

Linux Shell Script^s

What is Shell Script ?

- ⌚ We have seen some basic shell commands
move on to scripts.
- ⌚ There are two ways of writing shell progr^a⌚

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

⌚ You can store those commands in a file
t
then invoke as a program. This is known
Script.

⌚ We will use bash shell assuming that the
h
s installed as **/bin/sh** and that it is the default
your login.

Why Shell Script ?

user, file an ☰ Shell script can take input from

on screen.

the

☞ Useful to create own commands.

☞ Save lots of time.

⌚ To automate some task of day today life-

administration part can be also au⌚ Syste⌚

How to write and

exec⌚

⌚ Use any editor to write shell script-

⌚ The extension is *.sh*:

⌚ After writing shell script set execute

S
permi script:

chmod +x script_name ⌚

⌚ Execute your script

⌚ *./script_name*

⌚ This indicates that the script should be run in
shell regardless of which interactive shell the
chosen.

⌚ This is very important, since the syntax of
interactive shells can vary greatly.

⌚ A word beginning with # causes that word
remaining characters on that line to be

ign^o Shell script format^t

- 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

a
vari ⌚ System variables - created and
n
maintai 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

s
u ⌚ No spaces on either side of the equals

sign:

⌚ ***var_name=valu^e***

s
⌚ Within the shell we can access the content

by preceding its name with a \$·

myname=A [use quotes if the value
 |
 conta *myos=LinuX*

text = 1+2

echo Your name:\$myname [A]

echo Your os:\$myos [Linux]

echo \$text [1+2]

VariableS

- ⌚ If you enclose a \$variable expression in
 - do it's replaced with its value when the line is
 - e
 - ⌚ If you enclose it in single quotes, no
 - substi place. You can also remove the special
 - a
 - me symbol by prefacing it with a \·
- myvar="Hello"*

echo \$myvar [Hello]

echo "\$myvar" [Hello]

echo '\$myvar' [\$myvar]

echo \\$myvar [\$myvar]

echo "Your Age:\$stdno"

Rea^d

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 stdn
```

```
echo "Your Name:$name"
```

ShellArithmeti^C

- ⌚ The ***expr*** command evaluates its argument expression.

It is commonly used for simple arithmetic

o **`#!/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 [condition1]; then*ⁿ

statement¹

*elif [condition2]; then*ⁿ

statement²

else^e

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^e

If-Els^e

#!/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"

f

⌚ You can put multiple statements between *d* and the *e*
next, so a double semicolon is nee *d* where one

statement ends and the next pa *t* **Cas*e***

*case \$var i*n**

condition1) statement ;

condition2) statement ;

) *statement3

esa^c

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";i *)*

esa^c

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 r*")
```

esa^c

Command Line arguments

- - ⌚ Command line arguments can be passed to scripts. There exists a number of built-in variables:
 - ⌚ \$* - command line arguments
 - ⌚ \$# - number of arguments

- ⌚ $\$n$ - nth argument in \$^{*}
- ⌚ $./script_name arg1 arg2 arg^n$

For

for variable in list^t

d^o

statement^t

done^e

for ((expr1; expr2; expr3))

d^o

statement^t

done^e

[2]

#!/bin/bash

for i in `ls`

d^o

echo \$ⁱ

done

[3]

for((i=0;i<=5^o d^o

echo \$ⁱ

don^e

Fo^r

[1]

`#!/bin/bash`

echo "the number of args is

`$#"`

`a=1`

for i in \$*

`d0`

echo "The \$a No arg is \$i"

`a=`expr $a + 1``

done

While

`#!/bin/bash while condition do`

`password="abc" statement`s

`echo "Enter password" done`

`read pass`s

`w`
`while [$pass != $pass_r] do`

`echo "Wrong Password"` read pass

done

echo "Write Password"

Until

#!/bin/bash^h until condition do

password="abc" statement^s

echo "Enter password" done^e

read pass^s

until [\$pass = \$password^r] do^o

echo "Wrong Password" read pass^s

done

echo "Write Password"

⌚ Functions can be defined in the shell and it is used to

structure the code.

⌚ To define a shell function simply write its name by
empty parentheses and enclose the statements in braces.

⌚ Function must be defined before one can

iⁿ Functions

function

ction

*n**_n**t*^s

am

e ()
{

sta

}

Function^s

`#!/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 $*
```

```
d0
```

```
echo "The $ a=`expr $a + done`  
{
```

```
echo "Listing s showarg $*
```

```
echo "Total:$#
```

```
echo "Listing E
```

Function^S

When a function is invoked,^C

the parameters to the^e

script [**\$***, **\$#**, **\$1**, **\$2**] and^d

so on are replaced by the^e

parameters to the function.[·]

When the function finishes,^C

they are restored to their^r

previous values.[·]

FunctionS

- ⌚ Functions can return numeric values using
t
command.

- ⌚ 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 the^n  
echo "Hi $1"  
else  
echo "Never $1"
```

FunctionS

```
#!/bin/sh
```

```
yes_no()  
{
```

```
echo "Is your name $* ?"
```

```
echo "Enter yes or no:"
```

```
read X
```

```
case "$X" in
```

```
y | yes ) return 0;;
```

```
n | no ) return 1;;
```

```
esac
```

}

Function^s

⌚ Be careful :

⌚ Function calling can be recursive.

```
f()  
{
```

```
statements  
f  
}  
f
```

③ The parameter must be passed every t is invoked

either from main or from an `y` functions.

Thanks