Shell Programming

Conditional Execution

- Without control statements, execution within a shell scripts flows from **one statement to the next in succession.**
- **Control statements** control the flow of execution in a programming language

- The three most common types of control statements:
  - Conditionals: if/then/else, case, ...
  - Loop statements: while, for, until, do, ...
  - Branch statements: subroutine calls (good), goto (bad)

The Logical Operators (&& , ||)

- The shell provides two operators that allow conditional execution

Syntax

Command1 && Command2

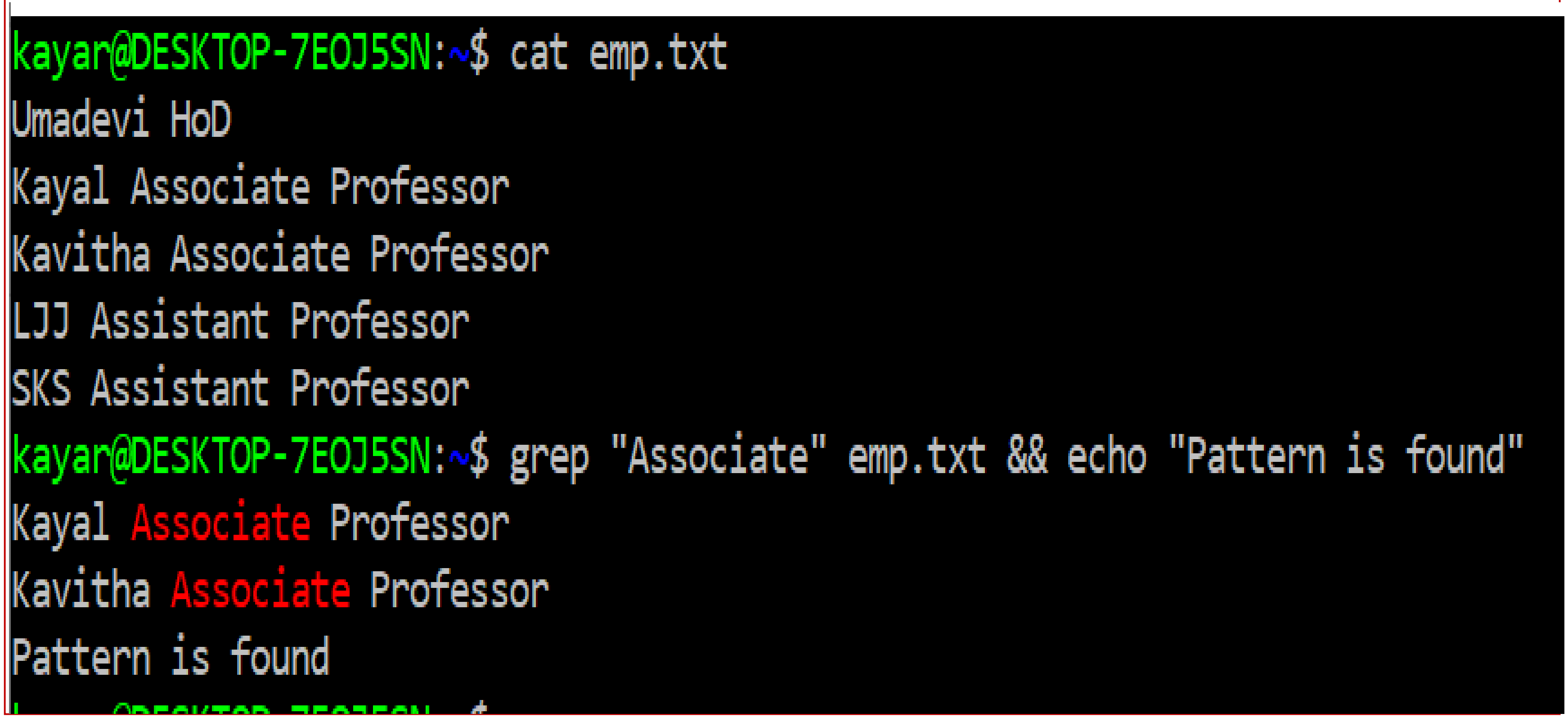
Note : Command 2 is executed only when Command1 succeeds

Command1 || Command2

Note: The second command is executed only when the first fails

Example

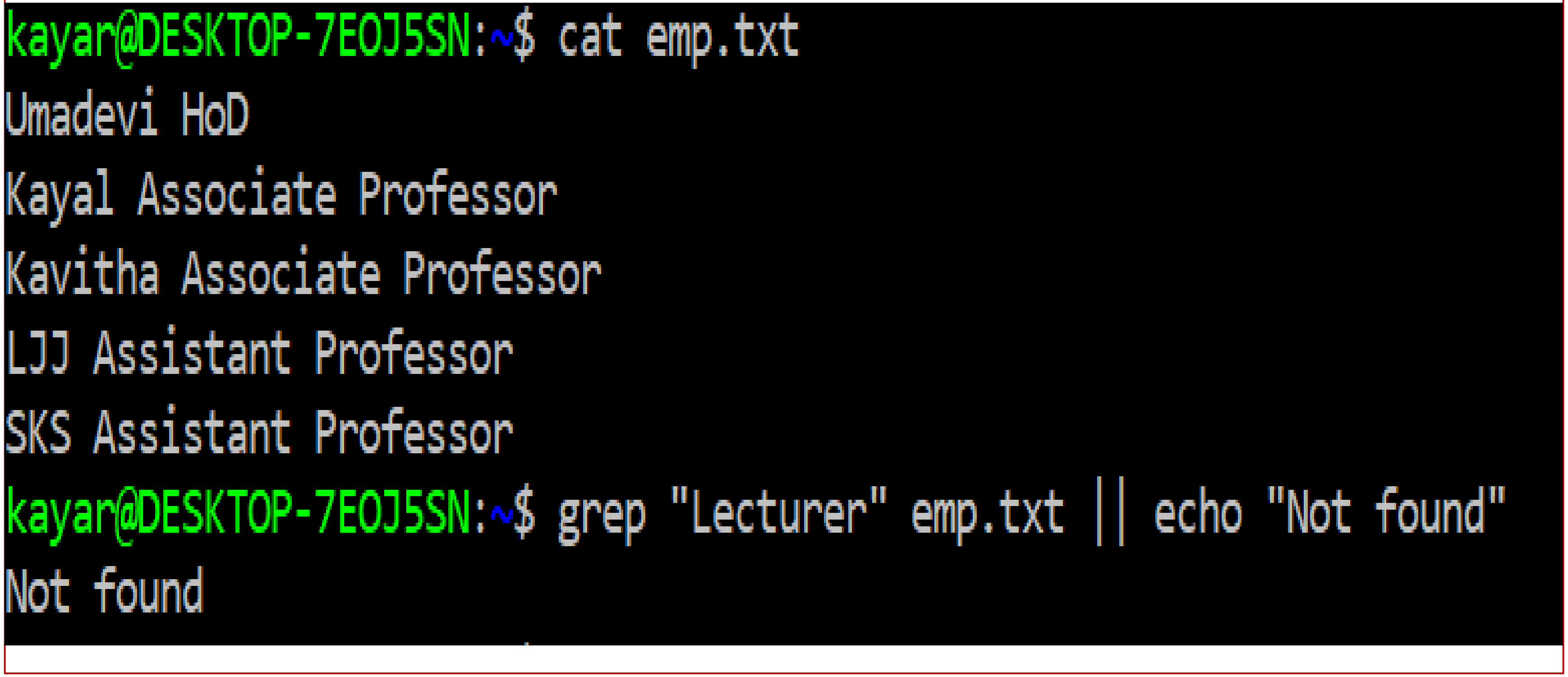
grep “Associate” emp.txt && echo “Pattern is found”



```
kayar@DESKTOP-7EOJ5SN:~$ cat emp.txt
Umadevi HoD
Kayal Associate Professor
Kavitha Associate Professor
LJJ Assistant Professor
SKS Assistant Professor
kayar@DESKTOP-7EOJ5SN:~$ grep "Associate" emp.txt && echo "Pattern is found"
Kayal Associate Professor
Kavitha Associate Professor
Pattern is found
kayar@DESKTOP-7EOJ5SN:~$
```

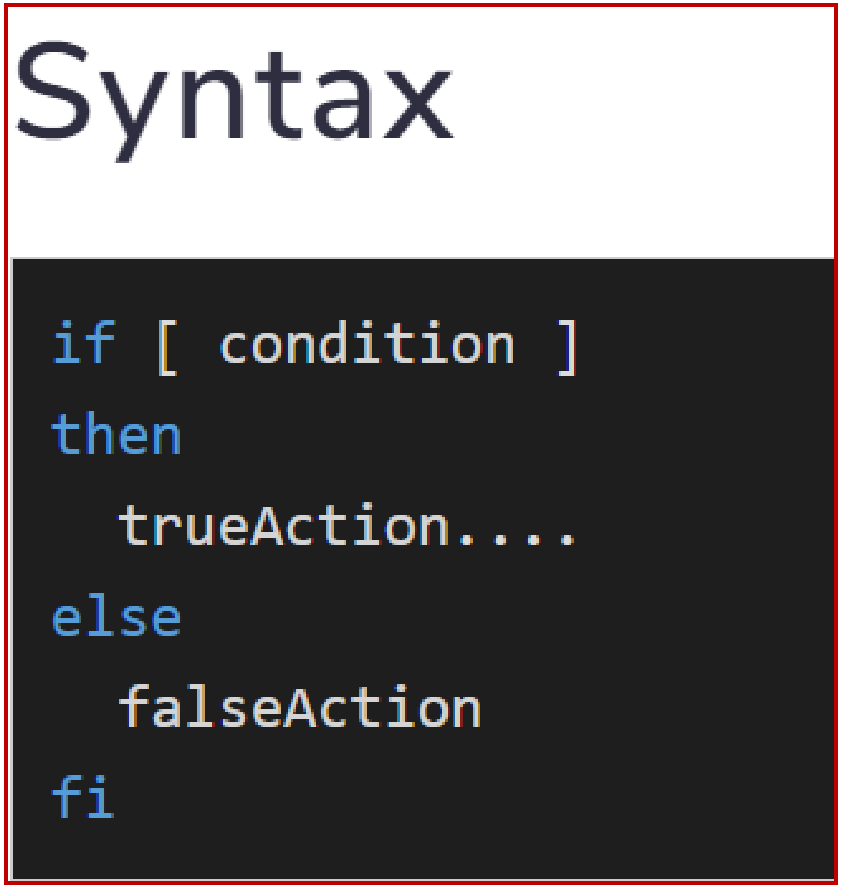
Example

grep “Lecturer” emp.txt | | echo “Not found”



```
kayar@DESKTOP-7EOJ5SN:~$ cat emp.txt
Umadevi HoD
Kayal Associate Professor
Kavitha Associate Professor
LJJ Assistant Professor
SKS Assistant Professor
kayar@DESKTOP-7EOJ5SN:~$ grep "Lecturer" emp.txt || echo "Not found"
Not found
```

if Condition



Syntax

```
if [ condition ]
then
  trueAction....
else
  falseAction
fi
```

Using test and [] to evaluate expression

- **test command** is used to evaluate expression in the if condition
- true or false returned by expression can’t be directly handled by if
- Test handles it and **returns either true or false exit status**, which is then used by if for making decision

test command

test works in three ways

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

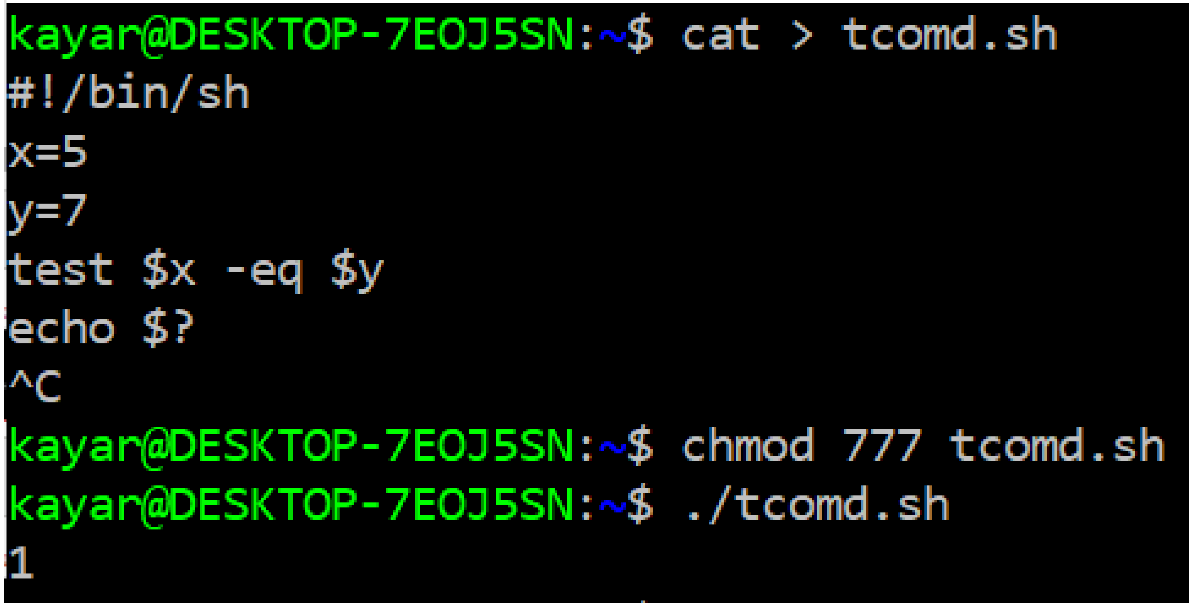
Numeric Comparison

Numerical Comparison operators used by test

Operator	Meaning
-eq	Equal to
-ne	Not Equal to
-gt	Greater than
-ge	Greater than or equal to
-lt	Less than
-le	Less than equal to

Numeric Comparison -Example

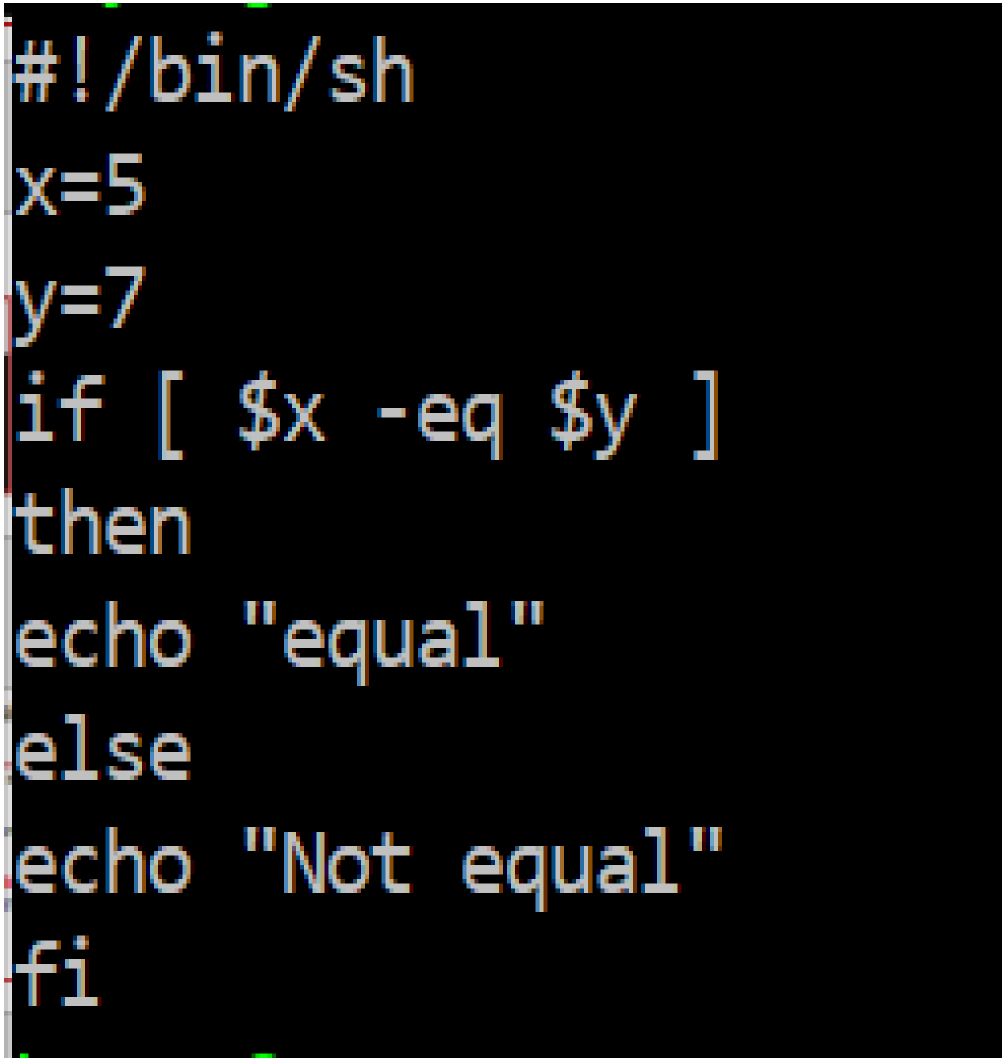
```
x=5
y=7
test $x –eq $y
echo $?
```



```
kayar@DESKTOP-7EOJ5SN:~$ cat > tcomd.sh
#!/bin/sh
x=5
y=7
test $x -eq $y
echo $?
^C
kayar@DESKTOP-7EOJ5SN:~$ chmod 777 tcomd.sh
kayar@DESKTOP-7EOJ5SN:~$ ./tcomd.sh
1
```

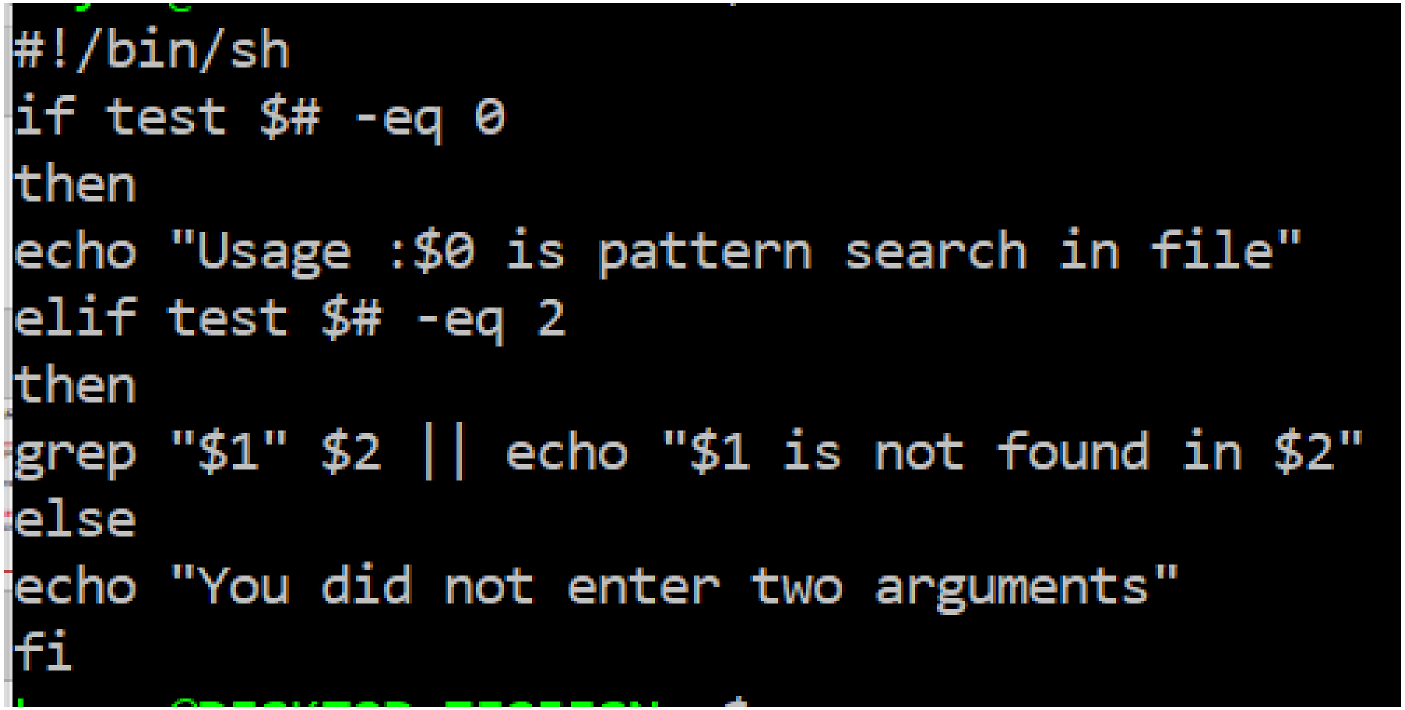
if - example

```
#!/bin/sh
a=10
b=20
 if [ $a –eq $b ]
then
   echo "a is equal to b"
fi
if [ $a -ne $b ]
then
   echo "a is not equal to b"
fi
```

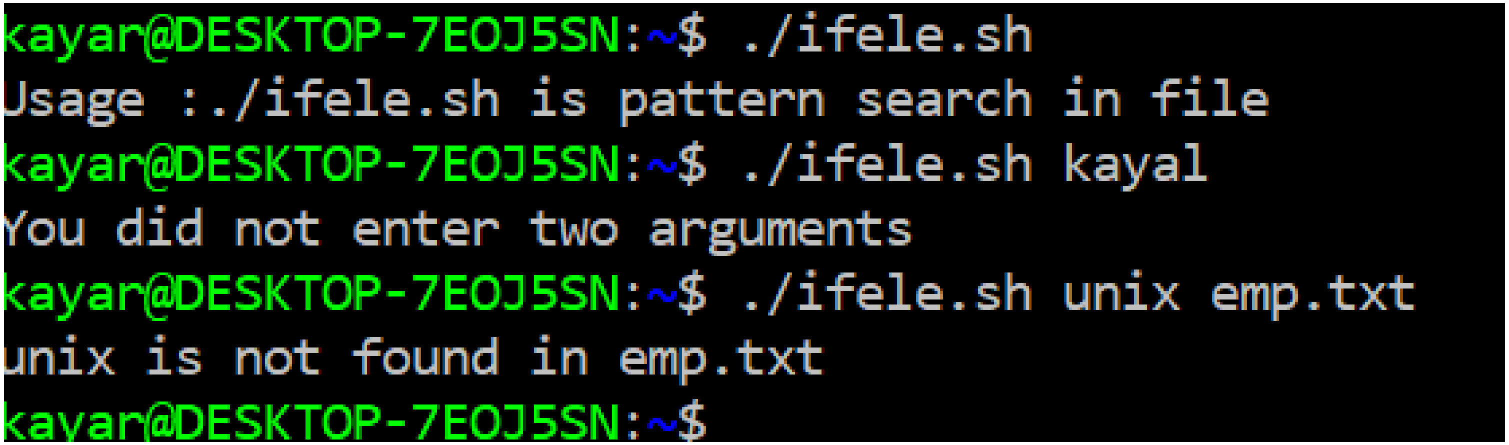


```
#!/bin/sh
x=5
y=7
if [ $x -eq $y ]
then
echo "equal"
else
echo "Not equal"
fi
```

If elif - Example



```
#!/bin/sh
if test $# -eq 0
then
echo "Usage :$0 is pattern search in file"
elif test $# -eq 2
then
grep "$1" $2 || echo "$1 is not found in $2"
else
echo "You did not enter two arguments"
fi
```



```
kayar@DESKTOP-7EOJ5SN:~$ ./ifele.sh
Usage :./ifele.sh is pattern search in file
kayar@DESKTOP-7EOJ5SN:~$ ./ifele.sh kayal
You did not enter two arguments
kayar@DESKTOP-7EOJ5SN:~$ ./ifele.sh unix emp.txt
unix is not found in emp.txt
kayar@DESKTOP-7EOJ5SN:~$
```

Exercise

Write a script that executes the command “cat/etc/shadow”. If the command return a 0 exit status, report “command succeeded” and exit with a 0 exit status. If the command returns a non-zero exit status, report “Command failed” and exit with a 1 exit status

Answer

```
#!/bin/sh
cat /home/shadow
if [ “$?” -eq  0 ]
  then
    echo “Command succeeded”
    exit 0
  else
    echo “Command failed”
    exit 1
fi
```

String Comparison

- Test can be used to compare strings

Test	True if
s1=s2	String s1=s2
s1 != s2	String s1 is not equal to s2
-n stg	String stg is not a null string
-z stg	String stg is a null string
s1==s2	String s1=s2 (Korn Shell)

String Comparison

```
#!/bin/sh


VAR1="Linuxize"
VAR2="Linuxize"


if [ "$VAR1" = "$VAR2" ]; then
    echo "Strings are equal."
else
    echo "Strings are not equal."
fi
```

```
#!/bin/sh

strval1="Ubuntu"

if [ $strval1 == “Windows” ]
then
echo "Strings are equal"
else
  echo "Strings are not equal"
fi
```

```
#!/bin/sh
if [ $# -eq 0 ] ; then
   echo “Enter the string to be searched :\c”
   read pname
   if [ -z “$pname” ]; then                    # -z checks for a null string
    echo “you have not entered the string “; exit 1
  fi
 echo “Enter the file name to be used :\c”
  read flname
  if [ ! –n “$flname” ]; then
      echo “You have not entered the filename”; exit 2
  fi
ifele.sh “$pname” “$flname”
else
     ifele.sh $*
fi
```

AND (-a) and OR (-o) Operators

```
#!/bin/sh
if [ $# -eq 0 ] ; then
   echo “Enter the string to be searched :\c”
   read pname
   echo “Enter the file name to be used :\c”
  read flname
   if [ -n  “$pname”  –a  –n “$flname”  ] ; then
   ifele.sh “$pname” “$flname”
   else
    echo “At least one input was a null string”; exit 1
    fi
fi
```

File Tests

- Test can be used to test the various file attributes (file, directory or symbolic link) or its permissions (read, write, execute, SUID etc)

Example

```
[ -f emp.txt ] ; echo $?
0
```

File Related Tests

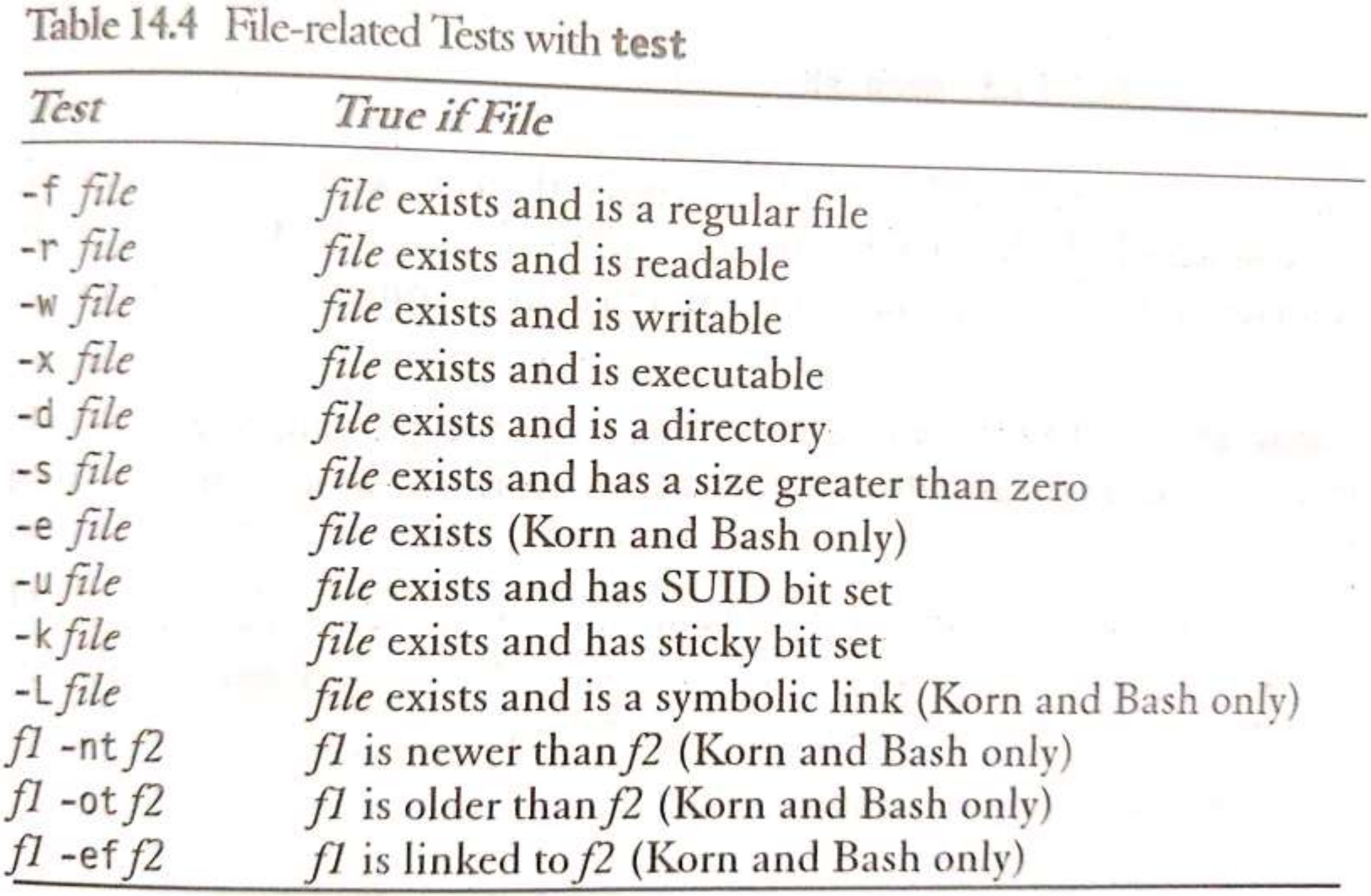
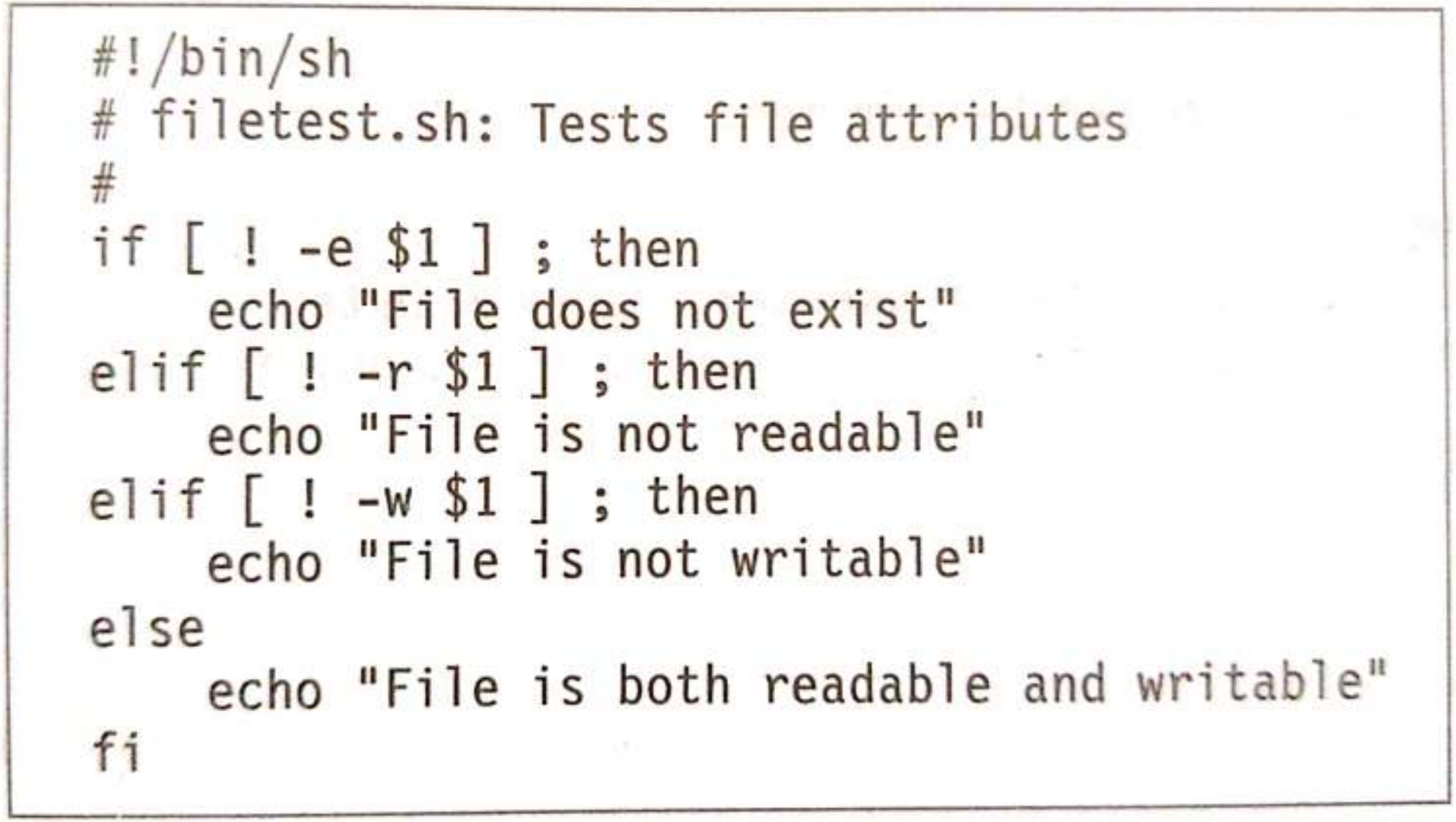


Table 14.4 File-related Tests with **test**

Test	*True if File*
`-f` *file*	*file* exists and is a regular file
`-r` *file*	*file* exists and is readable
`-w` *file*	*file* exists and is writable
`-x` *file*	*file* exists and is executable
`-d` *file*	*file* exists and is a directory
`-s` *file*	*file* exists and has a size greater than zero
`-e` *file*	*file* exists (Korn and Bash only)
`-u` *file*	*file* exists and has SUID bit set
`-k` *file*	*file* exists and has sticky bit set
`-L` *file*	*file* exists and is a symbolic link (Korn and Bash only)
f1 `-nt` *f2*	*f1* is newer than *f2* (Korn and Bash only)
f1 `-ot` *f2*	*f1* is older than *f2* (Korn and Bash only)
f1 `-ef` *f2*	*f1* is linked to *f2* (Korn and Bash only)

Example



```
#!/bin/sh
# filetest.sh: Tests file attributes
#
if [ ! -e $1 ] ; then
    echo "File does not exist"
elif [ ! -r $1 ] ; then
    echo "File is not readable"
elif [ ! -w $1 ] ; then
    echo "File is not writable"
else
    echo "File is both readable and writable"
fi
```

The case Conditional

- The case statement is the second conditional statement offered by the shell
- It is used for multiway branching
- case also handles string tests

Syntax

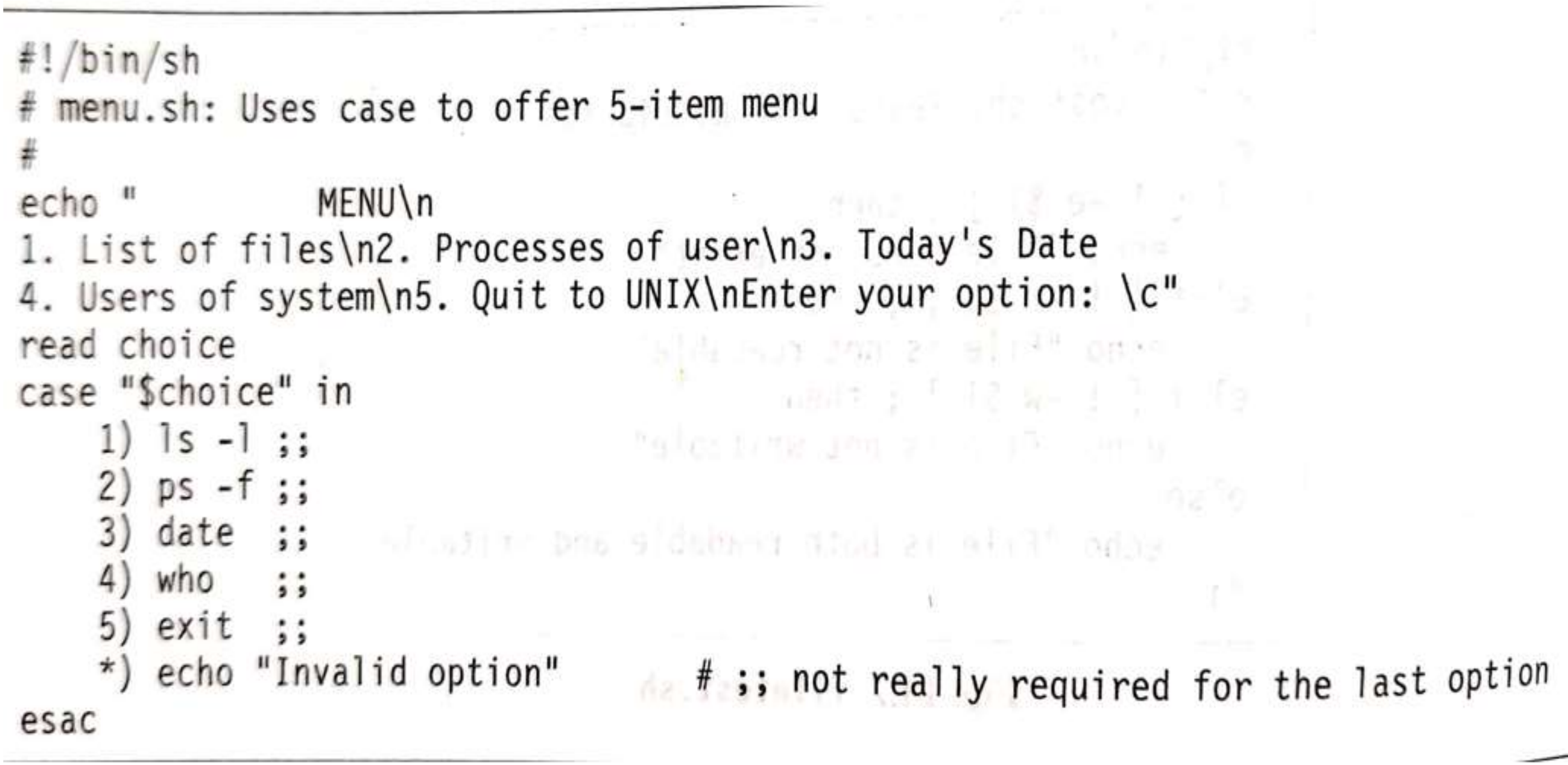
```
case expression in
       pattern1) command1 ;;
        pattern1) command1 ;;
        pattern1) command1 ;;
        ……..
esac
```

```
#!/bin/sh

FRUIT="kiwi"

case "$FRUIT" in
  "apple") echo "Apple pie is quite tasty."
  ;;
  "banana") echo "I like banana nut bread."
  ;;
  "kiwi") echo "New Zealand is famous for kiwi."
  ;;
esac
```

Example



```
#!/bin/sh
# menu.sh: Uses case to offer 5-item menu
#
echo "          MENU\n
1. List of files\n2. Processes of user\n3. Today's Date
4. Users of system\n5. Quit to UNIX\nEnter your option: \c"
read choice
case "$choice" in
    1) ls -l ;;
    2) ps -f ;;
    3) date  ;;
    4) who   ;;
    5) exit  ;;
    *) echo "Invalid option"       # ;; not really required for the last option
esac
```

Matching Multiple Patterns

- case can also specify the same action for more than one pattern
- case uses | to delimit multiple patterns

```
#!/bin/sh
echo “Do you wish to continue? (y/n): \c”
read answer
case “$answer” in
Y|y) ;;
N|n) exit ;;
esac
```

Wild-cards : case Uses them

- case has string matching feature that uses wild cards

```
#!/bin/sh
echo “Do you wish to continue? (y/n): \c”
read answer
case “$answer” in
[Y|y][eE]*) ;;
[N|n][oO]*) exit ;;
esac
```

Arithmetic Operations Using **`expr`**

- The shell is not intended for numerical work (use Java, C, or Perl instead).
- However, **`expr`** utility may be used for *simple* arithmetic operations on integers.
- **`expr`** is not a shell command but rather a UNIX utility.
- To use **`expr`** in a shell script, enclose the expression with backquotes.
- Example:

```
#!/bin/sh
sum=`expr $1 + $2`
echo $sum
```

- Note: spaces are required around the operator **+** (but not allowed around the equal sign).

expr : Computation and String Handling

- expr is used to do the following operations
  - Performs arithmetic operations on integers
  - Manipulating strings

Computation

- expr can perform the four basic arithmetic operations as well as the modulus (reminder) function

Example-1

```
expr 3 + 5
```

Example-2

```
x=3 ; y=4
expr $x + $y
```

Computation

Example -3 (Asterisk symbol should be used escaped to use as multiplication)

```
expr 3 \* 5
```

Example – 4 (To assign the value of the resultant expression)

```
x=6; y=2
z=`expr $x + $y`
echo $z
```

((..)) notation

- ((..)) notation can be used for integer computation
- The usage of $((...)) and ((...)) notation
- The difference is that $((...)) returns the result of the calculation and ((...)) does not.

Example

```
a=5; b=7
c=$(( $a + $b))
echo $c
```

Math in shell script -bc

- What if you want to do math with floating point numbers
- The bc command is needed. But you have to treat the variables as strings.
- An arbitrary precision calculator language. **bc** may either be run interactively, or as a shell script command. In interactive mode, type ctrl-d (EOF) to exit.

Math in shell script -bc

Example

```
r=3.5
s=`echo “$r +2.2” | bc`
echo $s
```

String Handling

- For manipulating string, expr uses two expressions separated by a colon
- The string to be worked upon is placed on the left side of the :
- The regular expression is placed on its right

Example

expr “abcdefgh” : ‘.*’

Note :it prints no of character matches the pattern. (i.e)the length of the entire strings

String Handling

Expr can perform three important string function

- Determine the length of the string
- Extract a substring
- Locate the position of a character in a string

String Handling

- **Determine the length of the string**

```
expr “abcdefgh” :  ‘.*’
```

- **Extract a substring**

expr can extract a string enclosed by the escaped characters \(and \)

```
stg=2004
expr “$stg” : ‘..\(..\)’
```

Note: The above example extracts last two characters

String Handling

- Locating Position of a character

expr can return the location of the first occurrence of a character inside a string

```
Stg=“Kayarvizhy”
expr “$stg” : ‘[^v]*v’
```

Note : the above example returns 6.

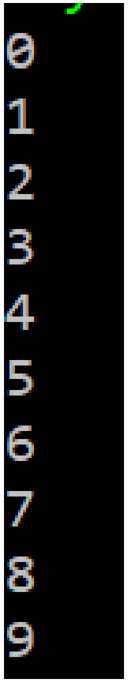
`while` Loops

```
while condition
do
    command(s)
done
```

- Command **`test`** is often used in ***`condition`***.
- Execute ***`command(s)`*** when ***`condition`*** is met.

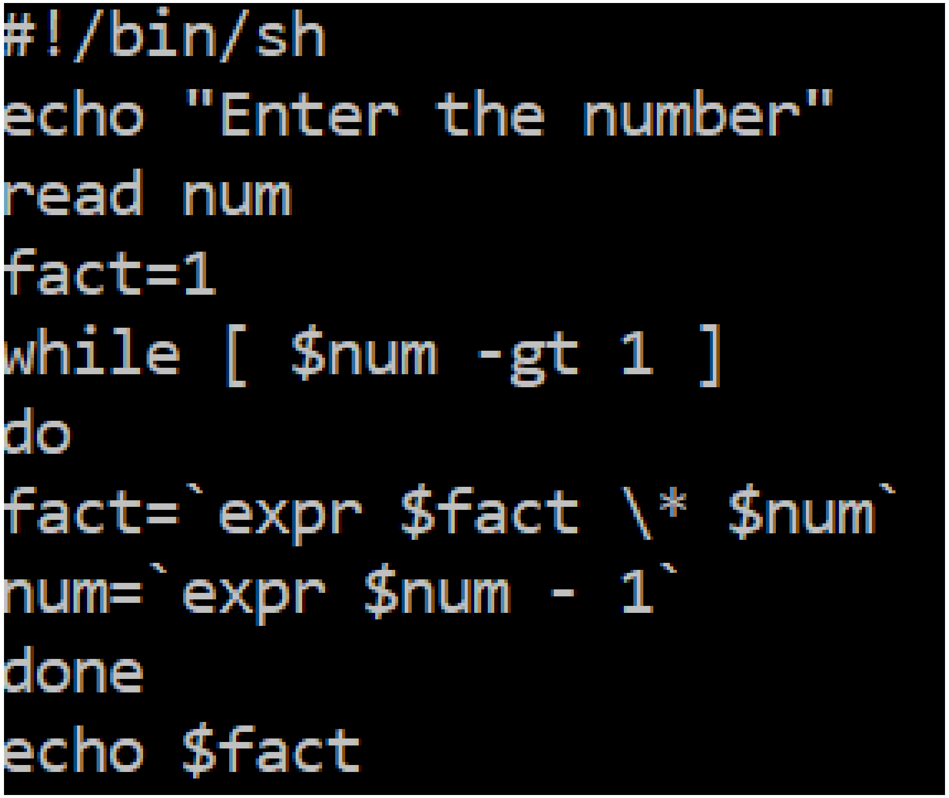
Example

```
#!/bin/sh
count=0
while [ $count –lt 10 ]
do
  echo $count
  count=`expr $count + 1`
done
```



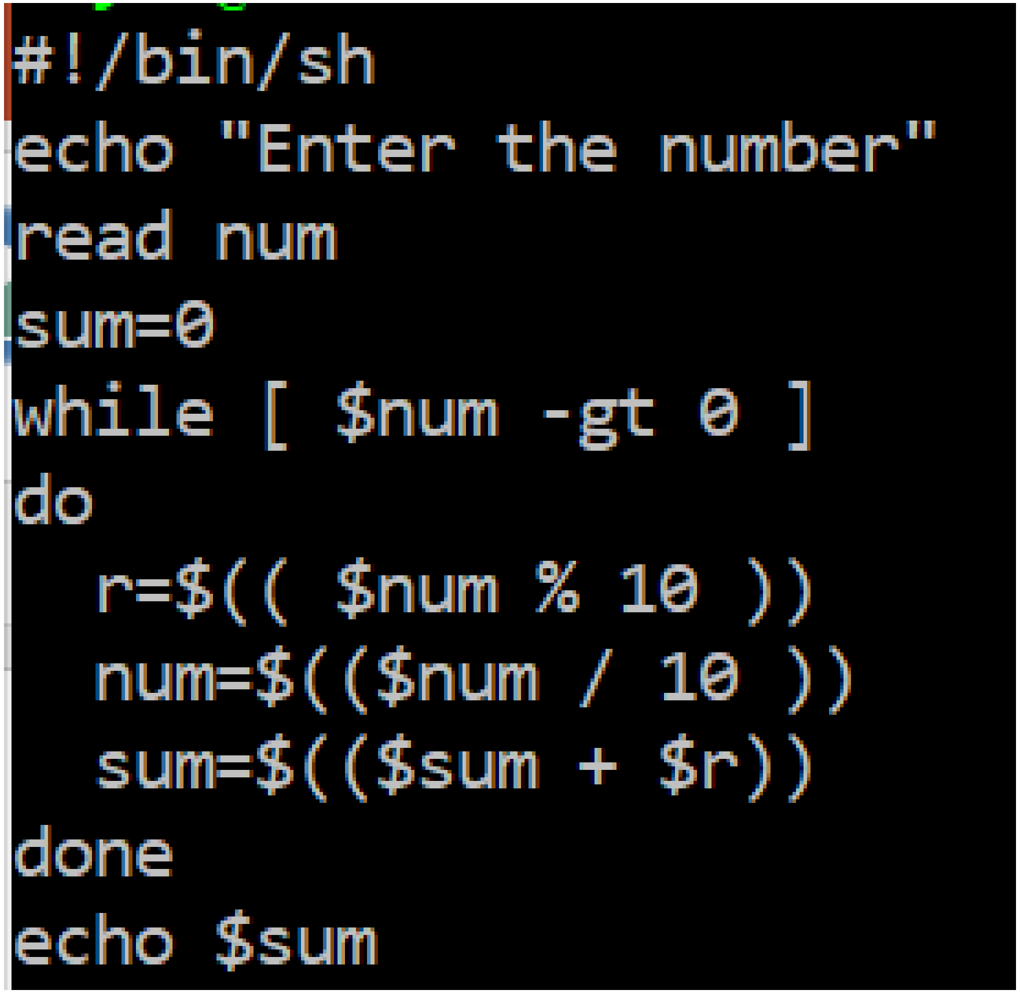
```
0
1
2
3
4
5
6
7
8
9
```

Factorial of a number

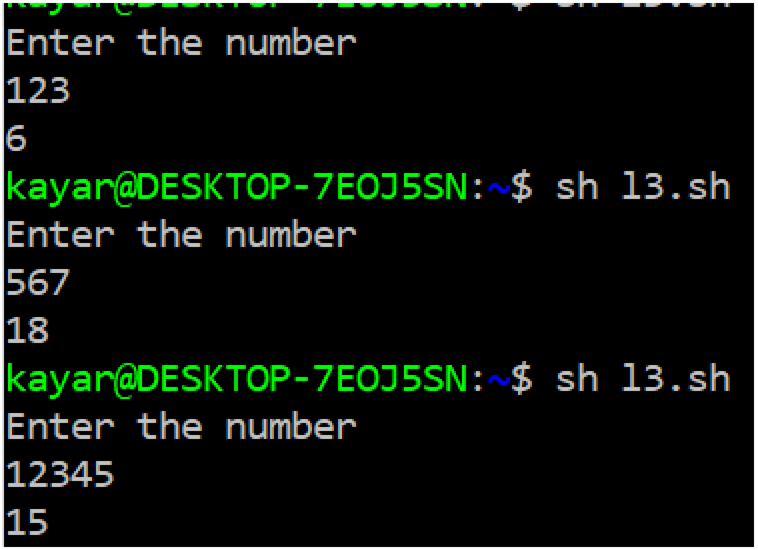


```
#!/bin/sh
echo "Enter the number"
read num
fact=1
while [ $num -gt 1 ]
do
fact=`expr $fact \* $num`
num=`expr $num - 1`
done
echo $fact
```

Sum of the digits



```
#!/bin/sh
echo "Enter the number"
read num
sum=0
while [ $num -gt 0 ]
do
  r=$(( $num % 10 ))
  num=$(($num / 10 ))
  sum=$(($sum + $r))
done
echo $sum
```



```
Enter the number
123
6
kayar@DESKTOP-7EOJ5SN:~$ sh 13.sh
Enter the number
567
18
kayar@DESKTOP-7EOJ5SN:~$ sh 13.sh
Enter the number
12345
15
```