Introduction to Shell scripting

John A. Juma

Animal and Human Health

International Livestock Research Institute (ILRI)

March, 2023

j.juma@cgiar.org

ILRI
INTERNATIONAL LIVESTOCK RESEARCH INSTITUTE

CGIAR

AFRICA CDC
Centres for Disease Control and Prevention
Safeguarding Africa's Health

African Union

ASLM
AFRICAN SOCIETY FOR LABORATORY MEDICINE

mastercard foundation

Objectives

- What is Shell scripting
- Keywords
- Variables

What is Shell scripting

Imagine the following task:

For a literature search, **you have to copy the third line of one thousand text files in one thousand different directories and paste it into a single file**.

Using a graphical user interface (GUI), you would not only be clicking at your desk for several hours, but you could potentially also commit an error in the process of completing this repetitive task.

The Unix shell is both a **command-line interface** (CLI) and a **scripting language**, allowing such repetitive tasks to be done automatically and fast.

With the proper commands, the shell can repeat tasks with or without some modification as many times as we want. Using the shell, the task in the literature example can be accomplished in seconds.

Shell

The shell is a program where users can type commands.

Shell programs are interpreted each time they are run.

In general, a shell program can be characterized by:

1. shell programs consist of **one or more primitive shell commands**.
2. shell programs are **created using your text editor** of choice, e.g., **vi, vim, nano, emacs**
3. shell programs are **executed just as shell commands** are, by typing the name of the program followed by the *[Enter]* key
4. shell programs have **permission modes** as do any other file and must have the correct permissions set to execute the program.
5. shell language has the functionality to **allow input & output, iteration, logical decision making, file creation and deletion, and system call** capability
6. shell programs are **free format**, as long as the **syntax of each shell command is correct**. This means that blank lines, indentation and abundant whitespace can be used freely.

https://swcarpentry.github.io/shell-novice/01-intro/index.html

Writing a shell script

1. Use any text editor
2. After writing the shell script, save it and set execution permissions as:

```
1. chmod +x script-filename
2. chmod 755 script-filename
```

3. Execute your script using either the following ways:

```
1. bash script-filename
2. sh script-filename
3. ./script-filename
```

https://swcarpentry.github.io/shell-novice/01-intro/index.html

Your first shell script

Write your first shell script that will print "**Knowledge is Power**" on screen.

Using your favourite text editor, type the following and save the file as **`my-first-shell-script.sh`**

```
#!/bin/bash

#
# My first shell script
#

clear
echo "Knowledge is power"
~
~
~
~
```

```
jjuma:practicals jjuma$ ls -lth
total 8
-rw-r--r--  1 jjuma  CGIARAD\Domain Users    74B Feb 10 12:19 my-first-shell-script.sh
jjuma:practicals jjuma$ chmod +x my-first-shell-script.sh
jjuma:practicals jjuma$ ls -lth
total 8
-rwxr-xr-x  1 jjuma  CGIARAD\Domain Users    74B Feb 10 12:19 my-first-shell-script.sh
jjuma:practicals jjuma$
```

```
Knowledge is power
jjuma:practicals jjuma$
```

shell script

Write a shell script that print user information who currently login , current date and time.

```
#!/bin/bash
#
# Script to print user information who currently login , current date
# & time
#
echo "Hello $USER"
echo -e "Today is \c ";date
echo -e "Number of user login : \c" ; who | wc -l
echo "Calendar"
cal
exit 0
```

Variables in a shell script

In UNIX, there are two types of variables:

- **System variables** - Created and maintained by UNIX. This type of variable defined in **CAPITAL LETTERS**.

System Variable	Meaning
BASH	Shell name
BASH_VERSION	Shell version name
COLUMNS	Number of columns for the screen
LINES	Number of lines on screen
PS1	Prompt settings
HOME	Home directory
OSTYPE	OS type path
PWD	Current working directory
PATH	Path settings
SHELL	Shell name
LOGNAME	Logging name
USERNAME	Username who is currently logged in to the PC
USER	User of the PC

Variables in a shell script

- **User-defined variables:** Created and maintained by user. This type of variable defined in **lowercase letters**.

```
#!/bin/bash


#
# Script to test MY knowledge about variables!
#
myname=jjuma
myos=TroubleOS
myno=5
echo "My name is $myname"
echo "My os is $myos"
echo "My number is $myno, can you see this number ?"
```

Shell Arithmetic

expr evaluate expression

Syntax **expr operand1** math-operator **operand2**

```
$ expr 1 + 3
$ expr 2 - 1
$ expr 10 / 2
$ expr 20 % 3
$ expr 10 \* 3
$ echo `expr 6 + 3`
```

We can use backticks and not single or double quotes

Command line arguments

- Telling the command/utility which option to use.
- Informing the utility/command which file or group of files to process (reading/writing of files).

```
#!/bin/sh
#
# Script that demos, command line args
#
echo "Total number of command line argument are $#"
echo "$0 is script name"
echo "$1 is first argument"
echo "$2 is second argument"
echo "All of them are :- $* or $@"
```

if condition

Decision making

- if condition which is used for decision making in shell script, If given condition is true then command1 is executed.

Syntax:

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

- Condition is nothing but comparison between two values.
- For compression you can use test or [expr] statements or even exist status can be also used.
- An expression is nothing but combination of values, relational operator (such as >, <> etc) and mathematical operators (such as +, −, / etc).

test or [expr]

test command or [expr]

- test command or [expr] is used to see if an expression is true, and if it is true it return zero (0), otherwise returns nonzero for false.

Syntax:

```
test expression OR [ expression ]
```

Determine whether given argument number is positive.

```
#!/bin/sh
#
# Script to see whether argument is positive
#
if test $1 -gt 0
then
        echo "$1 number is positive"
fi
```

test or [expr] works with: Integer (Number without decimal point), File types, Character strings

if…else…if

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

Syntax:
if condition
then
condition is zero (true - 0)
execute all commands up to else statement (**command1**)
else
if condition is not true then execute all commands up to fi (**command2**)
fi

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

Loops

– Loop defined as: Computer can repeat particular instruction again and again, until condition satisfies. A group of instruction that is executed repeatedly is called a loop.

– Bash supports:

1. for loop
2. while loop

– Note that in each loop,

* First, the variable used in loop condition must be initialized, then execution of the loop begins.
* A test (condition) is made at the beginning of each iteration.
* The body of loop ends with a statement that modifies the value of the test (condition) variable.

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

Try the following script:
testfor

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

For loop

```
#!/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=$1
for i in 1 2 3 4 5 6 7 8 9 10
do
        echo "$n * $i = `expr $i \* $n`"
done
```

For loop – C style

Syntax:

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

```
#!/bin/sh
# #Script to test for loop 2
#
for (( i = 0 ; i <= 5; i++ ))
do
       echo "Welcome $i times"
done
```

Nested for loop

```
#!/bin/sh

# Nesting of for loop

for (( i = 1; i <= 5; i++ )) ### Outer for loop ###
do
        for (( j = 1 ; j <= 5; j++ )) ### Inner for loop ###
        do
                echo -n "$i "
        done
        echo "" #### print the new line ###
done
```

while loop

Syntax:

```
while [ condition ]
do
        command1
        command2
        command3
        ..
        ..
done
```

Loop is executed as long as given condition is true.

```
#!/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
```

Conditional execution

The control operators are **&&** (read as **AND**) and **||** (read as **OR**).

- The syntax for **AND** list is as follows:

  command1 **&&** command2

command2 is executed if, and only if, command1 returns an exit status of zero.

- The syntax for **OR** list as follows:

  command1 **||** command2

command2 is executed if and only if command1 returns a non-zero exit status.

- You can use both as follows:

  command1 **&&** comamnd2 **||** command3

if command1 is executed successfully then shell will run command2 and if command1 is not successful then command3 is executed.

Example: `$ rm myf && echo "File is removed successfully" || echo "File is not removed"`

Operators

Option	Operator
-eq	Is equal to e.g., 5 == 6; if test 5 –eq 6; if [5 –eq 6]
-ne	Is not equal e.g., 5 != 6; if test 5 –ne 6; if [5 –ne 6]
-lt	Is less than e.g., 5 < 6; if test 5 –lt 6; if [5 –lt 6]
-le	Is less than or equal to e.g., 5 <= 6; if test 5 –le 6; if [5 –le 6]
-ge	Is greater than or equal to 5 >= 6; if test 5 –ge 6; if [5 –ge 6]
string1 == string2	string1 is equal to string2
string1 != string2	string1 is not equal to string2
-s file	Non-empty file
-f file	Is file exists or normal file and not a directory
-d dir	Is directory exists and not a file
-w file	Is a writeable file
-r file	Is a read-only file
-x file	Is an executable file
! Expression	Logical NOT
expr1 –a expr2	Logical AND
expr1 –o expr2	Logical OR

Setup

Download the data from https://swcarpentry.github.io/shell-novice/data/shell-lesson-data.zip