Installing Software and Writing Modules

Introduction

In the intro portion of the workshop you will learn:

- About downloading code
- About compiling code
- How to build a package from source code
  - configure
  - build
  - install
- How to write modules
- What are `yum`, `rpm`, `get-apt`, & `sudo`

Downloading Code

Source vs Executable

- In most cases you are better off downloading the source and building the code (aka the executable) yourself.
- Downloading executable is easier but is more likley not to work

Donwloading Executables

There are instances when available executables will run flawlessly on Hydra, but make sure that:

1. you can trust the origin,
2. sure you get a version compatible with Hydra,
    - *i.e.*, CentOS 7.x for Intel/AMD CPUs (`x86_64`)

Remember

- Hydra configuration is specific:
    - pre-built may code need *stuff* (dependencies) not on Hydra.

Notes on Downloading Executables

Risks

- Since users on Hydra do not have elevated privileges (root access) you are very unlikely to damage the cluster, but malicious software can still damage your files.
- In rare cases it may install a *trojan horse* that could exploit a known vulnerability.
  - Be vigilent and responsible.
  - In case of doubt, never hesitate to contact us.

Compiling code

Steps

Creating executable from source code is typically done as follows:

1. compile the source file(s) to produce object file(s),
2. link the object file(s) and libraries into an executable.

In Practice

- Often aided by a `makefile`,
- *Configuring* is creating such `makefile` or equivalent.

This will be illustrated in the hands on section.

Building from Source

Configure

- Most packages come with a configuration script, a list of pre-requisites (dependencies) and instructions,
- Some packages allow to build the code without some features in case you cannot satisfy some of the pre-requisites,
- You most likely need to load the right module(s) to use the appropriate compiler.

Build

- make sure you have loaded the right modules,
- run `make` to compile and link (aka build) the code.

Install

- copy the executable(s) to the right place(s),
  - usually defined by the configuration,
- best practice is to separate build from install locations.

Setting up Your Environment to Run Your Code

Likely Needed

You likely will need to adjust your *environment* to run some code:

1. the location of the code: `path` or `PATH`,
2. the location of the libraries: `LD_LIBRARY_PATH`,
3. you may need to also set some environment variables, etc.

Easier Way: modules

This is where using a module makes things easy:

- compact, and
- works with any shell.

Module and Module Files

The Command `module`

- convenient mechanism to configure your *environment*,
- reads a file, the *module file*, that holds instructions,
- a shell independent way to configure your environment:
  - *same* module file whether `sh/bash` or `csh/tcsh`.

Examples

- We provide module files, users can write their own.
  - look at all the module files we wrote,
  - they can be found in `/share/apps/modulefiles/`

Module File Syntax and Concepts

Special Instructions

- Instructions to configure your environment:

```
prepend-path PATH /location/of/the/code
setenv       BASE /scratch/demo
set-alias    crunch "crunch --with-that-option \*"
```

Syntax

- Module files can be complex, using `tcl` language
  - you **do not** need to know `tcl` to write simple module files.

Simple or Complex

- A simple module file can just list the modules that must be loaded to to run some analysis.
- Can write complex module files and leverage `tcl`.

Example of `module` Commands

Basic

	Info	Config	Details
`module`	`avail`	`load`	`list`
`module`	`whatis`	`unload`	`help <name>`
`module`	`whatis <name>`	`swap`	`show <name>`

More help

```
man module
```

Example of a Simple Module File

```
#%Module1.0
#
# load two modules and set the HEASOFT env variable
module load gcc/10.1.10
module load python/3.8
setenv HEASOFT /home/username/heasoft/6.3.1
```

Example of a More Elaborate Module File

rclone

```
#%Module1.0
#
# set some internal variables
set ver     1.53.1
set base    /scratch/hpc/haw/examples
#
# what to show for 'module whatis'
module-whatis "System paths to run rclone $ver"
#
# configure the PATH and the MANPATH
prepend-path PATH    $base/rclone/$ver
prepend-path MANPATH $base/rclone/$ver/man
```

Examples of Complex Module Files

Plenty of Examples

```
cd /share/apps/modulefiles

more intel/2022.2
more idl/8.8
more bio/blast2go/1.5.1
more bio/trinity/2.9.1
```

Module Files Organization

Where to Keep your Module Files

- in a central location using a tree stucture, or
- where you need them.

For Example

- You can load a module using the file full path:

```
module load /path/to/my/module/crunch
```

Recomended Approach

- use a central location under you home directory `~/modulefiles`,
- use a tree structure
- use version numbers if/when applicable,
- let `module` know where to find the module files.

Organization (cont'd) and Customization

For Example

```
~/modulefiles/crunch/
~/modulefiles/crunch/1.0
~/modulefiles/crunch/1.2
~/modulefiles/crunch/2.1
~/modulefiles/crunch/.version
~/modulefiles/viewit
```

Default Version

The file `.version` defines the default version:

```
#%Module1.0
set ModulesVersion "1.2"
```

Hence

```
module load crunch
module swap crunch/2.1
```

Customization (cont'd)

Let `module` Know Where to Find the Module Files

```
module use --append ~/modulefiles
```

Either

1. in your initialization file `~/.bashrc` or `~/.cshrc`
2. better yet in a `~/.modulerc` file

```
#%Module1.0
# adding my own module files
module use --append /home/username/modulefiles
```

The `yum`, `rpm`, `get-apt` and `sudo` Soup

Definitions

- `yum`: is a package-management utility for CentOS
- `rpm`: pre-built software package
  - *both* are for sys-admin,
  - help handle dependencies,
  - *yet* ...
- `get-apt`: Debian's version of `yum`, *does not work* on CentOS.

Also

- `sudo`: allows to run a command as 'root': **you can't!**

BTW

- Intructions that mention `yum`, `rpm`, `apt-get` or `sudo`
  - **will not work** on Hydra,
  - **yet** in most cases there is another way.

Hands-on Section

Hands-On

In the hands-on portion of the workshop you will

- Build and install software using best-practices,
  - trivial case,
  - simple/didactic example,
  - somehwat complex examples.
- Write simple and more elaborate module files.
- Run the software you installed in jobs.

But first, log in to Hydra

- If you need a reminder about how to log into Hydra and how to change your password, check the *Intro to Hydra* tutorial.
  - If the link does not work:

`https://github.com/SmithsonianWorkshops`
`> Hydra-introduction`
`> hydra_intro.md`

More to come . . .