# COMP1562, Lab #4 Shell Programming I

|  |  |  |  |
| --- | --- | --- | --- |
| Lecturer | **Mariusz Pelc** | Phone | **020 83318588** |
| Office | **QM366** | e-mail address | [**m.pelc@gre.ac.uk**](mailto:m.pelc@gre.ac.uk) |
| Office hours | **Mon 4-5pm, Wed 9-10pm** |  |  |

**(by M. Pelc and K. McManus)**

**Description:**

**This exercise is mainly focused on Linux introduction to Linux shell programming.**

**Learning Outcomes:**

**Students will be able to write very simple scripts in Linux shell. They will be able to apply conditional instructions and loop instructions. The report should contain results showing solutions for the tasks specified in the *Tasks* section**[[1]](#footnote-1)**. The below examples may be of used while trying to solve the tasks (the best idea is to check how the example scripts work).**

**Do not test ANY of the scripts if you are in your home directory. Test them in ~/bin DIRECTORY (create the directory if it does not exist).**

**EXERCISES**

You are supposed to check how each of the three below script works. Ideally you should re-write them in your system (either, stulinux.cms.gre.ac.uk or your virtual host system) and then execute them one by one thoroughly inspecting the way they work and what the commands that are used in these scripts work.

**Example 1: Handling improper number of script arguments**

#!/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

**Example 2: Simple calculator and user interaction**

#!/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 “reminder=arg1 % arg2”

echo –e “results:\n”

echo “$arg1+$arg2=$addition”

echo “$arg1-$arg2=$subtraction”

echo “$arg1\*$arg2=$multiplication”

echo “$arg1/$arg2=($division+$reminder/$arg2)”

**Example 3: Listing all subdirectories of the directory name given as the script argument**

#!/bin/bash

echo “Directory <$1> contains the following subdirectories:”

ls $1 |

while read file

do

if [ -d $1/$file ]

then

echo “[$file]”

fi

done

**Example 4: List files which size is an odd number**

#!/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

**TASKS**

**Assuming that you have checked how the above scripts work and also assuming that you read all the information about shell programming provided in the PowerPoint slides for Lecture 3 you are supposed to solve the below tasks. All scripts will be tested automatically with respect to satisfying the specified requirements. All scripts will have to be written so that they were processing command line parameters and returning result of the operation. This result will be evaluated against strictly specified test parameters (which you will not know, but these test parameters will have to produce strictly specified results; if the result will be as expected, you will get full marks for the script, otherwise you will get 0 marks). NO INFINITE LOOPS AND NO READ COMMAND – BOTH WILL HANG SCRIPTCHECK!**

* 1. (Script 1) You are supposed to MODIFY example script provided in the **Example 2** so that it would perform operations “+”, “-“, “\*”, “/” (only the whole part of the division operation, ignore the reminder part) and “^” (power). When it comes to the first four operations, they are normally supported by “let” command or the other alternative ways of calculating numerical expressions mentioned in the lecture slides (and these are actually already programmed in the **Example 2**). However, the power operation will need to be implemented by you, probably in the form of some sort of loop operations (I am leaving this to you to decide). Once you write the script, you will be required to upload it to the scriptcheck.cms.gre.ac.uk system for test purposes. The marking system will assume that the script will be processing three command line parameters, one for the first argument of arithmetic operation, the second command line parameter for the second argument of the arithmetic operations and third parameter expressing the arithmetic operation to be processed. Your script will be executed from the command line, for example, in order to check if your solution is correct it might be executed as:

scriptname 4 2 “operation”

The scriptcheck system will check if you script returns appropriate results for each operation; for the example execution parameters (4 and 2) it should return:

* 4 2 “+” 🡪**6**
* 4 2 “-“ 🡪**2**
* 4 2 “\*” 🡪**8**
* 4 2 “/” 🡪**2**
* 4 2 “^” 🡪**16**

The only thing your script is expected to **return** is **a number** corresponding to result of a given operation. Neither, runtime messages nor error messages should appear on the screen (you may need to re-direct the standard error stream for operations that might be source of any kind of error as “*command 2>/dev/null”*).

Your solution will be marked with respect to the correctness of the particular operation.

**#!/bin/bash**

**if [ $# -eq 2 ]; then**

**echo "You’ve entered: arg1=$1 and arg2=$2"**

**let "addition=$1+$2"**

**let "subtraction=$1-$2"**

**let "multiplication=$1\*$2"**

**let "division=$1/$2"**

**let "power=$1\*\*$2"**

**echo -e "results:"**

**echo "$1+$2=$addition"**

**echo "$1-$2=$subtraction"**

**echo "$1\*$2=$multiplication"**

**echo "$1/$2=$division"**

**echo "$1^$2=$power"**

**fi**

**#!/bin/bash**  
**echo "Enter the first number"**  
**read num1**  
**echo "Enter the second number"**  
**read num2**  
**echo "What operation do you want to use?"**  
**read op**  
  
**case "$op" in**  
**"+")**  
**echo `expr $num1 + $num2`;;**  
**"-")**  
**echo `expr $num1 - $num2`;;**  
**"\*")**  
**echo `expr $num1 \\* $num2`;;**  
**"/")**  
**echo `expr $num1 / $num2`;;**  
**"^")**  
**echo "$num1^$num2" | bc**

**easc**

**#!/bin/sh**

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

**echo `expr $1 + $2`**

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

**echo `expr $1 - $2`**

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

**echo `expr $1 / $2`**

**elif [ $# -eq 3 ] && [ $3 = "x" ]; then**

**echo `expr $1 \\* $2`**

**elif [ $# -eq 3 ] && [ $3 = "^" ]; then**

**echo "$1^$2" | bc**

**fi**

* 1. (Script 2) You are supposed to write a shell script that will count total size of files which **size**  is an odd or an even number (depending on the parameter given in the command line at the time the script is executed). The marking system will assume that the script will be processing two command line parameters, directory name should be first parameter and then “odd” or “even” should follow to express which files should be counted in. For the test purposes a test environment has been prepared in the scriptcheck system with a number of files. Your script will be executed in the command line as:

scriptname dir\_name odd

to calculate total size of all files of odd size.

scriptname dir\_name even

to calculate total size of all files of even size.

Suppose you’ve got the following directory structure:

A-┐

└-a.txt (size 5 bytes)

└-b.txt (size 7 bytes)

└-c.txt (size 1 byte)

└-d.txt (size 4 bytes)

For the example execution:

scriptname ./A odd

your script should return 13

and for example execution:

scriptname ./A even

your script should return 4.

The only thing your script is expected to **return** is **a number** reflecting the total size of files of odd or even size. Neither, runtime messages nor error messages should appear on the screen (you may need to re-direct the standard error stream for operations that might be source of any kind of error as “*command 2>/dev/null”*).

Your solution will be marked with respect to the correctness of the particular operation. Script shown in the **Example 4** may be a good start, but if you decide to go your own way, you can do it.

* 1. (Script 3) You are supposed to write a shell script that will count the number of files stored in the directory given as the script execution parameter. For the test purposes a test environment has been prepared in the scriptcheck system with a number of files. You **must** use loop instruction. Your script will be executed in the command line as:

scriptname dir\_name

Suppose you’ve got the following directory structure:

A-┐

└-a.txt (size 5 bytes)

└-b.txt (size 7 bytes)

└-c.txt (size 1 byte)

└-d.txt (size 4 bytes)

For the example execution:

scriptname ./A

your script should return 4.

The only thing your script is expected to **return** is **a number** reflecting the total number of files of the given directory. Neither, runtime messages nor error messages should appear on the screen (you may need to re-direct the standard error stream for operations that might be source of any kind of error as “*command 2>/dev/null”*).

Your solution will be marked with respect to the correctness of the files counting operation.

**Techniques/resources:**

**Solution of all the above tasks will require using any Linux machine (in the labs or at home) or your own virtual machines. The scriptcheck.cms.gre.ac.uk system should be used to evaluate your solution with respect to the specified requirements.**

**Marking:**

**The solutions will be marked in the range 0-100%.**

**Deadline:**

**The solutions should be delivered within one week from the lab date.**

1. Not to mention commenting on the results / critical evaluation of the results ☺ [↑](#footnote-ref-1)