Environment Modules

The HPC cluster system at the University of Iowa has many software packages installed and sometimes different versions of the same package. In general, these software programs are installed in the /opt/apps directory.

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.

Lmod provides a mechanism to save a set 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 this 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
```

That command could then be put in your shell initialization file. In addition to saving/restoring a default set you can also assign a name to the collection and save multiple collections.

```
module save mymodules
module restore mymodules
```

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 duo 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.

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

```
module list
```

Currently Loaded Modules:

```
Currently Loaded Modules:
1) intel/2017.4 4) ipp/2017.4 7) parallel_studio/2017.4 10) xz/5.2.3
2) daal/2017.4 5) mkl/2017.4 8) bzip2/1.0.6 11) zlib/1.2.11
3) intel_debugger/2017.4 6) tbb/2017.4 9) jdk/8u121 12) openmpi/2.1.2_parallel_studio-2017.4
```

To get a list of the modules installed
module avail

------------------------------------------- /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
To load a module, for example, python

```bash
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
To unload a module

```bash
$ module unload python/3.6.4
```

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.

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.

The recommendation for qsub jobs is to include the module load statements in your job script. 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.