Operating System Lab Report – Assignment 1

Name : Tanmoy Sarkar
Roll No : 002010501020
Class : BCSE 5th semester 3rd Year
Group : A1

1. Write a shell script that has 2 user created variables, uv1 and uv2. Ask for the values of the variables from the user and take in any values (real/integer/character) for the 2 variables. Test the program for different types of uv1 and uv2.
(a) Print them as:
    (i) value of uv1 followed by value of uv2 separated by a comma and
    (ii) value of uv2 followed by value of uv1 separated by the word “and”.
(b) Print the variables in reverse order [If uv1 is 1234, then output should be 4321

Code -

```
#! /bin/bash
# Read input
echo -n "Enter uv1 : "
read uv1
echo -n "Enter uv2 : "
read uv2
echo "============================="
echo $uv1$uv2
echo $uv2$uv1
echo "============================="
# Reverse function
function reverse() {
    local tmp
    local reverse
    local length
    tmp=$1
    reverse=""
    length=${#tmp}
    for ((i=length-1; i>=0; i--)); do
        reverse="$reverse${tmp:i:1}"
    done
    echo $reverse
}
uv1__reversed=$(reverse $uv1)
uv2__reversed=$(reverse $uv2)
# Print part b.
echo "============================="
echo $uv1__reversed
echo $uv2__reversed
echo "============================="
```

Output -

```
tanmoy@tanmoy-laptop:~/Desktop/Lab/OS/ass1$ ./q1.sh
Enter uv1 : 1234
Enter uv2 : 6789
```

```
============================
12346789
67891234
============================
============================
4321
9876
============================
```

Explanation -
We are taking the user input for uv1 and uv2 by using `read` and store in variables. Then we print the values in the preferred format using `echo`.
To reverse the string we have written a function ` reverse` which takes a string as an argument and returns the result . In that function, we iterate from the end of the string and append that to a temp variable. At the end , we get the reverse of the input.

2. Write a shell script to count the number of lines in a file. Test if the file is present. If not, create and write.

Code -

```
#! /bin/bash

FILENAME=$1
# Check if file exists
if [ -f $FILENAME ]; then
    echo "File exists"
else
    echo "File does not exist"
    echo "Creating file"
    read  -p "Enter something to write in file : " content
    echo $content >> $FILENAME
fi
lines=0
# Count lines
while read line; do
    lines=$(($lines+1))
done < $FILENAME

echo "Number of lines in file: $lines"
```

Output -

```
tanmoy@tanmoy-laptop:~/Desktop/Lab/OS/ass1$ ./q2.sh test.txt
File exists
Number of lines in file: 8
```

tanmoy@tanmoy-laptop:~/Desktop/Lab/OS/ass1$ ./q2.sh testfile.txt
File does not exist
Creating file
Enter something to write in file : ABCDEFGH
Number of lines in file: 1

Explanation -

In this program, we take the file name as a command line argument. In the program, we can access the filename for $1 . To check the existence of the file and also to ensure that it's not a directory, we are using the file operator ` -f FILENAME `. If the file doesn't exist, we will take content from user input by `read` and put the content in a new file by using the`>>` operator.
To count the number of lines, we will iterate over the lines of the file, and will increase the variable `lines` by 1 . At the end, we print the no of lines by `echo`

3. Write a shell script that counts the number of ordinary files (not directories) in the current working directory and its sub-directories. Repeat the count of files including the sub-directories that the current working directory has.

Code -

```
#! /bin/bash

BASE_PATH=$1
DIRECTORIES_COUNT=0
FILES_COUNT=0
if [ -d $BASE_PATH ]; then
    echo "Path exists"
else
    exit 0
fi
function countFiles(){
    local filesCount
    filesCount=0
    for f in "$1"/*; do
        if [ -d "$f" ]; then
            v=$(countFiles "$f")
            filesCount=$(($filesCount+$v))
        else
            filesCount=$(($filesCount+1))
        fi
    done
    echo $filesCount
}
```

```
x=$(countFiles $BASE_PATH)
echo "Total no of files : $x"
```

Output -

Folder Structure of folder “test” -

```
test
├── a.txt
├── b.txt
├── test copy
│   ├── a.txt
│   └── b.txt
└── test copy1
    ├── a.txt
    ├── b.txt
    └── test_copy
        ├── a.txt
        └── b.txt
```

Terminal output -

```
tanmoy@tanmoy-laptop:~/Desktop/Lab/OS/ass1$ ./q3.sh test
Path exists
Total no of files : 8
```

Explanation -

In this program, we take the folder name as a command line argument. In the program, we can access the folder name for $1 . To check the existence of the folder and also to ensure that it’s a directory, we are using the file operator ` -d FOLDER_NAME `. If the directory doesn’t exist, exit from the program.

To count the number of files in the directory and its sub-directory , create a function called `countFiles`. It’s a recursive function and it takes a directory path and returns the total number of files in it.

In the `countFiles` function it iterates over the files/folders in the path, and then checks whether it’s a file or directory. If it's a file, it will increase filesCount by 1, and if it’s a directory will call `countFiles` in that directory path.

At the end , the program will print the total no of files in that directory.

4. Write a shell program to duplicate the UNIX rm command with the following features:

a. Instead of deleting the files, it will move them to a my-deleted-files directory. If the file already exists in the my-deleted-files directory, then the existing file (in the my-deleted-files) will have the version number zero (0) appended to it and the newly deleted file will have version number one (1) appended to it. Go on incrementing the version nos., if required.

b. The command will have a switch -c that will clear the entire my-deleted-files directory after asking for confirmation.

Code -

```
#! /bin/bash
DELETED_FOLDER_NAME="my-deleted-files"
#functions
# Get file name/folder name from filepath
function getFileName(){
    IFS='/'
    read -a SPLITTED_AT_DELIMETER <<<"$1"
    unset IFS
    FILENAME=${SPLITTED_AT_DELIMETER[-1]}
    echo $FILENAME
}

# Get preferred version no to append to file name by looking in
my-deleted-files folder
function getPreferredVersionNo(){
    IFS="."
    read -a SPLITTED_AT_DELIMETER <<<"$1"
    unset IFS
    EXTENSION=""
    FILENAME=${SPLITTED_AT_DELIMETER[0]}
    if [ ${#SPLITTED_AT_DELIMETER[@]} -gt 1 ]; then
        EXTENSION=".${SPLITTED_AT_DELIMETER[1]}"
    fi
    VERSION_NO=0
    while :
    do
        if [ -z $EXTENSION ]; then
            # It maybe a folder
            if [ -d "${DELETED_FOLDER_NAME}/${FILENAME}_${VERSION_NO}"
]; then
                VERSION_NO=$(($VERSION_NO+1))
            else
                break
            fi
        else
            # It maybe a file
```

```
            if [ -f
"${DELETED_FOLDER_NAME}/${FILENAME}_${VERSION_NO}$EXTENSION" ]; then
                VERSION_NO=$(($VERSION_NO+1))
            else
                break
            fi
        fi
    done
    echo "${FILENAME}_${VERSION_NO}${EXTENSION}"
}


# MAIN PROGRAMM
# If found -c delete the folder
if (( $# == 2 )); then
    if (($2 == '-c' )); then
        echo "Deleting the folder"
        rm -r $DELETED_FOLDER_NAME
    fi
fi

# Check if folder exists
if [ -d $DELETED_FOLDER_NAME ]; then
    echo "Directory exists"
else
# If not found , create the folder
    echo "Directory does not exist, created a new one"
    mkdir $DELETED_FOLDER_NAME
fi

FILENAME=$(getFileName $1)
FILE_PATH_WITH_VERSION=$(getPreferredVersionNo $FILENAME)

# move the files/folder to the folder
mv $1 "$DELETED_FOLDER_NAME/$FILE_PATH_WITH_VERSION"
```

Output -

```
tanmoy@tanmoy-laptop:~/Desktop/Lab/OS/ass1$ ./q4.sh xx.zip
tanmoy@tanmoy-laptop:~/Desktop/Lab/OS/ass1$ ./q4.sh xx.zip
tanmoy@tanmoy-laptop:~/Desktop/Lab/OS/ass1$ ./q4.sh test
tanmoy@tanmoy-laptop:~/Desktop/Lab/OS/ass1$ ./q4.sh test
```

After running the scripts, the folder structure of `my-deleted-files`

```
my-deleted-files
```

```
├── test_0/
├── test_1/
├── xx_0.zip
└── xx_1.zip
```

Running the script with argument -c

tanmoy@tanmoy-laptop:~/Desktop/Lab/OS/ass1$ ./q4.sh test -c

After running the script, the folder structure of `my-deleted-files`

```
 my-deleted-files
├── test_0/
```

Explanation -
At the start of the program, we check for the command line argument `-c` , If found it will delete the `my-deleted-files` folder.

Then check the existence of “my-deleted-files”. If it does not exist , create the directory by `mkdir`. The program will accept the file/folder that needs to be deleted as a command line argument.

To get only the file name or directory name from the complete path, we have created a function `getFileName` . This function will set IFS to “/” and split the complete path at “/” and convert in an array. The last element of the array will be the file name / directory name.

Next, we need to check in the folder whether the same file/folder already exists or not. Depending on that, I need to generate the correct version no according to the question. For that purpose, we have created the `getPreferredVersionNo` function. It will iterate over the all files/folders in the `my-deleted-files` folder and will increment the version no accordingly. The function will return the new file/folder name.

At the end the program uses the `mv` to move the file/folder to the `my-deleted-files` folder with a new file/folder name.

5. Write a script called birthday_match.sh that takes two birthdays of the form DD/MM/YYYY (e.g., 15/05/2000) and returns whether there is a match if the two people were born on the same day of the week (e.g., Friday). And then find out the age/s in years/months/days.

Code -

```
#! /bin/bash
function formatDate(){
    # DD/MM/YYYY to YYYY/MM/DD
```

```
    IFS='/'
    read -a SPLITTED_AT_DELIMETER <<<"$1"
    unset IFS
    echo
"${SPLITTED_AT_DELIMETER[2]}/${SPLITTED_AT_DELIMETER[1]}/${SPLITTED_AT_D
ELIMETER[0]}"
}

function convertSecondsToHumaizedFormat(){
    # Seconds to humanized format
    SECONDS=$1
    YEARS=$(($SECONDS / (86400*365) ))
    SECONDS=$(($SECONDS % (86400*365) ))
    MONTHS=$(($SECONDS / (86400*30) ))
    SECONDS=$(($SECONDS % (86400*30) ))
    DAYS=$(($SECONDS / 86400 ))
    SECONDS=$(($SECONDS % 86400 ))
    echo "$YEARS years, $MONTHS months, $DAYS days"
}

echo -n "Enter the first birthday : "
read FIRST_BIRTHDATE
echo -n "Enter the second birthday : "
read SECOND_BIRTHDATE

IFS="/"
read -a FIRST_BIRTHDATE_ARRAY <<<"$FIRST_BIRTHDATE"
read -a SECOND_BIRTHDATE_ARRAY <<<"$SECOND_BIRTHDATE"
unset IFS

FIRST_BIRTHDATE_FORMATTED=$(formatDate $FIRST_BIRTHDATE)
SECOND_BIRTHDATE_FORMATTED=$(formatDate $SECOND_BIRTHDATE)

# Required format -> YYYY/MM/DD
FIRST_BIRTHDATE_DAY=$(date -d "$FIRST_BIRTHDATE_FORMATTED" '+%A')
SECOND_BIRTHDATE_DAY=$(date -d "$SECOND_BIRTHDATE_FORMATTED" '+%A')

if [ $FIRST_BIRTHDATE_DAY == $SECOND_BIRTHDATE_DAY ]; then
    echo "Birthday matches !!"
else
    echo "Birthday does not match !!"
fi

# Todays date
TODAYS_DATE_FORMATTED=$(date '+%Y/%m/%d')
```

```
# Difference in seconds
FIRST_AGE_SECONDS=$(( $(date -d "$TODAYS_DATE_FORMATTED" '+%s') - $(date
-d "$FIRST_BIRTHDATE_FORMATTED" '+%s') ))
SECOND_AGE_SECONDS=$(( $(date -d "$TODAYS_DATE_FORMATTED" '+%s') -
$(date -d "$SECOND_BIRTHDATE_FORMATTED" '+%s') ))

# Convert seconds to humanized format
FIRST_AGE_HUMANIZED=$(convertSecondsToHumaizedFormat $FIRST_AGE_SECONDS)
SECOND_AGE_HUMANIZED=$(convertSecondsToHumaizedFormat
$SECOND_AGE_SECONDS)

echo "first person age : $FIRST_AGE_HUMANIZED"
echo "second person age : $SECOND_AGE_HUMANIZED"
```

Output -

tanmoy@tanmoy-laptop:~/Desktop/Lab/OS/ass1$ ./q5.sh
Enter the first birthday : 15/07/2002
Enter the second birthday : 24/01/2001
Birthday does not match !!
first person age : 20 years, 1 months, 18 days
second person age : 21 years, 7 months, 10 days

tanmoy@tanmoy-laptop:~/Desktop/Lab/OS/ass1$ ./q5.sh
Enter the first birthday : 15/07/2002
Enter the second birthday : 22/08/2022
Birthday matches !!
first person age : 20 years, 1 months, 18 days
second person age : 0 years, 0 months, 5 days

Explanation -

We have created a function `formatDate` that will convert date format from DD/MM/YYYY to YYYY/MM/DD format by using IFS and convert it to an array of string and then restructure it in YYYY/MM/DD format.

There is another function `convertSecondsToHumaizedFormat` , that will take seconds and return in … years,... months, … days format.

At the start of the program, We take two birthdays as input from the user in DD/MM/YYYY format. At the next step, set IFS to “/” and convert to birthdays [DD/MM/YYYY] to array [DD,MM,YYYY]

Then used the `formatDate` function to convert it preferred format for `date` module We get the week day from date module by (date -d “Formatted_Date” ‘+%A’)

Then check both days of the dates , and print the result about whether they match or not in the terminal.
Then we extract seconds from the date module by (date -d “Formatted_Date” ‘+%s’). We subtract it from current seconds. Now we got the age in seconds. Now use the `convertSecondsToHumaizedFormat` function to convert it to …. Year, …month … days format.

At the end, we print the ages of the two birthdays in the terminal by `echo`.

6. Write a shell script that accepts a filename as an input and performs the following activities on the given file. The program asks for a string of characters (that is, any word) to be provided by the user. The file will be searched to find whether it contains the given word. If the file contains the given word, the program will display (a) the number of occurrences of the word. The program is also required to display (b) the line number in which the word has occurred and no. of times the word has occurred in that line (Note: the word may occur more than once in a given line). If the file does not contain the word, an appropriate error message will be displayed.

```
#! /bin/bash
read -p "Enter the file name: " FILE_NAME

if [ -f $FILE_NAME ]; then
    echo "File found !"
else
    echo "File not found"
    exit 0
fi

read -p "Enter the word to search: " SEARCH_WORD

lines=1
total_occurrences=0

while read line; do
    occurrences=0
    for word in $line; do
        if [ $word == $SEARCH_WORD ]; then
            occurrences=$(($occurrences+1))
        fi
    done
    total_occurrences=$(($total_occurrences+$occurrences))
    lines=$(($lines+1))
    if [ $occurrences -gt 0 ]; then
        echo "Line $lines: $occurrences"
    fi
```

```
done < $FILE_NAME

if [ $total_occurrences -gt 0 ]; then
    echo "Total occurrences of ->$SEARCH_WORD<- : $total_occurrences"
else
    echo "->$SEARCH_WORD<- not found"
fi
```

Output -

Content of “test.txt” -

Apple
Bag
Cow
Donkey
Apple
Cat

Terminal Output -

tanmoy@tanmoy-laptop:~/Desktop/Lab/OS/ass1$ ./q6.sh
Enter the file name: test.txt
File found !
Enter the word to search: Apple
Line 2: 1
Line 6: 1
Total occurrences of ->Apple<- : 2

Explanation -

The filename is collected from the terminal by showing a prompt by (read -p “enter file name” FILE_NAME) . Next to check the existence of the file we have used file operator (-f FILE_NAME). If this does not exist, the program will print “File not found !” in the terminal and exit the programme. If it exists, it will proceed to the next step.

The word that needs to be searched, will be taken as user input.

After that, we will start iterating lines of the content of the file. And then iterate each word in the file. If the word matches the searched word , we increment the `occurrences` in that line and print total no of occurrences of that word in that line. Also update the `total_occurrences` in file.

At the end, if `total_occurrences` seem to be zero, then it will print “Search_Word not found”, else print total number of occurrences of the word in file.

7. Extend the shell script written in (6) to perform the following task: User is asked to enter two different patterns or words. The first pattern will have to be matched with the contents of the file and replaced by the second pattern if a match occurs. If the first pattern does not occur in the file, an appropriate error message will be displayed.

Code -

```
#! /bin/bash

read -p "Enter the file name: " FILE_NAME
if [ -f $FILE_NAME ]; then
    echo "File found !"
else
    echo "File not found"
    exit 1
fi
read -p "Enter the word to search: " SEARCH_WORD
read -p "Enter the word to replace: " REPLACE_WORD

lines=1
total_occurrences=0
updatedContentFile=""
while read line; do
    occurrences=0
    for word in $line; do
        if [ $word == $SEARCH_WORD ]; then
            occurrences=$(($occurrences+1))
            updatedContentFile="$updatedContentFile$REPLACE_WORD "
        else
            updatedContentFile="$updatedContentFile$word "
        fi
    done
    updatedContentFile+="\n"
    total_occurrences=$(($total_occurrences+$occurrences))
    lines=$(($lines+1))
    if [ $occurrences -gt 0 ]; then
        echo "Line $lines: $occurrences"
    fi
done < $FILE_NAME

if [ $total_occurrences -gt 0 ]; then
    echo "Total occurrences of ->$SEARCH_WORD<- : $total_occurrences"
    echo -e "$updatedContentFile" > $FILE_NAME
    echo "File updated successfully"
else
    echo "->$SEARCH_WORD<- not found"
fi
```

Output -

Currently, content of “test.txt” -

Apple
Bag
Cow
Donkey
Apple
Cat

Terminal output -

tanmoy@tanmoy-laptop:~/Desktop/Lab/OS/ass1$ ./q7.sh
Enter the file name: test.txt
File found !
Enter the word to search: Apple
Enter the word to replace: Banana
Line 2: 1
Line 6: 1
Total occurrences of ->Apple<- : 2
File updated successfully

After running the script, content of “test.txt” -

Banana
Bag
Cow
Donkey
Banana
Cat

Explanation -

To add support to replace the search_word with an another word (given by user input), we have created a variable `updatedContentFile`. While iterating over the words, when we will found the search_word, inspite of that word we will add the replace_word and if not found, will add the word in updatedContentFile as it is. At the end of the program, we have updated content of file in `updatedContentFile` variable. We will write the content in the file by ‘>’ operator. To count ‘\n’ as line feed, we need to add -e after echo , (echo -e “Content” File_Name)