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
  - simple, elaborate and complex examples (`tcl` aka *tickle*)
  - `.version` file
- What are `yum`, `rpm`, `get-apt`, etc
  - how to use `yum` and `rpm`
  - what about `sudo`?

Downloading Code

In most cases you are better off downloading the source and building the code (aka the executable) yourself.

Donwloading Executables

There are instances when available executables will run flawlessly on Hydra, but

1. make sure you trust the origin of the code
2. make sure you get a version compatible with Hydra, *i.e.*, that will run on CentOS 7.x for Intel/AMD CPUs
3. Hydra configuration is specific, hence pre-built code may need other stuff (dependencies) that are not installed on Hydra

Notes on downloading executables

- 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 try to install a *trojan horse* that would try to exploit a know vulnerability. So be vigilent and responsible.
- In case of doubt, never hesitate to contact us.

Compiling code

- Creating executable from source code is typically done as follows:

1. compile the source file(s) to produce object file(s) (`.o`),
2. link the object file(s) and libraries into an executable.
    - This is often aided by a `makefile`,
    - “Configuring” is creating such `makefile` or an equivalent.

- This will be illustrated in the hands on section.

Building from Source

If you download source code you will need to build the code.

Typically:

1. **Configure**
    - Most packages come with a configuration script, a list of pre-requisites (aka 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 to use the appropriate compiler
2. **Build**
    - need to make sure you have loaded the right modules to use the right compiler
    - run `make` to compile and link (aka build) the code
3. **Install**
    - copy the executable(s) to the right place(s) (usually defined by the configuration)
        - best practice is to separate build from install locations

- This will be illustrated in the hands on section.

Setting up your Environment to Run the Code

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

1. the location of the code, aka `path` or `PATH`
2. the location of the libraries needed, aka `LD_LIBRARY_PATH`
3. you ay need to also set some environment variables

This is where using a module makes things easy: compact and works with any shell.

Module and Module Files

- The command `module`
  - is a convenient mechanism to configure your Unix/Linux environment.
  - reads a file, aka the *module file*, that holds a set of simple or complex instructions.
- This is a shell syntax independent way to configure your environment:
  - you use the *same* module file whether you use `sh/bash` or `csh/tcsh`.
- We provide a set of module files, but users can augment this by writing their own.
  - you are welcome to look at all the module files we wrote, most of them are under `/share/apps/modulefiles/`.

Module File Syntax and Concepts

- Module files can be complex, written following the `tcl` scripting language, *although* you **do not** need to know that language to write simple module files.
- The `tcl` syntax is augmented by commands specific to help configure your environment:

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

- For example a simple module file can just hold a list of modules that must be loaded to to run a given tool.
- You can write complex module files and leverage the `tcl` syntax.

Example of `module` commands

- Informational

```
module avail
module whatis
module whatis <name>
```

- Configure your environment

```
module load   <name>
module unload <name>
module swap   <name>
```

- Specific info

```
module list
module help <name>
module 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/sylvain/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

```
cd /share/apps/modulefiles

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

Organization and Customization

- You can keep your module files in a central location using a tree stucture (:thumbsup:), or
  - if you prefer where you need them.
- You can load a module file using the module file full path,

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

- or tell `module` where to look for your central location (:thumbsup:).
- The recommended approach:
  - use a central location under you home directory `~/modulefiles`
  - use a tree structure and use version numbers if/when applicable

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

  - The `.version` file defines a default version

```
#%Module1.0
```

The yum, rpm, get-apt and sudo soup

Hands-on Section

In the hands-on portion of the workshop you will

- How to find software to install,
- How to install software using best-practices,
- How to run the software you installed in jobs.

Log in to Hydra

If you need a reminder about how to log into Hydra and how to change your password, check out our Intro to Hydra tutorial: https://github.com/SmithsonianWorkshops/Hydra-introduction/blob/master/hydra_intro.md