Shell, Environment, and Modules

Overview

This page describes how to setup an initial shell and startup environment for computing, as well as the tools involved. These recommendations are based on the problems we've seen with new users in the past manipulating their shell init files and paths by hand, and are intended to provide a clean and direct approach for users to get started computing.

The Shell

As most if not all of the HPC resources are LINUX-based, we strongly recommend the use of Bourne Again Shell (BASH) for interactive and scripting use. This is installed on all LINUX and newer UNIX systems, and is the default shell on LINUX. It is tops for convenience, features, and standards compliance, and is the pre-eminent shell in use. For interactive use we will also support the TCSH, an updated version on the Berkely CSH, but in a more limited fashion than BASH, but not for scripting. We do not support the csh for interactive or scripting use.

Changing your shell:

Currently end users can not change the shell on the command line. Please contact help@seas.harvard.edu, and the sysadmins will do the change for you.

To change your shell from the default (which in the past has been the "csh"), use the "chsh" (or "ypchsh" on systems which use NIS) command to change your shell.

On SEAS systems, login to login.seas.harvard.edu and execute this from the command line:

```
username@login > ypchsh
Changing NIS account information for username on suite.
Please enter password:
Changing login shell for username on suite.
To accept the default, simply press return. To use the system's default shell, type the word "none".
Login shell [/bin/bash]: /bin/bash
The login shell has been changed on suite.
```

Shell Init Files

Hear we describe some basics for setting up simple shell init scripts under bash. Since the default system environment is usually customized to the specific system you’ve logged into, it’s important in general is that the default path on the system is maintained, and not overwritten by the shell init scripts.

Again IMPORTANT: don’t overwrite the default path in your shell's init scripts.

In general the user's startup scripts, which are placed in the user's home directory, function as follows:

- `~/.bash_profile`: sourced whenever the user logs in to an interactive shell session
- `~/.bashrc`: sourced every time bash is started, even in shell scripts

Here are some basic, clean init scripts for use in an HPC environment:
With this set of init scripts, your environment will be mostly based on the default system environment, and will be easily augmented by the modules package.

For info on editing and modifying your BASH prompt, take a look at the BASH Prompt HOWTO.
Using Modules to Manage Your Environment

To reduce the complexity of trying to manage various paths and environment variables on multiple systems, we have adopted the modules package. This set of commands allows one to add and remove sets of environment variables and paths from your shell's environment from a simple command line interface. It is the role of the system administrator to configure "modulefiles" that define the paths and environment variables needed to use an application, compiler, or library. The user adds or removes these configurations to their environment as needed, without needing to know the paths themselves.

The modules package will be made available for most shells when you login by the system shell init scripts. So when you login, you will find a minimal environment setup. This is by design so that users can choose sets of configuration as needed for their apps.

To begin, you can list the available modules by executing the command:

```bash
> module avail
```

The available modules are listed, some with a format like directories (for example "compilers/intel/9.1" activates the Intel v9.1 compilers.)

If you override the system paths in your shell init scripts, modules will likely not be available until you rectify this.

Runtime Usage

To enable a particular module in your current shell environment, you can use the "load" subcommand:

```bash
> module load [a module name]
```

This will overlay the environment for this module on the current environment, often prepending paths to $PATH environment variables such as $PATH and $LD_LIBRARY_PATH.

To remove this module, use the "unload" subcommand:

```bash
> module unload [a module name]
```

This will cleanly remove those added PATHs and environment variables, returning to the original environment.

To list your current environment, use the "list" subcommand:

```bash
> module list
```

this will list the currently loaded modules.

All the changes from these commands are runtime only, and will not affect the next time you start a shell.

Usage for Init Scripts

The previous changes will allow you to manage your environment in the current shell environment, but will not affect other shells, or new shells. Thus running something like mpirun may not capture these settings, since the new shells spawned by mpirun may not get the same environment as the original shell.

To manage your default environment through module, begin by adding to the end of your ~/.bashrc or ~/.cshrc startup script the line

```
```

All the changes from these commands are runtime only, and will not affect the next time you start a shell.
(Note that the init scripts above have this included, with a test for an installed module environment). This line is a place holder for loading other modules; the "null" module actually does nothing in and of itself to your environment.

Next, to add a module to your default environment, use the command

```
module initadd [module name]
```

This will alter the "module load" line in your startup script to load this module at each shell invocation. In this way you can manage your default environment through the modules package without editing your startup scripts.

To disable a default module, use "initrm"

```
module initrm [module name]
```

This removes that module from the module line in your startup script.

**Other Options**

To learn more about the module command, run

```
module --help
```

or

```
man module
```

You can also create your own modules to manage personal paths and environments. Modules are simply snippets of TCL code placed in a certain directory. For more information, look to the system modules installed in `/etc/modulefiles` and the module file man page

```
man modulefile
```

With the above `.bashrc`, you can place your own modulefiles into `~/.modulefiles`.

**More Info**

For more information on the modules package, see the package [website](#).