Lecture # 4 – Shell Basics

- What is the "shell"?

  The "shell" is the program that starts when you login that provides the interface between you and the rest of the operating system.

  There are two main responsibilities of the shell – one is to provide an interactive environment, and the other is to provide a programming interface.

  There are many shells available in UNIX. The most common ones include Bourne shell, C-shell, Korn shell, Bash, and Tcsh.

- Quick history of the shell

  The very first shell was the Bourne shell. Since it was the first shell, both its programming interface, and interactive capabilities are somewhat primitive compared with the other shells.

  With the introduction of BSD UNIX, we also gained the c-shell which included a more c-like programming interface, and some more advanced interactive features like command history.

  Korn shell came along a few years later and included an expansion of the Bourne shell programming interface, along with many interactive improvements like command line editing.

  Bash was introduced along with the advent of Linux, and it includes many of the features available in Korn shell, and some additional features as well.

  Tcsh came along around the same time as Bash, and provides an update to the c-shell to include things like command line editing, etc.

- The command line

  General format of a command is: cmd arg1 arg2 … argn
  Remember argc and argv from your C programming
  Options are a type of argument that typically starts with a '-' (i.e. ls –l)
  Multiple options can usually be grouped as one argument (i.e. ls –ls)

- Command line processing

  The different shells process input a line at a time
  Individual keystrokes are handled by the terminal driver.
  In other words, the shell really gets involved only when you hit enter.

Steps involved in command processing:

1. Determine command (first word on the line)
2. Separate and store arguments
3. Find the location of the command in the file system (absolute directory or PATH)
4. If command is not found, then tell the user and return a prompt
5. Otherwise, execute the program and wait for it to complete

- stdin, stdout, and stderr

Most everything in UNIX can be thought of as a file (a stream of bytes).
This includes normal files, terminals, modems, printers, disk drives, peripherals, etc.

The terminal is represented as a file (/dev/ttyxx).
By default, the shell opens stdin, stdout, and stderr and points them to your terminal.

Name	Descriptor Number
stdin	0
stdout	1
stderr	2

- Basic I/O Redirection

Sending output somewhere else or getting input from somewhere else (other than terminal)

This is one of the strengths of UNIX; the ability to combine smaller programs without programming.

1. '>' sends output (stdout) to a file (i.e. cmd [args] > file)

   Note: File will be overwritten if it exists

   Example: cat file1 > file2 (equivalent to cp file1 file2 for text files)
   Example: ls > ls.out

2. '<' gets input (stdin) from a file (i.e. cmd [args] < file)

   Note: Watch out for reading a file and redirecting output to it in the same command
   (i.e. cat file1 file2 > file2 or sort < file1 > file1)

   Example: cat < file1

   Note: What is the difference between "cat < file1" and "cat file1"

3. '>>' appends stdout to a file (i.e. cmd [args] >> file)

   Note: Creates file if it does not exist.

   Example: cat file1 >> file2
   Example: ls >> ls.out

4. '|' (pipe) sends stdout of one program to stdin of another (i.e. prog1 | prog2)

   Same as prog1 > temp; prog2 < temp; rm temp, but with no temporary file

   Examples: ps –ef | grep richj
   ls | more
   ls | wc
   who | sort

The pipe is one of the most important features of the shell since it allows us to combine commands to accomplish a larger purpose without writing any code.

- /dev/null

  /dev/null is a special file in UNIX which represents the "bit bucket"

  $ cat /dev/null > file1.txt
  $ grep cs390 file2.txt > /dev/null

- Filters

  Program who reads stdin, performs some processing, and writes to stdout
  Different than interactive programs (vi, mail, etc)
  Filters are significant since they are excellent candidates to be used in pipelines

  Examples:

  ps –ef | grep richj | more;
  ls | sort | uniq

- The tee command

  Send stdout to a file and the terminal

  $ who | tee who.out | wc –l

  This command displays a count of the number of users on the system, and stores a list of them in the file "who.out".

- Background processes

  '&' at end of command to put process in the background (i.e. get prompt back now)
  Note: You should redirect output for commands you run in background

- File name generation (wildcards)

  ? – match any single character
  * - match zero or more characters (but not a leading period)
  [ ] – match any character between the brackets

  Example: lpr part[012345]
  lpr part[0-35] means 0-3 or 5
  lpr part[0-9] part[12][0-9] part3[0-5]