

# Interlude: PA4a

These Practice Exercises are meant to help you review for our next IE.

## Exercise #1 - Warming up with Hello World

Write a bash script to echo the string “Hello World!”. Call the script `hello.sh`, an example execution is as follows:

```
tux@LinuxBox > ./hello.sh  
Hello World!  
tux@LinuxBox >
```

## Exercise #2 - parameters.sh - version #1

We need a bash script named `parameters.sh` able to display all the positional parameters that were passed to it.

Here is an example of how it would be used;

```
tux@LinBox > ./parameters.sh one two three four five
one two three four five
tux@LinBox >
```

Notes;

- We want to get this done by using a single `echo` statement
- Try using both the `$*` and `$@` bash special variables

## Exercise #3 - parameters.sh - version #2

Let us improve our parameters.sh script by allowing it to display a message stating how many positional parameters were received, followed by the list of parameters between parenthesis.

Here is an example of how it would be used;

```
tux@LinuxBox > ./parameters.sh one two three four five  
Received 5 parameters (one two three four five)  
tux@LinuxBox >
```

### Note;

- You should still be able to get this done using a single echo statement

## Exercise #4 - reverse.sh – version #1

We want a bash script named `reverse.sh` which is able, given two words as positional parameters, to display them in reverse order.

Here is an example of how it would be used;

```
tux@LinuxBox > ./reverse.sh first next
next
first
tux@LinuxBox >
```

To do so, write and test a Bash Script which will;

- Assign its first parameter to a variable named FIRST
- Assign its second parameter to a variable named SECOND
- Displays these two variables one after the other, starting with SECOND
- Use two echo statements, one for displaying each of the variables

## Exercise #5 - reverse.sh - version #2

We want to improve the above `reverse.sh` script by allowing it to display the two parameters, still in reverse order, but **on the same line**.

Here is an example of how it would be used;

```
tux@LinuxBox > ./reverse.sh first next
next first
tux@LinuxBox >
```

Please note the following;

- We want a space between the two parameters
- We do not want the next shell prompt to be on the same line than the parameters
- We could get this done by using a single echo statement but we already practiced this so avoid it here
- Instead, still use two echo statements, one to display each parameter, but search the manpage for details on how to use `echo -n`

## Exercise #6 - multigreps.sh - version #1

We are working on a task where we often find ourselves using `grep` to look for the lines of a file containing **three specific words**. Please do not ask what this task is :)

For example, let us say that we are working with a text file named `myfile.txt` that contains the following;

```
This first line only contains an irrelevant sentence.  
This second line is really not much better but contains the word bananas.  
Now, the following line has an interesting question about a movie;  
Which DreamWorks movie features fictive creatures obsessed with bananas  
for one hour and a half?  
That was it, no more interesting lines about DreamWorks in this file...
```

## Exercise #6 - multigreps.sh - version #1

We are looking for lines containing the words "bananas", "DreamWorks", and "movie", then we would get this done by doing something like;

```
grep bananas myfile.txt | grep DreamWorks | grep movie
```

There are better ways to get this done with grep. However, we are going to spare ourselves the redundant syntax by writing a script

Here is an example of how it would be used;

```
tux@LinuxBox > ./multigreps.sh myfile.txt banana DreamWorks movie  
Which DreamWorks movie features fictive creatures obsessed with  
bananas for one hour and a half?  
tux@LinuxBox >
```



## Exercise #6 - multigreps.sh - version #1

How to get it done?

- The first positional parameter must be the name of the file we want to `grep`, possibly with its full path
- The 2nd, 3rd and 4th positional parameters must be the three words that we want to `grep`
- Your script must invoke the `grep` tool as illustrated above

## Exercise #7 - multigreps.sh - version #2

Let us improve our multigreps.sh script. After it is done displaying the results of the `grep`, we want it to now also display the **exit status** of the `grep` pipeline we just executed

Here is an example of how it would be used;

```
tux@LinuxBox > ./multigreps.sh myfile.txt banana DreamWorks movie
Which DreamWorks movie features fictive creatures obsessed with
bananas for one hour and a half?
The search exited with status 0
tux@LinuxBox >
```

Out of curiosity, try using a data file and parameters which yield no results to see what the exit status value is in different scenarios

## Exercise #8 - Using Variables

- Write a bash script that takes 2 integer variables and outputs their sum, product, and difference.
- Name the script `variables.sh`, it would be used like this:

```
tux@LinuxBox> ./variables.sh 1 1  
Sum = 2  
Product = 1  
Difference = 0
```

- Assume that the user will always enter correct arguments. In other words, you do not have to worry about error checking.
- Use `expr`, `let`, and arithmetic expansion to implement three variants of your solution