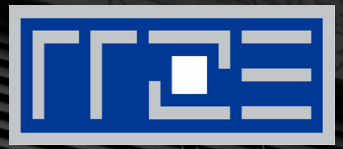
REGIONALES RECHENZENTRUM ERLANGEN [RRZE]



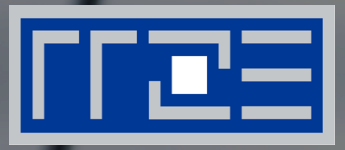
Modules and Software Installation

HPC Services, RRZE

Which packages are available

- RRZE HPC team uses standard distribution packages
  - On frontend nodes: Full installation and easy to add packages
  - On compute nodes: Stripped installation due to diskless setup
    - The OS and software is occupying memory
    - More software → Less memory for your applications
  - HPC software is centrally installed and accessible through `modules`
    - Compilers, libraries, commercial and open software

- If multiple users reasonably request packages, we can add them



The `module` system

The `modules` system

- The **`modules`** system is a de-facto standard on HPC systems
- BUT: Every facility uses the modules differently
  - RRZE: <name_of_software>/<version>(-<requirements>)
  - Examples: intel64/19.0up05 and openmpi/2.0.2-gcc
  - Some centers use categories like MATH, LIBRARIES, etc. (**`module load MATH; module load module_in_math`**)
- The **`modules`** system affects only current and sub-shell!

The `modules` system

RRZE: Modules in categories ‚deprecated' and ‚testing' might be deleted without notice

Show available modules: **module avail**

```
$ module avail
------------------------------ /apps/modules/data/applications ------------------------------
cp2k/2.5.1                      gromacs/2019.3-mkl-IVB(default)  openfoam/4.1-trusty
gnuplot/5.0.1                   gromacs/2019.5-mkl-IVB           openfoam/5.0-trusty
gromacs/2016.4-mkl              gromacs/2019.5-mkl-IVB-CUDA101   paraview/4.2.0-x86_64
[…]
```

Show all modules of software: **module avail gcc**

```
$ module avail gcc
------------------------------ /apps/modules/data/applications ------------------------------
-------------------------------/apps/modules/data/development -------------------------------
gcc/4.9.3 gcc/5.4.0 gcc/6.1.0 gcc/7.3.0 gcc/8.1.0
```

The `modules` system

Some module load / unload other modules automatically

Load a module: `module load <modulename>`

```
$ module load intel64
$ module list
Currently Loaded Modulefiles:
 1) intelmpi/2017up04-intel   2) mkl/2017up05              3) intel64/17.0up05
```

Unload a module: `module unload <modulename>`

```
$ module unload intel64
$ module list
No Modulefiles Currently Loaded.
```

The `modules` system

Command	What it does
`module avail`	List available modules
`module whatis`	Shows over-verbose listing of all modules
`module list`	Shows which modules are currently loaded
`module load <pkg>`	Loads module *pkg*, i.e., adjusts environment
`module load <pkg>/<version>`	Loads specific version of *pkg* instead of default
`module unload <pkg>`	Undoes what the load command did
`module help <pkg>`	Shows a detailed description of *pkg*
`module show <pkg>`	Shows what environment variables pkg modifies and how

The `modules` system and the batch system

- Shebang in batch scripts: `#!/bin/bash -l`
- Some packages compile library paths into the application
  → Don‘t rely on it! Always load required modules
- On meggie in batch scripts:
  - `#SBATCH --export=NONE`
  - `unset SLURM_EXPORT_env`

and load required modules

Software Installation

Software installation
The handy but later annoying approach

- Many packages use **`configure`** & **`make`**:
  - Setup: **`./configure --prefix=$HOME/Apps (other opts)`**
  - Compilation: **`make`**
  - Installation: **`make install`**
- Or **`cmake`** & **`make`**:
  - Setup: **`mkdir BUILD && cd BUILD`**
    `CMAKE_INSTALL_PREFIX=$HOME/Apps cmake ..`
  - Compilation: **`make`**
  - Installation: **`make install`**

Software installation
The handy but later annoying approach

- At compilation add `$HOME/Apps/include` and `$HOME/Apps/lib`
- Add library path at runtime (only current shell):

```
export LD_LIBRARY_PATH=$HOME/Apps/lib:$LD_LIBRARY_PATH
```

- But how to uninstall?

  Commonly there is no `make uninstall`
- But how to upgrade?

  Files from old install might not be overwritten

Software installation
Further pitfalls

- Check build before running in production!

- Some packages require special build preparations:
  - GROMACS: **`build_type=Release`** or binary crashes often
  - Adjust paths to dependencies in config files
  - Activate required feature only with non-default build flag

- What to do with compiler errors?

Software installation
Recommendations

- If you install own software, use distinct folder for each version:
  - Separate installations: **`$HOME/Apps/libX-(<cluster>-)Y.Z`**
  - Simple to delete old versions (**`rm -r $HOME/Apps/libX-Y.Y`**)
  - In case of space problems: Use **`$WORK`** (No backups!)

- Helpful but not a silver bullet: package managers
  - easy_build
  - spack

Package manager Spack

- Developed at LLNL
- Resolves complicated dependencies among the packages

- Issue: Spack always rebuilds complete tree, hard to integrate common OS packages
  - Check **`/apps/SPACK/spack/etc/spack/packages.yaml`** for cluster-specific settings (like always use installed SLURM version)
  - Problem: Space in **`$HOME`** directory limited, may use **`$WORK`**
- Issue: Currently no way to use Intel-MPI 19.X due to bug

Software installation
Language specific package managers

- Python: pip, virtualenv, conda
  System Python contains only few modules, try Anaconda Python!
- R: CRAN
- Perl: CPAN

- Use **`--user`** flag (or similar) to install in **`$HOME`**
  (for **`$WORK`** use **`export PYTHONUSERBASE=$WORK`**)

Thank you very much

HPC@RRZE

Adding `module` for own installations

- Assuming you have use distinct folders (`~/Apps/libX-x.y.z`)
  - Create hidden folder: `mkdir -p ~/.modfiles/libX/`
  - For each version create a file: `edit ~/.modfiles/libX/x.y.z`

```
#%Module1.0
## libX
##
module-whatis "libX: The library for doing stuff with X"
conflict libX
set          pkghome                 ~/Apps/libX-x.y.z
prepend-path PATH                    $pkghome/bin
prepend-path LD_LIBRARY_PATH         $pkghome/lib
setenv       LIBX_LIBDIR             "$pkghome/lib"
setenv       LIBX_INCDIR             "$pkghome/lib"
```

  - Finally: `module use ~/.modfiles` (e.g. in `~/.bashrc`)