Installing and Using Software on FASRC Clusters

Paula Sanematsu
Harvard - FAS Research Computing
Overview

- Introduction
  - Why move to Rocky 8?
  - Software changes
- Software modules
- Installing Python and R packages
- Using precompiled (numeric, I/O, etc) software libraries
- Singularity basics
- Installing software yourself
  - Spack
- RC VDI portal (GUI apps, such as MATLAB, RStudio Server, Jupyter Notebook, Remote Desktop, and more)
Introduction: Why move to Rocky 8?

- CentOS 7: non-commercial version of RedHat's Enterprise Linux (RHEL)
- CentOS is being discontinued by RedHat
- New development ceased at the end of 2021
  - potential security vulnerabilities
  - incompatibility with new software
- Rocky 8 Linux
  - created by original CentOS developers
  - good portion of HPC community is also adopting it => more community support

https://www.rc.fas.harvard.edu/blog/2023-downtime/
Introduction: Software changes

- Most software built on CentOS 7 will not work

- Rocky 8 software:
  - HeLmod (modules): only compilers, basic libraries, and few software packages
  - Spack for most software installs
  - Singularity
  - Julia
  - Python
  - R with Spack

- Legacy CentOS 7 support
  - Singularity container with full compute node environment and CentOS 7 modules
  - CentOS 7 will not be maintained for the compute environment
  - Slurm support for CentOS 7 will be dropped with the next major Slurm upgrade
HeLmod: software modules

Compilers: gcc, Intel
CUDA and cuDNN

MPI Libraries: OpenMPI, Mpich, Intel-MPI

Basic numerical and I/O libraries (e.g., gsl, HDF5, NetCDF)

Common software packages (e.g., Python/Mamba, R, PyCharm, vscode, etc., but customization will be left to the user through spack)

Commercial software (e.g., MATLAB, Mathematica, IDL, Stata, SAS)
Software Modules

- Software modules basics

  module load gcc/12.2.0-fasrc01  # Load Compiler
  module load openmpi/4.1.4-fasrc01 # Load MPI library
  module load netcdf-c/4.9.2-fasrc01 # Load NetCDF Library

  module list  # List loaded modules
  module purge  # Unload all modules
  module display netcdf-c/4.7.3-fasrc05  # Display module environment

  module avail  # Show ALL available modules in the MODULEPATH
  module avail netcdf-c  # Show available netcdf-c modules

- Finding modules

  module spider openmpi  # Find available OpenMPI modules
  module spider openmpi/4.1.4-fasrc01 # List more information including how to load it
Modules: How do they work? (1)

[pkrastev@holy7c24103 ~]$ module load gcc/12.2.0-fasrc01
[pkrastev@holy7c24103 ~]$ which gcc
/n/sw/helmod-rocky8/apps/Core/gcc/12.2.0-fasrc01/bin/gcc
[pkrastev@holy7c24103 ~]$ gcc --version
gcc (GCC) 12.2.0

# System Python3
[paulasan@holy7c24101 ~]$ module list
No modules loaded
[paulasan@holy7c24101 ~]$ python3 --version
Python 3.6.8
[paulasan@holy7c24101 ~]$ which python3
/usr/bin/python3

# Module Python3
[paulasan@holy7c24101 ~]$ module load python/3.10.9-fasrc01
[paulasan@holy7c24101 ~]$ python3 --version
Python 3.10.9
[paulasan@holy7c24101 ~]$ which python3
/n/sw/Mambaforge-22.11.1-4/bin/python3
[pkrastev@holy7c24103 ~]$ module display gcc/12.2.0-fasrc01
... 
/n/sw/helmod-rocky8/modulefiles/Core/gcc/12.2.0-fasrc01.lua:
...
whatis("Name: gcc")
whatis("Version: 12.2.0-fasrc01")
whatis("Description: the GNU Compiler Collection")
setenv("CC","gcc")
setenv("CXX","g++")
setenv("FC","gfortran")
setenv("F77","gfortran")
setenv("GCC_HOME","/n/sw/helmod-rocky8/apps/Core/gcc/12.2.0-fasrc01")
setenv("GCC_LIB","/n/sw/helmod-rocky8/apps/Core/gcc/12.2.0-fasrc01/lib64")
setenv("GCC_INCLUDE","/n/sw/helmod-rocky8/apps/Core/gcc/12.2.0-fasrc01/include")
prepend_path("PATH","/n/sw/helmod-rocky8/apps/Core/gcc/12.2.0-fasrc01/bin")
prepend_path("CXX","/n/sw/helmod-rocky8/apps/Core/gcc/12.2.0-fasrc01/lib64/gcc/x86_64-pc-linux-gnu/12.2.0/include")
prepend_path("CPATH","/n/sw/helmod-rocky8/apps/Core/gcc/12.2.0-fasrc01/lib64/gcc/x86_64-pc-linux-gnu/12.2.0/install-tools/include")
prepend_path("FPATH","/n/sw/helmod-rocky8/apps/Core/gcc/12.2.0-fasrc01/lib64/gcc/x86_64-pc-linux-gnu/12.2.0/plugin/include")
prepend_path("FPATH","/n/sw/helmod-rocky8/apps/Core/gcc/12.2.0-fasrc01/lib64/gcc/x86_64-pc-linux-gnu/12.2.0/include")
prepend_path("INFOPATH","/n/sw/helmod-rocky8/apps/Core/gcc/12.2.0-fasrc01/share/info")
prepend_path("LD_LIBRARY_PATH","/n/sw/helmod-rocky8/apps/Core/gcc/12.2.0-fasrc01/lib")
prepend_path("LIBRARY_PATH","/n/sw/helmod-rocky8/apps/Core/gcc/12.2.0-fasrc01/lib")
prepend_path("LD_LIBRARY_PATH","/n/sw/helmod-rocky8/apps/Core/gcc/12.2.0-fasrc01/lib64")
prepend_path("LIBRARY_PATH","/n/sw/helmod-rocky8/apps/Core/gcc/12.2.0-fasrc01/lib64")
prepend_path("MANPATH","/n/sw/helmod-rocky8/apps/Core/gcc/12.2.0-fasrc01/share/man")
prepend_path("MODULEPATH","/n/sw/helmod-rocky8/modulefiles/Comp/gcc/12.2.0-fasrc01")
setenv("FASRCSW_COMP_NAME","gcc")
setenv("FASRCSW_COMP_VERSION","12.2.0")
setenv("FASRCSW_COMP_RELEASE","fasrc01")
family("Comp")
Use of groupings is important for proper working of scientific applications.

- Applications built with specific compiler flavor/version need to be linked with libraries compiled with the same compiler flavor and version
- **Message Passing Interface (MPI)** library allows for communication between tasks on a distributed memory computers with many processors
- Parallel applications and libraries must be built with a matching MPI library and compiler

Instead of using a flat namespace, we can use module hierarchies:

- Simple technique because once users chose a compiler and MPI implementation, they can only load modules that match that compiler and MPI implementation
- FASRC follows TACC's convention:

```plaintext
# MODULEPATH_ROOT= /n/sw/helmod-rocky8/modulefiles
${MODULEPATH_ROOT}/{Core, Comp, MPI}
```
Modules: Hierarchies (2)

[pkra…]
$ module load gcc/12.2.0-fasrc01 openmpi/4.1.4-fasrc01 hdf5/1.14.0-fasrc01
[pkra…]
$ module spider hdf5/1.14.0-fasrc01

--------------------------------------------------------------------------------------------------------------------------
hdf5: hdf5/1.14.0-fasrc01
--------------------------------------------------------------------------------------------------------------------------
Description:
  HDF5 is a data model, library, and file format for storing and managing data.

You will need to load all module(s) on any one of the lines below before the "hdf5/1.14.0-fasrc01" module is available to load.

gcc/12.2.0-fasrc01  mpich/4.1-fasrc01
gcc/12.2.0-fasrc01  openmpi/4.1.4-fasrc01
intel/23.0.0-fasrc01  mpich/4.1-fasrc01
intel/23.0.0-fasrc01  openmpi/4.1.4-fasrc01

Help:
hdf5-1.14.0-fasrc01
HDF5 is a data model, library, and file format for storing and managing data.
Python Programs (1)

Mamba replacing Anaconda: fast, robust, and cross-platform package manager

Python 3
[pkrastev@holy7c24103 ~]$ module load python/3.10.9-fasrc01
[pkrastev@holy7c24103 ~]$ module list

Currently Loaded Modules:
1) Mambaforge/22.11.1-fasrc01  2) python/3.10.9-fasrc01

Python 2
[pkrastev@holy7c24103 ~]$ module load python/2.7.16-fasrc01
[pkrastev@holy7c24103 ~]$ module list

Currently Loaded Modules:
1) Anaconda2/2019.10-fasrc01  2) python/2.7.16-fasrc01
Python Programs (2)

- Mamba is a tool to manage conda environments
- Mamba uses the same commands and configuration options as conda
- You can swap almost all commands between conda & mamba
- https://github.com/fasrc/User_Codes/blob/master/Languages/Python/Mamba.md

```python
# Load a Python module
module load python/3.10.9-fasrc01
# Create local conda environment in ~/.conda/envs/ENV_NAME
mamba create -n ENV_NAME PACKAGE_LIST
# Use the new environment
mamba activate ENV_NAME
# Install a new package named MY_PACKAGE
mamba install MY_PACKAGE
# If the package is not available with conda/mamba use pip
pip install MY_PACKAGE
# If you have problems updating a package first remove it
mamba uninstall MY_PACKAGE
# Deactivate the environment
mamba deactivate
```
Python Programs (3)

For optimal performance it is recommended to use the `--prefix` option to create / relocate your conda environments to your LAB space, e.g.,

```
/n/holylabs/LABS/<PI_LAB>/Lab
```

or

```
/n/holylabs/LABS/<PI_LAB>/Users/${USER}
```

# Load a Python module, e.g.,
module load python/3.10.9-fasrc01

# Create a conda environment in LAB space, e.g.,
mamba install -y --prefix=/n/holylabs/LABS/<PI_LAB>/Lab/conda/<ENV_NAME> PACKAGE_LIST

# Activate the conda environment
mamba activate /n/holylabs/LABS/<PI_LAB>/Lab/conda/<ENV_NAME>
R Programs

- When loading R from the LMOD module system, only basic R packages are loaded in your environment.

- **Optional:** Set `R_LIBS_USER` environment variable to specify location of R package installations. Default location is `$HOME/R/x86_64-pc-linux-gnu-library/<Rversion>`.

- [https://docs.rc.fas.harvard.edu/kb/r-packages/](https://docs.rc.fas.harvard.edu/kb/r-packages/)

- For R packages that require additional software installs (`glmnet`, `sf`, `raster`, etc.): R packages with Spack

  ```
  # Load R module, e.g.,
  module load R/4.2.2-fasrc01

  # (Optional) Set R_LIBS_USER to your location for R packages, e.g.,
  export R_LIBS_USER=$HOME/apps/R:$R_LIBS_USER

  # Start R
  R

  # Inside R shell, install the desired package, e.g.,
  > install.packages("data.table")
  ```
Julia

# Use lab storage, e.g.,
[jharvard@holy7c12104 ~]$ cd /n/holylabs/LABS/jharvard_lab/Users/jharvard/software/

# Download julia and extract
[jharvard@holy7c12104 software]$ wget \ https://julialang-s3.julialang.org/bin/linux/x64/1.9/julia-1.9.1-linux-x86_64.tar.gz
[jharvard@holy7c12104 software]$ tar xvfz julia-1.9.1-linux-x86_64.tar.gz

# Add julia to path
[jharvard@holy7c12104 julia-1.9.1]$ export PATH=$PATH:/n/holylabs/LABS/jharvard_lab/Users/jharvard/software/julia-1.9.1/bin

[jharvard@holy7c12104 julia-1.9.1]$ julia

| Documentation: https://docs.julialang.org |
| Type "?" for help, "]?" for Pkg help. |
| Version 1.9.1 (2023-06-07) |
| Official https://julialang.org/ release |

julia>

https://github.com/fasrc/User_Codes/tree/master/Languages/Julia
Singularity (1) - Basics

Singularity provides a container runtime and an ecosystem for managing images that is suitable for multi-tenant systems and HPC environments.

Important aspects:

- Each application will have its own container
- Containers are not fully isolated (e.g., host network is available)
- Users have the same uid and gid when running an application
- You can build containers from:
  - from existing container in SingularityCE container library
  - from existing container in Docker Hub,
  - from SingularityCE definition file on Sylabs cloud
  - NEW: from SingularityCE definition file and proot directly on Cannon

https://github.com/fasrc/User_Codes/tree/master/Singularity_Containers
Singularity (2) – Pulling from repositories

Examples

- Preparation (start an interactive session and cd to $SCRATCH directory):
  
  ```
  [jharvard@holyl03 ~]$ salloc -p gpu_test --gres=gpu:1 --mem=4G -N 1 -t 60
  [compute-node]$ cd $SCRATCH/your_lab/your_user/
  ```

- Pulling from Docker:
  
  ```
  [jharvard@holylgpu2c0709 ~]$ singularity pull docker://tensorflow/tensorflow:latest
  ```

- Pulling from sylab / library -- https://cloud.sylabs.io/library
  
  ```
  [jharvard@holylgpu2c0709 ~]$ singularity pull library://library/default/ubuntu:21.04
  ```

- Pulling from NVIDIA’s NGC registry - https://catalog.ngc.nvidia.com
  
  ```
  [jharvard@holylgpu2c0709 ~]$ singularity pull docker://nvcr.io/nvidia/tensorflow:23.02-tf2-py3
  [jharvard@holylgpu2c0709 ~]$ singularity exec tensorflow_23.02-tf2-py3.sif python
  ```

  Python 3.8.10 (default, Nov 14 2022, 12:59:47)

  >>> import tensorflow as tf
  >>> print(tf.__version__)
  2.11.0
Singularity (3) – TensorFlow example

# --- Start an interactive session on a partition with GPUs, e.g.,
[jharvard@holylogin03 ~]$ salloc \-p gpu_test \--gres=gpu:1 \--mem=4G \-N 1 \-t 60

# --- cd to your SCRATCH folder (e.g., jharvard) ---
[jharvard@holygpu2c0709 ~]$ cd $SCRATCH/jharvard_lab/Users/jharvard

# --- Pull the latest TF GPU version from the Docker registry ---
[jharvard@holygpu2c0709 ~]$ singularity pull \--name tf2.13_gpu.sif docker://tensorflow/tensorflow:2.13.0-gpu

# --- Get examples from keras.io ---
[jharvard@holygpu2c0709 ~]$ git clone https://github.com/keras-team/keras-io.git

# --- Execute the code ---
[jharvard@holygpu2c0709 ~]$ singularity exec \--nv tf2.13_gpu.sif python ./keras-io/examples/vision/mnist_convnet.py

... (omitted output)
Test loss: 0.024948162958025932
Test accuracy: 0.9915000200271606
Singularity (5) – Building images on Cannon

Building Singularity images from *SingularityCE* definition file and *proot* directly on Cannon

```bash
# Make ~/bin directory
[jharvard@holy2c02302 ~]$ mkdir -p ~/bin

# Change to ~/bin directory, download proot, and change permissions to make it executable
[jharvard@holy2c02302 ~]$ cd ~/bin
[jharvard@holy2c02302 bin]$ curl -LO https://proot.gitlab.io/proot/bin/proot
[jharvard@holy2c02302 bin]$ chmod +x ./proot

# Obtain, or create a singularity definition (def) file and build the image, e.g.,
[jharvard@holy2c02302 ~]$ singularity build tf-2.12.sif tf-2.12.def
INFO: Using proot to build unprivileged. Not all builds are supported. If build fails, use --remote or --fakeroot.
INFO: Starting build...
(omitted output)
INFO: Creating SIF file...
INFO: Build complete: tf-2.12.sif
```

Singularity (6) – Building images on Cannon

Example *SingularityCE* def file: **tf-2.12.def**

```
Bootstrap: docker
From: tensorflow/tensorflow:2.12.0-gpu

%post
    pip install --upgrade pip
    pip install matplotlib
    pip install seaborn
    pip install scipy
    pip install scikit-learn
    pip install jupyterlab
    pip install notebook
```

Installing Software: Spack

One-time setup

Clone Spack repo in your lab storage (better performance than $HOME directory)

Source spack

Install packages with Spack - some software can take a few hours to build

Job submission

Source spack

Load packages/software with Spack

Run code

Spacc (1)

One-time setup

# Request interactive job
[jharvard@rockylogin ~]$ salloc -p test --mem 12g -t 0-04:00 -c 8

# Use lab storage
[jharvard@holy7c12104 ~]$ cd /n/holylabs/LABS/jharvard_lab/Lab/software/

# Clone spacc and check out specific release
[jharvard@holy7c12104 software]$ git clone -c feature.manyFiles=true https://github.com/spack/spack.git
[jharvard@holy7c12104 software]$ cd spack/
[jharvard@holy7c12104 spack]$ git checkout releases/v0.19

# Source spacc
[jharvard@holy7c12104 spack]$ . share/spack/setup-env.sh

# Install packages
[jharvard@holy7c12104 spack]$ spack install bzip2
[jharvard@holy7c12104 spack]$ spack install bzip2@1.0.8
[jharvard@holy7c12104 spack]$ spack install zlib@1.2.13%gcc@8.5.0
Spack (2)

One-time setup

# List installed packages
$ spack find

# Uninstall packages, e.g.,
$ spack uninstall zlib@1.2.13%gcc@8.5.0

# Load spack packages
$ spack load bzip2
$ which bzip2
/home/spack/opt/spack/linux-rocky8-icelake/gcc-8.5.0/bzip2-1.0.8-aohgpu7zn62kzpanpohuevbkufypbnff/bin/bzip2

# List the loaded packages
$ spack find --loaded
-- linux-rocky8-icelake / gcc@8.5.0 -------------------------------
bzip2@1.0.8  diffutils@3.8  libiconv@1.16
=> 3 loaded packages

# Unload spack packages
$ spack unload
$ spack find --loaded
=> 0 loaded packages
Group Permissions

By default Spack will match your usual file permissions which typically are set up without group write permission. For lab wide installs of Spack though you will want to ensure that it has group write enforced. You can set this by going to the etc/spack directory in your Spack installation and adding a file called packages.yaml (or editing the exiting one) with the following contents:

packages:
  all:
    permissions:
      write: group
      group: jharvard_lab

Default Architecture

By default Spack will autodetect which architecture your underlying hardware is and build software to match that. However in cases where you are running on heterogeneous hardware it is best to use a more generic flag. You can set this by going to the `etc/spack` directory in your Spack installation and adding a file called `packages.yaml` (or editing the exiting one) with the following contents:

```yaml
packages:
  all:
    target: [x86_64]
```

## Compiler Configuration

# List available compilers

$ spack compilers

===> Available compilers
-- gcc rocky8-x86_64 --------------------------------------------
gcc@8.5.0

# Load the required compiler software module, e.g.,

$ module load gcc/12.2.0-fasrc01

# Add this GCC compiler version to the spack compilers

$ spack compiler find

===> Added 1 new compiler to ~/.spack/linux/compilers.yaml
   gcc@12.2.0

===> Compilers are defined in the following files:
   ~/.spack/linux/compilers.yaml

$ spack compilers

===> Available compilers
-- gcc rocky8-x86_64 --------------------------------------------
gcc@12.2.0   gcc@8.5.0

Spack (6)

Compiler Configuration

Modify ~/.spack/linux/compilers.yaml to read:

- compiler:
  spec: gcc@12.2.0
  paths:
    cc: /n/sw/helmod-rocky8/apps/Core/gcc/12.2.0-fasrc01/bin/gcc
    cxx: /n/sw/helmod-rocky8/apps/Core/gcc/12.2.0-fasrc01/bin/g++
    f77: /n/sw/helmod-rocky8/apps/Core/gcc/12.2.0-fasrc01/bin/gfortran
    fc: /n/sw/helmod-rocky8/apps/Core/gcc/12.2.0-fasrc01/bin/gfortran
  flags: {}
  operating_system: rocky8
  target: x86_64
  modules: [gcc/12.2.0-fasrc01]
  environment: {}
  extra_rpaths: []

Spack (7)

MPI Configuration

# Determine the MPI location / prefix
$ module load gcc/12.2.0-fasrc01 openmpi/4.1.4-fasrc01
$ echo $MPI_HOME
/n/sw/helmod-rocky8/apps/Comp/gcc/12.2.0-fasrc01/openmpi/4.1.4-fasrc01

# Edit manually the packages configuration file ~/.spack/packages.yaml
# Include the following content:
packages:
  openmpi:
    externals:
    - spec: openmpi@4.1.4%gcc@12.2.0
      prefix: /n/sw/helmod-rocky8/apps/Comp/gcc/12.2.0-fasrc01/openmpi/4.1.4-fasrc01
    buildable: False

# Example: Build HDF5 version 1.12.2 with gcc@12.2.0 and openmpi@4.1.4
$ module purge
$ spack install hdf5@1.12.2 % gcc@12.2.0 ^ openmpi@4.1.4

# request interactive job
[jharvard@rockylogin ~]$ salloc -p test --mem 16g -t 0:04:00 -c 8

# use lab storage
[jharvard@holy7c12104 ~]$ cd /n/holylabs/LABS/jharvard_lab/Lab/software/spack

# source spack
[jharvard@holy7c12104 spack]$ . share/spack/setup-env.sh

# install R packages with spack
[jharvard@holy2c02302 spack]$ spack install r-rgdal

# load spack packages
[jharvard@holy2c02302 spack]$ spack load r-rgdal

# launch R and load libraries
[jharvard@holy2c02302 spack]$ R
R> library(rgdal)

https://github.com/fasrc/User_Codes/blob/master/Languages/R/R_packages_with_spacc.md
SLURM jobs with Spack

#!/bin/bash
#SBATCH -J r_spack  # Job name
#SBATCH -c 1       # Number of cores (--cpus-per-task)
#SBATCH -t 0-00:10  # Runtime in D-HH:MM, minimum of 10 minutes
#SBATCH -p test    # Partition to submit to
#SBATCH --mem=4g   # Memory for all cores in GB (see also --mem-per-cpu)
#SBATCH -o myoutput_%j.out  # File to which STDOUT will be written, %j inserts jobid
#SBATCH -e myerrors_%j.err  # File to which STDERR will be written, %j inserts jobid

# source spack
. /n/holylabs/LABS/jharvard_lab/Users/jharvard/spack/share/spack/setup-env.sh

# load spack packages
spack load r-codetools
spack load r-rgdal
spack load r-raster
spack load r-terra

# run R code
Rscript --vanilla r_spack_load_libs.R > r_spack_load_libs.Rout
Legacy CentOS 7 Support

**Last resort option**

Singularity image with CentOS 7
- same environment of compute nodes
- can load CentOS 7 modules
- cannot submit jobs inside the container
- can modify CentOS 7 image by building a new container based on CentOS 7 image

```
$ singularity exec /n/singularity_images/FAS/centos7/compute-el7-noslurm-2023-03-29.sif /bin/bash
Singularity> module load gcc
Singularity> module load matlab
Singularity> module list
```

Currently Loaded Modules:
1) gmp/6.2.1-fasrc01 2) mpfr/4.1.0-fasrc01 3) mpc/1.2.1-fasrc01 4) gcc/12.1.0-fasrc01
5) matlab/R2022b-fasrc01
RC Open OnDemand (former VDI) Portal

- Interactive computing portal
  [https://rcood.rc.fas.harvard.edu/](https://rcood.rc.fas.harvard.edu/)
- Need to be on RC VPN
- Provides GUI apps, such as
  - Remote Desktop – see "How to open software" from Remote Desktop
  - Rstudio Server
  - Jupyter Notebook
  - MATLAB
  - SAS
  - Stata
  - and more
Request Help - Resources

- [https://docs.rc.fas.harvard.edu/kb/support/](https://docs.rc.fas.harvard.edu/kb/support/)
  - Rocky 8 Transition Guide
    - [https://docs.rc.fas.harvard.edu/kb/rocky-8-transition-guide/](https://docs.rc.fas.harvard.edu/kb/rocky-8-transition-guide/)
  - Portal
    - [http://portal.rc.fas.harvard.edu/rcrt/submit_ticket](http://portal.rc.fas.harvard.edu/rcrt/submit_ticket)
  - Email
    - rchelp@rc.fas.harvard.edu
  - Office Hours
    - Wednesday noon-3pm [https://harvard.zoom.us/j/255102481](https://harvard.zoom.us/j/255102481)
  - Consulting Calendar
    - [https://www.rc.fas.harvard.edu/consulting-calendar/](https://www.rc.fas.harvard.edu/consulting-calendar/)
  - Training Calendar
    - [https://www.rc.fas.harvard.edu/upcoming-training/](https://www.rc.fas.harvard.edu/upcoming-training/)
Thank you! Questions? Comments?

Plamen Krastev, PhD
Harvard - FAS Research Computing
LMOD Module System

**LMOD**: ENVIRONMENTAL MODULES SYSTEM

https://www.tacc.utexas.edu/research-development/tacc-projects/lmod

- Environment Modules provide a convenient way to dynamically modify the user’s environment through module files (Lua-based scripting files)
- Software module files define various environment variables, such as `PATH`, `LD_LIBRARY_PATH`, `LIBRARY_PATH`, `CPATH`, `FPATH`, etc., so that executables, header files and required libraries can be found by the specific software application
- Software can be “loaded” and “unloaded” dynamically
- Hierarchies (incremental module loading/unloading) prevent software conflicts
- Various software versions can coexist
Using Software Libraries (1)

- Software libraries allow you to use precompiled functions and routines in your applications.

- Many are already installed on the cluster, e.g., GSL, NetCDF, HDF5, FFTW, MKL, etc., and are available as software modules.

- Software libraries could also be a part of the OS and are typically located in /lib and /lib64.

- Linking to specific libraries can be done by setting `-l` and `-L` flags, e.g.,

  ```
  # Load required software modules, e.g.,
  module load gsl/2.7-fasrc01
  
  # Compile and link the application, e.g.,
  gcc -o gsl_int_test.x gsl_int_test.c -O2 -lm -lgsl -lgslcblas
  ```

Using Software Libraries (2)

```bash
# Make file for gsl_int_test.c
CFLAGS = -c -O2
COMPILER = gcc
PRO = gsl_int_test
OBJECTS = gsl_int_test.o

LINK_GSL = -lm -lgsl -lgslcblas

${PRO}.x : $(OBJECTS)
   $(COMPILER) -o ${PRO}.x $(OBJECTS) $(LINK_GSL)

%.o : %.c
   $(COMPILER) $(CFLAGS) $(<F)

clean :
   rm -rf *.o *.x *.mod
```

https://github.com/fasrc/User_Codes/tree/master/Libraries/GSL
Running TensorFlow on a CPU node:

```
# --- Start an interactive session ---
[login-node] $ salloc -p test --mem=4G -N 1 -t 60
# --- cd to your SCRATCH folder ---
[compute-node] $ cd $SCRATCH/your_lab/your_user/
# --- Pull the latest TF version from the Docker registry ---
[compute-node] $ singularity pull --name tf2.12_cpu.simg docker://tensorflow/tensorflow:2.12.0
# --- Launch Python and print the TF version ---
[compute-node] $ singularity exec tf2.12_cpu.simg python
... (omitted output)

>>> import tensorflow as tf
>>> print(tf.__version__)
2.12.0
# --- Get examples from keras.io ---
[compute-node] $ git clone https://github.com/keras-team/keras-io.git
# --- Execute the code ---
[compute-node] $ singularity exec tf2.12_cpu.simg python ./keras-io/examples/vision/mnist_convnet.py
... (omitted output)
Test loss: 0.026334384456276894
Test accuracy: 0.9904999732971191
```