Linux Summer Camp 2017

Writing Bash Shell Script

Part 4. Writing Shell Scripts from *The Linux Command Line,* 3rd Internet edition *by* William Shotts

Shin Hong

Environment

- The environment is a set of name-value pairs stored in a shell session for configuration
  - established by start-up files such as `.bashrc`, `.bash_profile`

- Commands
  - set
  - env, printenv
  - export
  - echo
  - alias

Shell Script

- A shell like Bash is both a command line interface and a script language interpreter.
  - A shell script is a text file containing a series of commands.
  - A shell allows a user to compose a complicated operation by joining multiple programs

- Steps
  1. Write a script
  2. Make the script executable
  3. Put the script somewhere the shell can find

Example

```
#!/bin/bash

# This is our first script.

echo 'Hello World!'
```

```
[me@linuxbox ~]$ ls -l hello_world
-rw-r--r-- 1 me      me      63 2009-03-07 10:10 hello_world
[me@linuxbox ~]$ chmod 755 hello_world
[me@linuxbox ~]$ ls -l hello_world
-rwxr-xr-x 1 me      me      63 2009-03-07 10:10 hello_world
```

```
[me@linuxbox ~]$ ./hello_world
Hello World!
```

```
export PATH=~/bin:"$PATH"
```

Assignments

- *Variable=value*
- A variable *V* can be retrieved by $*V* and ${*V*}

```
a=z                     # Assign the string "z" to variable a.
b="a string"            # Embedded spaces must be within quotes.
c="a string and $b"     # Other expansions such as variables can be
                        # expanded into the assignment.
d=$(ls -l foo.txt)      # Results of a command.
e=$((5 * 7))            # Arithmetic expansion.
f="\t\ta string\n"      # Escape sequences such as tabs and newlines.
```

Example

```
#!/bin/bash

# Program to output a system information page

TITLE="System Information Report For $HOSTNAME"
CURRENT_TIME=$(date +"%x %r %Z")
TIMESTAMP="Generated $CURRENT_TIME, by $USER"

echo "<HTML>
        <HEAD>
                <TITLE>$TITLE</TITLE>
        </HEAD>
        <BODY>
                <H1>$TITLE</H1>
                <P>$TIMESTAMP</P>
        </BODY>
</HTML>"
```

Here Document

```
#!/bin/bash

# Script to retrieve a file via FTP

FTP_SERVER=ftp.nl.debian.org
FTP_PATH=/debian/dists/lenny/main/installer-i386/current/images/cdrom
REMOTE_FILE=debian-cd_info.tar.gz

ftp -n << _EOF_
open $FTP_SERVER
user anonymous me@linuxbox
cd $FTP_PATH
hash
get $REMOTE_FILE
bye
_EOF_
ls -l $REMOTE_FILE
```

Functions

- Shell functions are located inside other scripts and act as autonomous programs.
  - A local variable can be declared for a function scope

```
function name {
      commands
      return
}


name () {
      commands
      return
}
```

```
#!/bin/bash

# local-vars: script to demonstrate local variables

foo=0    # global variable foo

funct_1 () {

    local foo # variable foo local to funct_1

    foo=1
    echo "funct_1: foo = $foo"
}

funct_2 () {

    local foo # variable foo local to funct_2

    foo=2
    echo "funct_2: foo = $foo"
}

echo "global:  foo = $foo"
funct_1
```

Branching with If

```
if commands; then
     commands
[elif commands; then
     commands...]
[else
     commands]
fi
```

```
[me@linuxbox ~]$ x=5
[me@linuxbox ~]$ if [ $x -eq 5 ]; then echo "equals 5"; else echo
"does not equal 5"; fi
equals 5
[me@linuxbox ~]$ x=0
[me@linuxbox ~]$ if [ $x -eq 5 ]; then echo "equals 5"; else echo
"does not equal 5"; fi
does not equal 5
```

Test

- *test expression* or *[expression]*

```
test_file () {

    # test-file: Evaluate the status of a file

    FILE=~/.bashrc

    if [ -e "$FILE" ]; then
         if [ -f "$FILE" ]; then
              echo "$FILE is a regular file."
         fi
         if [ -d "$FILE" ]; then
              echo "$FILE is a directory."
         fi
         if [ -r "$FILE" ]; then
              echo "$FILE is readable."
         fi
         if [ -w "$FILE" ]; then
              echo "$FILE is writable."
         fi
         if [ -x "$FILE" ]; then
              echo "$FILE is executable/searchable."
         fi
    else
         echo "$FILE does not exist"
         return 1
    fi

}
```

File Expression (1/2)

Expression	Is True If:
`file1 -ef file2`	*file1* and *file2* have the same inode numbers (the two filenames refer to the same file by hard linking).
`file1 -nt file2`	*file1* is newer than *file2*.
`file1 -ot file2`	*file1* is older than *file2*.
`-b file`	*file* exists and is a block-special (device) file.
`-c file`	*file* exists and is a character-special (device) file.
`-d file`	*file* exists and is a directory.
`-e file`	*file* exists.
`-f file`	*file* exists and is a regular file.
`-g file`	*file* exists and is set-group-ID.
`-G file`	*file* exists and is owned by the effective group ID.
`-k file`	*file* exists and has its “sticky bit” set.

File Expression (2/2)

`-L` *file*	*file* exists and is a symbolic link.
`-O` *file*	*file* exists and is owned by the effective user ID.
`-p` *file*	*file* exists and is a named pipe.
`-r` *file*	*file* exists and is readable (has readable permission for the effective user).
`-s` *file*	*file* exists and has a length greater than zero.
`-S` *file*	*file* exists and is a network socket.
`-t` *fd*	*fd* is a file descriptor directed to/from the terminal. This can be used to determine whether standard input/output/error is being redirected.
`-u` *file*	*file* exists and is setuid.
`-w` *file*	*file* exists and is writable (has write permission for the effective user).
`-x` *file*	*file* exists and is executable (has execute/search permission for the effective user).

String Expression

Expression	Is True If...
`string`	*string* is not null.
`-n string`	The length of *string* is greater than zero.
`-z string`	The length of *string* is zero.
`string1 = string2` `string1 == string2`	*string1* and *string2* are equal. Single or double equal signs may be used, but the use of double equal signs is greatly preferred.
`string1 != string2`	*string1* and *string2* are not equal.
`string1 > string2`	*string1* sorts after *string2*.
`string1 < string2`	*string1* sorts before *string2*.

Integer Expression

Expression	Is True If...
`integer1 -eq integer2`	*integer1* is equal to *integer2*.
`integer1 -ne integer2`	*integer1* is not equal to *integer2*.
`integer1 -le integer2`	*integer1* is less than or equal to *integer2*.
`integer1 -lt integer2`	*integer1* is less than *integer2*.
`integer1 -ge integer2`	*integer1* is greater than or equal to *integer2*.
`integer1 -gt integer2`	*integer1* is greater than *integer2*.

Different Versions of Test

```
#!/bin/bash

# test-integer2: evaluate the value of an integer.

INT=-5

if [[ "$INT" =~ ^-?[0-9]+$ ]]; then
    if [ $INT -eq 0 ]; then
        echo "INT is zero."

[me@linuxbox ~]$ if [[ $FILE == foo.* ]]; then
> echo "$FILE matches pattern 'foo.*'"
> fi
foo.bar matches pattern 'foo.*'

[me@linuxbox ~]$ if ((1)); then echo "It is true."; fi
It is true.
[me@linuxbox ~]$ if ((0)); then echo "It is true."; fi
[me@linuxbox ~]$
```

Logical Operators As Branching

```
[me@linuxbox ~]$ mkdir temp && cd temp

[me@linuxbox ~]$ [[ -d temp ]] || mkdir temp

[me@linuxbox ~]$ [ -d temp ] || exit 1
```

While Loop (1/3)

```
#!/bin/bash

# while-count: display a series of numbers

count=1

while [[ $count -le 5 ]]; do
    echo $count
    count=$((count + 1))
done
echo "Finished."
```

While Loop (2/3)

```
#!/bin/bash

# while-menu2: a menu driven system information program

DELAY=3 # Number of seconds to display results

while true; do
    clear
    cat <<- _EOF_
         Please Select:

         1. Display System Information
         2. Display Disk Space
         3. Display Home Space Utilization
         0. Quit

    _EOF_
    read -p "Enter selection [0-3] > "

    if [[ $REPLY =~ ^[0-3]$ ]]; then
         if [[ $REPLY == 1 ]]; then
              echo "Hostname: $HOSTNAME"
              uptime
              sleep $DELAY
              continue
         fi
         if [[ $REPLY == 2 ]]; then
              df -h
              sleep $DELAY
              continue
         fi
         if [[ $REPLY == 3 ]]; then
              if [[ $(id -u) -eq 0 ]]; then
                   echo "Home Space Utilization (All Users)"
                   du -sh /home/*
              else
                   echo "Home Space Utilization ($USER)"
                   du -sh $HOME
              fi
              sleep $DELAY
              continue
         fi
         if [[ $REPLY == 0 ]]; then
              break
         fi
    else
         echo "Invalid entry."
         sleep $DELAY
    fi
done
echo "Program terminated."
```

While Loop (3/3)

```
#!/bin/bash

# while-read: read lines from a file

while read distro version release; do
    printf "Distro: %s\tVersion: %s\tReleased: %s\n" \
         $distro \
         $version \
         $release
done < distros.txt
```