UNIX 101

or "How to feel like a true hack3r"

Subject - shell scripting

Saturday 4th December, 2021

Contents

1 Foreword

1.1 Notions seen in the tutorials

Welcome to this subject! If you have finished the first tutorial and the one on git (and I hope you do, if you have not yet, please finish them now), you should know:

- What the syntax of a command is
- A few useful commands
- How to navigate in a filesystem
- How to create, edit and remove files
- What is a packet manager and how to use one
- **How to create executable scripts**
- **The basics of git**

This is just enough to begin working on scripts and handing them for review.

1.2 Objectives

The goal of this subject is to make you use the notions seen in the tutorial through coding exercises. Additionally, we will have a look over some aspects of the shell command language. That will allow you to write more advanced scripts.

There are new notions you are going to learn and a handful of exercises. Start working early!

2 Setup

Before beginning, make sure you:

- finished tutorial-1 and thus know how to write scripts
- finished tutorial-git and thus know how to regularily make commits and push your work
- have a Github account

Before starting anything, create a git repository named `shell-scripting`.

3 Instructions

You have to create one directory per exercise.

For each exercise, you have to write a script and provide a `README` file. The script should solve the exercise. The `README` file should be a text file that describes how you solved the exercise and the problems you encountered. In the scenario where you cannot have a good enough script, leave a `README` file with the solutions you tried to implement. Who knows, you might save some points :)

Also, you can provide a `tests` directory. It is a good practice to write automatic tests for your code. As we did not study that on tutorials, no point will be taken off if there are no tests. However, points can be granted if you provide good tests along with your scripts.

As it is a good practice to perform git commits regularily, you are asked to do it often. **If you do not commit your changes between each level, you will loose points on your final grade**. Also, remember that you can push your commits to your github repository at any time.

4 Evaluation

4.1 Instructions

You are asked to constituate a team of 2 persons. You will be graded as a team on:

- An oral presentation of your work (no need to prepare slides, we will discuss around your code)
- General UNIX knowledge questions

You have to push your work to your git repository before Wednesday 15th December, 2021, 7:00 AM.

The architecture of your repository should follow the following format:

```
$pwd
/home/nicolas/shell-scripting
$tree --charset=ascii
.
|-- exercise-1
|   |-- ls_on_steroids.sh
|   |-- README
|   `-- tests
|-- exercise-2
|   |-- ls_ordered.sh
|   |-- README
|   `-- tests
|-- exercise-3
|   |-- ls_ordered_with_arguments.sh
|   |-- README
|   `-- tests
|-- exercise-4
|   |-- README
|   |-- salutation.sh
|   `-- tests
|-- exercise-5
```

```
|   |-- README
|   |-- salutation_advanced.sh
|   `-- tests
|-- exercise-6
|   |-- my_ansible.sh
|   |-- README
|   `-- tests
|-- exercise-7
|   |-- my_ansible_advanced.sh
|   |-- README
|   `-- tests
|-- exercise-8
|   |-- guessing_game.sh
|   |-- README
|   `-- tests
`-- README.md
```

4.2 A word on cheating

You have to remember that you should be studying for your own good. Cheating will not bring you any good in the long term; it is fine not to be able to finish every exercise of the subject, your main goal is to train and learn things.

Any form of cheating will immediately bring your grade down to 0. Additionally, your main teacher will be taken notice of that.

A particular attention will be given to your `README` files.

Note: Changing the name of some variables will of course not trick the anti-cheat engine :)

5 Exercises

5.1 ls_on_steroids.sh

5.1.1 Goal

Script name: `ls_on_steroids.sh`

Write a script that recursively lists the content of the current directory. It should display everything in long-listing format, show hidden files and directories and display the size of files and directoroes in a human readable format.

5.1.2 Example

```
$ls
ls_on_steroids.sh*
$head -c 100 /dev/urandom > file.txt
$touch empty_file.txt
$mkdir dir_a
$mkdir dir_b
$head -c 10000 /dev/urandom > dir_b/random_file.txt
$./ls_on_steroids.sh
.:
total 52K
drwxrwxr-x  4 nicolas nicolas 4,0K déc.  12 01:20 .
drwxrwxrwt 27 root    root     28K déc.  12 01:20 ..
drwxrwxr-x  2 nicolas nicolas 4,0K déc.  12 01:20 dir_a
drwxrwxr-x  2 nicolas nicolas 4,0K déc.  12 01:21 dir_b
-rw-rw-r--  1 nicolas nicolas    0 déc.  12 01:20 empty_file.txt
-rw-rw-r--  1 nicolas nicolas  100 déc.  12 01:20 file.txt
-rwxrwxr-x  1 nicolas nicolas   23 déc.  12 01:19 ls_on_steroids.sh

./dir_a:
total 8,0K
drwxrwxr-x 2 nicolas nicolas 4,0K déc.  12 01:20 .
drwxrwxr-x 4 nicolas nicolas 4,0K déc.  12 01:20 ..

./dir_b:
total 20K
drwxrwxr-x 2 nicolas nicolas 4,0K déc.  12 01:21 .
drwxrwxr-x 4 nicolas nicolas 4,0K déc.  12 01:20 ..
-rw-rw-r-- 1 nicolas nicolas 9,8K déc.  12 01:21 random_file.txt
```

5.1.3 Solution

Now this part will not be there for other exercises but this is a guide on how to solve this exercise.

The script should be located in `exercise-1/ls_on_steroids.sh`. The `README` should be in the same directory.

```
$pwd
/home/nicolas/shell-scripting/exercise-1
```

```
$ls
ls_on_steroids.sh*  README  tests/
$cat ls_on_steroids.sh
#! /bin/bash

ls -Rahl
$cat README
No real difficulty. I checked ls's manual and found there all of the
   options that I needed to use.

-R is the recursive option
-l is the long-listing format
-h is for human readable file sizes
-a is to display hidden files
```

5.2 ls_ordered.sh

5.2.1 Goal

Script name: `ls_ordered.sh`

Write a script that lists the files and directories in the current directory. It should display everything in long-listing format, biggest files and directories first.

5.2.2 Example

```
$ls
ls_ordered.sh*
$touch empty
$head -c 100 /dev/urandom > small_file
$mkdir mydir
$head -c 10000 /dev/urandom > medium_file
$head -c 1000000 /dev/urandom > big_file
$./ls_ordered.sh
total 1004
-rw-rw-r-- 1 nicolas nicolas 1000000 déc.  12 01:48 big_file
-rw-rw-r-- 1 nicolas nicolas   10000 déc.  12 01:48 medium_file
drwxrwxr-x 2 nicolas nicolas    4096 déc.  12 01:47 mydir
-rw-rw-r-- 1 nicolas nicolas     100 déc.  12 01:47 small_file
-rwxrwxr-x 1 nicolas nicolas      21 déc.  12 01:36 ls_ordered.sh
```

```
-rw-rw-r-- 1 nicolas nicolas        0 déc.  12 01:46 empty
```

5.3 ls_ordered_with_arguments.sh

5.3.1 Goal

Script name: `ls_ordered_with_arguments.sh`
New commands: `echo`, `exit`
New notions: Shell variables, arguments retrieval

Write a script that does the same thing as in the previous section. However, when invoked with an argument, it should try to perform the listing on the argument path.

If there is more than one argument, it should only consider the first one.

If the argument is the path to a file, it should display the long-listing of that file.

If the argument is the path to a directory that does not exist, it should print an error message and return an exit code 2

5.3.2 New notions: arguments retrieval

To complete this exercise, you need to know a few more things.

UNIX programs are launched through the command line. As we have seen with the many commands we learnt in the tutorials, one can pass command line arguments to change their behavior.

So how does a program retrieve the user's arguments? Well it is system-dependant but usually, programs can access them using the `argv` variable. In Shell, variables are accessed with `$` and arguments are the variables `0`, `1`, `2`, `3`, etc. The variable 0 is always set: it is the name of the program which runs. The other variables are only set if the program is launched with arguments. These arguments are called **positional parameters**.

Let us check out an example:

```
$cat dummy_script.sh
#! /bin/bash

echo The name of the program is $0
echo The first argument is $1
echo The second argument is $2
echo The third argument is $3
echo The fourth argument is $4
```

```
$./dummy_script.sh uno dos tres
The name of the program is ./dummy_script.sh
The first argument is uno
The second argument is dos
The third argument is tres
The fourth argument is
```

5.3.3 New notions: exit

The command `exit` allows one to exit the current program and set a custom return code. Entering `exit` in an interactive shell closes the shell. Using it in a script shut downs the script. Shell scripts use that command to set their return code.

As we have seen just before, in a shell, variables can be accessed with `$`. Numeric variables are not the only special variables. As such, `$?` gives the return code of the latest command executed and `$#` gives the number of positional parameters.

Let us see an example:

```
$cat exit_script.sh
#! /bin/bash

echo Hi! Let us imagine something goes wrong and exit with a return
   code of 5
exit 5
$./exit_script.sh
Hi! Let us imagine something goes wrong and exit with a return code
   of 5
$echo $?
5 # Return code of ./exit_script.sh is 5
$pwd
/home/nicolas
$ echo $?
0 # Return code of pwd is 0
```

5.3.4 Example

```
$./ls_ordered_with_arguments.sh
total 1004
```

```
-rw-rw-r-- 1 nicolas nicolas 1000000 déc.  12 01:48 big_file
-rw-rw-r-- 1 nicolas nicolas   10000 déc.  12 01:48 empty_file
drwxrwxr-x 2 nicolas nicolas    4096 déc.  12 02:41 mydir
-rw-rw-r-- 1 nicolas nicolas     100 déc.  12 01:47 small_file
-rwxrwxr-x 1 nicolas nicolas      24 déc.  12 02:41
   ls_ordered_with_arguments.sh
-rw-rw-r-- 1 nicolas nicolas       0 déc.  12 01:46 empty
$./ls_ordered_with_arguments.sh mydir
total 4
-rw-rw-r-- 1 nicolas nicolas 5 déc.  12 02:41 file_with_text
-rw-rw-r-- 1 nicolas nicolas 0 déc.  12 02:41 hello
```

5.4 salutation.sh

5.4.1 Goal

Script name: `salutation.sh`
New notions: Conditions

Write a script that, given two arguments, prints `Hello ` , followed by the first argument, a whitespace, the second argument and `, have a nice day!`.

If there is no error, the return code should be 0. If there is an error, the return code should be 1 and the following line should be displayed: `Usage: ./salutation.sh firstname lastname`.

The error cases are:

- If no argument is supplied
- If only one argument is supplied
- If more than two arguments are supplied

5.4.2 New notions: conditions

Every programming language relies on conditional statements. The shell command language being a script language, it also has control structures as conditions and loops.

For this exercise, we will learn to use the simple `test` condition and the `if` control structure.

Consider the following pseudo-code

```
if CONDITION; then
        DO SOMETHING
else
```

```
        DO SOMETHING ELSE
fi
```

Here, if CONDITION is evaluated to `0` (True), the flow of execution will continue at `DO SOMETHING` and then "jump to" what is after `fi`. If the condition is evaluated to something else (False), the flow of execution will continue at `DO SOMETHING ELSE` and then "jump to" what is after `fi`[1]. Note that the `else` statement is optional.

`CONDITION` can be any shell command since every shell command has an exit code. However, we can use the builtin `test` which allows one to test simple conditions.

Let us see an example:

```
$cat simple_test.sh
#! /bin/bash

if test 10 -eq 100; then
        echo 10 is equal to 100? Wut?
else
        echo 10 is indeed not equal to 100, seems legit
fi
$./simple_test.sh
10 is indeed not equal to 100, seems legit
```

Here is a summary of the most common `test` options:

`int1 -eq int2`	Test that numbers int1 and int2 are the same
`int1 -lt int2`	Test that number int1 is **l**ower **t**han int2
`int1 -ge int2`	Test that number int1 is **g**reater or**e**qual to int2
`-e file`	Test that file `file` exists
`-d file`	Test that file `file` is a directory
`-z str`	Test that string `str` is empty or nonexistent
`str1 = str2`	Test that string `str1` is the same as `str2`
`str1 != str2`	Test that string `str1` is not the same as `str2`

Table 1: Most common `test` options

5.4.3 Example

[1] This is a **if** condition that is very common in programming

```
$./salutation.sh John Doe
Hello John Doe, have a nice day!
$echo $?
0
$./salutation.sh
Usage: ./salutation.sh firstname lastname
$echo $?
1
$./salutation.sh John Kevin Doe
Usage: ./salutation.sh firstname lastname
$echo $?
1
```

5.5 salutation_advanced.sh

5.5.1 Goal

Script name: `salutation_advanced.sh`

Write a script that behaves the same as the previous one. It however has a new error case: if the given firstname and lastname are the same, simply display `Firstname and lastname cannot be the same!` and exit with an error code 2.

5.5.2 Example

```
$./salutation_advanced.sh John Doe
Hello John Doe, have a nice day!
$echo $?
0
$./salutation_advanced.sh
Usage: ./salutation_advanced.sh firstname lastname
$echo $?
1
$./salutation_advanced.sh John Kevin Doe
Usage: ./salutation_advanced.sh firstname lastname
$echo $?
1
./salutation_advanced.sh John John
Firstname and lastname cannot be the same!
$echo $?
```

```
2
```

5.6 my_ansible.sh

5.6.1 Goal

Script name: `my_ansible.sh`

Write a script that creates the following files and directory architecture[2]:

```
.
|-- documents
|   |-- gamez
|   |   `-- csgo.exe
|   |-- images
|   `-- work
|       |-- code
|       `-- plannings
|           |-- april.xlsx
|           |-- february.xlsx
|           |-- january.xlsx
|           `-- march.xlsx
|-- meeting_notes.txt
```

If some files already exist, neither their content nor their last modification date should be modified.

Except if permission issues arise, the script should not print anything and always have a 0 exit code.

Bonus: Write the smallest script possible. The smallest one will be given extra points.

Note: Do not commit the files you generate. They should not be relevant.

5.6.2 Example

```
$ls
my_ansible.sh*
$./my_ansible.sh
$echo $?
0
$ls -R
```

[2] `images` is an empty directory

```
.:
documents/  meeting_notes.txt  my_ansible.sh*

./documents:
gamez/  images/  work/

./documents/gamez:
csgo.exe

./documents/images:

./documents/work:
code/  plannings/

./documents/work/code:

./documents/work/plannings:
april.xlsx  february.xlsx  january.xlsx  march.xlsx
```

5.7 my_ansible_advanced.sh

5.7.1 Goal

Script name: `my_ansible_advanced.sh`

Write a script similar to the one from the previous exercise. The behavior of the script should however be dependant on positional parameters:

- If no positional parameter is given, keep the behavior the same as for my_ansible.sh
- If a positional parameter is given, use it to create an intermediary directory in `documents`
- If more than one positional parameter is given, display `Usage: ./my_ansible_advanced.sh [username]` and exit with a code 1
- If a positional parameter is given but the directory `documents` does not exist, display `directory documents must exist` and exit with a code 2

Bonus: Prevent side-effects from malicious usages of that script (e.g. prevent what happens if the user inputs "`..`" as the positional argument)

5.7.2 Example

```
$ls
```

```
my_ansible_advanced.sh*
$./my_ansible_advanced.sh john
directory documents must exist
$echo $?
2
$ls
my_ansible_advanced.sh*
$./my_ansible_advanced.sh
$tree --charset=ascii
.
|-- documents
|   |-- gamez
|   |   `-- csgo.exe
|   |-- images
|   `-- work
|       |-- code
|       `-- plannings
|           |-- april.xlsx
|           |-- february.xlsx
|           |-- january.xlsx
|           `-- march.xlsx
|-- meeting_notes.txt
`-- my_ansible_advanced.sh

6 directories, 7 files
$./my_ansible_advanced.sh nicolas
$tree --charset=ascii
.
|-- documents
|   |-- gamez
|   |   `-- csgo.exe
|   |-- images
|   |-- nicolas
|   |   |-- gamez
|   |   |   `-- csgo.exe
|   |   |-- images
|   |   `-- work
|   |       |-- code
|   |       `-- plannings
|   |           |-- april.xlsx
```

```
|   |           |-- february.xlsx
|   |           |-- january.xlsx
|   |           `-- march.xlsx
|   `-- work
|       |-- code
|       `-- plannings
|           |-- april.xlsx
|           |-- february.xlsx
|           |-- january.xlsx
|           `-- march.xlsx
|-- meeting_notes.txt
`-- my_ansible_advanced.sh

12 directories, 12 files
$./my_ansible_advanced.sh nicolas doe
Usage: ./my_ansible_advanced.sh [username]
$echo $?
1
```

5.8 guessing_game.sh

5.8.1 Goal

Script name: guessing_game.sh

Write a script that behaves the same way as the guessing game from tutorial-1.

You are given the following chunk of code to help you. Feel free to copy/paste it to begin your script:

```
#! /bin/bash

echo At any time, hit CTRL+C to exit the script

number_to_guess=$(($RANDOM % 100))

while (( guess != number_to_guess )); do
        read -p "" guess # Asks the user for a number and put it in
           the variable "guess"
        echo Number given by user: $guess

done
```

Bonus: handle badly formatted user input (e.g. do not crash if the user inputs something that is not a number)

5.8.2 Example

```
./guessing_game.sh
I have in mind a number in between 1 and 100, can you find it?
10
The number to guess is higher
70
The number to guess is lower
45
The number to guess is lower
34
The number to guess is lower
23
The number to guess is higher
29
The number to guess is lower
26
You just found the number, it was indeed 26
```