# Lab 8
# Uniq, grep, diff, ping commands.
# Python code running.
# While in Bash.

Read carefully the instructions before performing!!!

Submission in pairs : 13/01/25 for the groups of Monday
16/01/25 for the groups of Thursday

## Linux Commands

1. **grep** = searches for PATTERNS in each FILE

**- grep OPTIONS PATTERN file**

**Options:**

-i : The search ignores differences between uppercase and lowercase letters.

-v: Display only lines that do not match the search pattern.

-w: Match only whole words.

-r: Search through subdirectories for the pattern.

-c: Count how many lines match the search pattern.

-n : Show the line number for each matching line.

**Examples:**

```
>cat foo1
My lovely name is Robert
My second name is Cohen
>cat foo2
The Name of my mother is Miriam
And the name of my father is Michael
>cat foo3
The name of my mother is different
And the name of my father is Michael too
>grep ‘Robert’ foo1
My lovely name is Robert
> grep ‘name’ foo*
```

```
foo1:My lovely name is Robert
foo1:My second name is Cohen
foo2:And the name of my father is Michael
foo3:The name of my mother is different
foo3:And the name of my father is Michael too
```

#Ignores differences between uppercase and lowercase letters

```
>grep -i ‘name’ foo*
foo1:My lovely name is Robert
foo1:My second name is Cohen
foo2:The Name of my mother is Miriam
foo2:And the name of my father is Michael
foo3:The name of my mother is different
foo3:And the name of my father is Michael too
```

#Display only lines that do not match the search pattern.

```
>grep -v 'my mother' foo*
foo1:My lovely name is Robert
foo1:My second name is Cohen
foo2:And the name of my father is Michael
foo3:And the name of my father is Michael too
```

# Counts how many lines match the search pattern

```
> grep -c 'my mother' foo*
foo1:0
foo2:1
foo3:1
```

#Show the line number for each matching line.

```
> grep -n 'my mother' foo*
foo2:1:The Name of my mother is Miriam
```

foo3:**1**:The name of my mother is different

2. **uniq**= reports or filters out the repeated lines in a file

- **uniq OPTIONS file**

NOTE: Can be used only after the **sort** command.

**Options:**

-i : Ignore differences in case when comparing lines.

-u: Only output lines that are unique in the input.

-d: Only output lines that are repeated in the input.

-c: Count the number of occurences.

**Examples:**

```
> cat unifoo2
I like music
I like music
I like music
I like music of Bethoven
I like music of Bethoven
> uniq unifoo2
I like music
I like music of Bethoven
> uniq -c unifoo2
    3 I like music
    2 I like music of Bethoven
```

```
>cat unifoo1
I like music
I like music
I aike music
I do not like music
I aike music
```

```
I do not like music
```

> **uniq unifoo1** – Does not work properly, since  unifoo1 was not sorted before

```
> sort unifoo1 | uniq
I aike music
I do not like music
I like music
```

3. **diff** = Compare two files and suggest which changes should be done in file1 to become identical to file2.

**- diff [option] file1 file2**

We concentrate on **diff -u** file1 file2, which shows the differences in a Unified Mode**.**

- If a line is unchanged, it is prefixed by one space.

- If a line needs to be changed, it is prefixed by a symbol. The symbols indicate:

  •`**+**`: A line in the second file to be added to the first file for identical results.

  •`**-**`: A line in the first file to be deleted for identical results.

**Example:**

```
> cat difffile1
Apple
Orange
Banana
Watermelon
Cherry
> cat difffile2
Orange
Peach
Apple
Banana
Melon
Cherry
```

```
> diff -u difffile1 difffile2
--- difffile1	2024-03-10 14:21:09.951906191 +0200
+++ difffile2	2024-03-10 14:21:38.584066666 +0200
@@ -1,5 +1,6 @@
-Apple
 Orange
+Peach
+Apple
 Banana
-Watermelon
+Melon
 Cherry
--------------------------------------
```

In the above output:

1). The first file is indicated by `**---**`, and the second file is indicated by `**+++**`.

2). The first two lines provide information about file 1 and file 2, including the modification date and time.

3). After that, `@@ -1,5 +1,6 @@` denotes the line range for both files. In this case, it represents lines 1 through 5 in first file and lines 1 through 6 in second file.

4). The subsequent lines represent the contents of the files with specific indicators:

- •Unchanged lines are displayed without any prefix.
- •Lines in the first file to be deleted are prefixed with -.
- •Lines in the second file to be added are prefixed with+.

4. **python** = runs the python file

- python filename

**Example:**

```
> python file.py
```

5. **Ping** = is a tool which is used to test whether a particular host is reachable across an IP network. A Ping measures the time it takes for packets to be sent from the local host to a destination computer and back.

**- ping OPTIONS “hostName”**

**Example:**

```
> ping -c 1 "google.com"

PING google.com (142.251.142.206) 56(84) bytes of data.

64 bytes from tlv03s02-in-f14.1e100.net (142.251.142.206): icmp_seq=1 ttl=114 time=16.1 ms

--- google.com ping statistics ---

1 packets transmitted, 1 received, 0% packet loss, time 0ms

rtt min/avg/max/mdev = 16.108/16.108/16.108/0.000 ms
```

**Bash**

**While loop**

```
while [ condition ]
do
  command1
  command2
  command3
done
```

```
while read var
do
        command1
        command2
        command3
done<name_of_file_to_read
```

**Example whileExample.sh**

```
#!/bin/bash

###########################################

#Example1 - Standard While

i=1

while [ $i -lt 6 ]

do

     echo "Create progect $i"

     if ! [ -d Project${i} ]

     then

           mkdir Project${i}
```

```
    fi
    let i++
done
########################################
```

<u>#Example2 - Reading a file line by line (from file foo1)</u>

```
Example2 - Reading a file line by line
LINE_NUM=1
while read -a LINE    #LINE becomes an array. Now it is possible to
                       #read each element of the LINE separetely
do
       echo "${LINE_NUM}: $LINE" #read the next line
        echo "LINE0=${LINE[0]}, ${LINE[1]}"
       let  LINE_NUM++
done < foo1
```

**Procedures**

1.Create directory Lab8.

2. a). Create file StudentsFile which contains the First name, Second name, year of birth and GPA (Grade point average) of at least 7 students. The fields will be separated by spaces.

b). Write a function calcStdNames which receives as a parameter StudentsFile and prints the first names of the students and number of the students with this name. The output should be sorted by the name in alphabet order.

For example, if StudentsFile is:

```
Moshe Porat 1990 85.4
Irit Rup   2002 74.2
Moshe Moshonov 1998 56.3
Olga Ivanova 1999 82.1
Muhamad Haib 2001 86.7
Moshe Perez 2006 92.3
Muhamad Musa 2005 75.3
```

Then calcStdNames will print

1 Irit

3 Moshe

2 Muhamad

1 Olga

c). Write a function CalcGPA which receives as parameters StudentsFile  and number YEAR in a range 1900-2010 (check that input is valid) and prints list of the students from StudentsFile which were born during YEAR year. The list will be sorted by the GPA.

d). Write a script **studentChecker.sh** which receives as a parameter StudentsFile, calls the function calcStdNames and puts the results into a file Names.txt. Then the script calls the function  CalcGPA and puts the results into a file GPA.txt.

**3**. Write a script **checkPythonFile.sh** that accepts two parameters: a python file and a text file.

The script runs the python file and puts the results into file named as the python file, without .py plus Result.txt and compares the result with a given text file. If there are no differences between the files, the script prints “Your python code is correct”. Otherwise, it prints ”Your python code is wrong.”

**Instructions:**

1). Download two files (can be found in Moodle): python file **isPalindrom.py** (which checks if the given string is a palindrom) and text file **isPaliRes.txt**.

2). For example, running the script with two parameters

> ./ checkPytonFile.sh  **isPalindrom.py  isPaliRes.txt**

will create a file  **isPalindromResult.txt** and compare it with **isPaliRes.txt.** Since there are differences between these files, the script will print: Your python code is wrong.

3). Put your attention that the input python and text files should be general. The files which are given above are only an example.

**4.** Write a script named **UniqD.sh** which accepts a file (the file can be unsorted) and implements the Linux command uniq -d.

**Note:** the commands uniq (with any flags) and sort -u can not be used in the script.

**5.** Hi-tech company Moogle has decided to open a new branch in Haifa, announced a day of interviews at the Faculty of Computer Science in Tel-Hai.

There are several interviewers, including the Moogle manager.

Each interviewer gives a score for the interviewee between 0 and 100.

The final grade will be determined by weighting all the grades.

The manager has a right of veto: an interviewee to whom the manager has given a NO can not be accepted for the job.

At the end of the interviews, each interviewer prepares a file in the following format about the people he interviewed . Here there are 4 fields separated by spaces. The name of the file is <interviewer's name>.grades.

**For example,**

**File eyal.grades:**

```
031243129 Moshe Levi 60
444422267 Shimon Cohen 90
555782311 David David 100
```

**File Moogle.grades:**

```
031243129 Moshe Levi 57
555782311  David David -NO-
444422267  Shimon Cohen 80
```

Write a script called **BestToWork.sh** that checks these .grades files and return the best group of interviewees. The script will receive as a parameter the number of places and will return such a number of outstanding (with a highest score) interviewees sorted by degree of success.

If there are not enough interviewees who have been interviewed / passed the interview, the script will print a message: "Not enough interviewees"

For example, for the running:

```
>BestToWork.sh 2
```

On the two files in the example, we get:

```
1) 444422267 Shimon Cohen
2) 031243129 Moshe Levy
```

**Notes:**

- Assume that the existing input files are correct

- It can be assumed that the same interviewers interviewed all the people

**Submission:**
Create one compressed file named Linux_Lab8_id1_id2.tar,
where id1 and id2 must be changed to the id numbers of the two partners.

This file will include the files **studentChecker.sh, checkPythonFile.sh, UniqD.sh, BestToWork.sh.**

Submit only the file Linux_Lab8_id1_id2.tar in Moodle and only by one of the partners.

**Appeals:**

Can be submitted by e-mail to Elad mail@eladhuttner.net within a week from the date of publication of the grades.