Student id:

Course:

Module: 5004CEM Operating Systems and Security Submission data: 23rd March 2020

Portfolio

Contents

[Lab Activity 1 – Operating Systems Tasks and Programming 5](#_Toc30774083)

[a. Comparison between the Harvard Architecture and Von Neumann Architecture 5](#_Toc30774084)

[b. Programming activity 5](#_Toc30774085)

[Lab Activity 2 – Linux Command Line (Commands and outcomes from a series of small tasks that require use of a number of Linux commands) 6](#_Toc30774086)

[Tasks – Files 6](#_Toc30774087)

[a. Create a directory in your /home/5004CEM\_Session/<username> directory with a name made up of you second name followed by 5004CEM and the year (mine would be **ELSHAW5004CEM2020**). Make the directory **read/write/executable** only for you, **read/write** for your groups and **read** only for others. 6](#_Toc30774088)

[b. Show evidence of this using the appropriate version of the **ls command**. 6](#_Toc30774089)

[c. Download the script http://www.centerkey.com/tree/tree.sh to your home directory using wget and make the file executable. 6](#_Toc30774090)

[d. Create a directory called **wrongDirectory**. You release it is not what you wanted so delete it. 6](#_Toc30774091)

[e. Create Portfolio1-2020 and Portfolio2-2020 directories in the directory you created in part a. 6](#_Toc30774092)

[f. Create numbered directories in the Portfolio1-2020 Directory (Lab0-5004CEM to Lab10-5004CEM) and in the Porfolio2-2020 Directory (Lab11-5004CEM to Lab20-5004CEM). 6](#_Toc30774093)

[g. In <YourSecondName>5004CEM2020 directory create a text file called **LastTask.txt** and then using the appropriate Linux command copy this document into Directory Lab0-5004CEM. 7](#_Toc30774094)

[h. Show how would you display all files in a directory including hidden ones files? 7](#_Toc30774095)

[Linux Commands – Mixed 7](#_Toc30774096)

[a. Using the date command show todays date and the time and date 5 years ago. Using the cal command show the month that you were born. Change this calendar to make Monday the first day of the week. 7](#_Toc30774097)

[b. Move into the lab1-5004CEM directory and use the appropriate command to show the current directory. 7](#_Toc30774098)

[c. Display the time when a user (ab0487) last login, the user’s home directory, and the user account’s full name. 7](#_Toc30774099)

[d. Find out how to prevent the effects of talk, write and wall from interrupting you. What command can you use? 7](#_Toc30774100)

[e. Show the command to verify that www.coventry.ac.uk exists and can accept requests. 7](#_Toc30774101)

[f. Display your name and favourite programming language on the screen using the echo command. 7](#_Toc30774102)

[g. Find out how you can display your username on the screen and at least two ways to display who is logged on. 8](#_Toc30774103)

[h. Use two ways to list the processes that are running. 8](#_Toc30774104)

[i. What are the differences between the Linux commands copy (cp), rename and move? 8](#_Toc30774105)

[j. With a single command, how would you get systems information such as processes, memory, paging and CPU activity? 8](#_Toc30774106)

[Tasks – Document Manipulation 8](#_Toc30774107)

[a. Use **cat** to show the contents of the file. 8](#_Toc30774108)

[b. Use an appropriate command to display the CRC checksum and byte count of the file. 8](#_Toc30774109)

[c. Use **grep** to show only lines not containing the words "she" or "he". Lines contain both “she” and “he” should be shown. 8](#_Toc30774110)

[d. Use **grep** to show the 5 lines above a line containing the text ‘the’. 8](#_Toc30774111)

[e. Using Linux commands you should count the lines containing “she” and “he” but not both and display the line numbers that “she” and “he” but not both appear on in the original document. 9](#_Toc30774112)

[f. Find a command to list the top 3 lines of the **poem.txt** file and then the bottom line of these top 3. 9](#_Toc30774113)

[g. Find a command to split the **poem.txt** file into different files each containing 2 lines. 9](#_Toc30774114)

[h. Use **sort** and **rev** to reverse the sorted contents of poem.txt and append the output to poem2.txt. 9](#_Toc30774115)

[i. Use at least two appropriate Linux commands to compare these two files (poem.txt and poem\_name.txt) and see if they are the same. 9](#_Toc30774116)

[j. Use **sort** to sort the content of poem.txt file in a random order and redirect the output to a new file called **poem2.txt**. 9](#_Toc30774117)

[k. Sort the **poem.txt** file, remove the duplicates and reverse the sorted contents and append the output to **poem2.txt**. 9](#_Toc30774118)

[l. Create an **alias** so rather than having to type the full command for k) you can type **yourSort**. 10](#_Toc30774119)

[Lab Activity 4 Bootloader 11](#_Toc30774120)

[a. Brief description of the Lab activity and what you did 11](#_Toc30774121)

[b. Boot pragma Linux with bochs 11](#_Toc30774122)

[c. Make a bootloader that displays your student details and diamond 11](#_Toc30774123)

[Commented bootloader code to display your student details and diamond 11](#_Toc30774124)

[Output from bochs showing student details and diamond 11](#_Toc30774125)

[Lab Activity 7 Buffer 12](#_Toc30774126)

[a. Explain what the code below does. 12](#_Toc30774127)

[b. Code to show error outputs. 12](#_Toc30774128)

[c. Code to create statistics on the document. 12](#_Toc30774129)

[d. Code to compare the documents. 12](#_Toc30774130)

[e. Code to create summary. 12](#_Toc30774131)

[Lab Activity 8 Cache Buffer 13](#_Toc30774132)

[a. Brief Description of Cache Buffer Activity 13](#_Toc30774133)

[b. Commented implementation of the cr\_handle function 13](#_Toc30774134)

[c. Comment updated code to show that each byte is being read, and when the buffer is being refilled. 13](#_Toc30774135)

[d. Commented updated code showing the required statistical information as well as how many times the words ‘Iran’, ‘Tehran’ and ‘email’ appear. 13](#_Toc30774136)

[Lab 10: The Cache Buffer from week 8 with system calls 14](#_Toc30774137)

[a. Brief description of the activity 14](#_Toc30774138)

[b. Changes the cache\_handle library from using the fopen, fread, fclose functions to the system call versions open, read, close 14](#_Toc30774139)

[c. Changes cache\_handle library to remove (as far as possible) the effects of caching on the library. 14](#_Toc30774140)

[Lab Activity 13 – Job control 15](#_Toc30774141)

[a. Description of the activity 15](#_Toc30774142)

[b. Menu System 15](#_Toc30774143)

[Lab Activity 14 – Linux command-line manipulation of processes 16](#_Toc30774144)

[a. Paragraph on disown and an alternative/ 16](#_Toc30774145)

[b. Description and example of using watch command 16](#_Toc30774146)

[c. Process manipulation 16](#_Toc30774147)

[Example(s) of how to start process 16](#_Toc30774148)

[Example(s) of how to suspend process 16](#_Toc30774149)

[Example(s) of how to run process in background 16](#_Toc30774150)

[Example(s) of how to run process in foreground and bring from background 16](#_Toc30774151)

[Example(s) of how to kill a process 16](#_Toc30774152)

[Lab Activity 15 IPC and Synchronisation 17](#_Toc30774153)

[a. Brief description of activity 17](#_Toc30774154)

[b. Modified semaphore example code so that the two processes output the song. 17](#_Toc30774155)

[c. Modified code to write Liza part to stderr and redirect the two parts to a file. 17](#_Toc30774156)

[References 18](#_Toc30774157)

# Lab Activity 1 – Operating Systems Tasks and Programming

## Comparison between the Harvard Architecture and Von Neumann Architecture

[Paragraph on the computer architectures]

## Programming activity

1. //This C++ code is used to take an input string and
2. // put the instructions in an array
3. #include <iostream>
4. #include <string>
5. #include <sstream>

8. **int** main() {
9. //set up input string
10. std::string input="orange lift right";
11. **bool** flag = **false**;
13. //set up map with instruction sets
14. std::string time[4] = {"1second", "2seconds", "5seconds", "unlimited"};
15. std::string move[5] = {"left", "right", "forward", "backward", "stop"};
16. std::string object[5] = {"orange", "apple", "car", "bus", "diamond"};
17. std::string action[6] = {"recognise", "eat", "see", "lift", "drop", "fetch"};
18. std::string size[4] = { "small", "big", "little", "massive"};
19. std::string location[3] = {"door", "kitchen", "table"};
21. //initalise input stream
22. std::stringstream currentstring(input);
23. **int** count=-1;
24. std::string instruction[10];
26. //Repeatedly put instruction in string array
27. **while** (currentstring.good())
28. {
29. count=count+1;
30. currentstring >> instruction[count];
31. std::cout << instruction[count] << " - - / - - " << **sizeof**(instruction)/**sizeof**(\*instruction) << std::endl;
32. }
34. //check for string structure
35. **if**(count == 1)
36. {
37. //check if instructions are of type <Move> <Time>
38. std::cout << "<Move> <Time>" << std::endl;
40. **for**(**int** i = -1; i < 5; i++)
41. {
42. **if**(instruction[0] == move[i])
43. {
44. std::cout << "Check" << std::endl;
45. **break**;
46. }
47. **else**
48. {
49. **if**(i == 4)
50. {
51. std::cout << "First bad" << std::endl;
52. **return** 0;
53. }
54. }
55. }
57. **for**(**int** i = -1; i < 4; i++)
58. {
59. **if**(instruction[1] == time[i])
60. {
61. std::cout << "Check" << std::endl;
62. **break**;
63. }
64. **else**
65. {
66. **if**(i == 3)
67. {
68. std::cout << "Second bad" << std::endl;
69. **return** 0;
70. }
71. }
72. }
73. **return** 0;
74. }
76. **else** **if**(count == 2)
77. {
78. /\*check if instructions are of type <Object> <Action> <Time> ...\*/
79. std::cout << "<Object> <Action> <Time> ..." << std::endl;
81. //check if first string is object if not then check if it is location if not then error
82. //Check object for first word
83. **for**(**int** i = -1; i < 5; i++)
84. {
85. **if**(instruction[0] == object[i])
86. {
87. //check if it is action or size
88. //check action for second word
89. **for**(**int** i = -1; i < 6; i++)
90. {
91. **if**(instruction[1] == action[i])
92. {
93. //check time for third word
94. **for**(**int** i = -1; i < 4; i++)
95. {
96. **if**(instruction[2] == time[i])
97. {
98. std::cout << "Check" << std::endl;
99. flag = **true**;
100. **break**;
101. }
102. **if**(flag == **true**)
103. {
104. **break**;
105. }
106. }
107. }
108. **else**
109. {
110. //check size for second word
111. **for**(**int** i = -1; i < 4; i++)
112. {
113. **if**(instruction[1] == size[i])
114. {
115. //check action for third word
116. **for**(**int** i = -1; i < 4; i++)
117. {
118. **if**(instruction[2] == action[i])
119. {
120. std::cout << "Check" << std::endl;
121. flag = **true**;
122. **break**;
123. }
124. **if**(flag == **true**)
125. {
126. **break**;
127. }
129. }
130. }
131. **if**(flag == **true**)
132. {
133. **break**;
134. }
135. }
136. }
137. **if**(flag == **true**)
138. {
139. **break**;
140. }
142. }
143. **if**(flag == **true**)
144. {
145. **break**;
146. }
147. }
148. **else**
149. {
150. //Check location for first word
151. **for**(**int** i = -1; i < 3; i++)
152. {
153. **if**(instruction[0] == location[i])
154. {
155. //check action for second word
156. **for**(**int** i = -1; i < 6; i++)
157. {
158. **if**(instruction[1] == action[i])
159. {
160. //check object for third word
161. **for**(**int** i = -1; i < 5; i++)
162. {
163. **if**(instruction[2] == object[i])
164. {
165. std::cout << "Check" << std::endl;
166. flag = **true**;
167. **break**;
168. }
169. **if**(flag == **true**)
170. {
171. **break**;
172. }
173. }
174. }
175. **if**(flag == **true**)
176. {
177. **break**;
178. }
179. }
180. }
181. **if**(flag == **true**)
182. {
183. **break**;
184. }
185. }
186. }
187. **if**(flag == **true**)
188. {
189. **break**;
190. }
192. **if**(i == 4)
193. std::cout << "Not the right syntax" << std::endl;
194. }
195. **return** 0;
196. }
197. **else** **if**(count == 3)
198. {
199. //check if instructions are of type <Move> <Time> <Move> <Time>
200. std::cout << "<Move> <Time> <Move> <Time>" << std::endl;
202. **for**(**int** i = -1; i < 5; i++)
203. {
204. **if**(instruction[0] == move[i])
205. {
206. std::cout << "Check" << std::endl;
207. **break**;
208. }
209. **else**
210. {
211. **if**(i == 4)
212. {
213. std::cout << "First bad" << std::endl;
214. **return** 0;
215. }
216. }
217. }
219. **for**(**int** i = -1; i < 4; i++)
220. {
221. **if**(instruction[1] == time[i])
222. {
223. std::cout << "Check" << std::endl;
224. **break**;
225. }
226. **else**
227. {
228. **if**(i == 3)
229. {
230. std::cout << "Second bad" << std::endl;
231. **return** 0;
232. }
233. }
234. }
236. **for**(**int** i = -1; i < 5; i++)
237. {
238. **if**(instruction[2] == move[i])
239. {
240. std::cout << "Check" << std::endl;
241. **break**;
242. }
243. **else**
244. {
245. **if**(i == 4)
246. {
247. std::cout << "Third bad" << std::endl;
248. **return** 0;
249. }
250. }
251. }
253. **for**(**int** i = -1; i < 4; i++)
254. {
255. **if**(instruction[3] == time[i])
256. {
257. std::cout << "Check" << std::endl;
258. **break**;
259. }
260. **else**
261. {
262. **if**(i == 3)
263. {
264. std::cout << "Fourth bad" << std::endl;
265. **return** 0;
266. }
267. }
268. }
269. **return** 0;
271. }
272. **else**
273. {
274. std::cout << "That syntax is not understandable by the robot.";
275. }
277. **return** 0;
278. }

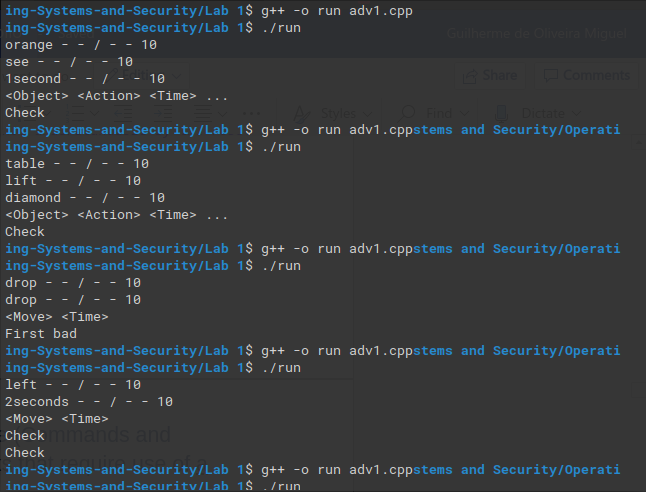


Fig 1 – Order Syntax checker 1

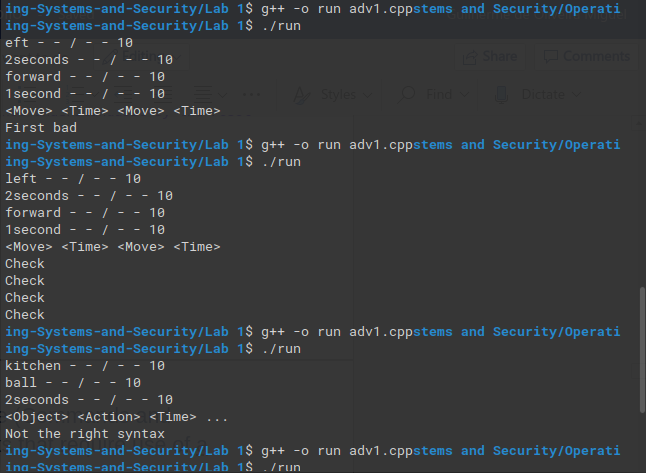
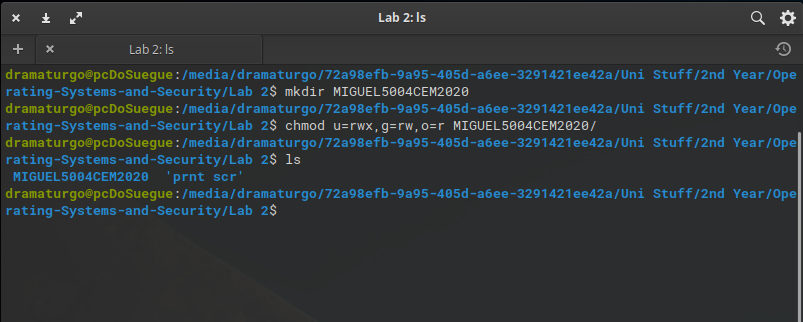


Fig 2 – Order Syntax checker 2

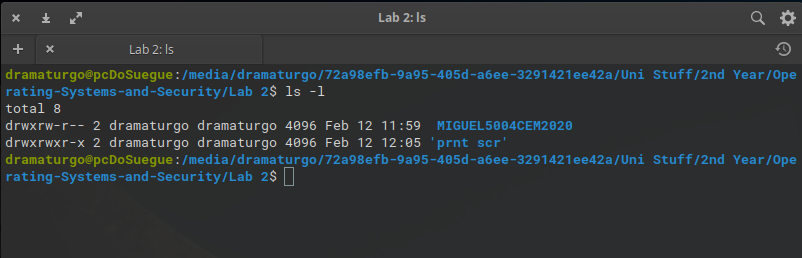
# Lab Activity 2 – Linux Command Line (Commands and outcomes from a series of small tasks that require use of a number of Linux commands)

## Tasks – Files

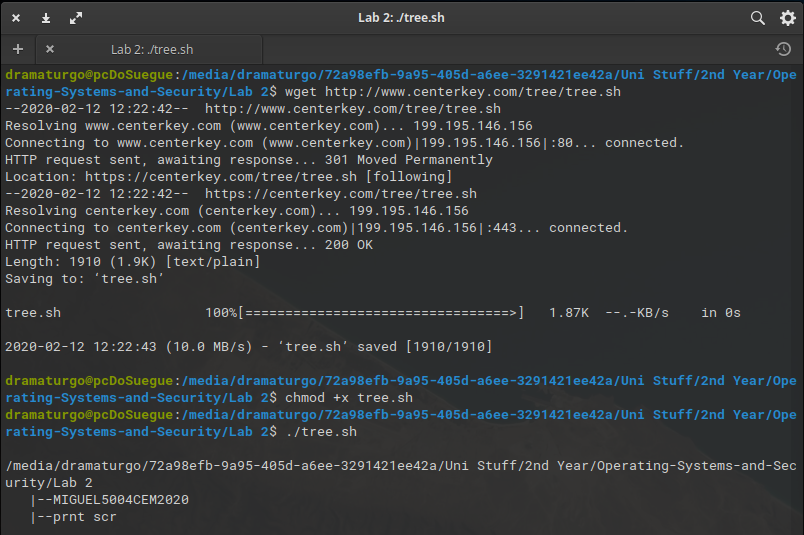
### Create a directory in your /home/5004CEM\_Session/<username> directory with a name made up of you second name followed by 5004CEM and the year (mine would be **ELSHAW5004CEM2020**). Make the directory **read/write/executable** only for you, **read/write** for your groups and **read** only for others.

 Fig 3 – Task a) directory creation and permissions manipulation

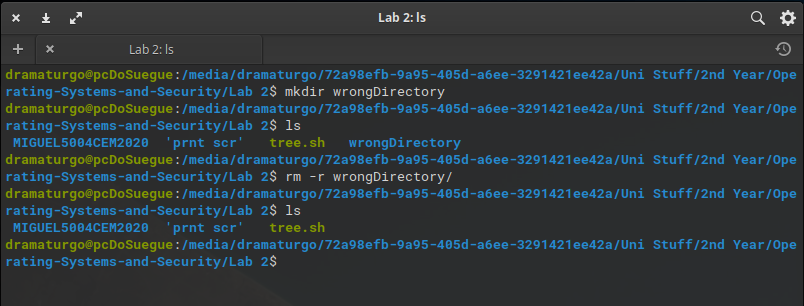
### Show evidence of this using the appropriate version of the **ls command**.

 Fig 4 – Task b) proof of task a) completion

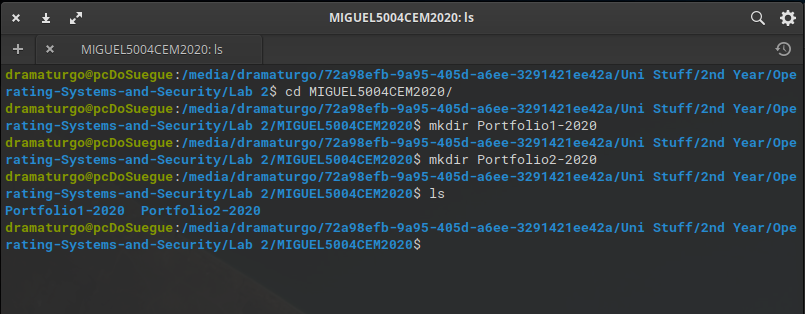
### Download the script http://www.centerkey.com/tree/tree.sh to your home directory using wget and make the file executable.

 Fig 5 – Task c) wget file download, executing the downloaded file

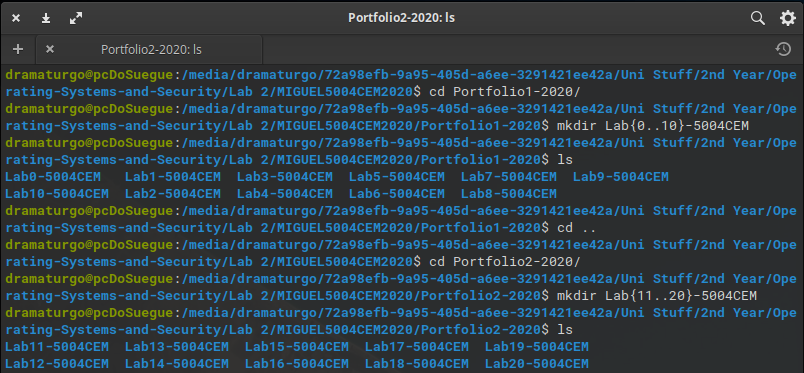
### Create a directory called **wrongDirectory**. You release it is not what you wanted so delete it.

 Fig 6 – Task d) wrong directory creation and deletion

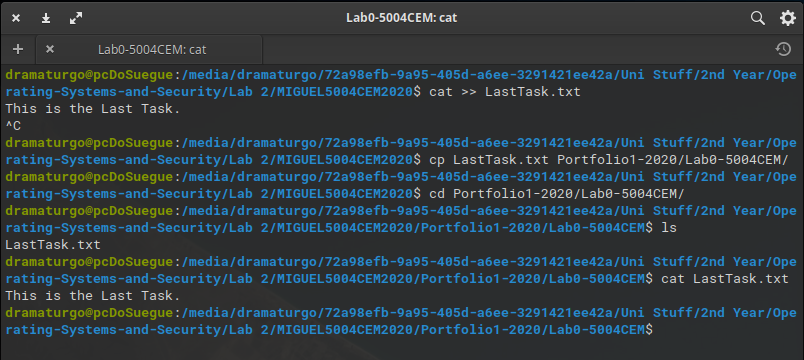
### Create Portfolio1-2020 and Portfolio2-2020 directories in the directory you created in part a.

 Fig 7 – Task e) portfolio directories creation

### Create numbered directories in the Portfolio1-2020 Directory (Lab0-5004CEM to Lab10-5004CEM) and in the Porfolio2-2020 Directory (Lab11-5004CEM to Lab20-5004CEM).

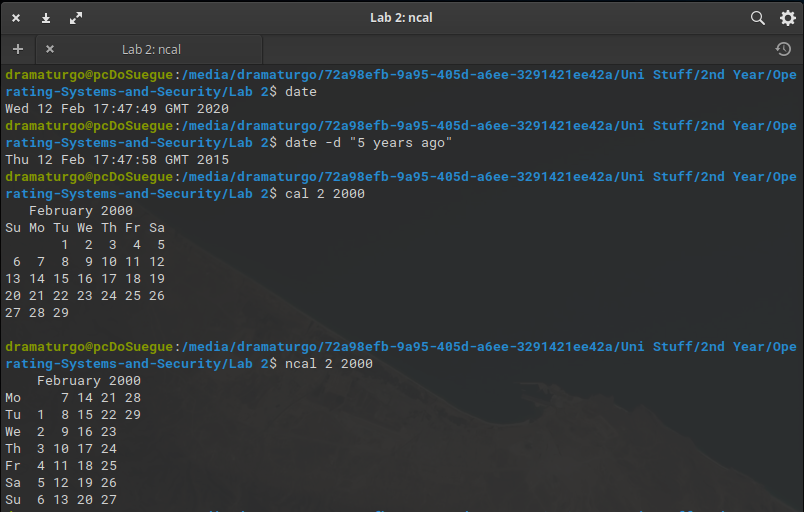
 Fig 8 – Task f) multiple directories creation

### In <YourSecondName>5004CEM2020 directory create a text file called **LastTask.txt** and then using the appropriate Linux command copy this document into Directory Lab0-5004CEM.

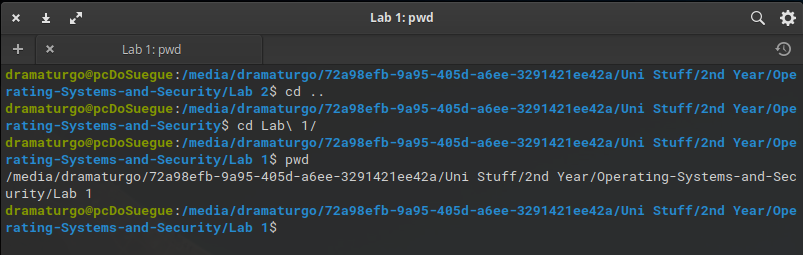
 Fig 9 – Task g) file copying to other directories

## Linux Commands – Mixed

### Using the date command show todays date and the time and date 5 years ago. Using the cal command show the month that you were born. Change this calendar to make Monday the first day of the week.

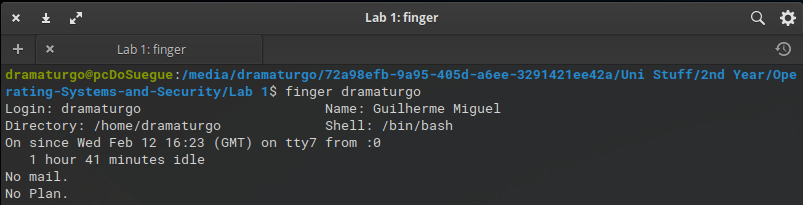
 Fig 10 – Task a) date and calendar commands

### Move into the lab1-5004CEM directory and use the appropriate command to show the current directory.

 Fig 11 – Task b) pwd command

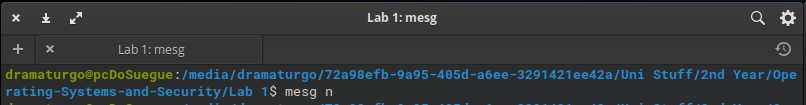
### Display the time when a user (ab0487) last login, the user’s home directory, and the user account’s full name.

Since I am not doing this exercise on my personal machine I will just present the same asked but with my personal user.

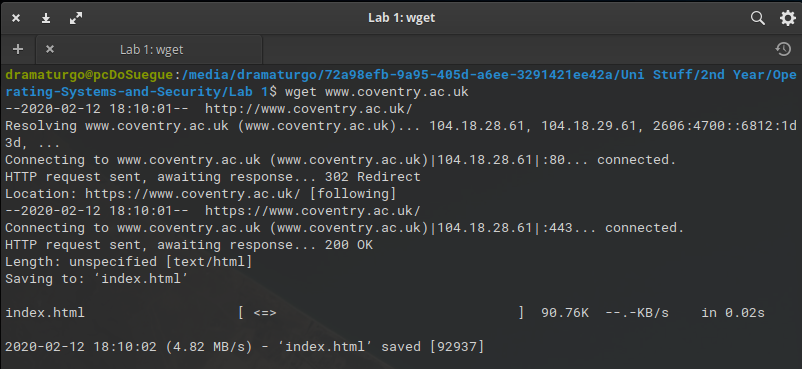
 Fig 12 – Task c) user login times

### Find out how to prevent the effects of talk, write and wall from interrupting you. What command can you use?

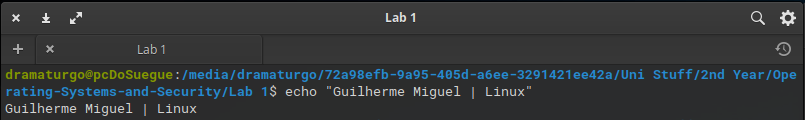
The mesg command blocks all of these notifications.

 Fig 13 – Task d) notification block

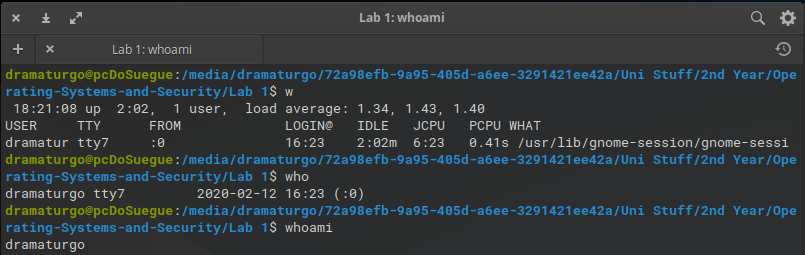
### Show the command to verify that www.coventry.ac.uk exists and can accept requests.

 Fig 14 – Task e) http requests

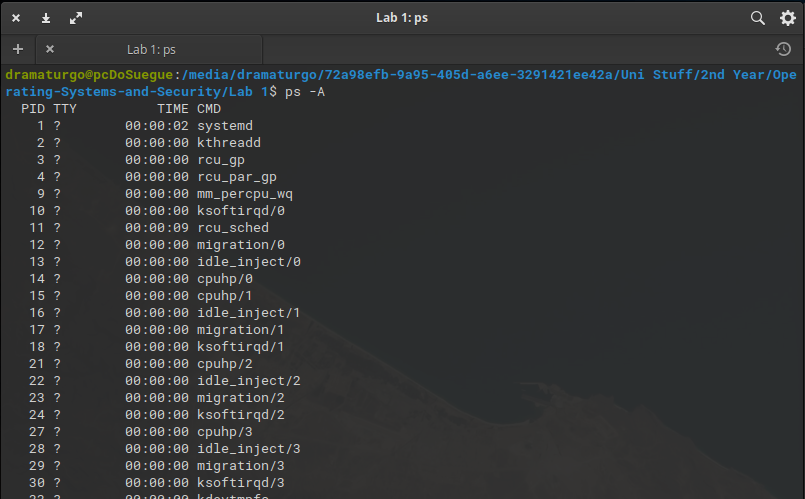
### Display your name and favourite programming language on the screen using the echo command.

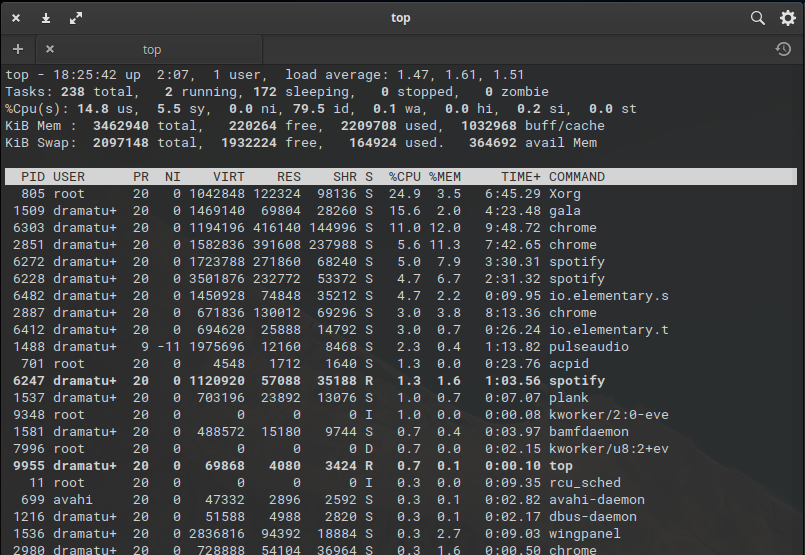
 Fig 15 – Task f) echo command

### Find out how you can display your username on the screen and at least two ways to display who is logged on.

 Fig 16 – Task g) w, who and whoami commands

### Use two ways to list the processes that are running.

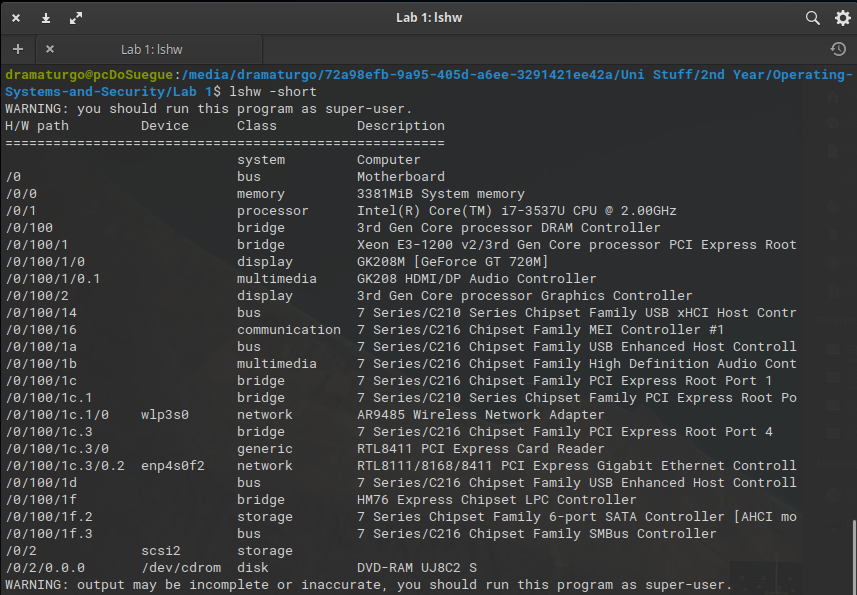
 Fig 17.1 – Task h) process list with ps –A

 Fig 17.2 – Task h) process list with top

### What are the differences between the Linux commands copy (cp), rename and move?

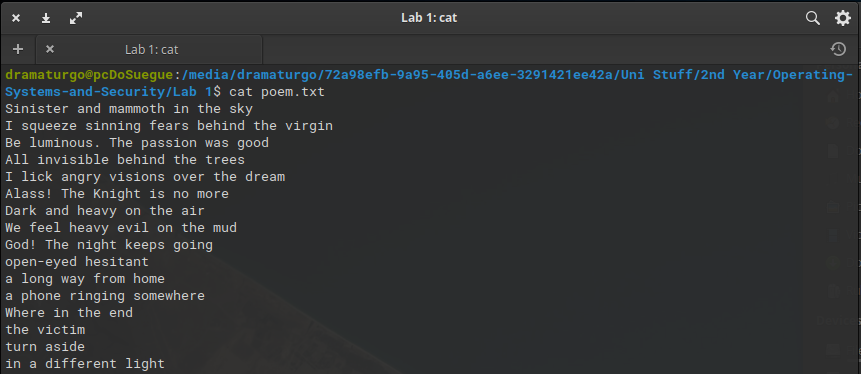
Rename only renames the file, so the file is still in the same directory. Copy copies the file to another directory, therefore there will be two files exactly identical in two different directories, where move is different since it deletes the original copy of the file by just moving it to another directory.

### With a single command, how would you get systems information such as processes, memory, paging and CPU activity?

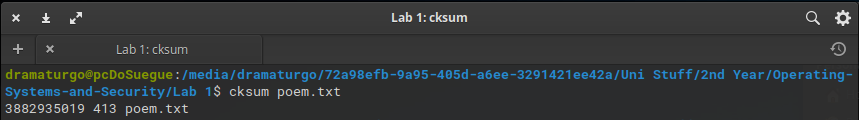
 Fig 18 – Task j) lshw command, -short to just get a short summary

## Tasks – Document Manipulation

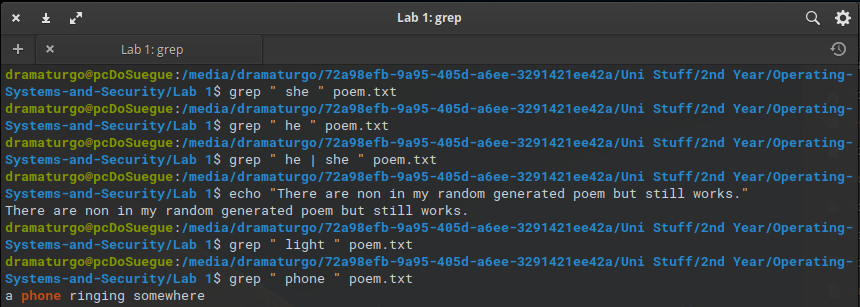
### Use **cat** to show the contents of the file.

 Fig 19 – Task a) cat to open text files

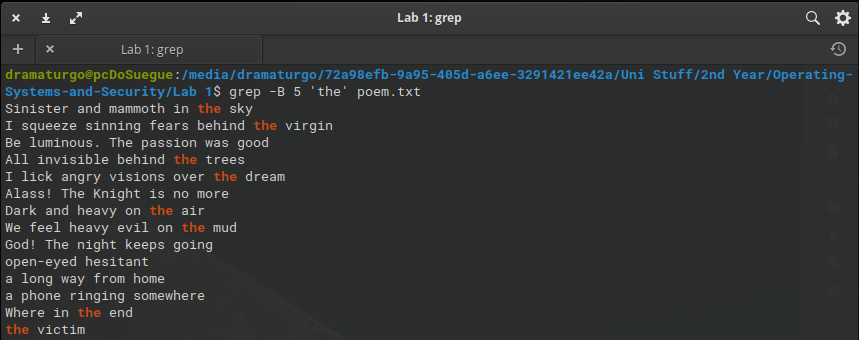
### Use an appropriate command to display the CRC checksum and byte count of the file.

 Fig 20 – Task b) cksum to check byte count and CRC checksum

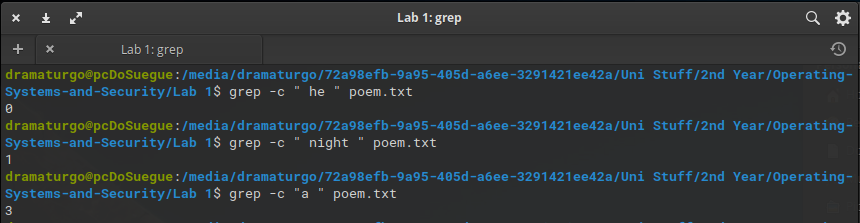
### Use **grep** to show only lines not containing the words "she" or "he". Lines contain both “she” and “he” should be shown.

 Fig 21 – Task c) grep to search certain strings

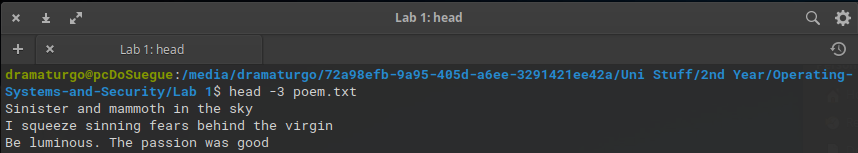
### Use **grep** to show the 5 lines above a line containing the text ‘the’.

 Fig 22 – Task d) grep to select lines

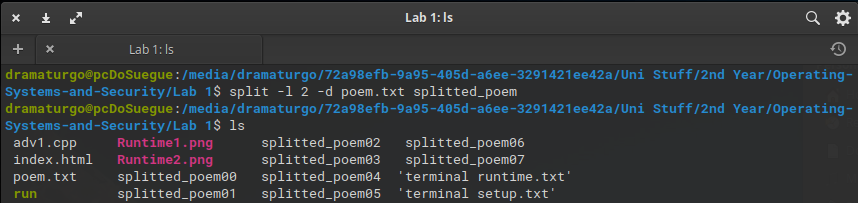
### Using Linux commands you should count the lines containing “she” and “he” but not both and display the line numbers that “she” and “he” but not both appear on in the original document.

 Fig 23 – Task e) grep to count number of word appearances

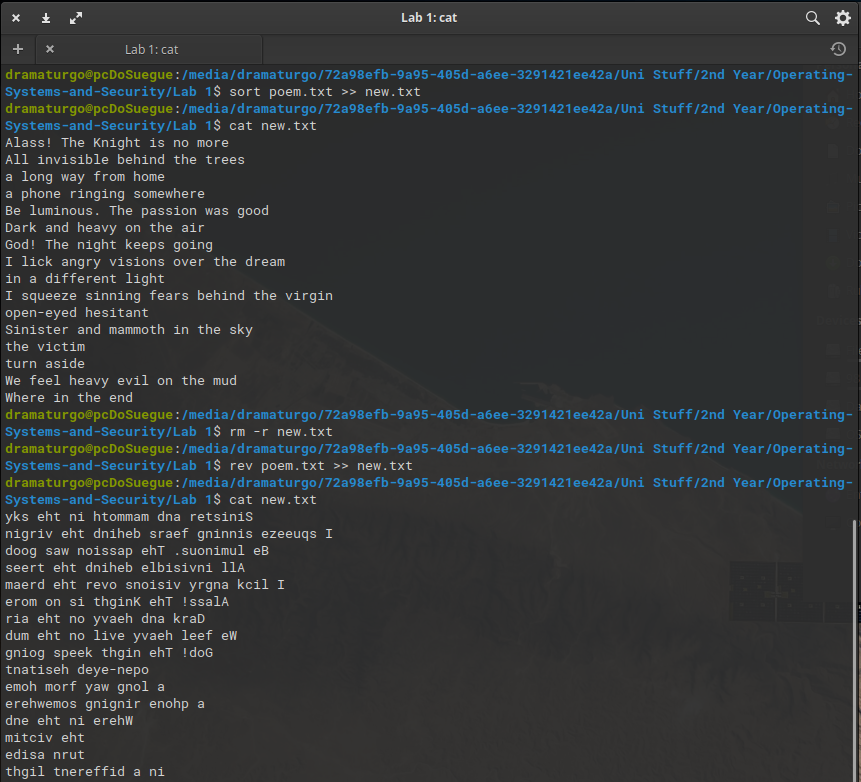
### Find a command to list the top 3 lines of the **poem.txt** file and then the bottom line of these top 3.

 Fig 24 – Task f) head to get parts of a file

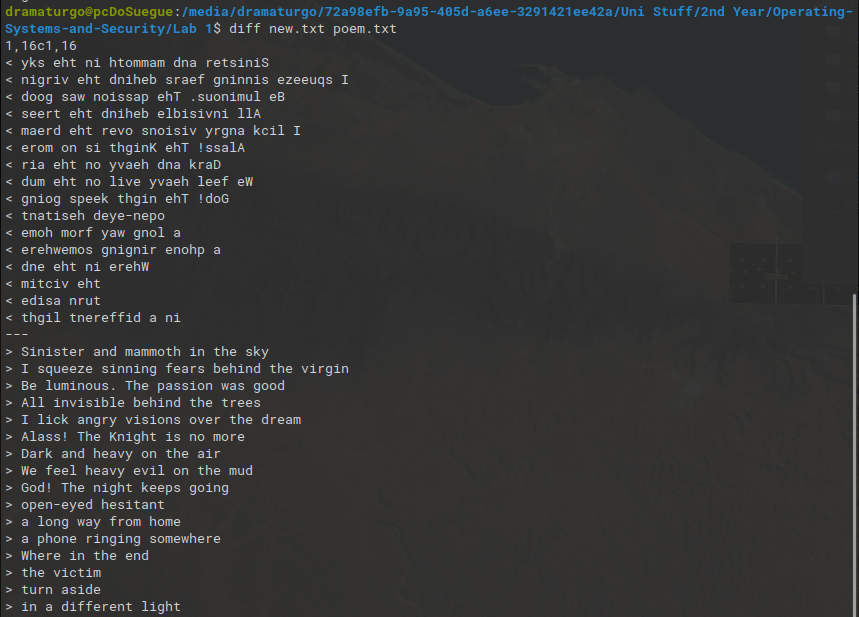
### Find a command to split the **poem.txt** file into different files each containing 2 lines.

 Fig 25 – Task g) split to split a txt file

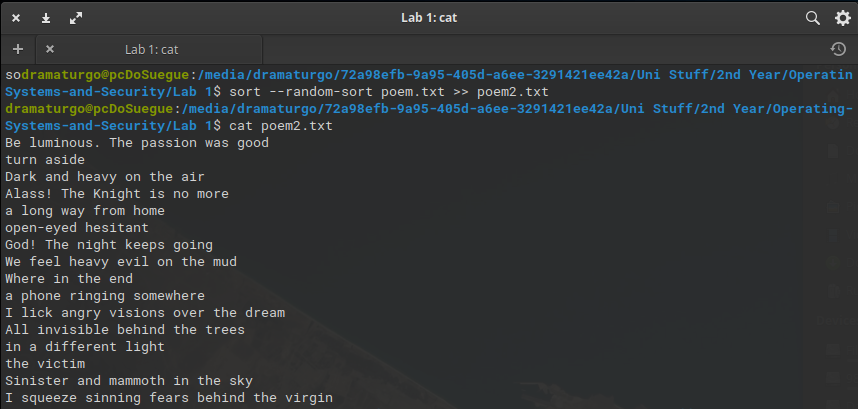
### Use **sort** and **rev** to reverse the sorted contents of poem.txt and append the output to poem2.txt.

 Fig 26 – Task h) sort and rev commands

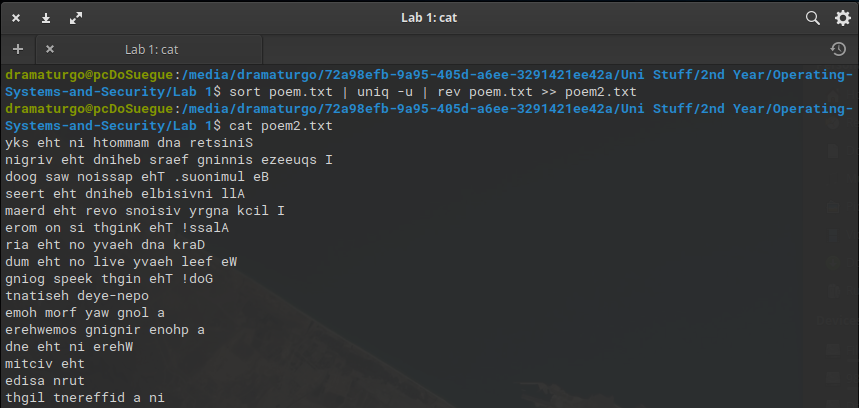
### Use at least two appropriate Linux commands to compare these two files (poem.txt and poem\_name.txt) and see if they are the same.

 Fig 27 – Task i) diff to compare text files

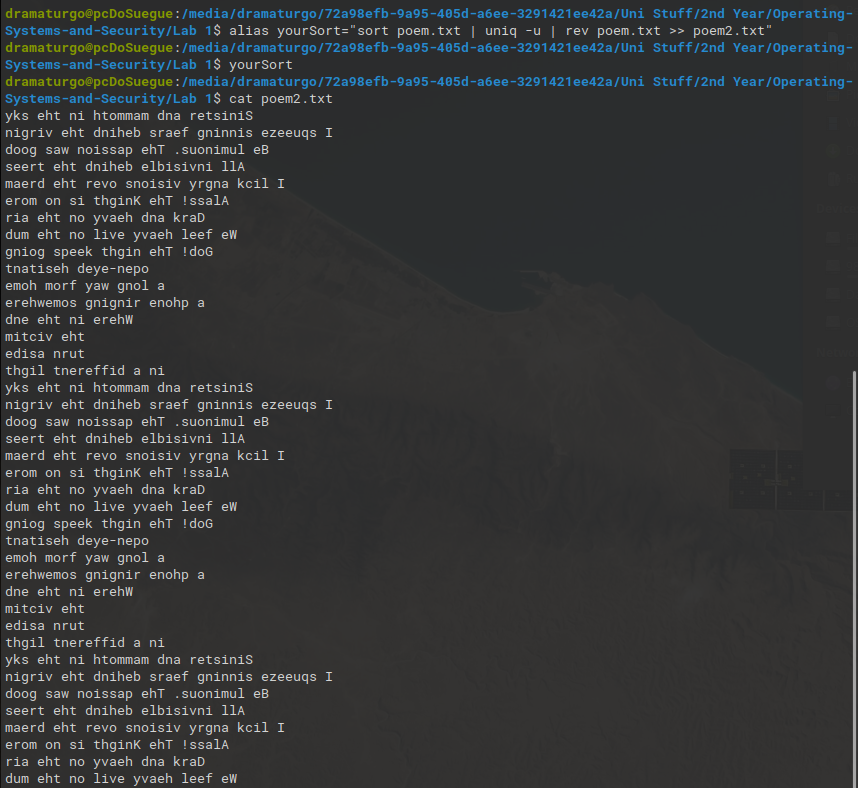
### Use **sort** to sort the content of poem.txt file in a random order and redirect the output to a new file called **poem2.txt**.

 Fig 28 – Task j) sort with –random-sort to randomize the file

### Sort the **poem.txt** file, remove the duplicates and reverse the sorted contents and append the output to **poem2.txt**.

 Fig 29 – Task k) sort rev and uniq in same line

### Create an **alias** so rather than having to type the full command for k) you can type **yourSort**.

 Fig 30 – Task l) alias creation

# Lab Activity 4 Bootloader

## Brief description of the Lab activity and what you did

[Brief Description of activity]

## Boot pragma Linux with bochs

[Screenshot or script showing how you compiled and created the Linux image and used Bochs to simulate bootloader]

## Make a bootloader that displays your student details and diamond

### Commented bootloader code to display your student details and diamond

[Commented code here]

### Output from bochs showing student details and diamond

[Output from bochs show student details and triangle]

# Lab Activity 7 Buffer

## Explain what the code below does.

[First fit, Best fit and Worst fit allocation]

## Code to show error outputs.

[Commented code showing the output]

## Code to create statistics on the document.

[Commented code and examples of the program running]

## Code to compare the documents.

[Commented code and examples of the program running]

## Code to create summary.

[Commented code and examples of the program running]

# Lab Activity 8 Cache Buffer

## Brief Description of Cache Buffer Activity

[Brief Description here]

## Commented implementation of the return\_character function

[Comment code of the return\_character function here]

[Output of running code here]

## Comment updated code to show that each byte is being read, and when the buffer is being refilled.

[Comment code outlining your changes here]

[Output of running code here]

## Commented updated code showing the required statistical information as well as how many times the words ‘Iran’, ‘Tehran’ and ‘email’ appear.

[Comment code outlining your changes here]

[Output of running code here]

# Lab 10: The Cache Buffer from week 8 with system calls

## Brief description of the activity

[Brief description here]

## Changes the cache\_handle library from using the fopen, fread, fclose functions to the system call versions open, read, close

[Commented code outlining your changes to the .h and .c files here]

[Output from running code here]

## Changes cache\_handle library to remove (as far as possible) the effects of caching on the library.

[Commented code outlining your changes to the .h and .c files here]

[Output from running code here and if possible prove not using cache here]

# Lab Activity 13 – Job control

## Description of the activity

[Descriptive here]

## Menu System

[Commented menu fork code]

[Output from code]

# Lab Activity 14 – Linux command-line manipulation of processes

## Paragraph on disown and an alternative.

[paragraph here]

## Description and example of using watch command

[description and example of watch here]

## Process manipulation

### Example(s) of how to start process

[How to started process]

[Screenshot or section of Linux script showing process  
 starting]

### Example(s) of how to suspend process

[How to suspended process]

[Screenshot or section of Linux script showing process suspended]

### Example(s) of how to run process in background

[How to run process in background]

[Screenshot or section of Linux script showing process running in background]

### Example(s) of how to run process in foreground and bring from background

[How to run process in foreground and bring from background]

[Screenshot or section of Linux script showing process running in foreground or brought from background]

### Example(s) of how to kill a process

[How to kill process]

[Screenshot or section of Linux script showing process killed]

# Lab Activity 15 IPC and Synchronisation

## Brief description of activity

## Modified semaphore example code so that the two processes output the song.

[Comment code here]

[Screenshot(s) showing code working]

## Modified code to write Liza part to stderr and redirect the two parts to a file.

[Changes to code above here with comments]

[Screenshot(s) showing code working]

# References