Comet to Expanse Transition Tutorial:
Environment Modules, Code Compilation and Optimization

Marty Kandes, Ph.D.

High-Performance Computing User Services Group
San Diego Supercomputer Center
University of California, San Diego

Thursday, October 29th, 2020
12:25PM - 1:25PM PDT

What is a Shell?

- A shell is a program that provides the traditional, text-only user interface for Linux and other Unix-like operating systems. Its primary function is to read in commands provided by a user, process those commands, and then executes them on the computer being run by the operating system.
- A shell is the fundamental interface between a user and a computer's operating system, which allows the user to interact with the computer and its operating system, while hiding all of the underlying details.
- A user is running a shell whenever they log into a system with their username and password.

Common Linux Shells

- sh: the orignal Unix shell
- bash: a Linux shell written by the GNU Project; default shell program for many of the most popular Linux distributions

```
[mkandes@login02 ~]$ hostname --fqdn
login02.expanse.sdsc.edu
[mkandes@login02 ~]$ echo $SHELL
/bin/bash
[mkandes@login02 ~]$
```

- csh/tcsh: a Linux shell modeled after C programming language.

* If you would like your default shell on Expanse to be something other than bash, then please contact us via the ticketing system with a change request.

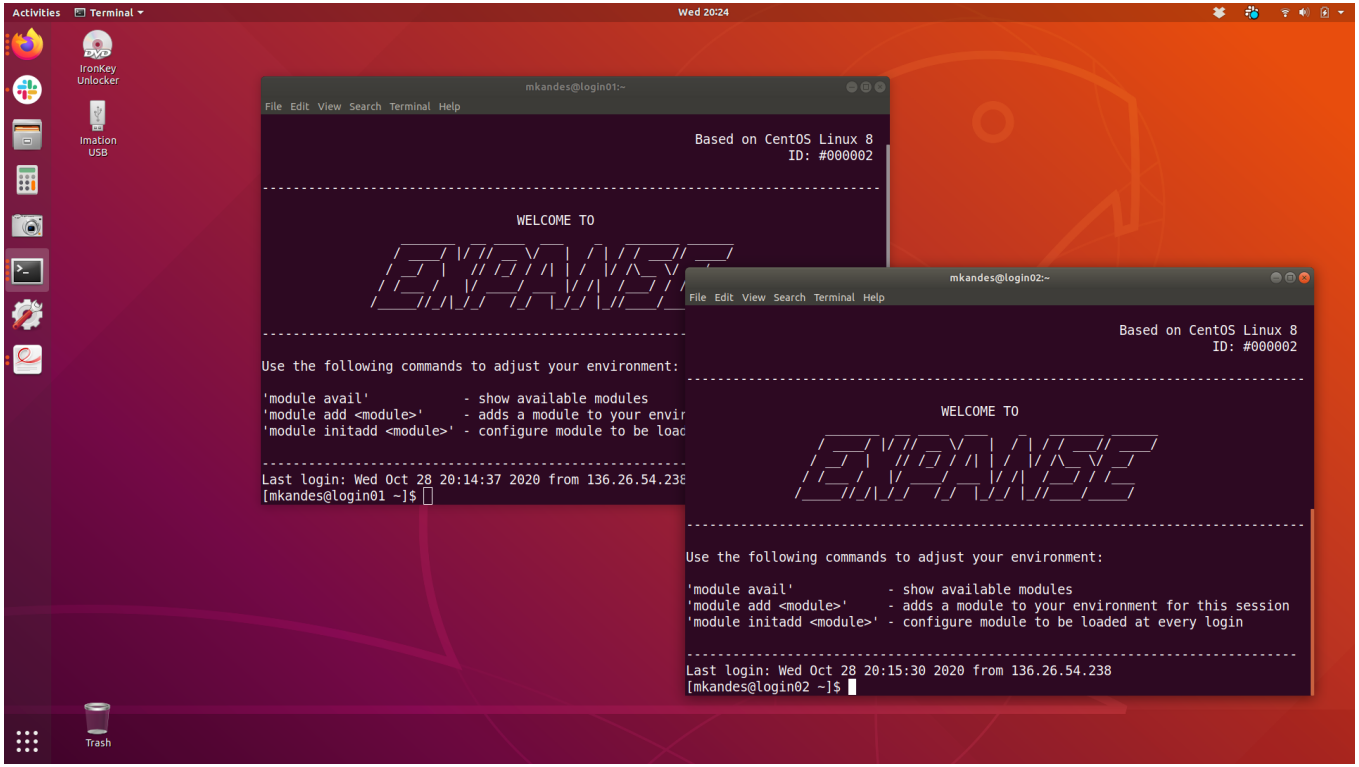
Shell Builtins

- A shell builtin is a command or function that is executed directly in the shell itself, instead of an external executable program which the shell would load and execute.

```
[mkandes@login02 ~]$ help cd
cd: cd [-L|[-P [-e]] [-@]] [dir]
    Change the shell working directory.

    Change the current directory to DIR.  The default
    DIR is the value of the
    HOME shell variable.
```

Shell Sessions



- A shell session is a running instance of a shell program.

Shell Variables

- A shell variable is a character string to which a value may be assigned a value.
- The value assigned to a shell variable could be a number, text, filename, device, or any other type of data.
- Most importantly, a shell variable is available only to the shell session in which it was defined and it is not inherited by child processes spawned by the shell.
- Shell variables are typically used to store temporary data local to a particular shell that may change frequently.

```
[mkandes@login01 ~]$ echo $PWD
/home/mkandes
[mkandes@login01 ~]$ cd /expanse/lustre/scratch/mkandes
[mkandes@login01 mkandes]$ echo $PWD
/expanse/lustre/scratch/mkandes
[mkandes@login01 mkandes]$
```

Environment Variables

- Environment variables are shell variables that have been exported to the shell's *environment* such that they may be inherited by any child processes spawned by the shell.
- Environment variables are used to define how a shell responds to user commands and how those commands are run when executed.
- Environment variables are typically used to store data that needs to be more persistent and utilized by other commands and processes run by the shell.

```
[mkandes@login01 ~]$ echo $HOME
/home/mkandes
[mkandes@login01 ~]$
```

Shell Initialization

- When you log in to a system, the shell undergoes a phase called initialization to set up the shell's environment.
- Shell initialization is usually a two-step process that involves the shell reading a set of shell configration files that will determine the state of the user's environment at startup.

Bash Shell Initialization Files

- `/etc/profile`: This is the system-wide initialization file executed during login. It usually contains environment variables, including an initial PATH, and startup programs.
- `/etc/bashrc`: This is another system-wide initialization file that may be executed by a users .bashrc for each bash shell launched. It usually contains functions and aliases.
- `$HOME/.bash_profile`: If this file exists, it is executed automatically after /etc/profile during login.
- `$HOME/.bash_login`: If `.bash_profile` doesnt exist, this file is executed automatically during login.

Bash Shell Initialization Files

- `$HOME/.profile`: If neither `.bash_profile` nor `.bash_login` exists, this file is executed automatically during login. This is the default in Debian-based distributions, such as Ubuntu. Note that this is the original bash configuration file.
- `$HOME/.bashrc`: This file is executed when a non-interactive bash shell starts. e.g., *when running a batch job script on Expanse*. This file is often referred to in the bash interactive scripts, such as `.bash_profile`.
- `$HOME/.bash_logout`: This file is executed automatically during logout.

Login vs. Non-Login Shells, Interactive vs. Non-Interactive Shells

- A *login* shell is the first process created and run when you log into a system via SSHi with your username and password, SSH keys, etc. When the login shells is run, it will read `/etc/profile` and `$HOME/.bash_profile` to configure the shell environment.
- A *non-login* shell is a one executed without having to login and authenticate with a system. e.g., When you open up a GUI-based terminal application on your desktop, you have already logged into the system. The new shell session opened in the terminal window is a non-login shell. When a non-login shell is run, it will read `/etc/bashrc` and `$HOME/.bashrc` to configrue the shell environment.

Interactive vs. Non-Interactive Shells

- An *interactive* shell, which can be either a login or non-login shell, is one where you can interactively type or interrupt commands issued to the shell.
- A `non-interactive` shell is one that is usually run by an automated process. e.g., **A shell running a script is always a non-interactive shell.** Most non-interactive shells are also non-login shells as the user calling the script has typically already logged into the system where the script will be run.

What's your `PATH`?

- `PATH` is an environmental variable in Linux and other Unix-like operating systems that tells the shell which directories to search for executable files in response to commands issued by a user.

- A user's `PATH` consists of a series of colon-separated absolute paths that are stored in plain text files. Whenever a user enters a command at the command line prompt that is not a shell builtin or does not include its absolute path, the shell then searches through the `PATH` directories until it either finds an executable file with that name or it does not.

- The use of the `PATH` variable to find executable files eliminates the need for users to remember which directories they are in and to type their absolute path names. That is, any such program can be run by merely typing its name, such as `ls` instead of `/bin/ls`, regardless of where the user is currently working on the filesystem.

LD_LIBRARY_PATH

- LD_LIBRARY_PATH is an environmental variable which sets the path which the linker should look in to while linking dynamic, shared libraries.
- LD_LIBRARY_PATH contains a colon separated list of absolute paths the linker uses to prioritize its search over the standard library paths /lib and /usr/lib. The standard paths will still be searched, but only after the list of paths in LD_LIBRARY_PATH has been exhausted.
- The best way to use LD_LIBRARY_PATH is to set it on the command line or script immediately before executing the program. This way the new LD_LIBRARY_PATH isolated from the rest of your system.

Environment Modules

The Environment Modules system is a tool to help users manage their Unix or Linux shell environment, by allowing groups of related environment-variable settings to be made or removed dynamically.

http://modules.sourceforge.net

Common `module` commands

- `module list`
- `module avail <package>`
- `module purge`
- `module load <package>`
- `module unload <package>`
- `module show <package>`

module list

```
[mkandes@login01 ~]$ module list

Currently Loaded Modules:
  1) shared    2) cpu/1.0    3) DefaultModules    4) slurm/
    expanse/20.02.3

[mkandes@login01 ~]$
```

module avail <package>

```
[mkandes@login02 ~]$ module avail gcc

---------- /cm/shared/apps/spack/cpu/lmod/linux-centos8-
    x86_64/Core ----------
   gcc/7.5.0    gcc/9.2.0    gcc/10.2.0 (D)

--------------------------- /cm/local/modulefiles
    ---------------------------
   gcc/9.2.0

--------------------------- /cm/shared/modulefiles
    ---------------------------
   netcdf/gcc/64/gcc/64/4.7.3    openmpi/gcc/64/1.10.7

  Where:
   D:  Default Module

Module defaults are chosen based on Find First Rules due to
    Name/Version/Version modules found in the module tree.
See https://lmod.readthedocs.io/en/latest/060_locating.html
    for details.

Use "module spider" to find all possible modules and
    extensions.
Use "module keyword key1 key2 ..." to search for all
    possible modules matching
```

module purge

```
[mkandes@login01 ~]$ module list

Currently Loaded Modules:
  1) shared   2) cpu/1.0   3) DefaultModules

[mkandes@login01 ~]$ module purge
[mkandes@login01 ~]$ module list
No modules loaded
[mkandes@login01 ~]$
```

module load <package>

```
[mkandes@login02 ~]$ module load gcc/10.2.0
[mkandes@login02 ~]$ module load openmpi/4.0.4
[mkandes@login02 ~]$ module list

Currently Loaded Modules:
  1) shared   2) cpu/1.0   3) DefaultModules   4) gcc/10.2.0
       5) openmpi/4.0.4

[mkandes@login02 ~]$
```

module unload <package>

```
[mkandes@login02 ~]$ module list

Currently Loaded Modules:
  1) shared     3) DefaultModules    5) openmpi/4.0.4
  2) cpu/1.0    4) gcc/10.2.0

[mkandes@login02 ~]$ module unload openmpi
[mkandes@login02 ~]$ module load intel-mpi
[mkandes@login02 ~]$ module list

Currently Loaded Modules:
  1) shared     3) DefaultModules    5) intel-mpi/2019.8.254
  2) cpu/1.0    4) gcc/10.2.0

[mkandes@login02 ~]$
```

module unload <package>

```
[mkandes@login02 ~]$ module show gcc/10.2.0
-------------------------------------------------------------------

   /cm/shared/apps/spack/cpu/lmod/linux-centos8-x86_64/Core/
    gcc/10.2.0.lua:
-------------------------------------------------------------------

whatis("Name : gcc")
whatis("Version : 10.2.0")
whatis("Target : zen")
whatis("Short description : The GNU Compiler Collection
    includes front ends for C, C++, Objective-C, Fortran,
    Ada, and Go, as well as libraries for these languages.")
whatis("Configure options : --with-pkgversion=Spack GCC --
    with-bugurl=https://github.com/spack/spack/issues --
    disable-multilib --enable-languages=c,c++,fortran --
    disable-nls --with-system-zlib --with-zstd=/cm/shared/
    apps/spack/cpu/opt/spack/linux-centos8-zen/gcc-8.3.1/
    zstd-1.4.5-mgyyroclnkiuyjh3xozza2ct66ccfitq --with-mpfr-
    include=/cm/shared/apps/spack/cpu/opt/spack/linux-
    centos8-zen/gcc-8.3.1/mpfr-4.0.2-
    oimqnncmbw46n3pfx3c4kx4h6usmzkpp/include --with-mpfr-lib
    =/cm/shared/apps/spack/cpu/opt/spack/linux-centos8-zen/
    gcc-8.3.1/mpfr-4.0.2-oimqnncmbw46n3pfx3c4kx4h6usmzkpp/
    lib --with-gmp-include=/cm/shared/apps/spack/cpu/opt/
    spack/linux-centos8-zen/gcc-8.3.1/gmp-6.1.2-
```

Lmod: A New Environment Module System

Lmod is a Lua based module system that easily handles the MODULEPATH Hierarchical problem. Environment Modules provide a convenient way to dynamically change the users environment through modulefiles. This includes easily adding or removing directories to the PATH environment variable. Modulefiles for Library packages provide environment variables that specify where the library and header files can be found.

https://lmod.readthedocs.io

Hierarchical Format

Searching for Modules