BASIC LINUX SHELL

INTERNAL COMMANDS:

1. exit : It exits the shell gracefully, displaying an exit message prior to exiting.
2. cd: cd refers to 'change directory'.
    - With one argument, cd <input> is used to change the present working directory to the one mentioned in the input.
    - cd.. changes the working directory to the parent directory of the current directory.
3. echo : The echo command is used to display the line of a text. Here it neglects the extra space if present before or after the command.

    - echo -E : It explicitly suppresses the interpretation of backslash escapes.
      ```
      >>echo -E abcdef\nghi      xyz
      ```
      It will output abcdefnghi xyz
      (Neglects extra space and ignores backslash escape sequences.)
    - echo -n : It doesn't append a new line.
      eg. `>>echo -n "abc          def"`
      It will output abc def (i.e it neglects the extra space)
4. history
    - history: shows the complete history of commands entered by the user. It will also display the latest command including the command history.
    - history -c: It clears the entire command history. To check it, run the command history and it will only give the output as the very latest command -history.

5. pwd : pwd(Present Working Directory) shows the current working directory.

EXTERNAL COMMANDS:

For external commands, I extracted the files from the bin folder(in Linux) and used the execlp commands.
So all the commands can be executed whose binary files are present at the bin location.
Some of the external commands are :
mkdir makes a new directory
ls:lists all files
date: displays the time date
cat: concatenates files
rm: deletes files

I have used an assumption that all external commands are functional if they have less than or equal to 2 command words. Eg. mkdir <input> , ls, rm <input>, rmdir <input>, cat<input-file> , touch <new-file> etc.

To execute the file, compile the C code using make command. Then run the file using ./code