PPA1

> Move all the .txt files present in the current directory to the directory level1 present inside the current directory, do not move any other files other than .txt files. The operation should not change your current working directory at the end. Hint: Can be done with a single command.

```
mv *.txt ./level1
```

PPA2

> Create a folder in the current working directory named folders and store a file named files.txt containing the list of files in the long format(i.e. using ls -l) in the parent directory of folders i.e in the current working directory. The operation should not change your current working directory at the end.

```
mkdir folders
ls -l > ./folders/files.txt
```

PPA1

> Create a file **documents.txt** containing all the possible file names in the format **file_XYZ.txt** where **X** is a lower case alphabet, **Y** is also a lower case alphabet and **Z** is a number between 0 and 4. Few examples of file names in this format are **'file_dh3.txt', 'file_sd1.txt', 'file_ja0.txt', 'file_at2.txt'**. The file names in **documents.txt** should be separated by a single space.
>
> **Hint:** Use echo to solve this with a single command

```
echo file_{a..z}{a..z}{0..4}.txt > documents.txt
```

PPA2

> **encoding-key** is a file located at the path **/encryption/two-level/binary/positive-offset/**(directory 'encryption' is located in current working directory) . The file **encoding_key** is updated often and shared between multiple users. This file is important to you and you are worried that the file could be deleted by mistake. Create a file **ek** in the current working directory, such that it is always in sync with the contents of file **encoding-key** and if **encoding-key** gets deleted by any chance the content in it should be available in file **ek**.

```
ln ./encryption/two-level/binary/positive-offset/encoding-key ek
```

PPA1

> List(in long format, use **ls -l**) all the **.txt** files in the current working directory and redirect the output to a file named **textFiles.txt** and also print 'found' to the terminal(without quotes, do not print anything else).
>
> If no **.txt** file exists redirect the error of your command to the file **noFiles.txt** and do not print anything.
>
> **Hint**: Make use of redirection to file and operators to write solution in one line.

```
ls -l *.txt > textFiles.txt 2> noFiles.txt && echo found
```

PPA2

> Given a shell variable **month** supposed to contain a string value corresponding to some calendar month. Use the **cal** command to create a file named as **X.txt** where **X** is the string value in the variable **month**. Your command should also create a file named **error.txt** that should contain the error message if the string **month** does not correspond to any calendar month. Create all the files in the current working directory.
>
> For example:
>
> If the variable **month** contains the string "nov", your solution should create a file named **nov.txt** containing the calendar of november month and **error.txt** should be empty. And if the variable **month** contains the string "garbage", your solution should create a file named **error.txt** containing the error from cal command and **garbage.txt** should be empty.

```
cal $month 2022 > $month.txt 2> error.txt
```

PPA3

> Execute the commands given below in the sequence and collect the outpu/error into a file **errorlog** as described below.
>
> Execute the command **test** with argument **2** and redirect the standard error to the file **errorlog**.

> Execute the command **test -e** and append the standard error output to the file **errorlog**.
>
> Execute the command **test -n** and append the standard error to the file **errorlog**.

```
test $2 2> errorlog ; test -e 2>> errorlog ; test -n 2>> errorlog
```

PPA1

> Write a command to print the name of directories(in the current working directory) that have read, write and execute permissions for other users. Print only the directory name on each line.

```
ls -l | grep '^d.*rwx\b' | cut -d ':' -f 2 | cut -d ' ' -f 2

# Another solution
ls -l | grep 'rwx\b' | cut -d ':' -f 2 | cut -d ' ' -f 2
```

PPA2

> The file **Pincode_info.csv** has information on the pin codes of some places. The output of the command **head -5 Pincode_info.csv** is given below. First line of this file gives the information about the sequence of fields in each line of file following it.
>
> Write a command to display the Circle name and Division name separated by space for the given pincode stored in a shell variable 'pin'. For e.g. if 'pincode=515002' then your command should display 'Andhra Pradesh Anantapur'
>
> Note: If your solution has more than one line, add a semicolon after each line.

> Circle Name,Region Name,Division Name,Office Name,Pincode,OfficeType,Delivery,District,StateName
> Andhra Pradesh Circle,Kurnool Region,Anantapur Division,A Narayanapuram B.O,515004,BO,Delivery,ANANTHAPUR,Andhra Pradesh Andhra Pradesh Circle,Kurnool Region,Anantapur Division,Akuledu B.O,515731,BO,Delivery,ANANTHAPUR,Andhra Pradesh Andhra Pradesh Circle,Kurnool Region,Anantapur Division,Alamuru B.O,515002,BO,Delivery,ANANTHAPUR,Andhra Pradesh Andhra Pradesh Circle,Kurnool Region,Anantapur Division,Allapuram B.O,515766,BO,Delivery,ANANTHAPUR,Andhra Pradesh

```
part=`cat Pincode_info.csv | grep "$pin" | cut -d ',' -f 1,4` ;
part2=`echo ${part/Circle,/}` ;
echo ${part2/ Division/}

# Another solution using sed which has not been taught yet.
cat Pincode_info.csv | grep $pin | cut -d ',' -f 1,4 | sed 's/ Circle,/ /' | sed
's/ Division//'
```

> **Note**: There seems to be an error with the input file. Instead of fields (1, 3) , fields (1,4) worked.

PPA1

> Consider a file named `data.txt` in the current working directory. Write a script to determine if this file has more than 16 lines or not. Your script should print `Yes` if the lines are more than 16; else print `No`.

```
if [ $(cat data.txt | wc -l) -gt 16 ]; then echo "Yes"; else echo "No"; fi;

# Other solutions

if [ `cat data.txt | wc -l` -gt 16 ]; then echo "Yes"; else echo "No"; fi;

lines=$(cat data.txt | wc -l)
if [ $lines -gt 16 ]; then echo "Yes"; else echo "No"; fi;
```

PPA2

> Write a code in awk that can print the sum of odd numbers and sum of even numbers on the next line from the set of natural numbers from 1 to n (ends inclusive), where n is a command line input for the script.

```
BEGIN{
        e=0
        o=0
}
{
    for (i=1;i<($1+1);i++) {
        if (i%2 == 1) {
                o += i
        }
        if (i%2 == 0) {
                e += i
        }
    }
}
END{
        print o
}
END{
        print e
}
```

PPA3

> Write an awk script to find unintentionally repeated (duplicate) words in the file 'myfile.txt'. For example, sometimes a file can contain sentences like "The the building is beautiful". Print the repeated words on the order of occurence at one per line.

```
BEGIN{
    FS=" "
}
{
    for (i=2;i<NF+1;i++)
    {
        if ($i == $(i-1))
        {
            print $i
        }
    }
}
```

PPA1

> Given a file `words.txt` containing a string in each line in the format `FIRST_second`. Every string is a combination of two words joined with an underscore(_), the first word `FIRST` consists of all uppercase letters and the second word `second` consists of all lowercase letters. Write a bash command/script using `sed` to convert all the string to `SECOND_first`. After conversion

- The first and the second words should be swapped.
- The uppercase word should be converted to lowercase word and vice versa.

The file `words.txt` is located in the current working directory.

```
sed -E -e "s/([[:upper:]]+)_([[:lower:]]+)/\U\2_\L\1/" words.txt
```

PPA2

> Without using the wc or awk commands , write a bash script that accepts any number of arguments. Out of these some would be options(hyphen plus a character like -l or -c) and the last argument will be a file path(use ${@: -1} to access the last argument, there is a space before -1). Only four options are accepted by your script -l, -w, -n and -s.

> Assume that file path given will always be for a valid file and we will refer it as file in the next lines. For options,

> If no option is supplied to your script do nothing.

> If -l option is supplied, print the number of lines in the file.

> If -w option is supplied, print the number of words in the file. Assume that any string between spaces is a word. i.e. if using awk count the number of fields in each line to get the word count.

> If -n option is supplied, print the number of lines having only digits(no alphabets or spaces) in the file.

> option -s also accepts an argument say str. In this case print the number of lines containing the string str.

> The above options can be supplied together or more than once. Print the required count for each appearance of the option on a new line. For e.g.

> if -l and -w are both supplied together in the sequence print count of lines and count of words each on separate lines. If -l, -n and -l options are supplied in the sequence then print number of lines, number of lines containing only digits and finally again number of lines in the file each on separate line. Note: Your bash script should not even contain any variable or comment that contians the string wc or awk.

> Hints:

> Use while getopts style code. Use sed to find the count of lines as specified in the conditions aboce.

> Sample

> Suppose your bash script is named as myCount.sh. In the below sample the argument to -s option is "say" so this should count all the lines containing the string "say". For the public test case all the commands given in the below sample are executed one by one on the input file.

```
$ cat somefile.txt
This is a sample file
this is not end justsay start
that contains say
some number
say like 10
or
20
or
233
444
or say 3444
and now it ends.

$ bash myCount.sh -l somefile.txt
12
$ bash myCount.sh -w somefile.txt
32
$ bash myCount.sh -n somefile.txt
3
$ bash myCount.sh -s say somefile.txt
4
$ bash myCount.sh -l -n somefile.txt
12
3
$ bash myCount.sh -l -s say -l -n somefile.txt
12
4
12
3
$ bash myCount.sh
$ bash myCount.sh somefile.txt
```

```
SOLUTION HERE
```