Installing and Using Software on CARC Systems

Cesar Sul
Research Computing Associate

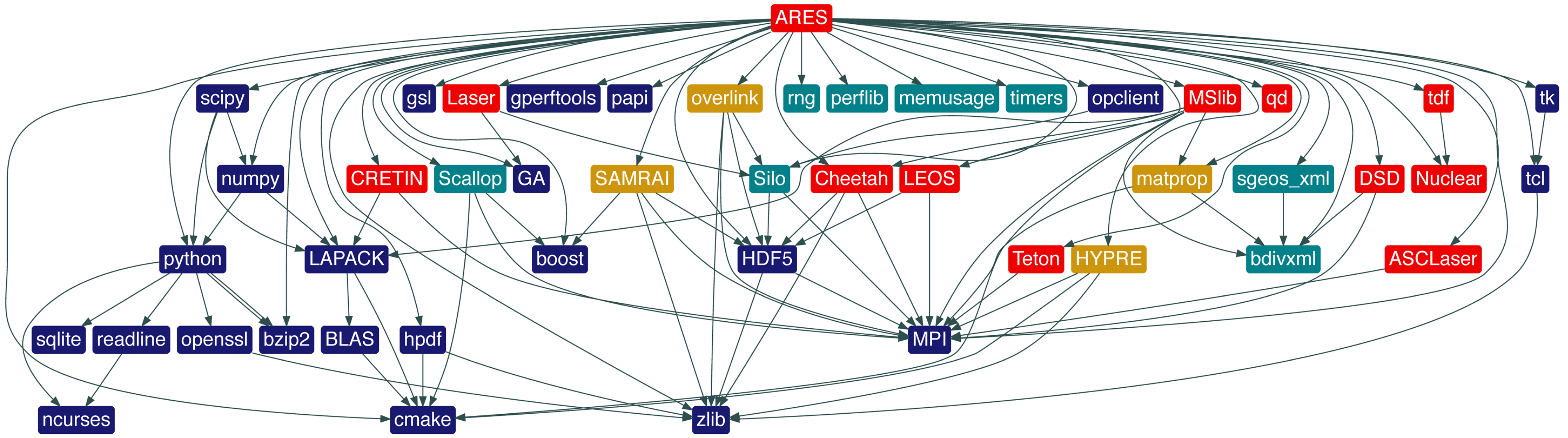
USC | Advanced Research Computing
Enabling scientific breakthroughs at scale

Outline

- CARC managed software
  - Finding software
  - Using modules
  - Neat features
- Installing software
  - Precompiled binary
  - Conda
  - Python
  - R
  - Singularity
  - Building Source Code

Software is complex!

- Dependency graph for ARES



https://computing.llnl.gov/projects/spack-hpc-package-manager

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
- Writen in Lua

What are software modules?

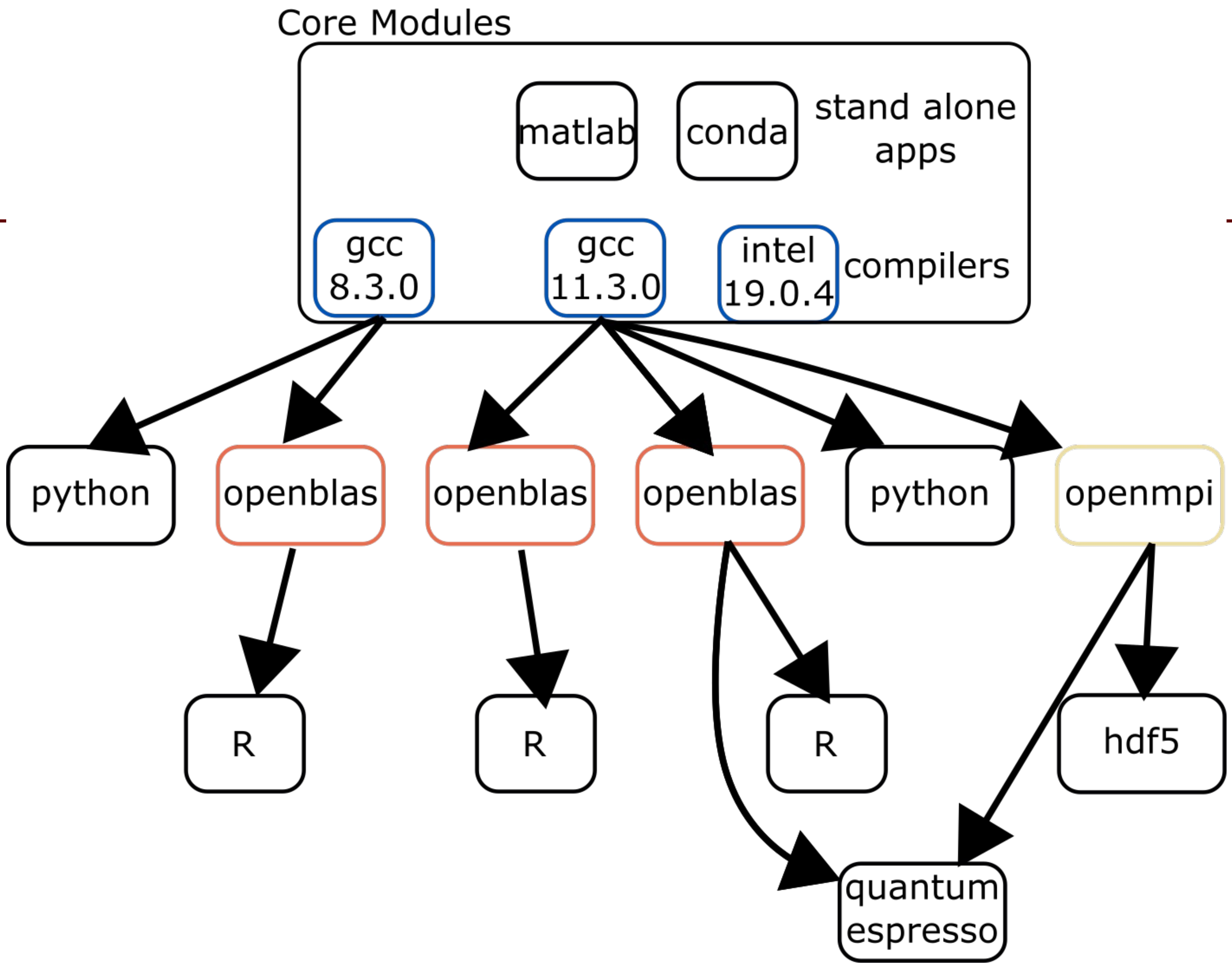
- There are 4 kinds of modules
  - Compiler (gcc, intel)
  - BLAS (Openblas, AMD-blis, netlib) # Math library
  - MPI (OpenMPI, mvapich2, IntelMPI) # For multi node apps
  - Application (most common, bamtools, cmake, libpng...)

What are software 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`


Core Modules
matlab
conda
stand alone apps
gcc 8.3.0
gcc 11.3.0
intel 19.0.4
compilers
python
openblas
openblas
openblas
python
openmpi
R
R
R
hdf5
quantum espresso

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

  cantera/2.4.                                                                    nblas-openmpi

   hypre/2.18.2-openblas-openmpi                    openblas-openmpi
              /spack/apps/lmod/linux-centos7-x86_64/openmpi/4.0.2-ipm3dnv/gcc/8.3.0

hdf5/1.10.6-op                                                                   /2.29-openmpi
hmmer/3.3-open
matio/1.5.13-openmpi  parmetis/4.0.3-openmpi        sundials/5.1.0-openmpi (D)
             /spack/apps/lmod/linux-centos7-x86_64/openblas/0.3.8-2no6mfz/gcc/8.3.0
       r/3.4.4                                                                   .0

 adapterremoval/2.3.1                kbproto/1.0.7                        pdt/3.25.1
  ananaconda3/2                                                           er/2.173
     argtable/2                                                           cale/1.05
  at-spi2-atk/2.26.2                   lcms/2.9                perl-extutils-config/0.008
```

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

Applications built with gcc 8.3.0 compiler

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-
dd2am3dyvlpovhd4rizwfzc45wnsajxf/bin/python
```

- Some modules 'unlock' more modules
  - Compiler
  - MPI
  - BLAS

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
```

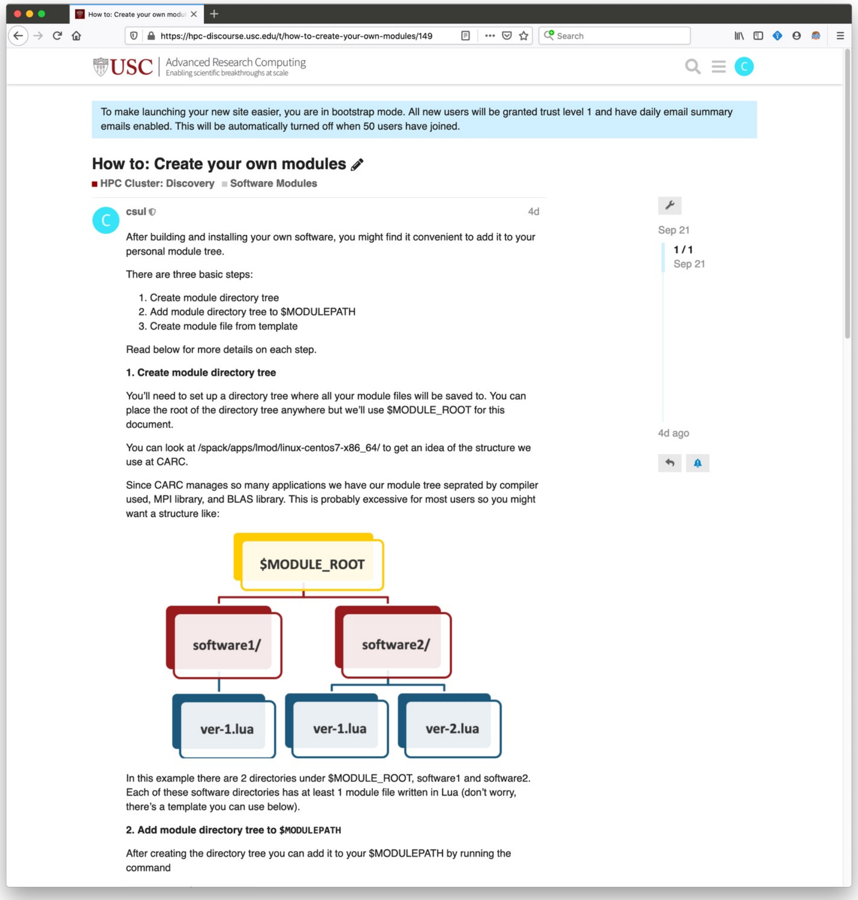
Saving sets of modules

- If you find yourself loading a set of modules frequently

`module save`	Save current modules to default collection
`module save <name>`	Save current modules as `<name>` collection
`module restore`	Load modules in default collection
`module restore <name>`	Load modules in `<name>` collection
`module describe <name>`	Show which modules are in `<name>` collection
`module savelist`	Show names of all collections

Creating your own modules

- Check our Discourse page for more details
- [How to: Create our own modules](https://hpc-discourse.usc.edu/t/how-to-create-your-own-modules/149)


How to: Create your own modul
https://hpc-discourse.usc.edu/t/how-to-create-your-own-modules/149
Search
USC | Advanced Research Computing
Enabling scientific breakthroughs at scale

To make launching your new site easier, you are in bootstrap mode. All new users will be granted trust level 1 and have daily email summary emails enabled. This will be automatically turned off when 50 users have joined.

How to: Create your own modules

HPC Cluster: Discovery Software Modules

csul 4d

Sep 21 1 / 1 Sep 21 4d ago

After building and installing your own software, you might find it convenient to add it to your personal module tree.

There are three basic steps:

1. Create module directory tree
2. Add module directory tree to $MODULEPATH
3. Create module file from template

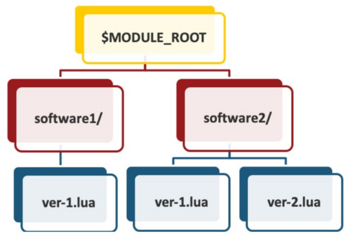
Read below for more details on each step.

1. Create module directory tree

You'll need to set up a directory tree where all your module files will be saved to. You can place the root of the directory tree anywhere but we'll use $MODULE_ROOT for this document.

You can look at /spack/apps/lmod/linux-centos7-x86_64/ to get an idea of the structure we use at CARC.

Since CARC manages so many applications we have our module tree seprated by compiler used, MPI library, and BLAS library. This is probably excessive for most users so you might want a structure like:



In this example there are 2 directories under $MODULE_ROOT, software1 and software2. Each of these software directories has at least 1 module file written in Lua (don't worry, there's a template you can use below).

2. Add module directory tree to `$MODULEPATH`

After creating the directory tree you can add it to your $MODULEPATH by running the command

Installing Your Own Software

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>/software
  - 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

```
$ 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
```

Conda Environments

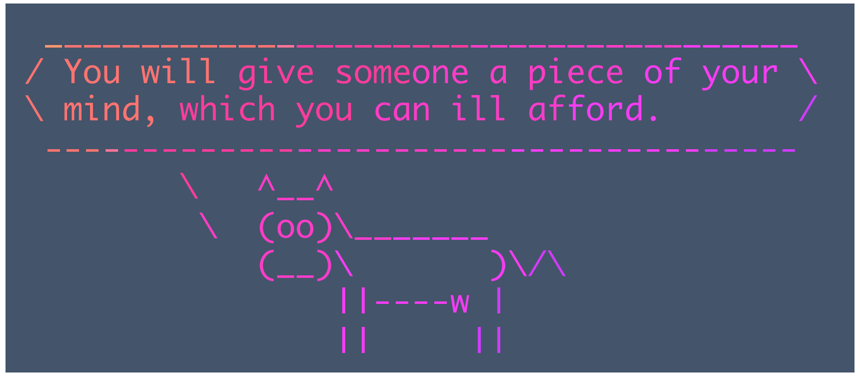
Conda Environments

- Some developers package their software as conda environments
- Create a collection of software packages
- Dependencies are managed for you
- `mamba env create -f envrionment.yml`
- Full documentation here

```
name: mustache
channels:
  - conda-forge
  - defaults
dependencies:
  - pip
  - h5py
  - hdf5
  - numpy
  - python=3.8
  - pip:
    - cooler
    - hic-straw
    - pandas
    - scipy
    - statsmodels
```

Our own environment

- We can also create our own custom environments
- Let's use "lolcow" example
- Create environment with these apps:
  - [fortune]()
  - [cowsay]()
  - [lolcat]()



```
 _______________________________________
/ You will give someone a piece of your \
\ mind, which you can ill afford.       /
 ---------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Our own environment

- Set up conda

```
$ module load conda

$ mamba init bash
```

- Create and name environment

```
$ mamba create -n lolcow
# Enter environment
$ mamba activate lolcow
```

- Install packages

```
$ mamba install auto::pyfortune auto::lolcat agilevic::cowsay
```

Our own environment

- Run "lolcow"

```
$ cowsay $(fortune) | lolcat
```

```
  _________________________________________________
 /                                                 \
| Hey, I had to let awk be better at *something*... |
|  :-) -- Larry Wall in <1991Nov7.200504.25280@netl |
| abs.com>1                                         |
 \                                                 /
  =================================================
                                               \
                                                \
                                                  ^__^
                                                  (oo)\_______
                                                  (__)\       )\/\
                                                      ||----w |
                                                      ||     ||
```

Installing Python Packages

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 list
```

- 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

Installing R packags

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

- Then load the library when you want to use

```
> library('<package_name>’)
> 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/Containers

Singularity

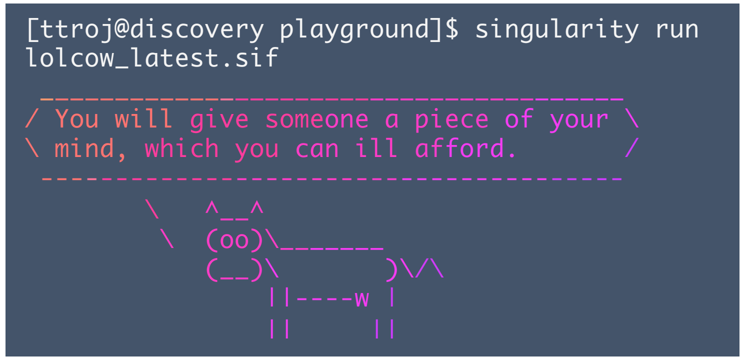
- For difficult installations
- Singularity provides packaged "computing environments"
- Works best with complex dependency chains
- Compatible with Docker

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



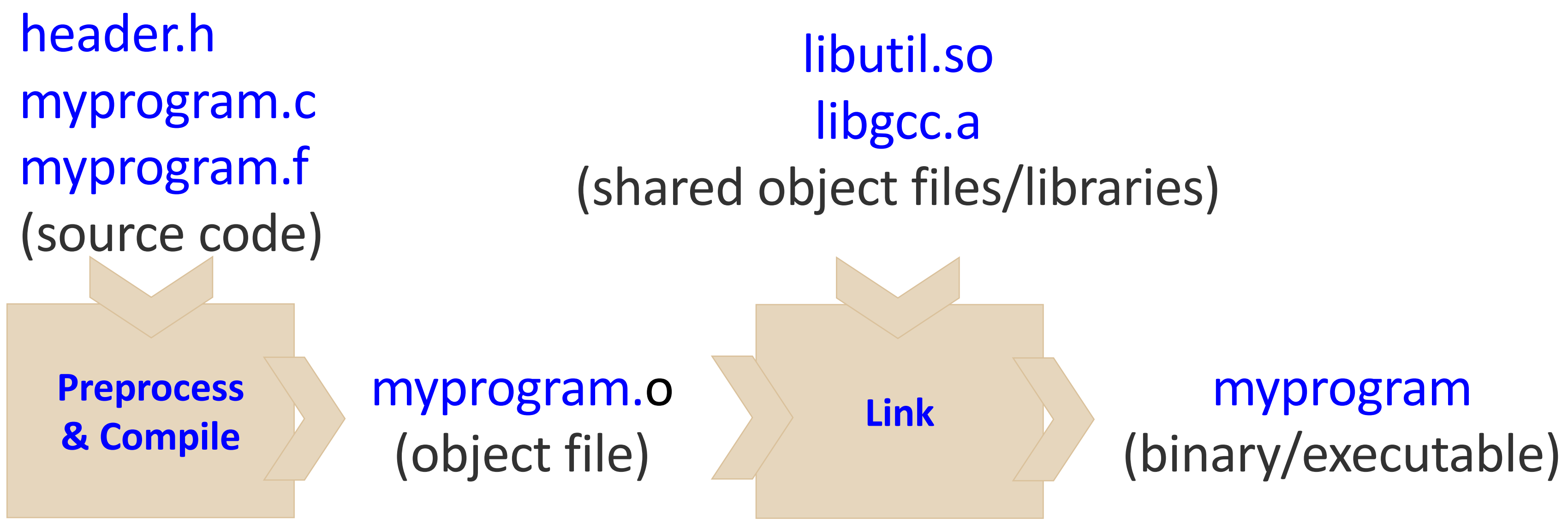
```
[ttroj@discovery playground]$ singularity run
lolcow_latest.sif
 _______________________________________
/ You will give someone a piece of your \
\ mind, which you can ill afford.       /
 ---------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

USC | Advanced Research Computing
Enabling scientific breakthroughs at scale

Building Source Code

Compiling to source code

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



Compilers

- A typical compile command for C code

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

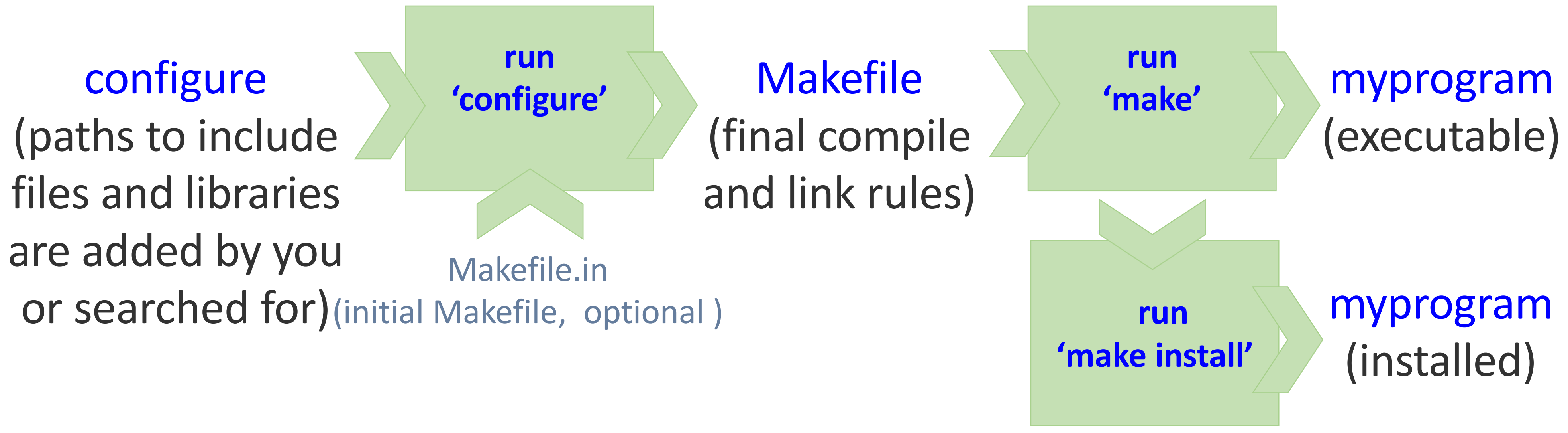
- Where these environment variables were pre-defined

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

CCFLAGS	Flags to pass the C compiler
CPPFLAGS	Where the C preprocessor can find include (.h) files
LDFLAGS	Which libraries (.so, .a files) to use and where the linker can find them

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.in
(initial Makefile, optional)
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

```
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: vim

```
#Download tarball
$ git clone https://github.com/vim/vim.git
$ cd vim

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

#Run makefile
$ make
$ make install
```

Compiling source code

- Configure script can't find libtinfo.so (provided by ncurses)

```
checking --with-tlib argument... empty: automatic terminal library
selection
checking for tgetent in -ltinfo... no
checking for tgetent in -lncurses... no
checking for tgetent in -ltermlib... no
checking for tgetent in -ltermcap... no
checking for tgetent in -lcurses... no
no terminal library found
checking for tgetent()... configure: error: NOT FOUND!
```

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

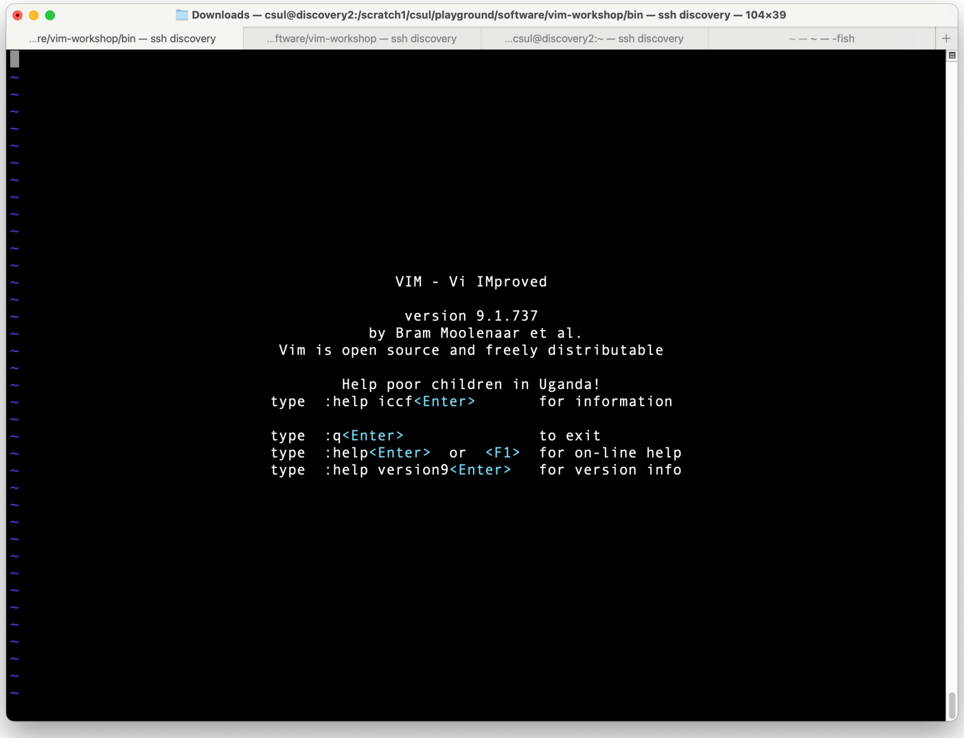
Example: vim

```
#Download tarball
$ git clone https://github.com/vim/vim.git
$ cd vim

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

#Run makefile
$ make
$ make install
```

Compiling source code

Test installation

```
$ ./vim # :q! to quit
```



Downloads — csul@discovery2:/scratch1/csul/playground/software/vim-workshop/bin — ssh discovery — 104×39
...re/vim-workshop/bin — ssh discovery
...ftware/vim-workshop — ssh discovery
...csul@discovery2:~ — ssh discovery
~ — ~ — -fish

```
                    VIM - Vi IMproved

                     version 9.1.737
                 by Bram Moolenaar et al.
        Vim is open source and freely distributable

               Help poor children in Uganda!
type  :help iccf<Enter>       for information

type  :q<Enter>               to exit
type  :help<Enter>  or  <F1>  for on-line help
type  :help version9<Enter>   for version info
```

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

USC | Advanced Research Computing
Enabling scientific breakthroughs at scale