Environment Modules

- Why use modules
- Loading modules
- Unloading modules
- Module sets
  - Default module set
- Using environment modules with SGE jobs and qlogin

The HPC cluster system at the University of Iowa has many software packages installed and sometimes different versions of the same package. These software programs are installed in several different directories. These software packages are made available for use via environment modules. See Argon Software List - HPC Documentation - UIowa Wiki for the software installed on Argon.

Like previous generation UI HPC systems, Argon uses environment modules for managing the shell environment needed by software packages. Argon uses LMod rather than the TCL modules used in previous generation UI HPC systems. More information about Lmod can be found in the Lmod: A New Environment Module System. Briefly, Lmod provides improvements over TCL modules in some key ways. One is that Lmod will automatically load and/or swap dependent environment modules when higher level modules are changed in the environment. It can also temporarily deactivate modules if a suitable alternative is not found, and can reactivate those modules when the environment changes back. We are not using all of the features that Lmod is capable of so the modules behavior should be very close to previous systems but with a more robust way of handling dependencies.

The software packages are built as sets, called "stacks". Each stack provides a set of software packages with a self-consistent set of dependencies. This scheme allows us to keep multiple versions of software packages, each with a consistent set of dependencies with other packages in that set. This can make switching versions of a particular package a little more challenging but the best way to address that situation is to load a different stack in a different ssh session.

Why use modules

Using modules allows one to set the appropriate environment variables needed for the respective software program. Often, this is simply adding the program to the $PATH variable, but software containing libraries and headers will also set $LD_LIBRARY_PATH and $CPATH. Any other variable that the software may need can be set and so the contents of the modules can be fairly simple or complex.

There are several advantages to using environment modules to set up your environment.

1. ease of use
2. ability to revert to your previous environment
3. ability to easily switch your environment to try different versions of a program

To view the modules that you currently have loaded

<table>
<thead>
<tr>
<th>Currently Loaded Modules:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) intel/2017.4</td>
</tr>
<tr>
<td>2) daal/2017.4</td>
</tr>
<tr>
<td>3) intel_debugger/2017.4</td>
</tr>
<tr>
<td>4) ipp/2017.4</td>
</tr>
<tr>
<td>5) mkl/2017.4</td>
</tr>
<tr>
<td>6) tbb/2017.4</td>
</tr>
<tr>
<td>7) parallel_studio/2017.4</td>
</tr>
<tr>
<td>8) bzip2/1.0.6</td>
</tr>
<tr>
<td>9) jdk/8u121</td>
</tr>
<tr>
<td>10) xx/5.2.3</td>
</tr>
<tr>
<td>11) zlib/1.2.11</td>
</tr>
<tr>
<td>12) openmpi/2.1.2_parallel_studio-2017.4</td>
</tr>
</tbody>
</table>

To get a list of the modules installed
module avail

------------------------------------------------------- /opt/modules
--------------------------------------------------------
BEAST/1.8.4
CGAL/4.10_parallel_studio-2017.4
CMAQ/5.1_openmpi-2.0.1_parallel_studio-2017.1
CMAQ/5.1_openmpi-2.1.2_parallel_studio-2017.4 (D)
CMAQ/5.1_openmpi-2.1.2_parallel_studio-2017.4 (D)
CNVnator/0.3.3_python-2.7.14_openmpi-2.1.2_parallel_studio-2017.4
CNVnator/0.3.3_python-2.7.14_parallel_studio-2017.4 (D)
InsightToolkit/4.13.0
OpenBLAS/0.2.19_gcc-5.4.0
OpenBLAS/0.2.19 (D)
OpenBUGS/3.2.3
OpenFOAM/4.1_openmpi-2.1.2_parallel_studio-2017.4
OpenFOAM/4.1_openmpi-2.1.2_parallel_studio-2017.4
OpenFOAM/5.0_openmpi-2.1.2_parallel_studio-2017.4
OpenFOAM/5.0_openmpi-2.1.2_parallel_studio-2017.4 (D)
ParaView/5.4.0_openmpi-2.0.1_gcc-4.8.5
ParaView/5.4.1_openmpi-2.1.2_gcc-4.8.5 (D)
Platypus/0.8.1_python-2.7.13_parallel_studio-2017.1
R/3.3.2_gcc-5.4.0
R/3.3.2_openmpi-2.0.1_parallel_studio-2017.1
R/3.3.2_openmpi-2.1.2_parallel_studio-2017.4
R/3.4.3_openmpi-2.1.2_parallel_studio-2017.4
R/3.4.3_openmpi-2.1.2
R/3.4.3_parallel_studio-2017.4
R/3.4.3
R/3.4.3_gcc-5.4.0
R/3.5.1_openmpi-2.1.2_gcc-5.4.0
R/3.5.1_openmpi-2.1.2_parallel_studio-2017.4
R/3.5.1_openmpi-2.1.2
R/3.5.1_parallel_studio-2017.4
R/3.5.1 (D)
...
zlib/1.2.11_gcc-5.4.0
zlib/1.2.11_parallel_studio-2017.1
zlib/1.2.11 (L,D)
zstd/1.3.5

------------------------------------------------------- /usr/share/lmod/lmod/modulefiles/Core
--------------------------------------------------------
lmod/6.6.3 settarg/6.6.3

Where:
L: Module is loaded
D: Default Module

Use "module spider" to find all possible modules.
Use "module keyword key1 key2 ..." to search for all possible modules matching any of the "keys".

To narrow the list down you can specify a string to search for
module avail python

module -r avail ^python

------------------------------------------------------- /opt/modules
--------------------------------------------------------
python/2.7.13_parallel_studio-2017.1                  python/2.7.15
python/2.7.13                                        python/3.5.3
python/2.7.14_openmpi-2.0.1_parallel_studio-2017.1   python/3.5.3
python/2.7.14_openmpi-2.0.1                           python/3.6.4_openmpi-2.1.2_parallel_studio-2017.4
python/2.7.14_openmpi-2.1.2_parallel_studio-2017.4    python/3.6.4_openmpi-2.1.2
python/2.7.14_parallel_studio-2017.1                  python/3.6.4
python/2.7.14_openmpi-2.1.2                           python/3.6.4_openmpi-2.1.2_parallel_studio-2017.4
python/2.7.14_openmpi-2.1.2_parallel_studio-2017.4    python/3.7.0_openmpi-2.1.2
python/2.7.14                                        python/3.7.0
python/2.7.15_openmpi-2.1.2_parallel_studio-2017.4    python/3.7.0_parallel_studio-2017.4
python/2.7.15_openmpi-2.1.2                           python/3.7.0
python/2.7.15_openmpi-2.1.2_parallel_studio-2017.4    python/3.7.0
python/2.7.15                                        python/3.7.0

Where:
D:  Default Module

Use "module spider" to find all possible modules.
Use "module keyword key1 key2 ..." to search for all possible modules matching any of the "keys".

Loading modules

module load python/3.6.4

Note that if there are multiple versions in the module directory and no default is listed, the modules program will try to load the latest version that it finds. This may not be what you want. To avoid surprises, you should specify the entire module name and version. If there is only one version in a module directory then the module version does not have to be specified, it will load the only version it finds. However, you should still specify the entire name and version to avoid surprises when a second version is added at a later point in time. It is often the case that you need to reference the path to the software managed by the module, such as specifying directory paths while compiling software. There is a special variable set in every module file that points to the root directory of the installation. The nomenclature is $ROOT_MODULENAME. For example, the zlib module sets

$ROOT_ZLIB=/opt/apps/zlib/1.2.11

You can switch modules easily. For instance, if you want to compare results between different python versions
Unloading modules

Unloading an environment module will undo the changes that module made to the environment, restoring any variables set to their previous values.

Writing your own module files is not covered here but they are not that difficult to create. If there is some environment variable that should be set for a software package that we missed please send a note to research-computing@uiowa.edu.

Module sets

Lmod provides a mechanism to save sets of modules that can then be restored. This is a convenient way to switch out a whole bunch of modules to change the environment quickly. While we strongly advise keeping a clean and minimal environment in your shell startup, for those who wish to load modules at shell startup, module sets provides a better mechanism than loading individual module files. The reasons are that

1. Only one command is needed
2. The same command can be used at any time
3. Restoring a module set runs a module purge which will ensure that the environment, at least the part controlled by modules, is predictable.

To use this, simply load the modules that you want to have loaded as a set. Then run the following command.

module save

That will save the loaded modules as the default set. To restore that run

module restore

In addition to saving/restoring a default set you can also assign a name to the collection and save multiple collections.
If you have several different environments needed for different job types then saving a module set for each job type can save time and ensure consistency. There is also a technical reason to use the module save/restore feature as opposed to individual modules that involves how the LD_LIBRARY_PATH environment variable is handled at shell initialization.

One of the things that environment modules sets up is the $LD_LIBRARY_PATH. However, when a setuid/setgid program runs it unsets $LD_LIBRARY_PATH for security reasons. One such setuid program is the do login program that runs as part of an ssh session. This will leave you with a partially broken environment as a module is loaded, sets $LD_LIBRARY_PATH but then has it get unset before shell initialization is complete. This is worked around on previous systems by always forcing a reload of the environment module but this is not very efficient. Use module restore to load saved modules if you are loading modules from your ~/.bashrc or similar.

Other than the above items, and some other additional features, the environment modules controlled by Lmod should behave very similarly to the TCL modules on previous UI HPC systems.

Default module set

The default module set generally represents a set of modules that meet all of your needs and/or provide a starting point for loading other modules. This is particularly important for Argon, which uses independent stacks of software. Once you have your workflow worked out you can save a default set of modules, as described above. That set will include a specific “stack” module as well. This allows you to maintain consistency of the environment for your jobs.

Beginning with the Spring 2021 Maintenance, the default module set, if you have one, will be loaded by default. If there is no default module set, a default stack module will be loaded.

As the default stack module loaded at startup is changed over time, having a default module set ensures that your module environment will stay the same. You can then verify your jobs with a new stack and then update the default module set when you are ready.

Using environment modules with SGE jobs and qlogin

For qsub jobs, if you have not overridden the default SGE parameters then the entire environment is passed to the job. This is because the –V qsub flag is set in the default request. Since environment modules set up the environment then the environment set up by the modules will be passed. Since the list of the loaded modules is also part of the environment then the list of loaded modules will be passed as well. However, the above does not apply to qlogin sessions as that creates a fresh environment. See the Qlogin for Interactive Sessions - HPC Documentation - UIowa Wiki page for more information.

However, the recommendation for qsub jobs is to include the module load statements in your job script and not depend on the environment passed in from the submit host. That way, setting up the needed environment is part of the job and is thus more reproducible. If depending on loading modules before qsub then you have to remember to get the environment set up before job submission. When loading modules in your job script, it is best to use module sets. If you would prefer to make sure that the module loads are more explicit then make sure to do a

module purge

before any "module load ..." commands.