Software Installation 101

Geoffrey Lentner, Lead Research Data Scientist

PURDUE UNIVERSITY | Rosen Center for Advanced Computing

About Me

Data Scientist, Astrophysicist, Research Software Engineer

Things I do at the Rosen Center for Advanced Computing (RCAC)

- **System Support**
- **Scientific Software**
- **Consulting**
- **Training and Teaching**
- **Outreach and Engagement**
- **Innovation**

Things we assume you already know

- Linux command-line basics (Unix 101 and Unix 102)
- Cluster basics (Clusters 101)

Overview

Topics covered

- Fundamentals *(discussion)*
- Unpack and go *(example 1)*
- Compile from source *(example 2)*
- Compile from source with dependencies *(example 3)*
- Questions

Overview

Topics not covered

This is *Software Installation* ***101***, only the fundamentals.

201, 202, 203, 204

Package Installation

(Python, R, Julia, …)

301, 302

Containers

(Docker, Singularity, …)

401

Advanced Topics

Linux, ELF, x86 (arch), linking, MPI, CUDA

0

Fundamentals

Filesystem and environment variables basics.
Programs vs Libraries.
Managing your environment.

Main take-away points

- You can “install” ***anything*** you want into any directory that you have ***write permissions*** on.
- You can *run* a program from ***anywhere***. It’s *installed* when your environment is made *aware* of it.
- Modules (LMOD) make ***composing*** environments on-the-fly easy on a shared system.

Let's start with *programs*

What does it mean for a program to be installed?

```
user@brown-fe01 ~$ myquota

Type     Location           Size    Limit    Use        Files    Limit     Use
=============================================================================
home     glentner         10.4GB   25.0GB    42%            -        -      -
scratch  brown            70.8GB  200.0TB  0.03%          409k   2,000k    20%
depot    itap            102.9TB  200.0TB    52%            -        -      -
depot    rcacdsg         161.7GB    1.0TB    16%            -        -      -
 ...


user@brown-fe01 ~$ which myquota
/usr/local/bin/myquota
```

What does it mean for a program to be installed?

```
user@brown-fe01 ~$ myquota

Type      Location            Size    Limit    Use         Files    Limit     Use
=============================================================================
home      glentner          10.4GB   25.0GB    42%             -        -       -
scratch   brown             70.8GB  200.0TB  0.03%           409k   2,000k     20%
depot     itap             102.9TB  200.0TB    52%             -        -       -
depot     rcacdsg          161.7GB    1.0TB    16%             -        -       -
...


user@brown-fe01 ~$ which myquota
/usr/local/bin/myquota
```

Directory on $PATH

```
/usr/local/bin/myquota
```

Can any file be executed?

```
user@brown-fe01 ~$ echo "seq 4" > my_program
```

Can any file be executed?

```
user@brown-fe01 ~$ echo "seq 4" > my_program

user@brown-fe01 ~$ my_program
zsh: command not found: my_program

user@brown-fe01 ~$ ./my_program
zsh: permission denied: ./my_program

user@brown-fe01 ~$ ls -lh my_program
-rw-r--r-- 1 glentner itap 6 Mar  3 11:40 my_program
```

Not on $PATH

Not *executable*

Can any file be executed?

```
user@brown-fe01 ~$ echo "seq 4" > my_program

user@brown-fe01 ~$ my_program
zsh: command not found: my_program

user@brown-fe01 ~$ ./my_program
zsh: permission denied: ./my_program

user@brown-fe01 ~$ ls -lh my_program
-rw-r--r-- 1 glentner itap 6 Mar  3 11:40 my_program

user@brown-fe01 ~$ chmod +x my_program

user@brown-fe01 ~$ ./my_program
1
2
3
4
```

Not on $PATH

Not *executable*

Where can you install programs?

- **Any directory that you control (have *write* permissions to)**
  - `$HOME` or `$RCAC_SCRATCH` (warning!)
  - `/depot/<group>/apps` (if you are a member of <group>-app)
- **We cannot allow everyone to install things system-wide**
  - It's actually a bit complicated to manage a fleet of identical servers
  - Chaos would ensue

How should you organize software?

Unified (single prefix)

```
user@brown-fe01 ~$ tree -lF -L1 /usr/local
/usr/local
├── bin/
├── etc/
├── include/
├── lib/
├── lib64/
├── libexec/
├── sbin/
├── share/
└── src/
```

Isolated (many prefix)

```
user@brown-fe01 ~$ tree -lF -L1 /apps/cent7
/apps/cent7
├── abaqus/
├── comsol/
├── gamess/
├── gaussian/
├── globus/
├── gurobi/
├── hyper-shell/
├── julia/
├── jupyterhub/
├── lammps/
├── matlab/
├── monitor/
...
```

How should you organize software?

Unified (single prefix)

```
user@brown-fe01 ~$ tree -lF -L1 /usr/local
/usr/local
├── bin/        <-- Programs
├── etc/        <-- Configuration
├── include/    <-- Headers
├── lib/        <-- Libraries (.so files)
├── lib64/
├── libexec/
├── sbin/
├── share/      <-- Manual pages
└── src/
```

Isolated (many prefix)

```
user@brown-fe01 ~$ tree -lF -L1 /apps/cent7
/apps/cent7
├── abaqus/
├── comsol/     <-- An entire installation prefix with only the assets specific to this application.
├── gamess/
├── gaussian/
├── globus/
├── gurobi/
├── hyper-shell/
├── julia/
├── jupyterhub/
├── lammps/
├── matlab/
├── monitor/
...
```

How should you expose software?

Your login configuration (e.g., `~/.bash_profile`) is sourced when you start a session.

```
user@brown-fe01 ~$ tail -4 ~/.bash_profile

# Install Software
export PATH="$HOME/apps/thing/2.1/bin:$PATH"
export LD_LIBRARY_PATH="$HOME/apps/thing/2.1/lib:$LD_LIBRARY_PATH"
```

Expose the **/bin** and **/lib** from the installation prefix.

How should you expose software?

Your login configuration (e.g. `~/.bash_profile`) is sourced when you start a session.

```
user@brown-

# Install S
export PATH
export LD_L
```

Maybe this is not the best approach:

- Forces all software (must always be compatible)
- Doesn't account for multiple versions
- Might conflict with other tools on occasion

How should you expose software?

- **Use the module system! (LMOD)**
  - Add directories with `` `module use <folder>` ``
  - Use directory structure to control access based on cluster.
- **Recommendations**
  - Manage a group-wide `/depot/<group>/etc/bashrc`
  - Do not load modules in your own `~/.bash_profile`

How should you expose software?

Modules are laid out in a simple hierarchy

```
user@brown-fe01 ~$ tree -lF $HOME/etc/module

├── X/
│   ├── 1.3.1.lua
│   ├── 2.1.0.lua
│   └── default → 2.1.0.lua
└── Y/
    ├── 3.2.0.lua
    └── default → 3.2.0.lua
```

How should you expose software?

Each file describes what environment variables to manipulate

```
user@brown-fe01 ~$ cat $HOME/etc/module/X/2.1.0.lua

whatis("Name: X")
whatis("Version: 2.1.0")
whatis("Description: X software that does interesting things")
prepend_path("PATH", "/home/user/apps/X/2.1.0/bin")
prepend_path("LD_LIBRARY_PATH", "/home/user/apps/X/2.1.0/lib")
prepend_path("MANPATH", "/home/user/apps/X/2.1.0/share/man")
```

How should you expose software?

Add the module tree to make it available

```
user@brown-fe01 ~$ module use $HOME/etc/module
user@brown-fe01 ~$ module avail X

---------------------------------- /home/<user>/etc/module -------------------------------------
    X/1.3.1        X/2.3.0 (D)

If the avail list is too long consider trying:

"module --default avail" or "ml -d av" to just list the default modules.
"module overview" or "ml ov" to display the number of modules for each name.

Use "module spider" to find all possible modules and extensions.
Use "module keyword key1 key2 ..." to search for all possible modules matching any of the "keys".
```

How should you expose software?

Load the software just like the core modules

```
user@brown-fe01 ~$ module load X
user@brown-fe01 ~$ x

...
```

1

Example 1

Just unpack the software anywhere and go.

Example 1

Unpack portable distribution anywhere you like

- **Easiest case is when the software is already built.**
  - Take the generic x86_64 Linux distributable.
  - Unpack anywhere and go.
- **Example**
  - *ncdu*

Example 1

DEMO

2

Example 2

Build software and install into some *prefix*.

Example 2

Build and install into some prefix

- **Load compilers**
  - Install with `--prefix=X`
- **Examples**
  - *GNU Parallel* (program)
  - *bzip2* (library)

Example 2

DEMO

3

Example 3

Build software *with dependencies* and install into some *prefix* - using the core modules.

Example 3

Build and install with dependencies into some prefix

- **Load compilers and core libraries**
  - E.g., gcc, openblas, openmpi, hdf5
  - Might use “configure” vs “cmake”
- **Examples**
  - *PETSc*

Example 3

DEMO

4

Questions

Discussion, more advanced scenarios?

Future Topics

Additional areas and more advanced topics

- **Software Installation**
  - Advanced / Integrated builds.
  - Python, R, Julia, MATLAB package management.
  - Spack (build tools)
  - Containers

Thank You

Please reach out to rcac-help@purdue.edu for questions.

See rcac.purdue.edu/training/software_installation_101 for slides.
See rcac.purdue.edu/training for additional training topics.
See rcac.purdue.edu/knowledge for guides.