Topic

Interlude: PA4b - Solutions

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!
```

```
#!/bin/bash
EXPECTED=5
if [[ $# == 5 ]]
then
    echo 'Master gave Dobby the right number of parameters!'
else
    echo "Sorry, you passed $# parameters but I expected
$EXPECTED parameters"
fi
```

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.

```
#!/bin/bash
EXPECTED=5
if [[ $# == $EXPECTED ]]
then
    echo 'Master gave Dobby the right number of parameters!'
else
    if [[ $# < $EXPECTED ]]
    then
        WORD="few“
    else
        WORD="many“
    fi
    MSG="Too $WORD parameters!"
    MSG="$MSG You gave me $# parameters but I expected $EXPECTED parameters"
    echo $MSG
fi
```

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?

```
#!/bin/bash

echo 'Just using $*'
echo -e "\t" $*

echo 'Just using $@'
echo -e "\t" $@

echo 'For loop with $*'
for p in $*
do
    echo -e "\tparameter = $p"
done
```

```
echo 'For loop with $@'
for p in $@
do
    echo -e "\tparameter = $p"
done

echo "no differences in the above..."
echo "now we quote like so;"

echo 'For loop with "$*"'
for p in "$*"
do
    echo -e "\tparameter = $p"
done
# "$*" expands as one entry


echo 'For loop with "$@"'
for p in "$@"
do
    echo -e "\tparameter = $p"
done
```

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
```

```bash
#!/bin/bash

echo 'While loop with shift'

while [[ $# -gt 0 ]]
do
    echo -e "\tparameter = $1"
    shift
done
```

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
```

```bash
#!/bin/bash

echo 'until loop with shift'
until [[ $# -eq 0 ]]
do
    echo -e "\tparameter = $1"
    shift
done
```

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.
- ${ ## }

```
#!/bin/bash

filename=$(basename "$1")
extension="${filename##*.}"
filename="${filename%.*}"

case $extension in
    "java") lang="Java"
    ;;
    "c")    lang="C"
    ;;
    "cpp")  lang="C++"
    ;;
    "ada")  lang="Ada"
    ;;
    "py")   lang="Python"
    ;;
    *)  lang="Unknown"
    ;;
esac

echo "$lang Source File"
```

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
```

```
#!/bin/bash

filename=$(basename "$1")
extension="${filename##*.}"
filename="${filename%.*}"
```

```
case $extension in
    "java") lang="Java"
    ;;
    "c")    lang="C"
    ;;
    "cpp")  lang="C++"
    ;;
    "ada")  lang="Ada"
    ;;
    "py")   lang="Python"
    ;;
    *)  lang="Unknown"
    ;;
esac

MSG="$lang Source File"

if [[ -e $1 ]]
then
    LINES=$(wc -l $1)
    MSG="File exists! $MSG with $LINES lines"
else
    MSG="File not found, $MSG"
fi
echo $MSG
```

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
```

```
#!/bin/bash
n=$1
msg=$2
while [[ $n -gt 0 ]]
do
        eval $msg
        (( n-- ))
done
```

Exercise #9 - factorial.sh

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

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

```
#!/bin/bash

result=1
number=$1
i=1
while [[ $i -le $number ]]
do
        result=$(( $result*$i ))
        (( i++ ))
done
echo result is $result
```

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

```
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:~$
```

```
#!/bin/bash
NUMGUESS=0

echo "$0 - Guess a number between 1 and 20"

(( secret = RANDOM % 20 + 1 ))

while [[ guess -ne secret ]]
do
(( NUMGUESS = NUMGUESS + 1 ))
read -p "Enter guess: " guess

if (( guess < $secret )); then
echo "Try higher..."
elif (( $guess > $secret )); then
echo "Try lower..."
fi
done

printf "Yes! You guessed it in $NUMGUESS guesses.\n"
```

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
```

```
#!/bin/bash

total=0
for name in "$@"
do
      nlines=$(wc -l $name | cut -d ' ' -f1)
      echo $name contains $nlines lines
      (( total += nlines ))
done
echo Total number of lines is $total
```

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

```
#!/bin/bash

DATETAG=$(date +%Y-%m-%d-%H:%M)
MAXLINES=$1
shift

for FILENAME in "$@"
do
        echo -n $FILENAME
        if [[ -e $FILENAME ]]
        then
                nlines=$(wc -l $FILENAME | cut -d ' ' -f1)
                echo -n " has $nlines lines"
                if [[ $nlines -gt $MAXLINES ]]
                then
                        BACKUPFILENAME="$FILENAME-$DATETAG.BAK"
                        if [[ -e $BACKUPFILENAME ]]
                        then
                                echo -n ", backup file already exists"
                        else
                                cp $FILENAME $BACKUPFILENAME
                                echo -n ", backed up to $BACKUPFILENAME"
                        fi
                else
                        echo -n ", no backup necessary"
                fi
        else
                echo -n " was not found"
        fi
        echo
done
```