INTERNAL COMMANDS:

- cd:
    - This command is used to help the user change the directory from inside the terminal.
    - I have implemented options here: cd, cd .. , cd <directory name>
    - Used the chdir function to handle this part of the code.
    - Used if/else to handle and throw errors
    - Here I have handled errors like:
        - cd <file_name that does not exist>: my code throws an error in a situation like this, "Error: cd, no such file or directory: file_name"
        - Further, the code also throws errors if more than one argument is passed. Exampled: cd <file1> <file2>.

- pwd:
    - This command prints the present working directory.
    - I have implemented the options of pwd, pwd -l, pwd -p
    - I get the output that tells me the present or current working directory.
    - Here I have handled errors like:
        - If the user tried to input more arguments after the letter pwd, for example pwd -l lk, the I raise an error that says: “Error: invalid command for pwd”.
        - Further, if the user tries to input a wrong letter as argument after pwd, example: pwd -z, an error is raised that specifies: “Error: -z is an invalid argument for pwd”.

- echo
    - In echo I have implemented echo, echo -n, echo –help
    - Here in echo -n, I print out the arguments of echo’s function onto the terminal without a trailing \n
    - Further, help is like a function that tells the user about the options I have created in echo, it provides the user with information about the echo internal command.

EXTERNAL COMMANDS

- mkdir
    - The use of this command is to create a directory inside the current path of the user.
    - I have implemented it as: mkdir, mkdir -v, mkdir -pv
    - mkdir: it creates the directory inside the current path, but throws an error if the file already exists or if for some reason, it cannot be created in the current directory.
    - Further, mkdir -v: gives the verbose information to the user, it tells the user about the successful creation of the directory, it throws an error in case of insufficient arguments, example: if a user has input “mkdir -v”, that is insufficient arguments

for us to make the directory, hence, I raise an error, as we need at least one file name.
    - Another error i have handled is if the user tries to input only “mkdir”, that is an insufficient command and it throws an error.
    - For mkdir -pv, I am able to create a nested directory when I pass args in form dir1/dir2/dir3, I have used recursion to achieve this.
    - If &t is at the end of the command, then we use systems and thread to implement the functionality.
    - Else we use execl and fork to do the same.

- date
    - A simple function that outputs the date and time as demanded by the user.
    - I have implemented the functions date, date -r, date -u
    - date -u: gives me the UTC date and time output
    - date -r: tells me about the date when the file was last modified
    - Error: if we miss a file argument, then it throws an error
    - If we provide too many invalid arguments, it throws an error
    - If &t is at the end of the command, then we use systems and thread to implement the functionality.
    - Else we use execl and fork to do the same.
- ls
    - Tells us about the contents of the directory
    - ls - i : prints the Inode numbers corresponding
    - ls - a: also includes directory ids that start with a dot
    - I have handled errors where the user is notified if the arguments given to ls are invalid or incorrect
    - If &t is at the end of the command, then we use systems and thread to implement the functionality.
    - Else we use execl and fork to do the same.
- rm
    - Used to remove files and directories from the directory.
    - rm -v gives a verbose explanation of the directory deleted and informs the user about the success of the process
    - rm -d allows the user to remove the directories
    - Error thrown is user tries to remove a non-existent directory or file
    - Error thrown if user tries to remove a directory from rm, instead of rm -d
    - Error thrown if user tries to input just rm, we tell him we need more arguments or file names
    - If &t is at the end of the command, then we use systems and thread to implement the functionality.
    - Else we use execl and fork to do the same.
- cat
    - cat, cat -n, cat -b implemented
    - If user tries to input cat -b with an invalid file or directory name an error is thrown

- We number all lines, including blank ones when we handle cat -n
- Don't number the blank lines in cat -b
- Further cat, just prints out the entire file for the user
- We can also operate cat and the options without a file name, there it lets user input words and lines and performs function accordingly.
- If &t is at the end of the command, then we use systems and thread to implement the functionality.
- Else we use execl and fork to do the same.

TEST CASE AS ON TERMINAL