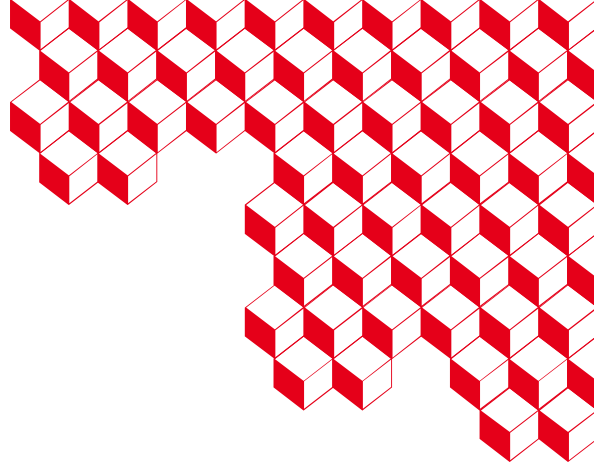

Environment Modules:
Why this old idea is still useful today and what's next

Marc Joos marc.joos@cea.fr, Xavier Delaruelle xavier.delaruelle@cea.fr

FOSDEM 2025, « HPC, Big Data & Data Science » devroom
February 2, 2025

What Modules is about?

- Modules, also called *Environment Modules*, is a tool that enables user to dynamically handle the environment of their shell session/script execution

- It evaluates script files named *modulefiles*

```
$ cat /path/to/modulefiles/foo
#%Module
prepend-path PATH /path/to/apps/foo/bin
```

- Modules is able to update current session with environment definition of modulefile

```
$ which foo
foo not found
$ module load foo
$ which foo
/path/to/apps/foo/bin/foo
```

- Afterward these environment changes can be undone

```
$ module unload foo
$ which foo
foo not found
```

How Modules works?

- A script, `modulecmd.tcl`, that evaluates modulefiles and output code corresponding to the environment changes they describe

```
$ /usr/share/Modules/libexec/modulecmd.tcl bash load foo
PATH=/path/to/apps/foo/bin:/usr/bin:/bin; export PATH;
_LMFILES_=/path/to/modulefiles/foo; export _LMFILES_;
LOADEDMODULES=foo; export LOADEDMODULES;
test 0;
```

- A shell or script function, named `module`, that evaluates the code produced in current session to update it.

```
$ type module
module is a function
module ()
{
    eval "$(tclsh '/path/to/libexec/modulecmd.tcl' bash "$@")";
}
```

What Modules is used for?

- Access software catalog of computing center
  - Where multiple versions of the same software should be provided to users
- Provide shell-agnostic environment management tool
  - Same command whether you use `bash`, `tcsh` or `fish` shell
- Handle software configuration or development environment
  - Example: Associate specific profile configuration to loaded MPI library

A long story project

When	What	Who
1991	Pioneering paper “Modules: Providing a Flexible User Environment”	John L. Furlan (Sun)
1991-1995	Modules version 1 (pure shell scripts) Modules version 2 (C evaluating Tcl modulefiles)	John L. Furlan (Sun)
1996-1999	Modules version 3.0 (C evaluating Tcl modulefiles)	Peter W. Osel (Siemens) Jens Hamisch (Strawberry)
1998-2012	Modules version 3.1 (Linux port, GPL license) Modules version 3.2	R.K. Owen (NERSC)
2002-2016	`modulecmd.tcl` (pure Tcl script evaluating Tcl modulefiles)	Mark Lakata (MIPS) Kent Mein (UMN)
2012-2017	Project hiatus: no new release after version 3.2.10 (2012)	-
2017-Now	Modules version 4 (based on `modulecmd.tcl`) Modules version 5	Xavier Delaruelle (CEA)

Modules development

- Project's forge: https://github.com/envmodules/modules
  - Language: Tcl
  - License: GPL-2.0-or-later
  - Size of `modulecmd.tcl`: 15k LOC

- 2 new feature releases drafted every year, in each of them
  - ~5 significant new features: https://modules.readthedocs.io/en/latest/MIGRATING.html
  - ~50 changes: https://modules.readthedocs.io/en/latest/NEWS.html

- Test Driven Development approach
  - 22k+ non-regression tests with 99.5% of code covered
  - Testsuite is 94k LOC

- Broadly listening to user and sysadmins needs expressed in Modules community or communities working same field (Spack, EasyBuild, Lmod, …)

Major new features since 2017

- Collections (`module save`/`restore`)
- Virtual modules
- I/O operations optimization
- Automated handling of module dependencies (prereqs & conflicts, unload/reload dependent)
- Advanced version specifiers (`module load foo@3.2:3.5`)
- Windows support (cmd and PowerShell)
- Hide/Forbid/Tag modules
- Sourcing shell script environment changes in modulefiles (`source-sh`)
- Module variants (`module load foo@2 +mpi bar=value`)
- Lmod Tcl modulefile support
- Initial environment and stashing (`module reset`/`stash`/`stashpop`)
- Linting modulefiles
- Module cache
- Querying available modules with extra specifiers (`module avail envvar:PATH require:foo`)
- Hooks (Tcl `trace` command and siteconfig variables)
- Logging activity
- …

Details and examples:

Module variants

- Pass arguments to evaluated modulefiles
  - Achieve different environment setup or module requirement with a single modulefile
- Using Spack's terminology and syntax (`module load foo@2 +mpi toolchain=foss24a`)
  - Support valued-variant and boolean-variant

```
$ module config editor cat
$ module edit bar/2.3
#%Module
variant toolchain foss22b foss24a

# select software depending on variant value
set suffix -[getvariant toolchain]

prepend-path PATH /path/to/apps/bar-2.3$suffix/bin
```

- Shortcuts could be set to ease specification

```
$ module config variant_shortcut toolchain=%
$ module load bar@2 %foss22b
Loading bar/2.3{%foss22b}
```

User activity logging

- The ability to log module command activity is now available out of the box
- Integrated logging feature relies on two configuration options:
  - `logger`: the command run to transmit messages to the log system
  - `logged_events`: list of module event to log

```
$ module config logged_events +requested_cmd:requested_eval
$ ml av
----------------- /path/to/modulefiles ------------------
app/3.2  app/4.1  toolchain/foss22b  toolchain/foss24a

Key:
modulepath
$ module load app
Loading app/4.1
  Loading requirement: toolchain/foss24a
$ journalctl -q -t modules
Apr 29 07:47:42 hostname modules[3777797]: user="username" command="avail" arguments=""
Apr 29 07:48:10 hostname modules[3777876]: user="username" command="load" arguments="app"
Apr 29 07:48:10 hostname modules[3777876]: user="username" mode="load" module="app/4.1" ...
```

- More details: https://modules.readthedocs.io/en/latest/MIGRATING.html#logging-activity

Lmod: another `module` implementation

- **Lmod** (https://github.com/TACC/Lmod) is another implementation of `module` written in Lua
  - Appeared in 2008 and developed by Robert McLay (TACC)
  - Able to evaluate both Tcl and Lua modulefiles

- Lmod is very popular across HPC sites
  - Many new features introduced at a time where Modules project was on halt
  - Some features: module hierarchy, module cache, family, one name rule, hooks, etc
  - As of today, this is the most deployed module tool in HPC world (Modules is now a challenger ☺)

- Both projects happily collaborate to provide users with similar interfaces

Software module hierarchy

- Software Module hierarchy is a way of organizing modulefiles popularized by Lmod
  - Core modules are available by default and when loaded they enable new entry in MODULEPATH
  - Loaded modules from these added modulepaths are reloaded when core modules are changed

```
$ module list

1) gcc/4.4.5  2) boost/1.45.0

$ module swap gcc ucc

Due to MODULEPATH changes the following modules have been reloaded: 1) boost
```

- More details: https://lmod.readthedocs.io/en/latest/010_user.html#module-hierarchy

- Support for software module hierarchy is planned for Modules 5.6 (Q2 2025)
  - Will be implemented an additional Automated module handling mechanism

Automated module handling mechanisms

- Handle module dependencies automatically (both requirements and conflicts)
- Satisfy users' request and provide them a consistent environment (no conflict, dependent reload if dependency changes)

```
$ module load foo/1
Loading foo/1
  Loading requirement: qux/1 bar/1
$ module config conflict_unload 1
$ module load foo/3
Loading foo/3
  Unloading conflict: foo/1 bar/1
  Loading requirement: bar/2
  Unloading useless requirement: qux/1
```

- More details
  - https://modules.readthedocs.io/en/latest/module.html#envvar-MODULES_AUTO_HANDLING
  - https://modules.readthedocs.io/en/latest/MIGRATING.html#conflict-unload

Why Modules is still useful?

- `module` is a well known interface to access installed software on HPC system

- Since the 90s they are now tools helping to install scientific software
  - EasyBuild: https://easybuild.io/
  - Spack*: https://spack.io/
  - Guix*: https://guix.gnu.org/

  * = does not require a module tool to provide access to installed software

- Yet, `module` helps providing the same interface whatever the underlying installation tool used
  - Important for sites working in the same federation where users move from site to site (ex: EuroHPC)
  - Also useful for sites using several tools simultaneously (we use EasyBuild+Spack at CEA)

- `module` is not only a tool to access installed software, it is a environment management tool
  - Useful to setup software configuration (ex: MPI profiles) or data endpoints

Future: View and load modules compatible with already loaded

- Modules currently loads default version whatever the current loaded environment

```
$ module avail
---------------- /path/to/modulefiles -----------------
app/3.2  app/4.1  toolchain/foss22b  toolchain/foss24a

Key:
loaded  modulepath
$ module load app
Loading app/4.1
  Unloading conflict: toolchain/foss22b
  Loading requirement: toolchain/foss24a
```

- Ease life by only view and load what is compatible with loaded environment

```
$ module avail
---------------- /path/to/modulefiles -----------------
app/3.2  toolchain/foss22b

Key:
loaded  modulepath
$ module load app
Loading app/3.2
```

Future: Search modulefiles to find a command

- Modern shells (`bash`, `zsh`, `fish`) have a hook to run function when typed command is not found

```
$ foo
bash: foo: command not found...
Install package 'foo' to provide command 'foo'? [N/y]
```

- Such hook (`command_not_found_handle` on `bash`/`zsh`) may be leveraged to provide direct access to software catalog without loading a module first:

```
$ app
Following modules provide 'app':
 1) app/3.2 (via toolchain/foss22b)
 2) app/4.1 (via toolchain/foss24a)
Select the module to load to run 'app' [1]:
```

- Original idea from **Mii** project (https://github.com/codeandkey/mii) that could be incorporated into Modules and benefit from its cache.

Future: sky is the limit

- If the community asks for it, support to evaluate Lua modulefiles can be added to Modules

- At some point, `modulecmd.tcl` may be rewritten into another language (Python? Rust?)
  - Tcl is well fitted for the job but nowadays developers do not know/like this language
  - Moving to a mainstream language will lower the contribution barrier

- Long term vision : be the glue between the different package manager tools
  - Provide users with a combined view of available software installed by multiple tools
  - Query the tools to know what is available through them

- Want specific development or expertise?
  - Ask XaDev (https://xadev.delaruelle.fr/), Xavier's self-company
  - Development and expertise on Modules and its ecosystem

XaDev

Joining HPSF and Linux Foundation

- Modules project is applying to join the High Performance Software Foundation (HPSF)
  - HPSF (https://hpsf.io) is a neutral hub for open source high performance software
  - HPSF is part of the nonprofit Linux Foundation

- An open governance and a technical charter is being defined for Modules project
  - Project has been moved from https://github.com/cea-hpc to https://github.com/envmodules
  - Project and its PI is transferred to a neutral body (LF) which will facilitate any future hand-over
  - Through this work, Modules will become a Linux Foundation project

- We are looking for additional maintainers to steer the project

HPSF
HIGH PERFORMANCE SOFTWARE FOUNDATION

THE LINUX FOUNDATION

Key takeaways

- Modules is an active project with a constant addition of new features
- There are still so many new things to add to ease life of users and sysadmins
- We happily collaborates with the community of projects working on the same field
  - Contributions made to EasyBuild, Spack, Lmod, SHPC
- Modules community is open and we are looking for additional people to run the project

Follow us

- Code/issues: https://github.com/envmodules/modules
- Mailing-list: modules-interest@lists.sourceforge.net
- Chat: **#modules:matrix.org** (new)
- Social media:
  - Twitter/X: @EnvModules
  - Mastodon: @EnvModules@mast.hpc.social
  - Bluesky: @EnvModules.bsky.social