Environment Modules

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>module list</th>
</tr>
</thead>
<tbody>
<tr>
<td>Currently Loaded Modules:</td>
</tr>
<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>
</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
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
InsightToolkit/4.13.0
OpenBLAS/0.2.19_gcc-5.4.0
OpenBLAS/0.2.19
OpenBLAS/0.2.19
OpenBUGS/3.2.3
OpenFOAM/4.1_openmpi-2.0.1_parallel_studio-2017.4
OpenFOAM/4.1_openmpi-2.1.2_parallel_studio-2017.4
OpenFOAM/5.0_openmpi-2.0.1_parallel_studio-2017.4
OpenFOAM/5.0_openmpi-2.1.2_parallel_studio-2017.4
ParaView/5.4.0_openmpi-2.0.1_gcc-4.8.5
ParaView/5.4.1_openmpi-2.1.2_gcc-4.8.5
Platypus/0.8.1_python-2.7.13_parallel_studio-2017.1
R/3.3.2_gcc-5.4.0
R/3.3.2_parallel_studio-2017.1
R/3.3.2_parallel_studio-2017.4
R/3.4.3_openmpi-2.0.1_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/3.5.1_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
...
zlib/1.2.11_gcc-5.4.0
zlib/1.2.11_parallel_studio-2017.1
zlib/1.2.11
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_parallel_studio-2017.1
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_parallel_studio-2017.4
python/2.7.14  python/3.7.0_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/2.7.15_openmpi-2.1.2_parallel_studio-2017.4
python/2.7.15_openmpi-2.1.2
python/2.7.15_parallel_studio-2017.4

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 pattern is like ROOT_MODULENAME. For example, the zlib/1.2.11 module sets this:

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
1. `module load python/2.7.15`
2. `module list`

Currently Loaded Modules:

```
1) bzip2/1.0.6         9) fontconfig/2.12.1  17) ncurses/6.0          25) grib_api/1.19.0  33) pcre/8.40
2) glib/2.51.1        10) freetype/2.7.1     18) instantclient/12.2.0.1.0  26) hdf5/1.8.18     34) proj/4.9.3
3) pkg-config/0.29.1   11) cfitsio/3.410    19) jasper/1.900.1       27) jpeg/9b         35) readline
/7.0
4) snappy/1.1.4       12) curl/7.52.1      20) libgeotiff/1.4.2     28) libffi/3.2.1     36) sqlite/3.
16.2
5) c-blosc/1.11.2     13) zip/2.1.1        21) libiconv/1.15        29) libpng/1.6.28    37) tiff/4.0.7
6) expat/2.2.0        14) hdf/4.2.12       22) unixODBC/2.3.4       30) lzo/2.09        38) zeromq/4.
2.5
7) xz/5.2.3           15) jdk/8u121        23) gdal/2.1.3           31) netcdf/4.4.1.1   39) zlib/1.
2.11
8) libxml2/2.9.4      16) libaio/0.3.111   24) geos/3.6.1           32) openjpeg/2.1.2   40) python/2.
7.15
```

3. `module swap python/2.7.15 python/3.6.4`
4. `module list`

Currently Loaded Modules:

```
1) bzip2/1.0.6         9) fontconfig/2.12.1  17) ncurses/6.0          25) grib_api/1.19.0  33) pcre/8.40
2) glib/2.51.1        10) freetype/2.7.1     18) instantclient/12.2.0.1.0  26) hdf5/1.8.18     34) proj/4.9.3
3) pkg-config/0.29.1   11) cfitsio/3.410    19) jasper/1.900.1       27) jpeg/9b         35) readline
/7.0
4) snappy/1.1.4       12) curl/7.52.1      20) libgeotiff/1.4.2     28) libffi/3.2.1     36) sqlite/3.
16.2
5) c-blosc/1.11.2     13) zip/2.1.1        21) libiconv/1.15        29) libpng/1.6.28    37) tiff/4.0.7
6) expat/2.2.0        14) hdf/4.2.12       22) unixODBC/2.3.4       30) lzo/2.09        38) zeromq/4.
2.5
7) xz/5.2.3           15) jdk/8u121        23) gdal/2.1.3           31) netcdf/4.4.1.1   39) zlib/1.
2.11
8) libxml2/2.9.4      16) libaio/0.3.111   24) geos/3.6.1           32) openjpeg/2.1.2   40) python/3.
6.4
```

5. `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.

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. If you typically need to load several modules, 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.

```bash
module save
```

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

```bash
module restore
```

Note that the above command will restore your modules to the system default if you do not have a default module set saved. 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 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.

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.

After the April 2021 HPC maintenance, the default module set, if you have one, will be loaded by default. Since your default module set will be loaded automatically, you should remove a ”module restore” command from your shell startup file, such as ~/.bashrc, as it is not necessary and will just duplicate what is done by default.

Having a default module set ensures that your module environment will stay the same over time. You can then verify your jobs with a new stack and then update the default module set when you are ready.

Resetting modules

After you have loaded modules over time you may want to clean up the environment a bit. You can unload individual modules but often you will simply want to reset. You could run module purge to clean out all modules but that will also remove the stack module. You would then need to reload a stack module to make program modules available again.

Using module purge is probably not what you want. If you have default module set then you can reset to that by simply running module restore

If you wish to clear out the environment even more then you could run module restore system

or

module reset

to reset to the default system modules, ignoring your default module set. If you do not have a default module set then any of the following

module restore
module restore system
module reset

will restore the module state to the system default.

Using environment modules with SGE jobs and qlogin

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 - Ulowa Wiki page for more information.

standard qsub

The recommendation for "standard" 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

```bash
module purge
```

before any "module load ..." commands.

High Throughput Computing (HTC) and/or High Volume Computing (HVC)

If your jobs are High Throughput Computing (HTC) and/or High Volume Computing (HVC) jobs, it is advised to not load modules per job but rather use the submission environment. The reason is that many thousands of jobs will need to do a large number of module loads in relatively short time frames, which could cause failures. For HTC/HVC type jobs, carefully load the modules that you will need for the jobs prior to job submission. The loaded modules are part of the environment so they will get passed to the job, unless you have overridden the default. Do not load any modules or module sets in the job script, although you may want to have comments regarding what modules are in use. Do not load any modules or module sets in your ~/.bashrc file. Finally, disable the default module set if you have one.