Shell Guide

Overview

- The shell should operate in a basic way: when you type in a command (in response to its prompt), the shell creates a child process that executes the command you entered and then prompts for more user input when it has finished.
- This simple linux shell built in C displays a prompt where the user can enter the commands to execute.
- The shell can be run in two modes : interactive or batch. In interactive mode, the user types a command at the prompt and the output will be instantly displayed there.

  In batch mode, your shell is started by specifying a batch file on its command line; the batch file contains the list of commands that should be executed. In batch mode, you should echo each line you read from the batch file back to the user before executing it.

  In each mode, your shell stops accepting new commands when it sees the quit command on a line or reaches the end of the input stream (i.e., the end of the batch file).

- This shell handles :
  - Simple commands
  - Commands with arguments
  - Commands with operators(&& , || , ;)
  - Pipes (|)
  - Redirections (> , >>)
  - Both interactive and batch modes
  - Errors
  - History

Functions specifications

Function	Inputs	Output
interactive_mode	None	void
batch_mode	filename	void
parse_command	command	void
execute_command	command	int
execute_composed_command	command1 / command2 / operator	void
pipe_launch	command1 /command2	void
redirection_launch	command1 / command2 / operator	void

interactive_mode

This function allows the user to interact with the shell by continuously prompting the user for input and executing commands. It includes a loop that reads the input from the user and calls the **parse_command** function with the input to execute the corresponding command. It uses the **readline** library to interact with the user.

batch_mode

The batch_mode function reads commands from a **batch** file and executes them one by one. It takes the name of the batch file as an input, opens the file, reads each line of the file, removes the newline characters, passes the command to **parse_command** function for execution, and finally closes the file.

Additionally, it prints each command before executing it, which can be useful for debugging or understanding the commands being executed.

parse_command

This function starts by using the **strtok** function to tokenize the command string into **an array of strings.** It checks the presence of any composed command operators **(;, &&, ||, |, >>, >)** in the array.

If the operator is found, the function splits the token array into two separate arrays, one for the **command1** and one for the **command2**.

The function then calls the **execute_composed_command** function to execute the composed command.

If the operator is not found, the function calls the **execute_command** function to execute the simple command.

execute_composed_command

This function ensures the execution of multiple commands. To do so,
if the command1 is **quit**, it exits the prompt.
If it is not the case, based on the given operator it determines which action to take next. If the operator is

- **'|'** : calls the **pipe_launch** function to handle pipes.
- **'>>'** or **'>' :** calls the **redirection_launch** function to handle redirection. If

it is not one of these operators, it executes **command1** then checks the operator again:

- **"&&"** : if command1 is **well executed** then and only then it executes **command2**.
- **"||" :** if there was **a problem** in **command1** execution then and only then it executes **command2.**
- **";" :** executes **command2**.

execute_command

This function allows us to execute a simple command which is given as an input .

For execution, it calls **execvp** that takes as inputs the command and its **arguments as an array of strings.**

pipe_launch

The **pipe()** system function is used to open **file descriptors**, which are used for **inter-process communication** in Linux.

You pass in an int type array **fd (file descriptors)** consisting of **two elements** to the function **pipe()** which creates **two file descriptors** in the fd array.

- The first element of the fd array, **fd[0]** is used for reading data from the pipe **(read end of pipe)** ➔ **the output of the command1 in our case.**
- The second element of the fd array, **fd[1]** is used for writing data to the pipe **(write end of pipe)** ➔ **the input of the command2 in our case .**

This function allows us to execute **command1** and save its output in **fd[1]**.
Then, the process accesses the read end of the pipe **fd[0]** to get the command1 output as an input for **command2.**

redirection_launch

This function enables us to change the standard output (stdout) of any executed command and **uses the specified file as standard output**.

- If file does not exist, it is created.
- If file exists, and the character used is '>', it is **truncated**, and its previous contents are lost.
- Else, if the file exists and the character used is '>>', it **appends** output to the end offile.

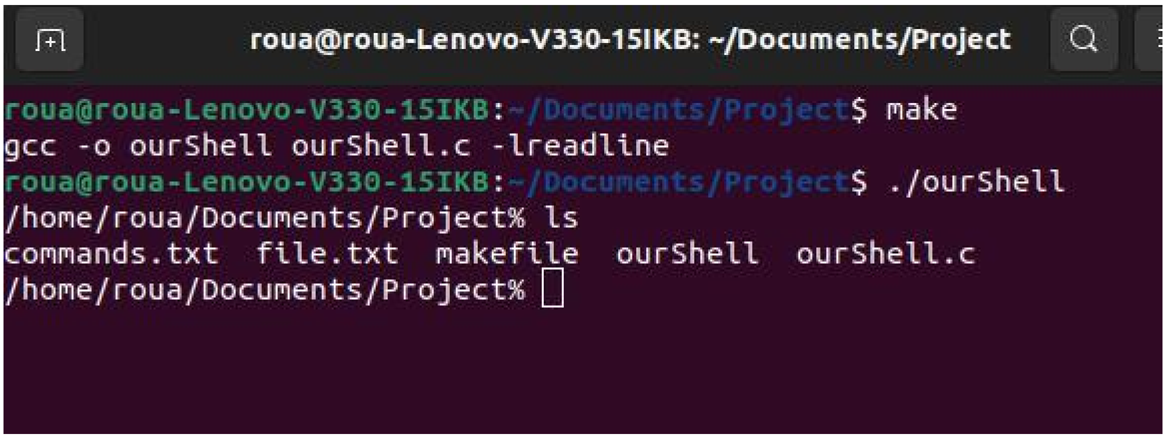
First, the child process opens the file using **fopen**() that returns a pointer to the file "fp" . File descriptor ' 1' or STDOUT_FILEND (responsible for directing the output) points to stdout (the screen).

To ensure redirection of the output to the given file, the child process changes STDOUT_FILEND to point to the file where we want to save the output "fp" using **dup2**().

Since dup2 accepts only file descriptors as input we used **fileno**() to convert "fp" to a file descriptor. After that it executes the command.

Execution examples

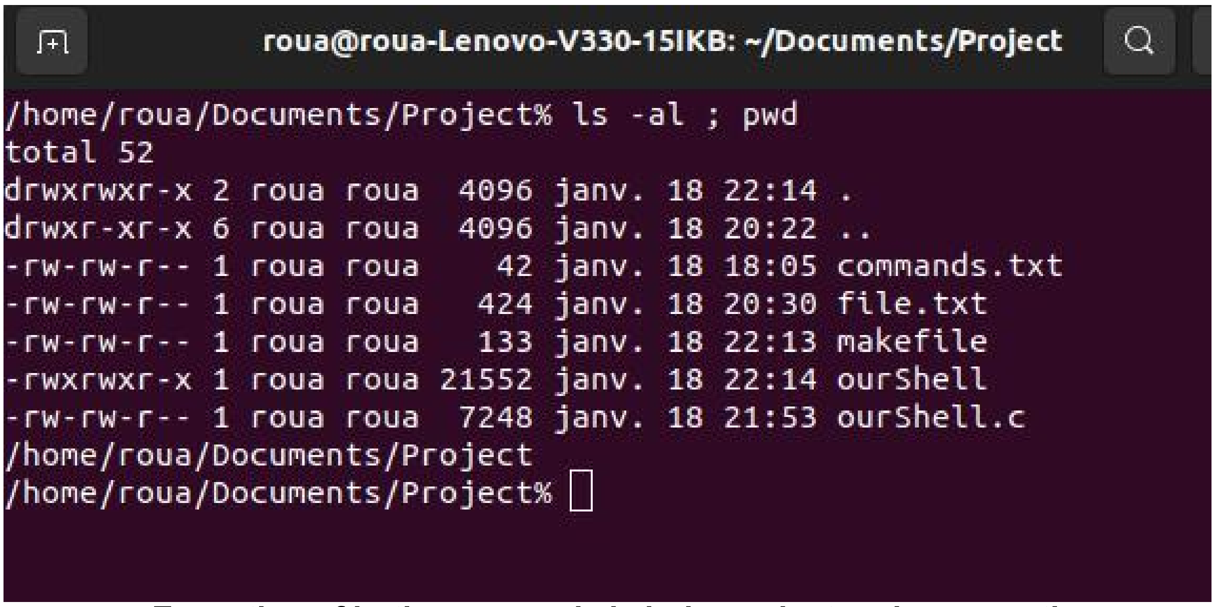
Simple command



```
roua@roua-Lenovo-V330-15IKB: ~/Documents/Project
roua@roua-Lenovo-V330-15IKB:~/Documents/Project$ make
gcc -o ourShell ourShell.c -lreadline
roua@roua-Lenovo-V330-15IKB:~/Documents/Project$ ./ourShell
/home/roua/Documents/Project% ls
commands.txt  file.txt  makefile  ourShell  ourShell.c
/home/roua/Documents/Project% 
```

Multiple commands

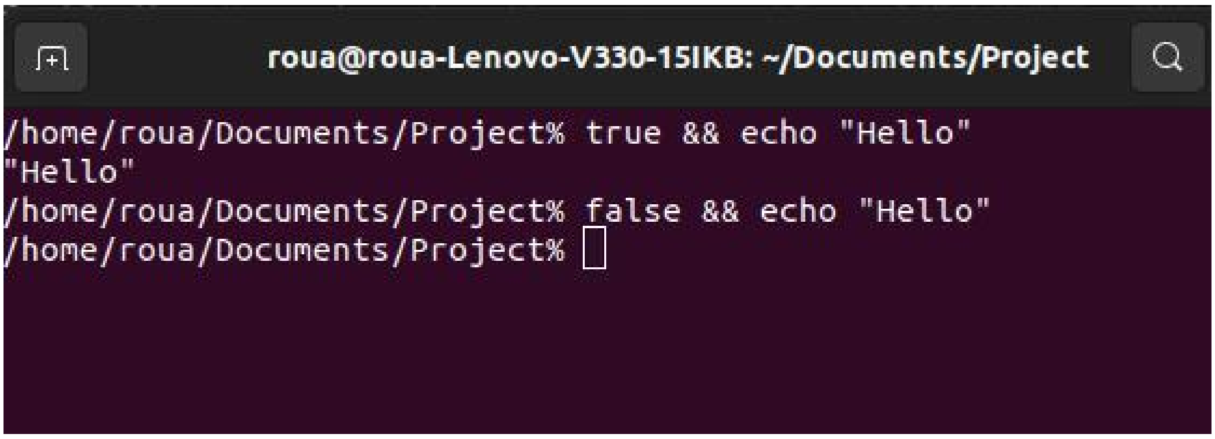
- **With ; operator**



```
/home/roua/Documents/Project% ls -al ; pwd
total 52
drwxrwxr-x 2 roua roua  4096 janv. 18 22:14 .
drwxr-xr-x 6 roua roua  4096 janv. 18 20:22 ..
-rw-rw-r-- 1 roua roua    42 janv. 18 18:05 commands.txt
-rw-rw-r-- 1 roua roua   424 janv. 18 20:30 file.txt
-rw-rw-r-- 1 roua roua   133 janv. 18 22:13 makefile
-rwxrwxr-x 1 roua roua 21552 janv. 18 22:14 ourShell
-rw-rw-r-- 1 roua roua  7248 janv. 18 21:53 ourShell.c
/home/roua/Documents/Project
/home/roua/Documents/Project% 
```

⇒ **Execution of both commands is independent and consecutive.**

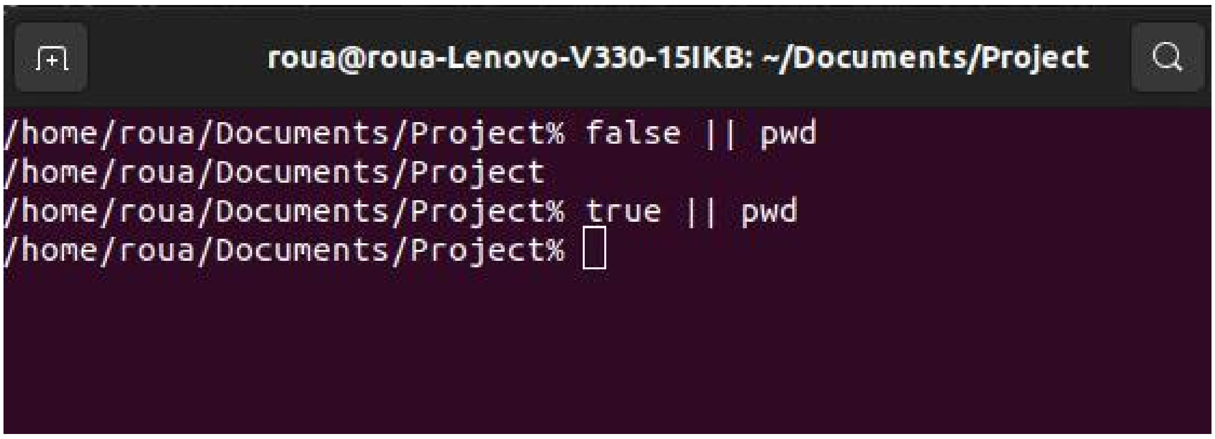
- **With && operator**



```
/home/roua/Documents/Project% true && echo "Hello"
"Hello"
/home/roua/Documents/Project% false && echo "Hello"
/home/roua/Documents/Project% 
```

⇒ **Command2 [echo “hello”] is executed only when the execution of command1 is valid (true).**

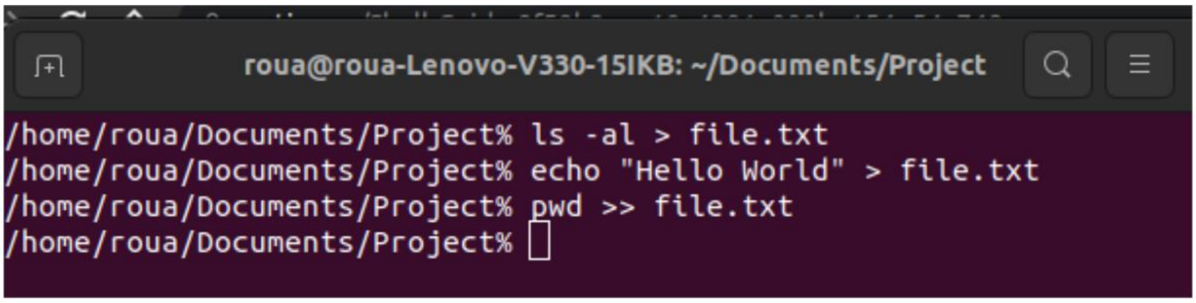
- **With || operator**


roua@roua-Lenovo-V330-15IKB: ~/Documents/Project

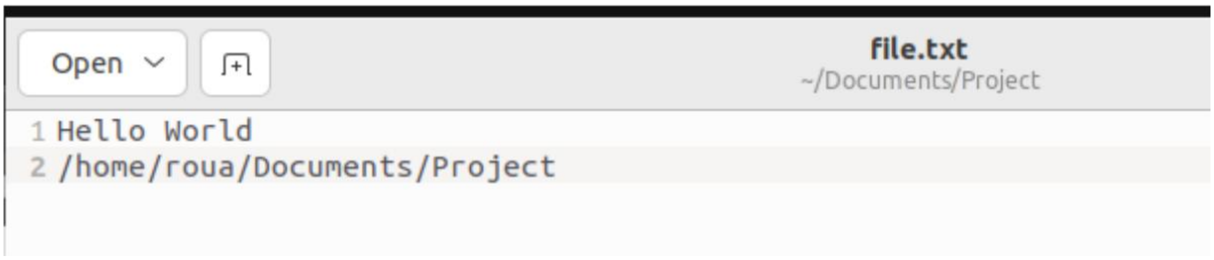
```
/home/roua/Documents/Project% false || pwd
/home/roua/Documents/Project
/home/roua/Documents/Project% true || pwd
/home/roua/Documents/Project%
```

⇒ Command2 [pwd] is executed only when the execution of command1 is not valid (false).

- **With >> operator**


roua@roua-Lenovo-V330-15IKB: ~/Documents/Project

```
/home/roua/Documents/Project% ls -al > file.txt
/home/roua/Documents/Project% echo "Hello World" > file.txt
/home/roua/Documents/Project% pwd >> file.txt
/home/roua/Documents/Project%
```

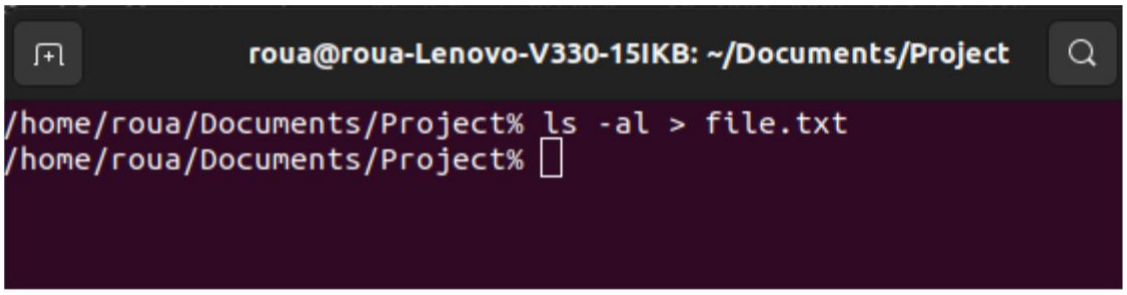


file.txt
~/Documents/Project

```
Hello World
/home/roua/Documents/Project
```

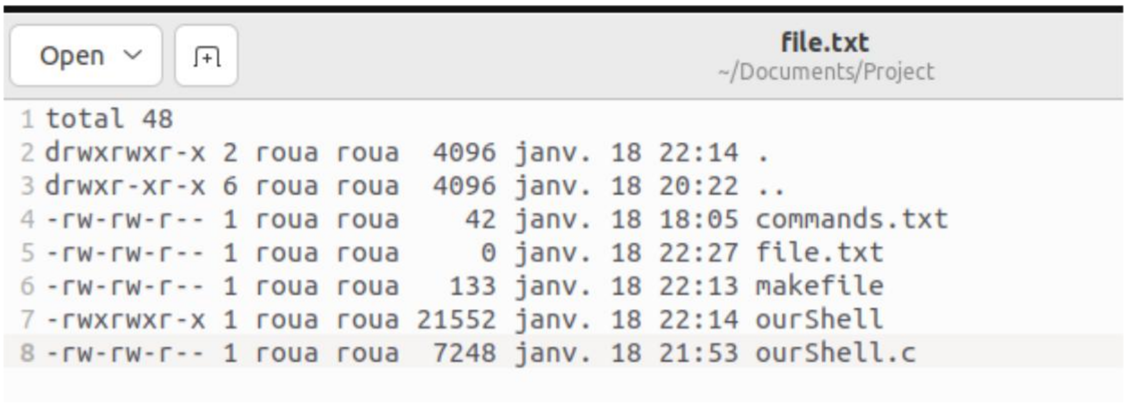
⇒ The outputs of the two redirection are appended since the operator of the second redirection is ‘>>’

- **<u>With > operator</u>**



roua@roua-Lenovo-V330-15IKB: ~/Documents/Project

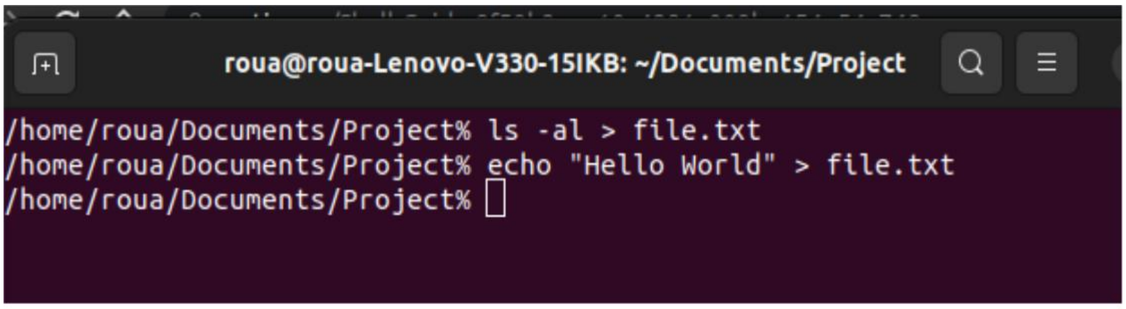
```
/home/roua/Documents/Project% ls -al > file.txt
/home/roua/Documents/Project%
```



Open | file.txt ~/Documents/Project

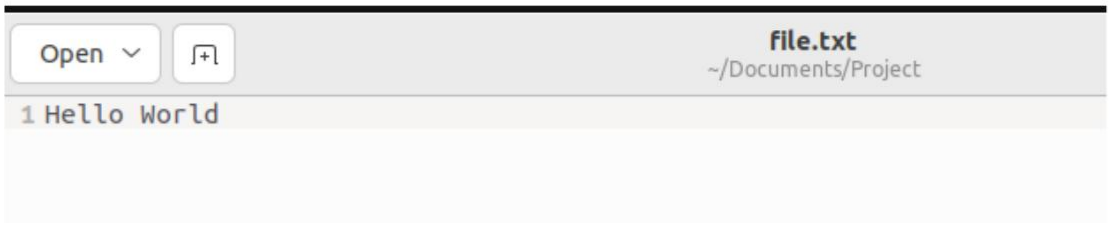
```
total 48
drwxrwxr-x 2 roua roua  4096 janv. 18 22:14 .
drwxr-xr-x 6 roua roua  4096 janv. 18 20:22 ..
-rw-rw-r-- 1 roua roua    42 janv. 18 18:05 commands.txt
-rw-rw-r-- 1 roua roua     0 janv. 18 22:27 file.txt
-rw-rw-r-- 1 roua roua   133 janv. 18 22:13 makefile
-rwxrwxr-x 1 roua roua 21552 janv. 18 22:14 ourShell
-rw-rw-r-- 1 roua roua  7248 janv. 18 21:53 ourShell.c
```

⇒ **Redirection of the command output to the file file.txt**



roua@roua-Lenovo-V330-15IKB: ~/Documents/Project

```
/home/roua/Documents/Project% ls -al > file.txt
/home/roua/Documents/Project% echo "Hello World" > file.txt
/home/roua/Documents/Project%
```

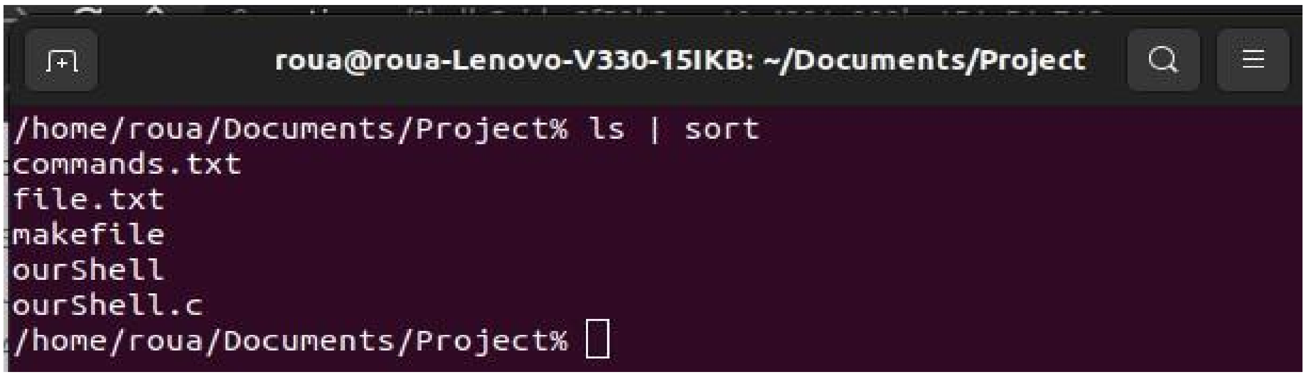


Open | file.txt ~/Documents/Project

```
Hello World
```

⇒ **The file is overwritten after the second redirection since the operator is '>'**

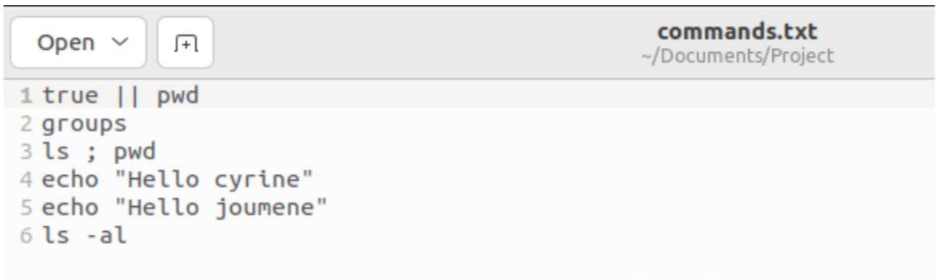
- **With | operator**



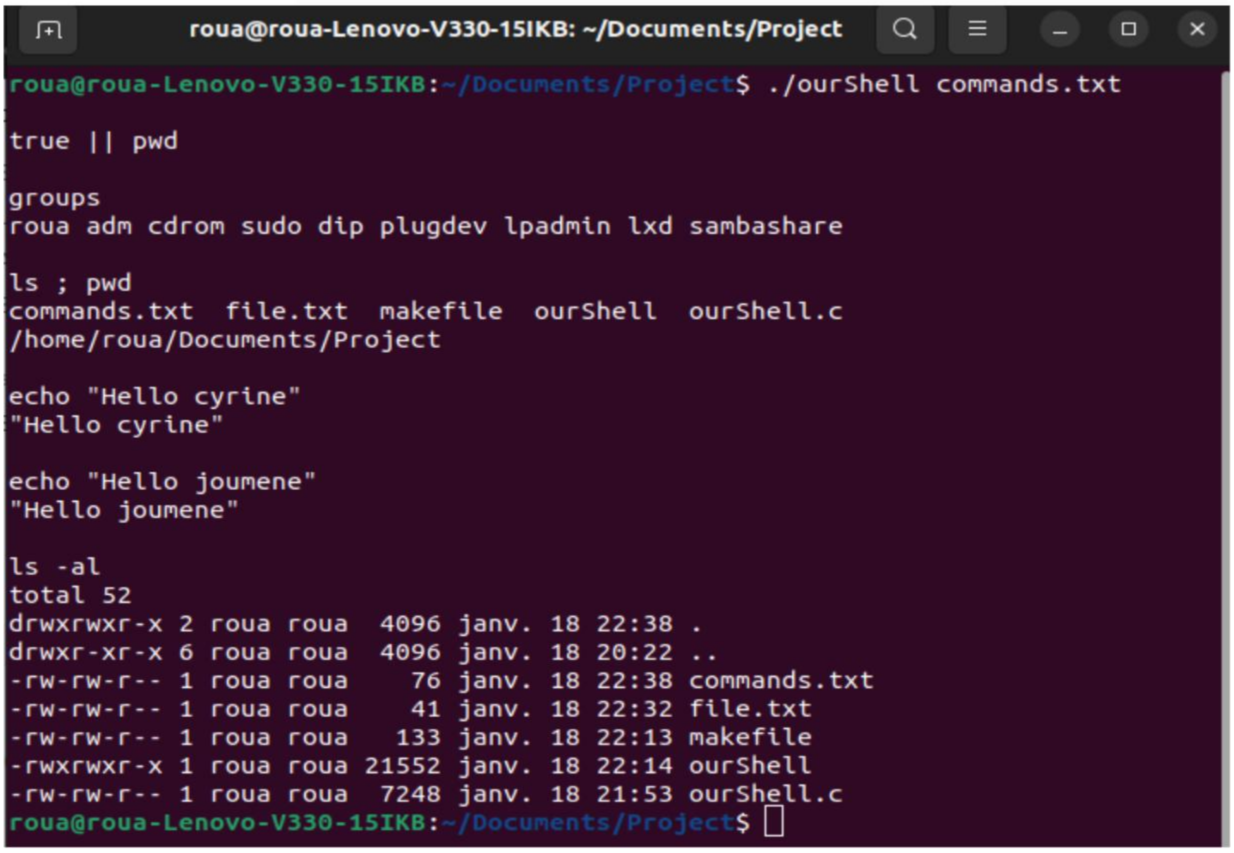
```
/home/roua/Documents/Project% ls | sort
commands.txt
file.txt
makefile
ourShell
ourShell.c
/home/roua/Documents/Project%
```

⇒**The output of command1[ls] was given as input of command2 [sort] to sort it**

Batch mode



```
true || pwd
groups
ls ; pwd
echo "Hello cyrine"
echo "Hello joumene"
ls -al
```



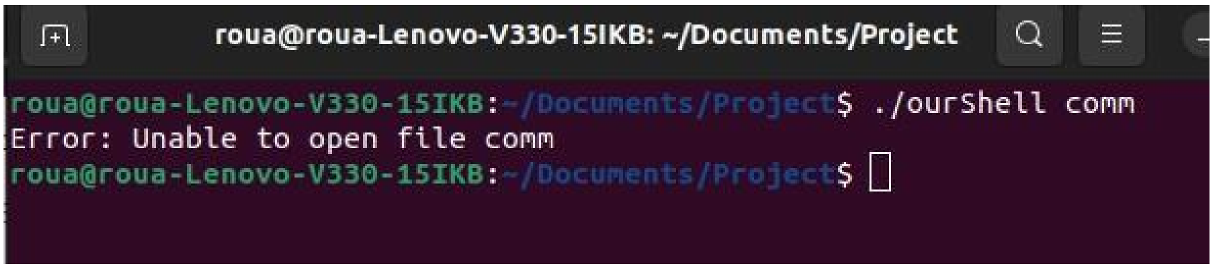
```
roua@roua-Lenovo-V330-15IKB:~/Documents/Project$ ./ourShell commands.txt

true || pwd

groups
roua adm cdrom sudo dip plugdev lpadmin lxd sambashare

ls ; pwd
commands.txt  file.txt  makefile  ourShell  ourShell.c
/home/roua/Documents/Project

echo "Hello cyrine"
"Hello cyrine"

echo "Hello joumene"
"Hello joumene"

ls -al
total 52
drwxrwxr-x 2 roua roua  4096 janv. 18 22:38 .
drwxr-xr-x 6 roua roua  4096 janv. 18 20:22 ..
-rw-rw-r-- 1 roua roua    76 janv. 18 22:38 commands.txt
-rw-rw-r-- 1 roua roua    41 janv. 18 22:32 file.txt
-rw-rw-r-- 1 roua roua   133 janv. 18 22:13 makefile
-rwxrwxr-x 1 roua roua 21552 janv. 18 22:14 ourShell
-rw-rw-r-- 1 roua roua  7248 janv. 18 21:53 ourShell.c
roua@roua-Lenovo-V330-15IKB:~/Documents/Project$
```

⇒ **All the commands in file “commands.txt” are executed one by one .**

⇒ **Every command and its output are shown in the prompt.**

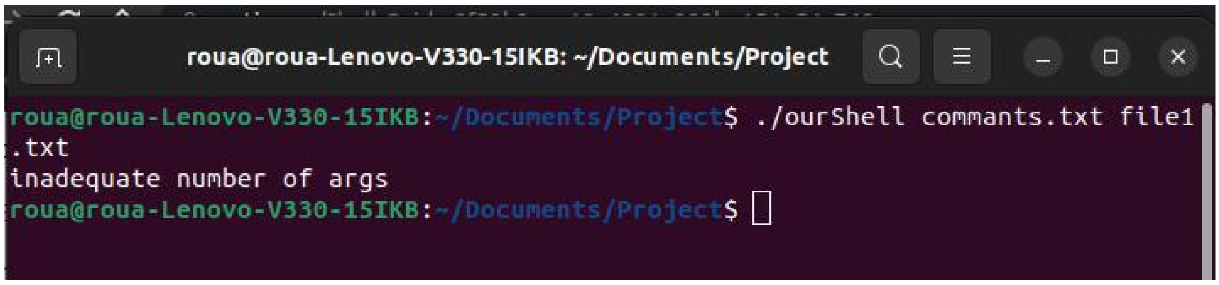
Errors

- **Batch file does not exist**



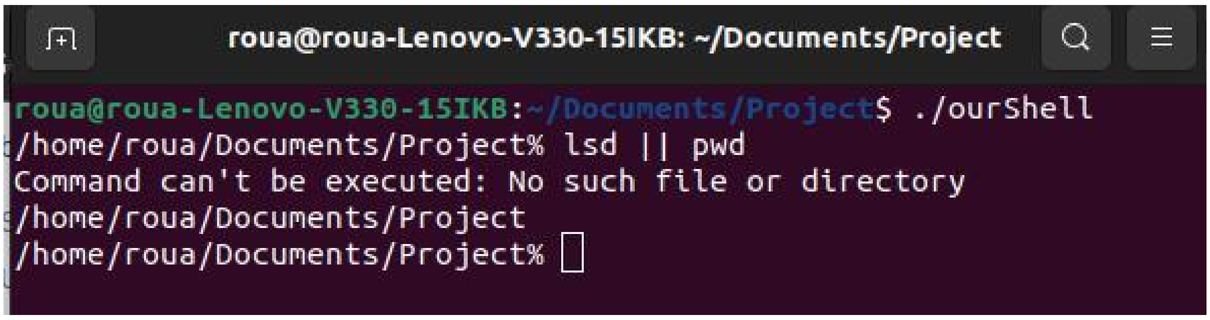
```
roua@roua-Lenovo-V330-15IKB:~/Documents/Project$ ./ourShell comm
Error: Unable to open file comm
roua@roua-Lenovo-V330-15IKB:~/Documents/Project$ 
```

- **Inadequate number of arguments**



```
roua@roua-Lenovo-V330-15IKB:~/Documents/Project$ ./ourShell commants.txt file1
.txt
inadequate number of args
roua@roua-Lenovo-V330-15IKB:~/Documents/Project$ 
```

- **Command does not exist or can't be executed**



```
roua@roua-Lenovo-V330-15IKB:~/Documents/Project$ ./ourShell
/home/roua/Documents/Project% lsd || pwd
Command can't be executed: No such file or directory
/home/roua/Documents/Project
/home/roua/Documents/Project% 
```