Linux Shells

- Book Chapter 5
- What is a shell?
- Examples:
  - bash Bourne Again shell
  - ksh Korn shell
  - tcsh C shell

Figure 5-1. Comparison of Linux and UNIX shell names.

Shell	Linux	UNIX
Bash	sh or bash	bash
Korn shell	ksh, pdksh, or zsh	ksh
C shell	csh or tcsh	csh

Linux Shells

- Linux default shell /bin/bash
- How do I know what shell I am running?
  - *echo $SHELL*
  - *env*

Shell Operations

- Shell invocation sequence

  1. read special startup file containing initialization info
  2. display prompt and wait for user command
  3. execute command

     if “end of file” exit shell

     otherwise execute command entered

Shells operations

- Shell command examples
  - ls
  - ps -ef | sort | ul -tdumb | lp
  - “\” serves as line extension character

    e.g.: *echo this is a long command that does *

    not fit in one line

Commands - executables

- Where are the shell commands?
  - most commands invoke utility programs
    - shell executes executable stored in file system
    - e.g., *ls* (\bin\ls)

Commands - built-in

- Where are the shell commands?
  - other commands are built-in, e.g.,
    - *echo [option]... [string]* which displays line of text
    - *cd*
    - bash built-ins *man cd* outputs
    - *bash, :, ., [, alias, bg, bind, break, builtin, cd, command, compgen, complete, continue, declare, dirs, disown, echo, enable, eval, exec, exit, export, fc, fg, getopts, hash, help, history, jobs, kill, let, local, logout, popd, printf, pushd, pwd, read, readonly, return, set, shift, shopt, source, suspend, test, times, trap, type, typeset, ulimit, umask, unalias, unset, wait - bash built-in commands, see bash(1)*

Shell Metacharacters

Symbol	Meaning
`>`	Output redirection; writes standard output to a file.
`>>`	Output redirection; appends standard output to a file.
`<`	Input redirection; reads standard input from a file.
`*`	File substitution wildcard; matches zero or more characters.
`?`	File substitution wildcard; matches any single character.
`[...]`	File substitution wildcard; matches any character between brackets.
`` `command` ``	Command substitution; replaced by the output from command.
`|`	Pipe symbol; sends the output of one process to the input of another.
`;`	Used to sequence commands.
`||`	Conditional execution; executes a command if the previous one failed.
`&&`	Conditional execution; executes a command if the previous one succeeded.
`(...)`	Groups commands.
`&`	Runs a command in the background.
`#`	All characters that follow up to a newline are ignored by the shell and programs (i.e., a comment).
`$`	Expands the value of a variable.
`\`	Prevents special interpretation of the next character.
`<<tok`	Input redirection; reads standard input from script up to `tok`.

Redirection

- Output Redirection
  - Store output of process in file
  - *echo hello world > file* writes string into file
  - *echo more words >> file* append string to file
- Input Redirection
  - use contents of file as input

Filename Substitution

- Shell wildcards
  - * matches any string including empty string
  - ? matches any single character
  - [..] matches any one of the chars in brackets
  - what do the following examples display?

*ls -l *.c*

*ls [A-Za-z]**

*ls */*.c*

Pipes

- Use output of one command as input to another
  - pipe output of one process to the input of another
  - example

ls |wc -w

cat /etc/passwd | gawk -F: '{ print $1 }' | sort

Pipes

- example: tee - read from standard input and write to standard output and files

```
$ who | tee who.capture | sort
ables    pts/6   May  3 17:54 (gw.waterloo.com)
glass    pts/0   May  3 18:49 (blackfoot.utdall)
posey    pts/2   Apr 23 17:44 (:0.0)
posey    pts/4   Apr 23 17:44 (:0.0)
$ cat who.capture          ...look at the captured data.
glass    pts/0   May  3 18:49 (blackfoot.utdalla)
posey    pts/2   Apr 23 17:44 (:0.0)
posey    pts/4   Apr 23 17:44 (:0.0)
ables    pts/6   May  3 17:54 (gw.waterloo.com)
```

Command Substitution

- A command surrounded by grave accents (`, *back quotes*) is executed and its output (after evaluation) is inserted in the command in its place.
  - Any newlines in the output are replaced by spaces
  -
```
-bash-3.2$ echo the date today is `date`
the date today is Mon Sep 20 10:44:18 PDT 2010
-bash-3.2$
```

```
-bash-3.2$ echo there are `who | wc -l` users on
the system
there are 3 users on the system
-bash-3.2$
```

Sequences

- A series of simple commands or pipelines separated by semicolons is executed in sequence, from left to right.

```
-bash-3.2$ who ; date; ps
krings   pts/0        2010-09-20 10:43 (star.cs.uidaho.edu)
jeffery  pts/1        2010-09-14 11:50 (clint2.cs.uidaho.edu)
jeffery  pts/3        2010-09-08 13:02 (75.87.248.45)
Mon Sep 20 10:51:47 PDT 2010
  PID TTY          TIME CMD
26826 pts/0    00:00:00 bash
26941 pts/0    00:00:00 ps
```

Conditional Sequences

- Every Linux process terminates with an exit value
  - 0 means normal execution
  - nonzero indicates failure
  - built-in commands return 1 if they fall

Conditional Sequences

- Using exit values
  - If you specify a series of commands separated by && tokens, the next command is executed only if the previous command returns an exit code of 0.
  - If you specify a series of commands separated by || tokens, the next command is executed only if the previous command returns a nonzero exit code
  - The && and || metacharacters mirror the operation of their counterpart C operators.
  - *-bash-3.2$ gcc cpu.c && ./a.out*
  - *-bash-3.2$ gcc myprog.c || echo compilation failed.*

Background Processing

- A command followed by the & metacharacter will be executed in the background
  - execute several programs in background
  - look at them using ps or top
  - bring them to the foreground
  - use *bg, fg, ^Z*

Background Processing

- A process running in the background may still output to the screen (stdout)
  - different ways to deal with that, e.g.
    - redirect to file
    - redirect to dummy device, /dev/null
    - mail to yourself, e.g. *find . -name a.c -print | mail glass &*