CS:3620 — Spring 2020 — Homework 1

This homework is about using the Linux operating system and its shell. In addition, in **task8** you will have to write some C code. This homework is composed of 9 tasks.

Please, ask general questions about the homework on ICON, so that everyone can benefit from the answers.

General Requirements Submit your homework as a *single* `tar` file. When unpacked, the `tar` file must have the following directory structure (substitute ⟨your_HawkID⟩ with your actual HawkID):

```
⟨your_HawkID⟩
    task1
        script.sh
    task2
        script.sh
    ...
    task7
        script.sh
    task8
        compile.sh
        task8.c
```

All the `.sh` files must have the `executable bit` set. No other file should be present in the archive.

When writing your code, you can ignore *hidden files* and *file links*, since we did not discuss these topics during the lectures.[1] You can assume that your code will be executed using as *current working directory* the corresponding `task1`, ..., `task8` directory. For instance, your script for `task1` will be executed by running:
`cd ⟨your_HawkID⟩/task1/`
`./script.sh`.

[1] For more information about these topics you can read: `https://en.wikipedia.org/wiki/Hidden_file_and_hidden_directory#Unix_and_Unix-like_environments`, `https://en.wikipedia.org/wiki/Symbolic_link#POSIX_and_Unix-like_operating_systems`, and `https://en.wikipedia.org/wiki/Hard_link`.

task1

Write a `bash` script accepting two arguments: ⟨`directory`⟩ and ⟨`extension`⟩. The script must print to stdout the path of all files in ⟨`directory`⟩ (and, recursively, its sub-directories) with a name ending with ⟨`extension`⟩. You can choose to print either relative paths (i.e., relative to the script current working directory) or absolute paths[2].

I suggest using the command `find` or `grep`, or a combination of both.

task2

Write a `bash` script accepting one argument: ⟨`directory`⟩. The script must print to stdout *the content* of any file in ⟨`directory`⟩ (and, recursively, its sub-directories) with an name ending with `.txt`. Assume that any file with a name ending in `.txt` contains only printable characters and use the command `cat` to print its content.

task3

Write a `bash` script accepting one argument: ⟨`directory`⟩. The script must print to stdout two numbers, separated by one space. The first number must be equal to the number of files contained in ⟨`directory`⟩ (and, recursively, its sub-directories). The second number must be equal to the number of directories contained in ⟨`directory`⟩ (and, recursively, its sub-directories), including ⟨`directory`⟩ itself.

I suggest using the command `wc`, together with other commands.

task4

Write a `bash` script accepting two arguments: ⟨`directory`⟩ and ⟨`name_tag`⟩. The script must create a `tar` archive containing, recursively, all files and directories in ⟨`directory`⟩ (including ⟨`directory`⟩ itself). The generated archive must be saved in the current working directory, using the following name:
⟨`name_tag`⟩`_`⟨`current_time`⟩`.tar`
Where ⟨`name_tag`⟩ is the string specified by the user as the second argument

[2] you can convert relative paths to absolute paths by using the command `realpath`

and ⟨current_time⟩ is the current Unix time[3] in seconds (I suggest using the command date, with proper parameters, to retrieve this value).

task5

Write a bash script accepting two arguments: ⟨source_directory⟩ and ⟨destination_directory⟩. The script must copy to ⟨destination_directory⟩ all the executable files (i.e., the files with the user's *executable bit* set) found in ⟨source_directory⟩ (and, recursively, its sub-directories). Assume that ⟨destination_directory⟩ already exists.

I suggest using find (specifying the arguments: -perm -u+x) together with xargs or a loop.

task6

Write a bash script accepting three arguments: ⟨stdout_file⟩, ⟨stderr_file⟩, and ⟨cwd⟩, followed by an arbitrary number of other arguments specifying a subcommand and its arguments. The script will run the specified subcommand with all the specified arguments, redirecting the subcommand's stdout to ⟨stdout_file⟩ and the subcommand's stderr to ⟨stderr_file⟩, and setting the subcommand's current working directory to ⟨cwd⟩.

For instance, if the script is invoked in the following way:

```
./script.sh /tmp/stdout /tmp/stderr /bin ls -l -a
```

the command ls will be run with the arguments -l and -a. When run, ls's current working directory must be /bin, the stdout output must be saved in ⟨stdout_file⟩, and the stderr output must be saved in ⟨stderr_file⟩. The script must not change its own current working directory. The script should print to stdout the *exit code* of the executed subcommand. Nothing else should be print to stdout or stderr.

To refer to all the arguments specified by the user after the argument ⟨cwd⟩, I suggest using the bash array variable "${@:⟨n⟩}" (substituting ⟨n⟩ with a proper number). I suggest using a subshell[4] not to change the script ⟨cwd⟩.

[3] https://en.wikipedia.org/wiki/Unix_time

[4] https://www.tldp.org/LDP/abs/html/subshells.html

task7

Write a `bash` script accepting an arbitrary number of arguments specifying a subcommand and its arguments (similar to what is required for **task6**). The script will run the specified subcommand with all the specified arguments, monitoring its execution using `strace`. For instance, if the script is invoked in the following way:

```
./script.sh ls -l -a
```

it will execute strace in the following way:

```
strace ls -l -a
```

The script must redirect the stderr ouput from `strace` and the executed subcommand properly. Specifically, the script must not print the mentioned stderr output, but process it as specified below.

When the executed subcommand finishes, the script has to print to stdout a list of all the system calls executed by the subcommand (in no particular order, but with no repetitions). For instance, if the script is invoked in the following way:

```
./script.sh uname -m
```

The output should be:

```
x86_64
access
arch_prctl
brk
close
execve
exit_group
fstat
mmap
mprotect
munmap
open
read
uname
write
```

I suggest redirecting stderr to a temporary file. By running the subcommand with `strace`, stderr will contain all the executed syscalls (and their parameters).

You can then process the content of this temporary file to extract just the syscall names, by splitting every line, using the character "`(`" as delimiter.

To do so, I suggest using `awk`[5]. Once you have a list of the executed syscalls, you can use `sort` and `uniq` to remove repetitions.

task8

Write a C program printing to sdout all its arguments (including `argv[0]`) hex-encoded. You must save your code in a file called `task8.c`. You must include a script called `compile.sh` that, when invoked, compiles the code in `task8.c` and generates an executable file called `task8`. Do not include this executable file in your submission `tar` file. You must come up with your own hex-encoding function. Do not just copy it from StackOverflow!

Examples (assuming you are opening your program from `bash`)
If the program is invoked in the following way:
`./task8`
The output should be:
`2e2f7461736b39`

If the program is invoked in the following way:
`./task8 firstargument`
The output should be:
`2e2f7461736b39`
`6669727374617267756d656e74`

If the program is invoked in the following way:
`./task8 'first argument with spaces'`
The output should be:
`2e2f7461736b39`
`666972737420617267756d656e7420776974682073706163657`

If the program is invoked in the following way:
`./task8 hello $'anewline\nishere'`
The output should be:
`2e2f7461736b39`
`68656c6c6f`
`616e65776c696e650a697368657265`

[5] I suggest reading the top answer here: `https://stackoverflow.com/questions/8009664/how-to-split-a-delimited-string-into-an-array-in-awk`