EECS2031 Winter 2018

Software Tools

Assignment 1 (15%): Shell Programming

Due Date: 11:59 pm on Friday, Feb 9, 2018

Objective

In this assignment, you will be writing four shell programs. The first program (**myrm.sh**) is designed to emulate a safer *rm* command. The second program (**guess.sh**) emulates a simple number guessing game. The third program (**calculator.sh**) is a tool that emulates a basic calculator. The fourth program (**utilities.sh**) is an interactive tool for running basic utility commands.

Important Notes:

- You must use the *submit* command to electronically submit your solution by the due date.
- All programs are to be written using *#!/bin/bash*.
- Your programs should be tested on the *EECS labs* before being submitted. Failure to test your programs on *EECS labs* will result in a mark of 0 being assigned.
- To get full marks, your code must be well-documented (code comments start with "#").

What To Submit

When you have completed the assignment, move or copy your four shell scripts in a directory (e.g., assignment1), and use the following command to electronically submit your files within that directory:

```
% submit 2031M a1 myrm.sh guess.sh calculator.sh utilities.sh
```

You can also submit the files individually after you complete each part of the assignment– simply execute the *submit* command and give the filename that you wish to submit. You may submit your solutions as many times as you wish prior to the submission deadline. Make sure you name your files **exactly** as stated (including lower/upper case letters). Failure to do so will result in a mark of 0 being assigned. You may check the status of your submission using the command:

```
% submit -l 2031M a1
```

A. Safe Delete (25%)

The UNIX command *rm* does not provide for recovery of deleted files through a mechanism like the recycle bin typically found in modern operating systems. Write a shell script "**myrm.sh**" that is designed to replace *rm*. It will move the target file(s) specified as parameters to a recycled bin directory instead of really deleting them. Your script should:

- Create a new directory to be used as the recycle bin.
- Print an error message if there are no files specified as parameters, where the correct usage is also explained.
- Print a feedback message "deleting <filename>" anytime a file with name *filename* is deleted.

Here is a sample output from execution of the program:

```
$ ./myrm.sh b c
deleting b
deleting c
$ ./myrm.sh
Error:no target specified
Usage:./myrm <files>
$
```

Example run

Hint: To make it easier for the marking process, your recycle bin must be located in the path:

*"/tmp/**yourlogin**/eecs2031m/a1/recycle-bin"*

Where ***yourlogin*** refers to your EECS lab login names. Use the unix "*mkdir –p*" command to create this directory path inside your script (the *–p* option of *mkdir* will create all parent directories up to recycle-bin if they don't exist).

B. Guess a Random Integer (25%)

Write a shell script "**guess.sh**" that emulates a simple number guessing game. Your script should:

- Define a random integer in the range of 1 to 64 (inclusive) for the user to guess.
- Display a welcoming message to the user.
- Prompt the user to guess a number between 1 to 64 (inclusive).
- Provide a hint to the user each time he/she enters a guess by displaying a message that reads "Too small." or "Too big." accordingly.
- If a correct guess is entered then display the message "You won!" and exit.
- Allow the user to guess a maximum of 6 times. After the 6th attempt display a message "You lost!" and exit.

Here is a sample output from multiple executions of the program:

$ ./guess.sh Welcome to the number game. Guess a number between 1 and 64 (inclusive). 32 Too small. Try again. 48 Too big. Try again. 40 Too small. Try again. 44 Too small. Try again. 46 You won! $	$ ./guess.sh Welcome to the number game. Guess a number between 1 and 64 (inclusive). 12 Too small. Try again. 17 Too small. Try again. 22 Too small. Try again. 30 Too small. Try again. 40 Too small. Try again. 50 Too small. You lost! $
Example run 1 (User won!)	Example run 2 (User lost!)

Hint: **$RANDOM** returns a different random integer at each invocation in range: 0-32767.

C. Basic Calculator (25%)

Write a shell script **"calculator.sh"** that performs basic calculator functions such as addition (+), subtraction (-), multiplication (x) and division (/). The program takes as parameters

- An integer value for the left operand.
- A character that represents the type of operation to be performed, identified by one of the characters ('+', '-', 'x', '/') respectively.
- An integer value for the right operand.

and outputs the result of the operation as an integer. If the requested operation would require a division-by-zero, your program should print an error message *"Division-by-zero Error!"* and not a result.

Here is a sample output from multiple executions of the program:

```
$ ./calculator.sh
Usage - ./calculator.sh  value1  operator value2
where,
value1: numeric value
value2: numeric value
operator: one of +, -, /, x
$ ./calculator.sh  3 + 4
7
$ ./calculator.sh  3 - 4
-1
$ ./calculator.sh  3 x 4
12
$ ./calculator.sh  3 * 4
Usage - ./calculator.sh  value1  operator value2
where,
value1: numeric value
value2: numeric value
operator: one of +, -, /, x
$ ./calculator.sh  3 / 6
0
$ ./calculator.sh  3 / 3
1
$ ./calculator.sh  3 / 0
Division-by-zero Error!
$
```

Example run

D. Basic Utilities Interactive Menu (25%)

Write a shell script **"utilities.sh"** that performs basic utility task using an interactive menu. Below is the main logic of your shell script. You will have to repeat the menu over and over in an eternal loop and perform the following steps:

- Clear the screen and display the menu.
- Read a line of input from the keyboard.
- Check the user answer and dispatch to the appropriate part of your script (use case statement).
- If the entry was invalid print a correct usage message.

Your menu should include options for showing:

1. What is the ongoing processor activity (use the *top* unix command).
2. Information about the users currently logged in.
3. The number of users currently logged in.
4. The login names only of the users that use bash, sorted and with duplicates removed (see shell slides)

Here is a sample output from multiple executions of the program:

---------- M A I N - M E N U ---------- 1. Ongoing Processor Activity 2. Users currently logged in 3. Number of users currently logged in 4. Users with bash shell 5. Exit ---------- Please enter option [1 - 5]: 4 manospapagelis Press [enter] key to continue. . .	---------- M A I N - M E N U ---------- 1. Ongoing Processor Activity 2. Users currently logged in 3. Number of users currently logged in 4. Users with bash shell 5. Exit ---------- Please enter option [1 - 5]: 3 4 Press [enter] key to continue. . .
Example run 1	Example run 2