The University of Queensland

Research Computing Centre

HPC Software User Guide

Document Status

This update: October 5 2018 by David Green

More about Software

- The ~~environment modules mechanism~~ Lmod modules mechanism is the best way to work with installed software on HPCs.
- The environment modules modify your linux environment so you can find the software you want to use and their manual pages and other settings.
- Some background about modules structure on HPCs is provided by a README module
  `module display README/Modules`
- Some other information about the use of modules is available in the **README** and **HOWTO** sections of the modules.
  `module avail README`
  `module avail HOWTO`
- Modules are loaded when the software is required and unloaded or purged when no longer required.
- You can even create and use your own personal module files.
  `module display HOWTO/PersonalModules`
- The Lmod modules system uses a per-user cache file to speed operations up for you.
  You may need to refresh your Lmod cache to be able to access a freshly installed module.
  You can refresh your Lmod cache file by running the command
  `module --ignore-cache avail`

Summary of module command options

- `module avail`
  A list of all available modules.
- `module display ModuleName`
  `module show ModuleName`
  A summary of what the loading the module will do.
- `module help ModuleName`
  A copy of the embedded help in the module.

- `module whatis ModuleName`
  A brief synopsis of the module.
- `module load ModuleName`
  `module add ModuleName`
  Loads/adds the module ModuleName
- `module list`
  A list of previously loaded modules.
- `module unload ModuleName`
  `module rm ModuleName`
  Unloads/removes the previously loaded module.

See module help for more information and options.

Usage Examples

- The list of available modules can be requested using

  ```
  module avail
  ```

  Use the -t option for terse single column output.
- **Not all modules show up when you run the `module avail` command.**
  Some modules that depend on compiler modules are hidden from view until the compiler module has been loaded. Other modules have a replica that fails to load whenever the compiler module has not been loaded.

For example, if you need to the BOOST maths library that was built with the same compiler as the software you want to use it with, then you need to first load the compiler module that it was built with (intel or gnu).

To load a module you simply

```
module load Name_of_Module
```

Although the default (or highest version number) module will get loaded if you do not specify a version, **you are strongly advised to always load modules by specific version** so you will know which one you used in, for example, a job submission script.

To see what a module load would do for you

```
module display Name_of_Module
```

To see what help a module provides

```
module help Name_of_Module
```

Not all modules provide extensive help.

Sometimes ... module avail ... fails

Sometimes the `module avail` command will fail in rather inglorious fashion:

```
davidg@flashlite2:~> module avail
/usr/bin/lua: /usr/share/lmod/lmod/libexec/Spider.lua:308: stack overflow
stack traceback:
.
.
.
```

If this happens you should delete your cache directory using the command `rm -rf $HOME/.lmod.d/.cache` and try `module avail` again.
You may need to repeat these steps a couple of times, and perhaps logout and log back in again.

It is sometimes necessary to perform this step on a different login node (same cluster or different cluster).

If it still persists after all that, then please submit a support request via email to **rcc-support@uq.edu.au** as it may be syntax error in a recently added or edited module file.

We will usually check that updated modules work for a regular user logins but sometimes that step can be missed when things are busy.

Using Your Own Software Modules

You are able to write and use your own module files.
This may be useful if you build your own software, or have a special combination of modules you need to use on a regular basis.
It can make things more convenient for setting PATH and LD_LIBRARY_PATH and the like.

See environment modules project website for information: http://modules.sourceforge.net/ for help with syntax.
You could also copy a similar module file from the many that are on the system.
In some circumstances you may wish to link to a special or a not-yet-public module file.

The steps are as follows:

1. Create the directory for your private module files at $HOME/privatemodules
2. Create your module file(s) in that place.
3. Load the use.own module that is in the `/usr/share/Modules/modulefiles` section of `module avail` output.
4. Load your personal module file by its path relative to $HOME/privatemodules

See also `module help use.own`.

A Sample Session

So as a worked example ...

```
uqdgree5@tinaroo1:~> ls -sal privatemodules/embeddednimrod/7
0 lrwxrwxrwx 1 uqdgree5 qris-uq 42 Sep 11 13:25 privatemodules/embeddednimrod/7 -> /gpfs1/sw7/RCC/NimrodG/embedded/m

uqdgree5@tinaroo1:~> module purge

uqdgree5@tinaroo1:~> module load use.own

uqdgree5@tinaroo1:~> module load embeddednimrod/7a

uqdgree5@tinaroo1:~> module list

Currently Loaded Modules:
  1) use.own   2) embeddednimrod/7a
```

A Sample Module File

The module file referred to in the sample session has most of the features you are likely to need.

```
uqdgree5@tinaroo1:~> cat privatemodules/embeddednimrod/7a.lua
--
-- Embedded Nimrod/G Modulefile, Lmod version
--
local nimrod_version = "nimrod-0.8.5"
local base_path = "/gpfs1/sw7/RCC/NimrodG/embedded2"
local nimrod_home = pathJoin(base_path, "opt/nimrod")
local java_home = pathJoin(base_path, "lib/jvm/jdk-10.0.1")
local qpid_home = pathJoin(base_path, "opt/qpid-broker/7.0.4")

whatis("Name: Embedded Nimrod/G")
whatis("Version: "..nimrod_version)
whatis("URL: https://rcc.uq.edu.au/nimrod")

help([[
To run a planfile:
    nimrun /path/to/planfile

To validate a planfile:
    nimrod compile --no-out /path/to/planfile
]])

setenv("JAVA_HOME", java_home)
setenv("NIMROD_HOME", nimrod_home)
setenv("QPID_HOME", qpid_home)

prepend_path("PATH", pathJoin(java_home, "bin"))
prepend_path("PATH", pathJoin(base_path, "bin"))

setenv("OMP_NUM_THREADS", os.getenv("OMP_NUM_THREADS") or "1")
```

The example above is written in the LUA language.
If you want to see more examples of module files, then locate a module file (`module -t avail` will tell you the location of the groups of modules) and use the `cat` command to view the content of the module file.
Be aware that some module files on the HPC are written in TCL/Tk syntax and get translated into LUA language syntax as they are loaded.

Compilers and Related Matters Debuggers

Because a number of the ROCKS software tools have been built for both intel and gnu compilers, you sometimes need to first load your preferred compiler module before loading the module for that software tool.

The list of such software is
`boost hdf4 hdf5 ipm lapack mvapich2 mxml nco openmpi2 openmpi papi parmetis pdt petsc scalapack slepc sprng sundials superlu tau`

Consider a scenario where you are compiling an OpenMPI v2 based code using the Intel compilers.

If you did not first load a compiler module, then you would get this (rather unhelpful) message:

```
uqdgree5@tinaroo1:~> module load openmpi2_ib/2.0.2
Lmod has detected the following error:  Unable to load module:
openmpi2_ib/2.0.2
     /opt/modulefiles/mpi/openmpi2_ib/2.0.2: Non-zero status returned

While processing the following module(s):
    Module fullname    Module Filename
    ---------------    ---------------
    openmpi2_ib/2.0.2  /opt/modulefiles/mpi/openmpi2_ib/2.0.2
```

however if you first load the compiler, then the openmpi things work as expected.

```
uqdgree5@tinaroo1:~> module load intel
uqdgree5@tinaroo1:~> module list

Currently Loaded Modules:
  1) intel/2018.2.046

uqdgree5@tinaroo1:~> module load openmpi2_ib

uqdgree5@tinaroo1:~> module list

Currently Loaded Modules:
  1) intel/2018.2.046   2) openmpi2_ib/2.0.2
```

```
uqdgree5@tinaroo1:~> module display openmpi2_ib
-------------------------------------------------------------------------------
   /opt/modulefiles/mpi/openmpi2_ib/2.0.2:
-------------------------------------------------------------------------------
whatis("openmpi2_ib mpi stack ")
whatis("Version: 2.0.2 ")
whatis("Compiler: gnu intel ")
setenv("MPIHOME","/opt/openmpi2/intel/ib")
prepend_path("PATH","/opt/openmpi2/intel/ib/bin")
prepend_path("LD_LIBRARY_PATH","/opt/openmpi2/intel/ib/lib")
```

PBS Script Snippets for Specific Applications

These sample scripts are intended to enhance the productivity of users of specific applications.
Use at your own risk.

Gaussian

- Gaussian creates checkpoint file(s) as it runs. This involves writing a lot of data to disk intermittently.
- It is best done using local disk on a compute node rather than a network drive.
- Our license for Gaussian is limited to single node computations, so it is well suited to using **`TMPDIR`**.
- The gaussian software module will automatically set the GAUSS_SCRDIR environment variable to the TMPDIR.
- Checkpoint files are written to PBS_O_WORKDIR so you can monitor progress.
- If you know roughly how much disk space a job will require for temporary files that get written into GAUSS_SCRDIR you should request it via PBS.

Job Script Snippets

```
#You can request a minimum amount of free scratch disk for the job start on a node
#PBS -l select=1:ncpus=12:mem=4000MB:scratch=50GB
```

```
module load gaussian/g09
printf "The location of temporary files is going to be %s\n" $GAUSS_SCRDIR
```