

The Modules System

9/7/98

1 Introduction

The Modules system enables easy environment setup and integration of course and project settings into any account, in a modular and organized fashion. The system consists of a few simple commands that allow one to make variable, path, alias and other environment settings in a well defined, shell independent manner. By using the modules system, one can easily add or remove modules, containing sets of environment settings.

2 How the System Works

The system works in two parts: the support programs and the modules themselves. When you log into your account, the module support code is invoked, which in turn loads a series of modules into your environment. This process is transparent to the user, and has the same end effect as making environment settings in a standard manner. Once logged in, however, you can view which modules are installed, add or remove settings dynamically, and ask for help on using the system.

3 What is a Module?

A module is simply a text file (really a very simple tcl script) that makes modifications to your environment. It may set variables, paths or aliases. Instead of placing most environment settings in your shell's resource file or or a .alias file, all settings can be placed in these modules. Also, you can include any number of modules, so instead of one large file with all your settings, one's setup can be moved into separate files, which helps to organize your environment resource files. Finally, courses or project groups can set up modules that make the simple environment changes they need to function, and anyone can include or remove these changes to their account by simply by adding or removing that module from their account. This will be explained in a minute.

4 How to Use the System

Setting up your account to use the modules system is extremely easy. It requires only a few additional lines in your `.cshrc` (or appropriate file depending on which shell you use). The following lines should be added to the start of the file (note that if this is confusing, you can always look at the default dotfiles in `/u/system/skel/user`):

```
# if you're using something like ksh or sh, set up the
# SHELLNAME variable in the correct manner
if ($?tcsh) then
setenv SHELLNAME 'tcsh'
else
setenv SHELLNAME 'csh'
endif

setenv MODULESHOME /opt/modules
source $MODULESHOME/init/$SHELLNAME

module use /opt/modules/lib
module use /opt/modules/lib/courses

module load osdetect
module load browncs.$ARCH cs.$ARCH news openwin
```

and that's it. It is also possible to make Xdefault settings using the modules system in a similar manner in you `.xinitrc` file.

5 A Walkthrough of the Included Lines

OK, so what was all that? Here it is a few lines at a time:

```
if ($?tcsh) then
setenv SHELLNAME 'tcsh'
else
setenv SHELLNAME 'csh'
endif
```

This is, for `csh` or `tsch`, a way to set the `SHELLNAME` environment variable. This is the only shell-dependent line in the setup. Once the shell has been determined, the modules system can load the appropriate files, and your subsequent files can be in a generic format that will be correct regardless of which shell you're using. Next, the `SHELLNAME` variable is used.

```
setenv MODULESHOME /opt/modules
source $MODULESHOME/init/$SHELLNAME
```

These two lines actually work to load the modules system. The second line uses the `MODULESHOME` variable to find the `Init` file, and then uses the `SHELLNAME` variable to pick the correct `init` file for your shell (supported shells are `bash`, `csh`, `ksh`, `perl`, `sh`, `tcsh` and `zsh`). When this file is loaded, a new function is added to your shell called `module`. Once you've logged in, try typing `module` in a shell. What are you doing is invoking a new shell function.

```
module use /opt/modules/lib
module use /opt/modules/lib/courses
```

These lines tell the modules system where to look for modules. If you want to include a new module, you'll need to make sure that the location of the module is included in the `module use` locations. The "courses" line lets your account load any modules that individual courses have made available.

```
module load osdetect
module load brownncs.$ARCH cs.$ARCH news openwin
```

Finally, these are the lines that actually load the modules. After the `module load` appears a list of modules to include in your environment. The first line, which includes the `osdetect` module sets up some variables based on the operating system and hardware you are running on. This info is then used in the second line to load appropriate files.

The default set of modules incrementally loads most of the path, `manpath` and environment settings that comprise the default course dotfiles. You should take some time to look in them, and see what they are actually setting up (remember, they are going to be found in one of the directories that was specified in the `use` lines).

6 Personal Environment Settings

If you look in the default `.cshrc` (`/u/system/skel/user/.cshrc`), you'll see that there's an additional `use` directive, and an additional module at the end of the list.

```
module use /opt/modules/lib
module use /opt/modules/lib/courses $HOME/.modules

module load osdetect
module load brownncs.$ARCH cs.$ARCH news openwin personal.mdl
```

This is set up to use a personal module in addition to the standard system modules. In this you could put any changes that you wanted to make to your environment (the default personal module is in `/u/system/skel/user/.modules/personal.mdl`).

7 Finally

That's really all there is. Look at the personal module as well as the various system modules to see how to write a module. The `module` command will allow you to include modules, remove them, list which you're using, etc. You can freely add or remove modules from your shell resource file. You can move between different shells, and the modules will work in the same way. The one thing to be careful of is not overwriting settings that previous modules have made. That is, by the time you load your personal module, you have already loaded several modules which may have made additions to your path. So, in your personal module, it is not a good idea to reset your path, but to instead append to it (see the sample personal module for an explanation of this).

By typing `module help` in a shell, you get a list of all module commands you can invoke. Some of these are `initadd` (to add a module to your startups), `initremove` (to remove the module from your startups), `list` (to list all the modules you're using), `help <module name>` (to get help on some module) and `display` to view what some particular module is adding to your environment.

For further help, you can man both `module(1)` and `modulefile(4)`. `module` explains the module system in more depth than does this document, and `modulefile` talks about the syntax for module files. Also, at <http://www.modules.org> there is a more in-depth description of the modules system. Finally, you can mail the Meta-TA with any questions that you have about the system.