Week 1

GRPA 1

We created some directories and change our current working directory using the cd command as given by the sequence of commands below. Write a bash command to make the directory "level2" as your current working directory. i.e. after executing your solution, if we execute the command "pwd" it should return the path of the directory "level2".
Write your solution as a single line bash command.

cd /
mkdir level1
cd level1
mkdir level2
cd level2
mkdir level3
cd ..
cd ..

Solution

```
script() {
cd /level1/level2
pwd
}
```

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

Solution

```
script() {
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.

Solution

```
script() {
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.

Solution

```
script() {
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.

Solution

```
script() {
mv dir_1/file_1 dir_2
rm -rf dir_1
}
```

WEEK 2

GRPA 1

Print the absolute path where the command wget is located.

Solution

```
script() { echo '
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.

Solution

```
script() {
mkdir dir_1 dir_2
touch dir_1/file_1
ln -s /dir_1/file_1 /dir_2/file_2
#
# OR
# ln -s /dir_1/file_1 dir_2/file_2
#
# Solution 2
# cd dir_2; ln -s ../dir_1/file_1 file_2
}
```

GRPA 3

Print the username associated with the current session.

Solution

```
script() {
echo $USER
}
```

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.

Solution

```
script() {
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.

Solution

```
script() {
bash -c "echo hello; exit 179"
}
```

WEEK 3

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.

Solution

```
script() {
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.

Solution

```
script() {
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.

Solution

```
script() {
head half1.txt -n2 >> master.txt
tail half2.txt -n3 >> 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

Solution

```
script() {
wc -l 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('|').

Solution

```
script() {
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.

Solution

```
script() {
test | tee templog
cat templog >> log
}
```

WEEK 4

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.

Solution

```
script() {
grep -e "\ba\b\|\ban\b\|\bthe\b" poem -v | grep -e "\w" | 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.

Solution

```
script() {
grep -i "manager" employees.csv | grep "R&D" > 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.

Solution

```
script() {
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

Solution

```
script() {
#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\}$"`)

lower=(`ls |egrep "[[:digit:]]{4}[[:xdigit:]]{6}"|egrep "[[:lower:]]|[[:digit:]]
{10}"`);
upper=(`ls |egrep "[[:digit:]]{4}[[:xdigit:]]{6}"|egrep "[[:upper:]]"`);
}
```

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.

Solution

```
script() {
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

# Alternate solution
# egrep "$state" -i Pincode_info.csv | egrep "[0-9]{6}" -o | egrep "(.)....\1" |
wc -l
}
```

WEEK 5

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.

Solution

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

GRPA 2-A

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.

Solution

```
script() {
if [ $# != 2 ]; then
  echo Error
fi

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

GRPA 2-B

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.

Solution

```
script() {
read n
num="^-?[0-9]*\.?[0-9]*$"
neg="^-"
if [[ $n =~ $num ]]; then
  [[ $n =~ $neg ]] && echo NNUM || echo PNUM
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.

Solution

```
script() {
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".

Solution

```
script() {
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.

Solution

```
script() {
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.txtscript.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

Solution

```
script() {
# 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.

Solution

```
script(){
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.

Solution

```
script() {
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
}
```

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.

Solution

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

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

Solution

```
script() {
# 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;}'
}
```

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.

Solution

```
script() {
sed -e '/^[[:digit:]]/s/delta/gamma/' input.txt
}
```

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.

Solution

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

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.

Solution

```
echo '
#!/usr/bin/sed -f
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\####
' | col > myscript.sed
```

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.

Solution

```
script() {
tar cf headers.tar project/*.h
gzip --keep headers.tar
}
```

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.

Solution

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

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 (,).

Solution

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

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.

Solution

```
script(){ echo '
BEGIN{
 FS=",";
 sum=0;
}
{
 sum=sum+NF;
}
END{
 print sum;
}
' >yourScript.awk
awk -f yourScript.awk marks.csv
}
```

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.

Solution

```
script(){ echo '
BEGIN{
 FS=",";
 sum=0;
}
/^[0-9].*[0-9]$/{
 print $0;
 sum=sum+1;
}
END{
 print sum;
}
' >yourScript.awk
awk -f yourScript.awk marks.csv
}
```

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

Solution

```
eof="EOF"
while read file; do
  if [[ $file =~ $eof ]]; then
    break
  fi
  while read line; do
    if [[ $line =~ $eof ]]; then
      break
    fi
    echo $line >>$file
  done
done
######### Driver code ends here

######### Script starts here
for file in *.c; do
  awk '


    BEGIN {
      flag=0;
    }

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

    END {
      if (flag==1) print FILENAME;
    }
  ' $file
done
```

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.

Solution

```
myCount(){
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

}
```

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.

Solution

```
script(){ echo '
BEGIN{
    FS = ","
}

{
    EID = $1
    Gender= $4
    if (Gender ~ /Female/) {
        print EID"@xyz.com"
        }
}
' >yourScript.awk
awk -f yourScript.awk EmployeeDetails.csv
}
```

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.

Solution

```
script() { echo '
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];
  }
}
' > yourScript.awk
awk -f yourScript.awk EmployeeDetails.csv
}
```

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.

Solution

```
script () { echo '
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))
}
' >yourScript.awk
awk -f yourScript.awk EmployeeDetails.csv
}
```

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.

Solution

```
script() { echo '

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

' > yourScript.awk
awk -v item=$1 -v n=$2 -f yourScript.awk groceries.csv
}
```

WEEK 8

GRPA 1

Write a Bash script to reconstruct the Shopping Bill dataset from the OCR data files and store it in a separate file for each card. The reconstructed dataset should contain one file for each card with the name shopping_bill_<card_number>.txt, "<card_number>" is a number of the card e.g. '1', '4', '12', '24' etc.

The generated files should contain the data in the format given below.
SHOPNAME:CUSTOMER_NAME:CARD_NO
Item:Category:Qty:Price:Cost
Carrots:Vegetable/Food:1.5:50:75
....

Where information like shopname, category, item. cost etc. can be fetched from their respective 'ocr_.txt' files. Or you can find all the information in a single file named 'ocr_full.txt'

Solution

```
script() {
#!/usr/bin/bash

# getting all the names and storing in an array
NAMES=($(cat ocr_names.txt |
  uniq |
  grep -v '\[' |
  egrep -o '[[:alpha:]]+$'))

NAMES[9]=Rajesh # manual correction of data
NAMES[22]=Julia # manual correction of data

# getting all the shop names and storing in an array
readarray -t SHOPS < <(cat ocr_shopname.txt |
  uniq |
  grep -v '\[')

FILEPAT='^\[' # to capture the metadata:card number
count=0 # card count to keep track of card number
while read line; do

  # the metadata about the card name is bounded by [ ]
  if [[ $line =~ $FILEPAT ]]; then
    ((count++))
    cardNo=$(echo $line | egrep -o '[[:digit:]]+')
    cardNo=${cardNo#*0}
    # replace with the count if cardNo read is not valid
    [[ $cardNo == "" ]] && cardNo=$count
    SHOP=${SHOPS[$((cardNo - 1))]} # current shop name
    NAME=${NAMES[$((cardNo - 1))]} # current customer name
    {
      echo "$SHOP:$NAME:$cardNo"
      echo "Item:Category:Qty:Price:Cost"
    } > shopping_bill_$cardNo.txt
  else
    it=($line) # convert the string `line` to array `it`

    # check for the enough fields and prevention of unwanted data
    echo $line | egrep -q "$SHOP|$NAME"
    if [[ $? != 0 ]] && [[ ${#it[@]} -ge 5 ]]; then
      # get the field values
      cost=${it[$((${#it[@]} - 1))]}
      price=${it[$((${#it[@]} - 2))]}
      quantity=${it[$((${#it[@]} - 3))]}
      category=${it[$((${#it[@]} - 4))]}
      item=${line% $category*}

      # correction done by observing the data from the data set
      # 3 is read as 'a' in some field
      echo $quantity | egrep -q '^[[:digit:].]+$'
      [[ $? != 0 ]] && quantity=${quantity/a/3}
      echo $price | egrep -q '^[[:digit:].]+$'
      [[ $? != 0 ]] && price=${price/a/3}

      # printing the field value in the desired structure
      echo $item:$category:$quantity:$price:$cost >> shopping_bill_$cardNo.txt
    fi
  fi
done < <(cat ocr_item_record.txt |
  uniq |
  grep '^[A-Z\[]' |
  grep -v '^Item' |
  sed 's/[=]//g') # preprocessing the data files

# manual validation with checking the total cost with the original dataset
#for file in shopping_bill_*.txt; do
#  echo $file:$(cat $file |
#    sed -n '1,2! p' |
#    rev |
#    cut -d: -f1 |
#    rev |
#    awk '{sum+=$0}END{print sum}')
#done
}
```