HPC Creating Custom Module | Physical Research Laboratory

You may wish to customize the Linux environment for your jobs on your own. You can create your own custom module to accomplish this. First, create a directory for your modules:

$ mkdir ~/my_modules

Then, add the module path to the end of your `~/.bashrc` file:

export MODULEPATH=${MODULEPATH}:${HOME}/my_modules

A module can be as simple as setting a few environment variables (such as PATH and LD_LIBRARY_PATH) or can be complicated Tcl scripts. You can refer to any of our existing modules in the `/shared/modulefiles/` directory to create templates for your own. For example, the `/shared/modulefiles/gcc4.9.2` module includes the following directives:

```
#%Module 1.0
###
# Module for gcc4.9.2

        set     topdir          /shared/GCC
        set     version         4.9.2

        setenv          CC              $topdir/bin/gcc
        setenv          GCC             $topdir/bin/gcc
        setenv          FC              $topdir/bin/gfortran
        setenv          F77             $topdir/bin/gfortran
        setenv          F90             $topdir/bin/gfortran
        prepend-path    PATH            $topdir/include
        prepend-path    PATH            $topdir/bin
        prepend-path    MANPATH         $topdir/man
        prepend-path    LD_LIBRARY_PATH $topdir/lib
```

Writing Modulefiles

A modulefile must start off with a shebang-like construct:

#%Module1.0

The version number (1.0) in this example is not necessary, but is considered helpful. Note that it is the modulefile version, not the environment-modules version. Since we're still on the first version of the modulefile format, 1.0 is used on all our modules. After this initial construct, a modulefile is

a series of executable statements. Comments in modulefiles are introduced by # signs. Particularly useful statements in modulefiles are:

setenv — set the specified environment variable to the supplied value.

unsetenv — unset the specified environment variable. If an argument is supplied, then, while unloading the module, the variable will be re-set, to that argument.

append-path — put the supplied argument on the end of the specified variable. The variable should be a list of colon-separated entries. PATH is one such variable.

prepend-path — put the supplied argument at the start of the specified variable.

remove-path — remove the supplied argument from the specified variable, wherever along its length it might be.

prereq — insist that the specified module be loaded before loading the current module. (Note: Usually, it's easier to just put a "module load" in to get the dependency.)

conflict — insist that the specified module not be loaded.

module load — load the specified module.

module-whatis — follow with a help string, to be printed whenever the user issues the "module whatis" command on this module. It should briefly describe the software loaded by this module.

For a complete reference of module file directives, refer to the modulefile man page (`man modulefile`).

Previous | Next