Shell Script

Dr. Vivek Shrivastava
IIPS, DAVV

Shell Scripting #1

- #!/bin/sh
- # hi
- echo "Hello, world!"
- exit 0

Shell Scripting #2

- #!/bin/sh
- # himike
- name=Mike
- echo "Hello, $name!"
- exit 0

Shell Scripting #3

- #!/bin/sh
- # rem
- rm junk
- echo "The return code from rm was $?"
- exit 0

- The return code from rm was 0/2→ exist/not exist

Shell Scripting #4

- #!/bin/sh
- # quiet
- rm junk 2> /dev/null
- echo "The return code from rm was $?"
- exit 0

Shell Scripting #4

- #!/bin/sh
- # pars echo
- "There are $# parameters."
- echo "The parameters are $@"
- echo "The script name is $0"
- echo "The first parameter is $1"
- echo "The second parameter is $2"
- exit 0
- To run:-pars apple orange

Shell Scripting #5

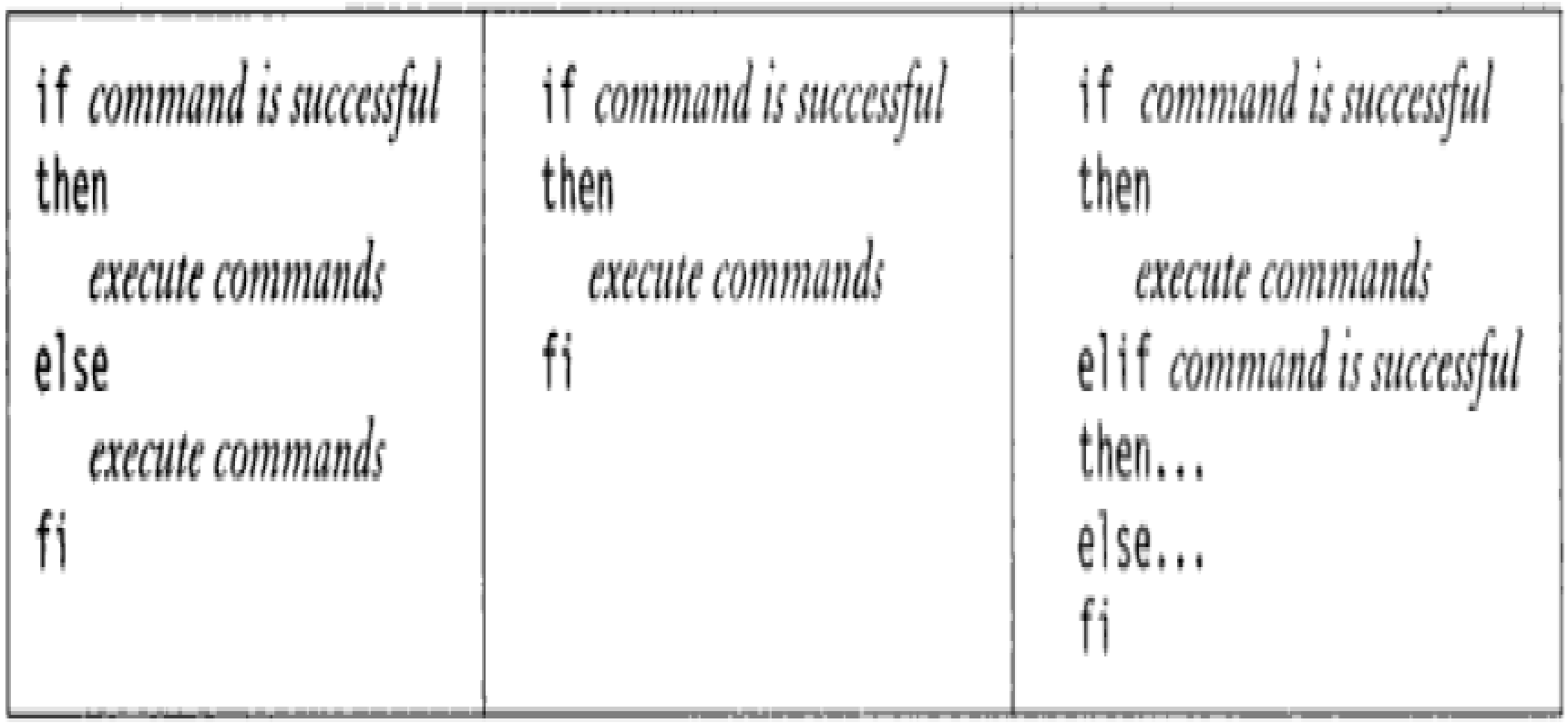
- #!/bin/sh
- # hiyou
- name=‘whoami‘
- echo "Hello, $name!"
- exit 0

Shell Scripting #6

```
#!/bin/sh
#  countem
echo "File \"$1\" contains \
exactly ‘wc $1 | cut -c6-7‘ lines."
exit 0
```

```
unix[25] countem text
File "text" contains exactly 21 lines.
```

The if Conditional



Shell Scripting #7

```
$ emp3.sh ftp
ftp:*:325:15:FTP User:/users1/home/ftp:/bin/true
Pattern found - Job Over
$ emp3.sh mail
Pattern not found
```

```
#!/bin/sh
# emp3.sh: Using if and else
#
if grep "^$1" /etc/passwd 2>/dev/null   # Search username at beginning of line
then
        echo "Pattern found - Job Over"
else
        echo "Pattern not found"
fi
```

Fig. 14.4 emp3.sh

14.7 USING test AND [] TO EVALUATE EXPRESSIONS

- Compares two numbers.
- Compares two strings or a single one for a null value.
- Checks a file's attributes.

and numerically compare them:

```
$ x=5; y=7; z=7.2
$ test $x -eq $y ; echo $?
1
$ test $x -lt $y ; echo $?
0
$ test $z -gt $y ; echo $?
1
$ test $z -eq $y ; echo $?
0
```

Not equal

True

7.2 is not greater than 7!

7.2 is equal to 7!

Shell script for finding greatest of two numbers

```
#shell script to find the greatest of two numbers

echo "Enter Num1"
read num1
echo "Enter Num2"
read num2

if [ $num1 -gt $num2 ]
then
    echo $num1
else
    echo $num2
fi
```

read function used to get the input from the user.

-gt stands for **g**reater **t**han.

Shell script for finding greatest of three numbers

```
#shell script to find the greatest of three numbers

echo "Enter Num1"
read num1
echo "Enter Num2"
read num2
echo "Enter Num3"
read num3

if [ $num1 -gt $num2 ] && [ $num1 -gt $num3 ]
then
    echo $num1
elif [ $num2 -gt $num1 ] && [ $num2 -gt $num3 ]
then
    echo $num2
else
    echo $num3
fi
```

read function used to get the input from the user.

-gt stands for **g**reater **t**han.

&& represents the logical AND condition.

```
#shell script to print numbers 1 to 100
#using while loop and expr

i=1
while [ $i -le 100 ]
do
    echo $i
    i=`expr $i + 1`
done
```

Shell script to print sum of all digit using expr

```
#sum of all digits - shell script

echo "Enter a number"
read num

sum=0

while [ $num -gt 0 ]
do
    mod=`expr $num % 10`    #It will split each digits
    sum=`expr $sum + $mod`  #Add each digit to sum
    num=`expr $num / 10`    #divide num by 10.
done

echo $sum
```

For loop

```
#!/bin/sh
#   adder
sum=0
for x in $@
do
   sum=‘expr $sum + $x‘
done
echo "The sum is $sum."
exit 0
```

```
unix[44] adder 1 2 3 4 5
The sum is 15.
```

```
case EXPRESSION in

   PATTERN_1)
      STATEMENTS
      ;;

   PATTERN_2)
      STATEMENTS
      ;;

   PATTERN_N)
      STATEMENTS
      ;;

   *)
      STATEMENTS
      ;;
esac
```

```
#!/bin/bash

echo -n "Enter the name of a country: "
read COUNTRY

echo -n "The official language of $COUNTRY is "

case $COUNTRY in

  Lithuania)
    echo -n "Lithuanian"
    ;;

  Romania | Moldova)
    echo -n "Romanian"
    ;;

  Italy | "San Marino" | Switzerland | "Vatican City")
    echo -n "Italian"
    ;;

  *)
    echo -n "unknown"
    ;;
esac
```

```
$ cat yorno.sh
#!/bin/bash

echo -n "Do you agree with this? [yes or no]: "
read yno
case $yno in

        [yY] | [yY][Ee][Ss] )
                echo "Agreed"
                ;;

        [nN] | [n|N][O|o] )
                echo "Not agreed, you can't proceed the installation";
                exit 1
                ;;
        *) echo "Invalid input"
            ;;
esac
```

cpio command

- **cpio** stands for "**copy in, copy out**". It is used for processing the archive files like **.cpio* or **.tar*.
- This command can copy files to and from archives.
- **Copy-out Mode:** Copy files named in name-list to the archive
  - cpio -o < name-list > archive
- **Copy-in Mode:** Extract files from the archive
  - cpio -i < archive

```
linux@ubuntu:~/files$ ls
file  file2
linux@ubuntu:~/files$ ls | cpio -ov > /home/linux/compress.cpio
file
file2
1 block
linux@ubuntu:~/files$
```

```
linux@ubuntu:~/files$ ls
linux@ubuntu:~/files$ cpio -iv < /home/linux/compress.cpio
file
file2
1 block
linux@ubuntu:~/files$
```

Command:-dd

- **dd** is a command-line utility for Unix and Unix-like operating systems whose primary purpose is to convert and copy files.
- **To backup the entire harddisk**
  - **dd if = /dev/sda of = /dev/sdb**
  - *"if"* represents inputfile, and *"of"* represents output file. So the exact copy of */dev/sda* will be available in */dev/sdb*.

Unix / Linux - Using Shell Variables

Variable Names

The name of a variable can contain only letters (a to z or A to Z), numbers (0 to 9) or the underscore character (_).

By convention, Unix shell variables will have their names in UPPERCASE.

The following examples are valid variable names −

```
_ALI
TOKEN_A
VAR_1
VAR_2
```

Following are the examples of invalid variable names −

```
2_VAR
-VARIABLE
VAR1-VAR2
VAR_A!
```

The reason you cannot use other characters such as **!**, *, or **-** is that these characters have a special meaning for the shell.

Defining Variables

Variables are defined as follows –

```
variable_name=variable_value
```

For example –

```
NAME="Zara Ali"
```

The above example defines the variable NAME and assigns the value "Zara Ali" to it. Variables of this type are called **scalar variables**. A scalar variable can hold only one value at a time.

Shell enables you to store any value you want in a variable. For example –

```
VAR1="Zara Ali"
VAR2=100
```

Accessing Values

To access the value stored in a variable, prefix its name with the dollar sign (**$**) –

```
NAME="Zara Ali"
echo $NAME
```

The above script will produce the following value −

```
Zara Ali
```

Read-only Variables

Shell provides a way to mark variables as read-only by using the read-only command. After a variable is marked read-only, its value cannot be changed.

For example, the following script generates an error while trying to change the value of NAME −

Live Demo

```
#!/bin/sh

NAME="Zara Ali"
readonly NAME
NAME="Qadiri"
```

The above script will generate the following result −

```
/bin/sh: NAME: This variable is read only.
```

Unsetting Variables

Unsetting or deleting a variable directs the shell to remove the variable from the list of variables that it tracks. Once you unset a variable, you cannot access the stored value in the variable.

Following is the syntax to unset a defined variable using the **unset** command −

```
unset variable_name
```

The above command unsets the value of a defined variable. Here is a simple example that demonstrates how the command works −

```
#!/bin/sh

NAME="Zara Ali"
unset NAME
echo $NAME
```

The above example does not print anything. You cannot use the unset command to **unset** variables that are marked **readonly**.

Shell Variables

Several system defined variables are set for you when you log in.

Name	Meaning
$HOME	Absolute pathname to your home directory.
$PATH	List of directories to search for commands.
$USER	Your user name.
$SHELL	Absolute pathname of your login shell.
$TERM	The type of your terminal.

Environment variables

- Environment variables are variables that are set up in your shell when you log in.
- They are called “environment variables” because most of them affect the way your Unix shell works for you.
  - One points to your home directory and another to your history file.
  - One identifies your mail file, while another controls the colors that you see when you ask for a file listing.
  - Still another sets up your default search path.

How to set environment variables?

To set a global ENV

```
$ export NAME=Value
or
$ set NAME=Value
```

To set a local ENV

Bibliography

1. https://www.rpi.edu/dept/arc/training/shell/slides.pdf
2. Unix Concepts And Applications , Das, Sumitabha - Operating systems (Computers) - 2006
3. https://www.log2base2.com/shell-script-examples
4. https://linuxize.com/post/bash-case-statement/
5. https://image.slidesharecdn.com/slides-170508152118/95/slides-28-638.jpg?cb=1494256903

Bibliography

6. https://www.geeksforgeeks.org/cpio-command-in-linux-with-examples/
7. https://www.tutorialspoint.com/unix/unix-using-variables.htm
8. https://www.geeksforgeeks.org/environment-variables-in-linux-unix/