The objective of this minor lab is to start using simple LINUX commands and bash scripts

UNIX file name convention

1. All file names are case sensitive. So `my.txt My.txt MY.txt` all are different files.
2. Typically, you should use upper and lowercase letters, digits, `.` (dot), and `_` (underscore) symbols.
3. You can use other special characters such as blank space, but they are hard to use and it is better to avoid them.
4. More precisely, file names may contain any character except `/`, which is reserved as the separator between files
   and directories in a pathname. You cannot use the null character.
5. Most modern UNIX systems limit file names to 255 characters.
6. A file name must be unique inside its directory. For example, inside `/home/my` directory you cannot create a file
   named `report` and a directory named `report` . However, other directory may have files with the same names,
   for example, you can create a directory named `report` in `/tmp` .
7. Avoid using the following characters from appearing in file names: `> < | : &`
8. Note that UNIX allows white spaces, `<`, `>`, `|`, `\`, `:`, `(`, `)`, `&`, `;`, as well as wildcards such as `?` and `*`, to be quoted
   or escaped using `\` symbol.

Simple LINUX commands

1. Create 10 text files on your local machine (for instance, using notepad). The contents of the files are not important, but the names are: XXX1.0, XXX1.1, XXX.1.2 … XXX1.9
2. Transfer the files from your local machine to the remote host
3. Then rename the transferred files on the remote host to yyy1.0, yyy1.1, yyy1.2 … yyy1.9
4. Then transfer the renamed file from the remote host to your local machine
5. Check if you have ***bash*** shell by executing command **which bash** (you should get response `/bin/bash`) and check where it is located by executing **whereis bash** -- you should get response
   `bash: /bin/bash /usr/share/man/man1/bash.1.gz`
6. Then create a directory named **2XA3** (use **mkdir** command) and make it your current directory (use **cd** command). Then make sure that it really is your current directory using **pwd** command.
7. Create a text file named **xxx.yyy** by executing command **man cat > xxx.yyy**
8. Display the content of the file by **cat xxx.yyy**
9. Rename the file **xxx.yyy** to **yyy.xxx** using **mv** command.
10. Move the file **yyy.xxx** from the current directory to your home directory using **mv** command, then **cd** to your home directory and using **ls** check that the file is there.
11. From the home directory check whether the directory **2XA3** is empty using **ls** command.
12. If not, still from the home directory remove all the files from **2XA3** using **rm** command, and then remove the directory **2XA3** using **rmdir** command.
13. Then create the directory **2XA3** anew, create the file **xxx.yyy** as before and place it in the directory **2XA3**. Now try to remove the directory **2XA3** by the command **rmdir** , it will not work.
14. Consult **man** to find more about **rm** command, and remove the directory **2XA3** with all the files in it.
15. Can you explain what will **man cat | grep cat** produce? How many lines of output?
16. Can you explain what will **man man | grep cat** produce? How many lines of output?

Simple bash scripts

1. A bash script is a simple ASCII text file whose first line is `#!/bin/bash` and which contains LINUX commands (to be more precise bash commands) and is executable. A script is executed simply by typing its name on the command line and hitting ***enter***.
   Create a bash script file named **s1** which displays `Hello world` on the screen when executed by entering

the following sequence of commands (make sure that a file **s1** does not exist prior to typing in the first command) :

```
echo '#!/bin/bash' > s1
echo 'echo "Hello world"' >> s1
```

Do you understand why in the first command we use `>` while in the second command we use `>>` ? (Look up file redirection in the bash manual in Help section) What would happen if we first used `>>` and then `>` ?

2. Make the file **s1** executable by entering **chmod u+x s1** and then execute the script. What would happen if you did not make it executable and tried to execute it?
   If you see a message **s1: command not found** it most likely means that the directory in which you created the script **s1** is not in your path, so either add this directory to your path, or add the *current directory* to your path (see item 25 above), or execute the the script by **./s1** rather than by the usual **s1** . Can you explain why?
3. Then create the file **s1a** on your workstation by ***notepad*** and transfer it to ***moore*** and modify it by ***dos2unix***, or by using ***nano*** on ***moore***. Make it executable by ***chmod***. It should contain the same commands as **s1** . You will be experimenting with this file. Or you can use **cp** command to create it.
4. Find out what happens if **"** (double quotes) were replaced by **'** (single quotes). Can you explain why? Look up quoting in ***bash*** in the Help section bash manual.
5. Find out what happens if **"** (double quotes) were replaced by **`** (back quotes). Can you explain why? Look up quoting in ***bash*** in the Help section bash manual.

```
6. Create a bash script file s2 by executing the following sequence of commands:
7. echo '#!/bin/bash' > s2
8. echo 'echo -e "Please enter your name: "' >> s2
9. echo 'read name' >> s2
   echo 'echo "Nice to meet you $name"' >> s2
```

and make the script **s2** executable. When executed, it will do the following (in black is the text entered by the user):

```
Please enter your name: franya
Nice to meet you franya
```

10. Make a copy of **s2** named **s2a** .
11. Write a bash script **s3** that displays the following system information and follows the display format as closely as possible:

```
Kernel Details: Linux 2.6.18-419.el5 x86_64
GNU bash, version 3.2.25(1)-release (x86_64-redhat-linux-gnu)
Copyright (C) 2005 Free Software Foundation, Inc.
Uptime: 21:31:39 up 162 days, 10:27, 3 users, load average: 0.00, 0.00,
0.00
Server time: Sat Sep 7 21:31:39 EDT 2019
```

You will need to look up (use ***man***) the info for:
    1. built-in command **uname**
    2. built-in command **bash --version**
    3. built-in command **uptime**
    4. built-in command **date**
    5. and details about built-in command **echo** and how it handles **newlines**
12. At this point you should have five script files **s1**, **s1a**, **s2**, **s2a**, and **s3** in your directory. Create and execute a bash script that creates a new subdirectory **XXX** and moves the files **s1**, **s1a**, **s2**, **s2a**, and **s3** there while renaming them to **t1**, **t1a**, **t2**, **t2a**, and **t3**. Using **ls** command it checks that your directory contains no longer the files **s1**, **s1a**, **s2**, **s2a**, and **s3** but contains the directory **XXX**, and then again using **ls** command checks that the directory **XXX** contains the files **t1**, **t1a**, **t2**, **t2a**, and **t3**.
13. Look up (use ***man***) the info for command **od**. Write a bash script that will display the hexadecimal codes for the first 10 bytes of the binary file **/bin/zcat**
14. Write a bash script that will do the following: it displays a message `guess for how long this computer had been up yesterday`. Then it reads the user's response into a variable. It then increments the number by one. Then in captures the output of the

command `uptime` in a variable. Then it extracts the substring of it that contains the number of days. It then displays a message `today  this computer has been up for X days` where `X` is the number of days obtained from `uptime`.It is followed by a message `you guessed Y days` where `Y` is the number entered by the user and incremented by 1. If the two numbers agree, the script displays a message `good guess` otherwise it displays a message `bad guess`.
How to increment a variable by 1: let `xxx` *hold the number, then* `xxx=$(($xxx+1))` *will now hold a value bigger by 1*
Hot to extract a substring: let `xxx` *hold the string, then* `${xxx:P:L}` *refers to the substring starting at position* `P` *and of length* `L` ; *for instance, if* `xxx="helloWorld"` then `${xxx:0:2}` is `he`, while `${xxx:2:4}` is `lloW`

A sample run:

```
15.  guess for how long this computer had been up yesterday
16.  130
17.  131
18.  today this computer has been up for 171 days
19.  you guessed 131 days
bad guess
```

A sample run:

```
guess for how long this computer had been up yesterday
170
171
today this computer has been up for 171 days
you guessed 171 days
good guess
```