Intro to using the Module System

Introduction: This workshop will introduce users on how to remotely access the clusters and how to set up and personalize their environment for their computational needs. After the workshop, participants will understand:

- How to set up various environments allowing the use of a variety of programming languages, applications, libraries and utilities.

Participants will require an intro level of experience of using Linux, the ability to use a text editor from the command line, and knowledge on how to access the cluster.

Sections

1. Your Basic Environment: Demonstrate what a *default* session starts as.
2. The Module System and What's Available: Introduce the LMOD Module System and the concept of dependencies.
3. Setting Up Environments: Understand how to load modules, reset your environment by purging, and potential dependency issues.
4. Using Modules: Considerations and Defaults: Highlight the modules have defaults, and good practices such as purging and defining module versions.
5. Required Software is Not Available: Discuss approaches and process for when software isn't available and/or users bring their own scripts/code.

Note: Versions on the System might change after scheduled maintenance and before these trainings can be updated.

The context of the training and what is being demonstrated still apply.

Versions of commands are correct as of 20240727.

Your Basic Environment

Goal: Demonstrate what a *default* session starts as.

- Bash
- Environment Variables

Bash

A command-line shell and scripting language that allows users to interact with Unix-based operating systems.

```
[salexan5@mblog2 bin]$ bash --version
GNU bash, version 5.1.8(1)-release (x86_64-redhat-linux-gnu)
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

Environment Variables

```
# List ALL environments currently set within your environment.
# There's a lot!
[]$ env

# View an individual environment variable.
[]$ echo $PATH
# Something of the form:
/home/salexan5/.local/bin:/home/salexan5/bin:/apps/s/arcc/1.0/bin:/apps/s/slu
rm/latest/bin:/usr/share/Modules/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:
/usr/sbin
```

Keep an Eye on the Message of the Day

This does get updated and details office hours and important messages.

```
    __  ___          ___       _                ____
   /  |/  /__  ____/ (_)____(_)___  ___       / __ )____ _      __
  / /|_/ / _ \/ __  / / ___/ / __ \/ _ \     / __  / __ \ | /| / /
 / /  / /  __/ /_/ / / /__/ / / / /  __/    / /_/ / /_/ / |/ |/ /
/_/  /_/\___/\__,_/_/\___/_/_/ /_/\___/    /_____/\____/|__/|__/
```

```
---------------------------------- Need Help? ---------------------------------
---
Check out the ARCC Wiki for more information at
https://arccwiki.atlassian.net/
Alternatively, you can e-mail arcc-help@uwyo.edu with any questions.

UW ARCC has office hours-weekly over zoom at https://bit.ly/3BgoGfS
Office hours are 11am-1pm, every Tuesday and 12-2pm Wednesdays. If you need
help but can't make it, email us @ the address above.

See more at https://bit.ly/42NGhHN
-------------------------------------------------------------------------------
---
>>>> Important Messages <<<<

No important messages.
```

Forgot Which Projects You're In? What's My Storage Use?

At the start of each new session, the `arccquota` command will be displayed.

Are you running out of storage?

```
+-------------------------------------------------------------------------+
|               arccquota              |            Block               |
+-------------------------------------------------------------------------+
|                 Path                 | Used       Limit     %         |
+-------------------------------------------------------------------------+
| /home/salexan5                       | 08.57 GB   50.00 GB  17.14     |
| /gscratch/salexan5                   | 00.00 GB   05.00 TB  00.00     |
| /project/arcc                        | 59.63 GB   100.00 TB 00.06     |
| /project/arccanetrain                | 00.00 GB   05.00 TB  00.00     |
+-------------------------------------------------------------------------+
```

Notice: This lists *your* associated storage across the File System: Your:

- `/home/<username>/`
- `/gscratch/<username>/`
- `/projects/<project-name>/`

Help and Manual Pages

Manual pages for commands are available on the login nodes.

```
[]$ man ls
LS(1)
User Commands
LS(1)
NAME
       ls - list directory contents
SYNOPSIS
       ls [OPTION]... [FILE]...
DESCRIPTION
       List information about the FILEs (the current directory by default).
       Sort entries alphabetically if none of -cftuvSUX nor --sort is
specified.
```

Clusters are updated during maintenance periods and some aspects can change.

With this in mind, `man` pages might/might not be available on the compute nodes.

Are There Compilers and Languages Available?

Let’s check what’s on the system:

```
# GCC stands for GNU Compiler Collection, a free and open-source collection
of compilers
# and development tools that support many programming languages, hardware
architectures,
and operating systems.
[]$ which gcc
/usr/bin/gcc

[]$ gcc --version
gcc (GCC) 11.4.1 20230605 (Red Hat 11.4.1-2)

# Let's look at Python:
[]$ which python
/usr/bin/python

[]$ python --version
Python 3.9.18
```

Let's run a Python script:

```
# Create a folder in your home folder:
[]$ cd
[~]$ mkdir intro_to_modules
[~]$ cd intro_to_modules/
```

```
[intro_to_modules]$ ls
[intro_to_modules]$ pwd
/home/<username>/intro_to_modules

[intro_to_modules]$ vim python01.py
"python01.py" [New File]
# Using vim: Press ESC followed by ‘i’ to INSERT: Start typing:
import sys
print("Python version: " + sys.version)
print("Version info: " + sys.version_info)
# Using vim: ESC followed by ‘:wq’, then Return
```

Let's run a Python script: Fixed:

```
[]$ python python01.py
Python version: 3.8.16 (default, May 31 2023, 12:44:21)
[GCC 8.5.0 20210514 (Red Hat 8.5.0-18)]
Traceback (most recent call last):
  File "python01.py", line 3, in <module>
    print("Version info: " + sys.version_info)
TypeError: can only concatenate str (not "sys.version_info") to str

# Let's update the code:
From: print("Version info: " + sys.version_info)
To:   print("Version info: " + str(sys.version_info))

[]$ python python01.py
Python version: 3.9.18 (main, Jan  4 2024, 00:00:00)
[GCC 11.4.1 20230605 (Red Hat 11.4.1-2)]
Version info: sys.version_info(major=3, minor=9, micro=18,
releaselevel='final', serial=0)
```

Let’s Run an R Script

Let’s check what’s on the system:

```
[salexan5@mblog2 intro_to_modules]$ which R
/usr/bin/which: no R in
(/apps/s/arcc/1.0/bin:/apps/s/slurm/latest/bin:/home/salexan5/.local/bin:/hom
e/salexan5/bin:/usr/share/Modules/bin:/usr/local/bin:/usr/bin:/usr/local/sbin
:/usr/sbin)

# Is it case-sensitive?
[salexan5@mblog2 intro_to_modules]$ which r
/usr/bin/which: no r in
(/apps/s/arcc/1.0/bin:/apps/s/slurm/latest/bin:/home/salexan5/.local/bin:/hom
e/salexan5/bin:/usr/share/Modules/bin:/usr/local/bin:/usr/bin:/usr/local/sbin
:/usr/sbin)
```

R is not available on the system by ***default***.

But I want to use R. What can I do?

This is the reason for this module. To demonstrate how you setup your environment to make things available to *you*.

You Can't Break the System

```
[]$ ls /usr/bin
[]$ cd /usr/bin
[bin]$ ls
[]$ pwd
/usr/bin
# Permissions (ugo:rwx) and Ownership (user:group)
[bin]$ ls -al
[bin]$ ls -al python*
You can't break the system.
[bin]$ rm python
rm: cannot remove 'python': Permission denied
```

sudo: you will ***not*** be granted sudo access – do not ask!

You ***can not use*** `dnf` or other Linux related software management commands you might come across `yum`, `apt-get…`

You won't have permissions, and we will not provide you permission - do not ask!

The Module System and What's Available

Goal: Introduce the LMOD Module System and the concept of dependencies.

- What Have We Seen
- LMOD: An Environment Module System
- What do we have available?
- What's available: Using the Module System
- Change Compiler
- Purge Your Modules
- Looking for Modules: module spider
- Module load/spider: Dependencies
- What's different between these command-lines?

What Have We Seen

Although on the login nodes we do have a `gcc` compiler and the Python language this only caters for a minority of users.

We can see that R is not available, nor most of the applications that researchers will require.

There are:

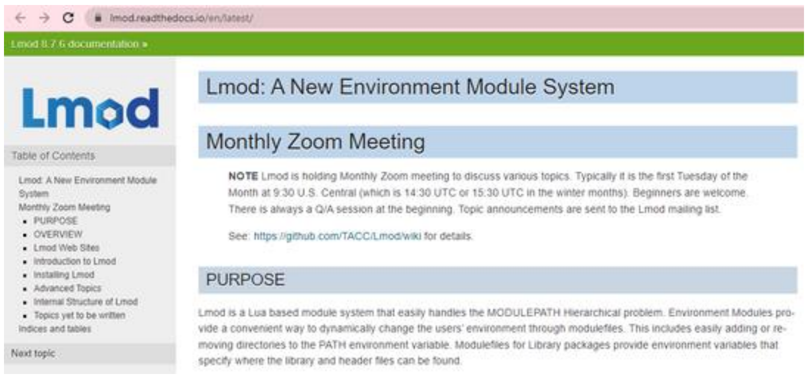
- 10s of compilers and languages
- 100s of applications and utilities.
- 1000s of underlying libraries and commands.

We can not provide a system that provides *everything all at once*.

What we need is a method to enable users to find and select that they specifically need, and a method to allow them to configure and setup their sessions/environment to allow them to perform their analysis and research.

LMOD: An Environment Module System

- ARCC: Lmod – Software



lmod.readthedocs.io/en/latest/

Lmod 8.7.6 documentation »

Table of Contents

Lmod: A New Environment Module System
Monthly Zoom Meeting
- PURPOSE
- OVERVIEW
- Lmod Web Sites
- Introduction to Lmod
- Installing Lmod
- Advanced Topics
- Internal Structure of Lmod
- Topics yet to be written

Indices and tables

Next topic

Lmod: A New Environment Module System

Monthly Zoom Meeting

NOTE Lmod is holding Monthly Zoom meeting to discuss various topics. Typically it is the first Tuesday of the Month at 9:30 U.S. Central (which is 14:30 UTC or 15:30 UTC in the winter months). Beginners are welcome. There is always a Q/A session at the beginning. Topic announcements are sent to the Lmod mailing list.

See: https://github.com/TACC/Lmod/wiki for details.

PURPOSE

Lmod is a Lua based module system that easily handles the MODULEPATH Hierarchical problem. Environment Modules provide a convenient way to dynamically change the users' environment through modulefiles. This includes easily adding or removing directories to the PATH environment variable. Modulefiles for Library packages provide environment variables that specify where the library and header files can be found.

What do we have available?

- **Compilers and eco systems**: GNU family, Intel’s oneAPI, Nvidia’s hpc-sdk
- **Languages**: C/C+, Fortran, Go, Java, Julia, Perl, Python, R, Ruby, Rust

- **Scientific libraries and toolkits**: Built with a specific compiler: GNU by default
- **Standalone applications and utilities**: Installed using:
    - Conda
    - Containers: running Singularity (not Docker)
        - We *can* create a Singularity image from a Docker image.
    - Binaries/Executables

Check the MedicineBow Software Page.

What is Available?

ARCC provides a small set of core compilers , languages, libraries and application, which will be updated on a semi regular basis. When updated, we try and use the latest versions, and typically will not be support older versions.

The GNU family of compilers (i.e. gcc) is are primary focus, but we do provide Intel's oneAPI suite of compilers/libraries as well as NVidia's hpc-sdk toolset.

Please refer to the MedicineBow Software List for what is available and/or use the `module spider` command to search.

ARCC will consider adding to this on a case-by-case basis.

What's available: Using the Module System

```
# Go back to your home folder.
[...]$ cd ~
[~]$ cd intro_to_modules/

# Modules currently loaded.
[]$ ml
Currently Loaded Modules:
  1) slurm/latest (S)   2) arcc/1.0 (S)

# What's available within my 'current' environment.
# With respect to what modules I 'currently' have loaded.
[]$ module avail
...
```

Notice: There are various categories of available software: containers, compilers, conda environments, Linux binaries…

Modules have a name and a version: `<module-name>/<module-version>`

What’s available? Compiler tree

Some software has to be built with respect to a compiler.

By *default* we build using the GNU family of compilers for C/C++ and Fortran.

```
[]$ module load gcc/13.2.0
[]$ ml
Currently Loaded Modules:
  1) slurm/latest (S)   3) gmp/6.2.1    5) mpc/1.3.1       7) zstd/1.5.5
  2) arcc/1.0     (S)   4) mpfr/4.2.0   6) zlib-ng/2.1.4   8) gcc/13.2.0
```

The `gmp`, `mpc`, `mpfr`, `zlib-ng` and `zstd` are know as dependencies.

There are libraries that need to be loaded for gcc to work.

They are loaded automatically so you don’t have to.

```
[] module avail
# What has changed?
```

We can now wee all the modules available that have been compiled with `gcc/13.2.0`

These are only available if/when the `gcc/13.2.0` module is loaded.

Change Compiler

Let’s see what happens to our environment if we load a different compiler.

```
[]# module load nvhpc-sdk/24.3
Lmod is automatically replacing "gcc/13.2.0" with "nvhpc-sdk/24.3".

# What happened?
[]$ ml
Currently Loaded Modules:
  1) slurm/latest (S)   2) arcc/1.0 (S)   3) nvhpc-sdk/24.3

[]$ module avail
# Modules relating to nvhpc-sdk/24.3 are now available.
```

You can only have ***one*** module compiler loaded at a time.

Compilers of the same name, but different versions (`gcc/12.2.0` vs `gcc/13.2.0`) are considered different.

Purge Your Modules

```
[]$ purge

# What has changed?
[]$ ml
[]$ module avail
```

Use the `purge` command to *reset* your environment to the initial *defaults*.

The more modules you have loaded, the more dependencies you will have loaded and the more likely you are to forget what you have loaded…

What's available? Modules loaded by default

```
[]$ module purge
The following modules were not unloaded:
  (Use "module --force purge" to unload all):
  1) slurm/latest   2) arcc/1.0

[]$ ml
Currently Loaded Modules:
  1) slurm/latest (S)   2) arcc/1.0 (S)
  Where:
   S:  Module is Sticky, requires --force to unload or purge
```

Looking for Modules: `module spider`

Remember: module avail only shows what is *available* to be loaded with respect to your *current* environment and what you currently have loaded. It does not show you *everything*.

```
[]$ module spider openmpi
----------------------------------------------------------------------------
  openmpi: openmpi/4.1.6
----------------------------------------------------------------------------
     Other possible modules matches:
        nvhpc-openmpi3

    You will need to load all module(s) on any one of the lines below before
the "openmpi/4.1.6" module is available to load.

      arcc/1.0  gcc/13.2.0
...
```

Module load/spider: Dependencies

```
[]$ module load openmpi/4.1.6
Lmod has detected the following error: These module(s) or extension(s) exist
but cannot be loaded as requested: "openmpi/4.1.6"
   Try: "module spider openmpi/4.1.6" to see how to load the module(s).

# What dependencies does openmpi have? And what is currently loaded?
[]$ module load gcc/13.2.0
[]$ module load openmpi/4.1.6

# What do you notice?
[]$ ml
[]$ module avail
```

Due to dependencies, there can be a hierarchy of core modules that need to be loaded to make other libraries and software available.

and what is loaded. For example, some libraries, such as `fftw`, need to be built with `gcc/13.2.0` and use the `openmpi/4.1.6` library.

Typically their version number will be prefixed with `-ompi`.

Try: `[]$ module spider fftw`

What's different between these command-lines?

```
[]$ module spider python/3.10.6
[]$ module spider python/3.12.0
[]$ module spider python2
```

What's different between these command-lines? Dependencies

```
[]$ module spider python/3.10.6
[]$ module spider python/3.12.0
...
    You will need to load all module(s) on any one of the lines below before
the "python/3.12.0" module is available to load.
      arcc/1.0  gcc/13.2.0

[]$ module spider python2
----------------------------------------------------------------------------
  python2: python2/2.7.18
----------------------------------------------------------------------------
    You will need to load all module(s) on any one of the lines below before
the "python2/2.7.18" module is available to load.
```

```
      arcc/1.0
```

Setting Up Environments

Goal: Understand how to load modules, reset your environment by purging, and potential dependency issues.

Setup Python environment

```
[]$ module purge
[]$ module load gcc/13.2.0
[]$ module load python/3.12.0
[]$ python -version

# Single line:
# Order matters:
[]$ module purge
[]$ module load python/3.12.0 gcc/13.2.0
vs
[]$ module load gcc/13.2.0 python/3.12.0
```

What’s happened to the PATH environment variable?

```
[]$ module purge
[]$ echo $PATH
[]$ module load gcc/13.2.0 python/3.12.0
[]$ echo $PATH
/apps/u/spack/gcc/13.2.0/python/3.12.0-
ovfqpv2/bin:/apps/u/spack/gcc/13.2.0/util-linux-uuid/2.38.1-x54sns7/bin:
/apps/u/spack/gcc/13.2.0/sqlite/3.43.2-
vkxcwyj/bin:/apps/u/spack/gcc/13.2.0/openssl/3.1.3-shbr5xo/bin:
/apps/u/spack/gcc/13.2.0/gettext/0.22.3-
54ib6sn/bin:/apps/u/spack/gcc/13.2.0/tar/1.34-drux4fh/bin:
/apps/u/spack/gcc/13.2.0/pigz/2.7-
6mr4kci/bin:/apps/u/spack/gcc/13.2.0/libxml2/2.10.3-5toq4pi/bin:
```

```
/apps/u/spack/gcc/13.2.0/xz/5.4.1-
uainak7/bin:/apps/u/spack/gcc/13.2.0/libiconv/1.17-3dj22ny/bin:
/apps/u/spack/gcc/13.2.0/gdbm/1.23-
idyayjv/bin:/apps/u/spack/gcc/13.2.0/readline/8.2-j6zrtuz/bin:
/apps/u/spack/gcc/13.2.0/ncurses/6.4-
y257enz/bin:/apps/u/spack/gcc/13.2.0/expat/2.5.0-3aw2urq/bin:
/apps/u/spack/gcc/13.2.0/bzip2/1.0.8-
7q5awx3/bin:/apps/u/spack/gcc/11.4.1/gcc/13.2.0-sxnvmb2/bin:
/apps/u/spack/gcc/13.2.0/zstd/1.5.5-ibvf7gj/bin:/apps/s/arcc/1.0/bin:
/apps/s/slurm/latest/bin:/home/salexan5/.local/bin:/home/salexan5/bin:/usr/sh
are/Modules/bin:
/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin

[]$ which python
/apps/u/spack/gcc/13.2.0/python/3.12.0-ovfqpv2/bin/python
```

Can we use the R language?

```
[]$ module purge

# Is R available?
[]$ module avail

[]$ module spider r
# What do we see and why?

# Does this work?
[]$ module load r/4.4.0

# How do we fix it?
```

Can we use the R language? Fixed

```
[]$ module purge
[]$ module load gcc/13.2.0 r/4.4.0

# How many dependencies have automatically been loaded?
[]$ ml

[]$ R --version
R version 4.4.0 (2024-04-24) -- "Puppy Cup"
```

Can we also use the Python language?

```
[]$ module purge
[]$ module load gcc/13.2.0 r/4.4.0

[]$ module load python/3.12.0
-----------------------------------------------------------------------------
--
```

```
The following dependent module(s) are not currently loaded: python/3.10.6
(required by: glib/2.78.0, xcb-proto/1.15.2, gobject-introspection/1.76.1)
-----------------------------------------------------------------------------
--
The following have been reloaded with a version change:
  1) python/3.10.6 => python/3.12.0

[]$  R --version
R version 4.4.0 (2024-04-24) -- "Puppy Cup"
[]$ python --version
Python 3.12.0
```

Notice: Loading R automatically loaded `python/3.10.6`.

After loading a module, consider calling `ml` to see what dependencies have also been loaded.

Loading `python/3.12.0` replaced this version.

Load Another Compiler:

```
[]$ module load gcc/13.2.0 r/4.4.0
[]$ R --version
R version 4.4.0 (2024-04-24) -- "Puppy Cup"
...
[]$ module load oneapi/2024.1.0
Lmod is automatically replacing "gcc/13.2.0" with "oneapi/2024.1.0".
Inactive Modules:
  1) cairo/1.16.0                    30) libxcb/1.14
...
 29) libxau/1.0.8                    58) xtrans/1.4.0
Due to MODULEPATH changes, the following have been reloaded:
  1) berkeley-db/18.1.40     6) libiconv/1.17     11) pigz/2.7         16)
zlib-ng/2.1.4
...
  5) libbsd/0.11.7          10) perl/5.38.0       15) xz/5.4.1

[]$ R --version
-bash: R: command not found
```

Remember:

- Only one compiler/version can be loaded into your environment at a time.
- Can only load languages/applications built with the same compiler.
- But, even this can introduce dependency issues.

Will changing the version of Python affect R? In this case *probably* not.

But if underlying versions of libraries are changing - then *maybe* - and that's the best we can say…

Remember: The more complicated your environments, the more dependencies there'll be, the more potential for ***dependency hell***.

Using Modules: Considerations and Defaults

Goal: Highlight the modules have defaults, and good practices such as purging and defining module versions.

Defaults: Why Consider?

Modules change and are updated.

The *defaults* will change and update – you might not realize.

Please *define the version* of a module you're using.

Helps to *replicate* and *triage*.

Good Practice: Module Purging

Over the course of a session, over many hours, you might be loading a number of different modules/software, you might be trying to install Python/R packages, you might be compiling source code…

We'd *suggest* when you switch context to a different application to perform a `module purge` and only load what is explicitly required.

This will avoid *dependency hell*.

Why do versions matter?

Consider Python Packages installed using `pip`:

- You install a package with *respect to the version of python* you are using.
- A packages is not automatically *available/installed* across different versions.
- If you start using `python/3.9` and then swap to `python/3.10` you will need to re-install any packages.

```
# Example:
[lib]$ pwd
/home/<username>/.local/lib
[]$ ls
python2.7  python3.9  python3.10  python3.12
# These folders will only be created if you use that version.
# Each child folder has its own site-packages folder.
```

It is the same concept for R libraries.

This is covered in more detail under the following workshops:

- Using Python, Conda and Pip on the Cluster
- Using R and RStudio on the Cluster

Required Software is Not Available

Goal: Discuss approaches and process for when software isn't available and/or users bring their own scripts/code.

- Software Installation Goals
- User Installed Software/Packages
- Consultation

Software Installation Goals

With the migration over to the MedicineBow cluster in August 2024, ARCC evolved and updated it's software policy with the following goals:

- Helping our users to become better HPC users,
- by reflect systems and processes similar to other HPC centers,
- by assisting via consultation and training to developing transferable skills and good practices
- as well as allowing easier transition and move to other HPC centers.

User Installed Software/Packages

First: Check the MedicineBow Software List for what is available and/or use the `module spider` command to search.

There are various ways to install software onto the cluster, including:

- Via Conda Environments: See the Intro to Using Conda: Creating Python/R and other Application Environments workshop.
- Create and/or import Singularity images.
- Compiling/Building software with a variety compilers and `make`/`cmake` tools. While making use of the various scientific libraries we currently have installed.
- Installing Python/R packages, and similar packages for other languages.

Users are encouraged to install their required software within the project space, under a specific `/project/<project-name>/software/` folder allowing the management and sharing of commonly used project based software. If ARCC assists any user/project with a software installation, this will be the location used unless somewhere else is explicitly requested.

Since this will use up project related storage space, the default project space has been increased to accommodate this.

Consultation

We appreciate that at first this might appear daunting, but be rest assured ARCC will continue to support users by providing assistance, consultation and trainings.

Assistance can be requested via our portal's HPC Software Consultation page.

Workshop Home	
Intro to using the Module System	

Use the following link to provide feedback on this training: https://forms.gle/2jXFJ8Bb4S5bftfv6 or use the QR code below.