DE LA RECHERCHE À L'INDUSTRIE

Modules

Automatic and consistent handling of modulefile dependencies

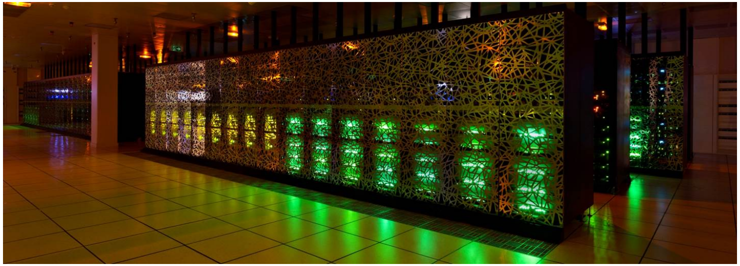
Xavier Delaruelle <xavier.delaruelle@cea.fr>

4th EasyBuild User Meeting
February 1st 2019, UCLouvain, Louvain-la-Neuve

www.cea.fr

whoami

- I am Xavier Delaruelle
- *Environment Modules* project leader since July 2017
- Work at CEA, a large research institute in France
- In the High Performance Computing (HPC) field



2018 report

- Many releases
  - **`4.1.0`** (2018-01-15), `4.1.1`, `4.1.2`, `4.1.3`, `4.1.4` **in RHEL8**
  - **`4.2.0`** (2018-10-18), `4.2.1`
- v4.1 serie has focused on tests and OS support
  - Code coverage maximized coverage 99%
  - Validate support for OS X, Solaris, FreeBSD and Windows
- v4.2 serie mainly focused on modulefile dependency management

A dive into dependency management enhancements

- Consistency
- Automation

Modulefile dependency consistency

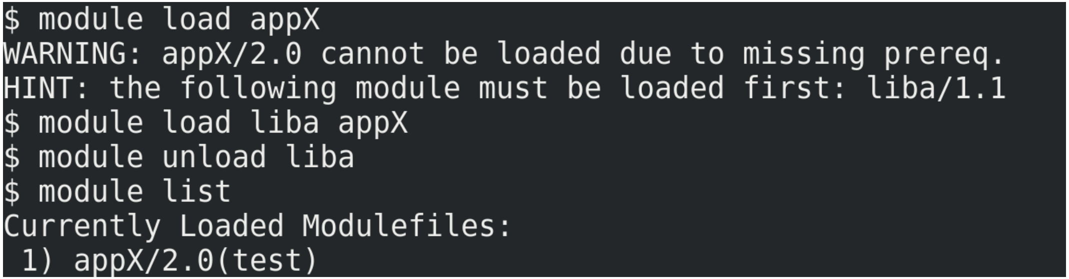
How to express a dependency?

- Requirement: a given modulefile should also be loaded
  - `prereq`: bare requirement declaration
  - `module load`: requirement declaration + load attempt
- Conflict: a given modulefile should not be loaded
  - `conflict`: bare conflict declaration
  - `module unload`: conflict declaration + unload attempt

Modulefile dependency consistency

Situation prior version 4.2

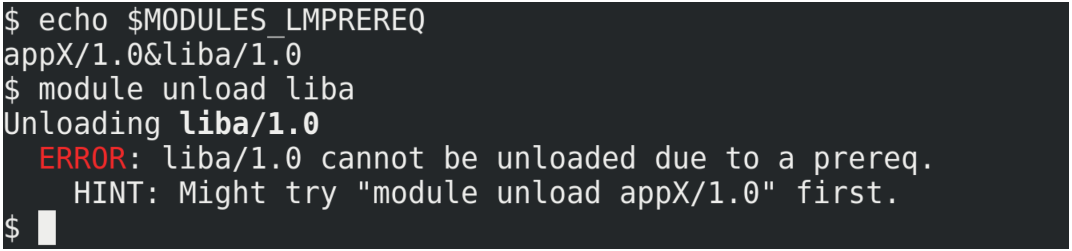
- No real consistency prior v4.2 (like on all other `module` implementation)



```
$ module load appX
WARNING: appX/2.0 cannot be loaded due to missing prereq.
HINT: the following module must be loaded first: liba/1.1
$ module load liba appX
$ module unload liba
$ module list
Currently Loaded Modulefiles:
 1) appX/2.0(test)
```

Making requirements consistent

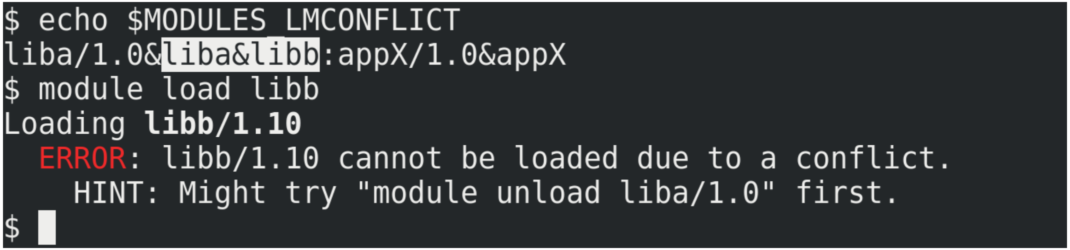
- Keep track of the requirements defined by loaded modules
- By using an environment variable (`MODULES_LMPREREQ`)
- Check this information when a module unload is attempted to ensure environment consistency is satisfied



```
$ echo $MODULES_LMPREREQ
appX/1.0&liba/1.0
$ module unload liba
Unloading liba/1.0
  ERROR: liba/1.0 cannot be unloaded due to a prereq.
    HINT: Might try "module unload appX/1.0" first.
$ 
```

Making conflicts consistent

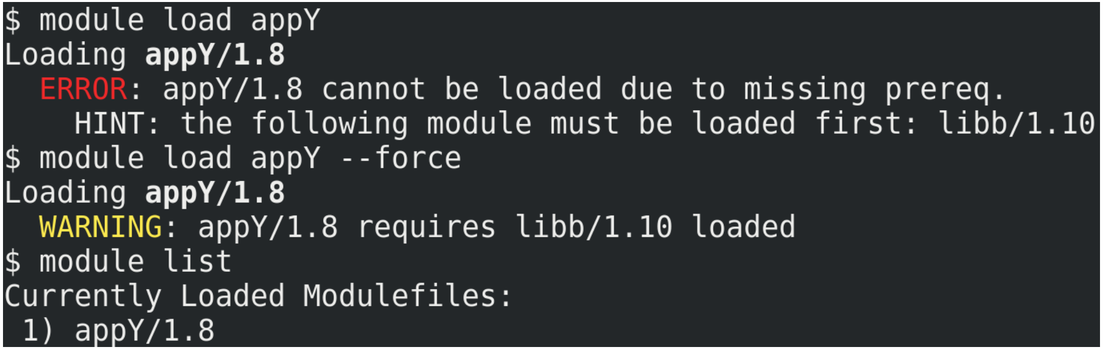
- Keep track of the conflicts defined by loaded modules
- By using an environment variable (`MODULES_LMCONFLICT`)
- Check this information when a module load is attempted to ensure environment consistency is satisfied



```
$ echo $MODULES_LMCONFLICT
liba/1.0&liba&libb:appX/1.0&appX
$ module load libb
Loading libb/1.10
  ERROR: libb/1.10 cannot be loaded due to a conflict.
    HINT: Might try "module unload liba/1.0" first.
$ 
```

By-passing consistency

- Use `--force` switch to by-pass any requirement or conflict
- Which results in an environment with some unsatisfied dependency rules



```
$ module load appY
Loading appY/1.8
  ERROR: appY/1.8 cannot be loaded due to missing prereq.
    HINT: the following module must be loaded first: libb/1.10
$ module load appY --force
Loading appY/1.8
  WARNING: appY/1.8 requires libb/1.10 loaded
$ module list
Currently Loaded Modulefiles:
 1) appY/1.8
```

Automated module handling

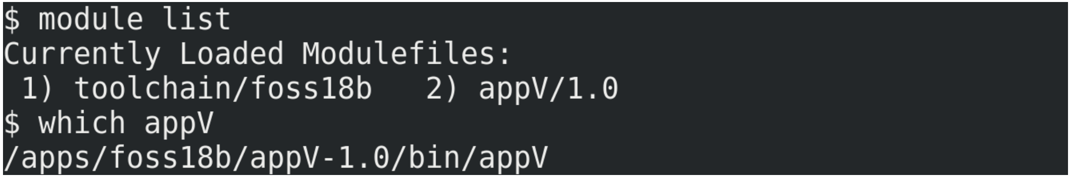
- Automatic management of the dependencies between modulefiles
- Set of mechanisms applied when a module is loaded or unloaded

Examples on the next slides are made with automated module handling mode enabled

```
$ export MODULES_AUTO_HANDLING=1
```

Module load order matters

- A requirement is loaded prior the module which depends on it
- So dependent module could adapt its definition when loading



```
$ module list
Currently Loaded Modulefiles:
 1) toolchain/foss18b   2) appV/1.0
$ which appV
/apps/foss18b/appV-1.0/bin/appV
```

- Unload must be done in reverse order to unset in a situation equivalent than during load

Automated module handling

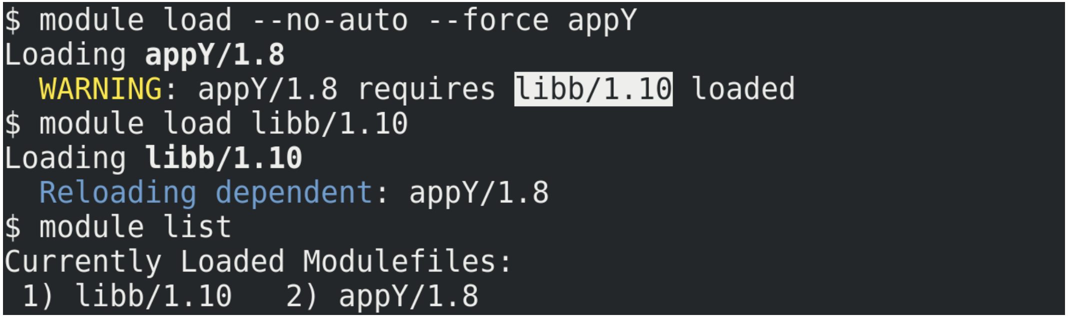
When loading a modulefile

1. Dependent Reload (unload phase)
2. Requirement Load
3. *Load of the asked modulefile*
4. Dependent Reload (load phase)

Automated mechanisms in load mode

Dependent Reload

- Reload of the modulefiles declaring a requirement onto loaded modulefile or declaring a requirement onto a modulefile part of this reloading batch

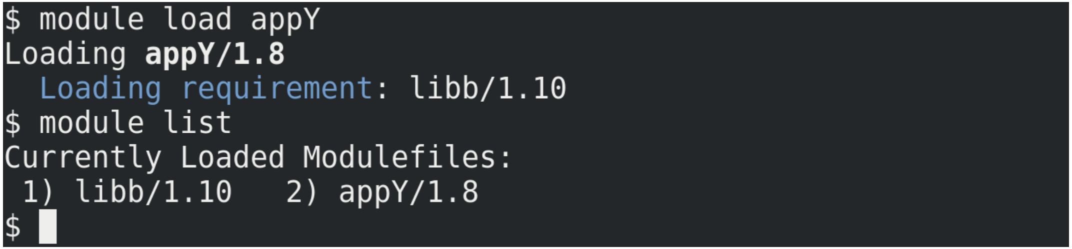


```
$ module load --no-auto --force appY
Loading appY/1.8
  WARNING: appY/1.8 requires libb/1.10 loaded
$ module load libb/1.10
Loading libb/1.10
  Reloading dependent: appY/1.8
$ module list
Currently Loaded Modulefiles:
 1) libb/1.10   2) appY/1.8
```

Automated mechanisms in load mode

Requirement Load

- Load of the modulefiles declared as a requirement of the loading modulefile



```
$ module load appY
Loading appY/1.8
  Loading requirement: libb/1.10
$ module list
Currently Loaded Modulefiles:
 1) libb/1.10   2) appY/1.8
$ 
```

Automated module handling

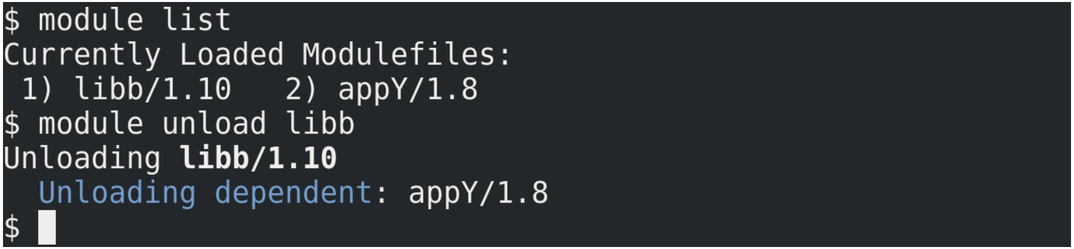
When unloading a modulefile

1. Dependent Reload (unload phase)
2. Dependent Unload
3. *Unload of the asked modulefile*
4. Useless Requirement Unload
5. Dependent Reload (load phase)

Automated mechanisms in unload mode

Dependent Unload

- Unload of the modulefiles declaring a non-optional requirement onto unloaded modulefile or declaring a non-optional requirement onto a modulefile part of this unloading batch

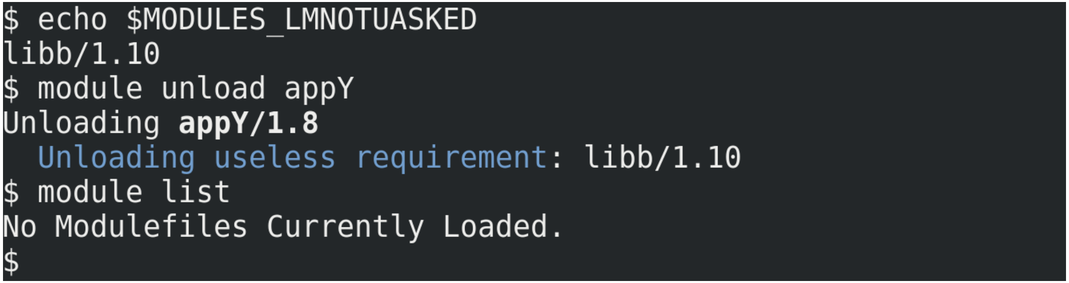


```
$ module list
Currently Loaded Modulefiles:
 1) libb/1.10   2) appY/1.8
$ module unload libb
Unloading libb/1.10
  Unloading dependent: appY/1.8
$ 
```

Automated mechanisms in unload mode

Useless Requirement Unload

- Unload of the required modulefiles that have been automatically loaded for either the unloaded modulefile, an unloaded dependent modulefile or a modulefile part of this useless requirement unloading batch
- Automatically loaded modulefiles are tracked with an environment variable (`MODULES_LMNOTUASKED`)

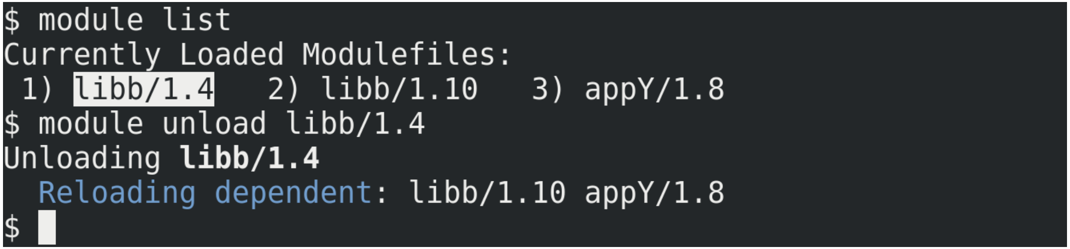


```
$ echo $MODULES_LMNOTUASKED
libb/1.10
$ module unload appY
Unloading appY/1.8
  Unloading useless requirement: libb/1.10
$ module list
No Modulefiles Currently Loaded.
$
```

Automated mechanisms in unload mode

Dependent Reload

- Reload of the modulefiles declaring a conflict or an optional requirement onto either the unloaded modulefile, an unloaded dependent or an unloaded useless requirement or declaring a requirement onto a modulefile part of this reloading batch



```
$ module list
Currently Loaded Modulefiles:
 1) libb/1.4   2) libb/1.10   3) appY/1.8
$ module unload libb/1.4
Unloading libb/1.4
  Reloading dependent: libb/1.10 appY/1.8
$ 
```

Automated module handling

Further improvements

- To complete this work, additional mechanisms should be added
  - Conflict Unload
  - Loaded Reload
- Evaluate modulefile dependencies ahead of the actual load evaluation
- Use *SAT* algorithm to solve dependencies

Modules 4 and introduced behavior changes

- Each new feature introduced that changes exiting behaviors are disabled by default
- To stay in version 4 as long as possible
- Feature can be enabled or disabled at various level:
  - at the `./configure` script time (`--enable-auto-handling`)
  - with an environment variable (`MODULES_AUTO_HANDLING`)
  - with a command-line switch (`--auto` or `--no-auto`)
- The `config` sub-command helps to track and set those parameters *(new in v4.3)*

```
$ module config auto_handling 1
```

So what?

- Q: What could be done with these consistency and automation features?
- A: Another way to write and organize your build-related modulefiles

Toward a more flexible module naming scheme

- Only one modulefile per software-version
- Describing its dependencies and especially on what toolchains/flavors it is available

```
prereq toolchain/foss18a toolchain/foss18b
```

- At load time, this modulefile checks what dependencies are loaded and adapt its installation path accordingly

```
# get loaded toolchain name
set tc [string range \
  [module-info loaded toolchain] 10 end]
append-path PATH /apps/$tc/$soft-$version/bin
```

Toward a more flexible module naming scheme

A reduced number of modulefiles

- Hierarchy setup

```
modulefiles/toolchain/foss18a
modulefiles/toolchain/foss18b
modulefiles/toolchain/foss19a
modulefiles-foss18a/app/3.2
modulefiles-foss18b/app/3.2
modulefiles-foss19a/app/3.2
```

- Dependency setup

```
modulefiles/toolchain/foss18a
modulefiles/toolchain/foss18b
modulefiles/toolchain/foss19a
modulefiles/app/3.2
```

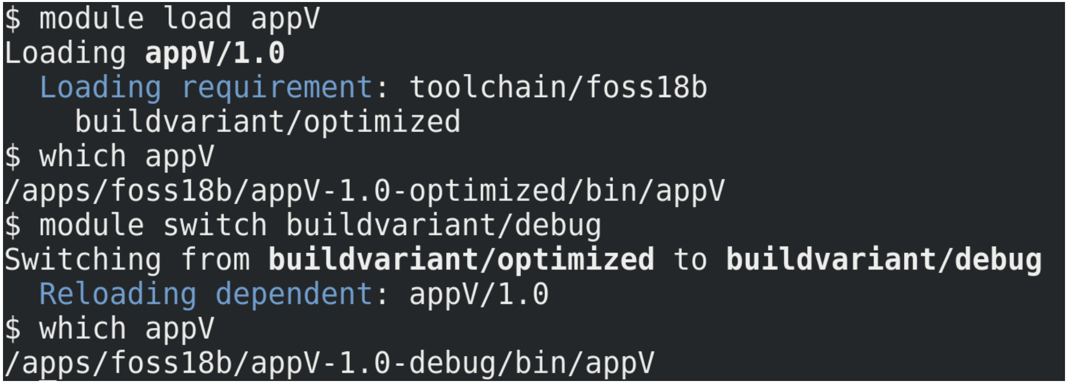
Toward a more flexible module naming scheme

Other advantages

- Free to organize your modulefiles in modulepaths the way you want
- Open path for multi-hierarchy
- All available modulefiles seen from the start

Toward a more flexible module naming scheme

Small multi-hierarchy example



```
$ module load appV
Loading appV/1.0
  Loading requirement: toolchain/foss18b
    buildvariant/optimized
$ which appV
/apps/foss18b/appV-1.0-optimized/bin/appV
$ module switch buildvariant/debug
Switching from buildvariant/optimized to buildvariant/debug
  Reloading dependent: appV/1.0
$ which appV
/apps/foss18b/appV-1.0-debug/bin/appV
```

Toward a more flexible module naming scheme

Things to adapt

- Shift the way modulefiles are generated
- Today, one modulefile equals to one build
- The proposed module naming scheme implies one modulefile corresponds to one or more builds
- Means modulefile has to be updated each time a build is added or removed

Toward a more flexible module naming scheme

Things to refine on Modules (1)

- Add an option for the `avail` sub-command to filter-out modulefiles not compatible with loaded modules

```
$ module avail app
----------- /apps/modulefiles -----------
app/3.2  app/4.1(default)
$ module list
Currently Loaded Modulefiles:
 1) toolchain/foss18b
$ module avail --only-compatible app
----------- /apps/modulefiles -----------
app/3.2
```

Toward a more flexible module naming scheme

Things to refine on Modules (2)

- Introduce a *Load Compatible* mechanism
  - If no specific version is asked for when loading a given module
  - Choose one compatible with the currently loaded modules
  - Instead of the default version

```
$ module load app
$ module list
Currently Loaded Modulefiles:
 1) toolchain/foss18b  2) app/3.2
```

Shaping future?

- Okay, but what is the current module naming scheme trend?

Software hierarchy

- A concept popularized by Lmod
    - Modulefiles are organized into modulepaths relative to the toolchain they have been built on
    - Core modulefiles enable the toolchain-specific modulepath at load
    - Which gives visibility of the available modules for this toolchain
    - And triggers automatic reload of modules to match the new toolchain instead of the previous one
- As of today, software hierarchy is the best known and supported way to organize the modulefile tree

A software hierarchy support in Modules?

- Software hierarchy is what people are currently asking for
- So would go for it to get compatible with what is out there (setup made by sites and naming scheme supported by tools like EasyBuild)

How to make Modules support software hierarchy?

- Treat ”`module use`” command in modulefile as a dependency
- And let it handled by the *Dependent Unload* and *Dependent Reload* mechanisms

DE LA RECHERCHE À L'INDUSTRIE

cea

A Lmod/Modules comparison (as of January 2019)

Lmod unique features

software hierarchy · modulefile cache · Lua modulefile support · one name rule · module auto-swap · inactive modules · `depends_on` · path entry priority · advanced version requirement (>, =, <, ...) · dynamic module hide · find best module version · `pushenv` · `family` · i18n · `ml` · nag message · in-depth documentation

Modules unique features

modulefile constraint consistency · automated module handling · explicit conflict constraint · modulescript sourcing · virtual modules · environment direct handling command · full path modulefile

Next steps: version `4.3`

- ETA: around March
- Restore some features of version 3.2, like the `clear` sub-command
- `config` sub-command
- Colored output
- Case insensitiveness

Next steps: version 4.4

- ETA: around June
- Focussed on Software hierarchy support

Other cool stuff afterward

- Modulefile cache
- Expiring modulefiles
- Sourcing modulescript when changing directory, à la `direnv`
- Support for modulefiles written in Python
- `module stash` à la `git`, relying on collections

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