Environment Modules:

Do what I say not what I do!

Kent Mein: mein@umn.edu
University of Minnesota 2013
Introduction

• ENVIRONMENT modules vs modules?
• Who is currently using modules?
• Who has written a module?
• Who doesn’t know what modules are?
• Planning…
What are modules?


• `module load (module)`  # Use a module

• `module unload (module)`  # Unload a module you have loaded

• `module avail`  # List all available modules

• `module list`  # List modules you have loaded

• `module show (module)`  # Show what a module will do

• `module help (module)`  # Show the help for the specified module
tcl only vs c version

Individual modules are written in tcl. (Mention the security feature.)

- started in 2002
- Smaller feature set
- Simpler to setup/maintain
- Local support
- Documentation and test suite not as extensive
- One file to look at

- Much older
- Larger feature set
- Need to do more planning for setup
- Need to compile for each platform
- More people using it
- More complex
Example module:
soft/gcc/4.7.2

#%Module
##      GNU Module
proc ModulesHelp { } {
    puts stderr "The GCC module contains gcc, g++, and f77, popular compilers for C, C++, and Fortran.\n    Version 4.7.2
    Website: http://gcc.gnu.org/"
}

set sys [uname sysname]
set os [uname release]
set arch [uname machine]
append-path MANPATH /soft/gcc-4.7.2/man

switch -glob $sys {
    SunOS* {
        switch -glob $os {
            5.10 {
                if { [string compare $arch "i86pc"] == 0} {
                    append-path PATH /soft/gcc-4.7.2/SunOS5.10x86/bin
                } else {
                    append-path PATH /soft/gcc-4.7.2/SunOS5.10/bin
                }
            } else {
                default {
                    append-path PATH /soft/gcc-4.7.2/SunOS5.8/bin
                }
            }
        }
    }
    Linux* {
        switch -glob $os {
            default {
                if { [string compare $arch "x86_64"] == 0} {
                    prepend-path PATH /soft/gcc-4.7.2/ubuntuamd2010/bin
                } else {
                    prepend-path PATH /soft/gcc-4.7.2/ubuntu1/bin
                }
            }
        }
    }
}

conflict ecad/vcs/2013
Tips for writing your own module

• Keep it simple

• Prepend-path vs append-path

• setenv vs prepend-path

• Make it future compatible

• Avoid modifying LD_LIBRARY_PATH and similar things

• Write a module help function

• Try to avoid conflicts
Writing your first module

- module use ~/.modules
- copy an existing module to start with.
- Write a ModulesHelp function
- Test it on all of your platforms before putting it into production.
- use a .version file
- If you use a switch make a default and make it future proof.
Our setup

- **MODULES_ROOT**: -> rdist to clusters, clusters automount the directory.
- **CSELabs**: ubuntu64, ubuntu32, SunOS5.10, SunOS5.10X86
- **CS**: In reality we have 3-4 versions of the OS on the various platforms and more than what is listed above…
- **DTC**: Only ubuntu64
The details

- MODULES_ROOT: cs:/project/rdist/config/opt/modules
- Git repository: https://github.umn.edu/mein/modules
- CS MODULES: /soft/rko-modules
- CS INITDIR: /soft/rko-modules/init
- CS MODULESFILES_DIR: /soft/rko-modules/modulefiles
- development MODULESFILES_DIR: ~mein/.modules
- development MODULES: ~mein/modules
- development INITDIR: ~mein/modules/init
### Module avail

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Version</th>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

### /soft/rko-modules/modulefiles

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Version</th>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

### /home/staff01/mein/.modules

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Version</th>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Automount tips/tricks

For languages like Perl that need a platform independent path:
mkdir /export/soft/perl5.14.1/$OSNAME$OSREL for each platform

auto.soft entries:
perl5.14.1 soft.cs.umn.edu:/export/soft/perl5.14.1

Then in /soft/perl.5.14.1 ln -s /soft/perl5.14.1/-bin bin

Inside of your modules do something like this:
if { [file isdirectory /soft/perl5.14.1/SunOS5.10] } { ### NOTE ###
    prepend-path PATH /soft/perl5.14.1/bin
    prepend-path MANPATH /soft/perl5.14.1/man
}
How do I distribute a new module?

module avail soft/gcc
soft/gcc/3.0  soft/gcc/4.0  soft/gcc/4.5.2 (default)
soft/gcc/3.2  soft/gcc/4.1  soft/gcc/4.7.2
soft/gcc/3.3  soft/gcc/4.2
soft/gcc/3.4  soft/gcc/4.3

cat .version
#%Module1.0
##
set ModulesVersion "4.5.2"

Create it in my development space ~/.modules
Test it on all platforms
Copy it over to MODULES ROOT
Check it into revision control
Rdist it out to clients
Test it again!
How do I retire an old module?

Retire an old module:
#%Module###########################################################
# gcc 3.0 module
set sys  [uname sysname]
set os   [uname release]
set arch [uname machine]
!
puts stderr "The soft/gcc/3.0 module is outdated and will be going away\n"
puts stderr "please update your .files to use the soft/gcc module instead.\n"
puts stderr "For questions or comments please email: operator@csealabs.umn.edu\n"
!
append-path MANPATH /soft/gcc-3.1/man
switch -glob $sys {
  IRIX* {

Test it in my development ~/.modules
Copy it over to MODULES ROOT
Check it in to Revision control.
Rdist it out to the clients.
Test it again.
How do users init modules?

- **.cshrc:**
  
  ```
  source /soft/rko-modules/tcl/init/tcsh
  module load soft/gcc java perl gnu local compilers system
  module load openwin math/mathematica scheme user
  ```

- **.bashrc:**
  
  ```
  ./soft/rko-modules/tcl/init/bash
  module load soft/gcc java perl gnu local compilers system
  module load openwin math/mathematica scheme user
  ```

- **.bash_profile:**
  
  ```
  [[ -f ~/.bashrc ]] && . ~/.bashrc
  ```
TCSH VS BASH

• TCSH module avail goes to stderr

• BASH has tab completion

• BASH has different ways of initializing the shell depending on if its an interactive shell or not.

• The more shells you support the more work for you, and it makes it harder to debug issues.
Extra stuff you need/want!

- ~template
- resetenv
- webpage explaining how modules work
- We have a webpage that lists available modules and when you click the module it displays the help info.
How do I upgrade the modules system?

- ~mein/modules

  - It has ~mein/modules/init which points to ~mein/modules and has a modulesrc file that just points to ~mein/.modules

- ~mein/.cshrc

  ```
  #set MODULESINIT="/home/staff01/mein/modules/init/tcsh"
  set MODULESINIT="/soft/rko-modules/tcl/init/tcsh"
  ```

Once I’m happy push the new version out to clients and reset my .cshrc and test on all platforms.
What do you need to do?

• Layout your modules directory (bin, init, man, modulefiles) and plan for how you will upgrade it. If you have multiple arch’s and your not using tcl version this is complicated.

• Put modulefiles directory under revision control

• Come up with a basic set of modules and groups for modules inside of the modulefiles directory.

• Scheme for testing new modules.

• Scheme for retiring old and deploying new modules.

• Figure out how you want users to init modules. (resetenv, shells supported etc)

• Plan for users to get more info on how modules work, init, resetenv, etc..