

Lab 4 for Operating Systems

Learning outcome

Upon successful completion of this lab, you will be able

- How to program the shell script into Fedora

Programming in Shell

- **Shell script structure**

- Name: `filename.sh`
- Content
 - First line: `#!/bin/sh`
 - `commands`
 - `exit 0`

- **Execute Shell script**

- **Case 1**

- Syntax: `/bin/sh filename.sh [argument]`
- Ex: `/bin/sh sum.sh 10`

- **Case 2**

- Syntax:
 - Assign the execute mode to file name: `chmod +x filename.sh`
 - Execute: `/path/filename.sh [argument]`
- Ex:
 - `chmod +x sum.sh`
 - `./sum.sh 10`

- **Variable**

- Declaration: `<variable_name>=<value>`
- Access: `variable1=$variable2`
- Input value of variable: `#read variable`
- Some system variables
 - `$#`: number of parameters
 - `$0`: name of command
 - `$*`: list of parameter
 - `$number`: `number>0` – the input argument
- Get input value to variable from the keyboard, file
 - Keyboard: `read var1 var2 ... varn`
 - File: `read var1 var2 ... varn <data_file`
 - if the variable name is not existing, the value is assigned to `$REPLY` variable

- Ex:
 - read
 - var1 = \$REPLY
- The “\” character allow the enter new line during the input value
- The “-r” option is ineffective the “\”
- Ex:

```
#!/bin/sh
while read line
do
    echo "$line"
done < /etc/passwd
exit 0

[root@localhost ~]# chmod +x readFile.sh
[root@localhost ~]# ./readFile.sh_
```

- Print the content to the screen: echo “string content”
- **expr**
 - **expr op1 operator op2**
 - Not support floating point
 - Operator: +, -, *, /, = (equal), <=, <, >, >=, != (different), &, |
 - Ex:
 - expr 1 + 3
 - expr 2 - 1
 - expr 10 / 2
 - expr 20 % 3
 - expr 10 * 3 (multiple – not use only * character)
 - echo `expr 6 + 3`
 - z=`expr \$z + 3`
- **let**
 - let “z=\$z+3”
 - let “z += 3”
 - let “z=\$m*\$n”
- **\$((...))**
 - z=\$((z+3))
 - z=\$((m*n))
- **If constructs**
 - **Syntax**

```
if <control command>
then
    command1
[else
    command2]
fi
```

```

○ Ex
#!/bin/sh
if cat $1
then
echo -e "\n\nFile $1, found and successfully"
fi

```

- If ... else constructs

```

○ Syntax
if <control command>
then
    command1
elif <control command >
then
    command2
[else
    command3]
fi

```

```

○ Ex
#!/bin/sh
if [ $1 -gt 0 ]
then
    echo "$1 is positive"
elif [ $1 -lt 0 ]
then
    echo "$1 is negative"
elif [ $1 -eq 0 ]
then
    echo "$1 is zero"
else
    echo "Opps! $1 is not number, give number"
fi

```

- The [] or test command

- Uses to compare the condition
- Syntax
 - test <expression>
 - [<blank><expression><blank>] (notes: must be the blank character in the [])

- Relation operator: -eq, -ne, -gt, -lt, -ge, -le

- File mode accession: -r (read), -w (write), -x (execute), -f (file is existing), -d (is directory), -e (existing on disk)

- **String operator**
 - `=, !=`
 - Unary: `-z` (string have zero length), `-n` (string have size)
- **Logic operator:** `!, -a, -o`
- **Select construct**
 - Syntax


```
case <var> in
    value1)
        command1
    ;;
    valueN)
        commandN
    ;;
*)
    command
;;
esac
```
 - Ex


```
#!/bin/sh
ftype=`file "$1"`
case "$ftype" in
    "$1: Zip archive"*)
        unzip "$1" ;;
    "$1: gzip compressed"*)
        gunzip "$1" ;;
    "$1: bzip2 compressed"*)
        bunzip2 "$1" ;;
    *) error "File $1 can not be uncompressed with smartzip";;
esac
```

Handwritten notes: "command" with an arrow pointing to the case branches; "value" with an arrow pointing to the case values.
- **For loop**
 - Syntax


```
for variable in const1 const2 ...
do
    commands
done
```
 - Ex:


```
#!/bin/sh
if [ $# -eq 0 ]
then
    echo "Thieu tham so"
```

```

echo "Syntax : $0 number"
echo "In bang cuu chuong cho 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

```

- while loop

- Syntax

```

while expression
do
    command
done

```

- Ex

```

#!/bin/sh
echo "Chương trình tính tổng 1- $1"
index=0
tong=0
while [ $index -lt $1 ]
do
    index=$((index + 1))
    tong=$((tong + index))
done
echo "Tổng 1-$1= $tong"
exit 0

```

- Until loop

- Syntax

```

until expression
do
    commands
done

```

- Keyword: break, exit (exit to shell), continue

- Delay the program in runtime: sleep <time>

- Select constructs

- Syntax

```

select var in ...
do
    break
done

```

```

○ Ex
#!/bin/sh
echo "What is your favourite OS?"
select var in "Linux" "Gnu Hurd" "Free BSD" "Other"
do
    break
done
echo "You have selected $var"

```

• Array

```

○ Declaration: ArrayName=("element 1" "element 2" "element 3")
○ Access : ${ArrayName[subscript]}
○ Get number of element
    ▪ ${#ArrayName[@]}
    ▪ ${#ArrayName[*]}
○ Get all element of array: ${ArrayName[*]}
○ Get the index of element: ${!ArrayName[*]}
○ Ex
#!/bin/bash
array=(one two three four [5]=five)
echo "Array size: ${#array[*]}"
echo "Array items:"
for item in ${array[*]} do
    printf "%s\n" $item
done
echo "Array indexes:"
for index in ${!array[*]} do
    printf "%d\n" $index
done
echo "Array items and indexes:"
for index in ${!array[*]} do
    printf "%4d: %s\n" $index ${array[$index]}
done

```

Some the examples and exercises

• Factorial

```

#!/bin/sh
echo "Chương trình tính $1!"
index=0
gt=1
while [ $index -lt $1 ]
do
    index=$((index + 1))
    gt=$((gt * index))
done

```

```
done
echo "$1!= $gt"
exit 0
```

- Count word in file

```
#!/bin/sh
echo "Chương trình đếm số từ của tập tin $1"
{
    n=0
    while read line
    do
        for wd in $line
        do
            n=$((n + 1))
        done
    done
    echo "Tổng số từ của tập tin $1 là : $n"
}<$1
exit 0
```

- Checking user in system

```
tmp=$( grep $1:x /etc/passwd | wc -l )
if [ $tmp -eq 0 ]
then
    echo "User $1 không tồn tại trong hệ thống"
else
    echo "User $1 tồn tại trong hệ thống"
    grep $1:x /etc/passwd
    kt=$( who | grep $1 | wc -l )
    if [ $kt -ne 0 ]
    then
        echo "User $1 đang logon vào hệ thống"
    else
        echo "User $1 không logon vào hệ thống"
    fi
fi
```

- Checking date

```
kiemtra(){
    if [ $1 = "Sat" ]
    then
        echo "Hôm nay là ngày nghỉ"
        sleep 60
        exit
    fi
}
```

```

else
    echo "Hom nay là ngày $1"
fi
}
tmp=$( date | cut -c 3 )
kiemtra $tmp

```

- **Arithmetic Operation**

```

tong=`expr $1 + $2`
echo "Tong của 2 số $1 và $2 là : $tong"
hieu=`expr $1 - $2`
echo "Hiệu của 2 số $1 và $2 là : $hieu"
tich=`expr $1 \* $2`
echo "Tích của 2 số $1 và $2 là : $tich"
th=`expr $1 / $2`
echo "Thương của 2 số $1 và $2 là : $th"

```

- **Luckily Number**

```

secretNumber=$(( (date +%N` / 1000) % 100) + 1 ))
guess=-1
while [ "$guess" != "$secretNumber" ]; do
    echo -n "I am thinking of a number between 1 and 100. Enter your guess:"
    read guess
    if [ "$guess" = "" ]; then
        echo "Please enter a number."
    elif [ "$guess" = "$secretNumber" ]; then
        echo -e "\aYes! $guess is the correct answer!"
    elif [ "$secretNumber" -gt "$guess" ]; then
        echo "The secret number is larger than your guess. Try again."
    else
        echo "The secret number is smaller than your guess. Try again."
    fi
done

```

Submission

Upload the word or pdf file to cms describes some questions as

- Submit the source of the shell script (*.sh) and capture the screen that present the result of shell script program of Fibonacci in Shell.

Requirement

All the capture must be combination with full the windows including your accounts on the windows and the Linux OS (if it is not, you will be taken 0 mark). Should be use the capture in windows with jpg format to reduce the file size with your submitting

END