Before we begin

- Best to have X display forwarding enabled
  - On Mac, download and install XQuartz from www.quartz.org
  - On PC, check that X forwarding is enabled in your client

- Log in to hpc-login2 or hpc-login3 with -X option
  - Go to your /staging directory and create workshop space
    $ cd /staging/<project>/<user>
    $ mkdir workshop
    $ cd workshop

- Slides and files used are here -- download slides to follow along
  - /home/rcf-proj/workshop/handouts/
  - /home/rcf-proj/workshop/installing_software/

Installing software on HPC
(Handouts available at /home/rcf-proj/workshop/handouts)

Erin Shaw and Cesar Sul
Advanced Cyberinfrastructure Research and Education Facilitation
USC Center for High-Performance Computing
HPC Facilitation

- Request assistance
  - Email hpc@usc.edu (email again!)
  - Drop-in to office hours
    - **UPC:** every Tuesday, 2:30p (LVL 3M)
    - **HSC:** after workshops, 4:30 (NML 203)
  - Request a consultation (request link is on web site)

- Learn more!
  - Webpages online at [https://hpcc.usc.edu](https://hpcc.usc.edu)
  - Attend a workshop (when scheduled)
    - **UPC:** Fridays, 2:30p (VPD 2016)
    - **HSC:** Wednesdays, 1:30 (MCA 249)

Outline

- **Introduction**
  - Setting your environment
  - Precompiled binary
  - Compiling source code
  - Compiling source code with dependencies
  - Installing from GitHub
  - Installing Python packages
  - Installing R libraries
  - Licensed software
  - Optimized compiling
Introduction

- Installing and running software typically requires knowledge of the computer it will run on, at some level (varies)
- Computers differ in some of the following ways:
  - Computer architecture (instruction set, logic)
  - Operating system (Linux, Windows, MacOSX)
- For example, HPC computers feature:
  - Intel 64 bit processors (x86_64)
  - CentOS 7.x operating systems
    • CentOS is a derivative of Red Hat Enterprise Linux
    • Linux is a derivative of Unix for personal computers

Introduction

- Installing software can be quick and painless
  - With precompiled binaries for your specific operating systems, for example, it can be as easy as unzipping a file
- Or neither
  - If you have to compile the software yourself using compilers, linkers, Makefiles, external libraries, etc.
- Or even worse...
  - It's from an academic lab from 1999 and requires old versions of multiple libraries which have multiple dependencies!
Introduction

- Software can be installed globally or locally
- HPC users will perform local, or "user", installs
  - Software installed to 'local' folders
    - `/home/rcf-proj/<project>/<user>`
    - Always work in project directory!
  - Requires write privileges, which you have in your own directories
  - Software will be accessible by you, even on compute nodes

Introduction

- Globally means system-wide, where software is installed to system locations like `/usr/bin` or `/usr/local`
  - Global installs require root privileges, and only systems administrators have root privileges on HPC
- So system-wide installations will not work on HPC

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>GNU/Linux</td>
<td><code>apt install</code></td>
</tr>
<tr>
<td>Debian/Ubuntu</td>
<td></td>
</tr>
<tr>
<td>Fedora</td>
<td><code>yum install</code></td>
</tr>
<tr>
<td>Gentoo</td>
<td><code>emerge install</code></td>
</tr>
<tr>
<td>Mac</td>
<td><code>brew install</code></td>
</tr>
<tr>
<td>Homebrew</td>
<td></td>
</tr>
<tr>
<td>MacPorts</td>
<td><code>port install</code></td>
</tr>
<tr>
<td>BSD</td>
<td><code>pkg install</code></td>
</tr>
<tr>
<td>FreeBSD</td>
<td><code>pkg_add</code></td>
</tr>
</tbody>
</table>
Outline

Introduction
Setting your environment
Precompiled binary
Compiling source code
Compiling source code with dependencies
Installing from GitHub
Installing Python packages
Installing R libraries
Licensed software
Optimized compiling

Setting your environment

- Environment variables!
  - *Are named string variables*, e.g.
    - HOME=/home/rcf-proj/erinshaw, USER=erinshaw
    - mydata=/home/rcf-proj/erinshaw/ess/data
  - By convention, **system-set** envs are all UPPERCASE and **user-set** envs are at least partially lowercase
  - The value is accessed by preceding the variable with a "$"
    - Type: echo \$HOME, echo \$SHELL, echo \$mydata
    - Type: env (to print all envs)

- Envs can be set by the system, programs, and users; and are available system-wide, on command line, and inside code
Setting your environment

- $PATH contains a list of directories (locations)
  - The locations are searched sequentially by the shell, when you type a command, to find the specified command
    - The first match will be executed
    - When there is no match, the shell will produce an error

```
$ python
$ which python
/usr/bin/python
$ echo $PATH
/usr/lib64/qt-3.3/bin:/opt/mam/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin
$ python3
-bash: python3: command not found
```

Setting your environment

- Changing the $PATH variable, by ‘sourcing’ (running/executing) HPC’s setup.sh script changes, will change which version of python is executed

```
$ source /usr/usc/python/2.7.8/setup.sh
$ echo $PATH
/usr/usc/python/2.7.8/bin:/usr/lib64/qt-3.3/bin:/opt/mam/bin:/opt/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin
$ which python
/usr/usc/python/2.7.8/bin/python
```

- Why has the /usr/usc version of python been selected?
Setting your environment

- Don’t add environment variables to shell files like `.bashrc`, `.bash_profile`, `.login`, or `.profile`!
  - The wrong environment variables could lead to conflicting settings
  - You’ll need to be able to ‘turn off’ environment variables/settings
  - This happens all the time!

- Best to create environment setup scripts that initialize the settings you need
  - e.g., `env.sh` (we will be creating these)

- Especially if you are doing work for multiple projects

- Keep track of how software is installed
  - You might have to do it again
  - You might have to reconfigure it
  - Environment settings could influence your results

- You want your results to be reproducible
Outline

Introduction
Setting your environment
**Precompiled binary**
Compiling source code
Compiling source code with dependencies
Installing from GitHub
Installing Python packages
Installing R libraries
Licensed software
Optimized compiling

Precompiled binary

Example: Xaos

Step 1: Acquire tarball

```
$ cp /home/rcf-proj/workshop/installing_software/xaos-3.6-bin.tar.gz .
```

Step 2: Extract (x=extract, v=verbose, f=file)

```
$ tar xvf xaos-3.6-bin.tar.gz
```

Step 3: Set your environment (adds a new location to your path)

```
$ export PATH=/staging/<prj>/usr/workshop/xaos/bin:$PATH
```

Step 4: Test installation (requires that X11 forwarding be enabled)

```
$ xaos
```
Outline

Introduction
Setting your environment
Precompiled binary

Compiling source code
Compiling source code with dependencies
Installing from GitHub
Installing Python packages
Installing R libraries
Licensed software
Optimized compiling

Compiling (or not)

- Scripts can be edited and run without formally compiling
  - Some programs (bash, perl) interpret the text in the files*
  - The interpreters themselves must be compiled for each OS

  myprogram.sh
  myprogram.pl
  myprogram.js
  bash myprogram.sh
  perl myprogram.pl
  javascript myprogram.js

- All HPC researchers write shell scripts (pbs files)
- Many system administrators use perl scripts
Compiling to bytecode

- Code can be compiled to “byte code” and run by a virtual machine (the VMs themselves are compiled for each OS)
  
  myprogram.java
  myprogram.py

  Compile

  myprogram.class
  myprogram.pyc

  java myprogram [class]
  python myprogram [pyc]

- GATK tools, for example, are written in java
- Many HPC researchers are using python

Compiling to machine code

- C/C++ and Fortran programs are compiled and assembled
  - The output is machine-executable object code

  myprogram.c
  myprogram.f
  (source code)
  x86_lib.so
  math_lib.so
  (shared object files/libraries)

  Preprocess & Compile

  myprogram.o
  (object file)

  Link & Assemble

  myprogram
  (executable program)

- Many HPC users write their own code in C, C++, or Fortran
HPC Compilers

- HPC supports open source and commercial compilers for C, C++, and Fortran programming languages

<table>
<thead>
<tr>
<th>Vendor</th>
<th>C/C++</th>
<th>Fortran</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>GNU (open source)</td>
<td>gcc/g++</td>
<td>gfortran</td>
<td>/usr/bin/gnu/gcc</td>
</tr>
<tr>
<td>Intel*</td>
<td>icc/icpc</td>
<td>ifort</td>
<td>/usr/us/intel</td>
</tr>
<tr>
<td>PGI*</td>
<td>pgcc/pgc++</td>
<td>pgfortran</td>
<td>/usr/us/sgi</td>
</tr>
<tr>
<td>OpenMPI**</td>
<td>mpicc/mpicxx</td>
<td>mpifort</td>
<td>/usr/us/openmpi</td>
</tr>
</tbody>
</table>

*Commercial compilers
**MPI compilers are "wrappers" that can use GNU/Intel/PGI

A typical compile command for C code looks like

```bash
$ gcc $(CCFLAGS) source.c $(CPPFLAGS) $(LDFLAGS) -o binary_name
```

Where the envs were predefined like

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

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>What is it?</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

- Applications may have hundreds of files and so manually typing compile and link commands is not feasible
- Software build utilities, e.g. gnu’s autotools, are used for this

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

- run ‘configure’

- Makefile
  (initial Makefile)

- run ‘make’

- Makefile
  (final compile and link rules)

- run ‘make install’

- myprogram
  (installed)

- myprogram
  (executable)

HPC Libraries

- Scientific software usually requires low level math libraries
  - BLAS (Basic Linear Algebra Subprograms) is the de-facto standard for linear algebra libraries for C and Fortran
  - Set of low-level routines for performing common linear algebra operations, such as vector addition and matrix multiplication

- HPC maintains optimized BLAS libraries for user installs
  - OpenBLAS, Intel MKL – Math Kernel Library

- HPC-installed software in /usr/usc/ is compiled with OpenBLAS
  - NumPy, Matlab, R, Mathematica
Compiling source code

Example: XaoS

Step 1: Acquire tarball

```bash
$ wget /home/rcf-proj/workshop/installing_software/xaos-3.6.tar.gz .
$ tar xvf xaos-3.6.tar.gz
$ cd xaos
```

Step 2: Read install directions

```bash
$ less INSTALL
```

Step 3: Run configure script

```bash
$ ./configure --prefix=/staging/prj/usr/workshop/XaoS
```

Step 4: Run makefile

```bash
$ make
$ make install
```

Compiling source code

Step 5: Set your environment (Put this in a script)

- edit (nano), save (^o) and exit (^x)

```bash
$ nano env.sh
```

```bash
#!/bin/bash
PATH=/staging/prj/usr/workshop/XaoS/bin:$PATH
MANPATH=/staging/prj/usr/workshop/Xaos/share/man:$MANPATH
export PATH MANPATH
```

```bash
$ source env.sh
```
Compiling source code

- **Step 6: Test installation**
  - Run XaoS (Need X11 forwarding enabled)
  - Click on part of the image to zoom in

```
$ xaos
```

---

**Troubleshooting**

<table>
<thead>
<tr>
<th>Type of problem</th>
<th>What does it mean</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 permission”</td>
</tr>
<tr>
<td>Read-only file system</td>
<td></td>
<td>ie, <code>./configure --prefix=path/to/dir</code></td>
</tr>
<tr>
<td>Library (.so or .a) file not found</td>
<td>Your program uses libraries (3rd party code) that the installer can’t find.</td>
<td>Set your library search path</td>
</tr>
<tr>
<td>Undefined reference to...</td>
<td></td>
<td>LDFLAGS=-L/path/to/lib ./configure</td>
</tr>
<tr>
<td>Cannot find shared object file</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Header (.h) files not found</td>
<td>Your program uses headers (3rd party code) that the installer can’t find</td>
<td>Set your “Include” path</td>
</tr>
<tr>
<td>Undefined reference to...</td>
<td></td>
<td>CPPFLAGS=-I/path/to/include ./configure</td>
</tr>
</tbody>
</table>
Outline

Introduction
Setting your environment
Precompiled binary
Compiling source code

Compiling source code with dependencies
Installing from GitHub
Installing Python packages
Installing R libraries
Licensed software
Optimized compiling

Compiling source with dependencies

- Example: NetCDF requires HDF5

Step 0: Install HDF5 (It’s already in /usr/usc/hdf5)
Step 1: Download tarball

```bash
$ wget https://github.com/Unidata/netcdf-c/archive/v4.4.1.tar.gz
```
Step 2: untar, cd to directory and read install directions

Step 3: Run configure script

```bash
$ ./configure --prefix=/path/to/install/dir
```

Example:
NetCDF requires HDF5

Step 0: Install HDF5 (It's already in /usr/usc/hdf5)
Step 1: Download tarball

```bash
$ wget https://github.com/Unidata/netcdf-c/archive/v4.4.1.tar.gz
```
Step 2: untar, cd to directory and read install directions

Step 3: Run configure script

```bash
$ ./configure --prefix=/path/to/install/dir
```
Step 4: Run makefile

$ make
$ make install

Step 5: Create an environment script

$ nano env.sh

#!/bin/bash
source /usr/usc/hdf5/default/setup.sh
PATH=/path/to/install/dir/bin:$PATH
LD_LIBRARY_PATH=/path/to/install/dir/lib:$LD_LIBRARY_PATH
export PATH LD_LIBRARY_PATH

Outline

Introduction
Setting your environment
Precompiled binary
Compiling source code
Compiling source code with dependencies
Installing from GitHub
Installing Python packages
Installing R libraries
Licensed software
Optimized compiling
Installing from GitHub

Step 1: Download source code
$ git clone <paste_project_url_here>

Step 2: Compile source
$ ./configure --prefix=/path/to/install/dir
$ make
$ make install

Step 3: Set environment
Outline

Introduction
Setting your environment
Precompiled binary
Compiling source code
Compiling source code with dependencies
Installing from GitHub
**Installing Python packages**
Installing R libraries
Licensed software
Optimized compiling

### Installing Python packages

- Don't forget to source the version of python you want to use

```bash
$ source /usr/usc/python/2.7.8/setup.sh
```

- To check what packages are available use the command

```bash
$ pip freeze
```

- Python's default installation location is `${HOME}/.local`
  - You have only 1GB disk quota and 100,000 file quota in your home directory and a python package install can easily be larger
  - Solution is to create a symbolic link, or 'symlink', which is a pointer (named .local) to another directory with more disk space
Initial Python package

- Create a directory outside of your home directory, for storing python packages, then create a symbolic link to it called .local

```bash
$ cd ~
$ mkdir -p /staging/prj/usr/python_packages
$ ln -s /staging/prj/usr/python_packages .local
```

- Install package (bash shell)

```bash
$ source /usr/usc/python/<version>/setup.sh
$ pip install <package_name> --user
```

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

```bash
$ 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

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

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

Introduction
Setting your environment
Precompiled binary
Compiling source code
Compiling source code with dependencies
Installing from GitHub
Installing Python packages
**Installing R libraries**
Licensed software
Optimized compiling

Installing R packages

- R’s default installation location is $(HOME)/R
  - You have only 1GB disk quota and 100,000 file quota in your home directory and a python package install can easily be larger
  - Solution is to create a symbolic link, or ‘symlink’, which is a pointer (named R) to another directory with more disk space
- Create a directory outside of your home directory, for storing python packages, then create a symbolic link to it called “R”

```
$ cd ~
$ mkdir -p /staging/<project>/<user>/R_packages
$ ln -s /staging/<project>/<user>/R_packages R
```
Installing R packages

- Source the version of R you want to use and start R
  
  ```
  $ source /usr/usc/R/default/setup.sh
  $ R
  ```

- Install package syntax (you may have to specify path)
  
  ```
  > install.packages("<package_name>"
  > install.packages("<package_name>", lib="/path/to/packages")
  > library (<package_name>, lib.loc="/path/to/packages")
  ```

- Then load the library when you want to use
  
  ```
  > library ('<package_name>')
  > library ('<package_name>', lib.loc="/path/to/packages")
  ```

## Example

Example

```
> install.packages("gplots")
Warning in install.packages("gplots"):
  'lib = "/auto/usc/R/3.3.1/lib64/R/library"' is not writable

Would you like to use a personal library instead? (y/n) y
Would you like to create a personal library
~/R/x86_64-pc-linux-gnu-library/3.3
to install packages into? (y/n) y
--- Please select a CRAN mirror for use in this session ---
also installing the dependencies 'bitops', 'gtools', 'gdata', 'caTools'
```
Installing R packages

- You can display packages that are installed in the USC versions of R, or that you have installed.
  - This is not an optimal way to view packages
  - Better if you can check one package at a time

```r
> installed.packages(lib.loc="/usr/usc/R/default/lib64/R/library")
> installed.packages(lib.loc="~/R/x86_64-pc-linux-gnu-library/3.3")
> system.file(package="gplots")
```

- Don’t forget to load the library!

```r
> library("gplots")
> library("gplots", lib.loc="~/R/x86_64-pc-linux-gnu-library/3.3")
```

Display your library paths in R with .libPaths()

```r
> .libPaths()
[1] "/auto/rcf-proj/ess/erinshaw/homeR/x86_64-pc-linux-gnu-library/3.3"
[2] "/auto/usc/R/3.3.1/lib64/R/library"
```

- You can also set R’s library path with the R_LIBS_USER environment variable or in .Rprofile
  - e.g., add line to your env.sh or PBS script

```
$ export R_LIBS_USER=/R/x86_64-pc-linux-gnu-library/3.3:$R_LIBS_USER
```

  - e.g. create an .Rprofile file in your home directory and add path

```
$ touch .Rprofile
$ nano .Rprofile
libPaths( c(.libPaths(), ":~/R/x86_64-pc-linux-gnu-library/3.3")
```
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

Outline

Introduction
Setting your environment
Precompiled binary
Compiling source code
Compiling source code with dependencies
Installing from GitHub
Installing Python packages
Installing R libraries
Licensed software
Optimized compiling
Licensed software

- HPC compute nodes live in a private network
  - There is no public internet access
- We recommend you get a floating license
  - HPC will host the license server
  - Compute nodes will have access to this server
  - HPC can set restrictions so only you or your user group can use it
  - You will be in charge of maintaining the license
- There are situations when licenses must reside outside HPC
  - Mixed use licenses for desktops and HPC
  - HPC can accommodate these situations, too

Outline

Introduction
Setting your environment
Precompiled binary
Compiling source code
Compiling source code with dependencies
Installing from GitHub
Installing Python packages
Installing R libraries
Licensed software
Optimized compiling
Optimized Compiling

- It's possible to increase performance through compilation
  - Use optimized libraries like FFTW, Intel MKL, OpenBLAS...
  - Use a commercial compiler
  - Intel allows you to compile for specific machine architecture
  - The binary detects the best instruction set based on which node it's running on
- HPC maintained software is compiled this way
- For more details, send questions to hpc@usc.edu

Thank you for attending!
Questions? (HPC@USC.EDU)