Linux Shell Scripts

What is Shell Script ?

- We have seen some basic shell commands, move on to scripts.
- There are two ways of writing shell programs

You can type a sequence of command s shell to execute them interactively·

- You can store those commands in a file t n then invoke as a program.This is know Script·

- We will use bash shell assuming that the h t s installed as ***/bin/sh*** and that it is the defaul your login·

Why Shell Script ?

user, file an ☾ Shell script can take input from

on screen·

them

☾ Useful to create own commands·

☾ Save lots of time·

- To automate some task of day today life·

administration part can be also aut- Systeˢ

How to write and execᵁ

- Use any editor to write shell script·

- The extension is ***.sh***.

- After writing shell script set execute permissions to script.

chmod +x script_name

- Execute your script

- ***./script_name***

- This indicates that the script should be run i

shell regardless of which interactive shell th

chosen·

- This is very important, since the syntax of
  i
d shells can vary greatly·

d
- A word beginning with # causes that wor

remaining characters on that line to be

ign o Shell script format

- Every script starts with the line
- ***#!/bin/bash***
- # is used as the comment character·

A sample shell script

#!/bin/bash

```
echo "HelloUser"
echo "See the files in current directory"
ls
```

Variables

- In Linux (Shell), there are two types of variables
  - System variables - created and maintained by itself.
- ***echo $USER***
- ***echo $PATH***

- User defined variables - created and
  i
ma user·

- All variables are considered and stored as
 t
s when they are assigned numeric values·

- Variables are case sensitive·

Variables

- When assigning a value to a variable, just use the equals sign:
- No spaces on either side of the equals sign.

- ***var_name=value***

- Within the shell we can access the contents

by preceding its name with a ***$***.

myname=A [use quotes if the value

conta ***myos=Linux***

text = 1+2

echoYour name:$myname [*A*]

echoYour os:$myos [*Linux*]

echo $text [*1+2*]

Variables

- If you enclose a $variable expression in double quotes, it's replaced with its value when the line is executed
- If you enclose it in single quotes, no substitution takes place. You can also remove the special meaning of a symbol by prefacing it with a \.

myvar="Hello"

echo $myvar [*Hello*]

echo “$myvar” [*Hello*]

***echo ‘$myvar’*** [*$myvar*]

echo \$myvar [*$myvar*]

echo "Your Age:$stdno”

Read

keyboard and store ☾ To read user input from

variable use ***read var1,var2,.....varn***

#!/bin/bash

echo -n "Enter your name:"

read name

echo -n "Enter your student no:"

read stdno

echo "Your Name:$name"

ShellArithmetic

- The ***expr*** command evaluates its argument expression·

It is commonly used for simple arithmetic

- ```
  #!/bin/bash
  expr 1 + 1
  expr 1 - 1
  expr 1 * 1
  expr 1 / 1
  va r=`expr 1 + 1`
  x=1
  x=`expr $x + 1`
  ```
  ```

ShellArithmetic

If-Else

```
if [ conditiong1 ]; then
statement1
elif [ condition2 ]; then
statement2
else
statement3
fi
```

- It is must to put spaces between the [brace

condition being checked·

- If you prefer putting then on the same line
a
f must add a semicolon to separate the test
then·

If-Else

If-Else

If-Else

#!/bin/bash

```
echo "Enter first number "
read num1
echo "Enter second number"
read num2
if [ $num1 -gt $num2 ] ; then
echo "$num1 is greater than $num2"
elif [ $num1 -lt $num2 ] ; then
echo "$num1 is less than $num2"
else
```

echo "$num1 and $num2 are equal"

fi

- You can put multiple statements between **e** and the next, so a double semicolon is nee**d** where one statement ends and the next pa**t**

Case

case $var in

condition1) statement ;;

condition2) statement ;;

) statement3

esac

d

- Notice that each pattern line is terminate

semicolons **;;** .

Case

```
#!/bin/sh
echo "Is it morning? Please answer yes or no"
read timeofday
case "$timeofday" in
yes) echo "Good Morning";;
no ) echo "Good Afternoon";;
echo "Good Morning";; y )
echo "Good Afternoon";; n )
```

```
    echo "Sorry, answer not recognized";;
esac
```

Case

```
#!/bin/sh
echo "Is it morning? Please answer yes or no"
read timeofday
case "$timeofday" in
yes | y | Yes | YES ) echo "Good Morning";;
echo "Good Afternoon";; n* | N* )
echo "Sorry, answer not recognized";;
```

esac

Command Line argume

- Command line arguments can be passed to scripts.There exists a number of built in var i
  - ***$**** - command line argument
  - ***$#*** - number of argument

- ***$n*** - nth argument in $*

- ***./script_name arg1 arg2 …. arg^{n}***

For

```
for variable in list
do
statement
done
for (( expr1; expr2; expr3 ))
do
statement
done
```

```
[2]
#!/bin/bash
for i in `ls`
do
echo $i
done
[3]
for(( i=0;i<=5;i++ ))
do
echo $i
```

done

For

[1]

```
#!/bin/bash
echo "the number of args is"
$#"
a=1
for i in $*
do
echo "The $a No arg is $i"
a=`expr $a + 1`
done
```

While

```
#!/bin/bash while condition do

password="abc" statements

echo "Enter password" done

read pass
while [ $pass != $pass ]
                    word  do
echo "Wrong Passwo"  read pass
```

done

echo "Write Password"

Until

```
#!/bin/bash until condition do

password="abc" statements

echo "Enter password" done

read pass
until [ $pass = $password ] do
echo "Wrong Password" read pass
```

done

echo "Write Password"

- Functions can be defined in the shell and it i to structure the code·
- To define a shell function simply write its name by empty parentheses and enclose the statement braces·
- Function must be defined before one can

n
i Functions

fun te

ctio me

*n_n nt*s

am

e ()
{

sta

}

Functions

```
#!/bin/sh
foo() {
echo "Function foo is executing"
```

```
}
echo "script starting"
foo
echo "script ending"
```

```
output
scriptstarting
Function foo is executing
script ending
```

```
#!/bin/bash
```

```
showarg()
{
a=1
for i in $*
do
echo "The $a a=`expr $a + 1`"
done
}
echo "Listing st showarg $*"
echo "Total:$#"
echo "Listing E
```

Functions

- When a function is invoked, the parameters to the script ***[$*, $#, $1, $2]*** and so on are replaced by the parameters to the function.
- When the function finishes, they are restored to their previous values.

Function[s]

- Functions can return numeric values using
t
command·

I

- Functions can also return strings by the fol [1]

```
f(){ var="123";}
f

echo $var
```

[2]

```
f(){ echo "123";}
result="$(f)"
if yes_or_no "
echo "Hi $1"
echo "Never"
```

Functions

```
#!/bin/sh
yes_or_no()
{
echo "Is your name $* ?"
echo "Enter yes or no:"
read x
case "$x" in
y | yes ) return 0;;
n | no ) return 1;;
esac
```

```
}
```

Functions

- Be careful：

- Function calling can be recursive·

```
f()
{

statements
f
}
f
```

③ The parameter must be passed every t i is invoked

either from main or from any functions.

Thanks