BASH cheat sheet - Level 3

Control Structure

Conditional statements

A) The *if* statement:

```
if [[ condition1 ]];then
    command1
elif [[ condition2 ]];then
    command2
else
    command3
fi
```

/ !\ The **spaces** are important in that syntax

Perform *command1* if *condition1* is true, elseif the *condition2* is true the *command2* is performed, else it's the *command3* that will be performed.

File tests:

-f *file*	True if *file* file exists
-d *dir*	True if *dir* dir exists
-z string	True if string is empty.
-n string	True if string is non-empty.

file1* –nt *file2
True if *file1* has been changed more recently than *file2*, or if *file1* exists and *file2* does not.

file1* –ot *file2
True if *file1* is older than *file2*, or if *file2* exits and *file1* does not.

String comparison operators:

string1* == *string2	Compare strings equality.
string1* != *string2	Compare strings inequality.

Arithmetic comparison operators:

Generally **numeric** comparisons on double square brackets are obsolete, however you may still used **-eq**, **-ne**, **-lt**, **-le**, **-gt**, or **-ge**, meaning **eq**ual, **n**ot **e**qual, **l**ess **t**han, **l**ess than or **e**qual, **g**reater **t**han, and **g**reater **t**han or **e**qual, respectively.

/ !\ For arithmetic values it's highly recommended to use the syntax ((condition)). In that case, the following operators must be used : **==** , **!=**, **<** , **<=**, **>** , **>=**

Pattern matching operator:

string* =~ *regularExpression
True if *string* match the pattern of the regular expresion.

Logical operators :

if ! [[*condition*]]
! - NOT - Negate the truth. Evaluate as true only if *condition* is false.

if [[*condition1*]] && [[*condition2*]]
&& - AND – Check if both conditions are true.

if [[*condition1*]] || [[*condition2*]]
|| - OR – Check if one of the conditions is true.

B) The *case* statement:

```
case $variable in
   pattern1)
      commands1
      ;;
   pattern2|pattern3|pattern4)
      commands2
      ;;
   patternN)
      commands3
      ;;
   *)
      commands4
      ;;
esac
```

It allows to check a value multiple times. If the *$variable* match the *pattern1*, the *commands1* are executed. If it matches none of them, the *commands4* are executed.

The loops

A) The *for* loop:

```
for i in element1 element2 element3 ; do
    command
done
```

Repeat the command by assigning list's elements to the variable *i*. The **list** can be implicit (e.g ***.txt** that iterates over all the txt files.)

```
for (( i=1 ; i<=10 ; i++ )) ; do
    command
done
```

This is a C-style for loop that iterates over the integers (here from 1 to 10 namely 10 times).

```
for i in {1..10}; do
    command
done
```

This syntax allows to iterate a selected number of time (here from 1 to 10 namely 10 times).

```
for i in ${!array[@]} ; do
  echo "key :" $i
  echo "value:" ${array[$i]}
done
```

Iterate over an associative array.

B) The *while* loop:

The while loop continue until the condition is false.

```
i=0
while (( $i <= 10 )); do
    command
    ((i++))
done
```

Iterate until ***$i*** is superior to 10, namely 10 times.

File reading

```
for line in $(cat file.txt); do
        command
 done
```
Read the file *file* line by line and execute the command. **/ !** Here the line is defined by the IFS variable (see section's end). Set it to '\n' to obtain the behavior expected.

```
while read line ;do
        command
done < file
```
Read the file *file* line by line and execute the command.

IFS=$'\n' Set the **I**nternal **F**ield **S**eparator (IFS) variable to '\n'. By default its value is ' \t\n' (space, tab and newline).

Arrays and Hashes

A) Indexed array :

array*=()** or **declare –a *array
Declare an indexed array and initialize it to be empty. In the second case an existing array is not initialized.

***array*=(Anna Par Ulla)**
The array *array* is initialized with three values.

array*[*N*]=*value
Set the element *N* of the array *array* to *value*

array+=(value1 value2 value3)
Append the array with three values.

${*array*[*N*]}
Expand the element referenced by the index *N* from *array*.

${#*array*[*N*]}
Size (string length) of the value referenced by the index *N* in *array*

${#*array*[@]}
Size (number of elements) of *array*.

${ !*array*[@]}
Expand each *array* index key as a separate argument.

${array[@]}
Expand all the values stored in *array*.

unset -v *array*[*N*]
Destroy the *array* element at index *N*.

unset -v *array*
Destroy the complete *array*.

B) Associative array / !\ From Bash 4 / !\:

declare –A *array*
Declare an associative array *array*.

***array* =([*string1*]=*value1* [*string2*]=*value2*)**
Assign two values in an associative *array*. You must declare the associative array first.

Omitting the append command (+=), all other commands are similar to those of indexed array. Except that the index key is no more a numerical value *N* but a **STRING**.

Programming in bash

#!/bin/bash
Written at the top of your script, it allows to define the shell to use. Option can be added as **–x** for debbug.

sleep 60 Suspend execution for an interval of 60 seconds.

exit Quit the program.

./*script*.sh Execute the script *script* (The file's executing right is needed).

# comment This is a comment.

A) Arguments

./*script*.sh *arg1 arg2*
With that command, the script receive values script.sh in **$0**, ***arg1*** in **$1**, ***arg2*** in **$2**.

$# The number of arguments.

$@ Array of arguments.

B) Functions

/ !\ A function must be defined before to call it.

```
function hello {
        command
        }
```
Definition of a function called *hello*. The *command* will be performed when the function is called.

hello *arg1 arg2*
Call the function named *hello* with 2 arguments.

/ !\ Within functions, arguments are treated in the same manner as arguments given to a script.

C) User interface

read *variable*
Wait for an user input and save the it in *variable*.

```
options="opt1 opt2"
select opt in $options; do
        if [ "$opt" = "opt1" ]; then
                command1
        elif [ "$opt" = "opt2" ]; then
                command2
        else
                command3
        fi
done
```
This is a text based user-friendly menu. It prompts the user for each 'opt' in $options.