“Network Monitoring Engineer”
Take Home Task

Víctor Miñano Belvis
Noltemeyerhöfe 3
38114 Braunschweig
Tel: (+34) 650851903
E-Mail: victor.mb1992@gmail.com

Table of Contents

1. Task 1

Please design a script that writes the numbers from 1 - 10 in random order.

Each number should appear only once. You can use bash only.

Please provide tests for the script, along with documentation which should include the following:

- Build instructions
- Usage
- Description
- Known limitations / bugs

1.1. Introduction

The objective of this task is to design a script that writes the numbers from 1 to 10 in random order. Each number should appear only once.

I wrote a bash shell script in order to achieve this goal.

1.2. Build instructions

I have used Atom software in MacOS System to write the script in Plain Text. The idea is to create an array to store each value. Therefore, the final array should have 10 values stored with numbers from 1 to 10 without repeating any of them and with a random order.

The first step is to create the script. So, a new file has to be created in Atom, in "File > New File". Then, it has to be saved in "File > Save As…". In this case I have saved it with the name "adjust_task_1". In order to make the file become an executable file, in the terminal, in the path where the file has been stored, it has to be introduced the following command:

```
chmod 700 adjust_task_1
```

Once we have the executable, we can write the script and test it. The array is going to be generated within a "while loop". The loop has to be executed until the index number "i" (which starts in 0) reaches the value 10:

```
i="0"
while [ $i -lt 10 ]
```

In this way, when the index is equal to 10, the loop is not executed anymore and therefore, we will have 10 values in the array.

Within the loop, a random number between 1 and 10 has to be generated and stored in the array. So, first, within the loop, the random number is generated and stored in the variable "value":

```
value="$((($RANDOM % 10) + 1))"
```

Then, this number has to be stored in the array only if it does not exist in the array yet, otherwise, it has to be skipped and a new random number has to be generated:

```
if [[ ! " ${array[@]} " =~ " ${value} " ]]
 then
  array[i]=$value
  ((i=i+1))
```

With the previous condition, if the value is not in the array, the value is stored in the array with the corresponding index, and then the index "i" has to be increased. "i" is only increased when a new value is stored in the array.

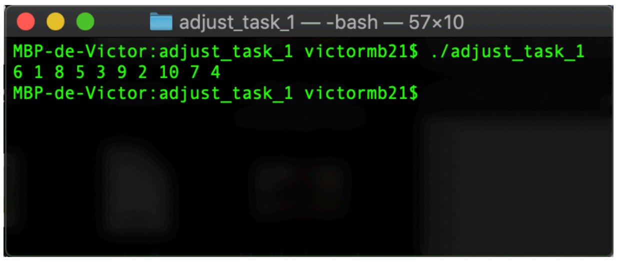
Finally, to check the numbers generated, it is printed the array:

```
echo "${array[*]}"
```

1.3. Usage

Once the script is generated and programmed, it can be executed via the console of the terminal. In the console, it has to be executed the script by introducing the path to the file *"path/to/file/adjust_task_1"* or going to the folder where the script is and the executing it *"./adjust_task_1"*. Then, the console will execute the script and, in the console, will be shown the 10 numbers from 1 to 10 in a random order and showing each number only once.

Figure 1.1 shows the execution of the script.



```
MBP-de-Victor:adjust_task_1 victormb21$ ./adjust_task_1
6 1 8 5 3 9 2 10 7 4
MBP-de-Victor:adjust_task_1 victormb21$
```

Figure 1.1. adjust_task_1 script execution

1.4. Description

Figure 1.2 shows the flowchart of the script where it is described its behaviour. When the program starts, and index "i" is initialized to 0. Then the program enters in the loop where it is checked if the index is less than 10. If so, a random number between 1 and 10 is generated. Then, it is checked if this number was already stored in the array. If not, then the number is stored in the array, the index is increased, and the program again goes to the point where the index is checked if it is less than 10 or not. If the random number generated already exists in the array, the value is skipped, the index is not increased, and the program again goes to the point where the index is checked. Once the index is equal to ten, the condition of the loop is not met anymore, and the program exists the loop. Then the array with the 10 numbers from 1 to 10 in a random order is printed and the program ends.

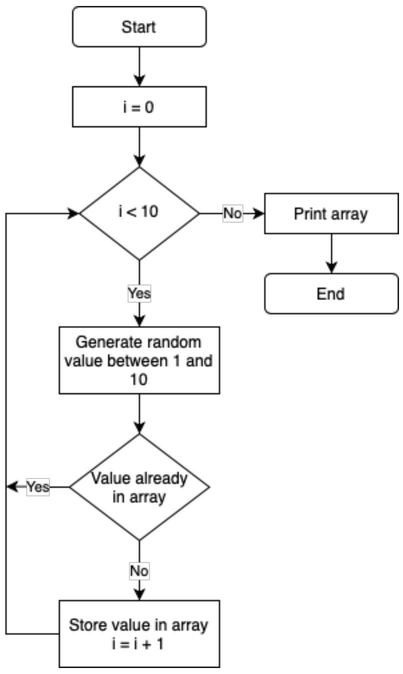
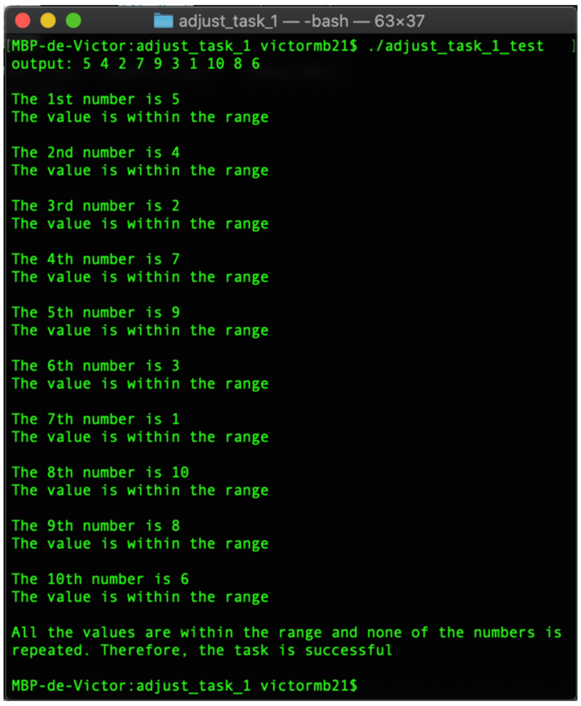


Figure 1.2. adjust_task_1 script flowchart

1.5. Tests

In order to perform some tests of the script, I have programmed another script. This new script executes the previous script and gets its output, that is, the array with the 10 numbers from 1 to 10 in a random order. Then the test script analyzes if each number is between 1 and 10 and if any of the numbers has been repeated. If all the conditions of the tasks are fulfilled, then the test script prints a message of success. In case the conditions are not met, the script prints a non-success message.

Figure 1.3 shows the execution of this script and the successful result.



```
MBP-de-Victor:adjust_task_1 victormb21$ ./adjust_task_1_test
output: 5 4 2 7 9 3 1 10 8 6

The 1st number is 5
The value is within the range

The 2nd number is 4
The value is within the range

The 3rd number is 2
The value is within the range

The 4th number is 7
The value is within the range

The 5th number is 9
The value is within the range

The 6th number is 3
The value is within the range

The 7th number is 1
The value is within the range

The 8th number is 10
The value is within the range

The 9th number is 8
The value is within the range

The 10th number is 6
The value is within the range

All the values are within the range and none of the numbers is
repeated. Therefore, the task is successful

MBP-de-Victor:adjust_task_1 victormb21$
```

Figure 1.3. adjust_task_1_test script execution

1.5.1. "adjust_task_1_test" script description

Figure 1.4 shows the flowchart of the script for the tests. First the program starts, runs the "adjust_task_1" script and stores the output in an array. Then a variable is created with value the length of the array, which is going to be useful to read each value from the array. Index "i" is initialized to 0 and the Boolean variable "successful" is initialized to "true".

Then the program enters in the loop where it is checked if the index is less than the length of the array. If so, it is got the value "i" from the array. First, it is checked whether the value is or not in the temporal array. If the value is not in the temporal array, the value is stored in this array. But it the value was already in the temporal array, before storing it, the "successful" Boolean variable is changed to "false". Then, it is checked whether the value is between 1 and 10. If the value is not between 1 and 10, the "successful" Boolean variable will be changed to "false". Then, the index "i" is increased and the program goes again to the point where the index is checked. Once the index is equal to the array length, the condition of the loop is not met anymore, and the program exists the loop.

Finally, it is checked whether "successful" is "true" or "false". If "true", all the conditions have been met and the task is successful. If "false", the task is not successful.

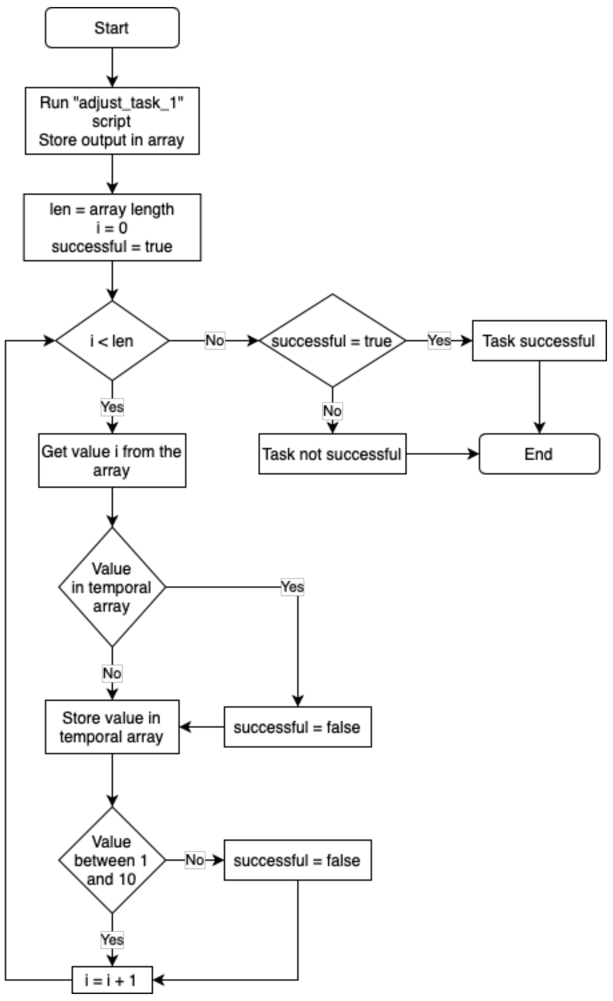


Figure 1.4. adjust_task_1_test script flowchart

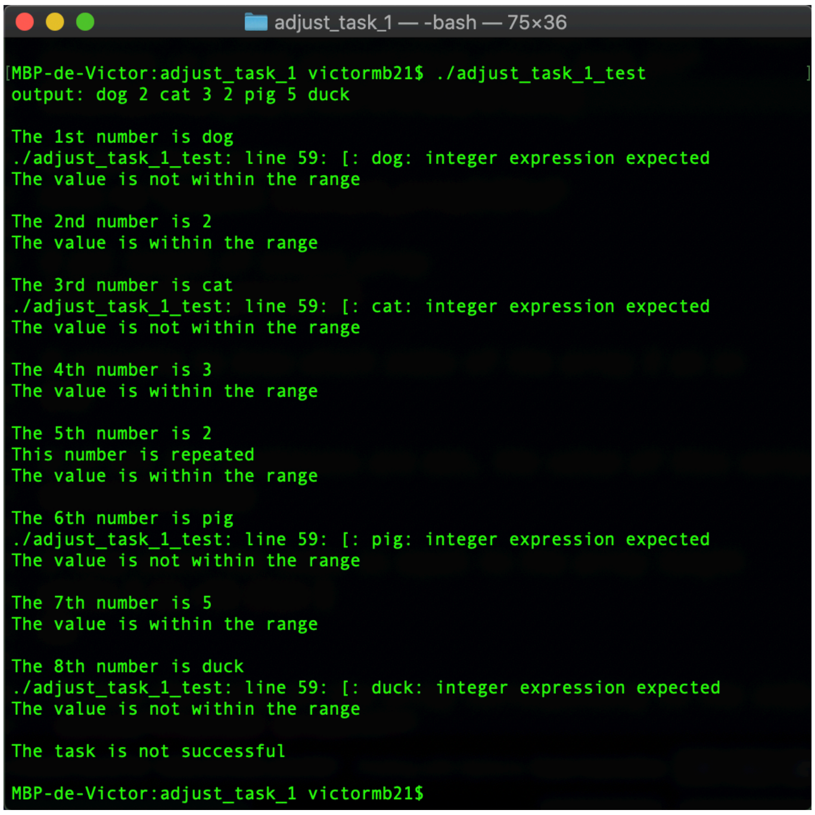
1.6. Known limitations/bugs

The script (*adjust_task_1*) has been programmed to always show the 10 numbers, each of them within the range from 1 to 10 and without repeating any.

The main limitations that I find are related to the way of testing this script. In section 1.5 I explain the way I test this script. In order to achieve this goal, I programmed another script (*adjust_task_1_test*) which executes the other script and analyzes its output. This test script can check the output of the other script only if it is an integer-array-based output. When the output of the script is an array of integers, then the test script can be executed successfully. However, in case of an unexpected output, e.g. other format, the test script would throw an error and cannot be executed properly.

In order to check this supposed wrong behaviour, only for this test, I have modified the array that is going to be handled throughout the test script. In this case the array, instead of being an array of 10 integers, it is going to be an array with both integers and strings. Figure 1.5 shows the execution of the script with this array. It can be seen that the script is not properly executed and the are some errors when trying to process the content of the array since the script expects to work only with integers.

Anyway, this is not a real case since the "*adjust_task_1*" always has an array of integers of size 10 as output.



adjust_task_1 — -bash — 75×36

```
[MBP-de-Victor:adjust_task_1 victormb21$ ./adjust_task_1_test              ]
output: dog 2 cat 3 2 pig 5 duck

The 1st number is dog
./adjust_task_1_test: line 59: [: dog: integer expression expected
The value is not within the range

The 2nd number is 2
The value is within the range

The 3rd number is cat
./adjust_task_1_test: line 59: [: cat: integer expression expected
The value is not within the range

The 4th number is 3
The value is within the range

The 5th number is 2
This number is repeated
The value is within the range

The 6th number is pig
./adjust_task_1_test: line 59: [: pig: integer expression expected
The value is not within the range

The 7th number is 5
The value is within the range

The 8th number is duck
./adjust_task_1_test: line 59: [: duck: integer expression expected
The value is not within the range

The task is not successful

MBP-de-Victor:adjust_task_1 victormb21$
```

Figure 1.5. adjust_task_test with wrong array

2. Task 2

Imagine a server with the following specs:

- 4 times Intel(R) Xeon(R) CPU E7-4830 v4 @ 2.00GHz
- 64GB of ram
- 2 tb HDD disk space
- 2 x 10Gbit/s nics

The server is used for SSL offloading and proxies around 25000 requests per second. Please let us know which metrics are interesting to monitor in that specific case and how would you do that? What are the challenges of monitoring this?

2.1. Answer

Interesting metrics

Since SSL offloading is capable of freeing up the CPU processing to servers, it is important to monitor the CPU-related metrics. Among these metrics it is important to monitor CPU usage, CPU load or CPU temperature.

Furthermore, for the proper operation of SSL offloading it has to be monitor other important metrics such as disk usage, network traffic or memory usage.

Summing up, the most interesting metrics to monitor in this specific case are:

- CPU usage
- CPU load
- CPU temperature
- disk usage
- Network traffic
- Memory usage

In order to monitor these metrics, it can be used a network monitoring software such as Zabbix which is an open source monitoring software solution for networks, operating systems and applications.

Challenges

Some important challenges of monitoring this is to know exactly the most interesting metrics that have to be monitor in order to perform a proper load balancing.

Another important challenge to take into account is that the monitoring tool should not affect the server performance.