# 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 (`x86_64`)
3. Hydra configuration is specific:
   - pre-built may code need *stuff* (dependencies) not 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 install a *trojan horse* that could exploit a know vulnerability.
  - 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),
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.

1. **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.
2. **Build**
   - make sure you have loaded the right modules,
   - 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 Your Code

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 ay need to also set some environment variables, etc.

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.
- This is a shell independent way to configure your environment:
  - *same* module file whether `sh/bash` or `csh/tcsh`.
- We provide module files, users can write their own.
  - look at all the module files we wrote:

`/share/apps/modulefiles/`

## Module File Syntax and Concepts

- Instructions specific to configure your environment:

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

- Module files can be complex, using `tcl` language
  - you **do not** need to know `tcl` to write simple module files.
- A simple module file can just list the modules that must be loaded to to run asome analysis.
  - you can write complex module files and leverage `tcl`.

## Example of `module` Commands

| | 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/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
```

## Module Files Organization

- Keep your module files:
  - in a central location using a tree stucture, or
  - 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.
- The recommended approach:
  - use a central location under you home directory `~/modulefiles`,
  - use a tree structure and use version numbers if/when applicable.

## Organization (cont'd) and Customization

- Example:

  `~/modulefiles/crunch/`

  `~/modulefiles/crunch/1.0`

  `~/modulefiles/crunch/1.2`

  `~/modulefiles/crunch/2.1`

  `~/modulefiles/crunch/.version`

  `~/modulefiles/viewit`

- the file `.version` file defines a default version:

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

## Customization (cont'd)

- to tell `module` where to find your module files:

```
module use --append /home/sylvain/modulefiles
```

- that instruction can be either:

1. in your initialization file `~/.bashrc` or `~/.cshrc`
2. better yet in a `~/.modulerc` file (shell independent):

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

## The `yum`, `rpm`, `get-apt` and `sudo` soup

- `yum` (Yellowdog Updater, Modifier) is a package-management utility
  - reserved for admin, handle dependencies, *but...*
- `rpm` (Red-hat Package Manager) pre-built software package
  - reserved for admin, *but...*
- `get-apt` Debian's version of `yum`: does not work with CentOS
- `sudo` allows to run a command as 'root': you can't..

Intructions that refer to `yum`, `rpm`, `apt-get` or `sudo` will not work for you on Hydra.

*In a lot of cases there is an other way to do the same.*

# 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