DE LA RECHERCHE À L'INDUSTRIE

www.cea.fr

Modules

Help users managing their shell environment

Xavier Delaruelle <xavier.delaruelle@cea.fr>

UST4HPC

May 15th 2018, Villa Clythia, Fréjus

DE LA RECHERCHE À L'INDUSTRIE

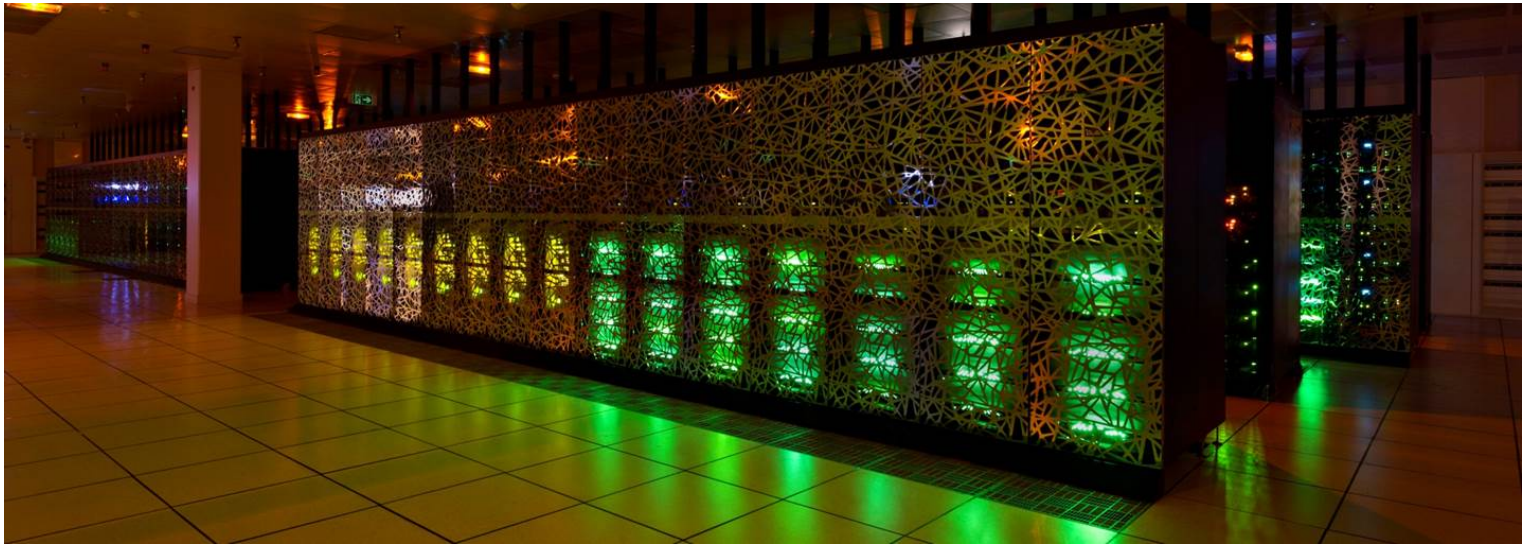
cea

whoami

- I am Xavier Delaruelle
- Joined CEA in 2007 as HPC system administrator
- Operations manager of CEA's TGCC computing center, since 2016
- Environment Modules project leader since July 2017

High Performance Computing at CEA

- Significant investment in terms of research and development.
- Operates 2 large computing facilities
  - TERA: computing center for defence-related programmes
  - TGCC: designed to serve European academic research community and French industry



The CURIE supercomputer installed at TGCC

`module` in a nutshell

- A shell command
- That changes the environment state of the current shell (environment variables, shell aliases)
- By interpreting Tcl script files (called modulefiles)
- Which load or unload environment configurations

Aim

Give users the ability to handle their environment

Give access to complex software catalogue

- On shared systems, multiple group of users may have conflicting software needs
- Group 1 wants software a in version 1 whereas Group 2 wants it in version 2
- Cannot used standard installation paths to satisfy everybody
- Which load or unload environment configurations

Quick example

- Activate a software

```
$ which gcc
gcc not found
$ module load gcc/6.1.1
$ which gcc
/apps/gcc/6.1.1/bin/gcc
```

- Check what a modulefile does

```
$ module display gcc/6.1.1
-------------------------------------------------------
/apps/modfiles/compilers/gcc/6.1.1:

prepend-path    PATH /apps/gcc/6.1.1/bin
-------------------------------------------------------
```

- De-activate a software

```
$ which gcc
/apps/gcc/6.1.1/bin/gcc
$ module unload gcc
$ which gcc
gcc not found
```

Quick example

- Activate a software

```
$ which gcc
gcc not found
$ module load gcc/6.1.1
$ which gcc
/apps/gcc/6.1.1/bin/gcc
```

- Check what a modulefile does

```
$ module display gcc/6.1.1
-------------------------------------------------------
/apps/modfiles/compilers/gcc/6.1.1:

prepend-path    PATH /apps/gcc/6.1.1/bin
-------------------------------------------------------
```

- De-activate a software

```
$ which gcc
/apps/gcc/6.1.1/bin/gcc
$ module unload gcc
$ which gcc
gcc not found
```

Quick example

- Activate a software

```
$ which gcc
gcc not found
$ module load gcc/6.1.1
$ which gcc
/apps/gcc/6.1.1/bin/gcc
```

- Check what a modulefile does

```
$ module display gcc/6.1.1
-------------------------------------------------------
/apps/modfiles/compilers/gcc/6.1.1:

prepend-path    PATH /apps/gcc/6.1.1/bin
-------------------------------------------------------
```

- De-activate a software

```
$ which gcc
/apps/gcc/6.1.1/bin/gcc
$ module unload gcc
$ which gcc
gcc not found
```

Manage software or global environment configuration

- Activate specific software configuration

```
$ module show mpiprofile/performance
-------------------------------------------------------
/apps/modfiles/parallel/mpiprofile/performance:

setenv          OMPI_MCA_mpi_leave_pinned 1
setenv          OMPI_MCA_btl_openib_use_eager_rdma 1
-------------------------------------------------------
```

- Advertise specific data end-points

```
$ module load datadir/myproj
$ echo $STOREDIR
/store/myproj/username
```

- Adapt user environment for specific work

```
$ umask
0077
$ module load collaborative_work
$ umask
0002
```

Write shell-agnostic environment guidelines

- Modules supports large number of shells (`sh`, `bash`, `ksh`, `zsh`, `csh`, `tcsh`, `fish`, `cmd`)
- Also supports scripting languages (`tcl`, `perl`, `python`, `ruby`, `cmake`, `R`)
- Do not write a different guide for each shell you support:

```
export PATH=/apps/gcc/6.1.1/bin:$PATH
export MANPATH=/apps/gcc/6.1.1/man:$MANPATH
```

- Just tell users to:

```
module load gcc/6.1.1
```

Environment Modules project history

- 1991: Concept and initial implementation of the `module` command laid down
- 1999: Modules ported to Linux, version 3.0 written in C
- 2017: Modules version 4.0 released, switched to a pure-Tcl implementation started in 2002

Today, `module` is a standard

- `module` command and modulefile syntax are de-facto standards to provide access to scientific software catalogue
- Well known tool in HPC world
- Available on most (all?) Linux distributions and other Unix-like systems

Other `module` implementations

- Pmodules: `https://gitlab.psi.ch/Pmodules/`
- pymod: `https://github.com/tjfulle/pymod/`
- Lmod: `https://github.com/TACC/Lmod/`

- Similar projects
  - direnv: `https://direnv.net/`

Objects handled by the `module` command

- Modulefile: Tcl script defining user-environment change orders
- Modulerc: Tcl script helping to locate modulefiles
- Modulepath: directory containing a tree of modulefiles and modulercs
- Collection: file recording an ordered modulepath list and an ordered modulefile list

Depending on called sub-command, `module` enables, disables or queries these objects on current shell environment

Modulefile

- Can leverage all Tcl scripting features `https://www.tcl.tk/man/tcl/TclCmd/contents.htm`
- Plus Modules-specific Tcl Commands to change or query user environment `https://modules.readthedocs.io/en/stable/modulefile.html`
- Evaluated depending on a mode triggered by module action: `load`, `unload`, `display`, `help`, `test`, `whatis`
- Change current shell state when evaluated on `load` or `unload` modes

Modulerc

- Evaluated when searching for modulefiles
- Can leverage all Tcl scripting features
- Plus a subset of the Modules-specific Tcl Commands to query user environment or locate modulefiles

Basic setup demo

- Setup container for demo

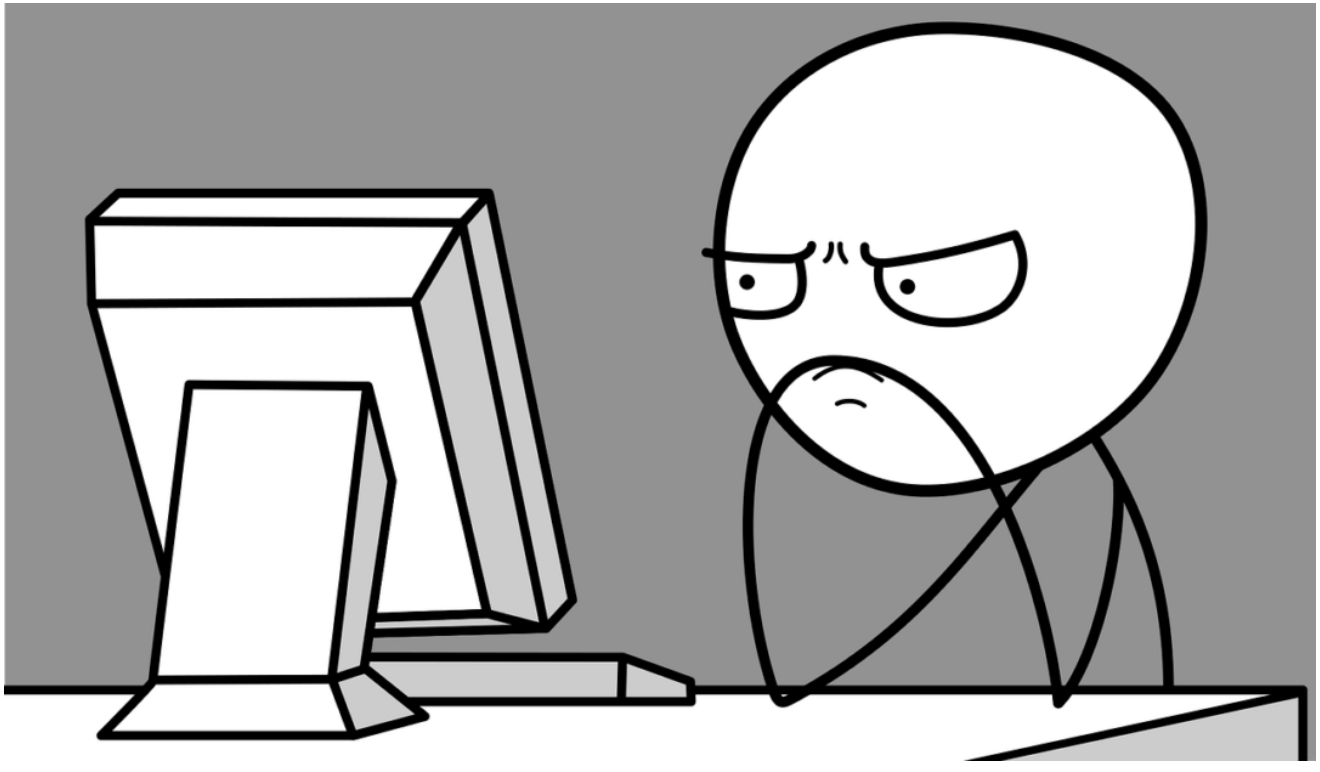
```
$ sudo service docker start
$ sudo docker pull fedora:28
$ sudo docker create -t -i --name modules-demo1 fedora:28 bash
$ sudo docker start -a -i modules-demo1
```

- Run in demo container

```
# dnf install -y environment-modules openmpi
# bash
# module
# module use
# module avail
# module show mpi
# module load mpi
# module list
# echo $PATH
# module unload mpi
# module list
# echo $PATH
# module whatis
```

How to setup and organize module environment?

- Many ways to proceed, depends of what you want to achieve



Writer’s block syndrome

Step by step Modules setup

- Installation
- `module` command setup
- `module` environment initialization
- Organizing modulefiles through modulepaths
- Writing modulefiles
- Locating modulefiles
- Express constraints between modulefiles
- Dependencies management
- Save and restore your environment with collections

Installation

- Setup container for the step by step Modules setup

```
$ sudo docker create -t -i --name modules-demo2 fedora:28 bash
$ sudo docker start -a -i modules-demo2
```

- Install tools and create users and groups needed for tests

```
# dnf install -y git less vim man grep sed procps-ng findutils \
    bash python

# useradd user1
# groupadd project1
# groupadd project2
# usermod -G project1 user1
```

Installation (2)

- Install Modules requirements

```
# dnf install -y dejagnu make tcl
```

- Installation from git repository

```
# git clone https://github.com/cea-hpc/modules.git ~/modules
# cd ~/modules
# ./configure --disable-compat-version
# make
# make install
# make testinstall
# find /usr/local/Modules
```

- Can also be installed from tarball (built from git repository or fetched online from SourceForge or GitHub project

`module` command setup

- `modulecmd.tcl` produces code to setup `module` command

```
# bash
# ./modulecmd.tcl bash autoinit
# eval $(./modulecmd.tcl bash autoinit)
# module
# exit
```

- Initialize `module` with shell-specific init script

```
# bash
# ls /usr/local/Modules/init
# less /usr/local/Modules/init/bash
# source /usr/local/Modules/init/bash
# module -V
# exit
```

- Automatically setup module command via system-wide profile.d scripts

```
# less /usr/local/Modules/init/profile.sh
# ln -s /usr/local/Modules/init/profile.sh /etc/profile.d/modules.sh
# bash
# module -V
```

`module` command setup (2)

- Setup and use `module` in scripting language

```
# cat test.py
import os
exec(open('/usr/local/Modules/init/python.py').read())
module('load', 'null')
if module('is-loaded', 'null'):
    print 'modulefile \'null\' is loaded'
# python test.py
```

`module` environment initialization

- Default configuration sourced from `<INSTALLPREFIX>/init/modulerc` during `module` command setup
- Can add there any Tcl or `module` command

```
# ./modulecmd.tcl bash autoinit >dflinit
# less /usr/local/Modules/init/modulerc
# echo 'module load null' >>/usr/local/Modules/init/modulerc
# ./modulecmd.tcl bash autoinit >newinit
# vimdiff dflinit newinit
# sed -i '/module load null/d' /usr/local/Modules/init/modulerc
```

Organizing modulefiles through modulepaths

- Modulepaths help to classify modulefiles
  - Per software kind: applications, libraries, compilers, tools, ...
  - Per scientific field: physics, chemistry, biology, ...
  - Per administrative domain: modulefiles provided by staff, by communities, user's own modulefiles
- Gives an easy-to-read view of availabilities

Organizing modulefiles through modulepaths (2)

- Creating and enabling modulepaths

```
# mkdir -p /apps/userenv/modules/{applications,environment}
# su - user1
$ module use
$ module use /apps/userenv/modules/*
$ module use
$ echo $MODULEPATH
```

- Enabling modulepaths at environment initialization

```
# cat <<EOF >>/usr/local/Modules/init/modulerc
module use /apps/userenv/modules/environment
module use /apps/userenv/modules/applications
EOF
# su - user1
$ module use
```

First modulefile

- Deploy dummy `bar` and `foo` applications

```
# mkdir -p /apps/bar-{1.0,2.1}/bin
# mkdir -p /apps/foo-{0.9,1.6}/bin

# for i in 1.0 2.1; do
cat <<EOF >/apps/bar-$i/bin/bar
#!/bin/bash
echo 'bar v$i'
EOF
done

# for i in 0.9 1.6; do
cat <<EOF >/apps/foo-$i/bin/foo
#!/bin/bash
echo 'foo v$i'
EOF
done

# chmod +x /apps/{foo,bar}-*/bin/*
```

First modulefile (2)

- Write our first modulefile for application `bar` version 1.0

```
# mkdir /apps/userenv/modules/applications/bar

# cat <<EOF >/apps/userenv/modules/applications/bar/1.0
#%Module
append-path PATH /apps/bar-1.0/bin
set-alias b /apps/bar-1.0/bin/bar
EOF
```

- Check this modulefile is correctly detected

```
# su - user1
$ module avail
$ module show bar
```

First modulefile (3)

- Load modulefile

```
$ module list
$ bar
$ b
$ module load bar
$ module list
$ bar
$ b
$ echo $PATH
```

- Modules-specific environment variables help to track what is currently loaded

```
$ echo $LOADEDMODULES
$ echo $_LMFILES_
```

- Unload modulefile

```
$ module unload bar
$ bar
$ module list
$ echo $LOADEDMODULES
```

Second modulefile

- Add a second modulefile application `bar` without code duplication

```
# cat <<EOF >/apps/userenv/modules/applications/bar/.common
append-path PATH /apps/bar-\$version/bin
set-alias b /apps/bar-\$version/bin/bar
EOF

# for i in 1.0 2.1; do
cat <<EOF >/apps/userenv/modules/applications/bar/$i
#%Module
set version ”$i”
source /apps/userenv/modules/applications/bar/.common
EOF
done
```

- Check both modulefiles are correctly detected

```
# su - user1
$ module avail bar
$ module show bar/1.0 bar/2.1
```

More modulefiles

- Add modulefiles for application `foo`

```
# mkdir /apps/userenv/modules/applications/foo

# cat <<EOF >/apps/userenv/modules/applications/foo/.common
prepend-path PATH /apps/foo-\$version/bin
EOF

# for i in 0.9 1.6; do
cat <<EOF >/apps/userenv/modules/applications/foo/$i
#%Module
set version ”$i”
source /apps/userenv/modules/applications/foo/.common
EOF
done
```

More modulefiles (2)

- Share code between application modulefiles

```
# for i in bar foo; do
echo 'source /apps/userenv/modules/.common' \
  >>/apps/userenv/modules/applications/$i/.common
done

# cat <<EOF >/apps/userenv/modules/.common
if {[module-info mode load] || [module-info mode unload]} {
    puts stderr "[module-info mode] module [module-info name]"
}
EOF
```

- Get a common message when loading/unloading a modulefile

```
# su - user1
$ module load bar
$ module unload bar
```

Use cases for sharing code between all modulefiles

- Log usages
- Inform about software depreciation
- Inform on required specific authorization to get access to software
- Harmonized `module help` messages
- Define normalized environment variables for all software (for instance: `FOO_BINDIR` or `BAR_LIBDIR`)

How modulefiles are found?

- No search when passed as full or relative path name

```
# su - user1
$ module load ../../apps/userenv/modules/applications/bar/1.0
$ module list
$ module unload /apps/userenv/modules/applications/bar/1.0
```

- Searched elsewhere when passed as a module name
  - Each modulepath is searched following their definition order
  - Looking for directory/files matching passed module name
  - `.modulerc` files found along the way are evaluated to get specific version definitions (alias, symbols, default version)
  - Search stops at first match

Guide modulefile search with `.modulerc`

- Define bunch of symbolic versions in `bar`'s `.modulerc` file

```
# cat <<EOF >/apps/userenv/modules/applications/bar/.modulerc
#%Module
module-version /1.0 default
module-version bar/2.1 2
EOF
```

- Search `bar`'s default version or symbols

```
# su - user1
$ module avail bar
$ module avail -d bar
$ module load bar
$ module load bar/2
$ module list
$ module purge
```

Guide modulefile search with `.modulerc` (2)

- Add some module aliases in `bar`'s `.modulerc` file

```
# cat <<EOF >>/apps/userenv/modules/applications/bar/.modulerc
module-alias bar/3.0 foo/1.6
module-alias foo/lastest foo/1.6
EOF
```

- Look at defined aliases

```
# su - user1
$ module avail bar
$ module avail -d bar
$ module avail -L bar
$ module load bar/3.0
$ module list

$ module avail foo
```

- `foo/latest` is not found as it is not defined in a location searched when looking for `foo` module name

Global `.modulercs`

- Global `.modulercs` help to define modulefile location orders at top scope
- They are searched and evaluated (if found) each time `module` is ran, prior to the execution of the called `module` sub-command
- Expected in fixed locations
  - `$MODULERCFILE`
  - `<INSTALLPREFIX>/etc/rc`
  - `$HOME/.modulerc`

Global `.modulercs` (2)

- Define a top-scope alias in `user1`-specific `.modulerc`

```
# cat <<EOF >~user1/.modulerc
#%Module
module-alias top bar/2.1
EOF
```

- Look at defined top alias

```
# su - user1
$ module avail
$ module avail top
$ module load top
$ module list
```

Restricting modulefile visibility

- Playing with file mode helps to restrict access to some modulefiles
- Some use cases: software restricted to a group of user, newer versions need first to be tested by local staff

```
# chmod go-rw /apps/userenv/modules/applications/bar/1.0
# chmod go-rwx /apps/userenv/modules/applications/foo

# su - user1
$ module avail
$ module avail -d bar
$ module load bar/1.0
```

- Restore previous mode for next tests

```
# chmod go+r /apps/userenv/modules/applications/bar/1.0
# chmod go+rx /apps/userenv/modules/applications/foo
```

Helping users to access data with modulefiles

- Modulefiles may also help to define pure-configuration to highlight the data end-points of the system
- Crafting some data end-points for test

```
# mkdir -m 770 -p /store/{user1,project1,project2}
# for i in user1 project1 project2; do
chgrp $i /store/$i
done
```

- Introducing a `datadir` module to help defining a `STOREDIR` variable relative to the module version loaded

```
# mkdir /apps/userenv/modules/environment/datadir
# cat <<EOF >/apps/userenv/modules/environment/datadir/.common
#%Module
setenv STOREDIR /store/[file tail [module-info name]]
EOF
```

Helping users to access data with modulefiles (2)

- Instanciate `datadir` module for each existing data space
- With module visibility restricted to people able to reach data space

```
# for i in user1 project1 project2; do
cat <<EOF >/apps/userenv/modules/environment/datadir/$i
#%Module
source /apps/userenv/modules/environment/datadir/.common
EOF
chmod 750 /apps/userenv/modules/environment/datadir/$i
chgrp $i /apps/userenv/modules/environment/datadir/$i
done
```

- Check new modulefiles

```
# su - user1
$ module avail
$ module load datadir/user1
$ module load datadir/project2
```

Dynamically define modulefiles with virtual modules

- Instanciate at `module` runtime virtual modules depending on current context
- Improve previous `datadir` modulefiles with `module-virtual` command

```
# for i in user1 project1 project2; do
rm /apps/userenv/modules/environment/datadir/$i
done

# cat <<EOF >/apps/userenv/modules/environment/datadir/.modulerc
#%Module
foreach grp [exec groups] {
    if {[string first proj \$grp] == 0 || \$grp eq $::env(USER)} {
        module-virtual /\$grp .common
    }
}
EOF
```

- Check new modulefiles

```
# su - user1
$ module avail
$ module load datadir/user1
$ module load datadir/project2
```

Ensure loaded module uniqueness

- `conflict` modulefile command may be used to ensure only one version of a module is loaded at a time

```
# for i in bar foo; do
sed -i ”1a conflict $i” /apps/userenv/modules/applications/$i/.common
done

# su - user1
$ module load bar/1.0
$ module load bar/2.1
```

- An error is returned when a conflict is detected
- Conflicts can also be declared between different modules

Ensure loaded module requirements are met

- `prereq` modulefile command helps declaring pre-requirements needed by modulefile to be loaded
- Declare a `prereq` on `foo` for `bar`

```
# sed -i ”1a prereq foo” /apps/userenv/modules/applications/bar/.common

# su - user1
$ module load bar
$ module load foo bar
```

- An error is returned if pre-requirements are not met

Dependency management

- How to handle software requirements and/or harmonized compilation toolchain between all loaded software?
- Solution 1: by settings a module hierarchy per compilation toolchain + handling other dependencies with RPATH edition
  - One modulefile per software compilation
  - Users have to learn about compilation toolchain to get access to the software catalogue
- Solution 2: flavoring modulefiles with pre-requirements adapting loaded modules according to
  - One modulefile to fit all compilation
  - Immediate access to all software catalogue
  - Can be hard to know what is available for a given compilation

Save and restore environment with collections

- Dump list of currently enabled modulepaths and modulefiles in a collection file

```
# su - user1
$ module load foo bar
$ module save
$ module savelist
```

- Restore collection on next connection

```
# su - user1
$ module restore
```

- Loaded order is preserved when restoring collection

```
# su - user1
$ module load datadir foo bar
$ module restore
```

Limit collection validity per target

- Collection may only be valid for a given machine, not for the whole infrastructure
- Defining a collection target helps to limit validity/visibility of collection for a given target

```
# echo ”setenv MODULES_COLLECTION_TARGET [uname nodename]” \
    >>/usr/local/Modules/init/modulerc
```

- Check collection handling when a target is set

```
# su - user1
$ module savelist
$ module load datadir
$ module save
$ find ~/.module
$ module restore
$ module saverm
```

Extend `module` command with site-specific Tcl code

- `modulecmd.tcl` sources a `siteconfig.tcl` script at the beginning of its main procedure code
- Enables to supersede any global variable or procedure definitions with site-specific code.

```
# mkdir /usr/local/Modules/etc

# cat <<EOF >/usr/local/Modules/etc/siteconfig.tcl
rename ::cmdModuleLoad ::__cmdModuleLoad
proc cmdModuleLoad {args} {
    report ”loading \$args”
    return [eval __cmdModuleLoad \$args]
}
EOF

# su - user1
$ module load null
```

Environment Modules project

- User community discussion and support via mailing-list
  `modules-interest@lists.sourceforge.net`
- Open development using modern code forge
  `https://github.com/cea-hpc/modules`
- Frequent software releases (both bugfix and feature releases)
- Continuous integration with >6k tests 

- Code coverage monitored 

- Automatically built and published documentation 

  `https://modules.readthedocs.io`

Current development focus

- Improved modulefile dependency specifications
- Automatic modulefile dependency management
- Meta-alias or package
- Modulefile cache mechanism

Automatic dependency management

- Advanced automatic handling
  - Load automatically pre-required modulefiles
  - Unload automatically conflicting modulefiles
  - Changing requirement reloads dependent modules
  - Unload dependent module unloads automatically loaded dependencies
- Handle multiple dependency chains as long as there is no conflict between them

DE LA RECHERCHE À L'INDUSTRIE

cea

How to contribute?

- Ask/answer questions, request/discuss features on mailing-list
- Spot bugs and report them (with a test case to integrate in non-regression testsuite)
- Share your modulecmd/modulefiles management recipes (docs for inclusion on ReadTheDocs)
- Code new features (discuss ideas on mailing-list or ticket then dive into `modulecmd.tcl`)

New features to develop

- New shell or scripting language support
- Colored output
- i18n
- Modulefile evaluation at directory changing time (à la `direnv`)
- Evaluate `stdin` like a modulefile content
- SAT solver to compute modulefile dependencies
- *<your idea here>*

Thanks for your attention

- Website: `http://modules.sourceforge.net/`
- Code: `https://github.com/cea-hpc/modules`
- Documentation: `https://modules.readthedocs.io`
- Questions, feedback, new use-cases, want to participate: `modules-interest@lists.sourceforge.net`

Commissariat à l’énergie atomique et aux énergies alternatives
Centre de Bruyères-le-Châtel │ 91297 Arpajon Cedex
T. +33 (0)1 69 26 40 00 │ F. +33 (0)1 69 26 40 00
Établissement public à caractère industriel et commercial
RCS Paris B 775 685 019

DAM
DIF
DSSI