Shell Scripts

Introduction

Shell scripts are text files that automate a series of UNIX environment-based commands that otherwise must be performed one at a time. Shell scripts are often used to automate command sequences that repeat, such as services that start or stop on system start up or shut down.

Any command that can be performed from the command line, such as ls, can be included in a shell script. Similarly, any command that can be included in a shell script can be performed on the UNIX environment command line.

Users with little or no programming experience can create and run shell scripts. You initiate the sequence of commands in the shell script by simply entering the name of the shell script on a command line.

Determining the Type of Shell to Run a Shell Script

There are several different shells available in the UNIX OS. Two of the most commonly used shells are the Bourne shell and the Korn shell.

To ensure that the correct shell is used to run a shell script, the first line of the script should always begin with the characters #!, followed immediately by the absolute path name of the shell required to run the script. These must be the only characters on the first line of the file.

#!/*full-pathname-of-shell*

For example:

```
#!/bin/sh
```

or

```
#!/bin/ksh
```

Comments

Comments are text entries that often provide information about a shell script. They are inserted into a shell script but have no effect on the script itself. Comments are ignored by the shell and are solely for the benefit of the user.

Comments are preceded by the hash (#) character. Whenever the shell encounters a word beginning with the # character it ignores all text on that line.

For example:

```
# this is a comment
ls -l                    # list the files in a directory
```

Introduction

The shell interprets shell scripts line by line. Shell scripts are not compiled into binary form. Because shell scripts have to be read line by line when they are run, the user must have read permissions to be able to run a shell script.

For example, to grant read permissions to the mycmd user type:

```
$ chmod u+rx mycmd
```

When a shell script is running, any applied changes occur in the sub-shell or child process. A sub-shell cannot change the values of a variable in the parent shell, or its working directory.

```
$ cat myvars
echo running myvars
FMHOME=/usr/frame
MYBIN=/export/home/user1/bin
$ ls -l myvars
-rw-r--r--   1 user1    other        65 Sep 15 16:14 myvars
$ chmod u+x myvars
$ ls -l myvars
-rwxr--r--   1 user1    other        65 Sep 15 16:14 myvars
$ mywars
running myvars
```

After running the script, the FMHOME and MYBIN variables are not available because the script is run in a sub-shell.

```
$ echo $FMHOME
$
```

One of the most frequently used shell scripts is a users initialization file (~/.profile). This script is specifically designed to set the working environment of the user in the current shell.

If you made changes to your .profile you need to implement these changes in the current shell without logging out and logging back in. The dot (.) command performs the commands in the specified script in the current shell, as if the commands were entered on the command line.

```
$ . myvars running myvars
$ echo $FMHOME
/usr/frame
```

Passing Values to a Shell Script

Introduction

Shell scripts become more useful when you pass values to them while you run them. When you run a shell script and pass values to it on the command line, the shell stores the first word after the script name in the $1 variable, the second in the $2 variable, and so on. These special variables ($1, $2, and so on) are called *positional parameters*, and they are very useful to verify that the user passed the correct number of values when the script was run.

For example:

```
$ cat greetings
#!/bin/sh
echo          #echo the first two parameters passed
```

Add execute permissions to greetings.

```
$ chmod u+x greetings
```

Run greetings while passing the hello and world values.

```
$ greetings hello world
hello world
```

The shift Command

In the Bourne and Korn shells you can pass as many values as necessary on the command line. However, the Bourne shell accepts only a single number after the $ sign. An attempt to access the value in the tenth argument by using the notation 0 results in the value of followed by a zero (0).

The shift command enables you to shift your positional parameter values back by one position. For example, the value of the parameter becomes assigned to the parameter. Similarly, the value of the parameter becomes assigned to the parameter, and so on.

Note: In the Korn shell, you can access the tenth parameter directly by referring to ${10}.

Checking the Exit Status

All commands in the UNIX environment return an exit status. This numeric value is used to indicate the success or failure of a command. A value of zero indicates success. A non-zero value indicates failure. This non-zero value can be any integer in the range of 1-255.

The program developer can use the exit status values to indicate different error situations. The exit status of the last command performed in the foreground is held in the $? special shell variable, and can be tested by using the echo command.

For example:

```
$ grep other /etc/group
other::1:
$
$ echo $?
0
$
$ grep others /etc/group
$ echo $?
1
2
$
```

Using the test Command

Introduction

The test command is used for testing conditions. This command is very useful in shell scripts. The test command can be used to verify many conditions, including:

- Variable contents
- File access permissions
- File types

The test command can be written as *test expression* or by using the [*expression*] special notation.

The test command does not return any output. If the condition being tested is true, the exit status of the test command is set to 0. If the condition being tested is false, the exit status is set to 1.

Examples of the test command include the following:

- Test if the value of the LOGNAME variable is user1.
- $ **echo $LOGNAME**
- user1
-
- $ **test "$LOGNAME" = "user1"**
- $ **echo $?**

  0

- Test if the value of the LOGNAME variable is user1 by using the [*expression*] notation.
- $ **echo $LOGNAME**
- user1
-
- $ [**"$LOGNAME" = "user1"**]
- $ **echo $?**

  0

- Test if the user has read permissions on the /etc/group file.
- $ **ls -l /etc/group**
- -rw-r--r-- 1 root sys 290 Sep 13 15:14 /etc/group
-
- $ **test -r /etc/group**
- $ **echo $?**

0

- Test if the user has read permissions on the /etc/group file by using the [*expression*] notation.
- $ **ls -l /etc/group**
- -rw-r--r-- 1 root sys 290 Sep 13 15:14 /etc/group
-
- $ **[-r /etc/group]**
- $ **echo $?**

0

- Determine if /etc is a directory.
- $ **ls -ld /etc**
- drwxr-xr-x 53 root sys 3584 Sep 18 11:48 /etc
-
- $ **test -d /etc**
- $ **echo $?**

0

- Determine if /etc is a directory using the [*expression*] notation.
- $ **[-d /etc]**
- $ **echo $?**

0

- Compare the result against a known file.
- $ **test -d /etc/group**
- $ **echo $?**

1

- Compare against a known file using the [*expression*] notation.
- $ **[-d /etc/group]**
- $ **echo $?**

1

Executing Conditional Commands

Introduction

The shell provides two special constructs that enable you to perform a command based on whether a proceeding command succeeds or fails.

The && construct ensures that a command is performed only if the preceding command succeeds.

For example:

$ **mkdir $HOME/newdir && cd $HOME/newdir**

The || construct ensures that a command is performed only if the preceding command fails.

For example:

$ **mkdir /usr/tmp/newdir || mkdir $HOME/newdir**

Using the if Command

The if command evaluates the exit status of a command and initiates additional actions based on the returned value. The if command syntax is as follows:

$ **if** *command1*
> **then**
> **execute** *command2*
> **else**
> **execute** *command3*
> **fi**

If the exit status is zero, any commands that follow the then statement are performed. If the exit status is non-zero, any commands that follow the else statement are performed.

The if command is always closed with the fi statement. The if command is often used in conjunction with the test command.

Examples of the if command include display a greetings message:

$ **id**
uid=101(frame) gid=1(other)
$

$ **if test "$LOGNAME" = root**
> **then echo Hello System Administrator**
> **else**
> **echo Hello "$LOGNAME"**
> **fi**
Hello frame

$ **if ["$LOGNAME" = "root"]**
> **then echo hello System Administrator**
> **else**
> **echo hello "$LOGNAME"**
> **fi**
hello frame

Confirm that the user has read permissions for the /etc/group file.

```
$ if test -r /etc/group
> then
> echo "You have read permission on /etc/group"
> else
> echo "Sorry unable to read /etc/group file"
> fi
You have read permission on for the /etc/group file

$ if [ -r /etc/group ]
> then
> echo "You have read permission on /etc/group"
> else
> echo "Sorry unable to read /etc/group file"
> fi
You have read permission on for the /etc/group file
```

Determine if a file is a directory.

```
$ ls -ld /etc
drwxr-xr-x  53 root     sys         3584 Sep 18 11:48 /etc

$ if test -d /etc
> then
> echo /etc is a directory
> else
> echo /etc is not a directory
> fi
/etc is a directory

$ if [ -d /etc ]
> then
> echo /etc is a directory
> else
> echo /etc is not a directory
> fi
/etc is a directory

$ if test -d /etc/group
> then
> echo /etc is a directory
> else
> echo /etc is not a directory
> fi
/etc is not a directory
```

Executing Conditional Commands

Using the while Command

The while command enables you to repeat a command or group of commands. The while command syntax is as follows:

```
$ while command1
> do
> command2
> done
```

In this example, the while command evaluates the exit status of the command1 command that follows it.

If the value is zero, any commands that follow the do statement are performed, command1 is run again, and the exit status checked again.

If the exit status of command1 is non-zero, the loop terminates.

For example, use the set command to assign values to the positional parameters as follows:

```
$ set this is a while loop
$ echo $*
this is a while loop

$ while [ $# -gt 0 ]
> do
> echo
> shift
> done
this
is
a
while
loop
```

Using the case Command

The case command compares a single value against other values, and performs a command or group of commands when a match is found. The case command syntax is as follows:

```
$ case value in
> pat1)     command
> command
> ...
> command
> ;;
> pat2)     command
> command
> ...
> command
> ;;
> ...
> patn)     command
> command
> ...
> command
> ;;
> esac
```

When a match is found and the respective commands are performed, no other patterns are checked.

For example:

```
#!/sbin/sh
#
# Copyright 2003 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# ident     "@(#)volmgt             1.7         03/12/09 SMI"

$ case "" in
> 'start')
>         if [ -f /etc/vold.conf -a -f /usr/sbin/vold -a \
>         "${_INIT_ZONENAME:=&backquot;/sbin/zonename&backquot;}" = "global" ]; then
>                 echo 'volume management starting.'
>                 /usr/sbin/vold >/dev/msglog 2>&1 &
>         fi
>         ;;
>
> 'stop')
>         /usr/bin/pkill -x -u 0 vold
>         ;;
>
>         *)
>         echo "Usage: PAGECONTENT { start | stop }"
>         exit 1
>         ;;
>
> esac
exit 0
```

As an example:

```
case $1 in
'sus')
echo " This is right "
;;
'shan')
echo " This is not right "
;;
*)
echo " Please input correct one"
;;
esac
```

Class Lab Work:

shellpractice.sh

```
#!/bin/bash
echo "This is my first Shell Script"
capture_out=`date`
echo $capture_out
num=1
if [ $# -lt $num ]
then
echo "Use <script_name> <argument>"
exit
fi

echo "Shell Script File Name is: " $0
echo "The given first argument is: " $1

echo "printing common shell variable" $LOGNAME $HOME $HADOOP_HOME
#Let us check Piping
cat /etc/passwd | awk -F: {'print $1'} | sort -n > userlist.txt

#While Loop Testing
set this is a while loop
echo $*
while [ $# -gt 0 ]
do
echo
shift
done

#For Loop Testing
set this is for loop
for i in $*
do
echo $i
done

#Accessing files under a directory
for file in /home/hadoop/*
do
echo $file
done

#Testing much more mature for
for((c=1;c<=5;c++))
do
echo "Welcom Number $c"
done
```

```
#Changing Datetime format
changedname=`date '+%Y%m%d%H%M%S'`
echo $changedname

#Retention of log files generated by Spark in /tmp directory
find /tmp/spark-events/ -mtime +15 -exec rm -rf {} \;
```

More Example Cases:

```
#More mature loop test for while
#!/bin/sh
INPUT_STRING=hello
while [ "$INPUT_STRING" != "bye" ]
do
  echo "Please type something in (bye to quit)"
  read INPUT_STRING
  echo "You typed: $INPUT_STRING"
done

#!/bin/sh
while :
do
  echo "Please type something in (^C to quit)"
  read INPUT_STRING
  echo "You typed: $INPUT_STRING"
done

while read f
do
  case $f in
        hello)          echo English    ;;
        howdy)          echo American   ;;
        gday)           echo Australian ;;
        bonjour)        echo French     ;;
        "guten tag")    echo German     ;;
        *)              echo Unknown Language: $f
                ;;
   esac
done < myfile


#Testing nomal for
for i in 1 2 3
do
echo $i
done
```

```
#Testing for to see all files in a directory
for i in *
do
echo $i
done
#Testing much more mature for
for((c=1;c<=5;c++))
do
echo "Welcom Number $c"
done
#Testing for loop for infinate loop
for ((;;))
do
echo "Press Ctrl+C to quit"
sleep 10
done
#Following script will go through a specific directory & will stop when it gets a file named
/etc/resolv.conf and count the number of name server

for file in /etc/*
do
 if [ "${file}" == "/etc/resolv.conf" ]
 then
   echo $file
   countofnameserver=`grep -c nameserver /etc/resolv.conf`
   echo "Count of name server is: $countofnameserver"
   break
  fi
done
set a.txt b.txt c.bak
for i in $*
do
 if [ ${i} == "c.bak" ]
 then
   echo "This is c.bak file... $i let's continue... for other files"
   continue
 fi
echo "This is not a c.bak file, let's do ops for $i"
done
```