

CSI_5_OSY - Coursework 2 (Shell Programming)

Deadline: 3rd of December 2020, 11:59 pm.

Kamran Bhatti 3807942

Contents

Introduction.....	3
Task 1: GCD Calculator	3
Task 2: CLI Calculator	5
Task 3: Number Converter.....	10
Task 4: File-Length Comparator	11
Task 5: Network Monitor	13
Bibliography	17

Figure 1.....	3
Figure 2.....	3
Figure 3.....	4
Figure 4.....	4
Figure 5.....	5
Figure 6.....	6
Figure 7.....	6
Figure 8.....	7
Figure 9.....	7
Figure 10.....	8
Figure 11.....	8
Figure 12.....	9
Figure 13.....	9
Figure 14.....	10
Figure 15.....	10
Figure 16.....	10
Figure 17.....	10
Figure 18.....	11
Figure 19.....	12
Figure 20.....	12
Figure 21.....	12
Figure 22.....	12
Figure 23.....	12
Figure 24.....	12
Figure 25.....	13
Figure 26.....	14
Figure 27.....	14
Figure 28.....	14
Figure 29.....	15
Figure 30.....	15
Figure 31.....	15
Figure 32.....	16
Figure 33.....	16

Introduction

This is the second coursework assignment for operating systems. All code can be found at (Bhatti, 2020).

Task 1: GCD Calculator

Write a shell script that calculates GCD (greatest common divisor) of two numbers.

- You may take the input numbers from either command line or using standard inputs.
- The program must check if the inputs are numbers.

```
if echo $1 | egrep -q '^-*[0-9]+$' && echo $2 | egrep -q '^-*[0-9]+$'; then
# test inputted values against regex for numbers in terminal window

if [ $1 -eq 0 -o $2 -eq 0 ]; then
    echo "The inputs must be non-zero values"
    # if one of the inputs is 0 the gcd is 0
else
    if [ $1 -gt $2 ]; then
        max=$1
    else
        max=$2
    fi
    # store the largest of the two inputs in the max variable

    for i in $(seq 1 1 $max); do
        if [ `expr $1 % $i` -eq 0 -a `expr $2 % $i` -eq 0 ]; then
            gcd=$i
        fi
    done
    # store the largest common divisor in the gcd variable if both numbers are divisible by it

    echo "$gcd"
    # output gcd value to terminal window
fi

else
    echo "Error: either the first or second value entered isnt a number"
    # if any of the inputted values are numbers, error message is displayed
fi
```

Figure 1

\$1 and \$2 are being used as the inputs for the program are passed to the program as the program is called. Line 1 uses (Rosenfield, 2008) to compare the inputted values with a regular expression representation of numerical values. The if statement with ^ and \$ values act as boundaries to dictate that the comparison should be carried out from the first character in the value to the last character in the value passed as input to the program. The - value allows for negative values to work in the program however, the program won't work if both values are negative. The + dictates that there are one or more numbers to compare for each value entered where each character in the inputted values would be compared against digits in a range of 0 to 9. If any of the values entered don't contain numbers, an error message as displayed in Figure 2 is displayed.

```
user1@tut1:~/cw2Work$ sh task1.sh a 10
Error: either the first or second value entered isnt a number
```

Figure 2

The second if statement checks if any of the inputted values are equal to 0, if they are the message in Figure 3 is displayed.

```
user1@tut1:~/cw2Work$ sh task1.sh 10 0  
The inputs must be non-zero values
```

Figure 3

The third if statement checks which of the inputted values is largest and stores them in a variable called max. The for loop iterates from 1 to the largest number entered where a variable named gcd is updated if both inputted values are divisible by the iterator i. After finding the greatest common divisor, the value stored in gcd is outputted to the terminal window as displayed in Figure 4.

```
user1@tut1:~/cw2Work$ sh task1.sh 70 80  
10
```

Figure 4

Task 2: CLI Calculator

Write a shell script to make a CLI based calculator, the program should behave as follows:

- The program must start with a clear screen, prompting to input two numbers.
- Once given, it must list 5 operations (Add, Sub, Mul, Div, and Mod).
- Based on the user's choice, it should perform addition, subtraction, multiplication, division, and modulus (manage any illegal operation e.g. 0/0).
- Once calculated, the result must be printed on a new line and the program must print "Press any key to continue" on the next line.
- Pressing any key, the program must go back to the first step.

```
readOne () {
    local oldstty
    oldstty=$(stty -g)
    stty -icanon -echo min 1 time 0
    dd bs=1 count=1 2>/dev/null
    stty "$oldstty"
}

while : ; do
    command clear
    # clears the terminal screen

    echo "Input the first number> \c"
    read num1
    echo "\nInput the second number> \c"
    read num2
    # read number input from user

    if echo $num1 | egrep -q '^[0-9]+$' && echo $num2 | egrep -q '^[0-9]+$'; then

        echo "\nWould you like to: \n-Add (Add)\n-Subtract (Sub)\n-Multiply (Mul)\n-Divide (Div)\n-Modulus (Mod)\n> \c"
        read operation
        # read operation to perform

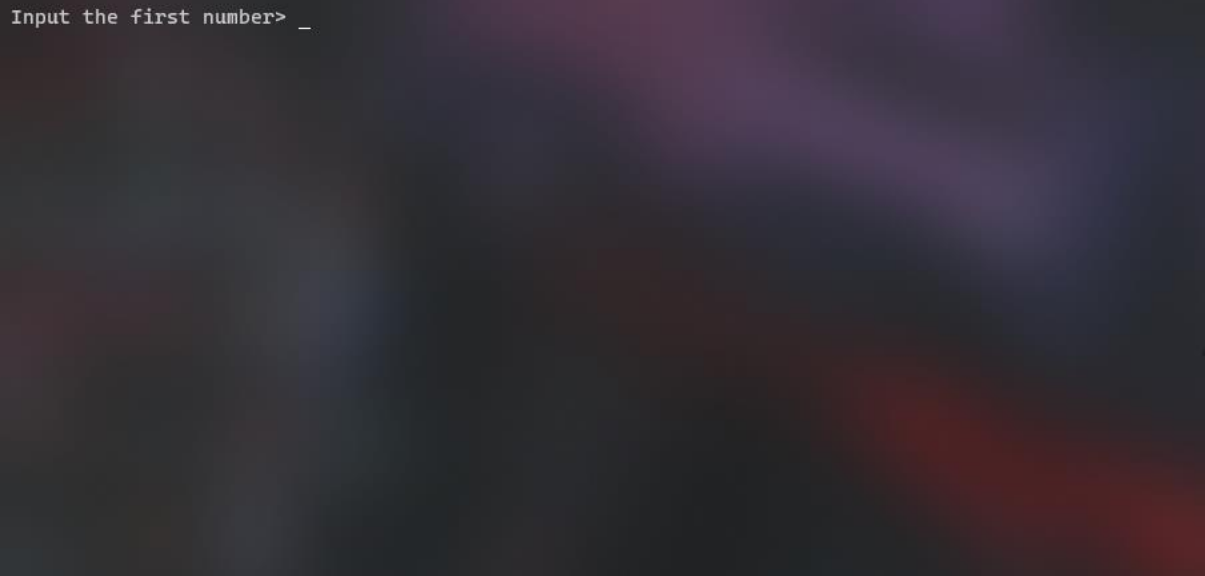
        case $operation in
            "Add")
                echo "\n$num1 + $num2 = `expr $num1 + $num2`";;
            "Sub")
                echo "\n$num1 - $num2 = `expr $num1 - $num2`";;
            "Mul")
                echo "\n$num1 x $num2 = `expr $num1 \* $num2`";;
            "Div")
                echo "\n$num1 / $num2 = `expr $num1 / $num2`";;
            "Mod")
                echo "\n$num1 % $num2 = `expr $num1 % $num2`";;
            *)
                echo "\nOperation not supported";;
        esac

    else
        echo "Error: either the first or second inputted value isn't a number"
    fi

    echo "Press any key to continue or Ctrl+C to exit \c"
    key=$(readOne)
done
```

Figure 5

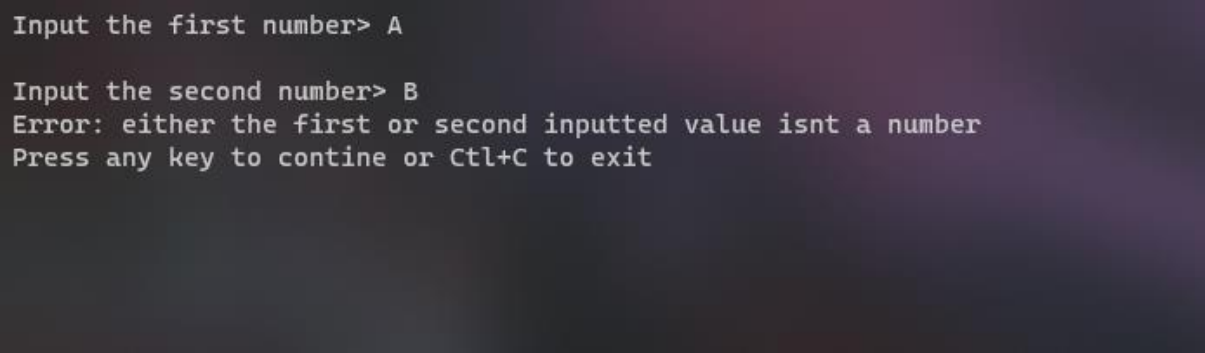
(notice., 2009) was used to create the readOne function where the input from a user is taken even if the user doesn't submit the input via the enter key, for instance, a user could press the "m" key and the program would receive this input even though the enter key wasn't pressed after. The while loop uses : ; to create an infinite loop that has no termination condition. The line stating command clear clears the terminal window as shown in Figure 6.



```
Input the first number> _
```

Figure 6

The program then reads the users input. As shown in Figure 7, (Rosenfield, 2008) has been used to check the users' input against a regular expression representation of numeric values similar to Figure 1.



```
Input the first number> A
Input the second number> B
Error: either the first or second inputted value isnt a number
Press any key to contine or Ctl+C to exit
```

Figure 7

The operation is then read where a case statement, as referenced in (Linuxize.com, 2019), to compare the inputted value against an operation. The default operation, represented by *), will execute if the operation entered doesn't match any of the case statements as shown in Figure 8.

```
Input the first number> 1
Input the second number> 2
Would you like to:
-Add (Add)
-Subtract (Sub)
-Multiply (Mul)
-Divide (Div)
-Modulus (Mod)
> a
Operation not supported
Press any key to continue or Ctrl+C to exit
```

Figure 8

To carry out an operation the “expr” keyword is used to evaluate the expression passed to it. To perform the multiplication operation, the * character must be escaped as the character is treated as a wild card in Unix as shown in Figure 9.

```
Input the first number> 40
Input the second number> 50
Would you like to:
-Add (Add)
-Subtract (Sub)
-Multiply (Mul)
-Divide (Div)
-Modulus (Mod)
> Mul
40 x 50 = 2000
Press any key to continue or Ctrl+C to exit
```

Figure 9

The other outputs for the program are depicted in Figure 10, Figure 11, Figure 12 and Figure 13.

```
Input the first number> 1

Input the second number> 2

Would you like to:
-Add (Add)
-Subtract (Sub)
-Multiply (Mul)
-Divide (Div)
-Modulus (Mod)
> Add

1 + 2 = 3
Press any key to contine or Ctl+C to exit
```

Figure 10

```
Input the first number> 1

Input the second number> 2

Would you like to:
-Add (Add)
-Subtract (Sub)
-Multiply (Mul)
-Divide (Div)
-Modulus (Mod)
> Sub

1 - 2 = -1
Press any key to contine or Ctl+C to exit _
```

Figure 11


```
Input the first number> 10  
Input the second number> 5  
Would you like to:  
-Add (Add)  
-Subtract (Sub)  
-Multiply (Mul)  
-Divide (Div)  
-Modulus (Mod)  
> Div  
  
10 / 5 = 2  
Press any key to continue or Ctrl+C to exit
```

Figure 12

```
Input the first number> 10  
Input the second number> 5  
Would you like to:  
-Add (Add)  
-Subtract (Sub)  
-Multiply (Mul)  
-Divide (Div)  
-Modulus (Mod)  
> Mod  
  
10 % 5 = 0  
Press any key to continue or Ctrl+C to exit
```

Figure 13

Task 3: Number Converter

Write a shell script that takes a decimal number as input and prints its binary, octal, and hexadecimal equivalents.

- Check if the input is a number.
- You may use any inbuilt unix tool to perform the conversion.

```
if echo $1 | egrep -q '^-*[0-9]+$'; then
    echo "Binary \c"
    echo "obase=2; $1 " | bc
    echo "Octal \c"
    echo "obase=8; $1 " | bc
    echo "Hexadecimal \c"
    echo "obase=16; $1 " | bc
else
    echo "Value entered isnt a whole decimal number"
fi
```

Figure 14

Similar to Figure 1, (Rosenfield, 2008) has been used to ensure that the value entered when the program is executed is a whole number as shown in Figure 15.

```
user1@tut1:~/cw2Work/BashScripts$ sh task3.sh a
Value entered isnt a whole decimal number
```

Figure 15

As seen in (Anne, 2012), the obase command is used to specify the output base of the value you enter. An ibase or input base isn't specified as Unix defaults to using decimal as its input base. The pipe command is used to pass the output base and value inputted by the user to the bc utility which is a precision calculator in Unix. The output of the program is shown in Figure 16 and Figure 17.

```
user1@tut1:~/cw2Work/BashScripts$ sh task3.sh -8
Binary -1000
Octal -10
Hexadecimal -8
```

Figure 16

```
user1@tut1:~/cw2Work/BashScripts$ sh task3.sh 24
Binary 11000
Octal 30
Hexadecimal 18
```

Figure 17

Task 4: File-Length Comparator

Write a script that takes two filenames and returns the one having more lines than the other.

- Filenames must be given with their relative paths.
- In case of a tie, the script must return both the filenames.
- In case the files are empty or do not exist, the program must print a custom error statement.

```
file1=$1
file2=$2

length1=0
length2=0

if [ -e $file1 -a -e $file2 ]; then
    while read line; do
        length1=$((length1+1))
    done < $file1

    while read line; do
        length2=$((length2+1))
    done < $file2

    if [ $length1 != 0 -a $length2 != 0 ]; then
        if [ $length1 -gt $length2 ]; then
            echo "$file1 has more lines"
        elif [ $length1 -eq $length2 ]; then
            echo "$file1 has the same amount of lines as $file2"
        else
            echo "$file2 has more lines"
        fi
    elif [ $length1 -eq 0 -a $length2 != 0 ]; then
        echo "$file1 is empty"
    elif [ $length2 -eq 0 -a $length1 != 0 ]; then
        echo "$file2 is empty"
    else
        echo "Both $file1 and $file2 are empty"
    fi
elif [ -e $file1 -a ! -e $file2 ]; then
    echo "$file2 doesnt exist"
elif [ -e $file2 -a ! -e $file1 ]; then
    echo "$file1 doesnt exist"
else
    echo "Both $file1 and $file2 dont exist"
fi
```

Figure 18

The file paths are given to the program from the command line while the program is being called to execute. The files are assigned to separate variables where (Gite, 2020) is used to check if a file exists by using the “-e” operator next to a file variable. If one of the files exists, an error message is displayed dictating that one of the files don’t exist as shown in Figure 19. If both files don’t exist then the program outputs an error message stating that both files don’t exist as shown in Figure 20. If both files exist the program executes normally.

```
user1@tut1:~/cw2Work/BashScripts$ sh task4.sh emp1.txt task2.sh  
emp1.txt doesnt exist
```

Figure 19

```
user1@tut1:~/cw2Work/BashScripts$ sh task4.sh emp1.txt emp2.txt  
Both emp1.txt and emp2.txt dont exist
```

Figure 20

After, (TecAdmin.net, 2019) is used to create a while loop which reads each line of the file passed to the program. Using methods in (vino, 2005), the length variables are incremented for each line read in a file passed to the program. If one of the files is empty the program outputs an appropriate error message as shown in Figure 21. If both files are empty the program outputs an error message as seen in Figure 22.

```
user1@tut1:~/cw2Work/BashScripts$ sh task4.sh empty1.sh task1.sh  
empty1.sh is empty
```

Figure 21

```
user1@tut1:~/cw2Work/BashScripts$ sh task4.sh empty1.sh empty2.sh  
Both empty1.sh and empty2.sh are empty
```

Figure 22

If both files have lines but one of the files has more lines than the other files, then the program outputs that the file with more lines has more lines as shown in Figure 23.

```
user1@tut1:~/cw2Work/BashScripts$ sh task4.sh task1.sh task2.sh  
task2.sh has more lines
```

Figure 23

If both files have the same amount of lines, the program will output that both files have an equal amount of lines as shown in Figure 24.

```
user1@tut1:~/cw2Work/BashScripts$ sh task4.sh ../../abc.sh ../../for_loop.sh  
../../abc.sh has the same amount of lines as ../../for_loop.sh
```

Figure 24

Task 5: Network Monitor

Write a shell script that creates a CSV file and records the number of packets sent and received by a given network interface ordered by the timestamp.

- The interface name and the interval must be provided by the user using a command-line argument.
- Please note, a CSV (Comma Separated Value) file is a text file that stores a table. The first line records the column names and the followig lines record individual rows. Each value seperated by comma (See the example below)

name	year	marks
abc	2	86
xyz	1	45
pqr	2	66

- the CSV formation of the given table is,

```
name,year,marks
abc,2,86
xyz,1,45
pqr,2,66
```

```
interface=$1
interval=$2

DIR="/sys/class/net"
check="$DIR/$interface"

if [ ! -z "$interface" -a ! -z "$interval" ]; then
    if [ -e $check ]; then
        if echo $interval | egrep -q '^[1-9][0-9]*$'; then
            echo "Executing on interface $interface with interval $interval"
            command touch packets.csv
            echo "Sent,Received,TimeStamp">packets.csv
            for i in $(seq 1 1 10); do
                sentLine=`ifconfig $interface | tail -3 | head -1`
                sentPackets=`echo $sentLine | cut -d ' ' -f3`
                receivedLine=`ifconfig $interface | tail -5 | head -1`
                receivedPackets=`echo $receivedLine | cut -d ' ' -f3`
                echo $sentPackets,$receivedPackets,`date +%T`>>packets.csv
                sleep $interval
            done
            echo "Finished running. Output is in packets.csv"
        else
            echo "Error: Interval must be a number"
        fi
    else
        echo "Error: Interface doesnt exist"
    fi
else
    echo "Error: Either the interface or interval havent been provided"
fi
```

Figure 25

The interface and interval are passed to the program during program execution which are stored in separate variables. Using (Gite, 2020), the ! -z arguments are used to ensure that the user has provided arguments for the interface and interval variables by checking if the values aren't empty otherwise an error message is displayed as shown in Figure 26.

```
user1@tut1:~/cw2Work/BashScripts$ sh task5.sh
Error: Either the interface or interval havent been provided
```

Figure 26

The /sys/class/net directory stores folders which are named after each interface on the system thus, by adding the interface that a user enters to the directory, a check is performed to validate if the directory for that interface exists using the -e argument as described in (Gite, 2020). If the interface doesn't exist then an error message is displayed as shown in Figure 27.

```
user1@tut1:~/cw2Work/BashScripts$ sh task5.sh enp 1
Error: Interface doesnt exist
```

Figure 27

(ghoti, 2012) was used to check that the interval entered is a number above 0 using the regular expression values where the first digit of the interval specified must be above 0 where if the validation fails an error message is displayed as shown in Figure 28.

```
user1@tut1:~/cw2Work/BashScripts$ sh task5.sh enp0s3 -1
Error: Interval must be a number
```

Figure 28

command touch packets.csv creates a CSV file in the current directory of the user to store the number of packets sent and received with their timestamp. The > operator writes the values on the left of the operator to the file on the write of the operator where echo is used to execute this command in the terminal. A for loop is used to execute the process of getting the sent and received packets and write them to the CSV file with a timestamp 10 times.

The tail and head operators are used for sentLine and receivedLine to retrieve the number of RX (received) and TX (sent) packets that would be displayed when ifconfig is run with a specified interface. The cut operation, referenced in (Ghosh, 2020), is used in sentPackets and receivedPackets with -d ' ' to remove the spaces in the lines retrieved from ifconfig which relate to the RX and TX packets. The -f3 operator gets the third value in sentLine and receivedLine which would return the numeric value for the number of packets sent and received. (Newell, 2020) is used to specify the date +%T variable which retrieves a timestamp in the form of hours: minutes: seconds. The >> operator appends the sentPackets, receivedPackets and timestamp to the packets.csv file where echo is used to execute the command in the terminal.

The sleep command in conjunction with the interval value specified by the user acts as an interval between each iteration of the for loop when retrieving the sent and received packets from ifconfig.

When the program is executed with a provided interface and interval, the program creates a packets.csv file as seen in Figure 29 and Figure 32, writes the sent and received packets to the CSV file with a timestamp as seen in Figure 33 and then outputs that the process is finished as seen in Figure 31.


```

user1@tut1:~/cw2Work/BashScripts$ ll
total 40
drwxrwxr-x 3 user1 user1 4096 Nov 26 11:53 ./
drwxrwxr-x 3 user1 user1 4096 Nov 22 12:29 ../
drwxrwxr-x 8 user1 user1 4096 Nov 25 12:45 .git/
-rw-rw-r-- 1 user1 user1 577 Nov 25 12:45 README.md
-rw-rw-r-- 1 user1 user1 854 Nov 21 21:09 task1.sh
-rw-rw-r-- 1 user1 user1 1107 Nov 22 12:14 task2.sh
-rw-rw-r-- 1 user1 user1 236 Nov 22 12:56 task3.sh
-rw-rw-r-- 1 user1 user1 839 Nov 22 14:59 task4.sh
-rw-rw-r-- 1 user1 user1 920 Nov 25 12:32 task5.sh
-rw-rw-r-- 1 user1 user1 213 Nov 22 12:35 .whitesource

```

Figure 29

```

user1@tut1:~/cw2Work/BashScripts$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.50.184 netmask 255.255.255.0 broadcast 192.168.50.255
    inet6 fe80::a00:27ff:fe4a:fb75 prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:4a:fb:75 txqueuelen 1000 (Ethernet)
    RX packets 3340 bytes 3243090 (3.2 MB)
    RX errors 0 dropped 749 overruns 0 frame 0
    TX packets 528 bytes 58411 (58.4 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 98 bytes 7660 (7.6 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 98 bytes 7660 (7.6 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

```

Figure 30

```

user1@tut1:~/cw2Work/BashScripts$ sh task5.sh enp0s3 1
Executing on interface enp0s3 with interval 1
Finished running. Output is in packets.csv

```

Figure 31

```

user1@tut1:~/cw2Work/BashScripts$ ll
total 44
drwxrwxr-x 3 user1 user1 4096 Nov 26 11:55 ./
drwxrwxr-x 3 user1 user1 4096 Nov 22 12:29 ../
drwxrwxr-x 8 user1 user1 4096 Nov 25 12:45 .git/
-rw-rw-r-- 1 user1 user1  204 Nov 26 11:55 packets.csv
-rw-rw-r-- 1 user1 user1  577 Nov 25 12:45 README.md
-rw-rw-r-- 1 user1 user1  854 Nov 21 21:09 task1.sh
-rw-rw-r-- 1 user1 user1 1107 Nov 22 12:14 task2.sh
-rw-rw-r-- 1 user1 user1  236 Nov 22 12:56 task3.sh
-rw-rw-r-- 1 user1 user1  839 Nov 22 14:59 task4.sh
-rw-rw-r-- 1 user1 user1  920 Nov 25 12:32 task5.sh
-rw-rw-r-- 1 user1 user1  213 Nov 22 12:35 .whitesource

```

Figure 32

```

user1@tut1:~/cw2Work/BashScripts$ cat packets.csv
Sent,Received,TimeStamp
559,3442,11:55:01
559,3443,11:55:02
559,3443,11:55:03
559,3445,11:55:04
559,3445,11:55:05
559,3445,11:55:06
559,3447,11:55:07
559,3447,11:55:08
559,3447,11:55:09
559,3449,11:55:10

```

Figure 33

Bibliography

Anne, S., 2012. *Convert Binary, HEX, Oct to decimal in Linux/Unix*. [Online]

Available at: <https://www.linuxnix.com/convert-binaryhex-oct-decimal-linuxunix/>

[Accessed 22 November 2020].

Bhatti, K., 2020. *ShellScripts*. [Online]

Available at: <https://github.com/k5924/ShellScripts>

[Accessed 22 November 2020].

Ghosh, S., 2020. *OSY_Tut3*. [Online]

Available at: <https://lsbu-lecturecast.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=3c6e145a-21c2-4ca8-8b9f-ac60013a2bbe&v=1&nomobileprompt=true>

[Accessed 25 November 2020].

ghoti, 2012. *Regular Expression for any number greater than 0? [closed]*. [Online]

Available at: <https://stackoverflow.com/questions/9038522/regular-expression-for-any-number-greater-than-0>

[Accessed 25 November 2020].

Gite, V., 2020. *Bash Shell Find Out If a Variable Is Empty Or Not*. [Online]

Available at: <https://www.cyberciti.biz/faq/unix-linux-bash-script-check-if-variable-is-empty/>

[Accessed 25 November 2020].

Gite, V., 2020. *Linux / UNIX: Find Out If File Exists With Conditional Expressions in a Bash Shell*.

[Online]

Available at: <https://www.cyberciti.biz/tips/find-out-if-file-exists-with-conditional-expressions.html>

[Accessed 11 November 2020].

Linuxize.com, 2019. *Bash Case Statement*. [Online]

Available at: <https://linuxize.com/post/bash-case-statement/>

[Accessed 22 November 2020].

Newell, G., 2020. *How to Display the Date and Time Using Linux Command Line*. [Online]

Available at: <https://www.lifewire.com/display-date-time-using-linux-command-line-4032698>

[Accessed 25 November 2020].

notice., P. u. f., 2009. *How to set read command to execute without having to press Enter button in shell?*. [Online]

Available at: <https://stackoverflow.com/questions/1816298/how-to-set-read-command-to-execute-without-having-to-press-enter-button-in-shell>

[Accessed 22 November 2020].

Rosenfield, A., 2008. *Check that a variable is a number in UNIX shell*. [Online]

Available at: <https://stackoverflow.com/questions/309745/check-that-a-variable-is-a-number-in-unix-shell>

[Accessed 11 November 2020].

TecAdmin.net, 2019. *Shell Script to Read File*. [Online]

Available at: <https://tecadmin.net/tutorial/bash/examples/read-file-line-by-line/>

[Accessed 22 November 2020].

vino, 2005. *increment a Variable*. [Online]

Available at: <https://www.unix.com/shell-programming-and-scripting/20128-increment-variable.html>

[Accessed 22 November 2020].