
# Linux Administration Project

## Part (1): Bash scripting:

**1. We would like to create a Bash script called "userdef" that creates a new user & new group, and assigns this user to this group. This script must:**

a) Take as arguments (username, userpass, groupname) which we will use in the script (Ex: ./userdef mostafa mos123 ASU)
b) Check that these arguments are really passed (i.e. they are not empty).
And if any argument wasn't passed, print a message and exit with failed status
c) Check that the script is run with "sudo" permission. If not, print a message and exit with failure. (**hint**: use the environment variable "USER")
d) Print to the display the arguments entered.
e) Create this new user, with these conditions:
    - Its name should be taken from the argument
    - Create a home directory for this user
    - Assign the default shell as "Bash"
    - Do not create a group with the same name
f) Assign a password for this new user without prompting on the display (i.e. without halting the execution of the script to enter the password); the password should be taken from the argument.
g) Display the user and group information about this user (**hint**: use the "id" command)
h) Create a new group with ID=200 and the name should be taken from the argument
i) Add the new user to this new group.
j) Again, display the user and group information about this user (to see the changes we made).
k) Modify the user in order to:
    - Make its UID=1600
    - Make its primary group to be the group we just created
l) Again, display the user and group information about this user (to see the changes we made).

**We want to make this script globally reachable (i.e. you can run from anywhere in the system without indicating its path, so that we could use "userdef" instead of "./userdef").**

m) How can we achieve this (2 methods) ? Write the steps to these 2 methods in a text file.
n) We would like this change to be permanent, how can we do this?
o) We just made this change permanent, but we don't want to wait until another terminal is opened, we would like to make this change NOW, and in this terminal. What could we do?

**2. We would like to create a bash script that creates some directories and files, archive them, copy them to the home directory of the new user (the one we created in the last example) and extract them there. The script must:**

(each point in this question must be solved in one line in the script, except for the conditions & loops of course)

a) Check if the directory we want to create already exists, and if it does, remove it.
b) Create the directory & create 4 files in this directory (main.c, main.h, hello.c, hello.h)
c) Loop on each of these files and write in them "this file is named …." (write the name of the file)
d) Compress this directory into a tar file
e) "We want to change directory "cd" into the home directory of the new user, can we ??" if we can't, solve this problem.
f) Copy this tar file to the new home directory, go there and extract it. (each step must execute if the previous succeeds).

**After running this script, do the following:**

(each point in this question must be solved in one command, write these commands in a text file that will be delivered with the 2 scripts)

g) Switch to the newly created user
h) Display all the files inside the extracted folder recursively and with long listing format
i) Change to owner of this directory recursively to be the new user
j) Again, display all the files inside the extracted folder recursively and with long listing format
k) Search for the word "file" in the extracted directory recursively
l) Remove all the files ending with ".c"

## Part (2): Makefiles & CMake:

### Consider this project hierarchy:

- We have 2 modules for cryptographic algorithms (Caesar cipher and XOR cipher)
- Each module has its own directory and generates its own library.
- We have an application that uses these modules
- All generated object files are in “gen”
- The 2 generated libraries are in “libs”
- The 2 libraries and the object file from the app are linked together to produce the executable.

**1. You should write the 3 Makefiles (Makefile, caeser_cipher.mk, xor_cipher.mk) so that:**

- “out”, “gen”, “libs” directories are created when running “make all” and are deleted when running “make clean”
- The final executable is called “output”
- Caesar cipher module should generate a static lib
- XOR cipher should generated a dynamic lib

**2. You should re-do this part with CMake, by writing 3 CMakeLists.txt files in the same place of the 3 Makefiles**

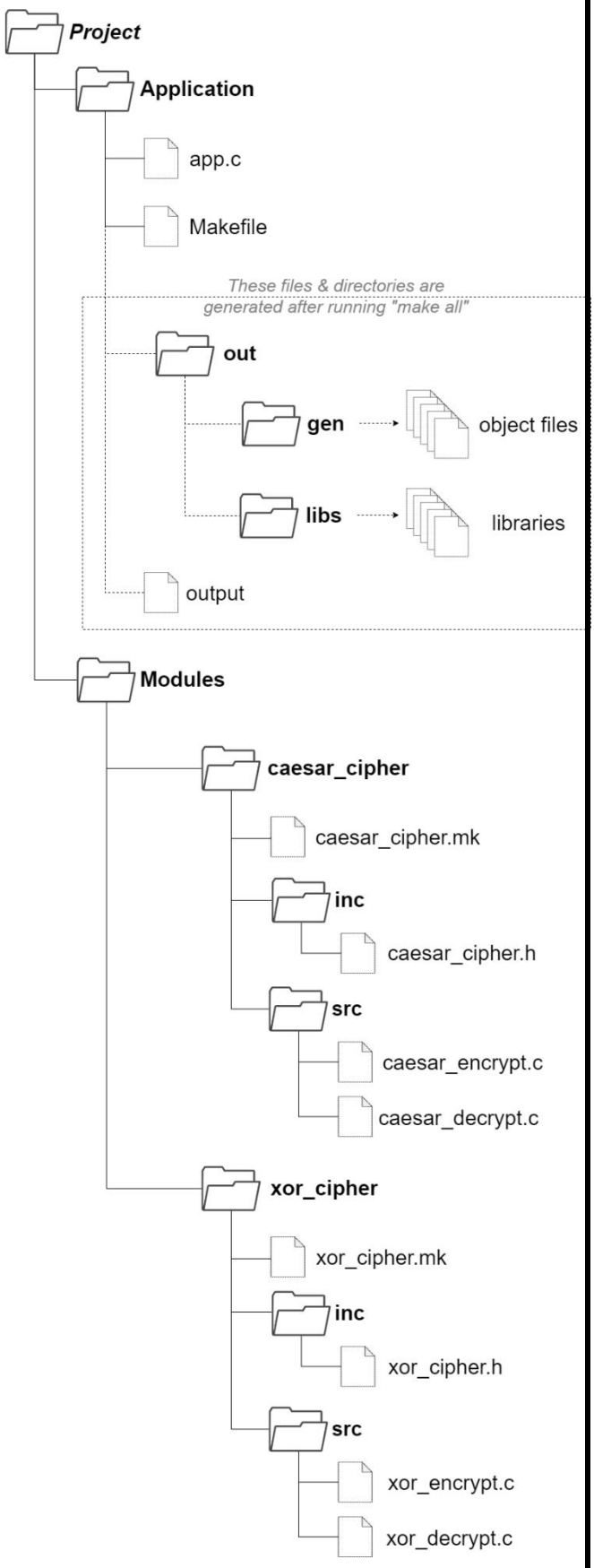
**As a test, you should run this command:**

**./output 3 K “aaa”**

and you should get this output::

**text encrypted with caesar: ddd**
**text decrypted with caesar: aaa**
**text encrypted with xor: ***
**text decrypted with xor: aaa**

Note:
- You can get the “.c” & “.h” files from the link we provided in the last post on the Facebook group



- You can also get a template for the Makefile, where I defined some variables you can use in you Makefile, in order to help you organize your thoughts.

## Part(4): Process Management:

We would like to create an Enhanced Remote Command Execution Application (ERCEA).

- Client/Server architecture using **TCP communication**
- Server must be able to receive packets **from any interface**
- Client must take remote IP/Port (that it should connect to) as **command line arguments**.
- Client must ask the **user to input a command to be executed on the server** (this command can include arguments or flags)
- Client must send this command to the server to be executed
- The **exit status from the executed command** (integer value) should be sent back from the server to the client. The client should print this command exit status.
- Client continue to send commands to the server until the user enters the **command "stop"**. Then, the client sends this command to the server, and both of them are terminated.
- The RCEA is enhanced by making the server able to **handle multiple connections** from multiple clients at the same time. This can be achieved by **creating a new child process for each connection** accepted from the client.
- Server should be able to **handle SIGINT signal** to terminate gracefully (just print that it received the SIGINT signal)
- Does this design create **zombie processes** ? if so, how can we handle it?

Hints to help you design the project: (these are suggestions not requirements)

- Start by implementing the RCEA (from the lab), then implement the ERCEA based on it.
- In the server, use an infinite loop to always accept new connections from clients (i.e. put "accept" function inside "while(1)")
- After receiving a connection, create a child process to handle it.
- Inside the child process, use another infinite loop to always receive commands until client sends "stop" (i.e. put "read" function inside the inner "while(1)")
- At the beginning of this loop, use "memset()" to reset the value of the array of char you use with "read()" function.
- Use "strncmp()" function to check if the received command is "stop"
- Use "system()" function to execute commands from C code
- Use "htonl()" to be able to send an integer (the exit status) over the socket connection

Run the following scenario:

- One server running on a terminal, and 2 clients, each one running on a separate terminal
- From client 1, you send "mkdir exam_dir", then "ls"
- From client 2, you send "touch exam_file", then "ls"
- From client 1, you send "mkdir -n dir2", and print the return value

- From client 2, you send “nooo”, and print the return value
- Send “stop” from both clients, and they are both terminated
- Interrupt the execution of the server using Ctrl+C, the server should print a message and gets terminated

### Deliverables for the Linux Administration project:

- For Part-1:
  - 2 Bash Scripts
  - 2 Text Files answering the second part of each question
  - 2 Screenshots of the execution
- For Part-2:
  - 3 Makefiles & 3 CMakeLists.txt
  - The whole project folder zipped in a zip file
    (2 projects: one for Makefile & one for CMake)
  - 2 screenshots of the execution (one for Makefile, and one for CMake)
- For Part-3:
  - 2 Source files (server.c & client.c)
  - A screenshot for the scenario explained

### IMPORTANT NOTE:

- You should try to implement the project one your own.
- **Don’t use chatgpt to generate the project.**
- You can use it for search purposes only (ex: to search for a function in CMake).

# Thank You

# Edges For Training Team