Bash Script Tutorial

Exercise 1 - Familiarization with common Unix/Linux commands and built-in bash commands.

1. Command man shows a description page for the command passed as argument. Use it to see the role of commands. It follows a list of common ones: ls, mkdir, rmdir, pwd, rm, mv, cat, echo, less, head, tail, find, cp, diff, wc, sort, grep, sed, tr, cut, paste, chmod, stat, history, ...
2. Commands usually have switches/options that allow you to change their default behaviour. Explore some of the ls command options: -l, -a, -lh, -R ...
3. In general, commands correspond to executable programs, but there are some that are built-in functions of the command interpreter. Command cd is one of them. To see a list of such commands, execute the built-in command help. To see a description of a given built-in command, execute help followed by its name, as, for instance, help cd.

Exercise 2 - Redirecting input and output.

Many commands receive input from the standard input (usually the keyboard) and deliver output to the standard output (usually the terminal display window). Error messages are usually and preferably sent to the standard error (also usually the terminal display window). However, it is possible to change this default behaviour.

1. Operators > and >> redirect the standard output to a given file. The following sequence of commands illustrates their use.

```
echo ola                # message "ola" is sent to the terminal display window
echo ola > z            # message "ola" is sent to file "z", content of "z" is replaced
cat z
echo ola >> z           # message "ola" is appended to the end of file "z"
cat z
```

2. Operators 2> and 2>> redirects the standard error to a given file. The following sequence of commands illustrates its use.

```
rm -f zzz               # to guarantee file "zzz" does not exist
cat zzz                 # an error message is sent to the terminal display window
cat zzz 2> z            # the error message is sent to file "z";
cat z
cat zzz 2>> z           # the error message is appended to the end of file "z"
cat z
cat zzz 2> z            # the error message was redirected. Why?
cat z
cat zzz > z             # the error message is sent to the terminal display window. Why?
```

3. Operators 2>&1 and >&2 (or 1>&2) redirect the standard error to the standard output and the standard output to the standard error, respectively. The following sequence of commands illustrates their use.

```
rm -f zzz                    # to guarantee zzz does not exist
cat zzz > err
cat zzz > err 2>&1
cat err
cat /etc/passwd 2> z
cat z
cat /etc/passwd 2> z >&2
cat z
```

Exercise 3 - Using special characters.

1. Analysing the result of the execution of the following sequence of commands, try to understand the meaning of characters * and ?

```
mkdir dir1
cd dir1
touch a a1 a2 a3 a11 b b1 b11        # create some files
ls
ls a*
ls a?
ls *
rm –f a* b*                          # delete files
```

2. Analysing the result of the execution of the following sequence of commands, try to understand the meaning of characters [and]

```
touch a a1 a2 a3 a11 b b1 b11 c c11    # create some files
ls
ls [ac]
ls [a-c]
ls [a-c]?
ls [ab]*
rm –f a* b* c*                         # delete files
```

3. Character \\ can be used to disable the special meaning of the character following it. The following sequence of commands illustrates its use.

```
touch a1 a2 a3 a4 a22        # create some files
echo a*
echo a\*
```

```
echo a?
echo a\?
echo a\[
echo a\\
rm –f a*
```

4. Characters ‘ and " can be used to disable the special meaning of a sequence of characters. The following sequence of commands illustrates their use.

```
touch a1 a2 a3 a4 a22        # create some files
echo a*
echo "a*"
echo ’a*’
rm –f a*                     # delete files
```

Exercise 4 - Pipes

Operator | redirects the standard output of a command to the standard input of the next, an operation known as piping of commands. The following sequence of commands illustrates its use.

```
cp /etc/passwd f             # copy /etc/passwd to file f
cat f | wc -l                # print the number of lines
head f -n +5 | sort          # sort and print the 5 first lines
head f -n +5 | tail –n 1     # print the 5th line
cat f | grep login | tail –n 1   # print the last line containing the sequence login
```

Exercise 5 - Declaring and using variables.

1. Variables are supported in bash. The following sequence of commands illustrates their use.

```
x=abc
xx=0123456789
echo $x
echo $xx
echo ${xx}
echo ${x}x
```

2. There are functions to manipulate the value of variables. The following sequence of commands illustrates the use of the substring and substitute functions.

```
x=0123456789
echo ${x:2:4}
echo ${x/123/ccc}
```

Exercise 6 - Declaring and using functions.

1. Functions are supported in bash. The following sequence of commands illustrates the declaration and use of a function.

```
# the following code declares function x
```

```
x() {
    ls –l
}
# the following code uses the function x previously defined
x
x | wc –l
```

2. Functions can accept arguments. Variables $1, $2, ..., $*, $@ and $# can be used to access them.

```
y() {
    echo $#             # the number of arguments
    echo $1             # the first argument
    echo $2             # the second argument
    echo $*             # the list of all arguments
    echo $@             # idem
    echo "$*"           # idem
   echo "$@"           # idem
}
y a bb ccc dddd eeeee
y a "b b" ccc "dd dd" eeeee
```

Exercise 7 - Bash script as a programming language.

Bash script supports additional programming functionalities, such as:

- conditional statements: if then else
- cycles: for, while, until
- arrays
- binary operators
- etc.

Find a tutorial and try it yourself.