

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 commands shell
to execute them interactively.

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

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

Why Shell Script ?

user, file and  Shell script can take input from
on screen.
the^m

 Useful to create own commands.

 Save lots of time.

☾ To automate some task of day today life.

administration part can be also aut☾ System

How to write and

exec^U

☾ Use any editor to write shell script.

☾ The extension is ***.sh***.

☾ After writing shell script set execute
permissions

chmod +x script_name ☾

☾ Execute your script

☾ ***./script_name***

☾ This indicates that the script should be run in any shell regardless of which interactive shell has been chosen.

☾ This is very important, since the syntax of different shells can vary greatly.

☾ A word beginning with # causes that word and the remaining characters on that line to be

ign^o Shell script format^t

☾ Every script starts with the lin^e

☾ *#!/bin/bas^h*

☾ # is used as the comment character.

A sample shell scrip^t

#!/bin/bas^h

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
maintain 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

u ☞ No spaces on either side of the equals sign.

☞ ***var_name=value***

☞ Within the shell we can access the content

by preceding its name with a \$.

myname=A [use quotes if the value
contains spaces]
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 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 symbol by prefacing it with a backslash.

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^h

echo -n "Enter your name:"

read nam^e

echo -n "Enter your student no:"

read stdn^o

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'

ShellArithmeti^C

If-Else^e

*if [condition₁]; theⁿ
statement¹*

*elif [condition₂]; theⁿ
statement²*

els^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^e

If-Else^e

If-Else^e

#!/bin/bas^h

```
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 need^d where one

statement ends and the next pa^t **Case**

case \$var iⁿ

condition1) statement ;i

condition2) statement ;i

****) statement³***

esa^c

⌋ Notice that each pattern line is terminate^d
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"; *)*

esa^C

Cas^e

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

```
echo "Is it morning? Please answer yes or n"
```

```
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 argument^e

⌋ Command line arguments can be passed to scripts. There exists a number of built in variables ⌋ $\$*$ - command line arguments^s

⌋ $\$ \#$ - number of arguments^s

☾ **$\$n$** - nth argument in $\*

☾ ***./script_name arg1 arg2 argⁿ***

For^r

for variable in list^t

do^o

statement^t

done^e

for ((expr1; expr2; expr3))

do^o

statement^t

done^e

[2]

#!/bin/bas^h

for i in `ls`

d⁰

echo \$ⁱ

don^e

[3]

for((i=0;i<=5⁰ d⁰

echo \$ⁱ

don^e

For

[1]

#!/bin/bas^h

echo "the number of args i^s

\$#"

a=1

*for i in \$**

do

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

a=`expr \$a + 1`

done^e

while

#!/bin/bash while condition do

password="abc" statement

echo "Enter password" done

read pass

while [\$pass != \$pass_r do

echo "Wrong Password" read pass

done^e

echo "Write Password"

Until

#!/bin/bash^h until condition d^o

password="abc" statement^s

echo "Enter password" don^e

read pas^s

until [\$pass = \$passw^o_r d^o

echo "Wrong Passwo read pas^s

done

echo "Write Password"

☾ Functions can be defined in the shell and it is 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

ⁿ i Function^s

fun te

ctiome

n_nnt^s

am

e ()

{

sta

}

Function^S

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

```
foo() {
```

```
  echo "Function foo is executing"
```


}

echo "script starting"

fo^o

echo "script ending"

output^t

scriptstartin^g

Function foo is executin^g

script endin^g

#!/bin/bas^h

```
showarg(  
{
```

```
a=1
```

```
for i in $*
```

```
do
```



```
    a=a+`expr $a + 1`  
    echo "The $a  
}
```

```
    t=$((t+a))  
echo "Listing $ showarg $*
```

```
echo "Total:$t"
```

```
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.

Functions

☾ Functions can return numeric values using the `echo` command.

☾ Functions can also return strings by the following example [1]

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

```
echo $var
```

[2]

```
f(){ echo "123"; }  
result="$(f)"  
if yes_or_no then  
echo "Hi $1" else  
echo "Neve fi"
```

Function^s

```
#!/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()
{

statement^s

f
}
f

③ The parameter must be passed every ⁱ t is invoked

either from main or from an^y functions.

Thank^S