Fundamentals of working at the command line at SHARCNET

General Interest Webinar 2015 10 28

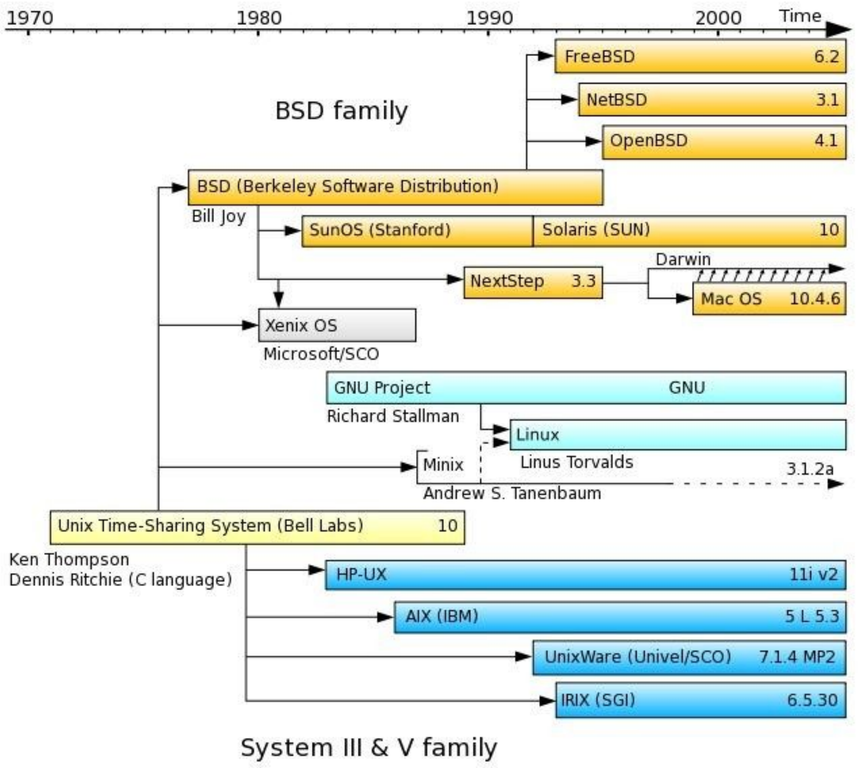
Hugh Merz

Topics

- Basic UNIX concepts
- Basic bash command line behavior
- File systems and permissions
- Managing files
- Text editing
- Command pipes and redirection
- Shell variables, initialization, and scripts

Behind the command line: Unix and bash

- Unix is an operating system, all SHARCNET systems run some variant of Unix (eg. Linux)
- Most Unix-based systems have a GUI interface, but the command line offers more complex and abstract interactions with far less effort
- At login the system starts a shell process for you which acts as your command line interpreter to interface with the operating system
- Borne Again Shell (bash) is the default shell at SHARCNET



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 (eg. 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, eg. /home/$USER/sim1
    - The base of the hierarchy is *root* , ie: / (forward-slash)

<u>Managing your files and processes is crucial to effectively using the systems!</u>

Logging in and getting started… (important 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…)

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…)
- General syntax of a command:

```
$ command [[-]option(s)] [option argument(s)] [command argument(s)]
```

- command: the name of the command or utility: ls, man, cat, mv
- options: change the behaviour of the basic command: ls -l vs. ls
  - may or may not be preceded by "-"
- option arguments: change the behaviour of an option: tail –c 5 file1 vs. tail –c 15 file1
- command arguments: what is affected by the command, usually files or the output of another command

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

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, eg.
  - one starts off in their SHARCNET /home directory after logging in: /home/$USER
    - Can also refer to this by a shortcut “~/”
    - Can always get to this directory by running cd without any arguments
- One can refer to file / directory locations by their *absolute* or *relative* path
  - The absolute path starts with the root and ends with the file or directory in question, eg.
    - /home/$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
    - eg. 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 “..”, eg.
  - one can go up a directory with cd .. , run a file in a subdirectory by ./simulation1/program.x

Structure 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
```

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, etc. permissions) with the chmod command

File permissions

- security is based on three basic actions
  - Read, Write, Execute
- listed in triplets: user permissions, group permissions, everyone (other)
- meaning depends on file type
  - regular, directory and special file distinctions

-rwxrwxrwx

Owner Group Other

- drwxr-xr-x   1   beaker   honeydew   4096   Oct 29 2015   test_dir
  - read, write and execute for the owner, read and execute only for group members and others

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
```

Examples of different file / directory permissions

```
-rw-rw-r-- 1 beaker honeydew 0    2015-09-16 20:54
default_file_at_SHARCNET

drwxrwxr-x 2 beaker honeydew 4096     2015-09-16 20:59
default_dir_at_SHARCNET

-rwxrwxrwx 1 beaker honeydew 0    2015-09-16 20:54 fully_executable

-------rwx 1 beaker honeydew 24   2015-09-16 20:57 public_executable

d--------- 2 beaker honeydew 4096     2015-09-16 20:59 root_only

-rwx------ 1 beaker honeydew 0    2015-09-16 20:55 user_executable

drwx------ 2 beaker honeydew 4096     2015-09-16 20:59 user_only
```

Access Control Lists: The smart way to share

- Access Control lists are implemented by the file system to support finer-grained permissions than are available via regular file permissions
  - 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, eg.

```
[beaker@dtn ~]$ getfacl /work/beaker
getfacl: Removing leading '/' from absolute path names
# file: work/beaker
# owner: beaker
# group: beaker
user::rwx
group::r-x
other::--x
```

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.

```
[beaker@dtn ~]$ setfacl -m u:bunsen:rx /work/beaker
```

Now there is an entry for *user:bunsen* with *r-x* permissions:

```
[beaker@dtn ~]$ getfacl /work/beaker
getfacl: Removing leading '/' from absolute path names
# file: work/beaker
# owner: beaker
# group: beaker
user::rwx
user:bunsen:r-x
group::r-x
mask::r-x
other::--x
```

Managing files: basic commands

- to list files in a directory: ls
    - one can use a shell metacharacter, * , to match files with particular names, eg. ls *.txt
        - other metacharacters exist and are very useful, see *regular expressions* if you're daring
    - 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, eg. `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"
```

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

- There are quite a few different text editors to choose from, some of the more portable editors which are on SHARCNET systems include:
    - vim
        - minimalist; modal: insert vs. editor mode; fast but arcane
    - emacs
        - very extensible, ability to add rich functionality
    - nano
        - a simplistic editor with basic emacs-like controls

- You can use the mouse to cut and paste within these editors - very useful
- Try them out to see what works for you!

Worst case, use a gui editor on one of the SHARCNET visualization systems :)

Command pipes 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 -1 * | wc -l  > number_files_in_directory
```

- By chaining together simple commands with pipes one can evaluate sophisticated expressions with little effort

Environment Variables

- The shell environment allows one to store values in *environment variables*
    - Many are provided by default when you login, eg. $USER , $HOME , $PWD , $PATH
- $ expands the value of a variable:

```
beaker@dtn:~$ echo $PATH
/bin:/usr/bin
```

- The export command allows one to set new variables or modify existing ones:

```
beaker@dtn:~$ export PATH=$HOME/bin:$PATH
beaker@dtn:~$ echo $PATH
/home/beaker/bin:/bin:/usr/bin
```

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

```
[beaker@orc-login1 ~]$ cat ~/.bashrc
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi

# User specific aliases and functions
module load fftw/intel/2.1.5
alias psme='ps aux | grep beaker | grep -v grep'
export WORKDIR=/work/beaker/simulation42
```

- Useful for setting up modules, environment variables or command aliases to persist between sessions

Shell scripts

- One can easily make a file with multiple commands that can be executed at once: a script
- Simply put commands in a file (make sure the header is there!) then change it to have executable permissions:

```
[beaker@dtn scripts]$ cat script.x
#!/bin/bash
echo 'this is a script'
whoami
echo my home is: $HOME
[beaker@dtn scripts]$ chmod +x script.x
[beaker@dtn scripts]$ ./script.x
this is a script
beaker
my home is: /home/beaker
```