COMP2113 Programming Technologies

ENGG1340 Computer Programming II

**Module 1 Checkpoint Exercise**

Name: Zhao Kexin

University ID: 3035845368

**Instructions:**

For each single question or each group of questions in the Checkpoint exercise, please type your answer right after the question in this Word document. Please refer to the example below.

Checkpoint 0:

What is the meaning of the command “date”?

Ans: The “date” command prints the current date of the current machine

**Checkpoint 1.1**

Now, let’s try to answer the following questions. Although you haven’t been taught the meaning of the following commands, you can display the manual page of these commands and learn their meanings by yourself

1. Why do we need to learn command line although we can use a GUI to control a computer?
2. What is the meaning of **ls -t**?
3. What is the meaning of the command **pwd**?
4. What is the meaning of the command **rm**?
5. What is the meaning of the command **mv**?
6. Suppose that the **fileA** does not exist in your present working directory, what is the meaning of the command **touch fileA**?
7. What is the meaning of the command **tar**?
8. What is the command for creating an archive **files.tar** from two files named **fileA** and **fileB**?

Ans:

1. The command line offers more control, speed, and efficiency, especially for automation and remote systems.
2. The ‘ls -t’ command lists files sorted by modification time, with the most recent first.
3. The ‘pwd’ command prints the current working directory.
4. The ‘rm’ command removes files or directories.
5. The ‘mv’ command renames files or directories.
6. The ‘touch fileA’ command creates an empty file named ‘fileA’.
7. The ‘tar’ command create, extracts, or manipulates archive files.
8. **tar -cvf files.tar fileA fileB**

**Checkpoint 1.2a**

Assume we have logged in Ubuntu and started a bash shell. The current directory is the home directory, i.e., ~ . We want to perform the following tasks sequentially. For each of the tasks below, please state the shell command(s) used to perform it.

1. Create a new subdirectory “***assignments***” under ~.
2. Create a new subdirectory “***assignment 1***” under “***assignments***”. (Note that we are creating one subdirectory “***assignment 1***” but not two subdirectories “assignment” and “1” )
3. Remove the directory “***assignments***” and all its subdirectories.

Ans:

1. **mkdir ~/assignments**
2. **mkdir ~/assignments/"assignment 1"**
3. **rm -r ~/assignments**

**Checkpoint 1.2b**

**[Self-learning question]** - You need to search for the information on the Internet to answer this question.

There is another way to modify the permission, which is called the Absolute mode.

* 1. Explain the meaning of chmod 666 hello.txt
  2. Explain the meaning of chmod 700 hello.txt
  3. What is the chmod command, in absolute mode, to set the following permission for *hello.txt*?

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **User** **permissions** | | | **Group Permission** | | | **Other permission** | | |
| r | w | x | - | w | - | r | - | - |

* 1. The administrator says that “One does not simply 777 their entire server”, explain what the problem is if we chmod 777 for all the files.

Ans:

1. **chmod 666** sets the file permissions to **rw-rw-rw-**. This means: The user, group, and others all have read and write permissions, but no execute permissions for the file ‘hello.txt’.
2. **chmod 700** sets the file permissions to **rwx------**. This means: Only the user has read, write, and execute permissions. The group and others have no permissions for the file ‘hello.txt’.
3. **chmod 741 hello.txt**
4. **chmod 777** gives read, write, and execute permissions to everyone (user, group, and others). This is a major security risk because: It makes the system vulnerable to unauthorized access, data corruption, and exploitation, especially on a multi-user or public server. It's generally advised to limit permissions based on necessity.

**Checkpoint 1.3**

Now you may have a doubt: I understand how **diff** works, but why the output is claimed to be the difference between the two files?

Consider the two files below:

|  |  |
| --- | --- |
| $cat question1A  Apple  Boy  Cat  Dog  Egg | $cat question1B  Boy  Cat  Egg |

Note that file **question1B** is created by removing “Apple” and “Dog” from the file **question1A**.

1. What will be the output if we execute the following command (Please try to think about the output before trying it in the shell)? Please explain your answer.

|  |
| --- |
| $diff question1A question1B |

1. What will be the output if we execute the following command (Please try to think about the output before trying it in the shell)? Please explain your answer.

|  |
| --- |
| $diff question1B question1A |

Ans:

1. 1d0

< Apple

4d2

< Dog

1d0 means to delete line 1 in question1A, and the file will then be in sync starting at line 0.

4d2 means to delete line 4 in question1A, and the file will then be in sync starting at line 2.

1. 0a1

> Apple

2a4

> Dog

0a1 means to add a line after line 0 in question1B, the line to be added is line 1 of question1A.

2a4 means to add a line after line 2 in question1B, the line to be added is line 4 of question1A.

Checkpoint 1.4

This is a challenging exercise! You need to understand the shell commands and the techniques introduced in the previous sections to work on this task.

The following C++ program *gen4.cpp* reads in a 4-character string from the input and generates all possible permutations from the 4 characters.

|  |
| --- |
| //gen4.cpp  #include <iostream>  #include <string>  int main() {  std::string s;  std::cin >> s;  for (int i = 0; i < s.length(); i++) {  for (int j = 0; j < s.length(); j++) {  for (int k = 0; k < s.length(); k++) {  for (int l = 0; l < s.length(); l++) {  if (i != j && i != k && i != l && j != l && j != k && k != l) {  std::cout << s[i] << s[j] << s[k] << s[l] << std::endl;  }  }  }  }  }  return 0;  } |

To compile *gen4.cpp*

|  |
| --- |
| $ g++ gen4.cpp -o gen4 |

The input of the program should be stored in the file *gen4\_input.txt* with the following content.

lopo

***gen4\_input.txt***

1. Give **ONE** command (one line of command(s)) to run the *gen4* with *gen4\_input.txt* as input and redirect the result to a file named *gen4\_output.txt*.

Hints:

|  |
| --- |
| $ *[your\_command]*  $ cat gen4\_output.txt  lopo  loop  lpoo  lpoo  …  $ wc gen4\_output.txt  24 24 120 gen4\_output.txt |

\*The output file should contain all permutations of the letters ‘l’, ‘o’, ’p’ and ‘o’. There should be 24 permutations in total.

Ans: ./gen4 < gen4\_input.txt > gen4\_output.txt

1. Give **ONE** command to sort the words in *gen4\_output.txt* in alphabetical order, and then also remove the adjacent duplicate lines and finally store the result in a file named *sort\_uniq.txt*.

Hints: Consider the command **uniq**

|  |
| --- |
| $ [your command]  $ cat sort\_uniq.txt  loop  lopo  lpoo  olop  olpo  oolp  oopl  oplo  opol  ploo  polo  pool  $ wc sort\_uniq.txt  12 12 60 sort\_uniq.txt |

\*There should be 12 unique words total.

Ans: sort gen4\_output.txt | uniq > sort\_uniq.txt

1. Give **ONE** command to check the spelling in *sort\_uniq.txt* and store the misspelled words into another file named *misspell.txt*.

Ans: aspell list < sort\_uniq.txt > misspell.txt

1. Now *sort\_uniq.txt* contains all distinct generated words, and *misspell.txt* contains all misspelled words. The differences between the two files are the meaningful 4-character words. Give **ONE** command to return the correctly spelled words as shown below:

|  |
| --- |
| $ [your command]  < loop  < polo  < pool |

Hints: Consider the command **diff** and **grep**.

Ans: diff sort\_uniq.txt misspell.txt | grep '^<'

Checkpoint 1.5

Consider the file *question1.txt*.

2011111111,John,M,98

2011222222,Marry,F,85

2011333333,Sally,F,85

2012111111,Kit,M,86

2012222222,Ben,M,97

2012333333,Smitty,F,92

2012444444,Jolly,F,93

2012555555,Ken,M,100

Figure 1 question1.txt

1. Give ONE command to return the lines that contain the record of Kit

Hints:

|  |
| --- |
| $ [Your command]  2012111111,Kit,M,86 |

Ans: grep 'Kit' question1.txt

1. Give ONE command to find the students with UID begin with “**2012**” (i.e., To find the lines that begin with 2012)

Hints:

|  |
| --- |
| $ [Your command]  2012111111,Kit,M,86  2012222222,Ben,M,97  2012333333,Smitty,F,92  2012444444,Jolly,F,93  2012555555,Ken,M,100 |

Ans: grep '^2012' question1.txt

1. Give ONE command to return the lines that contain the record of the students who are both:

* UID start at **2012**, and
* Name starts with the characters **J** or **S**

Hints:

|  |
| --- |
| $ [Your command]  2012333333,Smitty,F,92  2012444444,Jolly,F,93 |

Ans: grep '^2012.\*[JS]' question1.txt