Software Installation 101

Community Cluster Workshop Series

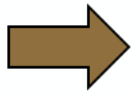
PURDUE UNIVERSITY | Rosen Center for Advanced Computing

Pre-Survey

Training Pre-Survey

Please fill out beforehand to let us know what you are starting from

Scan this!



https://purdue.ca1.qualtrics.com/jfe/form/SV_bBkYrTiNVW3net0?Q_CHL=qr

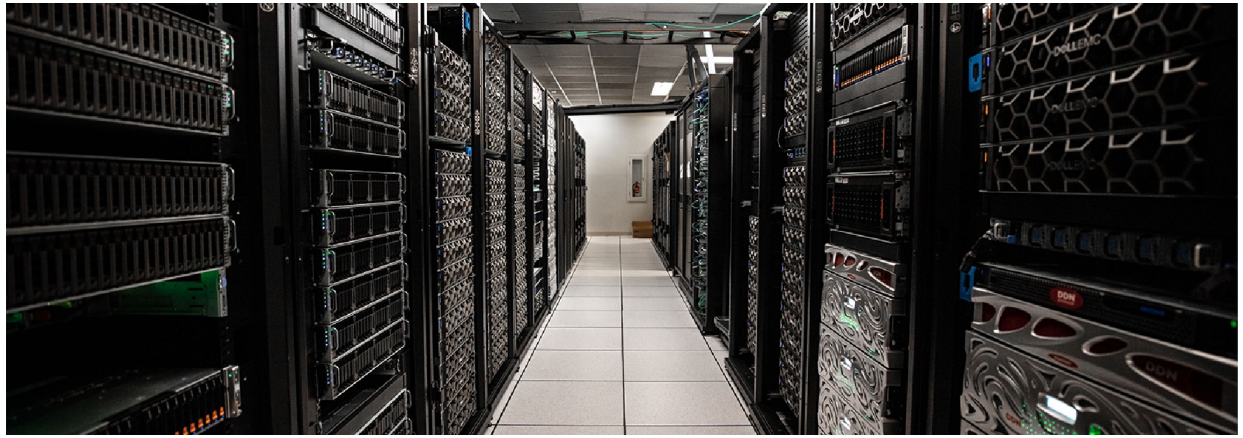
1

Preface

Introduction, expectations, and topics covered

Rosen Center for Advanced Computing

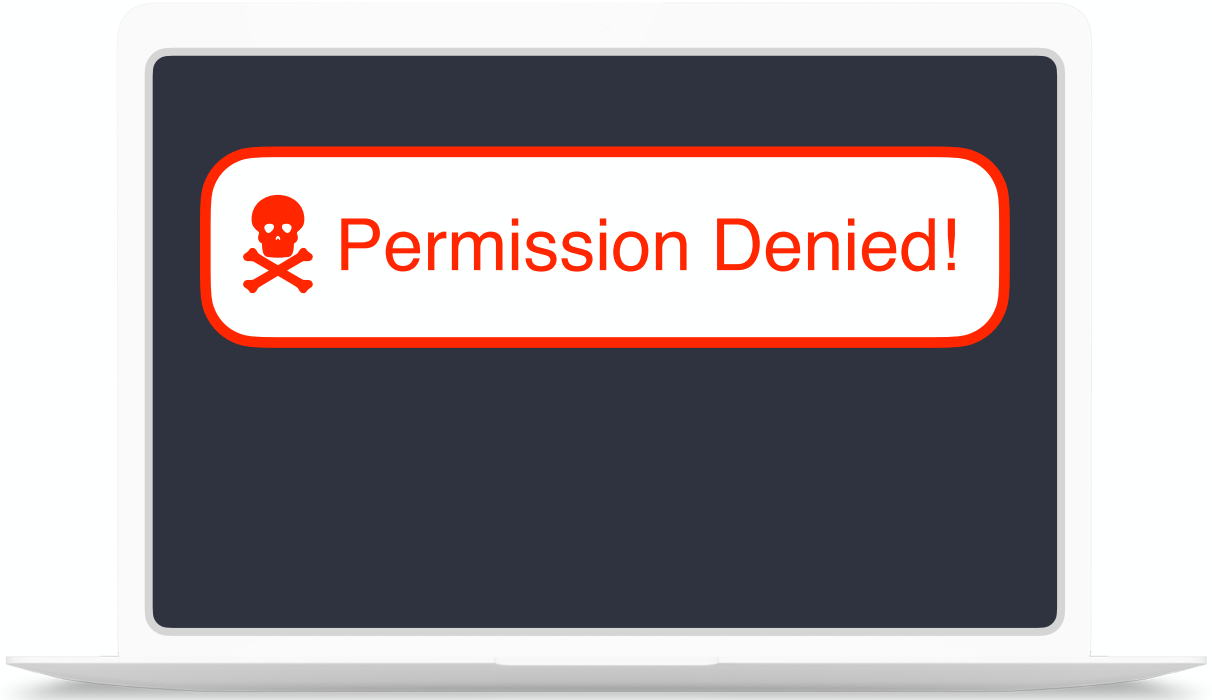
Research Computing and Data Services at Purdue University



- ***RCAC*** for short, part of *Purdue IT*
- Build and operate *Top500* supercomputers on campus, including petabyte-scale storage systems
- Scientific Applications (support and consulting)
- Envision Center (visualization and perception)
- Research Software Engineering (solutions)
- Visit rcac.purdue.edu to learn more

Prerequisites

Who is this for?



> Can you give me *sudo* access so I can install my software?”
>
> ~ a thousand users in the past decade

Prerequisites

Things we assume you know already

Linux shell and command-line basics

- UNIX 101 (rcac.purdue.edu/training/unix101)
- UNIX 102 (rcac.purdue.edu/training/unix102)

Cluster basics

- Clusters 101 (rcac.purdue.edu/training/clusters101)

Outline

List of topics covered in this training

➡ *Fundamentals and concepts*

- What does “installation” even mean?
- Where and how?
- Managing software environments

➡ *Examples*

- Simple distributable program (unpack and go)
- Simple build (compile and install)
- Intermediate library (compile with dependencies)

Topics Not Covered

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

201, 202, 203, 204

Package Installation

(Python, R, Julia, MATLAB, …)

301, 302, 303

Containers

(Docker, Singularity/Apptainer, SquashFS, …)

401, 402, 403, 404, 405

Advanced Topics in Linux

(Linux, ELF, x86/ARM, linking, MPI, CUDA/ROCM)

2

Fundamentals

Filesystems and environment variables
Programs vs Libraries
Managing your software environment

TLDR

The primary take-away message of this workshop!

- You can *install* **anything** you want into any directory that you have *write privileges* 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 computing system

Let’s get started with ***programs***

Fundamentals

What does it mean for a program to be installed?

Where is the program on the system?

```
user@login01.negishi:[~] $ myquota
Type     Location           Size    Limit    Use        Files    Limit     Use
=============================================================================
home     user             10.4GB   25.0GB    42%            -        -      -
scratch  negishi          70.8GB  200.0TB  0.03%         409k   2,000k    20%
depot    workshop        161.7GB    1.0TB    16%            -        -      -


user@login01.negishi:[~] $ which myquota
/usr/local/bin/myquota
```

Fundamentals

What does it mean for a program to be installed?

Where is the program on the system?

```
user@login01.negishi:[~] $ myquota
Type     Location            Size    Limit    Use        Files    Limit     Use
=============================================================================
home     user              10.4GB   25.0GB    42%            -        -      -
scratch  negishi           70.8GB  200.0TB  0.03%         409k   2,000k    20%
depot    workshop         161.7GB    1.0TB    16%            -        -      -


user@login01.negishi:[~] $ which myquota
/usr/local/bin/myquota
```

Directory on **$PATH**

/usr/local/bin/myquota

Fundamentals

Can any file be executed as a program?

Create a shell script and execute it

```
user@login01.negishi:[~] $ echo “seq 4” > my_program

user@login01.negishi:[~] $ my_program
-bash: ./myprogram: No such file or directory

user@login01.negishi:[~] $ ./my_program
-bash: permission denied: ./my_program

user@login01.negishi:[~] $ ls -lh my_program
-rw-r--r-- 1 user student 1 Mar  1 11:40 my_program
```

Fundamentals

Can any file be executed as a program?

Create a shell script and execute it

```
user@login01.negishi:[~] $ chmod +x my_program

user@login01.negishi:[~] $ ls -lh my_program
-rwxr-xr-x 1 user student 1 Mar  1 11:40 my_program

user@login01.negishi:[~] $ ./my_program
1
2
3
4
```

Fundamentals

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>-apps)

We cannot allow everyone to install things system-wide

- It's more complicated than just "`apt-get install xyz`"
- Chaos would ensure within minutes

Fundamentals

How should software be organized?

Unified environment (shared prefix)

```
user@login01.negishi:[~] $ tree -lF -L1 /usr/local/
/usr/local
├── bin/
├── etc/
├── include/
├── lib/
├── lib64/
├── libexec/
├── sbin/
├── share/
└── src/
```

Isolated environments (many prefixes)

```
user@login01.negishi:[~] $ tree -lF -L1 /apps/…
/apps/external/apps/
├── abaqus/
├── comsol/
├── gamess/
├── gaussian/
├── globus/
├── gurobi/
├── hyper-shell/
├── julia/
├── jupyterhub/
├── lammps/
├── matlab/
├── monitor/
...
```

Fundamentals

How should software be organized?

Unified environment (shared prefix)

```
user@login01.negishi:[~] $ 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 environments (many prefixes)

```
user@login01.negishi:[~] $ tree -lF -L1 /apps/…
/apps/external/apps/
├── abaqus/
├── comsol/     ← The single program contained isolated installation prefix
├── gamess/
├── gaussian/
├── globus/     With its own bin, lib, etc…
├── gurobi/
├── hyper-shell/
├── julia/
├── jupyterhub/
├── lammps/
├── matlab/
├── monitor/
…
```

Fundamentals

How should you expose software?

Your login configuration (E.g., ~/.bash_profile) is source when you start a session

Activate programs automatically using your shell profile

```
user@login01.negishi:[~] $ cat ~/.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

Fundamentals

How should you expose software?

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

Activate programs automatically using your shell profile

```
user@login01.negishi:[~] $ cat ~/.bash_profile

# Install Sof
export PATH="
export LD_LIB
```

Problematic Long-term Solution!

- Forces all software all of the time (cannot deactivate)
- Doesn't account for multiple versions
- Might conflict with other system tools

Fundamentals

How else can you manage software activation?

Use the module system (LMOD)

- Add directories with: **`module use <folder>`**
- Use directory structure to control access based on cluster (optional)

Recommendations

- Manage a group-wide shell profile at **`/depot/<group>/etc/bash_profile`**
- Do not load modules in your own **`~/.bash_profile`**

Fundamentals

Modules can be laid out in a simple hierarchy

Inspect module definition files

```
user@login01.negishi:[~] $ tree -lF ~/etc/module
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
```

Fundamentals

Module definitions are easy to write

Simple specification written in Lua

```
user@login01.negishi:[~] $ cat ~/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")
```

Fundamentals

Add the module tree to make it available

Expose private modules

```
user@login01.negishi:[~] $ module use ~/etc/module
user@login01.negishi:[~] $ module avail X

------------------------------- /home/user/etc/module -------------------------------------
    X/1.3.1        X/2.1.0 (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".
```

Fundamentals

Load the module to activate the program

Expose private modules

```
user@login01.negishi:[~] $ module load X
user@login01.negishi:[~] $ x

…
```

3

Demo A

Just unpack the software anywhere and go!

Demo A

Unpack portable distribution of pre-built software program

The easiest case is when the software is already built

- Download the generic ***x86_64*** Linux distributable binary
- Unpack anywhere and it's ready to execute

Example

- **duckdb** (https://duckdb.org):
  Fast, embedded database engine for analytical workloads

DEMO

3

Demo B

Compile software and install into some prefix

Demo B

Simple compile step with installation to chosen prefix

Load compilers, build, install

- `./configure --prefix=X`,
- `make`, and
- `make install`

Example

- **samtools** (github.com/samtools/samtools):
  Tools (written in C using htslib) for manipulating next-generation sequencing data

DEMO

3

Demo C

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

Demo C

Simple compile step with installation to chosen prefix

Load dependencies, build, install

- E.g., **`module load gcc openmpi openblas hdf5 fftw`**
- Might use **`./configure`** + **`make`**, or **`cmake`**

Example

- **PETSc** (petsc.org):

  Portable, Extensible Toolkit for Scientific Computation, pronounced PET-see (/ˈpɛt-siː/), is for the scalable (parallel) solution of scientific applications modeled by partial differential equations (PDEs).

DEMO

4

Questions

Discussion, clarification, questions, comments?

What's next?

Additional areas of exploration after the basics

Software Installation

- Build automation, archiving software as data, using Scratch
- **Spack** (spack.io):
  A flexible package manager for HPC environments;
  supporting multiple versions, configurations, platforms, and compilers.

Containers

- **Singularity / Apptainer** (apptainer.org):
  Encapsulate your complex software environment once, inside of a portable, hashable, image format that can run almost anywhere

Thank You

Slides and recording: rcac.purdue.edu/training/software_installation_101

Send an email to rcac-help@purdue.edu

Consulting hours: rcac.purdue.edu/coffee

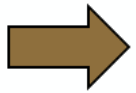
Post-Survey

PURDUE UNIVERSITY | Rosen Center for Advanced Computing

Training Post-Survey

Please fill out afterward to give us helpful feedback

Scan this!



https://purdue.ca1.qualtrics.com/jfe/form/SV_dnHn3aQoX7pLyrI?Q_CHL=qr