Shell Programming

Dr. Alekha Kumar Mishra

Shell programming/scripting

- A group of commands that are executed regularly are better stored in a file and executed when needed
- All such files are called shell scripts, shell programs, or shell procedures.
- There is no restriction on filenames extension, but it is universal convention that the extension .sh to be used for shell programs
- To run a shell script : $ sh filename.sh ↵

example

```
# example 1
date
ls -l
Echo ‘Welcome !! Shell User’
```

```
alekha@alekha-OptiPlex-3060:~/shellprog$ sh example1.sh
Wed Aug 26 13:15:08 IST 2020
total 80
drwxr-xr-x 13 alekha alekha 4096 Oct 17  2019 contiki
drwxr-xr-x  2 alekha alekha 4096 Feb 18  2020 cppprog
-rw-rw-r--  1 alekha alekha   35 Mar 11 11:57 datasheet.txt
drwxr-xr-x  2 alekha alekha 4096 Aug 25 13:43 Desktop
drwxr-xr-x 12 alekha alekha 4096 Aug 22 09:43 Documents
drwxr-xr-x  2 alekha alekha 4096 Aug 26 10:37 Downloads
-rw-r--r--  1 alekha alekha 8980 Oct 16  2019 examples.desktop
drwxr-xr-x  3 alekha alekha 4096 Aug 26 09:33 javaprogs
drwxr-xr-x  3 alekha alekha 4096 Jan 16  2020 Music
drwxr-xr-x  2 alekha alekha 4096 Oct 15  2019 nwp
drwxr-xr-x  3 alekha alekha 4096 Aug 26 12:42 Pictures
drwxr-xr-x  2 alekha alekha 4096 Oct 17  2019 Public
drwxr-xr-x  7 alekha alekha 4096 Aug  4 13:25 pyproj
drwxr-xr-x  3 alekha alekha 4096 Aug 26 13:14 shellprog
drwxr-xr-x  6 alekha alekha 4096 Jan 29  2020 snap
drwxr-xr-x  2 alekha alekha 4096 Oct 17  2019 Templates
drwxr-xr-x  2 alekha alekha 4096 Aug 22 09:45 Videos
drwxr-xr-x  3 alekha alekha 4096 Aug 15  2019 xfigfles
welcome !! Shell User
```

Reading input from the user

```
# example2.sh
# The read statement is used to take input from the user

echo “\n enter the pattern to search\c”
read pattern
echo “\n enter the filename \c”
read fname
grep “$pattern” $fname
echo “\n Result is shown above”
```

Assigning values to a variable

$ name=alekha ↵

$ age=40 ↵

$ mypath=/home/alekha ↵

- There must be no space on both the side of = sign
- While refering a variable the variable name should be prefixed by ‘$’ symbol.
  - $ echo $name ↵
- To wipe a variable created, use unset command
  - $ unset name ↵

System variables

- There are system variables of OS that can be accessed and modified (using set command)
- PS1 – system prompt 1
- PS2 – system prompt 2
- HOME – home directory path
- PATH – all executable paths
- IFS – inter field separator (black space by default)

Command line arguments (positional parameters)

- The first argument is referred by **$1**, second argument is referred by **$2**, and so on
- Command name **- $0**
- Total no. of arguments **- $#**
- The complete list of arguments **- $***
- **$?** is the parameter that stores the exit status of the last command (0- success, 1- failure, or may hold an error number)

```
# command line argument example
echo "program: $0
The number of arguments : $#
The arguments are : $*"
grep "$1" $2
echo "\n Job Over!"
```

Logical operators (&& and ||)

- When && used with commands as **command1 && command2**, the command2 is executed only if command1 is executed successful.
- When || is used as **command1 || command2** command2 is executed only if command1 fails.
- Examples

  $ cat xyz.txt || echo “file not found” ↵

  $ who && echo $?

- exit statement is used to prematurely terminate a program. exit argument is optional, when assigned/provided; it is assigned to $?

If condition statements

```
if condition
then
        execute commands
fi
```

```
if condition
then
        execute commands
else
        execute commands
fi
```

```
#conditional statement

echo "\n enter the source and target file\c"
read source target
if cp $source $target
then
        echo "File copied successfully"
else
        echo "Failed to copy the given file"
fi
```

Numeric comparison

- The test command is used for numeric comparison
- It is confined to integer comparison only
- Decimal values in a variable are simply trucated
- Requires the use of word ‘test’ with relational operation options.

Options	
-eq	equal to
-ne	not equal to
-gt	greater than
-lt	less than
-le	less than or equal to
-**ge**	greater than or equal to

Example

```
# test example

x=5;y=7;z=14
test $x -eq $y
echo $?
test $x -lt $y
echo $?
test $z -gt $y
echo $?
test $z -eq $y
echo $?
```

```
# test example 2
if test $# -ne 3
then
    echo "Arguments required = 3"
else
    if grep "$1" $2 > $3
    then
       echo "pattern found - job over"
    else
       echo "pattern not found -job over"
    fi
fi
```

[] - the alternate to test

```
# test example

x=5;y=7;z=14
[ $x -eq $y ]
echo $?
[ $x -lt $y ]
echo $?
[ $z -gt $y ]
echo $?
[ $z -eq $y ]
echo $?
```

```
# test example 2
if [ $# -ne 3 ]
then
    echo "Arguments required = 3"
else
    if grep "$1" $2 > $3
    then
       echo "pattern found - job over"
    else
       echo "pattern not found -job over"
    fi
fi
```

Nested if-elif-else

Syntax:

```
if condition
then
        command
elif condition
then
        command
...
...
else
        command
fi
```

String test

- string1 = string2 – True if strings are same
- string1 != string2 – True is strings are different
- -n string – True if the length of string is greater than zero
- -z string – True if the length of the string is zero
- string – True if the string is not a null string

Example

```
# example of if-elif-else with string

echo "enter your branch code"
read bcode
if [ "$bcode" = MCA ]
then
	echo "you are an MCA student"
elif [ "$bcode" = CS ]
then
	echo "you are a comp. sc. student"
elif [ "$bcode" = ECE ]
then
	echo "you are an electronics student"
elif [ "$bcode" = EE ]
then
	echo "you are an electrical student"
else
	echo "you are non-electrical sc group student"
fi
```

File test

- **-s file** – True if the **file** exists and has size greater than zero
- **-f file** – True if the **file** exists and not a directory
- **-d file** – True if the **file** exists and is a directory
- **-r file** – True for read permission
- **-w file** – True for write permission
- **-x file** – True for execution permission

case condition

Syntax:
case expression in
pattern1) execute command;;
pattern2) execute command;;
...
*) execute command;;
esac

```
# case example

case $1 in
cat) echo "you are a cat lover";;
dog) echo " you are a dog lover";;
parrot) echo " you are a parrot lover";;
*) echo "you may not be a pet lover";;
esac
```

case with RE

```
# case with regular expression

case $1 in
[a-z]) echo " Its a character in lower case";;
[A-Z]) echo " its a char in upper case";;
[0-9]) echo " its a numeric digit";;
?) echo "its a special symbol";;
*) echo " Invalid input";;
esac
```

Logical operators

- -a (AND)
- -o (OR)
- ! (NOT)

expr

- Since, shell does not have computing feature, expr command is used to calculate simple arithmetic operations
- Operators include +,-,*, /,%
- expr often used with command substitution to assign a variable

  $ x =6 ; y=z; z = `expr $x + $y` ↵

  $ x = `expr $x + 1` ↵

Some os related commands

- **sleep** – introduce delay
  - $ sleep 100 #This introduce delay of 100 seconds
- **wait** – checks for all background processes have been completed
  - $ wait #waits for all processes
  - $ wait 138 # waits for completion of pid 138

while loop

```
while control-condition
do
        command1
        command2
        ...
        ...
done
```

```
# while loop with touch command

echo "\n No. of files to be created\c"
read num
count=1
while [ $count -le $num ]
do
        echo "\n Enter filename\c"
        read fname
        touch $fname # creates and empty file with name $fname
        count=`expr $count + 1`
done
echo "\nAll file created"
```

Reading a file

```
# read a file
if [ $# -ne 1 ]
then
	echo "\nInvalid arguments"
else
	terminal=$(tty)
	echo "$terminal"
	exec<$1
	while read line
	do
		echo $line
	done
	exec < $terminal
fi
```

until loop

Syntax:

```
until condition
do
        execute command1
        execute command2
        ...
done
```

```
#until example

i=1
until [ $i -gt 10 ]
do
        echo $i
        i = `expr $i + 1`
done
```

For loop

```
for  control-var in val1 val2 val3 ...
do
      command1
      command2
      ...
done
```

```
# for example
for word in I am a student of MCA
do
      echo $word
done
```

```
# for example
for word in $*
do
      echo $word
done
```

For loop

```
# test if the given file exists
for file in $*
do
	if [ -f $file ]
	then
		if [ -s $file ]
		then
			echo "$file exists and size > 0"
			chmod a+x $file
		else
			echo "$file exists with size =0"
		fi
	elif [ -d $file ]
	then
		echo "$file is a directory"
	else
		echo "$file is an invalid filename or directory"
	fi
done
```

Further reading

- Additional UNIX commands
- More on Shell scripting