Bash Scripting

Fold

Table of Contents

Introduction

Linux Shell

Computer understand the language of 0's and 1's called binary language. Shell is a special program which accepts instruction or commands in English and if its a valid command, it passes the command to the kernel. Shell is not part of system kernel, but uses the system kernel to execute programs, create files etc.

Shell Script

Normally shells are interactive. It means shell accept command from you (via keyboard) and execute them. But if you use command one by one (sequence of 'n' number of commands) , the you can store this sequence of command to text file and tell the shell to execute this text file instead of entering the commands. This is know as shell script.

How to write shell script

- Write shell script with a text example (for example vi).
- set execute permission: `chmod +x your-script-name`
- Execute script:`bash your-script-name` or `./your-script-name`

Example
$vi first.sh

```
#
# My first shell script
#
clear
echo "first shell script"
```

$chmod +x first.sh
$./first.sh

Basics

Variables

In Linux (Shell), there are two types of variable:

- **System variables** - Created and maintained by Linux itself. This type of variable defined in CAPITAL LETTERS.
- **User defined variables** (UDV) - Created and maintained by user. This type of variable defined in lower letters.

```
#!/bin/bash

x=10            #NOT x = 10 no spaces
X=20            #variables are case sensitive
$y=             #NULL variable
echo "x = $x"
echo "X = $X"
echo "y = $y"
```

Comments

```
#!/bin/bash
# This line is a comment.
echo "A comment will follow." # Comment here.
echo "The # here does not begin a comment."
echo 'The # here does not begin a comment.'
echo The \# here does not begin a comment.
echo The # here begins a comment.

echo ${PATH#*:}       # Parameter substitution, not a comment.
echo $(( 2#101011 ))  # Base conversion, not a comment.
```

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

Shell Built in Variables

~ Shell Built in Variables	~ Meaning
$#	Number of command line arguments.
$?	Exit Status
$*	string that contains all arguments to shell
$@	Same as above, except when quoted.
$-	Option supplied to shell
$$	PID of shell
$!	PID of last started background process (started with &)

Input - Output redirection

Redirection symbols

- **>**: To output Linux-commands result to file.
- »: To output Linux-commands result to END of file.
- **<**: To take input to Linux-command from file instead of keyboard.

Example 1

```
#!/bin/bash
clear
echo "ls > file_list"
ls > file_list

echo "ls -la >> file_list"
ls -la >> file_list

echo "cat < file_list"
cat < file_list
```

Example 2
$cat > sname
vivek
ashish
zebra
babu
Press CTRL + D to save.
$ sort < sname > sorted_names
$ cat sorted_names

Example 3
$ tr "[a-z]" "[A-Z]" < sname > cap_names
$ cat cap_names

Example 4
$ sort > new_sorted_names < sname
$ cat new_sorted_names

Pipes

A pipe is a way to connect the output of one program to the input of another program without any temporary file.

Example
$ ls l
$ who l sort
$ who l sort > user_list
$ who l wc -l
$ ls -l l wc -l
$ who l grep chrys

Filter

If a Linux command accepts its input from the standard input and produces its output on standard output is know as a filter. A filter performs some kind of process on the input and gives output.
Example
Suppose you have file called 'hotel.txt' with 100 lines data, And from 'hotel.txt' you would like to print contains from line number 20 to line number 30 and store this result to file called 'hlist' then give command:
$ tail +20 < hotel.txt l head -n30 >hlist

Processes

A process is program (command given by user) to perform specific Job. In Linux when you start process, it gives a number to process (called PID or process-id), PID starts from 0 to 65535.

Language Constructs

if condition

Example 1: Mathematical Operators

```
#!/bin/bash

if test $1 -gt 0
then
        echo "$1 > 0"
fi

if test $1 -ge 0
then
        echo "$1 >= 0"
fi

if test $1 -eq 0
then
        echo "$1 == 0"
fi

if test $1 -ne 0
then
        echo "$1 != 0"
fi

if test $1 -lt 0
then
        echo "$1 < 0"
fi

if test $1 -le 0
then
        echo "$1 <= 0"
fi
```

Example 2: Logical Operators

```
if test $1 -lt 0
then
        echo "$1 < 0"
fi

if test $1 -le 0
then
        echo "$1 <= 0"
fi
```

Example 3: String Operators

```
string_null=""
string1="string1"

if [ $string_null -n ]
```

```
then
        echo "not null string"
else
        echo "null string"
fi

if [ $string_null -z ]
then
        echo "null string"
else
        echo "not null string"
fi

if [ "$string_null" == "$string1" ]
then
        echo "strings equal"
else
        echo "strings not equal"
fi

if [ "$string_null" != "$string1" ]
then
        echo "strings not equal"
else
        echo "strings equal"
fi
```

Example 4: Test for files and directories

```
#!/bin/bash

if test -s $1
then
        echo "$1 not empty file"
fi

if test -f $1
then
        echo "$1 normal file. Not a directory"
fi
if test -e $1
then
    echo "$1 exists"
fi

if test -d $1
then
        echo "$1 is directory and not a file"
fi

if test -r $1
then
        echo "$1 is read-only file"
fi

if test -x $1
then
        echo "$1 is executable"
fi
```

if...else...fi

If given condition is true then command1 is executed otherwise command2 is executed.

```
#!/bin/sh
#
# Script to see whether argument is positive or negative
#
if [ $# -eq 0 ]
then
echo "$0 : You must give/supply one integers"
exit 1
fi

if test $1 -gt 0
then
echo "$1 number is positive"
else
echo "$1 number is negative"
fi
```

Multilevel if-then-else

```
#
#!/bin/sh
# Script to test if..elif...else
#
if [ $1 -gt 0 ]; then
```

```
  echo "$1 is positive"
elif [ $1 -lt 0 ]
then
  echo "$1 is negative"
elif [ $1 -eq 0 ]
then
  echo "$1 is zero"
else
  echo "Opps! $1 is not number, give number"
fi
```

Loops

for loops

Example 1

```
$ cat > testfor
for i in 1 2 3 4 5
do
echo "Welcome $i times"
done
```

Example 2

```
#!/bin/sh
#
#Script to test for loop
#
#
if [ $# -eq 0 ]
then
echo "Error - Number missing form command line argument"
echo "Syntax : $0 number"
echo "Use to print multiplication table for given number"
exit 1
fi
n=$1
for i in 1 2 3 4 5 6 7 8 9 10 #or for ((  i =     0 ;  i <= 10;  i++  ))
do
echo "$n * $i = `expr $i \* $n`"
done
```

while loops

```
#!/bin/sh
#
#Script to test while statement
#
if [ $# -eq 0 ]
then
   echo "Error - Number missing form command line argument"
   echo "Syntax : $0 number"
   echo " Use to print multiplication table for given number"
exit 1
fi
n=$1
i=1
while [ $i -le 10 ]
do
  echo "$n * $i = `expr $i \* $n`"
  i=`expr $i + 1`
done
```

case

```
# if no vehicle name is given
# i.e. -z $1 is defined and it is NULL
#
# if no command line arg are sdf

if [ -z $1 ]
then
  rental="*** Unknown vehicle ***"
elif [ -n $1 ]
then
# otherwise make first arg as rental
  rental=$1
fi

case $rental in
   "car") echo "For $rental Rs.20 per k/m";;
   "van") echo "For $rental Rs.10 per k/m";;
   "jeep") echo "For $rental Rs.5 per k/m";;
   "bicycle") echo "For $rental 20 paisa per k/m";;
```

```
   *) echo "Sorry, I can not gat a $rental for you";;
esac
```

Debugging Shell Scripts

-v Print shell input lines as they are read.
-x After expanding each simple-command, bash displays the expanded value of PS4 system variable, followed by the command and its expanded arguments.

Advanced Features

Local and Global Shell variables

Local variable can be used in same shell only.
Global variables or environment variables are available in all shells. Commands env or printenv can be used to display environment variables.

Functions

Function is series of instruction/commands. Function performs particular activity in shell i.e. it had specific work to do or simply say task.

```
sum()
{
    if [ -z "$2" ]; then
        echo $1
    else
        a=$1;
        shift;
        b=`sum $@`
        echo `expr $a + $b`
    fi
}
```

Reading from the shell

Example 1

```
#/bin/sh

echo "Name?"
read name
echo "Age?"
read age
echo "Hello $name, you are $age years old"
```

Example 2

```
# Script to create simple menus and take action according to that selected
# menu item
#
while :
 do
    clear
    echo "-------------------------------------"
    echo " Main Menu "
    echo "-------------------------------------"
    echo "[1] Show Todays date/time"
    echo "[2] Show files in current directory"
    echo "[3] Show calendar"
    echo "[4] Start editor to write letters"
    echo "[5] Exit/Stop"
    echo "======================"
    echo -n "Enter your menu choice [1-5]: "
    read yourch
    case $yourch in
      1) echo "Today is `date` , press a key. . ." ; read ;;
      2) echo "Files in `pwd`" ; ls -l ; echo "Press a key. . ." ; read ;;
      3) cal ; echo "Press a key. . ." ; read ;;
      4) vi ;;
      5) exit 0 ;;
      *) echo "Opps!!! Please select choice 1,2,3,4, or 5";
         echo "Press a key. . ." ; read ;;
 esac
done
```

http://freeos.com/guides/lsst/ch04sec12.html
http://stackoverflow.com/questions/603696/linux-command-line-best-practices-and-tips
http://www.commandlinefu.com/commands/browse

Bibliography

1. Advanced Bash-Scripting Guide

Recommended Books for Bash Scripting