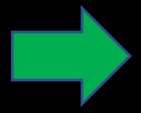
Operating Systems

Lab 06

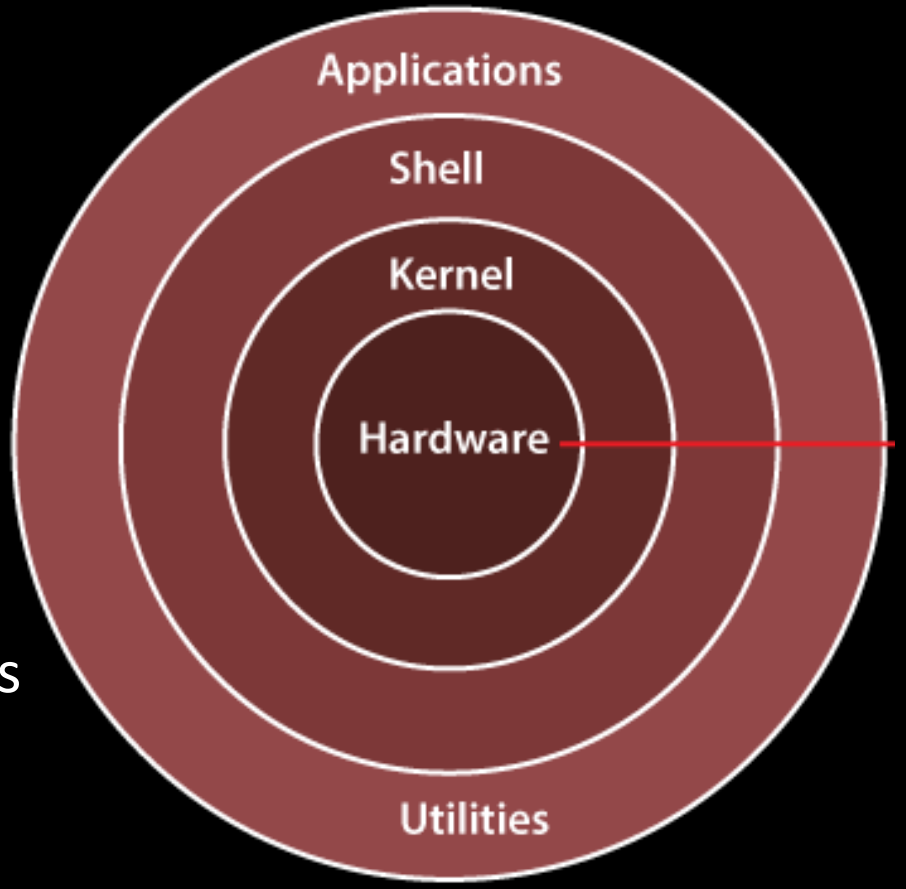
Content



Content
What is Shell?
Writing Scripts
Arithmetic Operations
Flow Control
Command Line Arguments
Arrays
Functions

What is Shell?

- Operating systems work as a bridge between hardware and applications
  - Kernel: hardware drivers etc.
  - Shell: user interface to kernel.
  - Some applications (system utilities).
- The kernel is the core component of most operating systems.
  - Kernel's responsibilities include managing the system's resources.



What is Shell?

- Shell
  - Shell is an interface for users or applications to interact with OS kernels.
  - Shell is a special program that executes user commands.
  - Shell is a high-level programming language (compared to C/C++, Fortran, . . .)
- Types of Shell
  - sh (Bourne Shell)
  - csh (C Shell)
  - ksh (Korn Shell)
  - bash (Bourne Again Shell) - Default Shell on Linux
  - tcsh (TENEX C Shell)

What is Shell?

- In Linux there are two types of variables
  - ENVIRONMENT: define configuration settings between applications and processes in Linux – valid for all subsequently opened shells.
  - User defined – ONLY valid within the current shell
- To reference a variable, append $ to the name of the variable.
  - Example: $\$PATH$ – to print the path of the current shell.
- To define a variable: $name = variable$ (NO SPACES)

```
lvl3@lvl3-vm:~/Desktop$ ahmed=55
lvl3@lvl3-vm:~/Desktop$ $ahmed
55: command not found
```

What is Shell?

- $echo$ used to print on the screen.
  - $echo < arguments >$

```
lvl3@lvl3-vm:~/Desktop$ echo Hello Linux
Hello Linux
```

- $read$ used to take input from keyboard
  - $read \ < variable\ name >$

```
lvl3@lvl3-vm:~/Desktop$ read x
12354
```

What is Shell?

- Example:

```
lvl3@lvl3-vm:~/Desktop$ echo $SHELL
/bin/bash
lvl3@lvl3-vm:~/Desktop$ echo my      name     is  omar
my name is omar
lvl3@lvl3-vm:~/Desktop$ echo "my     name     is  omar"
my     name     is  omar
```

Writing Scripts

- Create a text file
  - $gedit\ hello.sh$
- The first line is line is called the "Shebang" line. It tells the OS which interpreter to use.
- To print the output of a command, enclose it in ``

```
#!/bin/bash
## Print Hello world ...
echo
echo "Hello World!"
echo "Your name is " $USER
echo "Today is " `date`
echo
```

Writing Scripts

- To run the script, you have two options.
  - Set the permissions of the file to execute.

```
lvl3@lvl3-vm:~/Desktop/scripts$ chmod a+x hello.sh
lvl3@lvl3-vm:~/Desktop/scripts$ ./hello.sh

Hello World!
Your name is  lvl3
Today is  12 2022 ,أبر EET 01:28:54 ص
```

  - Call the bash command.

```
lvl3@lvl3-vm:~/Desktop/scripts$ bash hello.sh

Hello World!
Your name is  lvl3
Today is  12 2022 ,أبر EET 11:14:12 ص
```

Writing Scripts

- Special Characters and Operators

#	Starts a comment line.
$	Indicates the name of a variable.
\	Escape character to display next character literally
{ }	Used to enclose name of variable
;	Command separator, which allows one to put two or more commands on the same line.
&&	Logical AND
||	Logical OR
!	Logical NOT

Writing Scripts

- Double quotes “ ”
  - String is expanded
- Single quotation ‘ ’
  - String is printed literally
- Back quotation ` `
  - String is executed as a command

```
$ HI=Hello
$ echo HI
HI
$ echo $HI
Hello
$ echo \$HI
$HI
$ echo "$HI"
Hello
$ echo '$HI'
$HI
$ echo "$HILe"

$ echo "${HI}Le"
HelloLe
$ echo `pwd`
/home/lyan1/traininglab/bash_scripting_fall_2016
```

Arithmetic Operations

Operation	Operator
Addition	+
Subtraction	-
Multiplication	*
Division	/
Exponentiation (bash only)	**
Modulo	%

Arithmetic Operations

- $((\ldots))$ or $[\ldots]$ commands
  - Addition: $((1 + 2))$
  - Multiplication: $[a * b]$
  - *This notation works in shell files.*
- Or use the let command: $let\ c = \$a - \$b$
- Or use the $expr$ command: $c = `expr\ \$a\ -\ \$b`$
  - Add space after the $expr$ command and between operands and operators.
  - *This notation works in shell files.*
- You can also use C-style increment operators: $let\ c\ += 1$ or $let\ c --$

Arithmetic Operations

- Example

```
lvl3@lvl3-vm:~/Desktop/scripts$ echo $((1+2))
3
lvl3@lvl3-vm:~/Desktop/scripts$ echo $(1+2)
1+2: command not found

lvl3@lvl3-vm:~/Desktop/scripts$ echo $[1*2-5]
-3
```

Arithmetic Operations

- Example

```
lvl3@lvl3-vm:~/Desktop/scripts$ a=5
lvl3@lvl3-vm:~/Desktop/scripts$ b=10
lvl3@lvl3-vm:~/Desktop/scripts$ let c=$a-$b
lvl3@lvl3-vm:~/Desktop/scripts$ echo $c
-5
lvl3@lvl3-vm:~/Desktop/scripts$ c=`expr $a+$b`
lvl3@lvl3-vm:~/Desktop/scripts$ echo $c
5+10
lvl3@lvl3-vm:~/Desktop/scripts$ c=`expr $a + $b`
lvl3@lvl3-vm:~/Desktop/scripts$ echo $c
15
```

```
lvl3@lvl3-vm:~/Desktop/scripts$ let c++
lvl3@lvl3-vm:~/Desktop/scripts$ echo $c
16
```

Arithmetic Operations

- Integer Comparisons

Operation	bash
Equal to	`1 -eq 2`
Not equal to	`$a -ne $b`
Greater than	`$a -gt $b`
Greater than or equal to	`1 -ge $b`
Less than	`$a -lt 2`
Less than or equal to	`$a -le $b`

Arithmetic Operations

- String Comparisons

Operation	bash
Equal to	`$a == $b`
Not equal to	`$a != $b`
Zero length or null	`-z $a`
Non zero length	`-n $a`

Arithmetic Operations

- Logical Operators

Operation	bash
! (NOT)	test ! 5 -eq 4
-o (OR)	test 5 -eq 4 -o 6 -eq 6
-a (AND)	test 5 -eq 4 -a 6 -eq 6

Arithmetic Operations

- Use the *test* command for comparisons.
  - The command does not print anything, but acts on the **exit status variable (?)**
  - 0 means true. 1 means false
  - Use ; to separate multiple commands in the same line.

```
lvl3@lvl3-vm:~/Desktop$ x=1
lvl3@lvl3-vm:~/Desktop$ test $x == "1"
lvl3@lvl3-vm:~/Desktop$ echo $?
0
lvl3@lvl3-vm:~/Desktop$ test $x -eq 1
lvl3@lvl3-vm:~/Desktop$ echo $?
0
lvl3@lvl3-vm:~/Desktop$ test $x -eq 3; echo $?
1
```

Arithmetic Operations

- Example

```
# !/bin/bash
echo "Entet first number: "
read n1
echo "Enter second number: "
read n2
echo ---------------------
echo 'n1 -eq n2 ?'
echo `test $n1 -eq $n2; echo $?`
echo
echo 'n1 -ne n2'
echo `test $n1 -ne $n2; echo $?`
echo
echo 'n1 -gt n2'
echo `test $n1 -gt $n2; echo $?`
echo
```

```
echo 'n1 -lt n2'
echo `test $n1 -lt $n2; echo $?`
echo
echo 'n1 -ge n2'
echo `test $n1 -ge $n2; echo $?`
echo
echo 'n1 -le n2'
echo `test $n1 -le $n2; echo $?`
echo
c=`expr $n1 + $n2`
echo 'n1 + n2 = '; echo $c
c=`expr $n1 - $n2`
echo 'n1 - n2 = '; echo $c
c=$(($n1*$n2))
echo 'n1 * n2 = '; echo $c
c=$(($n1/$n2))
echo 'n1 / n2 = '; echo $c
```

Flow Control

- Control constructs can change the order of command execution
- Control constructs in bash are
  - Conditionals: if
  - Loops: $for, while, until$
  - Switches: $case, switch$

Flow Control

if … $else$ … fi

- Tests whether the exit status of a list of commands is 0, and if so, execute one or more commands

```
if [ condition 1 ]; then
   some commands
elif [ condition 2 ]; then
   some commands
else
   some commands
fi
```

  - Note the space between condition and the brackets

Flow Control

Example: check if a number is positive, negative, or 0.

```
# !/bin/bash
echo -n 'Enter a number: `
read num
if [ $num -gt 0 ] ; then
        echo 'The number is positive'
elif [ $num -lt 0 ] ; then
        echo 'The number is negative'
else
        echo 'The number is 0'
fi
```

- The $-n$ option make $echo$ command do not print new line

Flow Control

for loop

```
for arg in list
do
   some commands
done
```

Example

```
#!/bin/bash
for i in 1 2 3 4 5 do
        echo $i
done
```

```
#/bin/bash
for i in `seq 1 10` ; do
        echo $i
done
```

```
#!/bin/bash
for ((i = 0 ; i < 5 ; i++)); do
        echo "Welcome $i times."
done
```

Flow Control

- *while* loop

```
while [ condition ]
do
   some commands
done
```

```
#!/bin/bash
echo -n 'Enter a counter: '
read counter
factorial=1
while [ $counter -gt 0 ]
do
factorial=$(( $factorial * $counter ))
counter=$(( $counter - 1 ))
done
echo $factorial
```

Flow Control

- The *until* construct tests for a condition at the top of a loop, and keeps looping as long as that condition is **false** (the opposite of *while* loop)

```
until[ condition ]
do
   some commands
done
```

```
#!/bin/bash
x=1
until [ $x -ge 10 ] ; do
 echo $x
 x=$(( $x+1 ))
done
```

Flow Control

- Switching construct

```
# case construct
case variable in
  "condition1")
    some commands
    ;;
  "condition2")
    some commands
    ;;
esac
```

```
#!/bin/bash
echo -n "Enter the name of a country: "
read COUNTRY
echo -n "The official language of $COUNTRY is "
case $COUNTRY in
        France)
        echo "French"
        ;;
        England | "United Kingdom" | "United States" | USA)
        echo "English"
        ;;
        Egypt | "Libya" | "Saudia Arabia" )
        echo "Arabic"
        ;;
        *) #* means anything else
        echo "unknown"
        ;;
esac
```

Exercise

Write a shell script to prompt the user to enter a grade of a student,

if the grade is greater than 90, print A

if the grade is greater than 80 and less than 90, print B

If the grade is greater than 70 and less than 80, print C

If the grade is greater than 60 and less than 70, print D

If the grade is greater than 50 and less than 60, print D-

If the grade is less than 50, print Failed

Exercise

```
# !/bin/bash
echo -n 'Enter a studnet grade: `
read grade
if [ $grade -ge 90 ] ; then
      echo 'A'
elif [ $grade -ge 80 -a $grade -lt 90 ] ; then
      echo 'B'
elif [ $grade -ge 70 -a $grade -lt 80 ] ; then
      echo 'C'
elif [ $grade -ge 60 -a $grade -lt 70 ] ; then
      echo `D'
else
      echo 'Failed'
fi
```

Exercise

Write a shell script to compute the following equation

$$\sum_{i=1}^{10} i * 2$$

Exercise

Write a shell script to compute the following equation

$$\sum_{i=1}^{10} i * 2$$

```
#!/bin/bash
i=0
for ((x=1;x<=10;x++))
do
        i=$((($i+$x))
done
echo "The result=$i"
```