Software Management
Outline

• CARC managed software
  • Finding software
  • Using modules
  • Neat features
• Installing software
  • Precompiled binary
  • Python
  • R
  • Singularity
  • Compiling from source
  • Finding dependencies
Software is complex!

- Dependency graph for ARES

CARC managed software

• Software is built using **Spack** and saved to /spack/apps/linux-centos-x86_64

• Instead of manually managing environment, use **Lmod** software modules
What are software modules?

• Modules present installed software to users
• Set environment variables
  • PATH
  • PKG_CONFIG_PATH
  • LD_LIBRARY_PATH
  • <SOFTWARE>_ROOT
• Show how package was built
• Show where package was installed to
• Prevent loading incompatible software
How to use modules

• By default you have the recommended "usc" module loaded

$ module list
Currently Loaded Modules:
  1) gcc/8.3.0    3) openmpi/4.0.2
  2) openblas/0.3.8  4) pmix/3.1.3
  5) usc

• You can check what's available with module avail

• Depending on active modules, you will see different results from module avail

• There are 4 kinds of modules
  • Compiler (gcc, intel)
  • BLAS (Openblas, AMD-blis, netlib)
  • MPI (OpenMPI, mvapich2, IntelMPI)
  • Application (most common, bamtools, cmake, libpng...)
Example module avail listing

```
$ module avail

/spack/apps/lmod/linux-centos7-x86_64/openmpi/4.0.2-ipm3dnv/openblas/0.3.8-2no6mfz/gcc/8.3.0

Applications built with gcc 8.3.0 compiler AND openmpi 4.0.2 AND openblas 0.3.8
```

```
hypre/2.18.2-openblas-openmpi
```

```
hdf5/1.10.6-openmpi
```

```
hmmr/3.3-openmpi
```

```
scotch/6.0.8-openmpi
```

```
tau/2.29-openmpi
```

```
hmmer/3.3-openmpi
```

```
adapterremoval/2.3.1
```

```
ananaconda3/2019.10
```

```
argtable/2
```

```
kraken/1.0
```

```
perl-data-dumper/2.173
```

```
argtable/2
```

```
perl-extutils-config/0.008
```

```
lcms/2.9
```

```
at-spi2-atk/2.26.2
```

```
Applications built with gcc 8.3.0 compiler AND openblas 0.3.8
```

```
Applications built with gcc 8.3.0 compiler AND openmpi 4.0.2
```

```
Applications built with gcc 8.3.0 compiler
```

```
Applications built with gcc 8.3.0 compiler AND openmpi 4.0.2 AND openblas 0.3.8
```
Finding modules

• Use module spider to search for software that's not available
  • Might be hidden due to prerequisites

```
$ module spider r/3.4.4
r: r/3.4.4
```

You will need to load all module(s) on any one of the lines below before the "r/3.4.4" module is available to load.

```
gcc/8.3.0  openblas/0.3.8
```
Software stack compatibility

• Switch compilers, LMOD will switch loaded modules
• Prevents using incompatible software

$ module list
Currently Loaded Modules:
1)usc 3) pmix/3.1.3 5) zlib/1.2.117) libxml2/2.9.9 9) openmpi/4.0.2
2)gcc/8.3.0 4) cmake/3.16.2 6) ncurses/6.18) openblas/0.3.8 10) llvm/9.0.1

$ module swap gcc intel/19.0.4

Inactive Modules:
1) llvm 2) openblas/0.3.8 3) openmpi/4.0.2

Due to MODULEPATH changes, the following have been reloaded:
1) libxml2/2.9.9 2) ncurses/6.1 3) pmix/3.1.3

The following have been reloaded with a version change:
1) cmake/3.16.2 => cmake/3.15.4 2) zlib/1.2.11 => zlib/18.0.4
How to use modules

• Use **module avail** to see what's available
• Use **module load** to load the module

```
$ which python
/usr/bin/python

$ module load python

$ which python
/spack/apps/linux-centos7-x86_64/gcc-8.3.0/python-3.7.6-dd2am3dyv1povhd4rizwfzc45wnsajxf/bin/python
```

• Some modules 'unlock' more modules
  • Compiler
  • MPI
  • BLAS
Saving sets of modules

- If you find yourself loading a set of modules frequently

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>module save</td>
<td>Save current modules to default collection</td>
</tr>
<tr>
<td>module save &lt;name&gt;</td>
<td>Save current modules as &lt;name&gt; collection</td>
</tr>
<tr>
<td>module restore</td>
<td>Load modules in default collection</td>
</tr>
<tr>
<td>module restore &lt;name&gt;</td>
<td>Load modules in &lt;name&gt; collection</td>
</tr>
<tr>
<td>module describe &lt;name&gt;</td>
<td>Show which modules are in &lt;name&gt; collection</td>
</tr>
<tr>
<td>module savelist</td>
<td>Show names of all collections</td>
</tr>
</tbody>
</table>
Creating your own modules

- Module files can point to user-installed software
- They require a specific file structure
- Need to add to $MODULEPATH

![Diagram showing the file structure and paths for creating modules.](image-url)
Creating your own modules

• Check our Discourse page for more details

• How to: Create our own modules
Installing software

• Installing software can be quick and painless
  • With precompiled binaries for your specific operating system, it can be as easy as unzipping a file

• ... Or neither quick nor painless
  • If you have to compile the software yourself using compilers, linkers, Makefiles, external libraries, etc.

• (Or even worse...)
  • If it's from an academic lab from 1999 and requires old versions of multiple libraries which have multiple dependencies!
Installing software

• Generally speaking, software can be installed globally or locally
  • On your laptop, you are the system administrator
  • On HPC, you are not the system administrator

• Globally means system-wide
  • Software is installed to system locations like `/usr/bin` or `/usr/local`
  • Global installs require root privileges

• System-wide installations will not work on HPC
  • Only systems administrators have root privileges on HPC
  • E.g., ”yum install” and “apt install” will not work
Installing software

• CARC users must perform local, or “user”, installs
  • Software installed to 'local' folders
    • /project/<pi_id>/<user>
  • Requires write privileges, which you have in your own directories
  • Software will be accessible by you, even on compute nodes

• It is not always obvious how to perform a user install
  • Depends on software
  • You may have to check documentation
Precompiled binary

- Simplest case
- Just download and extract
- Not always available

```bash
$ cd /project/ttroj_412/software

#Copy tarball
$ wget https://example.com/sample.tar.gz

#Extract files
$ tar xvf sample.tar.gz

#Set your environment (adds a new location to your path)
$ export PATH=/project/ttroj_412/software/sample/bin:${PATH}

#Test installation
$ binary_name
```
Installing Python packages

• Don't forget to load the version of python you want to use

$ module load python/3.7.6

• To check what packages are available use the command

$ pip freeze

• Install package (bash shell)

$ pip install <package_name> --user

• Sometimes you’ll need to install the latest version of a package that is already installed

$ pip freeze
$ pip install <package_name> --upgrade --user
Dependencies for Python packages

- Some packages are Python wrappers for C/C++ libraries
- The installer needs to know where these libraries are
- The h5py package is one example

```
$ HDF5_DIR=/path/to/hdf5
$ HDF5_VERSION=X.Y.Z
$ CC="mpicc"
$ pip install h5py --user
```

- You might have to download the package tarball and edit some files like setup.py
Installing R packages

• Source the version of R you want to use and start R

```
$ module load r
$ R
```

• Install package syntax (you may have to specify a path)

```
> install.packages('<package_name>')</n>
> install.packages('<package_name>', lib='/path/to/packages')
```

• Then load the library when you want to use

```
> library('<package_name>')</n>
> library('<package_name>', lib.loc='/path/to/packages')
```
Dependencies for R packages

- Some packages are R wrappers for C/C++ libraries
  - The installer needs to know where these libraries are
  - You might have to download the package tarball and edit some files

- You can set compilation environment variables like LDFLAGS in the file \${HOME}/.R/Makevars
Singularity

• For difficult installations
• Singularity provides packaged "computing environments"
• Works best with complex dependency chains
• Compatible with Docker
Singularity

Example: lolcow (fortune | cowsay | lolcat)

#Download container image
$ singularity pull shub://GodloveD/lolcow

#Test
$ singularity run lolcow_latest.sif

See this page for more ways to interact with a container: https://sylabs.io/guides/3.0/user-guide/quick_start.html#interact-with-images
Compiling to source code

• C/C++ and Fortran programs are compiled and assembled

header.h
myprogram.c
myprogram.f
(source code)

libutil.so
libgcc.a
(shared object files/libraries)

Preprocess & Compile

myprogram.o
(object file)

Link

myprogram
(binary/executable)
Compilers

- A typical compile command for C code

```
$ gcc ${CCFLAGS} source.c ${CPPFLAGS} ${LDFLAGS} -o myprogram
```

- Where the envs were pre-defined

```
$ CCFLAGS=' -Wall -O3'
$ CPPFLAGS=' -I/path/to/include'
$ LDFLAGS=' -L/path/to/lib -lgsl -lgslcblas -lm'
```

<table>
<thead>
<tr>
<th>ENV</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCFLAGS</td>
<td>Flags to pass the C compiler</td>
</tr>
<tr>
<td>CPPFLAGS</td>
<td>Where the C preprocessor can find include (.h) files</td>
</tr>
<tr>
<td>LDFLAGS</td>
<td>Which libraries (.so, .a files) to use and where the linker can find them</td>
</tr>
</tbody>
</table>
With configure/make

- Manually typing compile and link commands is not feasible
- Software build utilities like autotools, cmake handle this

configure
(paths to include files and libraries are added by you or searched for)

run 'configure'

Makefile
(final compile and link rules)

run 'make'

myprogram
(executable)

run 'make install'

myprogram
(installed)
Building software with modules

• Most modules modify $PKG_CONFIG_PATH
• Many installer scripts check here for prerequisite software

$ module load ncurses
$ ./configure <options>
...
checking curses.h usability... yes
checking curses.h presence... yes
checking for curses.h... yes
checking ncurses.h usability... yes
checking ncurses.h presence... yes
checking for ncurses.h... yes
...

• If we're lucky it's that easy
Building software with modules

• If unlucky, specify with script options

```bash
module load ncurses
./configure --ncurses-root=${NCURSES_ROOT} <other options>
...
checking curses.h usability... yes
checking curses.h presence... yes
checking for curses.h... yes
checking ncurses.h usability... yes
checking ncurses.h presence... yes
checking for ncurses.h... yes
...
```

• Our modules set environment variable `<software>_ROOT` for just this occasion
Building software with modules

• If very unlucky, modify Makefile

... 
override LDFLAGS += -L./nicksrc -L$(GSL_ROOT)/lib
  -L$(OPENBLAS_ROOT)/lib

override CFLAGS += -c -g -p -Wimplicit -I./ -I./nicksrc
  -I$(GSL_ROOT)/include -I$(OPENBLAS_ROOT)/include
...

$ module load ncurses
$ make
Compiling source code

Example: libcaca

#Download tarball
$ git clone https://github.com/cacalabs/libcaca.git
$ cd libcaca

#Run configure script
$ ./bootstrap
$ ./configure --prefix=/project/<pi_id>/<username>/libcaca [other options]

#Run makefile
$ make
$ make install
Compiling source code

- Configure script does not find ncurses library

```sh
checking ncursesw/ncurses.h usability... no
checking ncursesw/ncurses.h presence... no
checking for ncursesw/ncurses.h... no
checking ncurses/ncurses.h usability... no
checking ncurses/ncurses.h presence... no
checking for ncurses/ncurses.h... no
checking ncurses.h usability... no
checking ncurses.h presence... no
checking for ncurses.h... no
checking curses.h usability... no
checking curses.h presence... no
checking for curses.h... no
```
Compiling source code

• In this case, pkg-config is not used
• Manually override with pkg-config, $LDFLAGS, and $CPPFLAGS

$ pkg-config --cflags-only-I ncurses
-D_GNU_SOURCE -I/spack/apps/linux-centos7-x86_64/gcc-8.3.0/ncurses-6.1-akiyo4qrgzlzxw3hggkc42nvv7hz2evj/include

$ pkg-config --libs-only-L ncurses
-L/spack/apps/linux-centos7-x86_64/gcc-8.3.0/ncurses-6.1-akiyo4qrgzlzxw3hggkc42nvv7hz2evj/lib
Compiling source code

Example: libcaca

#Download tarball
$ git clone https://github.com/cacalabs/libcaca.git
$ module load ncurses
$ cd libcaca

#Run configure script
$ CPPFLAGS=$(pkg-config --cflags ncurses) LDFLAGS=$(pkg-config --libs-only-L ncurses) ./configure --prefix=/project/<pi_id>/<username>/libcaca

#Run makefile
$ make
$ make install
Compiling source code

Test installation

$ ./cacafire # ctrl+c to quit
$ ./cacadem
## Troubleshooting

<table>
<thead>
<tr>
<th>Error</th>
<th>What it means</th>
<th>How to fix it</th>
</tr>
</thead>
<tbody>
<tr>
<td>Permission denied</td>
<td>You are trying to install somewhere you don’t have access to.</td>
<td>Install to a location where you have write permissions.</td>
</tr>
<tr>
<td>Read-only file system</td>
<td></td>
<td>e.g., use:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>/configure --prefix=path/to/dir</code></td>
</tr>
<tr>
<td>Library (.so,.a) file not found</td>
<td>Your program needs to link to libraries that the installer can’t find.</td>
<td>Set your library search path.</td>
</tr>
<tr>
<td>Undefined reference to...</td>
<td></td>
<td><code>LDFLAGS=-L/path/to/lib</code></td>
</tr>
<tr>
<td>Cannot find shared object file</td>
<td></td>
<td><code>/configure</code></td>
</tr>
<tr>
<td>Header (.h) files not found</td>
<td>Your program needs to include header files that the installer can’t find.</td>
<td>Set your Include file path.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>CPPFLAGS=-I/path/to/include</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>/configure</code></td>
</tr>
</tbody>
</table>
Getting Help

• Request assistance
  • Email carc-support@usc.edu
  • Office Hours (drop-in)
    • Every Tuesday@2:30pm (Zoom)

• Learn more!
  • Visit carc.usc.edu
  • Request a consultation (anytime)
  • Attend a Workshop (when scheduled)
  • Visit our Discourse page!
Thank you for attending!

Questions?

carc-support@USC.EDU