Unit 2
Introduction to Shells

Dr. S.Thenmozhi

Shell

- Part that is visible to the user
- Receives and interprets the command entered by the user
- If the command requires a utility, it requests the kernel to execute the utility
- If it requires to execute an application program, it requests kernel to execute the application program
- Major parts of shell – Command interpreter and Shell programming
- A shell script is a file that contains shell commands that perform a useful function. It is also known as shell program.

- **Bourne Shell** – Steve Bourne at AT&T Labs – Oldest Shell –Enhanced version – Bourne again shell(**Bash**) used in Linux.
- **C Shell** – Bill Joy at Berkeley – Commands look like C statements. **Tcsh** is compatible in Linux
- **Korn Shell** – David Korn at AT&T labs – compatible with Bourne Shell

Unix Sessions

- Login
- Display Prompt
- Receive command
- Is Executable?
  - IfYes, Execute command
  - Else, Display Error
- Display prompt
- Logout on logout command

- When you are logged in, you are in any of the five shells
- The administrator decides which shell you can start with
- But, the system can have more than one shells
- How to move to another shell?
  - $ksh
  - $csh
  - $bash

- **SHELL variable identifies the login shell**
  - $echo $SHELL
- **Current Shell Verification**
  - echo $0
- **Shell Relationships**
  - When we move from one shell to another, Unix remembers the path by creating a parent-child relationship
  - $ [you are in bash now]
  - $csh [now you are in cshell] [for this scenario, csh is child and bash is parent]
  - %ksh [now you are in korn shell] [for this scenario, ksh is child and csh is parent]
  - exit [exits from korn shell]
  - exit [exits from csh]
  - exit [exits from bash, i.e, logout]

- For logging-out you must in the original shell
- Exit command, terminates the session
- Logout command, Quits from the session
- You can logout from a child shell
- This is to take-care of the jobs running in the background
- **If any process started by the child shell and the shell terminates, the jobs will be taken care of by parent shell**

Standard Streams

- Unix takes input from a stream known as **standard input**
- If a command is correct, output is send to Output stream – **standard output**
- If command is not correct, Error message is send to a Error stream – **standard error**
- Stream Descriptors
  - Std. Input – 0 [default : keyboard]
  - Std. Output – 1 [default: monitor]
  - Std. Error – 2 [default: monitor]
- It is not mandatory that every command sends output to the std output. For ex: lpr command sends the output to printer.

Redirection

- Redirection is the process by which we specify that a file is to be used in place of one of the standard ones.
- With input files, we call input redirection
- Output files, output redirection
- Error files, error redirection
- Redirecting Input
  - Redirecting the standard input from keyboard to text file
  - Operator - < [arrow pointing to the command]
  - The command has to get input from the file
  - Usage: **command 0<file1** or **command < file1**

- Redirecting Output
  - Redirecting the standard output from monitor to text file
  - i.e, the command output is copied to a file
  - Operator >
  - The command has to send the output to the file
  - Usage: **command 1>file1** or **command > file1**
  - Usage: **command 1>| file1** or **command >| file1**
  - Usage: **command 1>>file1** or **command >> file1**

- If the file does not exist, Unix Creates the file and write the output
- If already exists, then action depends on **noclobber** setting
- When **noclobber is on**, it **prevents** redirected output from **destroying** the existing file i.e, **file will not be overwritten**
- To know the noclobber setting status of your machine
  - Set –o | grep noclobber
- Overriding output redirection
  - Usage: who >| out1

Redirecting Errors

- By default it is combined with standard output stream on the monitor.
- It can also be redirected to file
- Usage: **command 1> f1 2>f2**
- **Eg: ls –l file1 file2 1>out1 2>err1**

Pipes (|)

- Pipe is an **operator** that temporarily saves the output of one command in a buffer and provide that as input to the next command
- The first command must be able to send output to standard output and turn it into input for the second command
- The second command must be able to read the its input from the standard input
- i.e, the left command needs to send output to the right command.
- Usage: command1 | command2
- Eg: who | lpr

tee

- Copies standard input to standard output and at the same time copies it to one or more files
- First copy goes to std. output (monitor). At the same time, output is sent to the optional files specified in the arguments list.
- Tee command creates the specified files if it does not exists and overwrites them if already existing.
- To prevent from overwriting use –a option to append output in the file rather than deleting the existing content
- Eg: tee tout
- Eg: who | tee t1out

Command Execution

- Four Types
  - Sequenced Command
  - Grouped Command
  - Chained Command
  - Conditional Command
- **Sequenced Command**
  - Sequence of commands entered on one line. Each command Separated by semincolon (;)
  - Commands neednot have any relationship between them. It is just entered in one line
  - Usage: command1;command2;command3
  - Eg: echo welcome; cal 10 2018; ls -l

Command Execution

- **Grouped Commands**
  - Commands are grouped by placing them in parenthesis
  - This will help to group the command sequence and redirect to an output file
  - Usage: (command1;command2)
  - Eg: (echo welcome; cal 10 2018)>out1.txt
- **Chained Commands**
  - Inducting pipelined commands where the output of the first command goes as input to the second command
  - Usage: command1 | command2
  - Eg: ls –l | more

- **Conditional Commands**
  - Combine two or more commands using conditional relationships
  - And (&&) and or(||) operators can be used
  - && - second command is executed when the first is successful
  - || - Second command is executed if the first fails
  - Usage: command1 && command2
  - Eg: cp file1 file2 && echo “Copy sucessful”
  - Cp file file2 || “echo copy Failed”

Command-Line Editing

- vi, vim, emacs are command line editors
- vi is the default editor of Unix operating system
- The UNIX vi editor is a full screen editor and has two modes of operation:
  - *Command mode* commands which cause action to be taken on the file, and
  - *Insert mode* in which entered text is inserted into the file.
- In the command mode, every character typed is a command that does something to the text file being edited
- In the insert mode, every character typed is added to the text in the file; pressing the <Esc> (*Escape*) key turns off the Insert mode.

- **To Start vi**
  - vi filename
- **To save a file**
  - :w
- **To Exit vi**
  - :x or wq – save and quit vi
  - :q – quit
  - :q! – quit without save
- **Moving the cursor**
  - j – move cursor down one line
  - k – move cursor up one line
  - h – move cursor one left
  - l – move cursor one right
  - 0(zero) – move to beg of the line
  - $ - move to the end of the line
  - w – move to beg of word
  - b – move to the end of the word

- **Screen Manipulation**
  - ^f – move forward one screen
  - ^b – move backward one screen
- **Inserting and Adding Text**
  - i – insert text before cursor
  - I – insert text at the beginning of the current line
  - a – append text after cursor
  - A – append text to the end of the current line

Quotes

- Shells use selected set of meta-characters in commands
- Meta-characters have special interpretation
- Some metacharacters are |, “ “, ‘ ‘, ` `, $, &, ~, *, ?, space, >, < (redirection) etc.
- Quotes can be
  - Backslash(\)
  - Double Quotes (“…”)
  - Single Quotes (‘…’)
- Backslash converts the literal characters into special characters and vice versa.

- Literal characters are interpreted in the normal, non-computer meaning
- Special character are identified as shell metacharacter
- Eg: < is idetified as “lessthan” in literal meaning and “input direction” as special character
- Suppose You need a message “Hello! Welcome” as such how is given
- Then you will write echo “Hello! Welcome” which will display only Hello! Welcome, Now to make the quotes as such as literal meaning include \ before the character “
- echo \“ Hello! Welcome\” will display as you expected.

- Single and double quotes has to be entered in pairs.
- Every starting quote should have an ending quote
- Double Quote will display as such how it is given
- Special characters will be treated as special characters
- Single Quote is like double quotes but it treats the special characters as literals
- Eg: a=“red”
- Echo “The colour I like is $a” will give an output as The colour I like is red
- Whereas in single quote, the output will be The colour I like is $a

Command Substitution

- Command Substitution can be done with ` ` (backquotes)
- This can be used to generate useful messages
- Eg: echo Todays date is `date`
- Eg: echo ‘There are `ls | wc –l ` files in the current directory’
- This will be more helpful while writing shell scripts

Job Control

- A job is a user task run on a computer
- Editing, sorting, printing, displaying, reading are all jobs
- A job is a set of command or commands entered on one command line.
- Eg: $ ls
- Unix is a multitasking operating system, hence it can run multiple jobs at a time
- Multitasking is supported in the form of
  - Foreground Jobs
  - Background Jobs

- **Foreground Jobs**
  - Jobs run under the supervision of the user
  - While it is running no other jobs can be started
  - It keeps control over the keyboard and the monitor
  - You can suspend the foreground job by giving ^z
  - Later it can be resumed back by giving **fg**
  - To cancel the job give **^c**
- **Background Jobs**
  - Jobs which take longer time to execute can be run in background
  - Background jobs do not control the keyboard and the monitor
  - However, they can send the results to monitor once the job is completed
  - The messages sent will be mingled with the foreground jobs

- To **suspend** background job – stop command
- To **restart** – bg
- To **terminate** – kill command
- All these commands require the job number. The job number has to be preceded by %
- Eg: $ kill %1 [here 1 represents the job number]

  $ stop %1
- To move jobs from **foreground to background**
  - Suspend the job first
  - bg

- To move jobs from **background to foreground**
  - Suspend the job first
  - fg %1
- Job numbers are related to the user session and the terminal
- But unix assigns another identification for the process, which is global. It is called **ProcessID.** Generally refered as **PID.**
- **ps** command displays all the process that is currently running in that terminal
- ps command displays the processid, terminal number, the cumulative execution time and command name.

Aliases

- Creating a customized command by assigning name to a command
- Usage: **$alias name=command-definition**
- Eg: $alias dir=ls
  $dir
- Alias can be given with options also
  - $alias dir='ls –l'
- Multiple Commands can also be given
  - $alias dir="ls –l | more"
- You can nest your alias
  - Alias dir=ls
  - Alias lndir='dir –l | more'
- You can give arguments to alias command
  - $dir f1.c
- Removing aliases – **unalias command**
  - unalias -a [deletes all aliases created]
  - unalias dir [deletes alias named dir]

Variables

- A variable is a location in memory where values can be stored.
- Variables can be
  - User defined
  - Predefined
- User variables are not separately defined In Unix. The first reference to variable establishes it.
- A variable will start with an alphabet or _. It can further be followed by alphabet or numerals.
- Predefined variables are used to configure the shell environment.

- The system variables store information about the environment.
- Assignment of a variable
  - $x=23
- To display the value of the variable
  - echo $x

Predefined Varaibles

- Predefined variables are of two categories
  - Shell variables
  - Environment variables
- Shell varaibles are used to customize the shell itself
- The environment variables are used to control user environment
- Environment variables can be exported to the subshells
- HOME, HOMEHISTFILE, PATH, SHELL, PS1, TERM
- To remove a variable
  - Unset var
- To display all variables
  - set

Shell Options

- noglob – Discards wildcard expansion
- verbose – prints commands before executing them
- xtrace – prints commands and arguments before executing them
- emacs – uses emacs for command-line editing
- ignoreeof – disallows ^d to exit the shell
- noclobber – Does not allow redirection to clobber existing file
- vi - uses vi for command-line editing

- Bash shell uses ~/.bash_profile, ~/.bash_login or ~/.profile
- Env file BASH_ENV variable
- Shutdown ~/.bash_logout.