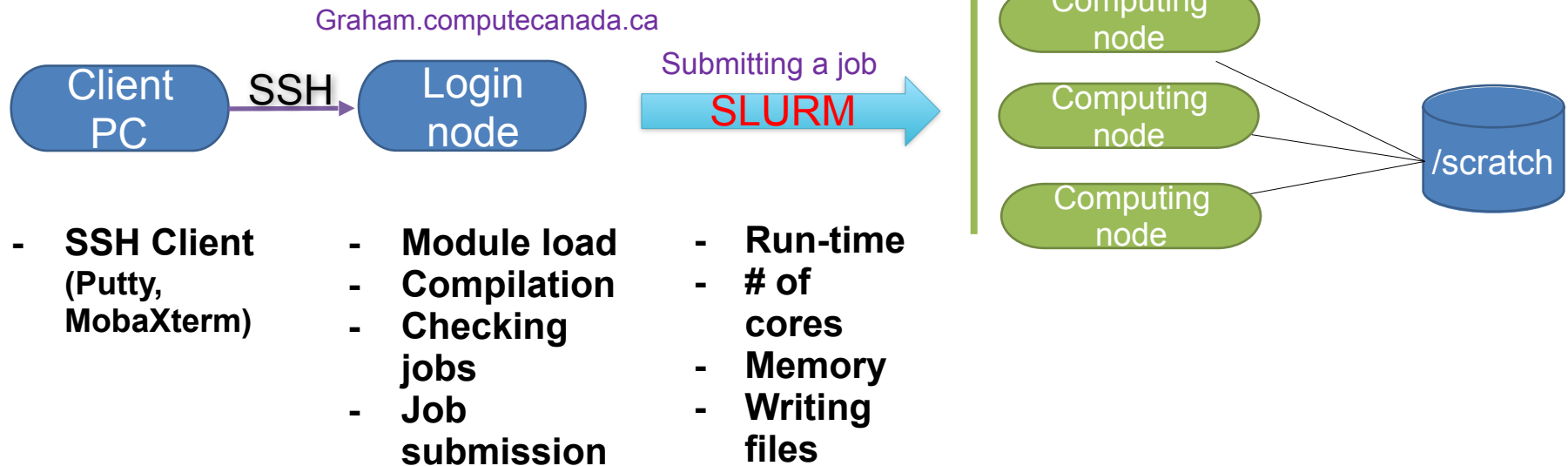




Fundamentals of working at the command line at Graham

Isaac Ye, High Performance Technical Consultant
SHARCNET, York University
isaac@sharcnet.ca

JOB, SCHEDULER, RESOURCE

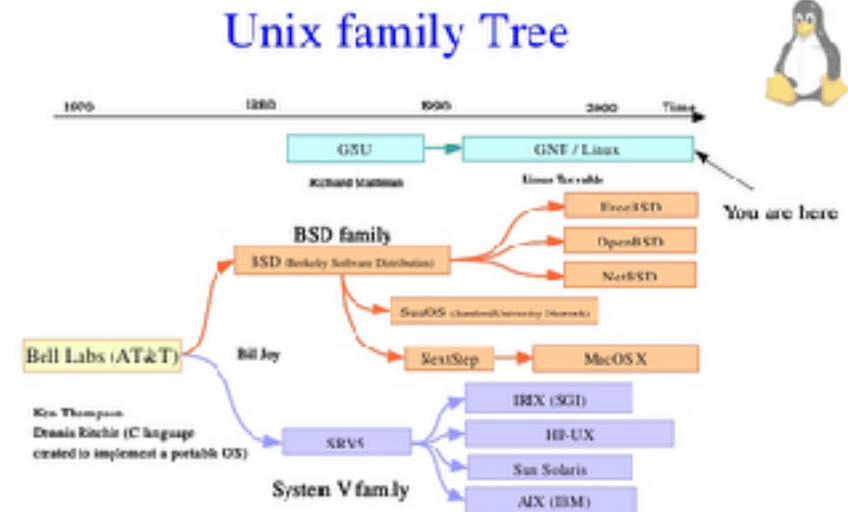


Outlines

- **Basic LINUX concepts**
- **BASH command line essentials**
 - **Command pipes/redirection**
 - **SHELL variables/scripts**
- **Filesystems and permissions**
 - **Managing files**
- **Working environment**
- **Modules (if time permitted)**

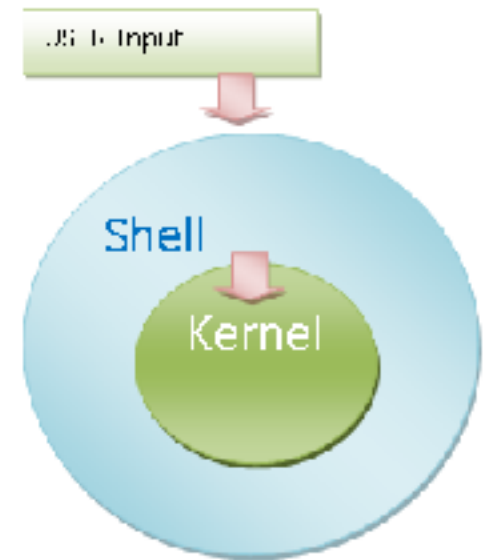
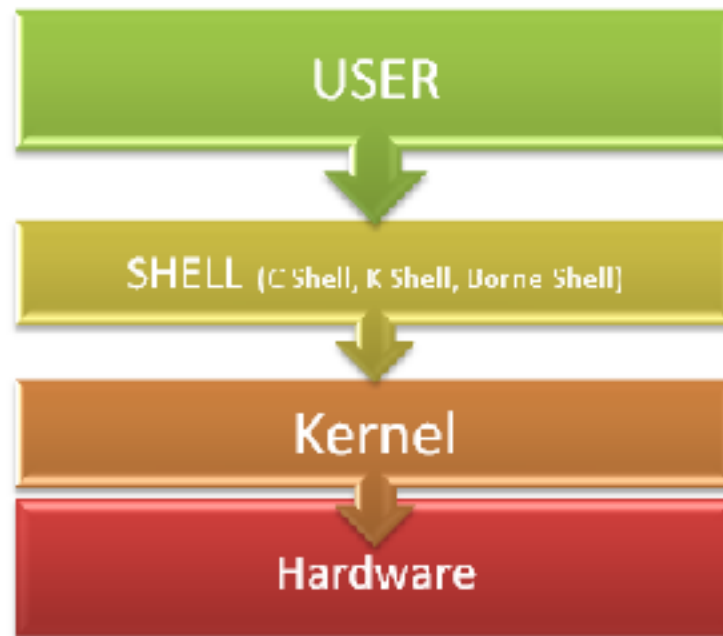
Behind the command line: UNIX & SHELL

- Unix is an operating system, Graham runs CentOS Linux, a distributions of Linux operating systems
- Most Unix-based systems have a GUI interface, but the Command Line (CL) offers more complex and abstract interactions with far less effort
- At login the system starts a SHELL process for you that acts as your CL interpreter to interface with the operation system
- Borne Again SHELL (BASH) is the default shell at SHRCNET



What is Shell?

- Shell is the interface between end user and the system



The shell of Linux

- Linux has a variety of different shells:
 - Bourne shell (sh), C shell (csh), Korn shell (ksh), TC shell (tcsh), Bourne Again shell (bash).
- Certainly **the most popular shell is “bash”**. Bash is an **sh-compatible** shell that **incorporates useful features from the Korn shell (ksh) and C shell (csh)**.
- It is intended to **conform to the IEEE POSIX P1003.2/ISO 9945.2 Shell and Tools standard**.
- It offers functional **improvements** over sh for both **programming and interactive use**.

```
[isaac@saw377 ~]$ echo $SHELL  
/bin/bash
```

Basic Unix Concepts

- File
 - Data stored in a standard format that behaves in a certain way depending on it's function in the system; everything is a file in UNIX
- Program
 - A file that can be executed (run)
- Process
 - A program that is being executed (e.g. your computing job is made of one or more processes)
- Ownership
 - Files/programs/processes are owned by a user and group
- Hierarchical Directory Structure
 - Files are organized in directories (folders) that can have a parent (e.g. /home/isaac/simulation)
 - The base of the hierarchy is 'root', i.e: "/" (forward-slash)
- Managing your files and processes is crucial to effectively using the systems!

Logging in and getting started (some tips!)

- **ssh** to the system you'd like to use
 - you see the message of the day and are left at a command prompt
- each time you type in a command you are executing one or more processes
- you can see commands you ran in the past with **history**
- you can scroll through previous commands with the **↑** and **↓** arrow keys
- you can complete commands / arguments with the **Tab** **↩** key !!!
- depending on your terminal (the software you are connecting with) you should be able to go to the start of a line with **Ctrl-a** or the end with **Ctrl-e**, and cut to the end with **Ctrl-k**
- to exit, run the **exit** command
 - if your terminal is not responding you may be able to disconnect your ssh session gracefully by entering **~**. (sometimes repeatedly, while mashing the **Enter** **↵** key in between...)

The Command Prompt

- Commands are the way to “do things” in UNIX
- A command consists of a command name and options called “flags”
- Commands are typed at the *command prompt*
- In Unix, *everything* (including commands) is case-sensitive

```
[prompt]$ <command> <flags> <args>
```

```
[isaac@saw377 ~]$ ls -l -a my_project
```

↑
Command Prompt

↑
Command

↑
↑
(Optional) flags

↑
(Optional) arguments

Note: In UNIX, you’re expected to know what you’re doing. Many commands will print a message only if something went wrong.

Executing commands

- To run a command you simply type its name in and hit **Enter**.
- The command must be in your **\$PATH** and be executable (we'll get to that later...)

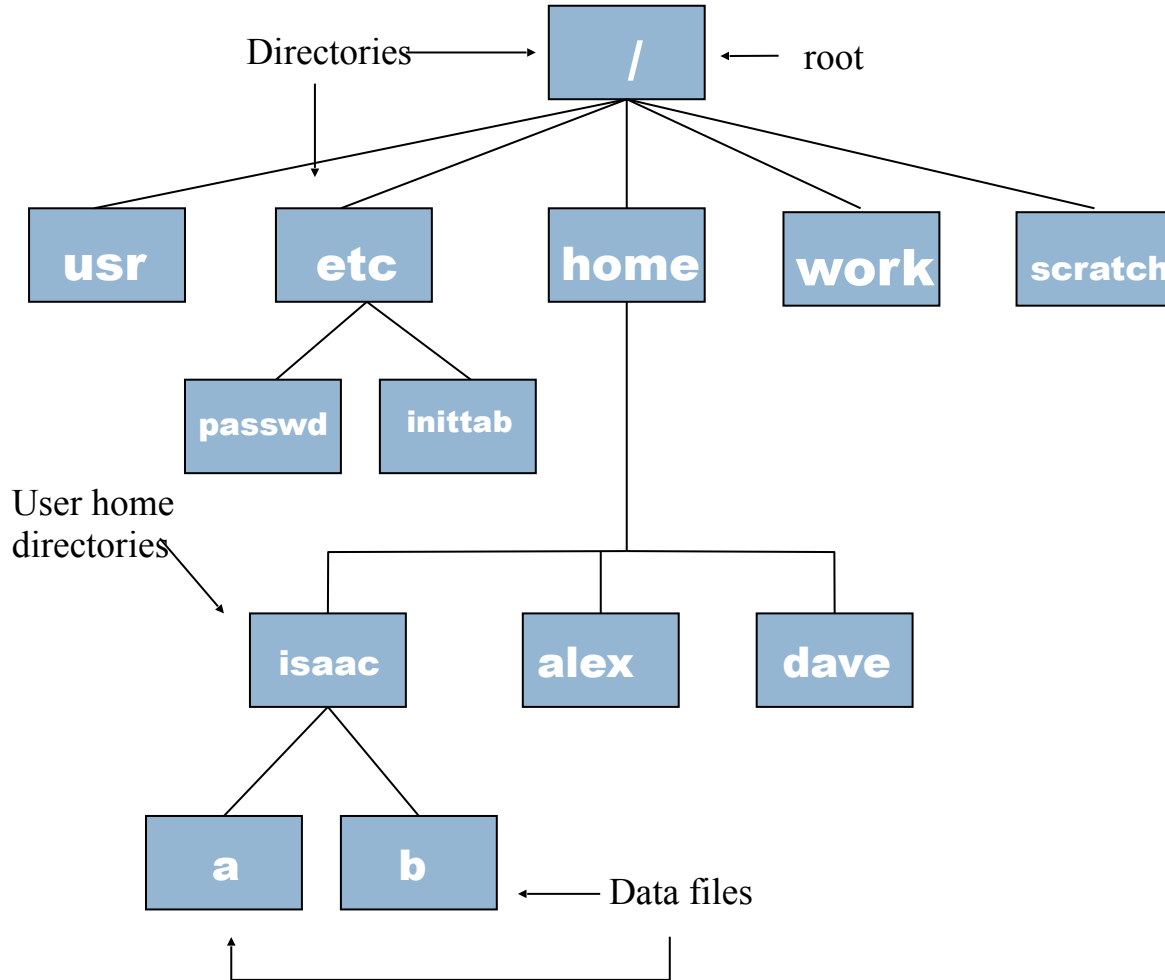
```
[isaac@gra-login4 ~]$ echo $PATH
/opt/software/slurm/current/bin:/cvmfs/soft.computeCanada.ca/easybuild/software/
2017/avx2/Compiler/intel2016.4/openmpi/2.1.1/bin:/cvmfs/soft.computeCanada.ca/
easybuild/software/2017/Core/imkl/11.3.4.258/mkl/bin:/cvmfs/
soft.computeCanada.ca/easybuild/software/2017/Core/imkl/11.3.4.258/bin:/cvmfs/
soft.computeCanada.ca/easybuild/software/2017/Core/ifort/2016.4.258/
compilers_and_libraries_2016.4.258/linux/bin/intel64:/cvmfs/
soft.computeCanada.ca/nix/var/nix/profiles/gcc-5.4.0/bin:/cvmfs/
soft.computeCanada.ca/easybuild/software/2017/Core/icc/2016.4.258/
compilers_and_libraries_2016.4.258/linux/bin/intel64:/cvmfs/
soft.computeCanada.ca/easybuild/bin:/cvmfs/soft.computeCanada.ca/nix/var/nix/
profiles/16.09/bin:/cvmfs/soft.computeCanada.ca/nix/var/nix/profiles/16.09/
sbin:/cvmfs/soft.computeCanada.ca/custom/bin:/opt/software/bin:/opt/software/
slurm/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/isaac/.local/
bin:/home/isaac/bin
```

Basic commands

- Getting help with commands (the most important command!):
 - **man**
- Figuring out who we are and where we are:
 - **whoami, hostname, date**
- Navigating directories:
 - **cd, pwd**
- Manipulating files and directories:
 - **cp, mv, rm, rmdir, mkdir**
- Listing files and their properties:
 - **ls, file**
- Displaying the contents of files:
 - **cat, tail, head, more, wc**
- Investigating running programs:
 - **ps, top**

Linux File System Basics

- Linux files are stored in a single rooted, hierarchical file system
- Data files are stored in directories (folders)
- Directories may be nested as deep as needed



File system structure and shortcuts

- The root of the file system hierarchy is `/`
 - it contains subdirectories which may contain further subdirectories
- File systems are mounted within the hierarchy,
 - e.g.: one starts off in their SHARCNET `/home` directory after logging in: `/home/$USER`
 - Can also refer to this by a shortcut “`~/`”
 - Can always get to this directory by running `cd` without any arguments
- One can refer to file / directory locations by their absolute or relative path
 - The absolute path starts with the root and ends with the file or directory in question, e.g. `/home/$USER/simulation1/output.txt`
 - The relative path depends on which directory you are presently in within the filesystem
 - Run the `pwd` command to see which directory you are in
 - e.g. if we are in `/home/$USER` the relative path to the above file is `simulation1/output.txt`
- Shortcut for current directory is “`.`”; for parent directory it is “`..`”,
 - e.g. one can go up a directory with `cd ..`, run a file in a subdirectory by `./simulation1/program.x`

Some Special File Names

- Some file names are special:
 - / The root directory (not to be confused with the root user)
 - . The current directory
 - .. The parent (previous) directory
 - ~ My home directory
- Examples:
 - ./a same as a
 - ../isaac/x go up one level then look in directory isaac for x

Structures of files

- names can be up to 255 characters, use non-standard characters and file
- name extensions do not matter to most command line programs
- files starting with a “.” are hidden, one can see them by specifying: **ls -a**
- files have a set of attributes associated with them, you can see a long listing
- that includes some of the more pertinent values by running: **ls -l**
 - For each file / directory it will return a record like

```
drwxr-xr-x 1 beaker honeydew 4096 Oct 29 2015 test_dir
```

File Permissions

- The long version of a file listing (`ls -l`) will display the file permissions:

```

-rwxrwxr-x  1 rvdheij  rvdheij      5224 Dec 30 03:22 hello
-rw-rw-r--  1 rvdheij  rvdheij         221 Dec 30 03:59 hello.c
-rw-rw-r--  1 rvdheij  rvdheij        1514 Dec 30 03:59 hello.s
drwxrwxr-x  7 rvdheij  rvdheij        1024 Dec 31 14:52 posixuft
    
```

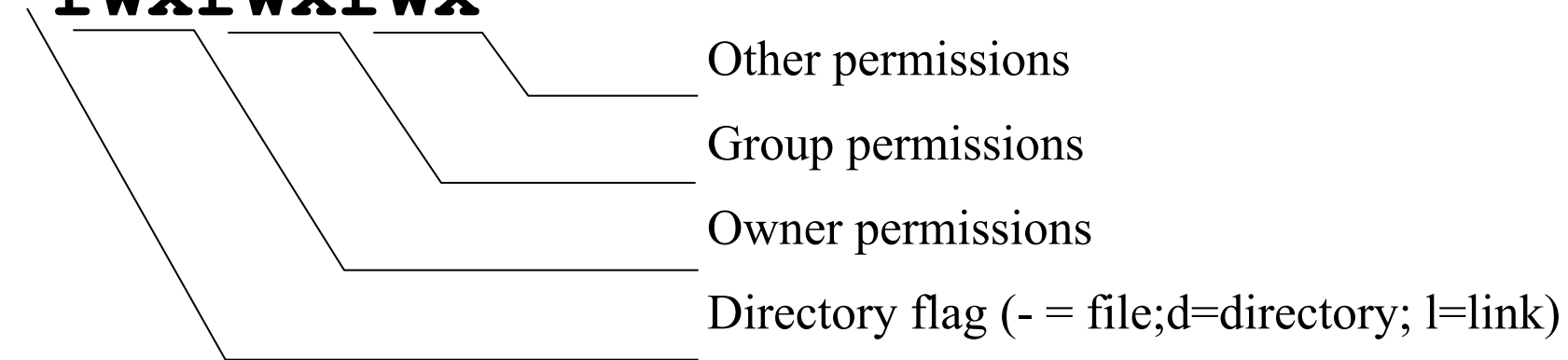
Permissions

Group

Owner

Interpreting File Permissions

-rwxrwxrwx



File Permissions

- Linux provides three kinds of permissions:
 - Read (r, 4) - users with read permission may read the file or list the directory
 - Write (w, 2) - users with write permission may write to the file or new files to the directory
 - Execute (x, 1) - users with execute permission may execute the file or lookup a specific file within a directory

Meaning of output: ls -l for a simple directory

```
drwxr-xr-x 1 beaker honeydew 4096 Oct 29 2015 test_dir
```

1 2 3 4 5 6 7

1. file type and permissions
 - a. File types: - (regular), d (directory), c (character), b (block) , l (link), s (socket), p (pipe)
 - b. Permissions: r (read), w (write), x (execute), s (setgid, setuid) and t (sticky bit)
2. hard link count
 - a. Indicates the number of copies of the particular file
3. user owner or UID of the file
4. group owner or GID of the file
5. size of the file in bytes
6. date and time that the file was last modified (versus access / creation)
7. name of the file

Users and Groups

- Each user on the system is identified by a unique username (stored as an environment variable: **\$USER**) and associated with a numeric UID
 - This is your SHARCNET username @ SHARCNET
- Each user belongs to one or more groups. Each group has a unique group name and numeric GID associated with it
 - At SHARCNET each sponsor has their own group for them and their group members
 - Other groups exist (eg. to get access to commercial software, institutional groups, etc.)
- These are the ownership associated with the file permissions settings
- A file owner can possibly** change the group ownership of a file (**chgrp**) but only the superuser (root user) can change ownership (**chown**)
- One can change specific file permissions (read, write, execute permissions) with the **chmod** command

File Permissions: umask and chmod

- Default file permissions are applied to a file based on a mask value: **umask**
- One can set this value so that when new directories / files are created they are created with different permissions
- Permissions are either represented as octal values or symbolic:

rwX---r-x = **0705** or **u+rwX,o+rx**

rwXr-Xr-- = **0754** or **u+rwX,g+rx,o+r**

- To change a file or directory's permissions use **chmod**:

```
[merz@fenrir ~]$ ls -l t1
```

```
-rw-rw-r--. 1 merz merz 0 Oct 28 09:28 t1
```

```
[merz@fenrir ~]$ chmod u+x,g+x,o+wx t1
```

```
[merz@fenrir ~]$ ls -l t1
```

```
-rwxrwxrwx. 1 merz merz 0 Oct 28 09:28 t1
```

Access Control Lists: the smart way to share

- Access Control Lists are implemented by the file system to support finer-grained permission than are available via regular file permission
 - Can share files or directories with independent permissions for multiple users and groups
- One can see the ACL for a particular file/directory with the **getfacl** command

```
[isaac@gra-login4 ~]$ getfacl testing1
# file: testing1
# owner: isaac
# group: isaac
user::rw-
group::r--
other::---
```

Modifying the access control list

One uses the **setfacl** command to modify the ACL for a file/directory. To add read and execute permissions for /work/beaker for user bunsen, eg.

```
[isaac@gra-login4 ~]$ setfacl -m u:feimao:rx testing1
[isaac@gra-login4 ~]$ getfacl testing1
# file: testing1
# owner: isaac
# group: isaac
user::rw-
user:feimao:r-x
group::r--
mask::r-x
other::---
```

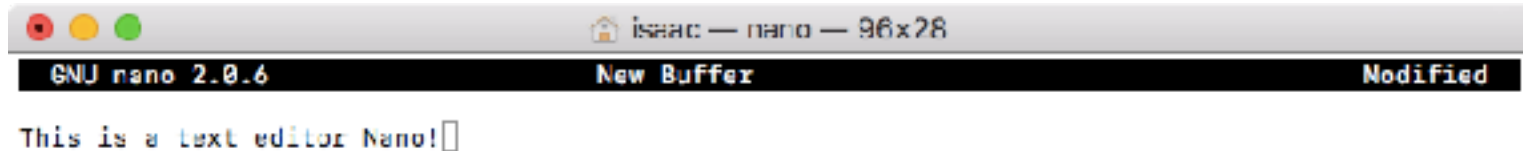
Managing files: tar

- The **tar** command is used to archive files
 - allows one to pack multiple files into a single file
 - allows one to mathematically compress the files into a smaller amount of data
 - very useful for cleaning up, saving data for long term, etc.
- To **create** an archive with **bzip2** compression, with **verbose output**:
 - **tar -cvjf tar_file_output.tar.bz2 <files_or_dirs_to_pack>**
- To **unpack** an archive with **bzip2** compression, with **verbose output**:
 - **tar -xvjf tar_file_output.tar.bz2**
- There are many switches available to control the behavior of tar, eg. **-z** allows one to use the more common gzip compression algorithm

Editing Text

- Which text editor is “the best” is a holy war. Pick one and get comfortable with it.
- Three text editors you should be aware of:
 - nano – An improved ‘pico’ editor
 - To quit: Ctrl-x
 - emacs/xemacs – A heavily-featured editor commonly used in programming
 - To quit: Ctrl-x Ctrl-c
 - vim/vi – Another editor, also used in programming
 - To quit: <Esc> : q <Enter> (or QQ -- capitals matter)

'nano' text editor in command line



```
isaac — nano — 96x28
GNU nano 2.0.6      New Buffer      Modified
This is a text editor Nano!|
```

^G Get Help
^X Exit

^O WriteOut
^J Justify

^R Read File
^W Where Is

^Y Prev Page
^V Next Page

^K Cut Text
^U UnCut Text

^C Cur Pos
^T To Spell

Command Pipe and Redirection

- Pipes (|) offer a way to chain together commands (sending output from one to the input for another)
 - Redirection (> , >>) lets one store the output of a command in a file
 - > will overwrite the output file, >> will append to it instead
- For example, if we want to count all the files in a directory and store that value in a file we can do it in one command:

```
ls -l * | wc -l > number_files_in_directory
```
- By chaining together simple commands with pipes one can evaluate sophisticated expressions with little effort

Command aliases

- The **alias** command allows you to build new commands
- For example, one can set up an alias to run a command with particular, common switches:

```
alias lsfull='ls -larth'
```
- Now when one runs **lsfull** it will automatically expand to execute **ls** with the **-larth** switches / options
- One can also use environment variables and pipes, etc.:

```
alias wh='echo "["$USER"] ["$HOSTNAME"] ["$PWD"] ["`date`"] ["`uptime`"]'
```

```
alias psme='ps aux | grep $USER | grep -v grep'
```
- Run **alias** without any arguments to see which are presently set

SHELL initialization: /home/\$USER/.bashrc

- When you first login or start a new bash process, it's environment and
- configuration is set based on a configuration file: /home/\$USER/.bashrc
- Useful for setting up modules, environment variables or command aliases to persist between sessions

```
[isaac@gra-login4 ~]$ cat .bashrc
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
    test -n "$__ETC_PROFILE_SOURCED" || . /etc/bashrc
fi

# Uncomment the following line if you don't like
# export SYSTEMD_PAGER=

# User specific aliases and functions

export HISTSIZE=
export HISTFILESIZE=

export SLURM_ACCOUNT=def-isaac
export SBATCH_ACCOUNT=$SLURM_ACCOUNT
export SALLOC_ACCOUNT=$SLURM_ACCOUNT

export NIXPKGS_ALLOW_UNFREE=1
export SKIP_CC_CVMFS=1

alias ml='module list'
alias psme='ps aux | grep $USER | grep -v grep'
```

Environment Variables

- Environment variables are global settings that control the function of the shell and other Linux programs. They are sometimes referred to global shell variables.
- Check your environment

```
[isaac@saw377 ~]$ env
MKLROOT=/opt/sharcnet/intel/11.0.083/ifc/mkl
MODULE_VERSION_STACK=3.2.6
MANPATH=/opt/sharcnet/octave/current/share/man:/opt/sharcnet/netcdf/current/man:
FOAM_SOLVERS=/work/isaac/OpenFOAM/OpenFOAM-1.6/applications/solvers
FOAM_APPBIN=/work/isaac/OpenFOAM/OpenFOAM-1.6/applications/bin/linux64GccDPOpt
FOAM_TUTORIALS=/work/isaac/OpenFOAM/OpenFOAM-1.6/tutorials
FOAM_JOB_DIR=/work/isaac/OpenFOAM/jobControl
HOSTNAME=saw377
snrestart=--nosrun /opt/sharcnet/blcr/current/bin/sn_restart.sh
IPPROOT=/opt/sharcnet/intel/11.0.083/icc/ipp/em64t
INTEL_LICENSE_FILE=/opt/sharcnet/intel/11.0.083/ifc/licensesADFBIN=/opt/sharcnet/adf/current/bin
```

Some Important Environment Variables

- HOME
 - Your home directory (often be abbreviated as “~”)
- TERM
 - The type of terminal you are running (for example vt100, xterm, and ansi)
- PWD
 - Current working directory
- PATH
 - List of directories to search for commands

PATH Environment Variable

- Controls where commands are found
 - PATH is a list of directory pathnames separated by colons. For example:
`? PATH=/bin:/usr/bin:/usr/X11R6/bin:/usr/local/bin:/home/alex/bin`
 - If a command does not contain a slash, the shell tries finding the command in each directory in PATH. The first match is the command that will run
 - Set in `/etc/profile`, `~/.profile`, `~/.bashrc`

Managing files: basic command

- to list files in a directory: **ls**
 - one can use a shell metacharacter, *****, to match files with particular names, e.g. **ls *.txt**
 - a useful set of options is **ls -larth**; display all files, with long information, sorted from oldest to
 - newest, with sizes in human readable format
- to see how much space is being used on different file systems: **df -h**
- to see how much space is being used in a folder: **du -h**
- to see your disk usage quota status at SHARCNET: **quota**
- a useful utility is **dos2unix**. Text files created in Windows typically do not work on Unix-based systems. You need to run them through this command first,
 - e.g. **dos2unix -n windows_file.txt linux_file.txt**
- another useful utility is the **find** command. It can search within file trees for files of a particular type, with particular names, dates, contents, etc.

Find syntax

- `find <path to start searching from> <expression>`
- Where `<expression>` can be one or more of the following (plus others, see `-exec !`):
 - `-name <filename pattern>`
 - `-type <file type: block (b), char (c), regular (f)>`
 - `-user <owner of the file (uid)>`
 - `-group <group owner of the file (gid)>`
 - `-mtime <files modified within x days>`
 - `+x` : older than, `-x` : newer than, `x` : equal to
 - `-ls`
 - display file attributes
 - `-print`
 - print the names of the files returned, without `-print` results are suppressed
- Example:
 - `find /work/beaker -type f -uid 1008 -exec mv { } /tmp \;`
 - “find all files in `/work/beaker` owned by UID 1008, and move them to `/tmp`”

Module

- **LMOD** tool developed at TACC replaces 'Environment modules'(flat structure) in most legacy servers
- Lmod is a Lua based module system that easily handles the MODULEPATH Hierarchical problem
- Similar to the module in legacy system
- Supports flat layout of modules and software hierarchy
- A "modulefile" contains the information needed to make an application or library available in the user's login session. Typically a module file contains instructions that modify or initialize environment variables such as PATH and LD_LIBRARY_PATH in order to use different installed programs.

Module search

- Search for modules is with the “module spider” command. This command searches the entire list of possible modules. The difference between “module avail” and “module spider” is explained in the “Module Hierarchy” and “Searching for Modules” section.

```
[isaac@gra-login4 ~]$ module spider openmpi/2.1.1
```

```
-----  
openmpi: openmpi/2.1.1  
-----
```

Description:

The Open MPI Project is an open source MPI-2

Properties:

MPI implementations / Implémentations MPI

You will need to load all module(s) on any one of

```
nixpkgs/16.09 gcc/4.8.5  
nixpkgs/16.09 gcc/5.4.0  
nixpkgs/16.09 gcc/5.4.0 cuda/8.0.44  
nixpkgs/16.09 gcc/5.4.0 cuda/9.0.176  
nixpkgs/16.09 gcc/6.4.0  
nixpkgs/16.09 gcc/6.4.0 cuda/9.0.176  
nixpkgs/16.09 gcc/7.3.0  
nixpkgs/16.09 intel/2014.6  
nixpkgs/16.09 intel/2016.4  
nixpkgs/16.09 intel/2016.4 cuda/8.0.44  
nixpkgs/16.09 intel/2016.4 cuda/9.0.176  
nixpkgs/16.09 intel/2017.1
```

...

Module collections

- Lmod allows you to create a collection of modules. To do so, first load the desired modules. For example:

```
module load gcc/4.8 openmpi/1.8 mkl  
module save my_modules
```

- The my_modules argument is a name you give the collection.
- Then in a later session or in a job you can restore the collection with the command

```
module restore my_modules
```

- a user can print the contents of a collection with:

```
module describe my_modules
```

Module collections

- To see a list of your saved modules (stored @ ~/.lmod.d)

```
module savelist
```

- The my_modules argument is a name you give the collection.
- Then in a later session or in a job you can restore the collection with the command

```
module restore my_modules
```

- a user can print the contents of a collection with:

```
module describe my_modules
```

Module features

- Only one version loaded at a time
 - Lmod will refuse to load two versions of the same module at the same time. For example, you cannot have versions 4.8 and 5.4 of the GCC compilers loaded at once.
- Only one module in the same family loaded at a time
 - It is possible for administrators to specify that two modules with different names are of the same family. Lmod will refuse to load two modules of the same family. Typical examples are compiler modules (gcc, intel), MPI modules (openmpi, mvapich2), or BLAS library modules (mkl, openblas).
- Automatic replacement of modules
 - When Lmod detects two modules of the same family, or two version of the same module, the command module load will automatically replace the original module with the one to be loaded. In the cases where the replaced module is a node in the module hierarchy, dependent modules will be reloaded if



THANK YOU!