CSE 314: OS Sessional

Shell Scripts

What is Shell Script?

- Normally shells are interactive.
- It means shell accept command from you (via keyboard) and execute them.
- But if you use command one by one (sequence of n number of commands), the you can store this sequence of command to text file and tell the shell to execute this text file instead of entering the commands.
- This is know as shell script.

Why Shell Script?

- Shell script can take input from user, file and output them on screen.
- Useful to create our own commands.
- Save lots of time.
- To automate some task of day today life.
- System administration part can be also automated.

How to Write and Execute Shell Script?

- Use any editor like emacs/xemacs, vi, nedit or mcedit to write shell script.
- After writing shell script set execute permission for your script

```
$ chmod +x your-script-name
```

- Execute your script as

```
$ ./your-script-name
```

Comment Character

- # is used as the comment character
- A word beginning with # causes that word and all remaining characters on that line to be ignored

Shell to Run

- `#!/bin/bash`
- This indicates that the script should be run in the bash shell regardless of which interactive shell the user has chosen.
- This is very important, since the syntax of different shells can vary greatly.

yourfiles

```
1   #!/bin/bash
2   echo "Hello, $USER."
3   echo "I wish to list some files of yours"
4   echo "Listing files in"
5   echo "the current directory, $PWD"
6   ls  # list files
```

- Notice the comment on line 4.
- USER and PWD are variables. These are standard variables defined by the bash shell itself, they needn't be defined in the script.

Variables in Shell

In Linux (Shell), there are two types of variable:

- System variables - Created and maintained by Linux itself. This type of variables are defined in CAPITAL LETTERS.
- User defined variables (UDV) - Created and maintained by user. This type of variable are normally defined in lower letters.
- You can see system variables by giving command like
  ```
  $ set
  ```
- *Do not modify system the variables, this can some time create problems.*

How to Define User Defined Variables (UDV)?

- `variable_name=value`
- Don’t put spaces on either side of the equal sign when assigning value to variable. There will be problem for the following,
  ```
  $ no =10
  $ no= 10
  $ no = 10
  ```
- Variables are case-sensitive
- Do not use ?, * etc, to name your variable names.

myvars

```
#!/bin/bash
#
myname="Linux Learner"
myos="New OS"
myno=9999
echo "My name is $myname"
echo "My os is $myos"
echo "My number is $myno"
echo "Can you see this number?"
```

Shell Arithmetic

```
expr op1 math-operator op2
```

shellaritmetic

```
#!/bin/bash
expr 1 + 3
expr 2 - 1
expr 10 / 2
expr 20 % 3
expr 10 \* 3
echo ‘expr 6 + 3‘
```

The `read` Statement

- Get input (data from user) from keyboard and store (data) to variable
  `read variable1, variable2,...variableN`

befriends

```
#!/bin/bash
echo "Your name please:"
read fname
echo "Hello $fname, Lets be friends!"
```

Conditionals — `if`

```
if TEST-COMMANDS; then
   CONSEQUENT-COMMANDS;
elif MORE-TEST-COMMANDS; then
   MORE-CONSEQUENT-COMMANDS;
else ALTERNATE-CONSEQUENT-COMMANDS; then
fi
```

Conditionals — `if` — continued

Primary	*Meaning*
`[ -fFILE ]`	True if FILE exists and is a regular file
`[ -z string ]`	If the string is of zero length
`[ STRING1 == STRING2 ]`	True if the strings are equal
`[ STRING1 != STRING2 ]`	True if the strings are not equal
`[ STRING1 < STRING2 ]`	True if “STRING1” sorts before “STRING2” lexicographically
`[ STRING1 > STRING2 ]`	True if “STRING1” sorts after “STRING2” lexicographically
`[ ARG1 OP ARG2 ]`	“OP” is one of -eq, -ne, -lt, -le, -gt or -ge. “ARG1” and “ARG2” are integers.

testnumbers

```
#!/bin/bash
echo "Give me the first number"
read none
echo "Give me the second number"
read ntwo
# Compare the numbers
if [ $none -gt $ntwo ] ; then
    echo "The first number is greater"
elif [ $none -eq $ntwo ] ; then
    echo "The numbers are equal"
else
    echo "The second one is greader"
fi
```

testleapyear

```
#!/bin/bash
# This script will test if we’re in a leap year
year=‘date +%Y‘
if [ $[$year % 400] -eq 0 ]; then
 echo "This is a leap year.  February has 29 days."
elif [ $[$year % 4] -eq 0 ]; then
 if [ $[$year % 100] -ne 0 ]; then
  echo "This is a leap year, February has 29 days."
 else
  echo "This is not a leap year. February has 28 days."
 fi
else
 echo "This is not a leap year. February has 28 days."
fi
```

`for` Loop

Form 1:

```
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
```

Form 2:

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

listusers

```
#!/bin/bash
PASSWORDFILE=/etc/passwd
n=1

for name in $(cut -f 1 -d : $PASSWORDFILE)
do
  echo "User #$n = $name"
  let "n += 1"
done

```

fornumberloop

```
#!/bin/bash
for ((  i = 0 ;  i <= 50;  i++  ))
do
  echo "Welcome $i times"
done
```

Testing and Branching — `case`

```
case "$variable" in

 "$condition1" )
 command...
 ;;

 "$condition2" )
 command...
 ;;

esac
```

Command Line Arguments

Variable	*Meaning*
$*	Command line arguments
$#	Number of arguments
$n	nth argument in $*

carrent

```
#!/bin/bash
# Check for command line argument
if [ -z $1 ]; then
 rental="unknown vehile type"
else
 rental=$1
fi
case $rental in
 "car") echo "For $rental Tk 20 per k/m";;
 "van") echo "For $rental Tk 10 per k/m";;
 "jeep") echo "For $rental Tk 5 per k/m";;
 "bicycle") echo "For $rental Tk 1 per k/m";;
 *) echo "Sorry, I can not gat a/an $rental for you";
esac
```

Practice Problems

1. Write a shell script which shows the number of command line arguments and then prints each argument one each line.
2. Write a shell script which reads a number from keyboard and then prints its square.