Computer Engineering Dept.
Cairo University
Fall 2023

Operating Systems
Advanced Bash Scripting

Cairo University

Objectives

After finishing this lab, you should be able to do the following:

- Learn how to use arrays, strings and various conditional statements including loops.
- Be able to answer the following:

  Scripting/Interpreted vs Compiled

  Why using Bash

Hello World Bash:

1. Create an empty file with any name, .sh, on the desktop
2. Type the following inside

```
#!/bin/bash
echo Hello World Bash
```

3. Save, open the terminal (command line of linux), and type inside it

```
cd ~/Desktop
chmod +x yourfile.sh
./yourfile.sh
```

Bash Commands:

(1) Variables assignments

- No space before and after the assignment "="
- "=" can be either an assignment or a test operator, depending on context.
- **"expr" command** is an all-purpose expression evaluator: Concatenates and evaluates the arguments according to the operation given (arguments must be separated by spaces). Operations may be arithmetic, comparison, string, or logical. Notice that it's sensitive to spaces.
- Place the string to be evaluated inside a $(evaluate_this) (parentheses) or ` evaluate_this ` (backticks)

```
#!/bin/bash
string_var1=abcABC123ABCabc;
string_var2="Hello World!";
num_var=1234;
echo `expr 2 \* 3`                # 6
echo `expr 2 + 3`                 # 5
echo $(expr 2 + 3)                # 5
echo `expr 1.5 \* 3`              # error: expr: non-integer argument
```

(2) Arithmetic Expansion:

Arithmetic expansion provides a powerful tool for performing (integer) arithmetic operations in scripts. Translating a string into a numerical expression is relatively straightforward using backticks,double parentheses, or let.

```
#!/bin/bash
y=1;
y=$y+1; echo $y;                  # 1+1 ! so we need to use let
let y=$y+1; echo $y;              # 2
y=$((y+1)); echo $y;              # 2
y=`expr $y + 1`; echo $y;         # 2
y=$((2 ** 3))                     # 8; it calculates 2^3
```

(3) Substring Manipulation

- Strings start at index 0
- You can access the end of the string using index -1
- Substring Extraction is done by accessing string at a certain position for a certain length ${string:position:length}

```
#!/bin/bash
# Length of a string
stringZ=abcABC123ABCabc
echo ${#stringZ}                # 15
echo `expr length $stringZ`     # 15

# Substring extraction
echo ${stringZ:0}               # abcABC123ABCabc
echo ${stringZ:1}                # bcABC123ABCabc
echo ${stringZ:7:3}               # 23A, Three characters of substring.
echo ${stringZ: -4}              # Cabc
echo ${stringZ: -4:1}             # C
```

(4) **Arrays and Sequences**

- To refer to the content of an item in an array, use curly braces, otherwise it will get the first element and anything follows the variable name will be treated as a string
- If the index number is @ or *, all members of an array are referenced.
- Indices start at 0

```
#!/bin/bash
farm_hosts=(web03 web04 web05 web06 web07)
for i in ${farm_hosts[*]}; do
        echo item: $i
done
# using $farm_hosts[*] produces web03[*]
# for i in web03 web04 web05 web06 web07; do # is correct too
```

(5) **If/else statements**

```
#!/bin/bash
T1=foo
T2=bar
if [ $T1 = $T2 ]; then
        echo expression evaluated as true
else
        echo expression evaluated as false
fi
```

(6) **Loops for, while and until**

a. For Loop

- We want to customize our *ls* command display, so we'll use a for loop to loop on the output of a *ls* command and print it the way we want.
- As you saw earlier in the expr evaluation; your commands should be enclosed between parentheses or backticks $(command -option) or `command -option` in order to be evaluated, otherwise they'll be treated like a string

```
#!/bin/bash
for i in $( ls ); do
   echo item: $i
done
# for i in `ls`; do # is correct too

for i in {1..5..2}; do
   echo item: $i
done
# for i in {1..5..1}; do #is the same as for i in {1..5}; do
```

b. While Loop

```
#!/bin/bash
COUNTER=0
while [  $COUNTER -lt 10 ];
do
        echo The counter is $COUNTER
        let COUNTER+=1
done
```

c. Until Loop

```
#!/bin/bash
COUNTER=20
until [  $COUNTER -lt 10 ];
do
        echo COUNTER $COUNTER
        let COUNTER+=1
done
```

(7) Functions

a. Function without/with parameters

```
#!/bin/bash

function increment {
        counter=0;
        inc=1;
        # Check that function has no arguments
        if [ "$#" -ne 0 ];
        then
                inc=$1; # if it has arguments set the increment with the first argument
        fi

        for i in `seq 10 -1 1`;
        do
          echo "The counter is $counter";
          let counter=counter+$inc;
        done
}

# Call functions
# without arguments
increment;
echo "calling increment with an increment of 3";
#with arguments
increment 3;
```

b. Reading input from user

```
#!/bin/bash
read string
echo $string
```

References

- Advanced Bash-Scripting Guide
- Comparison Operators