# MC214 LAB-1 REPORT

**Nisarg Suthar**

**202003030**

[31 Aug, 2021]

---

**EXERCISE 1:** WRITE A SHELL SCRIPT sum.sh THAT TAKES AN UNSPECIFIED NUMBER OF COMMAND LINE ARGUMENTS (UP TO 9) OF INTS AND FINDS THEIR SUM.

## SHELL SCRIPT:

```
#!/bin/bash

# Ex1 : Shell script to sum all numbers given as command line
arguments

if [ $# -gt 9 ]       # Warning if more than 9 arguments are passed
then

     echo -e "\n Warning : Atmost 9 arguments expected. \n"      #
printing warning

     exit  # terminating shell script

else

     echo -e "\n $# Numbers entered."      # printing the number of
arguments

fi

let sum=0       # initialising sum variable to store the sum of the
numbers

for i in $@           # loop to add all the numbers
do

     sum=`expr $sum + $i`        # adding the numbers to sum
variable

done
```

```
echo -e "\n The sum of numbers is : $sum \n"     # printing the sum

# end of script
```

**INPUT** (As Command Line Argument)**:**

```
13 4 7 2 6
```

**OUTPUT:**

```
5 numbers entered.
The sum of numbers is : 32
```

**EXPLANANTION:**

Here the numbers are taken as command line arguments. Then we initialize a variable **`sum`** with value 0. The we iterate over the arguments and add the value to the sum variable. Finally, we print the sum to the terminal. If there are no arguments given the sum remains 0 and if the number of arguments is greater than 9 then, we give a warning message.

**EXERCISE 2:** MODIFY THE TASK 1 CODE TO ADD A NUMBER TO THE SUM ONLY IF THE NUMBER IS GREATER THAN 10.

**SHELL SCRIPT:**

```
#!/bin/bash

# Ex2 : Shell script to sum all numbers given as command line
arguments if they are greater than 10

if [ $# -gt 9 ]       # Warning if more than 9 arguments are passed
then

     echo -e "\n Warning : Atmost 9 arguments expected. \n"      #
printing warning

     exit  # terminating shell script

else

     echo -e "\n $# Numbers entered."      # printing the number of
arguments

fi

let sum=0       # initialising sum variable to store the sum of the
numbers

for i in $@           # loop to add all the numbers
do

     if(($i > 10))
     then

           sum=`expr $sum + $i`        # adding the numbers to sum
variable
     fi

done

echo -e "\n The sum of numbers greater than 10 is : $sum \n"#
printing the sum

# end of script
```

**INPUT** (As Command Line Argument)**:**

```
11 2 27 3 9 13
```

**OUTPUT:**

```
6 Numbers entered.
The sum of numbers greater than 10 are : 51
```

**EXPLANANTION:**

Same as the previous question. However, here we add only when the number is greater than 10 by checking each number using **`if(($i > 10)).`**

**EXERCISE 3:** DISPLAY THE NAME OF FOLDERS WHICH CONTAIN MORE THAN 2 FILES.

**SHELL SCRIPT:**

```
#!/bin/bash

# Ex3 : Shell script to display the names of folders containing more
than two files.

# creating a sample directory for testing

mkdir -p "sample"

cd "sample"                # making test files and directories in
the sample directory

mkdir -p "a"
mkdir -p "aa"
mkdir -p "ab"
mkdir -p "b"
mkdir -p "bb"
mkdir -p "ba"

touch "a/f1"; touch "a/f2"
touch "aa/f1"
touch "ab/f1"; touch "ab/f2"; touch "ab/f3"
touch "b/f1"
touch "bb/f1"; touch "bb/f2"
touch "ba/f1"; touch "ba/f2"; mkdir -p "ba/f3"

cd ..           # back to parent directory

echo -e "\nThe created sample directories and files:\n"

ls -R sample          # printing all the directories and contained
files


# finding the directories having more than 2 files

echo -e "\nThe directories having more than 2 files/directories are
:"
```

```
folders=$(ls sample)       # storing all the directories in folders
variable

for i in $folders               # iterating over all the
directories
do

     if(($(ls "sample/$i"|wc -l) > 2))     # checking if directory
has more than 2 files/directories
     then

          echo $i               # printing the directory having
more than 2 files

     fi              # end of if

done            # end of loop

echo            # printing an empty line for aesthetic purpose

# end of script
```

## INPUT:

```
 Nil
```

## OUTPUT:

The created sample directories and files:

sample:
a aa ab b ba bb

sample/a:
f1 f2

sample/aa:
f1

sample/ab:
f1 f2 f3

sample/b:
f1

sample/ba:
f1  f2   f3

sample/ba/f3:

sample/bb:
f1  f2

The directories having more than 2 files/directories are :
ab
ba

## EXPLANANTION:

Here we have first created a **sample** directory containing some test directories with a few files. We display them on terminal using the **ls -R sample** command. Then we have to find all the directories containing more than 2 files or directories.
First, we store all the directories in the sample folder in the **folders** variable. Then we iterate over them and find the number of files or directories in them using command:

**if(($(ls "sample/$i"|wc -l) > 2))**.

Here, **ls "sample/$i"** gives the list of files and directories in the **$i** directory. Then we send this output to **wc -l** command using pipe (**|**) which then gives us the number of the files and directories. Finally, if the number is greater than 2 then, we print the name of the directory (**$i)** on the terminal.

**EXERCISE 4:** WRITE A SHELL SCRIPT WHICH CAN DELETE FOLDERS NAMES STARTING WITH “A” AND CONTAINS EXACTLY 1 FILE ONLY.

**SHELL SCRIPT:**

```
#!/bin/bash

# Ex4 : Shell script to remove the folders whose name starts with
'a' and contain only one file.

# creating a sample directory for testing

mkdir -p "sample"

cd "sample"                # making test files and directories in
the sample directory

mkdir -p "a"
mkdir -p "aa"
mkdir -p "ab"
mkdir -p "b"
mkdir -p "bb"
mkdir -p "ba"

touch "a/f1"; touch "a/f2"
touch "aa/f1"
touch "ab/f1"; touch "ab/f2"; touch "ab/f3"
touch "b/f1"
touch "bb/f1"; touch "bb/f2"
touch "ba/f1"; touch "ba/f2"; touch "ba/f3"

cd ..           # back to parent directory

echo -e "\nCreated sample directories and files:\n"

ls -R sample          # printing all the directories and contained
files


# finding the directories having only 1 file and whose name starts
with 'a'

echo -e "\nDeleting directories whose name starts with 'a' having
only 1 file : \n"
```

```
folders=$(ls sample|grep "^a")        # storing the directories in
folders variable

for i in $folders                # iterating over all the
directories in folders variable
do

     if(($(ls "sample/$i"|wc -l) == 1))    # checking if directory
has only 1 file
     then

          rm -r "sample/$i"          # removing the directory
          echo "Directory $i has been removed."

     fi              # end of if

done            # end of loop

echo -e -n "\nRemaining directories : "
ls sample  # printing the directories of sample for confirmation

echo       # printing an empty line for aesthetic purpose

# end of script
```

## INPUT:

```
 Nil
```

## OUTPUT:

```
Created sample directories and files:

sample:
a  aa  ab  b  ba  bb

sample/a:
f1  f2

sample/aa:
f1

sample/ab:
f1  f2     f3
```

```
sample/b:
f1

sample/ba:
f1  f2    f3

sample/bb:
f1  f2

Deleting directories whose name starts with 'a' having only 1 file :

Directory aa has been removed.

Remaining directories : a  ab  b  ba  bb
```

## EXPLANANTION:

Here also we create a **sample** directory. Now since we want the directories whose names start with 'a', we filter out the result of **ls sample** using grep:

```
folders=$(ls sample|grep "^a")
```

Here we pipe in the result to **grep "^a"** command. The use of **^** in **grep "^a"** means that it will find out only those names which start with 'a'. So, the variable **folders** contains only those directories of sample whose name starts with 'a'. Now we iterate over **folders** and check for the number of files in them. If there's only 1 file then we delete that directory.
At last, we display the remaining directories in the sample directory.

**EXERCISE 5:** WRITE A PROGRAM WHICH WILL DISPLAY FILE/DIRECTORY NAME WITH THE STATUS THAT IT'S A FILE OR DIRECTORY.

**SHELL SCRIPT:**

```
#!bin/bash

# Ex5 : Shell script to display file and directory name with its
status

# creating a sample folder containin files and directories

mkdir -p sample

cd sample       # creating test files and directories

mkdir -p Jo
touch baka
mkdir -p Takleef
touch to
touch rahevani

echo -e -n "\n The test objects created are:\n "
ls              # listing down the created files
echo

for i in $(ls)        # iterating over the files
do

     if [ -f $i ]              # checking if it is a file
     then

          echo " File      : $i"

     elif [ -d $i ]            # checking if it is a directory
     then

          echo " Directory : $i"

     else                      # exceptional case

          echo " Warning : Unknown file type."

     fi         # end of if
```

```
done       # end of loop

cd ..      # returning to parent directory.

echo       # end of script
```

## INPUT:

```
Nil
```

## OUTPUT:

```
The test objects created are:
baka  Jo  rahevani  Takleef  to

File      : baka
Directory : Jo
File      : rahevani
Directory : Takleef
File      : to
```

## EXPLANANTION:

Here also we create a **sample** directory. Then we iterate over the contents of sample and check whether it is a file or a directory using the commands:

**if [ -f $i ] & if [ -d $i ]**

Here -f checks for files and -d checks for directories. Also if it is an unknown file type then we issue a warning.

**EXERCISE:** NOW FIND THE STRING WHICH ENDS WITH THE EITHER A OR B AND ATTACH YOUR EXECUTION ON REPORT.

**SHELL SCRIPT:**

```
#!/bin/bash

# Ex : Shell script to display the strings which end with a or b.

# creating a sample file for testing

cat > test.txt << EOF      # making a new file with cat command
aaaa
bbbb
abcd
abab
xyzb
airbnb
adastra
elon
jeff
mukesh
ratan
mark
larry
sergey
sundar
satya
EOF
# EOF marks the end of the file

echo -e "\nHere is a sample test file :\n"       # displaying the
file on the terminal
cat test.txt

echo -e "\nThe strings ending with a or b are :\n"
grep "[ab]$" test.txt            # using grep to filter out strings
ending with 'a' or 'b'

echo

# end of script
```

## INPUT:

```
Nil
```

## OUTPUT:

```
Here is a sample test file :

aaaa
bbbb
abcd
abab
xyzb
airbnb
adastra
elon
jeff
mukesh
ratan
mark
larry
sergey
sundar
satya

The strings ending with a or b are :

aaaa
bbbb
abab
xyzb
airbnb
adastra
satya
```

## EXPLANANTION:

Here I have made a **test.txt** file using the indirection operators **>** & **<<**. **EOF** marks the end of the file. Then to display the strings ending with 'a' or 'b' I have used the command:

**grep "[ab]$" test.txt**

Here using **[ab]$** finds only those strings which end with either 'a' or 'b'.

# EXPLORATION OF EXAMPLE SCRIPTS

## SCRIPT:1

```
#!/bin/bash

function f1 {
echo Hello from $FUNCNAME!
VAR="123"
}
f2() {
p1=$1
p2=$2
sum=$((${p1} + ${p2}))
echo "${sum}"
}
f1
echo ${VAR}
mySum="$(f2 1 2)"
echo mySum = $mySum

mySum="$(f2 10 -2)"
echo mySum = $mySum
```

## INPUT:

```
Nil
```

## OUTPUT:

```
Hello from f1!
123
mySum = 3
mySum = 8
```

## EXPLANATION:

Here we have defined two functions `f1` and `f2` by using the function command. f1 expects no arguments while we can provide arguments to f2.
f1 displays "`Hello from f1!`". Here `$FUNCNAME` gives the function name. It then initializes the global variable `VAR` to 123.
f2 stores two arguments in variables `p1` and `p2`. It then computes their sum and stores it in `sum` variable. Then it returns the value of sum.

By calling f1 we get the “Hello from f1!” statement and VAR is initialized to 123.
We print value of VAR.
We call f2 with arguments 1 and 2 and store the result in **`mySum`** variable. Then we print **`mySum`** variable with its value.
Again, we call f2 with arguments 10 and -2 and store the result in mySum variable. Then again, we print mySum variable with its value.

## SCRIPT:2

```
#!/bin/bash

# test that at least one argument was passed
if [[ $# -le 0 ]]
then
printf "Not enough arguments!\n"
exit
fi

count=1

for arg in "$@"
do
if [[ $arg =~ ^-?[0-9]+([0-9]+)?$ ]]
then
n[$count]=${arg}
let "count += 1"
else
echo "$arg is not a valid integer!"
fi
done

sort -n <(printf "%s\n" "${n[@]}")
```

## INPUT (As command line arguments):

```
2 6 3 1 9 0 n
```

## OUTPUT:

```
n is not a valid integer!
0
1
2
3
6
9
```

## EXPLANATION:

This function sorts the numbers passed as command line arguments.

First the script check for non-zero number of arguments. Otherwise, the script terminates.
For non-zero arguments we first initialize a counter variable **`count`** with value 1. Then we check for each argument whether it is an integer or not and if it is an integer then we store the variable in an array n, at index count by incrementing it successively. If the argument is not an integer, then we issue a warning.
Finally, we print the array by sorting it in increasing order.

**SCRIPT:3**

```
#!/bin/bash

if [ -z "$1" ]; then
echo "Usage: $0 filename."
exit 1
fi

filename=$1

while read -n 1 c
do
echo "$c"
done < "$filename" | grep '[[:alpha:]]' | sort | uniq -c | sort -nr
```

**INPUT** (As text file)**:**

```
Hello.
This is MC 212 OS Course.
```

**OUTPUT:**

```
     3 s
     2 o
     2 l
     2 i
     2 e
     2 C
     1 u
     1 T
     1 S
     1 r
     1 O
     1 M
     1 H
     1 h
```

**EXPLANATION:**

This script displays the character count in descending order.
It takes a text file as CMD line input. If the file is empty script is terminated.

If the file is non-empty it is piped to the **`grep '[[:alpha:]]'`** command which filters out only alphabets in the file. Then it is sorted. Then the **`uniq -c`** command finds only the unique alphabets and appends the frequency in front of it. Then finally it is sorted in reverse order of frequency by **`sort -nr`**.
This output is redirected to the while loop which reads each character one-by-one using **`read -n 1 c`** command and prints it on the terminal screen. The while loop gets terminated when no character remains to be read.

**SCRIPT:4**

```
#!/bin/bash

# Print default output
echo `date`

# Print current date without the time
echo `date +"%m-%d-%y"`

# Use 4 digits for year
echo `date +"%m-%d-%Y"`

# Display time only
echo `date +"%T"`

# Display 12 hour time
echo `date +"%r"`

# Time without seconds
echo `date +"%H:%M"`

# Print full date
echo `date +"%A %d %b %Y %H:%M:%S"`

# Nanoseconds

echo Nanoseconds: `date +"%s-%N"`
# Different timezone by name
echo Timezone: `TZ=":US/Eastern" date +"%T"`
echo Timezone: `TZ=":Europe/UK" date +"%T"`

# Print epoch time - convenient for filenames
echo `date +"%s"`
# Print week number
echo Week number: `date +"%V"`

# Create unique filename
f=`date +"%s"`
touch $f
ls -l $f
rm $f

# Add epoch time to existing file
f="/tmp/test"
```

```
touch $f
mv $f $f.`date +"%s"`
ls -l "$f".*
rm "$f".*
```

**INPUT** (As command line arguments)**:**

```
 Nil
```

## OUTPUT:

```
Friday 03 September 2021 08:00:01 PM IST
09-03-21
09-03-2021
20:00:01
08:00:01 PM IST
20:00
Friday 03 Sep 2021 20:00:01
Nanoseconds: 1630679401-044918775
Timezone: 10:30:01
Timezone: 14:30:01
1630679401
Week number: 35
-rw-rw-r-- 1 adastra adastra 0 Sep  3 20:00 1630679401
-rw-rw-r-- 1 adastra adastra 0 Sep  3 20:00 /tmp/test.1630679401
```

## EXPLANATION:

This script displays date in variety of formats.
`date` gives us the day, date and time in default format.
`%m` is used to get month, `%d` for day, `%y` for year, `%Y` for full year, `%T` for time in 24-hour format, `%r` for time in 12-hour format, `%H` for hours, `%M` for minutes, `%S` for seconds, **`%N`** for nanoseconds, `%A` for week day and **`%V`** for week number. These are used as options for **`date`** command. `%s` gives the time since epoch.
We can display time in different time zones by using TZ option.
Next, we use the epoch time to give unique names to files. Either we can use the epoch time as a name of the given file (**`f=`date +"%s"`**) or we can append the epoch time to existing file name by renaming it (**`mv $f $f.`date +"%s"`).**