Week 2

GRPA 2

We have a file named "systemcommands.txt" in the present working directory. Write a Bash command to change its permissions to

```
user: read, write, execute
group: execute
others: write
```

```
chmod 712 systemcommands.txt
```

GRPA 3

We want to change the file permissions of "someFile.txt" file as follows.

```
user: execute
group: execute, read
others: write
```

We will use the command chmod XXX someFile.txt where XXX represents a 3 digit number used to set the above permissions. Write a bash command to create a file named XXX.digits in the current working directory such XXX is the same three digit number used to set the permissions as mentioned above. The file your command creates can be empty. For e.g. If your think the command chmod 111 someFile.txt will change the permission of file someFile.txt as mentioned above, then your solution should create a file named 111.digits in the current working directory.

```
touch 152.digits
```

GRPA 4

Create two folders named dir1 and dir2 in the current working directory. Try to write a single line bash command to perform the above task.

```
mkdir dir1 dir2
```

GRPA 5

Write two commands one on each line for the following two tasks.

```
Move only the file file_1 present in dir_1 to the empty directory dir_2.
Delete the directory dir_1.
```

dir_1 and dir_2 are directories in the current working directory. The operation should not change your current working directory.

```
mv dir_1/file_1 dir_2
rm -rf dir_1
or
mv ../dir_1/file_1 ../dir_2/file_1
rm -r ../dir_1
```

Week 2

PPA 1

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
```

PPA 2

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
```

GRPA 1

Print the absolute path where the command wget is located.

```
which wget
```

GRPA 2

"dir_1" and "dir_2" are directories in current working directory. Create a symbolic(soft) link to the file "file_1" present in "dir_1" and store it as "file_2" in "dir_2". Hint: The link to file_2 should be either absolute from current working directory i.e. / or relative to dir_2.

```
ln -s /dir_1/file_1 /dir_2/file_2
or
cd dir_2; ln -s ../dir_1/file_1 file_2
```

GRPA 3

Print the username associated with the current session.

```
whoami
```

GRPA 4

Print to the output containing the name of the shell being used, its PID and the flags in the following format "Shell:|PID:|Flags:". There are no spaces in the string.

```
echo "Shell:$SHELL|PID:$$|Flags:$-"
```

GRPA 5

Write a command that runs in a child shell, prints "hello" and exits with the exit code 179.

```
bash -c "echo hello; exit 179"
```

Week 3

PPA 1

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
```

PPA 2

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 -m $month > $month.txt 2> error.txt
```

PPA 3

Execute the commands given below in the sequence and collect the output/error into a file errorlog as described below.

```
Execute the command test 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> errorlog
test -e 2>> errorlog
test -n 2>> errorlog
```

GRPA 1

Add the string “EOF alpha” at the end of the file(starting at a new line) alpha.txt then append the contents of the file numbers.txt at the end of the file(starting at a new line) alpha.txt. alpha.txt and numbers.txt are located in the current working directory.

```
echo "EOF alpha" >> alpha.txt | cat numbers.txt >> alpha.txt
```

GRPA 2

Print the number of lines present in ‘file1’ and ‘file2’ combined, your solution should not print anything else. ‘file1’ and ‘file2’ are located in the current working directory.

Hint: Multiple files can be given as argument to ‘cat’ command.

```
cat file1 file2 | wc -l

# Solution 2
# cat file2 >> file1; wc -l file1

# Solution 3
# cat file2 >> file1; cat file1 | wc -l
```

GRPA 3

There are three files master.txt, half1.txt and half2.txt in the current working directory. Add first 2 lines of half1.txt to the file master.txt at the end(starting at a new line) then append the last 3 lines of the file half2.txt to the file master.txt at the end(starting at a new line). Append the lines in the sequence mentioned.

```
head half1.txt -n2 >> master.txt
tail half2.txt -n3 >> master.txt
or
cat half1.txt |head -2 >> master.txt
cat half2.txt  |tail -3 >> master.txt
```

GRPA 4

An observer wrote a script named createTwingle that produces a file twingle containing names of all the visible stars present in the sky at that instant. Every line in the file twingle is the name of a star. In your current directory the file twingle may or may not be present. If the file twingle is present in the directory then print the number of lines in the file, else execute the command createTwingle it will create the file twingle in the current working directory then print the number of lines in the file twingle.

Hint: Try to use operators discussed in the lectures to give a single line solution for the task.

Note: stderr will not be displayed

```
wc -l twingle || (createTwingle && wc -l twingle)
or
wc -l twingle || createTwingle | wc -l twingle
or
[ -f "twingle"] || createTwingle | wc -l twingle
```

GRPA 5

Print the number of directories in the current working directory. Do not print anything else.

Hint: One solution is to make use of 'ls', 'wc' and pipes('|').

```
ls -d */ | wc -l
```

GRPA 6

The script test will print some text to the standard output, it can be run similar to any other command and does not accept any arguments.

Your task is to print the output after running test on the screen and also append the output at the end(starting at new line) of the file log.
File log is located in the current working directory.

Hint: To solve it in one line check the man page of tee command for appending to the file.

```
test | tee templog
cat templog >> log
or
echo test | tee -a log
```

Week 4

PPA 1

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" | grep -o "\S\+$"
or
ls -l | grep '^d.*rwx\b' | cut -d ' ' -f 9
```

PPA 2

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.

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

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=`grep $pin Pincode_info.csv | grep -e ".* Circle" -o`
circle=${circle% Circle}
division=`grep $pin Pincode_info.csv | grep -e ",\w* Division" -o`
division=${division:1}
division=${division% Division}
echo $circle $division

# Alternative solution
# circle=`grep "$pin" Pincode_info.csv| grep -oP ".*(?= Circle)"`
# division=`grep "$pin" Pincode_info.csv| grep -oP "\w+(?= Division)"`
# echo $circle $division
```

alternate d=`cat Pincode_info.csv | grep $pin | grep -o '.*Circle' | grep -o ".* "` a=`cat Pincode_info.csv | grep $pin | grep -o '[aA-zZ]* Division' | grep -o ".* "` echo $d $a

GRPA 1

The poem “Sail away” by Rabindranath Tagore is stored in the file named poem.

Early in the day it was whispered that we should sail in a boat, only thou and I, and never a soul in the world would know of this our pilgrimage to no country and to no end.

In that shoreless ocean, at thy silently listening smile my songs would swell in melodies, free as waves, free from all bondage of words.

Is the time not come yet? Are there works still to do? Lo, the evening has come down upon the shore and in the fading light the seabirds come flying to their nests.

Who knows when the chains will be off, and the boat, like the last glimmer of sunset, vanish into the night?

Write a command to print the number of non-empty lines that do not contain an article (a, an, the) in it. The command should print a number that is the count of lines, and should not print the lines.

```
grep -e "\ba\b\|\ban\b\|\bthe\b" poem -v | grep -e "\w" | wc -l
or
cat poem | grep '[[:print:]]' | grep -v "\bthe\b"|grep -v "\ba\b"|grep -v "\ban\b"
| wc -l
or
egrep -v '\b(an?|the)\b|(^\s*$)' poem | wc -l
```

GRPA 2

Each line in the file employees.csv contains the name, role and division of employees separated by a comma. Every line corresponds to one employee. The user wants to collect the details of employees who are managers in the R&D division. For managers the string for the role is ‘Manager’ and the division string for employees working in the R&D division is ‘R&D’.

Write a command to collect the required details and redirect the output to a file named “info.csv”. "info.csv" should contain the name, role and division (separated by a comma) of each employee (as per the above criteria) on a separate line.

```
grep -i "manager" employees.csv | grep "R&D" > info.csv
or
grep 'Manager,R&D' employees.csv > info.csv
```

GRPA 3

Write a command that will print all the lines not containing the word gnu (case-insensitive) in the file test.txt present in the current working directory.

```
grep -vi "gnu" test.txt
```

GRPA 4

In a course, the instructor asked the students to submit their projects in a single file named as the student’s roll number. A typical roll number of a student is a 10 character string which is a combination of a four digit(decimal) year and six character hexadecimal number, e.g. "20201f3acd". The instructor specified that the name of the file should be in lower case but some students mistakenly used uppercase for their file names. Each file name is either entirely in lower case or entirely in upper case with numbers.

Your task is to create two arrays(shell variables) named lower and upper. Array lower should not contain the file names that have upper case letters and array upper should contain all the file names that have upper case letters.

Note: The project files are located in the current directory

Hint:

arr=(`ls`) # Each element in arr corresponds to the output from the ls

```
#lower=(`ls | grep "^[0-9]\{4\}[0-9a-f]\{6\}"`)
#upper=(`ls | grep "^[0-9]\{4\}[0-9A-F]\{6\}" | grep -v "[0-9]\{6\}$"`)
or
lower=(`ls |egrep "[[:digit:]]{4}[[:xdigit:]]{6}"|egrep "[[:lower:]]|[[:digit:]]
{10}"`);
upper=(`ls |egrep "[[:digit:]]{4}[[:xdigit:]]{6}"|egrep "[[:upper:]]"`);
or
```

```
lower=(`ls | grep -v '[A-Z]'`)
upper=(`ls | grep '[A-Z]'`)
```

GRPA 5

The file Pincode_info.csv has information on the pin codes of some places. A sample 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.

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,515005,BO,Delivery,ANANTHAPUR,Andhra Pradesh Andhra Pradesh Circle,Kurnool Region,Anantapur Division,Allapuram B.O,515766,BO,Delivery,ANANTHAPUR,Andhra Pradesh

Assume that there are only 10 states for which this system works and the first digit of the pin code is unique for each state. That means for all the places in the entire state the first digit will be same. You are given a shell variable named state that contains a state name(Example: state="Punjab"). Display the number of pin codes available in the file Pincode_info.csv within the state given in the variable state that has the same first and the last digit. For e.g. if the value of state = "Andhra Pradesh", one such pin code is 515005(for the file given above).

Hint: First find the first digit that represents the given state.

```
number = `egrep -i "$state" Pincode_info.csv | head -1 | egrep -o [0-9]{6} | cut -
c1`
egrep -i "$state" Pincode_info.csv | egrep -o "$number[0-9]{4}$number" | wc -l
or
egrep "$state" -i Pincode_info.csv | egrep "[0-9]{6}" -o | egrep "(.)....\1" | wc
-l
```

Week 5

PPA 1

Write a bash script that accepts any number of arguments and print the odd numbered arguments, i.e. first argument($1), third argument($3), fifth argument($5) and so on. In the output the values of the arguments should be separated by a space and printed on the same line.

```
count=1
for var in "$@"; do
        if [ $((count%2)) -ne 0 ]; then
                echo -n "$var "
        fi
        ((count=count+1))
done
or
m=0
for i in $@
do
    (( m++ ))
    if ((m%2!=0))
    then
        echo $i | tr '\n' ' '
    fi
done
```

PPA 2

A finance company called `Fintech` does not have its own analytics team, rather they outsources their analytics work to external vendors. The company maintains a file for each investor they are interested in, with the filename in the format 'firstname_lastname' in the directory 'data'(directory 'data' is in the current working directory). They do not want to make their investor details public so they rename each investor file to the hash value of filename before giving it to the external vendors and store the mapping in the file named map. Assume that no two investors have same combination of 'firstname' and 'lastname'.

The analytics team(external vendor) received the files of several investors named as some hash value. They analysed and identified some potential investors for the company and stored the hashed file names(one on each line) corresponding to the identified investors in a file named 'result'. The 'firstname_lastname' file contains the potential investment amount and duration details. Refer the provided sample data for sample file formats and architecture.

Write a bash script to print the sum of investment amounts of all the investors identified by the analytics team whose names(hash values) are present in the file 'result'.

Architecture and Sample Data

$ ls -Rl .: total 4 drwxrwxrwx 1 user user 512 Dec 11 19:48 data -rwxrwxrwx 1 user user 435 Dec 11 20:53 map -rwxrwxrwx 1 user user 195 Dec 11 20:53 result ./data: total 0 -rwxrwxrwx 1 user user 36 Dec 11 20:53 Billie_Barron -rwxrwxrwx 1 user user 36 Dec 11 20:53 Jeremiah_Brennan -rwxrwxrwx 1 user user 36 Dec 11 20:53 Long_Mclaughlin -rwxrwxrwx 1 user user 35 Dec 11 20:53 Lorna_Trevino -rwxrwxrwx 1 user user 35 Dec 11 20:53 Mandy_Mueller $ cat ./data/Billie_Barron INVESTMENT $16319 FROM 2026 TO 2026 $ cat map bb1cc74d6e8c40efdbbbc0e6a657fca02a533fe7f438d5b09b47b43e31cb9a45 ./data/Mandy_Mueller fe818c4ac047523ac14dbb341f365bbfcd857268a6bfb70d9abb701a80bfb9c3 ./data/Lorna_Trevino e8b80161e4110a791a0d7c3e40a04099fc75f0e348c2033efe79a2a930a71e98 ./data/Long_Mclaughlin b1b1222a15fcd532e511d0f461dbe7ae7bda825b68bc510eec2a22cbddd5dad2 ./data/Billie_Barron 6051c4f27079a41afc99a97f0fc7bb8ba2789cd282f4cd10a71c6a954089b63e ./data/Jeremiah_Brennan $ cat result bb1cc74d6e8c40efdbbbc0e6a657fca02a533fe7f438d5b09b47b43e31cb9a45 fe818c4ac047523ac14dbb341f365bbfcd857268a6bfb70d9abb701a80bfb9c3 e8b80161e4110a791a0d7c3e40a04099fc75f0e348c2033efe79a2a930a71e98 $ grep INVESTMENT -r data/Billie_Barron:INVESTMENT $16319 data/Jeremiah_Brennan:INVESTMENT $29440 data/Long_Mclaughlin:INVESTMENT $25906 data/Lorna_Trevino:INVESTMENT $1360 data/Mandy_Mueller:INVESTMENT $7979

Expected output for above data

35245

```
  sum=0
  for i in $(cat result); do
    while read hash name; do
      if [ $i == $hash ]; then
        inv=$(grep INVESTMENT $name)
        inv=${inv//INVESTMENT $/}
        sum=$((sum+inv))
      fi
    done < map
  done
  echo $sum
or
investors=cat result
sum=0
for i in $investors;do
x=grep $i map | cut -d " " -f 2 # | cut -d "/" -f 3
# cat $x
((sum+=grep -i "investment" $x | cut -d " " -f 2 | cut -d "$" -f 2`))
done
echo $sum
```

PPA 3

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
```

GRPA 1

Write a bash script which takes one argument as the name of a file and prints Yes if the file has read permission only for the owner and no other permissions for owner or other users, else do not print anything. The file given in the argument will be present in the current working directory.

```
if [[ `ls -l $1` =~ .r\-\-\-\-\-\-\-\- ]]; then
    echo Yes
```

```
fi
or
if [[ $(ls -l $1 | grep -e "^-r--------.*") ]] ; then
    echo "Yes"
fi
```

GRPA 2a

Write a bash script that accepts a few arguments(all numbers) and performs the following functions.

```
Prints the string Error if the number of arguments supplied is not equal to 2.
If the number of arguments is equal to two, print their sum.
```

```
if [[ $# -ne 2 ]]; then
    echo Error
else
    sum=$((${1}+${2}))
    echo $sum
fi
or
if [ $# != 2 ]; then
  echo Error
fi

if [ $# = 2 ]; then
  echo $(( $1 + $2 ))
fi
```

GRPA 2b

Write a bash script that reads a value from the standard input stream and prints PNUM if the value is a postive number or 0; prints NNUM if it is a negative number; else print STRING.

```
read n
num="^-?[0-9]*\.?[0-9]*$"
neg="^-"
if [[ $n =~ $num ]]; then
  [[ $n =~ $neg ]] && echo NNUM || echo PNUM
else
  echo STRING
fi
or
pos="^[0-9]*\.?[0-9]*$"
neg="^-[0-9]*\.?[0-9]*$"
read line
if [[ $line =~ $pos ]]; then
    echo PNUM
    exit
fi
if [[ $line =~ $neg ]]; then
    echo NNUM
    exit
else
    echo STRING
fi
```

GRPA 3

Write a bash script that takes any number of inputs(all numbers) and prints the maximum and minimum value from all the inputs in the format Maximum: max | Minimum: min, where max is the maximum value and min is the minimum value.

```
max=$1
min=$1
while ! [[ -z "$1" ]]
do
    if (( $1 > $max )); then
        max=$1
    fi
    if (( $1 < $min )); then
        min=$1
    fi
    shift
done
echo "Maximum: $max | Minimum: $min"
or
max=$1
min=$1

for i in "$@"; do
    if [ $i -ge $max ]; then
        max=$i
    fi
    if [ $i -le $min ]; then
        min=$i
    fi
done;

echo "Maximum: $max | Minimum: $min"
```

GRPA 4

Write a bash script that takes a number as an argument and prints "Yes" if the number is a prime number, else prints "No".

```
if [[ $1 -le 1 ]]; then
    echo No
    exit
fi
if [[ $1 -eq 2 || $1 -eq 3 ]]; then
    echo Yes
    exit
else
    for ((i=2; i<$1; i++))
    # n=$(($1/2))
# echo $n
    # while [[ $i -lt $1 ]]
    do
        if [[ $(($1%i)) -eq 0 ]]; then
            echo No
            exit
        # else
        #     ((i++))
        fi
            # echo Yes
    done
    echo Yes
fi
or
flag=0
number=$1
check=`echo "sqrt($number)" | bc`
for (( i=2; i<=check; i++ )); do
    if [ $((number%i)) -eq 0 ]; then
        flag=1
    fi
done
if [ $flag -eq 0 ]; then
    echo Yes
else
   echo No
fi
```

GRPA 5

Write a bash script that takes two integer values as input, and prints the product table of first integer with all the integers from 1 to the value in second argument as described in the format below.

Let the first argument be 3 and the second argument be 4, then your script should print.

3*1=3 3*2=6 3*3=9 3*4=12

If the first argument is 12 and second argument is 3, then your script should print

12*1=12 12*2=24 12*3=36

Note that there is no space between any numbers, * or = sign in each line. And every product is printed on a new line.

```
for  ((i=1; i<="$2"; i++ ))
# for i in $n
do
mult=$(($1*$i))
echo "$1*$i=$mult"
done
or
for (( i=1; i<=$2; i++ )); do
  echo $1*$i=$(($1*i))
done
```

GRPA 6

Consider a directory named "perf_folder" containing some files with different extensions, present in the current working directory. Write a bash script that accepts an argument(name of destination directory), adds a prefix string "program_" to the file names in the directory "perf_folder" meeting the below criteria.

```
The file extension is ".c".
The file names should containing the substring perf.
```

Also move all the files meeting the above criteria after renaming to the directory(destination) whose name is specified as an argument to your script. The destination directory may or may not be present in the current working directory, if not present create the directory under current working directory.

For e.g. the argument to your script is perf_programs, i.e. perf_programs is the destination directory for renamed files.

If below is the output of ls perf_folder when run in your current working directory.

perf_results.cvc perf_conf.xml set_perf_input.c perf_params.c start_test.c stop_test.c results.txt script.sh

Then after running your script, the new output of running ls perf_folder in your current working directory should be,

perf_results.cvc perf_conf.xml start_test.c stop_test.c results.txt script.sh

and output of running command ls perf_programs in your current working directory should be,

program_set_perf_input.c program_perf_params.c

```
mkdir $1
for file in `ls perf_folder/`
do
    if [[ $file =~ .*perf.*\.c$ ]]
    then
```

```
        newfile="program_$file"
        mv "perf_folder/$file" "$1/$newfile"
    fi
done
or
# If directory as argument one is not present, create it.
ls -d $1
if [[ $? -ne 0 ]]; then
        mkdir $1
fi

cd perf_folder
for file in *perf*.c; do
       mv $file ../$1/program_$file
done
```

GRPA 7

Write a bash script that prints the sum of all even numbers of an array of numbers. The array variable is named as number_arr.

```
read -a number_arr
sum=0

len=${#number_arr[@]}
# echo $len
for ((i=0; i<len; i++))
do
    num=number_arr[$i]
    if [[ $(($num%2)) -eq 0 ]]; then
        sum=$(($sum+$num))
    fi
done
echo $sum
or
read -a number_arr
sum=0
for num in ${number_arr[@]}; do
    if [ $((num%2)) -eq 0 ]; then
        ((sum+=$num))
    fi
done
echo $sum
```

GRPA 8

Write a bash script that accepts an integer as argument and prints the corresponding day of week in capitals as given in the table below. Argument 1 or 8 2 or 9 3 4 5 6 7 Output SUNDAY MONDAY TUESDAY WEDNESDAY THURSDAY FRIDAY SATURDAY

If the argument is greater than 9 print ERROR

Hint: Use case statement.

```
case $1 in
     1 | 8)
        echo SUNDAY ;;
    2 | 9)
    echo MONDAY ;;
    3)
    echo TUESDAY;;
    4)
    echo WEDNESDAY;;
    5)
    echo THURSDAY;;
    6)
    echo FRIDAY;;
    7)
    echo SATURDAY;;
    *)
    echo ERROR;;
esac
```

Refresher Week 1-5

PPA 1

Write a command to extract the system's processor architecture from the output of uname -a. The output should be something like 'arm64', 'x86_64' etc.

```
uname -a | grep "\bx.*\b" -o | cut -d ' ' -f1
```

PPA 2

Write a Bash command to print the number of failed login attempts which are recorded in the file myauth.log located in the current working directory.

Contents of myauth.log is given below

Jan 20 20:11:34 IITMBSC systemd-logind[897]: Session 27 logged out. Waiting for processes to exit. Jan 20 20:11:34 IITMBSC systemd-logind[897]: Removed session 27. Jan 20 20:17:01 IITMBSC CRON[70999]: pam_unix(cron:session): session opened for user root by (uid=0) Jan 20 20:17:01 IITMBSC CRON[70999]: pam_unix(cron:session): session closed for user root Jan 20 20:21:10 IITMBSC su: (to root) student on pts/4 Jan 20 20:21:10 IITMBSC su: pam_unix(su:session): session opened for user root by student(uid=0) Jan 20 20:21:21 IITMBSC su: pam_unix(su:session): session closed for user root Jan 20 20:21:21 IITMBSC sudo: pam_unix(sudo:session): session closed for user root Jan 20 20:22:02 IITMBSC gdm-launch-environment]: pam_unix(gdm-launch-environment:session): session opened for user gdm by (uid=0) Jan 20 20:22:02

```
IITMBSC systemd-logind[897]: New session c2 of user gdm. Jan 20 20:22:02 IITMBSC systemd:
pam_unix(systemd-user:session): session opened for user gdm by (uid=0) Jan 20 20:22:03 IITMBSC
polkitd(authority=local): Registered Authentication Agent for unix-session:c2 (system bus name :1.357
[/usr/bin/gnome-shell], object path /org/freedesktop/PolicyKit1/AuthenticationAgent, locale en_US.UTF-8) Jan
20 20:22:21 IITMBSC gdm-launch-environment]: pam_unix(gdm-launch-environment:session): session closed
for user gdm Jan 20 20:22:21 IITMBSC systemd-logind[897]: Session c2 logged out. Waiting for processes to
exit. Jan 20 20:22:22 IITMBSC systemd-logind[897]: Removed session c2. Jan 20 20:22:22 IITMBSC
polkitd(authority=local): Unregistered Authentication Agent for unix-session:c2 (system bus name :1.357,
object path /org/freedesktop/PolicyKit1/AuthenticationAgent, locale en_US.UTF-8) (disconnected from bus)
Jan 20 20:22:53 IITMBSC login[70867]: pam_unix(login:auth): authentication failure; logname=LOGIN uid=0
euid=0 tty=/dev/tty3 ruser= rhost= user=guest Jan 20 20:22:56 IITMBSC login[70867]: FAILED LOGIN (1) on
'/dev/tty3' FOR 'guest', Authentication failure Jan 20 20:23:01 IITMBSC login[70550]: pam_unix(login:session):
session closed for user guest2 Jan 20 20:23:01 IITMBSC systemd-logind[897]: Session 25 logged out. Waiting
for processes to exit. Jan 20 20:23:01 IITMBSC systemd-logind[897]: Removed session 25. Jan 20 20:23:18
IITMBSC login[71620]: pam_unix(login:auth): authentication failure; logname=LOGIN uid=0 euid=0
tty=/dev/tty4 ruser= rhost= user=guest2 Jan 20 20:23:20 IITMBSC login[71620]: FAILED LOGIN (1) on
'/dev/tty4' FOR 'guest2', Authentication failure Jan 20 20:23:27 IITMBSC login[71620]: pam_unix(login:auth):
authentication failure; logname=LOGIN uid=0 euid=0 tty=/dev/tty4 ruser= rhost= user=root Jan 20 20:23:30
IITMBSC login[71620]: FAILED LOGIN (2) on '/dev/tty4' FOR 'root', Authentication failure Jan 20 20:23:39
IITMBSC login[71620]: pam_unix(login:auth): check pass; user unknown Jan 20 20:23:39 IITMBSC login[71620]:
pam_unix(login:auth): authentication failure; logname=LOGIN uid=0 euid=0 tty=/dev/tty4 ruser= rhost= Jan
20 20:23:43 IITMBSC login[71620]: FAILED LOGIN (3) on '/dev/tty4' FOR 'UNKNOWN', Authentication failure
Jan 20 20:23:48 IITMBSC login[71620]: pam_unix(login:session): session opened for user guest by
LOGIN(uid=0) Jan 20 20:23:48 IITMBSC systemd-logind[897]: New session 30 of user guest. Jan 20 20:23:48
IITMBSC systemd: pam_unix(systemd-user:session): session opened for user guest by (uid=0) Jan 20 20:23:51
IITMBSC login[71620]: pam_unix(login:session): session closed for user guest Jan 20 20:23:51 IITMBSC
systemd-logind[897]: Session 30 logged out. Waiting for processes to exit. Jan 20 20:23:51 IITMBSC systemd-
logind[897]: Removed session 30. Jan 20 20:24:01 IITMBSC login[71803]: pam_unix(login:session): session
opened for user student by LOGIN(uid=0) Jan 20 20:24:01 IITMBSC systemd-logind[897]: New session 32 of
user student. Jan 20 20:24:02 IITMBSC login[71803]: pam_unix(login:session): session closed for user student
Jan 20 20:24:02 IITMBSC systemd-logind[897]: Session 32 logged out. Waiting for processes to exit. Jan 20
20:24:02 IITMBSC systemd-logind[897]: Removed session 32. Jan 20 20:24:06 IITMBSC login[71900]:
pam_unix(login:auth): authentication failure; logname=LOGIN uid=0 euid=0 tty=/dev/tty4 ruser= rhost=
user=root Jan 20 20:24:09 IITMBSC login[71900]: FAILED LOGIN (1) on '/dev/tty4' FOR 'root', Authentication
failure Jan 20 20:24:16 IITMBSC login[71900]: pam_unix(login:session): session opened for user guest by
LOGIN(uid=0) Jan 20 20:24:16 IITMBSC systemd-logind[897]: New session 33 of user guest. Jan 20 20:24:16
IITMBSC systemd: pam_unix(systemd-user:session): session opened for user guest by (uid=0) Jan 20 20:24:17
IITMBSC login[71900]: pam_unix(login:session): session closed for user guest Jan 20 20:24:17 IITMBSC
systemd-logind[897]: Session 33 logged out. Waiting for processes to exit. Jan 20 20:24:17 IITMBSC systemd-
logind[897]: Removed session 33. Jan 20 20:24:22 IITMBSC login[72064]: pam_unix(login:session): session
opened for user guest2 by LOGIN(uid=0) Jan 20 20:24:22 IITMBSC systemd-logind[897]: New session 35 of
user guest2. Jan 20 20:24:22 IITMBSC systemd: pam_unix(systemd-user:session): session opened for user
guest2 by (uid=0) Jan 20 20:24:24 IITMBSC login[72064]: pam_unix(login:session): session closed for user
guest2 Jan 20 20:24:24 IITMBSC systemd-logind[897]: Session 35 logged out. Waiting for processes to exit. Jan
20 20:24:24 IITMBSC systemd-logind[897]: Removed session 35. Jan 20 20:24:27 IITMBSC agetty[72229]: tty4:
invalid character 0x1b in login name Jan 20 20:24:43 IITMBSC login[72242]: pam_unix(login:auth):
```

authentication failure; logname=LOGIN uid=0 euid=0 tty=/dev/tty4 ruser= rhost= user=guest2 Jan 20 20:24:47 IITMBSC login[72242]: FAILED LOGIN (1) on '/dev/tty4' FOR 'guest2', Authentication failure Jan 20 20:24:48 IITMBSC login[72242]: pam_nologin(login:auth): unexpected response from failed conversation function Jan 20 20:24:48 IITMBSC login[72242]: pam_nologin(login:auth): cannot determine username Jan 20 20:24:51 IITMBSC login[72242]: FAILED LOGIN (2) on '/dev/tty4' FOR 'UNKNOWN', User not known to the underlying authentication module

```
egrep "\bFAILED LOGIN\b" myauth.log | wc -l
```

PPA 3

Write a script to print the users(one on each line) who are logged in successfully. Extract the information from the file named myauth.log located in the current working directory. The output should contain usernames only and should be unique.

Hint: Use uniq command to get all distinct lines of the output.

Contents of myauth.log is given below

```
egrep "systemd-logind\[[0-9]+\]" myauth.log | grep user | cut -d " " -f 11 | cut -
d "." -f 1 | sort | uniq
```

PPA 4

Print the previous login time of the user guest in the format MMM DD HH:MM:SS. Where MMM, DD, HH, MM and SS corresponds to Month (E.g. Nov), Date, Hours, Minutes and Seconds respectively. Extract the information from the logs available in the file myauth.log in the current directory. Sample log file below.

```
grep "\bsession opened for user guest\b" myauth.log | tail -1 | cut -d " " -f 1-3
```

PPA 5

Mine the logs given in the file myauth.log present in the current working directory to print all the usernames to which user student switched to using su command.

Note: switching back to the previous user should not be accounted.

Hint: Basically you have to grep all the lines where 'su' command is run successfully and fetch the username to which the user student switched to.

Contents of myauth.log is given below

```
egrep "\bsu\b" myauth.log | grep -v FAILED | egrep "\(to .*\)" -o | egrep
"\b\w*\b" -o | grep -v to
```

Week 6

PPA 1

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.

```
cat words.txt | sed -e "s/^\(.*\)_\(.*\)\$/\2_\1/g" | sed
"y/abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ/ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz/"
```

Week 6

PPA 2

Without using the wc or awk commands(instead use sed as Bash command), 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

```
filename=${@: -1}

while getopts "wlns:" options; do
  case "${options}" in
    s)
      str=${OPTARG}
      sed "s/ /\n/g" $filename | sed -ne "/$str/ p" | sed -n "\$="
      ;;
    w)
      sed "s/ /\n/g" $filename | sed -ne "/^\$/! p" | sed -n "\$="
      ;;
    l)
      sed -n "\$=" $filename
      ;;
    n)
      sed -ne "/^[[:digit:]][[:digit:]]*$/ p" $filename | sed -n "\$="
      ;;
    *)
      echo "ERROR"
      ;;
  esac
```

```
done
```

Week 6

PPA 3

Write a bash command using find that copies all the files within the directory /source to /destination. Note that all the files within all hierarchy have distinct names and should be copied to /destination .

```
find /source/ -exec cp {} /destination/ \;
```

Week 6

GRPA 1

Write a sed command to print the count of lines that starts with a digit in the file input.txt. Assume that there is at least one line in the file input.txt that starts with a digit. Do not use the commands wc or awk , or even these keywords in comments or anywhere in your answer.

```
sed -ne "/^[[:digit:]]/p" input.txt | sed -n "\$="
```

Week 5

GRPA 2

Given a file input.txt containing a word on each line, print all the words(one in each line) that occur between the words "FROM" and "TO"(but excluding these words). The match should be case sensitive for the given words and the words in the file are not unique, they can repeat. For e.g. for Input file

$ cat input.txt This is TO some 4word FROM from FROM THE b45eginning TO OKAY FRom okay FROM give me 44some FROM SOME TO TO 54TO4 FROM from

Output should be

THE b45eginning give me 44some SOME from

```
touch temp.txt
cat input.txt | sed -n '/FROM/,/TO/p' >> temp.txt
cat temp.txt | sed -e "1d" | sed -e "$ d"

or
# Solution 1
sed -n '/FROM/,/TO/p' input.txt | sed '/FROM/d' | sed '/TO/d'
```

```
# Solution 2
# sed -n '/FROM/,/TO/{//!p;}' input.txt
# Solution 3
# sed -n '/FROM/,/TO/{/FROM/d;/TO/d;p;}'
```

Week 6

GRPA 3

In the lines that start with a digit, if there is a words "delta"(case sensitive) replace it with the word "gamma". Replace only the first occurrence of the word "delta" in the desired lines. The filename where the contents present are input.txt.

```
sed -e '/^[[:digit:]]/s/delta/gamma/' input.txt

or
```

Week 6

GRPA 4

Consider a special programming file functions.sh that contains several functions (A function is a block of code). Write a bash script/command using sed to insert a line "# START FUNCTION" before the starting of a function and a line "# END FUNCTION" at the end of the function. Starting of a function in this file can be identified as a line that has some string followed by "(", then followed by ")" or some string followed by ")", and this line should end with "{". Ending of a function can be identified by a line containing only "}" in the whole line. In this file curly braces "{" and "}" are not used for any other purpose. Do not change the original file just print the output to STDOUT.

```
sed -e "/[[:alnum:]+](.*)[[:space:]]*{/i # START FUNCTION" \
    -e "/^[[:space:]]*}/a # END FUNCTION" functions.sh
```

Week 6

GRPA 5

Given some raw programming files, we want them to adhere to the company guidelines. Write a sed script that will run for all ".sh" files in the current directory and print the contents after performing the following actions. You just need to write the sed script, running that for all the files will be taken care of by our driver bash script.

Insert a copyright message at the start of the file(before the first line) as "# Copyright IITM 2022"(Note that there is a space after #). Insert a copyright message at the end of the file(after the last line) as "# Copyright

IITM 2022". Insert a line "# START FUNCTION" before the starting of a function and a line "# END FUNCTION" at the end of the function. Check GrPA 4 for more details on identifying function boundaries. Use the same logic here. Change the function "background_sleep" to "inactive_sleep". So replace all the occurrences of the word "background_sleep" in any line with "inactive_sleep". Assume that these keywords are used only in context of a function and nothing else. Also, the function "active_sleep" is deprecated and we do not have an immediate replacement. So insert a line "# TODO:DEPRECATED" before the function "active_sleep" and in every instance. i.e. before every line containing the word "active_sleep". After every 10th line (in line numbers 10, 20, 30,...) add a line with four hashes such as "####" after applying all the above actions.

```
Perform all the above actions in the order given from top to bottom.

For example, for the input file

echo Hello
EOF
analysis.sh
script() {
  sum=0
  for i in $(cat result); do
    while read hash name; do
      if [ $i == $hash ]; then
        inv=$(grep INVESTMENT $name)
        inv=${inv//INVESTMENT $/}
        sum=$((sum+inv))
      fi
    done < map
  done
  echo $sum
}

mkdir data

read fnos
for (( i=0; i<fnos; i++ )); do
  read line
  echo $line | cut -d ":" -f 2- | tr '#' '\n' > ./data/${line%%:*}
done

read mnos
for (( i=0; i<mnos; i++ )); do
  read line
  echo $line  >> map
done

read rnos
for (( i=0; i<rnos; i++ )); do
  read line
  echo $line  >> result
done
```

```
script

Output should be

# Copyright IITM 2022
echo Hello
EOF
analysis.sh
# START FUNCTION
script() {
  sum=0
  for i in $(cat result); do
    while read hash name; do
      if [ $i == $hash ]; then
        inv=$(grep INVESTMENT $name)
####
        inv=${inv//INVESTMENT $/}
        sum=$((sum+inv))
      fi
    done < map
  done
  echo $sum
}
# END FUNCTION

mkdir data
####

read fnos
for (( i=0; i<fnos; i++ )); do
  read line
  echo $line | cut -d ":" -f 2- | tr '#' '\n' > ./data/${line%%:*}
done

read mnos
for (( i=0; i<mnos; i++ )); do
####
  read line
  echo $line  >> map
done

read rnos
for (( i=0; i<rnos; i++ )); do
  read line
  echo $line  >> result
done
####
script
# Copyright IITM 2022
```

```
1 i\# Copyright IITM 2022
$ a\# Copyright IITM 2022
/[[:alnum:]+](.*)[[:space:]]*{/i\# START FUNCTION
/^[[:space:]]*}/a\# END FUNCTION
s/background_sleep/inactive_sleep/g
/\bactive_sleep/ i\# TODO:DEPRECATED
10~9 i\####
```

Week 6

GRPA 6

project is a directory present in the current working directory that has some text files. Write a Bash script that takes all files with the extension .h to create a tarball named headers.tar. Then compress the tarball with gzip named as headers.tar.gz without losing the headers.tar file.

```
tar cf headers.tar project/*.h
gzip --keep headers.tar
```

Week 7

PPA 1

Write an awk script that reads a value n from the stdin within awk script, then prints the sum of odd numbers and sum of even numbers each on a separate line respectively, from the set of natural numbers from 1 to n (ends inclusive).

```
BEGIN {
  getline n < "-";
  sumodd=0;
  sumeven=0;
  for(i=1;i<=n;i++) {
    if (i%2 == 1) {
      sumodd = sumodd + i;
    }
    else {
      sumeven = sumeven + i;
    }
  }
  print sumodd;
  print sumeven;
}
```

Week 7

PPA 2

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 {
  prev="";

}

{
  $0=tolower($0);
  #gsub(/[^A-Za-z0-9 \t]/, "");
  for (i = 1; i <= NF; i++) {
    if ($i == prev) {
      print $i

    }
    prev = $i;
  }
}
```

Week 7

PPA 3

Write a script using AWK to print the file with the maximum number of lines. Assume only one file that have the maximum number of line among the given files.

```
{
  if(max < FNR) {
    max=FNR;
    f=FILENAME;
  }
}
END { print f; }
```

Week 7

GRPA 1-A

Consider a file named marks.csv containing roll number and marks of variable number of subjects of students. The values are comma separated values and in the format RollNo,Subject1,Subject2,Subject3,So on...

Write an Awk command to print all the roll numbers(RollNo) in the file.

```
awk -F, '{print $1}' marks.csv
```

Week 7

GRPA 1-B

Write an Awk command to print the first field of the all the lines containing more than 20 characters in the file marks.csv. The field separator in the file is comma (,).

```
awk -F, 'length($0)>20{print $1}' marks.csv
```

Week 7

GRPA 1-C

Write an awk script to print the total number of fields in a csv file with the field separator as comma (,). Print only the number and nothing else.

```
BEGIN{
 FS=",";
 sum=0;
}
{
 sum=sum+NF;
}
END{
 print sum;
}
```

Week 7

GRPA 1-D

Write an Awk Script to print all the lines whose starting and ending character is a digit. Also print the count of these lines(only the number) on a new line at the last in your output. The field separator in the file is comma (,).

Note, that here it is asked to write an Awk script. Read the Programming questions instructions for more clarity.

```
BEGIN{
 FS=",";
 sum=0;
}
/^[0-9].*[0-9]$/{
 print $0;
 sum=sum+1;
}
END{ print sum;
}
```

Week 7

GRPA 2

A software company has published some best practices for writing the code. One of the best practice mentioned is that if no line in your code should exceed 50 characters in total including all type of characters or spaces.

Given a bash script that intends to print the names of all .c files that contain one or more lines with length more than 50 characters(as specified above).

The awk script within this bash script to check the files as per above condition is missing in the code, complete that

```
BEGIN {
      flag=0;
    }

    {
      if (length($0)>50) flag=1;
    }

    END {
      if (flag==1) print FILENAME;
    }
```

Week 7

GRPA 3

Without using the wc command , 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. If no argument is specified with -s option print 0. 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.

Hints:

```
Use while getopts style code.
Use awk to find the count. Or a combination of egrep and awk for counting lines
which matches some pattern. Or pattern block of awk.
```

Note: Do not use single quotes in your script. Either replace each single quote with double quotes(if you are not using any double quotes in your awk script) or replace each single quote with `'''`

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 inthe same sequence 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 31 $ 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

```
filename=${@: -1}

while getopts "wlns:" options; do
  case "${options}" in
    s)
      str=${OPTARG}
      grep $str $filename | awk "END{print NR}"
      ;;
    w)
      awk "BEGIN{c=0} {c+=NF} END{print c}" $filename
      ;;
    l)
      awk "END{print NR}" $filename
      ;;
    n)
```

```
        awk "BEGIN{c=0} /^[[:digit:]]+$/{c++} END{print c}" $filename
        ;;
      *)
        echo "ERROR"
        ;;
    esac
  done
```

Week 7

GRPA 4

EmployeeDetails.csv file contains the Employee ID, Employee Name, Leaves taken this year and Gender, of all the employees working in a company XYZ, born between the years 1997 and 2000 (including both). Total employees in the company is less than 1000.

The employee ID is of the format: DepartmentYearOfBirthCode Where:

- Department is the department to which the employee belongs to (Department A to G)
- YearOfBirth is the birth year of the employee (Eg. 2000)>
- Code is a three digit number unique to each employee.

For e.g. B1997122 is employee id of an employee working in department B, born in the year 1997 having unique code as 122. The email ID of an employee is in the format EmployeeID@xyz.com, where EmployeeID is the employee id of the employee. For example email id of Ram having employee id as A1998001 is A1998001@xyz.com. Email ids are case sensitive.

Write an awk script that that takes the file EmployeeDetails.csv as input and prints the email ids of all the female employees of the company in the same sequence as the employee details appear in the file EmployeeDetails.csv.

Sample Suppose your awk script is named as yourScript.awk

$ cat EmployeeDetails.csv A1998001,Ram Kumar,10,Male A1998002,Mohammed Iqbal,5,Male A1998003,Priya Lal,7,Female A1999001,Sunita Sharma,25,Female A2000001,Rose Mary Thomas,3,Female B1999001,Sri Lakshmi Jai,5,Female $ awk -f yourScript.awk EmployeeDetails.csv
A1998003@xyz.com A1999001@xyz.com A2000001@xyz.com B1999001@xyz.com

```
BEGIN{
    FS = ","
}

{
    EID = $1
    Gender= $4
    if (Gender ~ /Female/) {
        print EID"@xyz.com"
        }
```

```
  }
```

Week 7

GRPA 5

EmployeeDetails.csv file contains the Employee ID, Employee Name, Leaves taken this year and Gender, of all the employees working in a company XYZ, born between the years 1997 and 2000 (including both). Total employees in the company is less than 1000.

The employee ID is of the format: DepartmentYearOfBirthCode Where:

- Department is the department to which the employee belongs to (Department A to G)
- YearOfBirth is the birth year of the employee (Eg. 2000)>
- Code is a three digit number unique to each employee.

For e.g. B1997122 is employee id of an employee working in department B, born in the year 1997 having unique code as 122. The email ID of an employee is in the format EmployeeID@xyz.com, where EmployeeID is the employee id of the employee. For example email id of Ram having employee id as A1998001 is A1998001@xyz.com. Email ids are case sensitive.

Write an awk script takes the file EmployeeDetails.csv as input and prints the name of the employee(s) with lowest number of leaves taken this year. If there are more than one employees with the lowest number of leaves, print the name of each employee on a new line.

Sample Suppose your awk script is named as yourScript.awk. For the below sample file the lowest number of leaves taken by any employee is 5. And there are two employees who have taken only 5 leaves, print both employee names on a separate line.

$ cat EmployeeDetails.csv A1998001,Ram Kumar,10,Male A1998002,Mohammed Iqbal,5,Male A1998003,Priya Lal,7,Female A1999001,Sunita Sharma,25,Female A2000001,Rose Mary Thomas,13,Female B1999001,Sri Lakshmi Jai,5,Female $ awk -f yourScript.awk EmployeeDetails.csv
Mohammed Iqbal Sri Lakshmi Jai

```
BEGIN{
  FS = ",";
}
{
  if (NR == 1)
  {
    lowc=int($3);
    count =0;
    name[count] = $2;
    next;
  }
  Name = $1;
  leave = $3;
  if (leave < lowc)
  {
    lowc = leave;
    delete name;
    count = 0;
    name[count] = $2;
  }
  else if (leave == lowc)
  {
    count++; name[count] = $2
  }
}

END{
  for (i=0; i<=count; i++)
  {
    print name[i];
  }
}
```

Week 7

GRPA 6

EmployeeDetails.csv file contains the Employee ID, Employee Name, Leaves taken this year and Gender, of all the employees working in a company XYZ, born between the years 1997 and 2000 (including both). Total employees in the company is less than 1000.

The employee ID is of the format: DepartmentYearOfBirthCode Where:

- Department is the department to which the employee belongs to (Department A to G)
- YearOfBirth is the birth year of the employee (Eg. 2000)
- Code is a three digit number unique to each employee.

For e.g. B1997122 is employee id of an employee working in department B, born in the year 1997 having unique code as 122. The email ID of an employee is in the format EmployeeID@xyz.com, where EmployeeID is the employee id of the employee. For example email id of Ram having employee id as A1998001 is A1998001@xyz.com. Email ids are case sensitive.

Write an awk script that takes input as file EmployeeDetails.csv and calculate and prints the average number of leaves taken by the employees born in each year from 1997 to 2000(both 1997 and 2000 included). The average for each year should be printed on a newline starting from the year 1997 to 2000 in the same sequence i.e. your script should print 4 lines of output always one for each year 1997, 1998, 1999 and 2000. If there are no employees born in some year, print 0 for that years average leaves. Print only the integer part of the average(i.e. if the average is 7.3333 print 7). Use int() function to get the integer part of any float number.

Sample Suppose your awk script is named as yourScript.awk. For the below sample file the average number of leaves taken by employees born in years 1997 to 2000 is printed in 4 lines in the same sequence.

$ cat EmployeeDetails.csv A1998001,Ram Kumar,10,Male A1998002,Mohammed Iqbal,5,Male A1998003,Priya Lal,7,Female A1999001,Sunita Sharma,25,Female A2000001,Rose Mary Thomas,13,Female B1999001,Sri

Lakshmi Jai,5,Female $ awk -f yourScript.awk EmployeeDetails.csv 0 7 15 13

```
BEGIN{
  FS = ","
}

{
  if (NR == 1){
    l1997 = 0; c1997 = 0; av1997 = 0;
    l1998 = 0; c1998 = 0; av1998 = 0;
    l1999 = 0; c1999 = 0; av1999 = 0;
    l2000 = 0; c2000 = 0; av2000 = 0;
  }
  EID = $1;
  leave = int($3);
  # to obtain year from employee ID
  year = int(substr(EID, 2, 4));
  if (year == 1997)
  {
    l1997 = l1997 + leave; c1997++;}
  else if (year == 1998)
  {
    l1998 = l1998 + leave; c1998++;}
  else if (year == 1999)
  {
    l1999 = l1999 + leave; c1999++;}
  else if (year == 2000)
  {
    l2000 = l2000 + leave; c2000++;}
}

END{
  if (c1997 != 0)
    {av1997 = l1997/c1997;}
  if (c1998 != 0)
    {av1998 = l1998/c1998;}
  if (c1999 != 0)
    {av1999 = l1999/c1999;}
  if (c2000 != 0)
    {av2000 = l2000/c2000;}
  print (int(av1997))
  print (int(av1998))
  print (int(av1999))
  print (int(av2000))
}
```

Week 7

GRPA 7

You have a csv file named groceries.csv that contains a list of grocery items and their unit cost. The two fields are separated by comma(,). This file will be given as input to your Awk script.

Write an Awk script that takes two arguments(command line) named item and n, where item is the item name and n is the number of units, then prints the total cost of purchasing n units of the item item. The script prints only a number. i.e. you need to find the item cost of the item given in argument while parsing the input file. Note: You can directly use these variables with the given name in your Awk script. Assume that the item given in the argument will always be present in the csv file.

Sample(suppose your script is named as yourScript.awk) Here the cost of 3 Tomatoes needs to be calculated. Cost of one Tomato is 40 as seen from the csv file. So total cost = 3*40

$ cat groceries.csv 1,Tomato,40 2,Brinjal,35 3,Banana,60 $ awk -f yourScript.awk groceries.csv Tomato 3 120

```
BEGIN {
 FS = ",";
}
{
  a = $2
  b = $3
  if (a ~ item) {
    ans = b*n;
    print ans;
    exit;
  }
}
```

Week 5

PPA 1

```

```

Replit

Week - 2, Code with us

Problem 1

Find the CPU used in the machine and print it. Ex: `Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz`

Solution

```
# It will be different for every user. Read the file /proc/cpuinfo. The cpu used
will be mention after `model name`
echo AMD EPYC 7B12
```

Problem 1

There is a file named `final.txt` in the current working directory, that contains some text that you do not want it to be edited. Write a bash command to remove write permissions of user from this file. Do not edit any other file permissions.

Solution

```
chmod u-w final1.txt
```

Problem 2

In a school consists of first to twelfth standard, each standard have 5 sections named from `A` to `E`. So the class room number is given as standard in number followed by section, for an example `12E`, `3C`, etc..

Your task is to create a directory for every class room. All the directories should be located inside the current directory.

Solution

~~~
mkdir {1..12}{A..E}
```ls -d */ | sed -e "s/\///g" | sort -n

---
## Problem 3

In a school consists of first to twelfth standard, each standard have 5 sections
named from `A` to `E`.
So the class room number is given as standard in number followed by section, for
an example `12E`, `3C`, etc..
Each classroom have 40 students. There is a directory for every class room. All
the directories can be located inside the current directory.

Your task is to create a file for each student named from 1 to 40 in every
directory.


### Solution
```bash
touch {1..12}{A..E}/{1..40}
~~~

Problem 4

In a school consists of first to twelfth standard, each standard have 5 sections named from `A` to `E`. So the class room number is given as standard in number followed by section, for an example `12E`, `3C`, etc.. Each classroom have 40 students. There is a directory for every class room. All the directories can be located inside the current directory. There is a file for every student named from 1 to 40 in every directory.

Suddenly the school experienced 40% drop in first and sixth standard so they reduced the number of students in each class room to 32.

Your task is to remove the extra files in the respective directories.

Solution

```
rm {1,6}{A..E}/{33..40}
```

Problem 6

Find the mime type of `somefile`. Hint: Use `file`. Refer manual of `file` or `file --help

Solution

```
file somefile --mime-type
```

Problem 7

Find the day of week of April 1st of 2024 and print it to the output in full format with first letter capitalised. Ex: `Wednesday` Hint: Use `date` command.

Solution

```
date -d "2024-04-01" +%A
```

Problem 8

The shell variable `logfile` contains the absolute path to some file. Your task is to print two lines, where first line is the filename alone and the second line is the path of the directory in which the file `logfile` is located(print the path without the trail slash `/`).

Example shell variable: `logfile=/home/student78/daily.log`

Output

```
daily.log
/home/student78
```

Solution

```
logfile=/home/student78/daily.log
echo "${logfile##*\/}"
echo "${logfile%\/*}"
```

Problem 9

A shell varialbe named `TOTALCOST` contaisn a string which is in the format `XYZ_ABC_PQR`, where `XYZ`, `ABC` and `PQR` are three digit numbers. Underscore `_` is used to separate the three digit numbers

Write a bash script that replaces all the underscores(`_`) with commas(`,`) in the variable `TOTALCOST` and displays the final string on the screen.

Example: `TOTALCOST=198_890_128`

Output

```
198,890,128
```

Solution

```
echo "${TOTALCOST//_/,}"
```

Problem 10

Display only the filename without extension whose absolute path is stored in a shell variable named `file`. The extension should not include the dot `.`.

Example: `file=/home/student56/tmp/artic.jpg`

Output

```
artic
```

Solution

```
temp="${file%.*}"
echo "${temp##*\/}"
```

Problem 11

The array `colors` contains the name of the colors. Your task is to remove all vowels present in the array and display them.

Example: `colors=(violet indigo blue green yellow orange red)`

Output

```
vlt ndg bl grn yllw rng rd
```

Solution

```
colors="${colors[@]//a/}"
colors="${colors[@]//e/}"
colors="${colors[@]//i/}"
colors="${colors[@]//o/}"
colors="${colors[@]//u/}"
echo "$colors"

# or

echo "${colors[@]//[aeiou]/}"
```

Problem 13

Write a Bash command that displays the name and size in bytes of all the files/directory present in the current directory.

Hint: Refer man page of `stat`

Example:

Output

```
au 0
other 4096
temp.sh 259
```

Solution

```
stat -c "%n %s" *
```

Problem 14

Write a command to display the today's date in a format `18th Jan 2022`.

Solution

```
date +"%dth %b %Y"
```

Week 3—Live Sessions

Problem 1

Prepend the contents of file1.txt to file2.txt. That is, the contents of file2.txt should be the contents of file1.txt followed by file2.txt.

Solution

```
cat file1.txt file2.txt > file3.txt
cat file3.txt > file2.txt
rm file3.txt
```

Problem 2

Redirect the stderr and stdout of the ls file1.txt file2.txt command in that particular order to output.txt

Solution

```
ls file1.txt file2.txt 2> output.txt 1>>output.txt
```

Problem 3

The variable filename has the name of file along with extension. Find the file extension and print it.

Solution

```
echo ${filename##*.}
```

Problem 4

The employee ID of a certain company is in the following format: DDYYPXXX where - DD stands for department; ex: FN for Finance, SD for Software Development - YY stands for work experience - P stands for position; M for Manager, etc. - XXX stands for ID number. Given a Employee ID, redirect just the Department and the ID Number to output.txt .

Solution

```
echo ${id1:0:2}${id1: -3:3} > output.txt
echo ${id2:0:2}${id2: -3:3} >> output.txt
echo ${id3:0:2}${id3: -3:3} >> output.txt
```

Problem 5

Write a script to redirect the manual entry for a command which is specified via variable "command" to a text file man_command.txt (example: if the command is ls, then man_ls.txt) and redirect the standard error to man_command.err. If there is any error, execute the help command and redirect the output to man_command.txt

Prefix

command="fg"

Solution

```
man $command 1> man_$command.txt 2> man_$command.err || help $command >
man_$command.txt
```

Problem 6

Write a script to redirect the stderr of a command which is specified via variable "command" to a text file error.txt and print the std out only if there is no error.

Hint: Store the std out to some file then based on success or failure of the command print that file.

Prefix

```
command="cat qqq.txt"
echo "hello world" >qqq.txt
```

Solution

```
$command 1>temp.txt 2> error.txt && cat temp.txt
```

Problem 7

You are writing a Bash Script and in one of the task, you have to count the number of occurrences of some word in several files and finally the count should be in the variable `ct`. Write a Bash command that can be used in a loop to count the number of occurrences in all files, supposing that word count is stored in the variable named `words`. Your command will be used in a loop that will be run for each file. The value of `ct` should be initialized to 0, at the beginning, and the number of words stored in the variable `words` is to be added for your command. Achieve this task in just one line.

Hint: Write a Bash command that is equivalent to pseudocode "ct=ct+words", where `ct` and `words` are variables.

Solution

```
ct=$((${ct:=0}+words))
```

Week - 5, Code with us

Problem 1a

Write a Bash script that accepts a name(string) as input from `stdin` and prints "hello `input-name`>" as output, where `input- name` is the input string.

Example: If the input string is `Raghu` the output should be `hello Raghu`

Write your script in the file `myScript.sh`.

main.sh

```
bash myScript.sh
```

Solution(myScript.sh)

```
read name
echo "hello $name"
```

Problem 1b

Write a Bash script that accepts a name(string) as command line argument to your script `myScript.sh` and prints "hello `input-name`" as output, where `input- name` is the input string.

Example: If the command line input string is `Raghu` the output should be `hello Raghu`

Write your script in the file `myScript.sh`.

main.sh

```
read var
bash myScript.sh $var
```

Solution(myScript.sh)

```
echo "hello $1"
```

Problem 1c

Write a bash script that reads two numbers from the standard input and prints the sum of the numbers. Assume that the input will be numeric only. Write your script in the file `myScript.sh`.

main.sh

```
bash myScript.sh
```

Solution(myScript.sh)

```
read a
read b
echo "$[ $a+$b ]"
# OR
#echo $(($a+$b))
#OR
#echo `expr $a + $b` #space before and after '+' is important here
#OR
#sum=`expr $var1 + $var2`
#echo $sum
```

Problem 2

Write a bash script that takes two arguments, checks if both the arguments are positive integers then prints their sum; else prints "NOT INTEGERS" to STDERR and exit with exit code 1.

Note: Use the below if else conditional statment if needed

```
if condition; then
    ...
    ...
else
    ...
    ...
fi
```

Main.sh

```
read arg1
read arg2
((arg1 + arg2))
if [ $? = 0 ]; then
  bash ./script.sh $arg1 $arg2
else
  bash ./script.sh $arg1 $arg2 1> /dev/null
  echo $?
fi
```

Solution

```
if [ `echo $1 | egrep '^[0-9]+$'` ] && [ `echo $2 | egrep '^[0-9]+$'` ]; then
  echo $(($1+$2))
else
  echo "NOT INTEGERS" >&2
  exit 1
fi

# Or can use the below if condition to check for integers
#pat='^[0-9]+$'
#if ! [[ $1 =~ $pat && $2 =~ $pat ]]; then
```

Problem 3

Write a bash script that takes two arguments, checks if both the arguments are positive integers then prints their sum; else concatenate the string values in both the arguments and prints the combined string.

Solution

```
if [ `echo $1 | egrep '^[0-9]+$'` ] && [ `echo $2 | egrep '^[0-9]+$'` ]; then
  echo $(($1+$2))
```

```
else
  echo $1$2
fi

# Or can use the below if condition to check for integers
#pat='^[0-9]+$'
#if ! [[ $1 =~ $pat && $2 =~ $pat ]]; then
```

Problem 4

Write a bash script that accepts a file path as an argument and checks if that exists and is readable by current user and prints the output as below.

- Prints "DOES NOT EXIST" on STDERR and return with error code 1 if the file does not exist at the given path.
- Prints "NOT READABLE" on STDERR and return with error code 2 if the file is not readable by current user.
- Prints "WOO HOO" if the file exists and is readable too.

Note: Use the below if elif conditional statment if needed

```
if condition; then
    ...
    ...
elif condition; then
    ...
    ...
else
    ...
    ...
fi
```

Solution

```
if ! [ -e $1 ]; then
  echo "DOES NOT EXISTS" >&2
  exit 1
elif ! [ -r $1 ]; then
  echo "NOT READABLE" >&2
  exit 2
else
  echo "WOO HOO"
fi
```

Problem 5

Write a bash script which takes in the value of n and prints it in reverse order (For example if input number is 123, the Output will be 321)

Solution

```
n=$1
counter=0
ans=0
while [ $n -gt 0 ]
do
    counter=$(( $n % 10 ))
    ans=$(( $ans * 10 + $counter ))
    n=$(( $n / 10 ))
done
echo $ans
```

Problem 6

Write a bash script that accepts an integer say n and prints the below pattern till n lines.

```
*
**
***
****
*****
```

In the above sample the value of n is 5.

Solution

```
n=$1
i=1
while [ $i -le $n ]; do
  j=1
  while [ $j -le $i ]; do
    #printf "*"
    echo -n "*"
    j=$((j+1))
  done
  echo
  i=$((i+1))
done
```

Problem 7

Write a bash script which takes in the value of n and prints whether the number is prime or not. If n is a prime number, the program must print "Prime" and if not, it must print "Not Prime"

Solution

```
flag=0
n=$1
for (( i=2; i<=$n; i++ )); do
    if [ $((n%i)) -eq 0 ]; then
        flag=1
    fi
done
if [ $flag -eq 0 ]; then
    echo "Prime"
else
    echo "Not Prime"
fi
```

Problem 8

`df -h` gives the disk/filesystem usage information. Write a bash script to list all the filesystem mount point names based on their percentage usage divided in 5 categories in the format below.

```
0-50
(names of filesystem one in each line with usage between 0 to 50%)
50-75
(names of filesystem one in each line with usage between 50 to 75%)
75-85
(names of filesystem one in each line with usage between 75 to 85%)
85-95
(names of filesystem one in each line with usage between 85 to 95%)
>95
(names of filesystem one in each line with usage above 95%)
```

In each category print the range in one line followed by the filesystem mount point names. Print the range string even if there are no filesystem with usage in that range. Your script should not print anything else, all other errors and ouput from your script should be redirected to /dev/null.

Filesystem mount point name is the last field in the output of df -h.

The categories are

- 0% to less than 50% usage.
- 50% to less than 75% usage.
- 75% to less than 85% usage.
- 85% to less than 95% usage.
- Equal and above 95% usage.

Hint: Can store the df command output in a file. Then work on the file named `dfOutput.csv` line by line using

```
while read -r line;
do
   echo $line; # To print the line.
   # Write your code to process the line.
done < dfOutput.csv
```

Use ${var:0:-1} to remove the last character of string var.

Solution with arrays

```
df -h >dfOutput.csv

ar1=()
ar2=()
ar3=()
ar4=()
ar5=()
while read -r line
do
  var=`echo $line | cut -d " " -f 5`
  usage=${var:0:-1}
  if [[ `echo $usage | egrep '^[0-9]+$'` ]]; then
    if [[ $usage < 50 ]]; then
      ar1+=(`echo $line | cut -d " " -f 6`)
    elif [[ $usage < 75 ]]; then
      ar2+=(`echo $line | cut -d " " -f 6`)
    elif [[ $usage < 85 ]]; then
      ar3+=(`echo $line | cut -d " " -f 6`)
    elif [[ $usage < 95 ]]; then
      ar4+=(`echo $line | cut -d " " -f 6`)
    else
      ar5+=(`echo $line | cut -d " " -f 6`)
    fi
  fi
done < dfOutput.csv

echo '0-50'
printf '%s\n' "${ar1[@]}"
echo '50-75'
printf '%s\n' "${ar2[@]}"
echo '75-85'
printf '%s\n' "${ar3[@]}"
echo '85-95'
printf '%s\n' "${ar4[@]}"
echo '>95'
printf '%s\n' "${ar5[@]}"

rm dfOutput.csv 2>/dev/null
```

Solution with files

```
df -h >dfOutput.csv

touch range1 range2 range3 range4 range5 2>/dev/null
while read -r line
do
  var=`echo $line | cut -d " " -f 5`
  usage=${var:0:-1}
  if [[ `echo $usage | egrep '^[0-9]+$'` ]]; then
    if [[ $usage < 50 ]]; then
      echo $line | cut -d " " -f 6  >>range1
    elif [[ $usage < 75 ]]; then
      echo $line | cut -d " " -f 6  >>range2
    elif [[ $usage < 85 ]]; then
      echo $line | cut -d " " -f 6  >>range3
    elif [[ $usage < 95 ]]; then
      echo $line | cut -d " " -f 6  >>range4
    else
      echo $line | cut -d " " -f 6  >>range5
    fi
  fi
done < dfOutput.csv

echo '0-50'
cat range1
echo '50-75'
cat range2
echo '75-85'
cat range3
echo '85-95'
cat range4
echo '>95'
cat range5

rm dfOutput.csv range1 range2 range3 range4 range5 2>/dev/null
```

Problem 9

In Problem 8, modify the output of your script as below.

- Print the range string only if there is a filesystem in that range.

For example if there is no filesystem with usage >95% and also none in the range 75-85, and rest all range has atleast one filesystem with usage in that range than your output should be

```
0-50

(names of filesystem one in each line with usage between 0 to 50%)
```

```
50-75

(names of filesystem one in each line with usage between 50 to 75%)

85-95

(names of filesystem one in each line with usage between 85 to 95%)
```

Solution

```
df -h >dfOutput.csv

touch range1 range2 range3 range4 range5 2>/dev/null
while read -r line
do
  var=`echo $line | cut -d " " -f 5`
  usage=${var:0:-1}
  if [[ `echo $usage | egrep '^[0-9]+$'` ]]; then
    if [[ $usage < 50 ]]; then
      echo $line | cut -d " " -f 6  >>range1
    elif [[ $usage < 75 ]]; then
      echo $line | cut -d " " -f 6  >>range2
    elif [[ $usage < 85 ]]; then
      echo $line | cut -d " " -f 6  >>range3
    elif [[ $usage < 95 ]]; then
      echo $line | cut -d " " -f 6  >>range4
    else
      echo $line >&2
      echo $line | cut -d " " -f 6  >>range5
    fi
  fi
done < dfOutput.csv

if [[ -s range1 ]]; then
    echo '0-50'
    cat range1
fi
if [[ -s range2 ]]; then
    echo '50-75'
    cat range2
fi
if [[ -s range3 ]]; then
    echo '75-85'
    cat range3
fi
if [[ -s range4 ]]; then
    echo '85-95'
    cat range4
fi
if [[ -s range5 ]]; then
```

```
    echo '>95'
    cat range5
fi

rm dfOutput.csv range1 range2 range3 range4 range5 2>/dev/null
```

OPPE

Question 1 [15 mark] Write a Bash script that reads two inputs from stdin. The first line is a directory name present in the current directory. The second line is a filename. Print present if the file with filename given as second input is present in the directory name given as the first input else print absent . Hint: Refer the cheat sheet for file operators. Test case description: Input are the two lines of input that your script should read. Output is the output printed by your script. Solution read dir read file [[-a "$dir/$file"]] && echo present || echo absent

Question 2 [15 marks] Write a Bash script that reads an integer value (say n) from the standard input, creates a directory named image_files (in the current working directory) then creates n empty files inside the directory image_files each file with the name as image_i_rgb.txt , where i is the number of the file between 1 to n . (n>=1) For example, if n is given as 3, the following three files should be created in the directory image_files : image_1_rgb.txt , image_2_rgb.txt and image_3_rgb.txt . Initially the directory image_files does not exist in the current working directory. Test Case description: Input is the value of n and output is the the sorted list of files created. Printing of output is taken care by evaluation script, your script does not need to print anything. Solution mkdir image_files cd image_files read n for i in $(seq 1 $n); do touch "image_${i}_rgb.txt" done

or

#!/bin/bash num=3 mkdir test1 cd test1 for ((i=1; i<=$num; i++)) do touch file1_$i done

```
```