Topic

Interlude: PA4b

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

Exercise #1 - howmany.sh - version #1

- Let us start with a script which will define a variable containing a small integer of your choice. This value will represent the number of positional parameters the script expects to be passed. We want our script to display a message stating whether the expected number of parameters was passed or not.

- Here is an example of how it would be used;

```
tux@LinBox > ./howmany.sh one two three
Sorry, you passed 3 parameters but I expected 5 parameters

tux@LinBox > ./howmany one two three four five
Master gave Dobby the right number of parameters!
```

Exercise #2 - howmany.sh - version #2

- Let us improve our `howmany.sh` script by enabling it to tell us whether we provided too many or too few parameters.
- Here is an example of how it would be used;

```
tux@LinBox > ./howmany.sh one two three
Too few parameters! You gave me 3 parameters, but I expected 5

tux@LinBox > ./howmany one two three four five six seven eight
Too many parameters! You gave me 8 parameters, but I expected 5

tux@LinBox > ./howmany one two three four five
Master gave Dobby the right number of parameters!
```

- Make sure that the part of the two messages telling you how many parameters were given vs. how many were expected is not typed twice in your script.

Exercise #3 - paramsloop-v1.sh

- Write a Bash script that displays all of its parameters by iterating over them with a for loop after expanding the list of parameters first.
- Here is an example of how it would be used;

```
tux@LinBox > ./paramsloop.sh one two three
one
two
three
```

- You will use different ways to expand the list of parameters;

```
  $*        $@        "$*"       "$@"
```

- What are the differences you observe? Which of the above syntaxes are equivalent?

Exercise #4 - paramsloop-v2.sh

- Write and test a Bash script which will displays all of its parameters by iterating over them by using a while loop with shift.

- Here is an example of how it would be used;

```
tux@LinuxBox > ./paramsloop.sh one two three
one
two
three
```

Exercise #5 - paramsloop-v3.sh

- Write and test a Bash Script which will displays all of its parameters by iterating over them by using a until loop with shift.

- Here is an example of how it would be used;

```
tux@LinuxBox > ./paramsloop.sh one two three
one
two
three
```

Exercise #6 - whatisthis.sh - version #1

- Write and test a Bash Script which, given a single parameter, a file name, will display one of the following messages
  - "Java Source File" if the file suffix is .java
  - "C Source File" for .c
  - "C++ Source File" for .cpp
  - "Ada Source File" for .ada
  - "Python Source File" for .py
  - If the file doesn't have one of these suffix, your script will simply display "Unknown Source File"
- Here is an example of how it would be used;

```
tux@LinuxBox > ./whatisthis.sh something.diff
Unknown File Suffix
tux@LinuxBox > ./whatisthis.sh Something.java
Java Source File
```

Please note:

- Use the CASE statement
- Do not verify that the parameter given to your script is the name of a file that actually exists. For now, just take the name given to us and look at its suffix.
- In order to get the file suffix out of a string containing the whole filename, you might want to read section 10.1 Manipulating Strings of the Advanced Bash Scripting Guide.
- ${ ## }

https://tldp.org/LDP/abs/html/abs-guide.html#MANIPULATINGVARS

Exercise #7 - whatisthis.sh - version #2

- The next version of our whatisthis.sh script will also verify whether the file actually exists and display a message accordingly. In addition, if the file exists, it will display the number of lines it contains
- Here is an example of how it would be used;

```
tux@LinuxBox > ./whatisthis.sh something.diff
File not found, Unknown File Suffix

tux@LinuxBox > ./whatisthis.sh Something.java
File not found, Java Source File

tux@LinuxBox > ./whatisthis.sh Something.java
File exists! Java Source File with 23 lines
```

Exercise #8 - repeat.sh

- We want a script to repeat a given command an arbitrary number of times.
- Here is an example of how it would be used;

```
tux@LinuxBox > ./repeat.sh 5 "echo something"
something
something
something
something
something
```

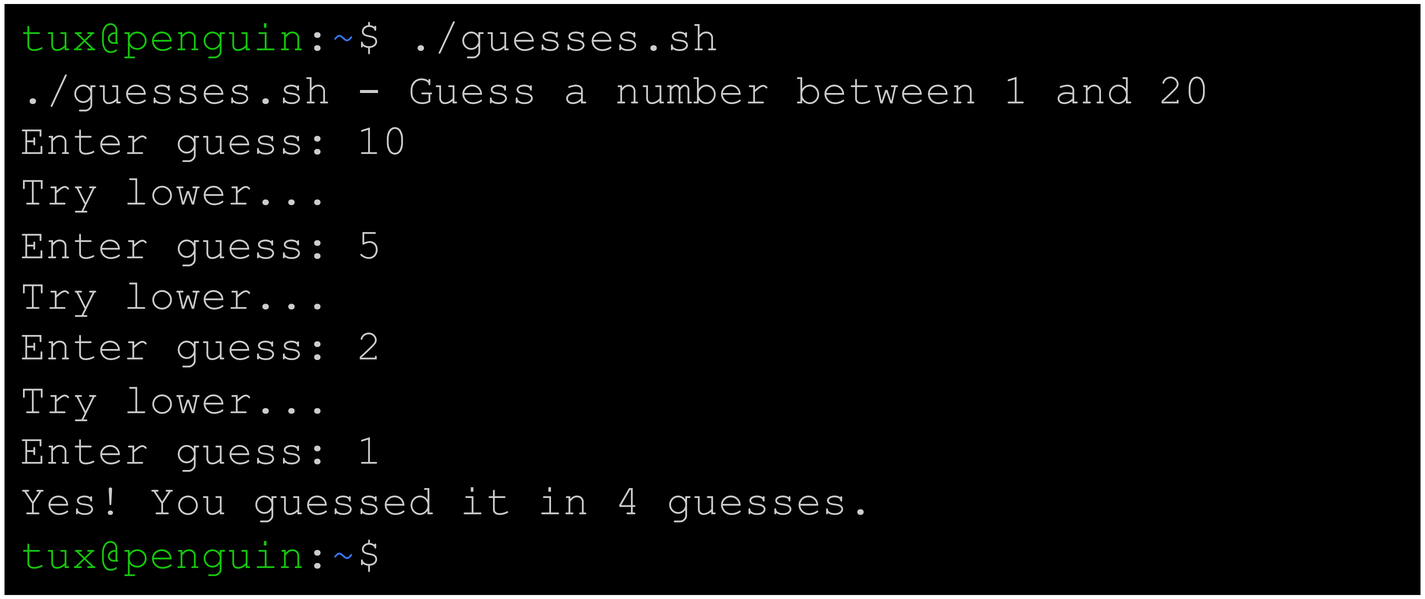
Exercise #9 - factorial.sh

- Let’s compute the factorial of the script’s first argument

```
tux@LinuxBox > ./factorial.sh 5
120
```

Exercise #10 - guess.sh

- Implement a classic game of guess-the-random-number. You will keep track of the number of guesses used before to find the solution
- To read input from the user, use the read keyword as follows: `read INPUT`



```
tux@penguin:~$ ./guesses.sh
./guesses.sh - Guess a number between 1 and 20
Enter guess: 10
Try lower...
Enter guess: 5
Try lower...
Enter guess: 2
Try lower...
Enter guess: 1
Yes! You guessed it in 4 guesses.
tux@penguin:~$
```

Exercise #11 - sumlines.sh

- Given a list of arguments representing text files, let’s compute the total number of lines they contain

```
tux@LinuxBox > ./sumlines.sh one.txt two.txt three.txt four.txt
Total number of lines is 539
```

Exercise #12 - archive.sh

- Given a list of arguments representing text files, let’s make a backup copy of those which contain more lines than the first argument to the script

```
tux@LinuxBox > ./archive.sh 23 one.txt two.txt three.txt
four.txt
one.txt has 15 lines, no backup necessary
two.txt has 25 lines, backed up to two.txt-2022-11-22-14:16.BAK
three.txt has 359 lines, backup file already exists
four.txt was not found
```

- Please note that:
  - If the file has less than the number of lines specified, we do nothing to it
  - If it has more lines than that, we check if a backup copy has been made today. If so, we display a message but leave it unchanged
  - If not backup copy has been made today, we make one using the date and time in the name of the backup copy as illustrated above