CSE231: OPERATING SYSTEMS

ASSIGNMENT-1 (WRITE-UP)

RAHUL KHATOLIYA (2019265)

1. INTERNAL COMMANDS :

a.) Echo Command : “ Write arguments to the standard output ”

# Options Handled :

1. “-e” option :

Using this Option The User can Print formatted Text in the standard output , using format specifiers , which in our case are \t ,\n,\b, and \v .

2. Standard option :

This is the conventional method , which simply takes arguments , and print it untouched on standard output .

# Implementation :

Inorder to determine the option first , I have checked whether is their any given token token passed as an option indicator , if yes , than check , is it “-e” or not , and if it is actually , then proceed to formatted text parsing. For the standard case , i have parsed the string after the ‘echo’ , and printed as it is, whereas in the case of ‘-e’ option , i have created a short algorithm , which will work on character by character of the respective parsed string , and will check for those special format specifiers to work on , and eventually printing out the filtered result .

# Assumptions :

We had assumed that whenever , the user needs to work on “-e” option , then she/he should type the string to be formatted , inside the “ ” . However , for the standard case the user can provide the text either in “ ” or simply as it is , it won’t matter . Moreover, special characters such as “ / ” , would be removed/ignored after processing the format , and the string will be free of this character .

# Errord Handled :

1. **It may happen that by mistake if user forgot to type any argument in the standard case , and only write echo ,then user will be promted “ Incomplete arguments ... ”.**
2. **Similarly , suppose for the “-e” option , if the user doesn’t provide any text , meaning only types “echo -e” , then here the user will get the prompt “Arguments Missing for "-e" type echo command ”.**
3. **Finally , suppose for using the “-e” option , if the user typed it like “echo -E <text>” , then he/she will be asked whether “ Do you mean "echo -e <text>” ”**

# Test Cases :

1 . echo “Hello,World”

output: Hello,World

2. echo Ram is a good boy

output: Ram is a good boy

3. echo -e “ram\tgood\tperson”

output: ram good person

b.) Cd command : “ Changing to a given directory if exists ”

# Options Handled :

1. “ .. ” :
This option is used to move to the just previous directory of the current directory , and is useful for quick navigations around directories .

2. Standard Option :
This is the standard option , which will take path as an argument from the user , and will try to enter that respective directory if possible .

# Implementation :
So for this command , the ultimate feature “chdir” is used , which is a method in c library that accepts string buffer as an argument , and change the directory to the given directory respectively . However , their might be the case where the directory may have a name containing space, thus i had made a small algorithm for that rescue.

# Assumptions :

We have assumed that whenever , the user needs to locate a directory having spaces in its name ,then the user must provide the entire name within “ ” , to get to this location correctly , However if the user fails doing so , then he won’t get to this location , as far as she/he maintain the format . Whereas , for directories which don’t have any space or special character , can simply pass the argument as it is after the command cd , to get to this path successfully . And for any othe inputs , the program will look for the exact name if present , and will ignore , even if a small or minor difference is present . At last , it is assumed that directories will not have any names containing special characters.

# Errors Handled :

1. **Similar to original terminal , if the path provided along with the command , was invalid , i.e doesn’t exist , then the user will be prompted message , claiming “ No such Directory Exists ”.**
2. **Moreover, for the Directories having names containing spaces , might be invalid or**

mistyped , by the user so , she/he will be prompted for the same , if any such circumstances are found .

# Test Cases :

1. For folder/directory name ram currently present in the same directory in the program is running :

```
  >>home/
   cd ram
  output : >>home/ram
```

2. Similarly , to go to a directory having name “ram personal” :

```
>>home/
 cd “ram personal”
>>home/ram personal
```

3.

```
>>home/rahul
  cd ..
>>home/
```

c.) Exit Command :   “ Used for terminating a running program , with different exit status”

# Options Handled :

1. exit --help :
This options enables the user , to have an Idea , of what type of command is this , and what are its available options .

2. exit [n] :
This option is responsible , for terminating with a particular number in return , such as 0 for success , -1 for fail etc .

3. exit:
Simply terminate and return to the shell .

# Implementation :

The approach for this implementation was brief and simple , for “--help” case , we just have to throw the output which usuallly appears in the original terminal , on execution of the same command option , whereas for the other option , we just passed the argument provided by the user as , the termination status of the exit code .

# Assumptions :

The command replicates the exact behaviour of the original command , when we give some irrelevant arguments such as , exit xyz , where it simply end the programa and get terminated , irrespective of the arguments passed . Therefore , we have provided the legal termination according to the implemented option if any , otherwise it simply get terminated .

# Error Handled :

As described above , their is no as such error case for this type of internal command , as it runs perfectly according to the condition.

d.) History Command : “ Used to print the terminal’s entire entered history ”

#Options Handled :

1. History !! :
It simply print out the last executed/entered command on the terminal/shell .

2. Standard Command :
It is the standard command for printing out the finite number of commands which were previously entered/executed in the shell.

#Implementation :

For this Commands implementation I have used pointers to the HISTORY_STATE and HIST_ENTRY using the methods history_get_history_state () and history_list() respectively using a, and then printed out in sequence the desired history , with respectives attributes such as time stamps and line of the pointers above stated .

# Assumptions :

Only a single Assumption , as we are working on the Current Session History , the user must use the command option History !! only when , she/he has types a few commands on terminal earlier , since if their were no commands then the pointer will have null values and we will be getting in Segmenatation fault Core dumped error .
Moreover , In original terminal , to throw out the last executed command , the syntax is !! which we have assumed to be history !! .

# Errors Handled :

1. The User might mistyped , or provide unrelevant flags , for e.g history -m etc , which are not available in our shell , then the user will be prompted for the same , getting “Unsupported/Invalid history Type Command”, in the standard output .

2.Moreover , the casing of the letter ‘h’ of history might be used as ‘H’ as History , therefore , we will give the suggestion to the user , that "Do you mean "history" ", so that to make things clear about the format used for the command to user.

#Test Case :

1.history !!
output : Last executed command : pwd

2.history
output :
Current Session history

Exit
Cd
clear
cleaar
history
clear

e.) pwd Command : " Used for generating the path of program's current working Directory"

# Options Handled :

1. pwd -P :
This is similar to the standard pwd function , which prints the current working directory.

2.pwd -L:
This is However different , since it uses PWD from environment , even if it contains symbolic links , and print the resultant directory.

3.pwd :
This is the standard pwd command .

# Implementations :

For the Implementation purpose of both the commands , we get handy with the getcwd method of the c library, which strictly makes a string corresponding to path using the environment PWD , and even work if their are symbolic links . Thats how both the flags are able to achieved from the getcwd method , which not only here , but is useful in many other cases .

# Assumptions :

No Assumptions .

# Error Handled :

1. As usual , their might be times , that user might mistypes irrelevant flags with the pwd command , for e.g :- pwd -o , which doesn't have any actual implementation inside the shell , hence we throw out a prompt showing the user " Unsupported or Invalid type pwd Option " , which will make things clearer to user.

2.However , their might be minor cases , like the user typed casing different , for e.g Pwd instead of pwd , so in that case we will print “ Do You Mean “pwd” ”, in the standard output of the terminal .

# Test Cases :

1. pwd -L
output:home/user/rahul

2.pwd -P or pwd
output: home/user/rahul/assignment

2. EXTERNAL COMMANDS :

a.) ls Command :

“ Used to print the names of the files currently present in the current directory ”

# Options Handled :

1. ls -a :

   Used for printing All the files , including the Hidden Ones .

2. ls -1 :

   Used for printing the non-hidden files , each at a new line .

3. Standard ls command :

   Similar to printing line -a option , but excludes hidden files if any .

# Implementations :

Here in our code , we have used Dirent Structure , getcwd() method , and readdir() method of the C library , Which helps to view the details of Files present in the current directory using pointers to each corresponding file .
However , to work on command specific , we had not printed the hidden files for -1 and standard ls command option .

# Assumptions :
No Assumptions .

# Errors Handled :

1. Here the behaviour is similar to the original terminal command , when the user use this command inside a directory , where no hidden/ non-hidden files are present , then shell will not crash , instead it ignores such cases and came back to the initial stage of accepting inputs for commands.

2. By some mistake , if the user types an invalid flag or command , then user will be prompted for the " Unsupported/Invalid type ls command " , and return to the shell .

b.) mkdir Command : " Used to Create Directories , inside the current working directory "

# Options Handled :

1.mkdir -p file1/file2/file3 :
This command is used to create parent directories , in the sequence provided for the respective names.

2. mkdir -v file1 file2 file3 :
This command is used to create multiple directories within the current working directory .

3. standard mkdir <filename> :
This is simplest command including one argument , i.e to make a directory of a given name inside the current directory .

# Implementations :

For this command , the most helpful command was mkdir() , which accepts a file name and mode , and create the directory for it . For -p we will parse the Buffer input into meaningful and desired names and loop through each to make corresponding directories for each . Similarly , for -v Command option , the methodology is same , but here the task is somewhat easy as , we just have to loop in the current directories itself for creating these desired directories , unlike in the case of -p where each time the directory inside which we have to made is changed subsequently .At last the simplest implementation was for standard command .

# Assumptions :

Here we have assumed , that for the standar mkdir command , suppose a user want to create a directory having space in its name , then user just need to type the file name as is , for e.g to make a folder "ram shyam" , user needs to type – mkdir ram shyam , this will successfully create the directory for the same . Secondly , for command option -p , the user should be successful only if the the the format is – mkdir -p ram/shyam , moreover the file name here can't include space or special character , otherwise it will work correctly .At last , for -v command , the user should use format – mkdir -v f1 f2 to make directories having name f1 and f2 in the current working directory respectively , here also the filename can't include space or special character .
In the command options , if the directory was found to be already existing then it will get ignored and the remaining shall be proclaimed , maintaining the order .

# Errors Handled :
1. if the user tried to make a directory which is either present already , or might don’t get created due to some internal error , then the mkdir() method will return some integer correponding to it , and hence using the perror method , we will throw that corresponding error on the standard output .

2.It may happer , that users forgot to throw argument to the command , and simply type only mkdir , then user will be prompted “ mkdir: missing operand ” , so that to clear things to user .

3. Similar to original terminal , for command option -v , the user shall be prompted , everytime when a file is created or not created , to make the user aware of pre-existing directories and bugs creating the same if any .

# Test Case :

1. mkdir ram
creates a directory having name “ram”

2. mkdir -p ram/shyam/suraj
follows the order -> suraj inside shyam inside ram . i.e ram/shyam/suraj

3. mkdir -v hello bye hii
makes the three directories , “hello” , “bye”, “hii” in the current working directory .

4. mkdir romantic songs
makes a folder namin “romantic songs” in the current directory .

c.) Cat command : “ Used to get/print details of the file in the directory ” .

# Options Handled :

1. cat -n <filename> :
To print the contents of the file , line by line with line number on the standard output .

2. cat file1 > file2 :
To Copy the contents of file1 into file 2, in exact fashion inside file1 .

3. cat <filename> :
To print the contents of the file on the standard ouptut .

# Implementations :

Here the entire mehtodology revolves around I/O services provided by the C library , in which , by using suitable input/output stream File type pointers and handy methods such as fgets(), fputc() etc , we can read or write successfully on the given file .

However , only for the “>” option , we are using both read and write operations at the same time , while not in the case of “-n” command .

# Assumptions :

Here we have Assumed that , for Command option “-n” if the user needs to perform it for file name having space in it , then user should simply write it as its, for e.g -
cat -n Ram shyam , to perform command for the file “Ram shyam” . Similarly for , the standard option , the same convention is to be followed . Note that , for “>” option the user can’t provide a filename having space or special character in it .
# Error Handled :

1. For any of the command options , it may happen that the filename user is providing might not be presented actually , or by some internal mishappening the pointers are unable to fetch informations for the same , so for these cases , we have used the perror , which will indicate the type of misunctioning to the standard output for the user.

2. Secondly , it may happen that the user mistyped some irrelevant / or in appropriate flags along with the command , so for such cases , the user will be prompted “ Unsupported/Invalid Cat type Command ” , on the standard output , to make the things clear to the user .

# Test Cases :

1.cat ram shyam
output :
prints the entire contents of the file “ram shyam” on the standard output .

2.cat -n ram shyam
output :
prints the entire contents of the file “ram shyam” line by line along with the line number on the standard output .

3. cat file1 > file2
output :
copy entire contents of file1 in file2 .

d.) Date Command : “ To get the details of the time and date in specified format ”

# Options Handled :

1. date -u :
Used for getting the Date and Time format in the UTC/GMT zone .

2. date -R:
Used for getting output date and time in RFC 5322 format.

3. date :
Used for getting output date and time in National format , IST in our case .

# Implementations :

We have used the Time.h C library to easily get details , and pointers according to desired options , and eventually specifying the format in which the user actually looking for . Some useful methods such as localtime() , time() , gmtime() were frequently used for the desired results .

#Assumptions :

No Assumptions .

# Errors Handled :

1. If due to some internal Api failures , or misfunctioning , the methods such as time() , localtime() , etc failed to provide legal values , then the situation is immediately prompted to user , using the perror .

2.Now it may happen the entire command provided by the user might be correct but the flag provided differs in casing , therefore for each such flag , if it matches with our flags then the user will be prompoted " Do you mean <flag> " , to make the user aware for such cases .

2. At last , it may happen that the user mistyped some irrelevant / or in appropriate flags along with the command , so for such cases , the user will be prompted " Unsupported/Invalid date type Command " , on the standard output , to make the things clear to the user .

# Test Cases :

1. date
output : Wednesday 30 September 2020 04:36:47 PM IST

2. date -u
output : Wednesday 30 September 2020 11:07:05 AM GMT

3. date -R
output : Wed, 30 Sep 2020 16:37:49 +0530

e.) Rm command : " Used for removing files from directories " .

# Options Handled :

1. rm -i <file1> <file2> :
Used for removing each file , but taking confirmation from user for every file .

2. rm -f <file1> <file2>   :
Used for removing each file , and not even asking once , since its a strict rm command .

3. rm <file name> :
Standard command to remove the file if present in the directory .

# Implementations :

So here we have just processed the input buffer , and filetered it into corresponding file name to be removed , and finally removed that using the method remove() using C library . However , for option "-i" we will ensure from the user that she/he agreed or not for the removal of the file or not ,and perform accordingly .However for the other option , the scenario is completely opposite , the user will not be confirmed , but the file if exists will automatically get removed for this command .

# Assumptions :

For using the "-i" and "-f" options , the user must provide the filename , not having spaces or special characters , for e.g :- rm -i r1.txt r2.txt r3.txt , or similarly
rm -f r1.txt r2.txt r3.txt to removes the files in sequence , asking or not asking the user accordingly .
However , to remove a single file having name containing spaces , user can simply type rm ram shyam , which will successfully remove the file "ram shyam" if present in the directory .

# Errors Handled :

1.Similar to original terminal command , here for our every implemented command , if the file name provided do not exist or can't be removed because of internal failures , then such cases would be directly informed to the user using perror method .

2.if the user forgot to pass any argument after rm , then she/he will be notified , prompting " rm: missing operand " .

3.It may happen that flag option provided by the user might differ in casing , therefore for such cases we will prompt " Do You mean rm <flag> " .

4.At last , the user might sometimes provide ,irrelevant flags which are not handled originally , therefore user will be prompted " Unsupported/Invalid rm type Command ".

# Test Cases :

1. rm ram shyam
output: removes the file having name "ram shyam" if any , or throw suitable error .

2. rm -i ram shyam geeta
output: removes the files "ram" , "shyam" , and "geeta" if exist or throwing error , and asking each time for confirmation .

3. rm -f ram shyam geeta
output: removes the files “ram” , “shyam” , and “geeta” if exist or throwing error , not asking for any confirmation .

Note : for any minor change such as casing of letters , the command line will throw suitable errors .