Lecture: Basic Shell Scripting

How to write/execute a shell script

- use any text editor (vi, vim, gedit etc.)
- fill the script file with what is required to work on
- save and close the file
- set execute permissions on file, as necessary
- ways to execute a script – !keep in mind where you are currently located in the directory tree; if necessary use absolute paths

$ bash your-script
or
$ ./your-script

Example

- create a file and name it *simple_script*
- fill it with the following – the lines beginning with # are commented, so you can write everything you want there, it will not impact the script execution

```
#
#First simple script
#
clear
echo “This is the first one”
```

- save the file, change permissions, and run it.

- you can run your script like any usual command that you know already (eg: ls)
- what you have to do is to copy/move your file in */bin* directory; doing so, I you don’t have to specify the full path for your script to be executed, you can run it from any location you are

Variables in shell

- there are two types of variables: SYSTEM_VARIABLES and user defined variables (udv)
- see system variables with *$ env* command
- print any from the listed variables contains as follows:

$ echo $HOSTNAME
$ echo $PS1

- you can define your own variables (udv) as below:

variable_name=value

Example

- define a variable that keeps a number (attention: write as it is, no spaces, take care of small caps-big caps)

$ numar=5
$ Numar=50

- print variables values (to reffer to a variable, you have to use $ sign before variable name)

$ echo $numar ### will return 5
$ echo $Numar ### will return 50

Exercise

- you have to customize your prompt (this is my current prompt, for example:
[alin@linuxhost ~]$
- hint: you have to change the PS1 variable; please do a backup to the current PS1 value, in case you have to return to the previous setup

Shell arithmetic

- syntax: *$ expr operand1 math-operator operand2*
- examples:

$ expr 5 + 3
$ expr 4 – 1
$ expr 20 % 3
$ expr 20 * 3 ##### Multiplication use * and not * since * its wild card.
$ echo `expr 5 + 3`

- in the last example we use ` (back quote) sign not the ‘ (single quote) sign.
- Here expr 5 + 3 is evaluated to 8, then echo command prints 8 as sum
- If you use double quote or single quote, it will NOT work – try it!
- We use back quote when we want to execute command inside
- try the following and see the differences:

$ echo "Today is date"
$ echo "Today is `date`"

The Exit Status

- a script can be executed successfully or not
- the success value is known as “exit status”, which is zero (0) if success, non-zero if it failed.

- to check the exit status, you can use the **$?** special variable

Example

- create a script which will execute a file creation

#Create new file

touch file1
echo $?

- execute the script, once with no **file1** created previously, and once with an already existing **file1**

Read statement

- is used to get data from user, and store it in a variable
- syntax: read var1 var2 … varN
- the following script ask user to insert name, age:

```
#
#Client database
#
echo “Your first name is: ”     ###press Enter after writing the name
read cfname
echo “Your last name is: ”
read clname
echo “Your age is: ”
read cage
echo “Your full name is $cfname $clname and you are $cage years old. Thanks!”
```

IF condition

Syntax:

```
if condition
then
    command1 if condition is true or if exit status of condition is 0 (zero)
    ...
    ...
fi
```

- condition = the result of comparing two values

Example

- we'll create a script that will check if a file already exist in our working directory, and if exist it will print a specific message

```
$ cat > printfile
#!/bin/sh
#
#Script to print file
#
if cat $1
then
echo -e "\n\nFile $1 was found!"
fi
```

- we'll run now the script with $ ./printfile file1 (file1 is the argument received by our script and should exist). Run the script for a file that don't exist, also.

Exercise

- Based on the example above create a script that can rename a file and print a message if the execution was successful.

Test command or [expression]

- sometimes we need to compare different arguments
- Syntax: test expression or [expression]
- below you can find a list of useful operators:

For Mathematics:

-eq = is equal to (equivalent with: 5 == 6)
-ne = is not equal to (5 != 6)
-lt = is less than (5 < 6)
-le = is less than or equal to (5 <=6)
-gt = is greater than (5 > 6)
-ge = is greater than or equal to (5 >= 6)

For string comparisons:

string1 = string2 (string1 is equal to string2)
string1 != string2 (string1 is not equal to string2)
string1 (string1 is not null)
-n string1 (string1 is not null and does exist)
-z string1 (string1 is null and does exist)

Test for file or directory types

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

Example

- the following script should check if a number is greater than 1001.

```
 $ cat > istrue
#!/bin/sh
#
# Script to see whether argument is greater than 1001
#
if test $1 -gt 1001
then
echo "Number $1  is greater than 1001"
fi
```

- we can write the condition also, like this: if [$1 -gt 1001]

Exercise

- as you know already, 1001 is the UID for the first user in CentOS. Make a script based on the example above that can tell if a given user name is a daemon or a regular user.

If-else-fi condition

Syntax:

```
if condition
then
        condition is zero (true - 0)
        execute all commands up to else statement

else
        if condition is not true then
        execute all commands up to fi
fi
```

- going to work on the previous example (comparing a number with 1001), we'll go further and we'll implement the else branch, in order to provide a response if the number is not greather than 1001

```
#!/bin/sh
#
# Script to see whether argument is greater than 1001
#
##Check if an argument was provided as input from keyboard
if [ $# -eq 0 ]
then
echo "$0 : You must type a number"
exit 1
fi
###Comparing structure
if test $1 -gt 1001
then
echo "Number $1 is greater than 1001"
else
echo "Number $1 is smaller than 1001"
fi
```

Nested if-else-fi condition

Example

```
###

alege=0

echo "1. Optiune 1"
echo "2. Optiune 2"
echo -n "Selecteaza optiune [1 sau 2]? "
read alege

if [ $alege -eq 1 ] ; then

        echo "Ai ales Optiune 1"

else #### if within if ######

if [ $alege -eq 2 ] ; then
        echo "Ai ales Optiune 2"
else
        echo "Nu a fost selectata nici una din optiuni."
```

```
        fi
fi
```

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
```

- practice **for** with the following script, to have a practical understanding

```
##for loop script test
for i in 1 2 3
do
        echo „Acesta este testul $i”
done
```

- also, instead of „1 2 3” list, we can specify an interval using expressions

Example

```
##another for example
for ((i = 0 ; i <= 3; i++ ))
do
        echo „Acesta este versiunea 2 de test $i”
done
```

- as a general rule, the syntax will look something like below:

```
for (( expr1; expr2; expr3 ))
      do
          .....
                ...
          repeat all statements between do and
          done until expr2 is TRUE
      Done
```

- i++ is equavalent with i=i+1, known also as an increment

- write the following script and try to understand how the result is obtained

```
#!/bin/sh
#
#Script to test for loop
#
#
if [ $# -eq 0 ]
then
echo "Error - Number missing form command line argument"
echo "Syntax : $0 number"
exit 1
fi
n=$1
for i in 1 2 3 4 5
do
echo "$n * $i = `expr $i \* $n`"
done
```

Debug a shell script

- Syntax:
$ sh option script-name argument(s)
or
$bash option script-name (arguments),

where option can be:
-v: print shell input lines as they are read
-x: expand each simple-command

-take the following example:

```
$cat > bashdebug
#
# shell debug
#
sum=`expr $1 + $2`
echo $sum
```

- now run the scripts as *$ bash -x bashdebug 2 4* and also as *$ bash -v bashdebug 2 4* and read the output line by line

Practice Exercises:

- check the following scripts:

```
-----------------------------------------------------
#!/bin/bash
# Basic arithmetic using let
let a=5+4
echo $a       # 9
let "a = 5 + 4"
echo $a       # 9
let a++
echo $a       # 10
let "a = 4 * 5"
echo $a       # 20
let "a = $1 + 30"
echo $a        # 30 + first command line argument

-----------------------------------------------------------
#!/bin/bash
MAX=10000
  for ((nr=1; nr<$MAX; nr++))
  do

    let "t1 = nr % 5"
    if [ "$t1" -ne 3 ]
    then
      continue
    fi

    let "t2 = nr % 7"
    if [ "$t2" -ne 4 ]
    then
      continue
    fi

    let "t3 = nr % 9"
    if [ "$t3" -ne 5 ]
    then
      continue
    fi

  break   # What happens when you comment out this line? Why?
  done
  echo "Number = $nr"
exit 0
-----------------------------------------------------
```

```
echo -e "Kernel Details: " `uname -smr`
echo -e "`bash --version`"
echo -ne "Uptime: "; uptime
echo -ne "Server time : "; date
```

```
#!/bin/bash
# Basic arithmetic using expr
expr 5 + 4
expr "5 + 4"
expr 5+4
expr 5 \* $1
expr 11 % 2
a=$( expr 10 - 3 )
echo $a # 7
```

```
#!/bin/bash
# Basic arithmetic using double parentheses
a=$(( 4 + 5 ))
echo $a # 9
a=$((3+5))
echo $a # 8
b=$(( a + 3 ))
echo $b # 11
b=$(( $a + 4 ))
echo $b # 12
(( b++ ))
echo $b # 13
(( b += 3 ))
echo $b # 16
a=$(( 4 * 5 ))
echo $a # 20
```

```
#!/bin/bash
# Show the length of a variable.
a='Hello World'
echo ${#a}       # 11
b=4953
echo ${#b}       # 4
```

```
#!/bin/bash
# Nested if statements
if [ $1 -gt 100 ]
        then
                echo Hey that's a large number.
                if (( $1 % 2 == 0 ))
                then
                        echo And is also an even number.
                fi
        fi
```

```
#!/bin/bash
# Basic for loop
names='Stan Kyle Cartman'
for name in $names
do
echo $name
done
echo All done
```

```
#!/bin/bash
# Basic range in for loop
for value in {1..5}
do
echo $value
done
echo All done
```

```
#!/bin/bash
# Passing arguments to a function
print_something () {
echo Hello $1
}
print_something Mars
print_something Jupiter
```

```
#!/bin/bash
# Setting a return value to a function
lines_in_file () {
cat $1 | wc -l
}
num_lines=$( lines_in_file $1 )
echo The file $1 has $num_lines lines in it.
```

```
#!/bin/bash
STRING="Primul script"
echo $STRING
```

```
#!/bin/bash
for fn in Tom Dick Harry; do
echo "Numele este $fn"
done
```

```
#!/bin/bash
n=1
while [ $n -le 6 ]; do
echo $n
let n++
done
```

Exercise

1. Create a bash script which will solve the following requirements:
    a. will create 100 directories from a1 to a100
    b. in directory a58 create 81 files from f1 to f81
    c. write in file f58 all the numbers from 1 to 2000 in reverse order, measure the file size and write it at the end of the file