Dr. Sanjay Adiwal
Joint Director
ITSS
C-DAC Bangalore

LINUX SHELL SCRIPTING

OUTLINE

- What is Kernel, Shell, Process and Redirectors
- Shell Programming
- Writing Shell Scripts
- Advanced Shell Scripts Commands

WHAT IS KERNEL?

- It manages resource of Linux O/S.
- Kernel decides who will use this resource, for how long and when.
- It performance following task:
  - I/O management
  - Process management
  - Device management
  - File management
  - Memory management

LINUX SHELL

- It's environment provided for user interaction.
- Shell is an command language interpreter that executes commands read from the standard input device (keyboard) or from a file.

Shell Name	Developed By	Where	Remark
BASH (Bourne-Again SHell)	Brian Fox and Chet Ramey	Free Software Foundation	Most common shell in Linux. It's Freeware shell.
CSH (C SHell)	Bill Joy	University of California (For BSD)	The C shell's Syntax and usage are very similar to the C programming language.
KSH (Korn SHell)	David Korn	AT & T Bell Labs	

WHAT IS PROCESSES?

- Process is any kind of program or task carried out by your PC.
- $ ls -lR, is command or a request to list files in a directory and all subdirectory in your current directory. So it's a process.
- A process is program (command given by user) to perform some Job.
- In Linux when you start process, it gives a number (called PID or process-id), PID starts from 0 to 65535.

LINUX PROCESS

- Linux is multi-user, multitasking o/s. It means you can run more than two process simultaneously if you wish.
- $ ls / -R | wc –l
- $ ls / -R | wc -l &
- An instance of running command is called process and the number printed by shell is called process-id (PID), this PID can be use to refer specific running process.

LINUX COMMAND RELATED WITH PROCESS

For this purpose	Use this Command	Example
To see currently running process	ps	$ps
To stop any process i.e. to kill process	Kill {PID}	$kill 1001
To get information about all running process	ps -ag	$ ps -ag
To stop all process except your shell	Kill 0	$kill 0
For background processing (With &, use to put particular command and program in background)	linux-command &	$ ls / -R | wc -l &

REDIRECTION OF STANDARD OUTPUT/INPUT

- Mostly all command gives output on screen or take input from keyboard, but in Linux it's possible to send output to file or to read input from file.
- For e.g. $ ls command gives output to screen; to send output to file of ls give command , $ ls > filename. It means put output of ls command to filename.
- There are three main redirection symbols >,>>,<

> REDIRECTOR SYMBOL

- Syntax: *Linux-command > filename*
- To output Linux-commands result to file. Note that If file already exist, it will be overwritten else new file is created.
- **$ ls > myfiles**
- Now if 'myfiles' file exist in your current directory it will be overwritten without any type of warning.

>> REDIRECTOR SYMBOL

- Syntax: *Linux-command >> filename*
- To output Linux-commands result to END of file. Note that If file exist , it will be opened and new information / data will be written to END of file, without losing previous information/data, And if file is not exist, then new file is created.
- **$ date >> myfiles**

< REDIRECTOR SYMBOL

- **Input redirection**
- Syntax: *Linux-command < filename*
- To take input to Linux-command from file instead of key-board. For e.g. To take input for cat command give
- **$ cat < myfiles**
- **# mail mike@somewhere.org < to_do**

PIPS

- A pipe is a way to connect the output of one program to the input of another program without any temporary file.
- A pipe is nothing but a temporary storage place where the output of one command is stored and then passed as the input for second command. Pipes are used to run more than two commands (Multiple commands) from same command line.
- $ls|more
- $ ls -l | wc -l

INTRODUCTION TO SHELL PROGRAMMING

- Shell program is series of Linux commands.
- Shell script can take input from user, file and output them on screen. Useful to create our own commands that can save our lots of time and to automate some task of day today life.
- Is a Text File.
- With Instructions.
- And Exicutables

WRITING A SCRIPT

- Use text editor to generate the “first” file

```
#!/bin/sh
# first
# this file looks for the files containing POSIX
# and print it
for file in *
do
    if grep -q POSIX $file
    then
       echo $file
    fi
done
exit 0
```

```
% /bin/sh first
% chmod +x first
%./first (make sure . is include in PATH parameter)
```

exit code, 0 means successful

SYNTAX

- Variables
- Conditions
- Control
- Lists
- Shell Commands
- Result
- Document

VARIABLES IN SHELL SCRIPT

- In Linux, there are two types of variable
- 1) System variables - Created and maintained by Linux itself. This type of variable defined in
- CAPITAL LETTERS.
- 2) User defined variables (UDV) - Created and maintained by user. This type of variable defined
- in lower LETTERS.

SOME SYSTEM VARIABLES

System Variable	Meaning
BASH=/bin/bash	Our shell name
BASH_VERSION=3.2.25(1)	Our shell version name
LOGNAME=students	Our logging name
OSTYPE=Linux	Our o/s type
PATH=/usr/bin:/sbin:/bin:/usr/sbin	Our path settings
PS1=[\u@\h \W]\$	Our prompt settings
PWD=/home/students/Common	Our current working directory
SHELL=/bin/bash	Our shell name
USERNAME=sanjay	User name who is currently login to this PC

USER DEFINED VARIABLES

- To define UDV use following syntax
- Syntax: *variablename=value*
- NOTE: Here 'value' is assigned to given 'variablename' and Value must be on right side = sign For
- **$ no=10 # this is ok**
- **$ 10=no # Error, NOT Ok, Value must be on right side of = sign.**

RULES FOR NAMING VARIABLE NAME

- Variable name must begin with Alphanumeric character or underscore character (_), followed by one or more Alphanumeric character. (e.g HOME, SYSTEM_VERSION, abc)
- Don't put spaces on either side of the equal sign when assigning value to variable.
  - **$ no=10 OK**
- Variables are case-sensitive, just like filename in Linux.
- You can define NULL variable as follows:
  - **$ vech=**
  - **$ vech="“**
- Do not use ?,* etc, to name your variable names.

PRINT OR ACCESS VALUE OF UDV

- To print or access UDV use following syntax
- Syntax: *$variablename*
- **$ echo $vech**
- How to Define variable x with value 10 and print it on screen.
- How to print sum of two numbers, let's say 6 and 3
- How to define two variable x=20, y=5 and then to print division of x and y
- Modify above and store division of x and y to variable called z

HOW TO WRITE SHELL SCRIPT

- Use VI editor
- $vi hello-world.sh
- #!/bin/bash
- echo “hello world”
- $chmod +x hello-world.sh
- $./hello-world.sh Or
- $sh hello-world.sh

ABOUT QUOTES

- There are three types of quotes:
  - “Double Quotes” - Anything enclose in double quotes removed meaning of that characters (except \ and $).
  - 'Single quotes' - Enclosed in single quotes remains unchanged.
  - `Back quote` - To execute command.
- **$ echo “Today is date”**
- **$ echo “Today is `date`”.**

SHELL ARITHMETIC

- Expr command
- expr 1 + 3
- $ expr 2 - 1
- $ expr 10 / 2
- $ expr 20 % 3 # remainder read as 20 mod 3 and remainder is 2)
- $ expr 10 * 3 # Multiplication use * not * since its wild card)
- $ echo `expr 6 + 3`

COMMAND LINE PROCESSING

- $command-name {arguments}
- $ls –l –t
- ls – command name
- -l – argument 1
- -t – argument 2
- Address or access command line argument in our script:

EXIT STATUS

- By default in Linux if particular command is executed, it return two type of values, (Values are used to see whether command is successful or not) if return value is zero (0), command is successful, if return value is nonzero (>0), command is not successful or some sort of error executing command/shell script. This value is know as Exit Status of that command.
- To determine this exit Status we use $? variable of shell.
- $ls myfile
- $echo $?

CONDITIONAL STATEMENT

- Syntax:
  - *if condition (Command or expression)*
  - *then*
  - *command1 if condition is true or if exit status*
  - *of condition is 0 (zero)*
  - *…*
  - *…*
  - *Fi*
- *Example*
  - *If test $? –eq 0 then echo “command executed successful”; else echo “Error in execution”; fi*

CONDITIONAL STATEMENT

Operator	Meaning	Mathematical Statements	But in Shell	
			For test statement with if command	For [expr] statement with if command
-eq	is equal to	5 == 6	if test 5 -eq 6	if expr [5 -eq 6]
-ne	is not equal to	5 != 6	if test 5 -ne 6	if expr [5 -ne 6]
-lt	is less than	5 < 6	if test 5 -lt 6	if expr [5 -lt 6]
-le	is less than or equal to	5 <= 6	if test 5 -le 6	if expr [5 -le 6]
-gt	is greater than	5 > 6	if test 5 -gt 6	if expr [5 -gt 6]
-ge	is greater than or equal to	5 >= 6	if test 5 -ge 6	if expr [5 -ge 6]

STRING COMPARISONS, LOGICAL OPERATORS

Operator	Meaning
string1 = string2	string1 is equal to string2
string1 != string2	string1 is NOT equal to string2
string1	string1 is NOT NULL or not defined
-n string1	string1 is NOT NULL and does exist
-z string1	string1 is NULL and does exist

Operator	Meaning
! expression	Logical NOT
expression1 -a expression2	Logical AND
expression1 -o expression2	Logical OR

TEST FOR FILE AND DIRECTORY TYPES

Test	Meaning
-s file	Non empty file
-f file	Is File exist or normal file and not a directory
-d dir	Is Directory exist and not a file
-w file	Is writeable file
-r file	Is read-only file
-x file	Is file is executable

IF…ELSE…FI

- Syntax:
  - *if condition*
  - *then*
  - *command1 if condition is true or if exit status*
  - *of condition is 0(zero)*
  - *…*
  - *…*
  - *else*
  - *command2 if condition is false or if exit status*
  - *of condition is >0 (nonzero)*
  - *…*
  - *…*
  - *Fi*
- *Examples*

MULTILEVEL IF-THEN-ELSE

- Syntax:
  - *if condition*
  - *then*
  - *condition is zero (true - 0)*
  - *execute all commands up to elif statement*
  - *elif condition1*
  - *condition1 is zero (true - 0)*
  - *execute all commands up to elif statement*
  - *elif condition2*
  - *condition2 is zero (true - 0)*
  - *execute all commands up to elif statement*
  - *else*
  - *None of the above condtion,condtion1,condtion2 are true (i.e.*
  - *all of the above nonzero or false)*
  - *execute all commands up to fi*
  - *Fi*
- *Examples*

LOOPS IN SHELL SCRIPTS

- Computer can repeat particular instruction again and again, until particular condition satisfies. A group of instruction that is executed repeatedly is called a loop.
- **for loop**
- Syntax:
  - *for { variable name } in { list }*
  - *do*
  - *execute one for each item in the list until the list is*
  - *not finished (And repeat all statement between do and done)*
  - *Done*
- *Examples*

WHILE LOOP

- Syntax:
  - *while [condition]*
  - *do*
  - *command1*
  - *command2*
  - *command3*
  - *..*
  - *....*
  - *Done*
- *Examples*
- Loop is executed as long as given condition is true.