	append to a path-like variable the value
`append_path ("PATH", "/path/to/pkg/bin", "delim")`	append to a path-like variable the value. It is possible to add a third argument to be the delimiter
`remove_path ("PATH", "/path/to/pkg/bin")`	remove value from a path-like variable for both load and unload modes
`setenv ("NAME", "value")`	assigns to the environment variable "NAME" the value
`unsetenv ("NAME")`	unset the value associated with "NAME"
`whatis ("STRING")`	modulefiles can include a description section
`help ("STRING")`	help string to be printed for the module
`load ("pkgA", "pkgB", "pkgC")`	load all modules, report error if unable to load
`unload ("pkgA", "pkgB")`	unload all modules
`source_sh ("shellName","shell_script arg1 ...")`	source a shell-script as a part of a module
`prereq ("name1", "name2")`	modulefile will load only if all the listed modules are already loaded

https://lmod.readthedocs.io/en/latest/050_lua_modulefiles.html

Writing Custom Modulefiles

- Writing custom modulefiles involves using Lmod provided functions to set/unset environment variables or modify existing ones such as `$PATH`, `$LD_LIBRARY_PATH` etc. Users can also use Lua constructs, define custom functions etc.
- All modulefiles must end with a .lua extension or will be ignored, TCL modulefiles **MUST** begin with `#%Module`.
- Module names usually have one of the following naming schemes:
  - **`Shortname/Version`** – modulefiles named as `shortname.<version>`
    - Lmod simply picks the highest version if modulefile is loaded only using the shortname.
    - A leading '.' character in version means it is hidden, in the shortname means all versions are hidden.
  - **`Category/Name/Version`** – modules grouped by categories, and further organized as versions
    - this is a popular layout which can significantly improve readability and usability
    - meta-module – if the fullname is same as shortname and it has no version
    - Lmod supports as many directory levels as required
  - **`Name/Version/Version`** – this is a complex scheme which includes directories with version numbers



https://lmod.readthedocs.io/en/latest/055_module_names.html

How Lmod Picks which Modulefiles to Load?

- Lmod uses the directories listed in `MODULEPATH` to find the modulefiles to load.

Modulefiles organized by directories using Category/Name/Version scheme

```
.
├── doxygen
│   ├── default -> v1.9.6.lua
│   ├── v1.9.5
│   ├── v1.9.6.lua
│   ├── v1.9.7.lua
│   └── v1.9.8.lua
├── gcc
│   ├── v11.2.lua
│   ├── v14.2.lua
│   └── v9.3.lua
├── gdb
│   ├── v10.1.lua
│   ├── v12.1.lua
│   ├── v14.2.lua
│   └── v15.2.lua
└── StdEnv.lua
```

To mark a modulefile as default, simply create a symbolic link to the desired modulefile and name it as 'default'.

*There are many other rules on how Lmod picks which module file to load, see - https://lmod.readthedocs.io/en/latest/060_locating.html

A modulefile with no version associated with it is typically used as a meta-module – a module which loads other modules.

Lmod reports the following modules available for loading:

When a module is loaded using the shortname, Lmod by default picks the highest version.

```
$ module avail
------------------------------------------ /home/santoshsmalagi/modulefiles -------------------------
   StdEnv                doxygen/v1.9.7    gcc/v9.3     gcc/v14.2 (D)    gdb/v12.1    gdb/v15.2 (D)
   doxygen/v1.9.6 (D)    doxygen/v1.9.8    gcc/v11.2    gdb/v10.1        gdb/v14.2

---------------------------------------- /usr/share/lmod/lmod/modulefiles/Core ---------------------
   lmod    settarg

  Where:
   D:  Default Module
```

Basic Commands

Once modulefiles have been created, users run the '`module`' command with arguments to interact with Lmod system.

```
# display list of modules available to be loaded
$ module avail

# load list of modules currently loaded in the shell session
$ module list

# to load a package
$ module load pkg1 pk2 …

# to unload a package
$ module unload pkg1 pkg2

# to unload all modules
$ module purge

# refresh list of available modules
$ module refresh

# display help for module if defined in the modulefile using help() function
$ module help <modulename>

# print module description if defined using the whatis() function
$ module whatis <modulename>
```

https://lmod.readthedocs.io/en/latest/010_user.html

References

- Florida State University (FSU) Research Computing Centre (RCC)
  - Using Environment Modules - https://docs.rcc.fsu.edu/hpc/environment-modules/
- Lmod Environment Modules by Texas Advanced Computing Center (TACC)
  - https://tacc.utexas.edu/research/tacc-research/lmod/
  - https://lmod.readthedocs.io/
- Tmod Environment Modules by High Performance Computing at the French Alternative Energies and Atomic Energy Commission (CEA HPC)
  - https://modules.sourceforge.net/
  - https://modules.readthedocs.io/
- https://en.wikipedia.org/wiki/Environment_Modules_(software)