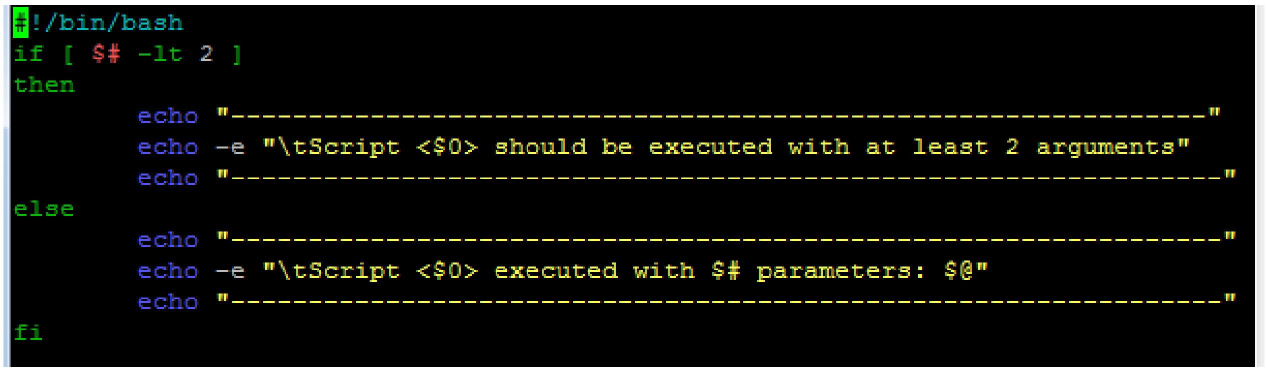
COMP1562 – Operating System
Laboratory 4

Shell Programming 1
Group ID: 21
Group Task: Task 3

Usman Basharat
Mohamed Aden
Yunus Hassan
Derek U. Mafohla

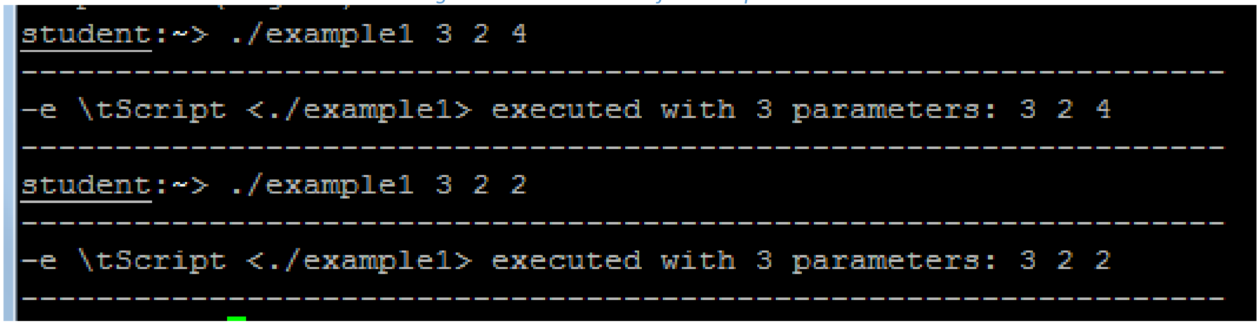
Table of Contents

Example 1



```
#!/bin/bash
if [ $# -lt 2 ]
then
        echo "----------------------------------------------------------------"
        echo -e "\tScript <$0> should be executed with at least 2 arguments"
        echo "-----------------------------------------------------------------"
else
        echo "-----------------------------------------------------------------"
        echo -e "\tScript <$0> executed with $# parameters: $@"
        echo "-----------------------------------------------------------------"
fi
```

Figure 1 shows the code for Example 1

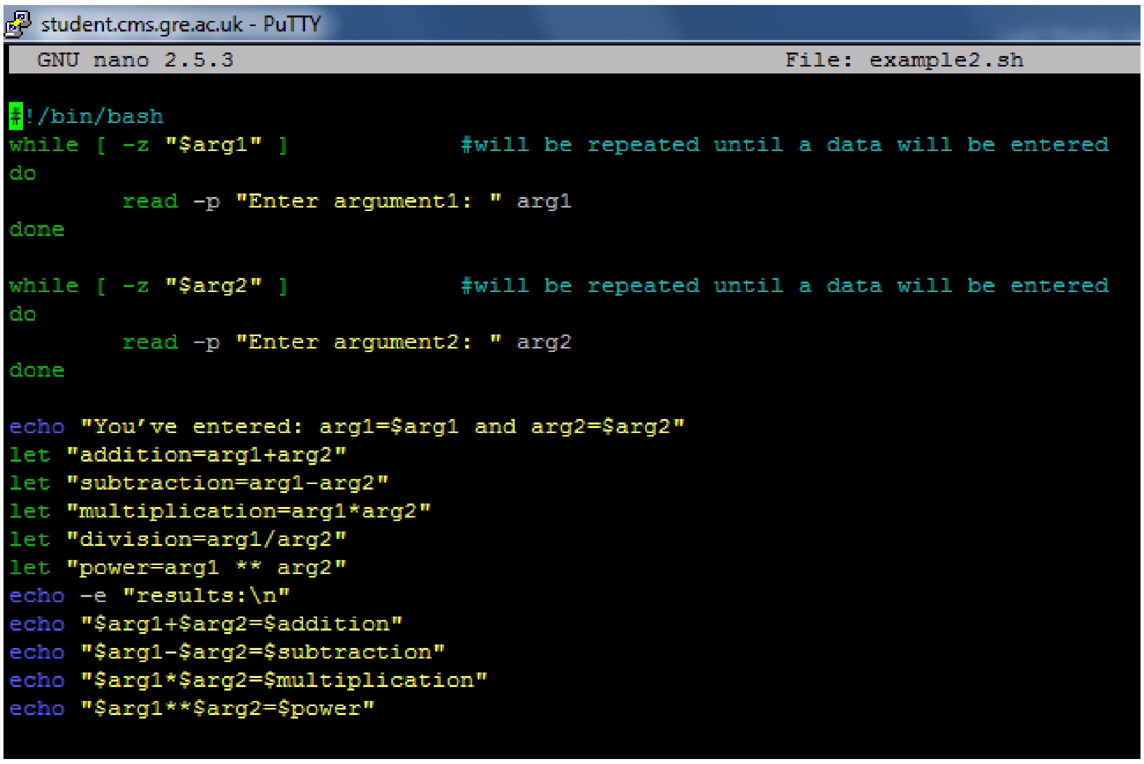


```
student:~> ./example1 3 2 4
-----------------------------------------------------------------
-e \tScript <./example1> executed with 3 parameters: 3 2 4
-----------------------------------------------------------------
student:~> ./example1 3 2 2
-----------------------------------------------------------------
-e \tScript <./example1> executed with 3 parameters: 3 2 2
-----------------------------------------------------------------
```

Figure 2 shows the code being executed

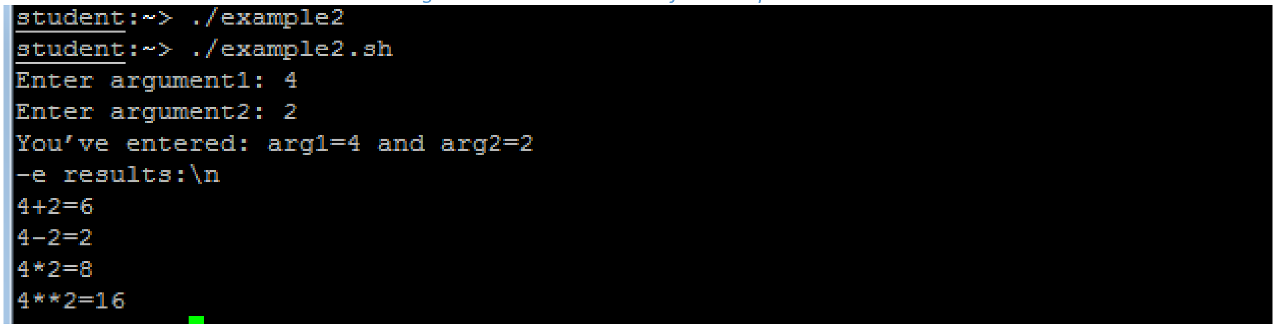
This example shows us whenever the user types in 3 arguments. Referring to Figure 2, the result recognises this and displays the return of how many has been typed in and the result of this. This screenshot shows us how the code being executed. When this was ran at the start, there was errors at the start. I had to change the quotation marks for it to go yellow.

Example 2



```
student.cms.gre.ac.uk - PuTTY
  GNU nano 2.5.3                                      File: example2.sh

#!/bin/bash
while [ -z "$arg1" ]              #will be repeated until a data will be entered
do
        read -p "Enter argument1: " arg1
done

while [ -z "$arg2" ]              #will be repeated until a data will be entered
do
        read -p "Enter argument2: " arg2
done

echo "You've entered: arg1=$arg1 and arg2=$arg2"
let "addition=arg1+arg2"
let "subtraction=arg1-arg2"
let "multiplication=arg1*arg2"
let "division=arg1/arg2"
let "power=arg1 ** arg2"
echo -e "results:\n"
echo "$arg1+$arg2=$addition"
echo "$arg1-$arg2=$subtraction"
echo "$arg1*$arg2=$multiplication"
echo "$arg1**$arg2=$power"
```

Figure 3 shows us the code for Example 2

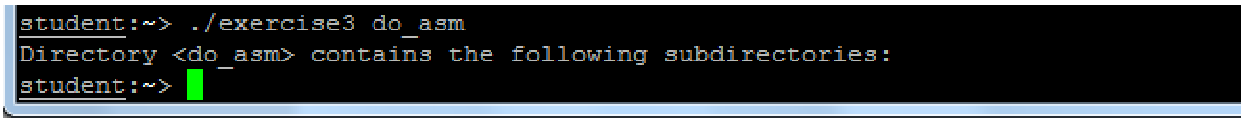


```
student:~> ./example2
student:~> ./example2.sh
Enter argument1: 4
Enter argument2: 2
You've entered: arg1=4 and arg2=2
-e results:\n
4+2=6
4-2=2
4*2=8
4**2=16
```

Figure 4 shows us the code being executed

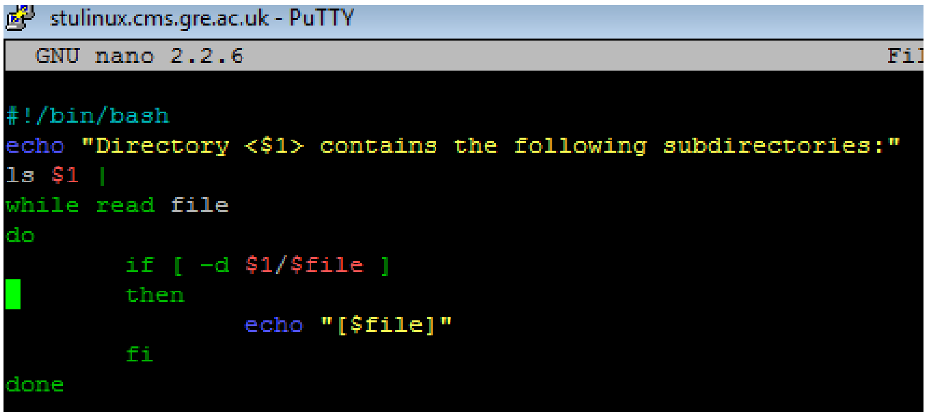
This example shows whenever we type in two numbers, it does the calculations for us in one list. As you can see in Figure 4, it clearly shows us the results of 2 arguments being entered. Figure 3 is the whole code of Figure 3 being executed. When the original code was run, there was errors for the reminder. Therefore, this was replaced with the power of two numbers. When this was ran at the start, there was errors at the start. I had to change the quotation marks for it to go yellow.

Example 3



```
student:~> ./exercise3 do_asm
Directory <do_asm> contains the following subdirectories:
student:~>
```

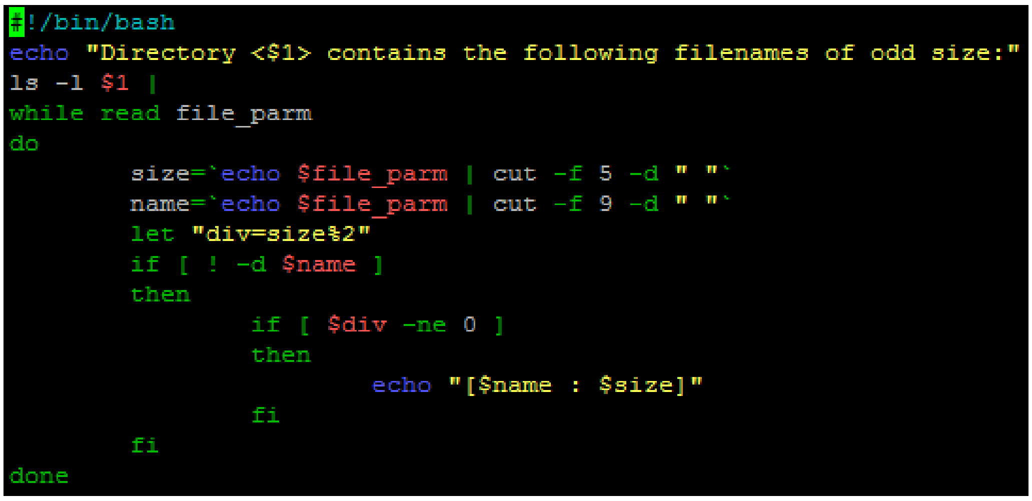
Figure 5 shows Example 3 being executed.



```
stulinux.cms.gre.ac.uk - PuTTY
  GNU nano 2.2.6                                                   Fil

#!/bin/bash
echo "Directory <$1> contains the following subdirectories:"
ls $1 |
while read file
do
        if [ -d $1/$file ]
        then
                echo "[$file]"
        fi
done
```

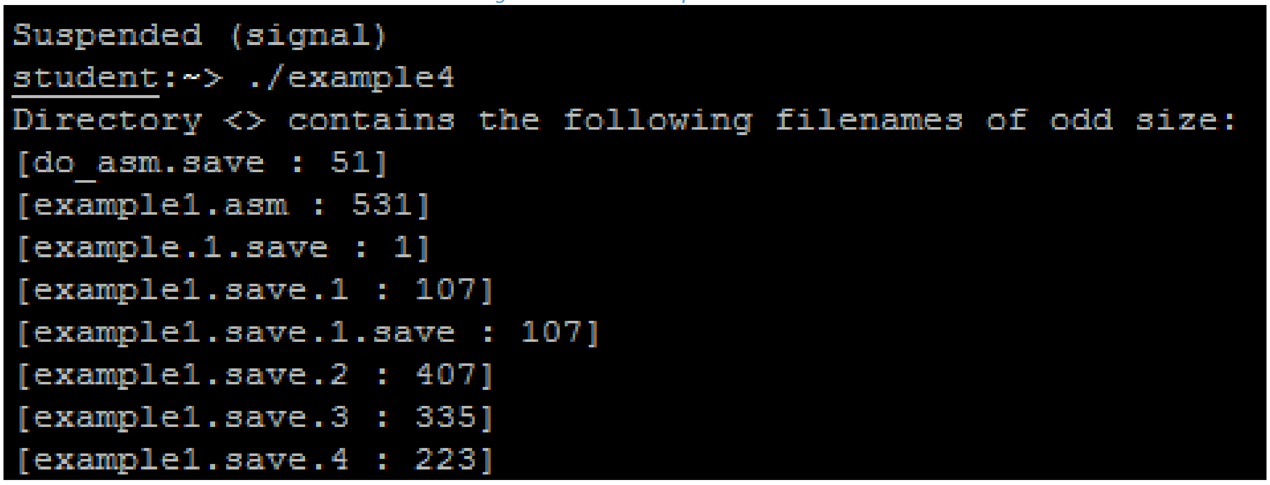
Figure 6 shows the code for Example 3

Example 4



```
#!/bin/bash
echo "Directory <$1> contains the following filenames of odd size:"
ls -l $1 |
while read file_parm
do
        size=`echo $file_parm | cut -f 5 -d " "`
        name=`echo $file_parm | cut -f 9 -d " "`
        let "div=size%2"
        if [ ! -d $name ]
        then
                if [ $div -ne 0 ]
                then
                        echo "[$name : $size]"
                fi
        fi
done
```

Figure 7 shows Example 4 code

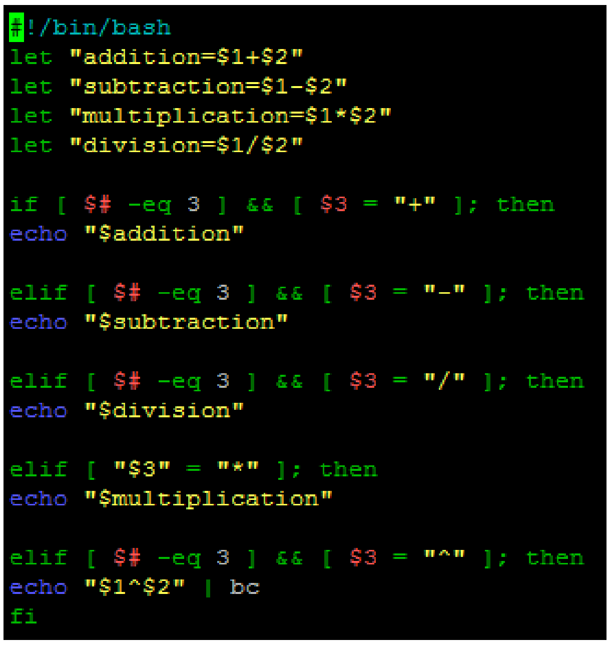


```
Suspended (signal)
student:~> ./example4
Directory <> contains the following filenames of odd size:
[do_asm.save : 51]
[example1.asm : 531]
[example.1.save : 1]
[example1.save.1 : 107]
[example1.save.1.save : 107]
[example1.save.2 : 407]
[example1.save.3 : 335]
[example1.save.4 : 223]
```

Figure 8 shows the code being executed.

Example 3 was showing us a file and its subdirectory. Example 4 showed us the list of files and its sizes. This was later implemented to be used for Task 3.3. All of the examples were great and a start ahead of the later tasks that were complete. This helped us and gave us a good idea of how to start and correct all the errors that came up at the start. Figure 8 is a list. The list is too long. Therefore, I put in the first few and showing the example being executed.

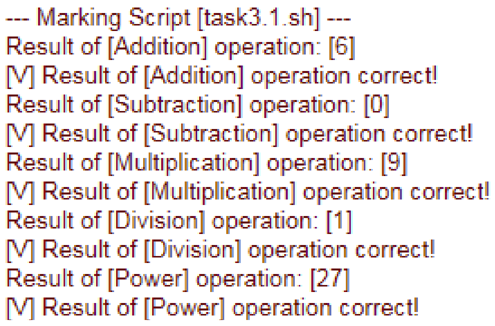
Task 3.1



```
#!/bin/bash
let "addition=$1+$2"
let "subtraction=$1-$2"
let "multiplication=$1*$2"
let "division=$1/$2"

if [ $# -eq 3 ] && [ $3 = "+" ]; then
echo "$addition"

elif [ $# -eq 3 ] && [ $3 = "-" ]; then
echo "$subtraction"

elif [ $# -eq 3 ] && [ $3 = "/" ]; then
echo "$division"

elif [ "$3" = "*" ]; then
echo "$multiplication"

elif [ $# -eq 3 ] && [ $3 = "^" ]; then
echo "$1^$2" | bc
fi
```

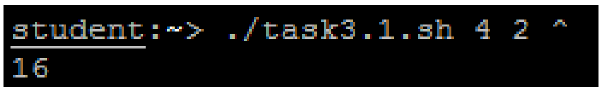
Figure 9 shows Task 3.1 code

Task 3.1 was complete by modifying Example 2. As I was doing this, I realised that I needed to put if statements in for each one to execute if it is not the right one. However, I realised that after I did the first few, the multiplication and powering the two numbers was the one that was wrong. After an intense amount of researching the reason, I realised that I needed to put test the multiplication another way to see if the code is correct. For powering, I researched a way of using the bc command. This manually by using a library does the calculation together for both of the arguments. This works and I found this way easier than the loop.



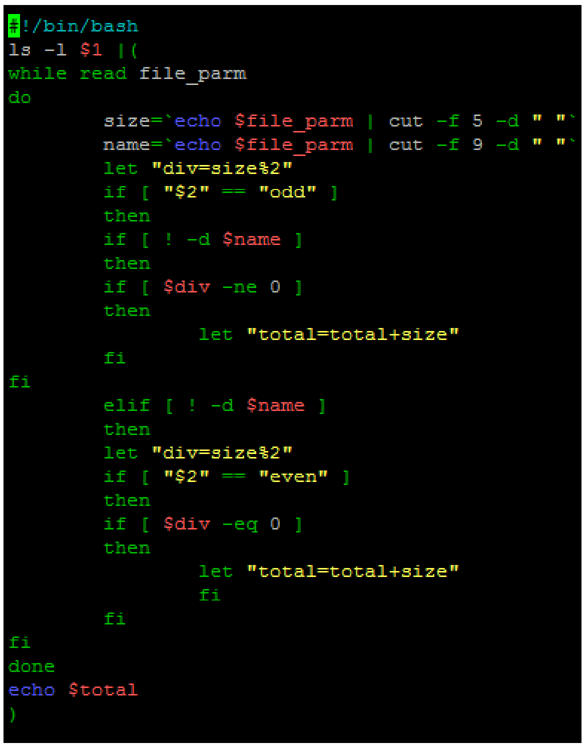
```
--- Marking Script [task3.1.sh] ---
Result of [Addition] operation: [6]
[V] Result of [Addition] operation correct!
Result of [Subtraction] operation: [0]
[V] Result of [Subtraction] operation correct!
Result of [Multiplication] operation: [9]
[V] Result of [Multiplication] operation correct!
Result of [Division] operation: [1]
[V] Result of [Division] operation correct!
Result of [Power] operation: [27]
[V] Result of [Power] operation correct!
```

Figure 10 shows that this code is correct through Script Check.



```
student:~> ./task3.1.sh 4 2 ^
16
```

Task 3.2

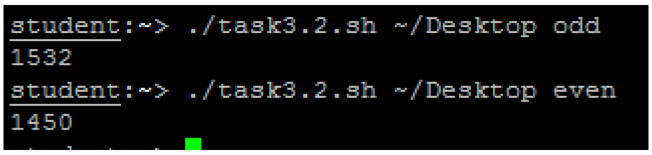


```
#!/bin/bash
ls -l $1 |(
while read file_parm
do
        size=`echo $file_parm | cut -f 5 -d " "`
        name=`echo $file_parm | cut -f 9 -d " "`
        let "div=size%2"
        if [ "$2" == "odd" ]
        then
        if [ ! -d $name ]
        then
        if [ $div -ne 0 ]
        then
                let "total=total+size"
        fi
fi
        elif [ ! -d $name ]
        then
        let "div=size%2"
        if [ "$2" == "even" ]
        then
        if [ $div -eq 0 ]
        then
                let "total=total+size"
                fi
        fi
fi
done
echo $total
)
```

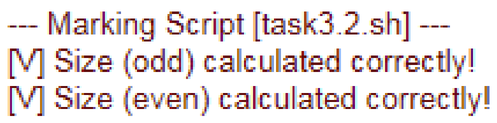
Figure 11 shows the code for Task 3.2

Referring to Figure 11, Task 3.1 shows us the total of all of the files of the choice of directory and if it is odd or even. For this to be done, I used if statements. This is done by the odd being size being divided by 2 and the remainder being recognised by it being odd if it is not equal. This is the same for even. Whilst the if statement is going for both even and odd, it adds both of them together. One this is complete, "echo $total" displays the total number. The same process is done for both even and odd.

Figure 12 shows the result of the code running in through script check.



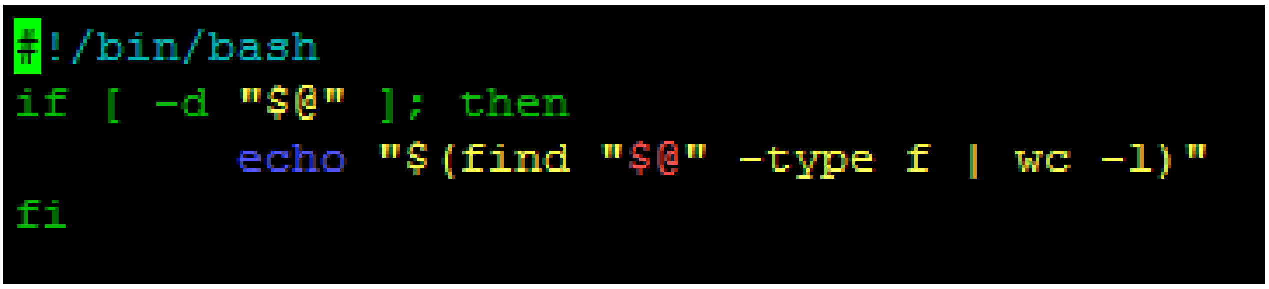
```
student:~> ./task3.2.sh ~/Desktop odd
1532
student:~> ./task3.2.sh ~/Desktop even
1450
```



```
--- Marking Script [task3.2.sh] ---
[V] Size (odd) calculated correctly!
[V] Size (even) calculated correctly!
```

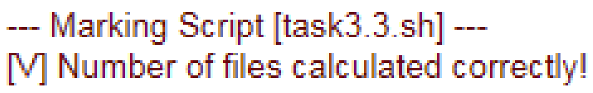
Figure 12 shows that this code is correct through Script Check.

Task 3.3

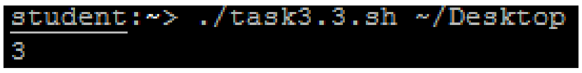


```
#!/bin/bash
if [ -d "$@" ]; then
        echo "$(find "$@" -type f | wc -l)"
fi
```

Figure 13 shows the code for Task 3.3



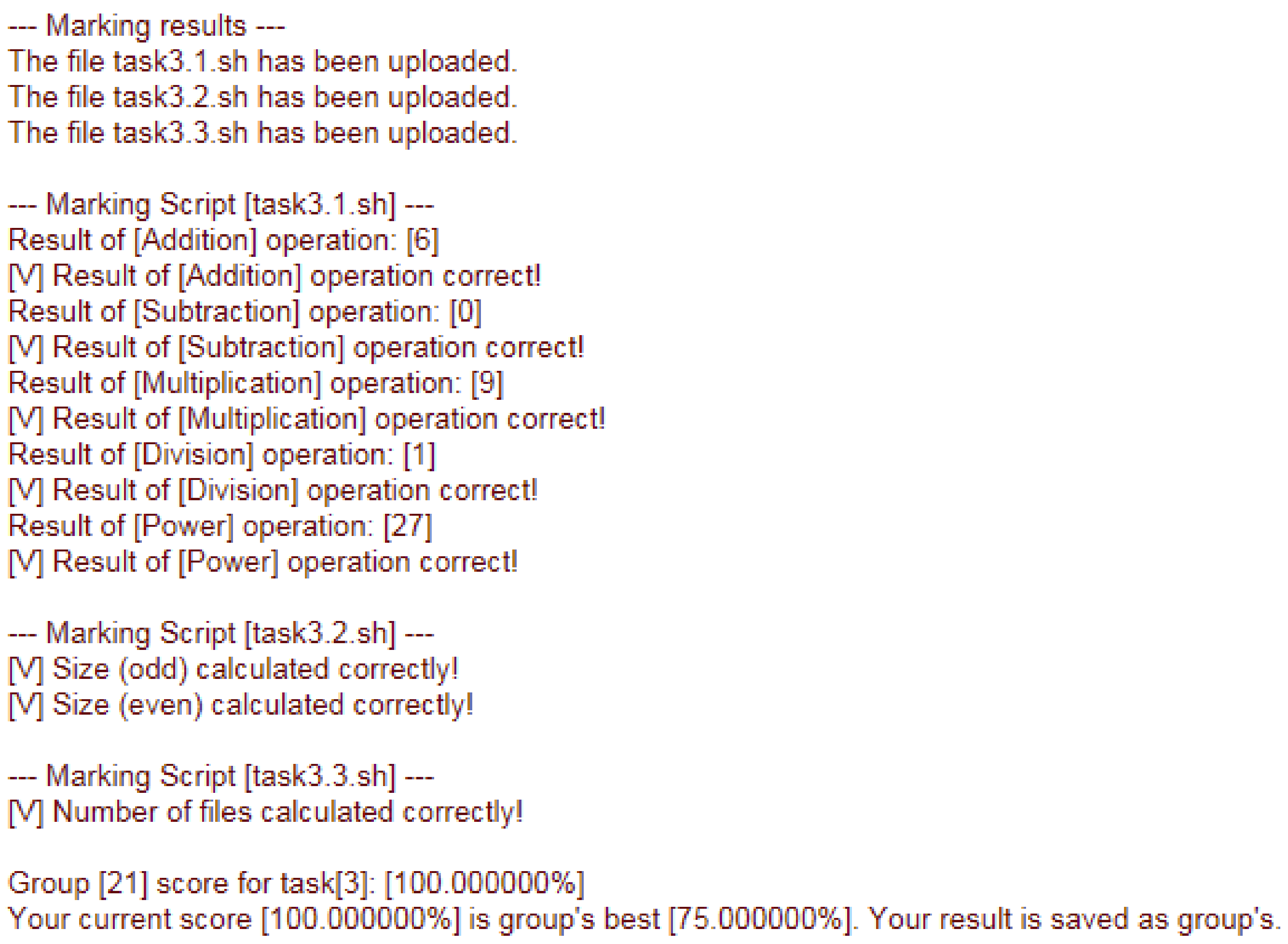
```
--- Marking Script [task3.3.sh] ---
[V] Number of files calculated correctly!
```



```
student:~> ./task3.3.sh ~/Desktop
3
```

Figure 14 shows that this code is correct through Script Check.

Figure 13 shows the code of how it was implemented. This code finds how many files are there in the example and displays it as a number. As you can see, I showed an example of this on my Desktop drive and it says 3. Figure 14 shows that this is correct.



```
--- Marking results ---
The file task3.1.sh has been uploaded.
The file task3.2.sh has been uploaded.
The file task3.3.sh has been uploaded.

--- Marking Script [task3.1.sh] ---
Result of [Addition] operation: [6]
[V] Result of [Addition] operation correct!
Result of [Subtraction] operation: [0]
[V] Result of [Subtraction] operation correct!
Result of [Multiplication] operation: [9]
[V] Result of [Multiplication] operation correct!
Result of [Division] operation: [1]
[V] Result of [Division] operation correct!
Result of [Power] operation: [27]
[V] Result of [Power] operation correct!

--- Marking Script [task3.2.sh] ---
[V] Size (odd) calculated correctly!
[V] Size (even) calculated correctly!

--- Marking Script [task3.3.sh] ---
[V] Number of files calculated correctly!

Group [21] score for task[3]: [100.000000%]
Your current score [100.000000%] is group's best [75.000000%]. Your result is saved as group's.
```

Reflection

During the time and effort I spent on this, I felt that I did much research on all the examples for me to understand the each task. I felt that the examples helped very much, because it gave me a platform to work on. The one I struggled on was the first one the most. I felt that I spent much time on this the most, because it was the most challenging one out of them all. These challenges took time and effort to be completed and I felt that with my group members, we were able to complete this task and get the result above. Overall, I felt that I have understood the basic understanding and importance of how to run and produce the code effectively.