



boa3444 / Linux\_Lab



&lt;&gt; Code

Issues

Pull requests

Actions

Projects

Wiki

Security



Linux\_Lab / Assignments / LAB3.md



boa3444 Update LAB3.md

5447456 · now



135 lines (103 loc) · 5.65 KB

Preview

Code

Blame



Raw



# Lab 3 : Modifying an Existing Script

Objective: Enhance and customize a script.

Script I used: `print_numbers.sh` from `Scripts` folder



## Original vs. Enhanced Script Behavior



### Original Script: `print_numbers.sh`




A humble loop with a fixed mindset — it prints numbers 1 to 5, no questions asked. Like a playlist stuck on repeat, it delivers the same output every time. No input, no validation, no surprises. It's reliable, but rigid — a one-size-fits-all solution in a world that craves customization.



```
for i in 1 2 3 4 5
do
  echo "Number: $i"
done
```



### Enhanced Script: `enhanced_numbers.sh`

This isn't your average loop. It's a command-line chameleon:

-  Smart enough to validate your inputs
-  Flexible enough to count up or down
-  Safe enough to reject nonsense like a zero step

-  You can flip the direction of loop.
-  Customizable with just three arguments: `start` , `end` , and `step`

## How It Works

- You feed it three numbers.
- It checks if they make sense.
- Then it prints a sequence tailored to your input — ascending or descending, fast or slow.

## Sample Run Code Outputs:

```
vboxuser@ubuntu: ~/Documents/Linux_Lab/Arrays
vboxuser@ubuntu:~/Documents/Linux_Lab/Arrays$ ./enhanced_numbers.sh 1 4 1
Iteration: 1
Iteration: 2
Iteration: 3
Iteration: 4
vboxuser@ubuntu:~/Documents/Linux_Lab/Arrays$ ./enhanced_numbers.sh 4 1 1
Iteration: 4
Iteration: 3
Iteration: 2
Iteration: 1
vboxuser@ubuntu:~/Documents/Linux_Lab/Arrays$ ./enhanced_numbers.sh 4 1 A
Your step value should be a non-zero, positive integer only.
vboxuser@ubuntu:~/Documents/Linux_Lab/Arrays$ ./enhanced_numbers.sh 4 0 -2
Your step value should be a non-zero, positive integer only.
vboxuser@ubuntu:~/Documents/Linux_Lab/Arrays$ ./enhanced_numbers.sh A B 9
Your range of numbers must only be an integer.
vboxuser@ubuntu:~/Documents/Linux_Lab/Arrays$ ./enhanced_numbers.sh 3 9 0
Your step value should be a non-zero, positive integer only.
vboxuser@ubuntu:~/Documents/Linux_Lab/Arrays$ ./enhanced_numbers.sh 3 9 3
Iteration: 3
Iteration: 6
Iteration: 9
vboxuser@ubuntu:~/Documents/Linux_Lab/Arrays$ ./enhanced_numbers.sh 6 10 4
Iteration: 6
Iteration: 10
vboxuser@ubuntu:~/Documents/Linux_Lab/Arrays$ ./enhanced_numbers.sh 9 11 2
Iteration: 9
Iteration: 11
vboxuser@ubuntu:~/Documents/Linux_Lab/Arrays$ ./enhanced_numbers.sh 9 h 2
Your range of numbers must only be an integer.
vboxuser@ubuntu:~/Documents/Linux_Lab/Arrays$
```

## EXTRA QUESTIONS:

### Q1: What does `$1` mean in a Bash script?

A: `$1` refers to the first argument passed to the script when it's executed. For example, if you run `./script.sh apple banana` , then `$1` will be `apple` .

### Q2: What is `$@` used for?

A: `$@` represents all the arguments passed to the script, treated as separate words. It's perfect for looping through each input individually. Example:

```
for item in "$@"; do
    echo "$item"
done
```



### Q3: What does \$# tell us?

A:

You can't use 'macro parameter character #' in math mode

`$#` gives the total number of arguments passed to the script. If you run `./script`.



`#` will be 3.

### Q4. What does exit 1 do in a script?

A: `exit 1` stops the script and returns an error code (1) to the shell, signaling that something went wrong. It's commonly used after input validation fails or when a critical condition isn't met.

```
if [ $# -ne 3 ]; then
    echo "Error: Please provide exactly three arguments."
    exit 1
fi
```



### Q5: What's the difference between exit 0 and exit 1?

A:

- `exit 0` → Success. The script completed without issues.
- `exit 1` (or any non-zero value) → Failure. Something went wrong, and the script exited early.



## Appendix: Raw Markdown Source (LAB3.md)

Due to a limitation on the submission portal — which only allows uploading **two files** — I've embedded the full raw Markdown source of LAB3.md directly into this PDF. This ensures all three required deliverables ( enhanced\_numbers.sh , LAB3.md , and LAB3.pdf ) are included and accessible for review. The content below reflects the original Markdown file exactly as written.



# Lab 3 : Modifying an Existing Script

Objective: Enhance and customize a script.

## Script I used: `print\_numbers.sh` from `Scripts` folder

## 🌱 Original vs. 🚀 Enhanced Script Behavior

### 📁 Original Script: `print\_numbers.sh`

A humble loop with a fixed mindset — it prints numbers 1 to 5, no questions asked. Like a playlist stuck on repeat, it delivers the same output every time. No input, no validation, no surprises. It's reliable, but rigid — a one-size-fits-all solution in a world that craves customization.

```
```bash
for i in 1 2 3 4 5
do
    echo "Number: $i"
done
```
```

## 🚀 Enhanced Script: `enhanced\_numbers.sh`

This isn't your average loop. It's a command-line chameleon:

- 🧠 Smart enough to validate your inputs
- 🔄 Flexible enough to count up or down
- 🛡️ Safe enough to reject nonsense like a zero step
- 🔄 You can flip the direction of loop.
- 🛠️ Customizable with just three arguments: `start`, `end`, and `step`

### 🔍 How It Works

- You feed it three numbers.
- It checks if they make sense.
- Then it prints a sequence tailored to your input — ascending or descending, fast or slow.

### 📄 Sample Run Code Outputs:

![]

([https://github.com/boa3444/Linux\\_Lab/blob/1473fcac56a2b52f7fca02f8f7d29435](https://github.com/boa3444/Linux_Lab/blob/1473fcac56a2b52f7fca02f8f7d29435))

## EXTRA QUESTIONS:

### Q1: What does `\$1` mean in a Bash script?

**\*\*A:\*\*** ``$1`` refers to the first argument passed to the script when it's executed. For example, if you run `./script.sh apple banana``, then ``$1`` will be ``apple``.

---

**### Q2:** What is ``$@`` used for?

**\*\*A:\*\*** ``$@`` represents all the arguments passed to the script, treated as separate words. It's perfect for looping through each input individually. Example:

```
```bash
for item in "$@"; do
    echo "$item"
done
```
```

**### Q3:** What does ``$#`` tell us?

**\*\*A:\*\*** ``$#`` gives the total number of arguments passed to the script. If you run `./script.sh apple banana cherry``, then ``$#`` will be ``3``.

**### Q4:** What does ``exit 1`` do in a script?

**\*\*A:\*\*** ``exit 1`` stops the script and returns an error code (``1``) to the shell, signaling that something went wrong. It's commonly used after input validation fails or when a critical condition isn't met.

```
```bash
if [ $# -ne 3 ]; then
    echo "Error: Please provide exactly three arguments."
    exit 1
fi
```
```

**### Q5:** What's the difference between ``exit 0`` and ``exit 1``?

**\*\*A:\*\***

- ``exit 0`` → Success. The script completed without issues.
- ``exit 1`` (or any non-zero value) → Failure. Something went wrong, and the script exited early.