Using and Installing Software

Peter Ruprecht
peter.ruprecht@colorado.edu
www.rc.colorado.edu

Slides:
https://github.com/ResearchComputing/Basics_Supercomputing

Outline

- Quick Linux review
- Environment modules (motivation and examples)
- Installing your own versions of software using autoconf
- Installing personal python modules
- Installing personal R modules

Get Logged in

- `ssh user00XY@tutorial-login.rc.colorado.edu`

- Once you are on the login node, from there ssh to a Summit compile node:
- `ssh scompile`

Quick Linux Review

- `pwd` – print path of current directory
- `cd` – change to a different directory
  - `.` – current directory
  - `..` – one directory higher
  - `~` – home directory

- `mv file1 file2` – rename “file1” to “file2”
- `cp file1 dir2` – copy “file 1” into “directory 2”
- `rm` – remove a file
- `mkdir` – create a directory
- `rmdir` – remove and empty directory

- `ls -l` – detailed listing of contents of current directory

More Linux Review

- `cmd1|cmd2` – redirect output of "command 1" as input to "command 2". Make a pipeline!
- `cmd1 >file.txt` – redirect output of "command 1" into a new file named "file.txt"

- `echo $VARIABLE` – print the value of an environment variable
- `export VARIABLE=value` – set (or reset) the value of an environment variable

Environment Modules – *why?*

- The de facto standard for managing software packages on shared HPC systems
- "yum install" or "apt get" aren't flexible enough
  - Might not be optimized for the local hardware
  - Doesn't support multiple versions
- *Setting up your shell environment by hand each time you want to use a software package is tedious, time-consuming, error-prone, and non-reproducible*

Linux Environment Review

- Shell is command-line interface between user and operating system
- The shell's behavior can be customized by setting environment variables

  (eg, `export PATH=$PATH:~/bin`)
- Some commonly used environment variables include
  - `PATH`: list of directories to search for commands
  - `DISPLAY`: screen where graphical output will appear
  - `MANPATH`: directories to search for manual pages
  - `LANG`: current language encoding
  - `LD_LIBRARY_PATH`: directories to search for shared objects (dynamically-loaded libs)
  - `LM_LICENSE_FILE`: files to search for FlexLM software licenses

Environment Modules

- "module" command is an easier way to set the appropriate environment for using a specific application
- The necessary environment variable settings or modifications are defined in a "modulefile", normally maintained by the system administrator
- However, you can create your own modulefiles
- Modules are "loaded" prior to using the corresponding application

Examples

- Show what modules you currently have loaded
  - `module list`
- Show what modules are currently available to load
  - `module avail`
- Load a module
  - `module load intel/16.0.3`
- Unload a module
  - `module unload fftw/3.3.4`
- Clear all loaded modules
  - `module purge`

Hierarchical Modules

- Important to avoid loading incompatible modules
- Need to ensure that prerequisite modules are loaded
- Hierarchical modules to the rescue!
  - Maintains a consistent set of loaded modules
  - Can't load a module until its prereqs are loaded
  - *Can't even see a module in the list until its prereqs are loaded*
- `module spider` allows searching for a package

Module Practice

- SSH to tutorial-login.rc.colorado.edu , then
- `ssh scompile`

- What versions of hdf5 are available?
- Choose your favorite version and load it
  - load prerequisites first!
  - `module list` to confirm everything looks ok
- Clear loaded modules (purge)
- Load a module for lammps that uses the gcc compiler and openmpi MPI implementation
- What does the LAMMPS software do?

Compiling

- Start with source code (in, eg, C or Fortran) in a text file
- Use a program called a “compiler” to turn the source code in to an executable program
- Additional function calls that are needed by your program but aren’t in your source code can be linked into the executable via external libraries
- A “Makefile” can contain instructions about how to build an application

Building Programs w/ “autoconf”

- aka “configure, make, make install”
- “configure” checks whether you have all the prerequisite software available
- You can add your own configuration options
- Then, automatically creates appropriate Makefiles that are used to build an application
- “make” reads the newly-created Makefiles and uses them to compile the application
- “make install” installs the compiled software

autoconf Example

- `ssh scompile`
- `module purge`
- `cd /projects/$USER`
- `mkdir –p software/src ; cd software/src`
- `wget http://ftp.gnu.org/gnu/bison/bison-3.0.4.tar.gz`
- `tar -xzf bison-3.0.4.tar.gz`
- `cd bison-3.0.4`
- `./configure --help`
- `module load intel`
- `CC=icc CXX=icpc ./configure --prefix=/projects/$USER/software/bison-3.0.4`
- `make`
- `make install`

autoconf Example, cont'd

- `bison --version`
- `ls /projects/$USER/software/bison-3.0.4`
- `which bison`
- `/projects/$USER/software/bison-3.0.4/bin/bison --version`
- `echo $PATH`
- `export PATH=/projects/$USER/software/bison-3.0.4:$PATH`
- `which bison`
- `bison --version`

Personal Python Packages

- Is the package you need already available in the python module?

- `module purge`
- `module load python/3.5.1`
- `pip list   (or pip freeze)`
- `pip freeze | grep numpy`
- `pip freeze | grep -i pydoe`
-

Install a Python Package

- Use “pip” to install pyDOE for our own personal use

- `mkdir /projects/$USER/python_libs`
- `module load gcc`
- `pip3 install --install-option="--prefix=/projects/$USER/python_libs" pyDOE`
- `python -c "import pydoe"`
- `export PYTHONPATH=$PYTHONPATH:/projects/$USER/python_libs/lib/python3.5/site-packages`
- `python -c "import pydoe"`
-

More on Python Packages

- If you want to do the installation a bit more “by hand” can download the package source code and run
- `python setup.py install --prefix=/projects/$USER/python_libs`
- Perhaps after editing setup.py

- Python “virtualenv”
  - Great if you want to install several versions of your own packages
  - If you need several specific packages together
  - Create an “isolated” python environment

Personal R Packages

- Is the package you need already available in the R module?

- `module purge`
- `module load R/3.3.0`
- `R`

```
> ip <- installed.packages();
> ip
```

Install an R Package

- Install “abind” for our own personal use
- Need to set .libPaths and then use install.packages

- `mkdir /projects/$USER/R_libs`
- `module load R`
- `R`

```
> .libPaths(c('/projects/user0039/R_libs',.libPaths()))
> install.packages("abind")
> library(abind)

> search()
```

Questions?

- Email rc-help@colorado.edu

- Link to survey on this topic:
  http://tinyurl.com/curc-survey16
  Speaker: Peter Ruprecht
  Title: Using and Installing Software (July 2017 BSW)

- Slides:
  https://github.com/ResearchComputing/Basics_Supercomputing