Installing and Using Software on FASRC Cluster

Harvard - FAS Research Computing

Objectives

- To advise you on the best practices for using software applications on the FASRC cluster
- To provide the basic knowledge required for installing your software applications on the FASRC cluster

Software overview

- Software modules
- Using precompiled (numeric, I/O, etc) software libraries
- Installing `Python` and `R` packages
- Containers basics (`Singularity` and `Podman`)
- Installing software with `Spack`

Survey

- Menti https://www.menti.com/
- Code 9667 5642

LMOD Module System

LMOD: Environmental Modules Systems

- Convenient way to dynamically modify the user's environment
- Define environment variables
  - `PATH`, `LD_LIBRARY_PATH`, `LIBRARY_PATH`, `CPATH`, `FPATH`
- Software can be "loaded" and "unloaded" dynamically
- Hierarchies (incremental module loading/unloading) prevent software conflicts
- Various software versions can coexist
- Harvard modified LMOD software modules system: `HeLMOD` (Harvard Extensions for Lmod deployment) - https://github.com/fasrc/helmod

HeLmod: software modules

- Documentation: https://docs.rc.fas.harvard.edu/kb/modules-intro/
- 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`, `R`)
- Commercial software (e.g., `MATLAB`, `Mathematica`, `IDL`, `Stata`, `SAS`)

Software Modules

- Software modules basics

```
module load gcc/14.2.0-fasrc01         # Load Compiler
module load openmpi/5.0.5-fasrc01      # Load MPI library
module load netcdf-c/4.9.2-fasrc06     # Load NetCDF Library

module list                            # List loaded modules
module purge                           # Unload all modules
module display netcdf-c/4.9.2-fasrc06 # 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/5.0.5-fasrc01    # List more information including how to load it
```

Modules: How do they work? (1)

★ ssh to the cluster and try these commands

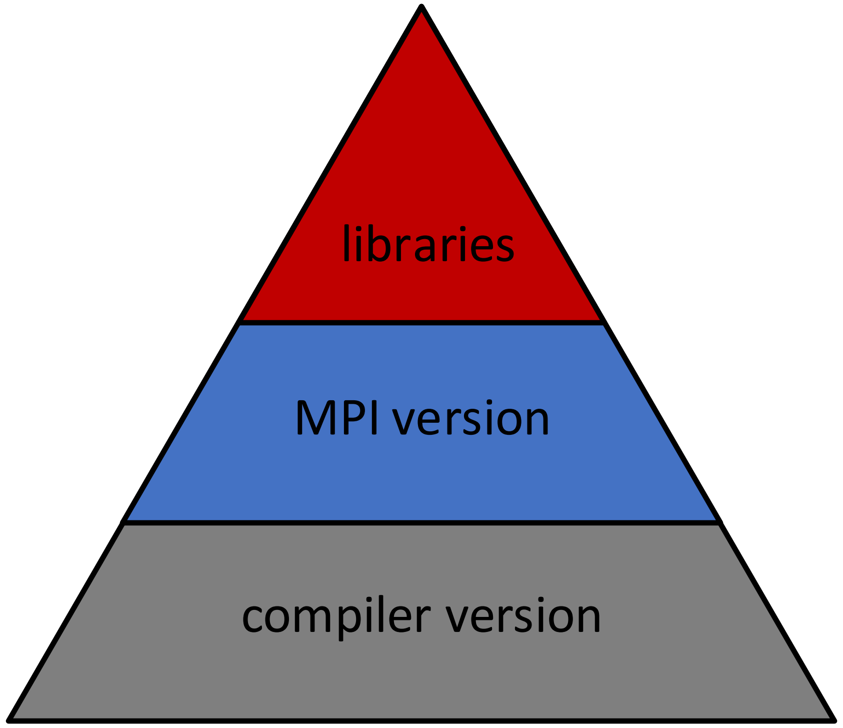
```
[jharvard@boslogin08 ~]$ module list
[jharvard@boslogin08 ~]$ which gcc
[jharvard@boslogin08 ~]$ gcc --version

[jharvard@boslogin08 ~]$ module load gcc/14.2.0-fasrc01
[jharvard@boslogin08 ~]$ module list
[jharvard@boslogin08 ~]$ which gcc
[jharvard@boslogin08 ~]$ gcc --version

[jharvard@boslogin08 ~]$ module purge
[jharvard@boslogin08 ~]$ gcc --version
```

Modules: How do they work? (2)

```
[pkrastev@holy7c24103 ~]$ module display gcc/14.2.0-fasrc01
...
   /n/sw/helmod-rocky8/modulefiles/Core/gcc/14.2.0-fasrc01.lua:
...
whatis("Name: gcc")
whatis("Version: 14.2.0-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/14.2.0-fasrc01")
setenv("GCC_LIB","/n/sw/helmod-rocky8/apps/Core/gcc/14.2.0-fasrc01/lib64")
setenv("GCC_INCLUDE","/n/sw/helmod-rocky8/apps/Core/gcc/14.2.0-fasrc01/include")
prepend_path("PATH","/n/sw/helmod-rocky8/apps/Core/gcc/14.2.0-fasrc01/bin")
prepend_path("CPATH","/n/sw/helmod-rocky8/apps/Core/gcc/14.2.0-fasrc01/lib64/gcc/x86_64-pc-linux-gnu/14.2.0/include")
prepend_path("CPATH","/n/sw/helmod-rocky8/apps/Core/gcc/14.2.0-fasrc01/lib64/gcc/x86_64-pc-linux-gnu/14.2.0/install-tools/include")
prepend_path("CPATH","/n/sw/helmod-rocky8/apps/Core/gcc/14.2.0-fasrc01/lib64/gcc/x86_64-pc-linux-gnu/14.2.0/plugin/include")
prepend_path("CPATH","/n/sw/helmod-rocky8/apps/Core/gcc/14.2.0-fasrc01/include")
prepend_path("FPATH","/n/sw/helmod-rocky8/apps/Core/gcc/14.2.0-fasrc01/lib64/gcc/x86_64-pc-linux-gnu/14.2.0/include")
prepend_path("FPATH","/n/sw/helmod-rocky8/apps/Core/gcc/14.2.0-fasrc01/lib64/gcc/x86_64-pc-linux-gnu/14.2.0/install-tools/include")
prepend_path("FPATH","/n/sw/helmod-rocky8/apps/Core/gcc/14.2.0-fasrc01/lib64/gcc/x86_64-pc-linux-gnu/14.2.0/plugin/include")
prepend_path("FPATH","/n/sw/helmod-rocky8/apps/Core/gcc/14.2.0-fasrc01/include")
prepend_path("INFOPATH","/n/sw/helmod-rocky8/apps/Core/gcc/14.2.0-fasrc01/share/info")
prepend_path("LD_LIBRARY_PATH","/n/sw/helmod-rocky8/apps/Core/gcc/14.2.0-fasrc01/lib")
prepend_path("LIBRARY_PATH","/n/sw/helmod-rocky8/apps/Core/gcc/14.2.0-fasrc01/lib")
prepend_path("LD_LIBRARY_PATH","/n/sw/helmod-rocky8/apps/Core/gcc/14.2.0-fasrc01/lib64")
prepend_path("LIBRARY_PATH","/n/sw/helmod-rocky8/apps/Core/gcc/14.2.0-fasrc01/lib64")
prepend_path("MANPATH","/n/sw/helmod-rocky8/apps/Core/gcc/14.2.0-fasrc01/share/man")
prepend_path("MODULEPATH","/n/sw/helmod-rocky8/modulefiles/Comp/gcc/14.2.0-fasrc01")
setenv("FASRCSW_COMP_NAME","gcc")
setenv("FASRCSW_COMP_VERSION","14.2.0")
setenv("FASRCSW_COMP_RELEASE","fasrc01")
family("Comp")
```

Modules: Hierarchies (1)

- Applications built with specific compiler version need to be linked with libraries compiled with the same compiler version
  - `gcc, intel`
- **Message Passing Interface (MPI)** library allows between nodes
  - `OpenMPI, Mpich, Intel-MPI`
- Parallel applications and libraries must be built with a matching MPI library and compiler
  - `gsl, HDF5, NetCDF`



Modules: Hierarchies (2)

try these commands

```
[jharvard@boslogin08 ~]$ module purge
[jharvard@boslogin08 ~]$ module list
[jharvard@boslogin08 ~]$ module avail

[jharvard@boslogin08 ~]$ module load gcc/14.2.0-fasrc01
[jharvard@boslogin08 ~]$ module list
[jharvard@boslogin08 ~]$ module avail

[jharvard@boslogin08 ~]$ module load openmpi/5.0.5-fasrc01
[jharvard@boslogin08 ~]$ module avail
```

How to find out about dependencies of parallel library:

```
[jharvard@boslogin08 ~]$ module spider hdf5
[jharvard@boslogin08 ~]$ module spider hdf5/1.14.4-fasrc01
```

Python modules

- Miniforge+mamba replacing Anaconda+conda: fast, robust, and cross-platform package manager
  - 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`
- Python 3

```
[pkrastev@holy7c24103 ~]$ module load python/3.12.5-fasrc01
[pkrastev@holy7c24103 ~]$ module list

Currently Loaded Modules:
  1) Miniforge3/24.7.1-fasrc01   2) python/3.12.5-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 package install

Documentation
https://docs.rc.fas.harvard.edu/kb/python-package-installation

```
# Load a Python module
module load python/3.12.5-fasrc01

# Create 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 – conda env in Lab storage

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

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

```
# Load a Python module, e.g.,
module load python/3.12.5-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 Packages

- Documentation: https://docs.rc.fas.harvard.edu/kb/r-and-rstudio/
- ★ First: install packages on RStudio Server on Open OnDemand
  - Cannon: https://rcood.rc.fas.harvard.edu/
  - FASSE: https://fasseood.rc.fas.harvard.edu/
- Run jobs
  - Interactively on RStudio Server
  - Via batch job

R batch jobs

```
#!/bin/bash
#SBATCH -c 1               # Number of cores (-c)
#SBATCH -t 0-01:00         # Runtime in D-HH:MM
#SBATCH -p test            # Partition to submit to
#SBATCH --mem=1G           # Memory pool for all cores (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

# set R packages and rstudio server singularity image locations
my_packages=${HOME}/R/ifxrstudio/RELEASE_3_18
rstudio_singularity_image="/n/singularity_images/informatics/ifxrstudio/ifxrstudio:RELEASE_3_18.sif"

# run myscript.R using RStudio Server signularity image
singularity exec --cleanenv --env R_LIBS_USER=${my_packages} ${rstudio_singularity_image} Rscript myscript.R
```

Submit the job

```
sbatch runscript.sh
```

Documentation https://docs.rc.fas.harvard.edu/kb/r-and-rstudio/#Use_R_packages_from_RStudio_Server_in_a_batch_job

Containers

- Documentation: https://docs.rc.fas.harvard.edu/kb/containers/
- Two main types

OCI (Open Container Initiative)

- industry standard
- one deamon/priviledged process per host → security issue in HPC (shared) systems
- solution: rootless

SIF (Singularity Image Format)

- created specific for HPC
- single read-only file

Rootless

- Documentation: https://docs.rc.fas.harvard.edu/kb/containers/
- Allows the unprivileged (non-root) user to spoof being root inside the container

Inside the Container [username (uid)]	Outside of the container [username(uid)]
root(0)	userA(20000)
apache(48)	1020048
ubuntu(1000)	1021000

you pretend to be another user and have all the privileges of root

you are acting as your user, and the uid's subordinated to your user

- Caution!! Lustre filesystems (e.g., /n/holylabs, /n/holylfs) does not recognize subuid's

Shell

- Documentation: https://docs.rc.fas.harvard.edu/kb/containers/#Shell
- How to a shell prompt inside the container

Singularity

```
[jharvard@holy8a26602 ~]$ singularity shell docker://godlovedc/lolcow
Singularity>
```

Podman

```
[jharvard@holy8a26601 ~]$ podman run --rm -it --entrypoint bash docker://godlovedc/lolcow
root@holy8a26601:/#
```

GPU

- Documentation: https://docs.rc.fas.harvard.edu/kb/containers/#GPU
- How to use a gpu in a container

Singularity: add flag `--nv`

```
[jharvard@holy8a26602 ~]$ singularity shell --nv docker://godlovedc/lolcow
Singularity>
```

Podman: add flag `--device nvidia.com/gpu=all`

```
[jharvard@holy8a26601 ~]$ podman run --rm --device nvidia.com/gpu=all -it --entrypoint bash
docker://godlovedc/lolcow
root@holy8a26601:/#
```

Containers advanced usage

- Singularity: https://docs.rc.fas.harvard.edu/kb/singularity-on-the-cluster/
  - commands (shell, run, exec)
  - batch jobs with containers
  - accessing files via `--bind`
  - building containers
- Podman: https://docs.rc.fas.harvard.edu/kb/podman/
  - commands
  - accessing files via `--volume`
  - environmental variables
  - building containers
  - pushing containers to registry

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

https://github.com/fasrc/User_Codes/blob/master/Documents/Software/Spack.md

Spack package installation

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 spack
[jharvard@holy7c12104 software]$ git clone -c feature.manyFiles=true https://github.com/spack/spack.git

# Source spack (sets environmental variables)
[jharvard@holy7c12104 software]$ cd spack/
[jharvard@holy7c12104 spack]$ . share/spack/setup-env.sh

# Install packages
[jharvard@holy7c12104 spack]$ spack install bzip2                  # Install default/latest version
[jharvard@holy7c12104 spack]$ spack install bzip2@1.0.8            # Specify version
[jharvard@holy7c12104 spack]$ spack install zlib@1.2.13%gcc@8.5.0  # Specify version and compiler
```

Spack common commands

```
# 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-x86_64/gcc-8.5.0/bzip2-1.0.8-qimzsx6zy45aww52i3uowomnsho5muep/bin/bzip2

# List the loaded packages
$ spack find --loaded
-- linux-rocky8-x86_64 / gcc@8.5.0 ------------------------------
bzip2@1.0.8
==> 1 loaded package

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

Sharing your Spack packages with your lab

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
```

https://spack.readthedocs.io/en/latest/build_settings.html#package-permissions

Spack architecture

- Documentation
- Spack autodetects architecture
- Spack builds software to match architecture
- Cannon has heterogeneous architecture
- Solution: set generic architecture
  - In the Spack folder, go to `etc/spack`
  - Open or create file `packages.yaml`
  - Add to `packages.yaml`:

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

Spack 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/14.2.0-fasrc01

# Add this GCC compiler version to the spack compilers
$ spack compiler find
==> Added 1 new compiler to ~/.spack/linux/compilers.yaml
    gcc@14.2.0
==> Compilers are defined in the following files:
    ~/.spack/linux/compilers.yaml

$ spack compilers
==> Available compilers
-- gcc rocky8-x86_64 -------------------------------------------
gcc@8.5.0  gcc@14.2.0
```

https://github.com/fasrc/User_Codes/blob/master/Documents/Software/Spack.md#compiler-configuration

Spack compiler configuration

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

```
- compiler:
    spec: gcc@=14.2.0
    paths:
      cc: /n/sw/helmod-rocky8/apps/Core/gcc/14.2.0-fasrc01/bin/gcc
      cxx: /n/sw/helmod-rocky8/apps/Core/gcc/14.2.0-fasrc01/bin/g++
      f77: /n/sw/helmod-rocky8/apps/Core/gcc/14.2.0-fasrc01/bin/gfortran
      fc: /n/sw/helmod-rocky8/apps/Core/gcc/14.2.0-fasrc01/bin/gfortran
    flags: {}
    operating_system: rocky8
    target: x86_64
    modules: [gcc/14.2.0-fasrc01]
    environment: {}
    extra_rpaths: []
```

https://github.com/fasrc/User_Codes/blob/master/Documents/Software/Spack.md#compiler-configuration

Spack MPI configuration

```
# Determine the MPI location / prefix
$ module load gcc/14.2.0-fasrc01 openmpi/5.0.5-fasrc01
$ echo $MPI_HOME
/n/sw/helmod-rocky8/apps/Comp/gcc/14.2.0-fasrc01/openmpi/5.0.5-fasrc01

# Edit manually the packages configuration file ~/.spack/packages.yaml
# Include the following content:
packages:
  openmpi:
    externals:
    - spec: openmpi@5.0.5%gcc@14.2.0
      prefix: /n/sw/helmod-rocky8/apps/Comp/gcc/14.2.0-fasrc01/openmpi/5.0.5-fasrc01
    buildable: False

# Example: Build HDF5 version 1.14.0 with gcc@14.2.0 and openmpi@5.0.5
$ module purge
$ spack install hdf5@1.14.0 % gcc@14.2.0 ^ openmpi@5.0.5
```

https://github.com/fasrc/User_Codes/blob/master/Documents/Software/Spack.md#mpi-configuration

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
```

Open OnDemand

- Interactive computing portal https://rcood.rc.fas.harvard.edu/
- Need to be on RC VPN
- Provides GUI apps, such as
  - Remote Desktop
  - Rstudio Server
  - Jupyter Notebook
  - MATLAB
  - SAS
  - Stata
- Check our training calendar for OOD training sessions!

Training calendar

- https://www.rc.fas.harvard.edu/upcoming-training/

Request Help - Resources

- Support https://docs.rc.fas.harvard.edu/kb/support/
  - Portal
    - 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/97676134704
  - Training Materials
    - https://docs.rc.fas.harvard.edu/kb/training-materials/
  - Training Calendar
    - https://www.rc.fas.harvard.edu/upcoming-training/

Training session evaluation

Please, fill out our training session evaluation. Your feedback is essential for us to improve our trainings!!

https://tinyurl.com/FASRC-training

Thank you! Questions? Comments?

Plamen Krastev, PhD

Harvard - FAS Research Computing

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/x64/1.10/julia-1.10.5-linux-x86_64.tar.gz
[jharvard@holy7c12104 software]$ tar xvfz julia-1.10.5-linux-x86_64.tar.gz

# Add julia to path
[jharvard@holy7c12104 julia-1.10.5]$ export PATH=$PATH:/n/holylabs/LABS/jharvard_lab/Users/jharvard/software/julia-1.10.5/bin

[jharvard@holy7c12104 julia-1.10.5]$ julia
               _
   _       _ _(_)_     |  Documentation: https://docs.julialang.org
  (_)     | (_) (_)    |
   _ _   _| |_  __ _   |  Type "?" for help, "]?" for Pkg help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 1.10.5 (2024-08-27)
 _/ |\__'_|_|_|\__'_|  |  Official https://julialang.org/ release
|__/                   |


julia>
```

https://docs.rc.fas.harvard.edu/kb/julia

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`, `BLAS`, `LAPACK`, `NetCDF`, `HDF5`, `FFTW`, `MKL`, `BOOST`, 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/2.8-fasrc01

# Compile and link the application, e.g.,
gcc -o gsl_int_test.x gsl_int_test.c -O2 -lm -lgsl -lgslcblas
```

https://github.com/fasrc/User_Codes/tree/master/Libraries/GSL

Using Software Libraries (2)

```
#================================================================
# 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