Assumptions

The project development and testing involve the use of both Windows and Mac OS. We have structured our tests into distinct folders for unit tests ("unit_tests"), integration tests ("integration_tests"), and system tests ("system_tests"). Additionally, auto-generated tests derived from tools such as PITest and Evosuite are available for reference within the "automated_tests" folder.

General

1. The shell will persistently wait for and execute commands until the "exit" command is explicitly provided.
2. Our program supports the following consoles and methods to end standard input:
    a. For Windows Command Prompt and Windows PowerShell: Ctrl+Z
    b. For MacOS Terminal, Windows WSL, and Linux terminal: Ctrl+D

Basic Functionality (BF)

1. Calling applications
    a. No assumptions.
2. Quoting
    a. Unmatched double quotes within double quotes throws invalid syntax error since double quote content should not contain double quotes.
        i. **Example**: echo """
    b. Unmatched backquotes within double quotes throws invalid syntax error since double quote content should not contain backquotes.
        i. **Example**: echo "`"
3. Command substitution
    a. If the program encounters command substitutions with newlines within backticks, an Invalid Syntax exception is thrown.
        i. **Example**: echo `echo hello <\n>`
        returns: shell: Invalid syntax
        ii. This behaviour deviates from Bash, where Bash does not consider newlines within backticks as invalid syntax.
4. echo
    a. No assumptions.
5. cd
    a. "-" is treated as a path/directory when used as an argument for this command.
        i. **Example**: cd -
        ii. This behaviour deviates from Bash, where Bash will change the current directory to the previous directory you were in. Instead, our program will print "cd: -: No such file or directory" if the current directory does not have a child directory called "-".
6. wc
    a. If the program encounters non-existent files/files that cannot be read/files that are directories, it will continue to parse and count the valid files, while throwing errors for the invalid ones
        i. **Example**: wc <NonExistentFile.txt> <ExistentFile.txt>
        returns: wc 'NonExistentFile.txt': No such file or directory

```
<lines>   <words>   <bytes>   <ExistentFile.txt>
<lines>   <words>   <bytes>   total
```

    b. If the program takes in both standard input (stdin) and file, stdin is taken in all at once and then merged to the file contents.
        i. **Example**: wc <-> <test.txt>
        ii. This behaviour deviates from Bash, where Bash takes in stdin and merges with file contents line by line.

7. mkdir
    a. Files are created relative to the current working directory.
    b. The program allows taking a root directory as the first character of the argument.
        i. **Example**: mkdir /A
        ii. This would create the file "A" at the root of your program instead of the current working directory.
        iii. This behaviour deviates from Bash, where Bash does not allow it.
    c. If the program receives multiple directories, and one of the directory is invalid, the program will still make all the valid directories and throw errors for the invalids.
        i. **Example:** mkdir <valid> <invalid> <valid>
        returns: both valid directories created + program prints out "mkdir: cannot create directory: '<invalid>': File or directory already exists"
8. sort
    a. When both the \`-n\` and \`-f\` flags are specified together, the \`-n\` flag takes precedence over \`-f\`. This precedence means that the sorting will be performed numerically based on \`-n\` and the \`-f\` flag will be ignored.
    b. For this application, a dash ("-") is treated as a file input rather than as a standard input.
        i. This behaviour differs from Bash, where it is treated as a standard input.
9. cat
    a. Like wc, commands taking in standard input (e.g., \`cat -\`) takes in all lines at once before printing out on standard output, as compared to being printed line by line in Bash.
10. exit
    a. The shell does not support running applications in the background; therefore, the "exit" command does not need to perform any cleanup work, such as releasing resources.

Extended Functionality 1 (EF1)

1. Pipe operator
    a. No assumptions.
2. Globbing
    a. Globbing is applicable for asterisk (*) only. It does not extend to other wildcard patterns such as the question mark (?).
    b. Globbing is applicable only when it is not nested within directories.
        i. **Valid example:** src/sg/* OR src/sg/*.java
        ii. **Invalid example:** src/*/edu OR /*/src
3. ls
    a. If the program encounters non-directory file arguments, it will proceed by printing out the absolute file name instead of throwing an error.
        i. **Example**: ls path/filename
        returns: filename
    b. If the program receives multiple arguments, and one of the arguments is a non-existent file, the program will not print out those valid files. Instead, it will throw a single error.
        i. **Example:** ls <ExistDir1> <NonExistDir1> <ExistDir2>
        returns: ls: cannot access '<NonExistDir1>': No such file or directory

        ii. **Example:** ls <ExistDirWithNoReadPerm1> <NonExistDir1> <ExistDir1>
        returns: ls: cannot access '<NonExistDir1>': No such file or directory
    c. If the program receives multiple non-existent files as arguments, it will only throw the first non-existent file encountered.
        i. **Example:** ls <NonExistDir1> <NonExistDir2>
        returns: ls: cannot access '<NonExistDir1>': No such file or directory
    d. If the program receives multiple existent files as arguments, and some files do not have permission to read, it will print the contents in the files and permission denied error in order of the files.
        i. **Example:** ls <ExistDir1> <ExistDirWithNoReadPerm1>
        returns:
        ExistDir1:
        ls: cannot open directory '<ExistDirWithNoReadPerm1>': Permission denied
        ii. **Example:** ls <ExistDirWithNoReadPerm1> <ExistDir1>
        returns:
        ls: cannot open directory '<ExistDirWithNoReadPerm1>': Permission denied
        <empty line>
        ExistDir1:

4. paste
    a. If the program receives both files and standard input ("-") as arguments, and a file that cannot be found is specified, the program will immediately throw FileNotFoundException without waiting for standard input.
        i. **Example:** paste <NonExistFile1.txt> <->
        returns: paste: '<NonExistFile1.txt>': No such file or directory
    b. Valid directories given as argument will simply be skipped, and other arguments will be processed. No exception is thrown.
    c. When processing in parallel and there are multiple standard inputs ("-") entries, each standard input entry takes one line at a time sequentially, rotating among them. For example, the first "-" entry takes the first line, the second "-" entry takes the second line, and so on. Once all "-" entries have taken one line each, the process repeats.
    d. When processing in serial and there are multiple standard inputs ("-") entries, the first standard input entry will consume the entire line.
5. uniq
    a. If the user wishes for input should be read from standard input but output to a file, then INPUT_FILE should be specified to be "-" instead of not specifying at all.
    b. Processed string will end with a newline
    c. When writing to standard output, newline (before new command) is provided by *Application
6. mv
    a. If the target directory is provided as a source directory, an IOException will be thrown.
    b. If the program receives multiple source arguments, and one of the source arguments is a non-existent file, the program will still move the valid sources to the destination and throw errors for the invalids.
    c. **Example:** mv <valid> <valid> <invalid> <target>
    returns: both valid files moved to target + program prints out "mv: '<invalid>': No such file or directory"

Extended Functionality 2 (EF2)

1. IO-redirection
    a. Unused methods such as terminate() and trivial methods such as getters were not tested.
2. Semicolon operator
    a. Unused methods such as terminate() and trivial methods such as getters were not tested.

   b. Commands ending with semicolon behave exactly as though as the command’s last semicolon is not present, like Bash.
   c. In the case of an exit command being part of a sequence of commands separated by semicolon, only commands before the exit command will be executed before the shell is terminated by exit. Commands after the exit command will not be executed, like Bash.

3. cut
   a. Either the "-b" or "-c" flag must be included, and only one is permitted. Otherwise, it will throw an error message stating "cut: Only one flag can be selected".
      i. **Example:** cut -c -b 1-5 file.txt
         return: cut: Only one flag can be selected
   b. A range must also be included to specify the position for cutting. If an invalid range is provided, it will throw an error message stating "cut: invalid byte/character position: '<invalid>'".
      i. **Example:** cut -c a file.txt
         return: cut: invalid byte/character position: 'a'
   c. The cutting range must begin with a positive integer. Otherwise, it will throw an error message stating “cut: byte/character positions are numbered from 1”.
      i. **Example:** cut -c 0 file.txt
         return: cut: byte/character positions are numbered from 1
      ii. **Example:** cut -c 0-10 file.txt
         return: cut: byte/character positions are numbered from 1
   d. The cutting range must be in increasing order. Otherwise, it will throw an error message stating "cut: invalid decreasing range: '<range>'".
      i. **Example:** cut -c 10-1 file.txt
         return: cut: invalid decreasing range: ‘10-1’
4. rm
   a. The -d flag becomes effectively useless when used in conjunction with the -r flag, as the -r flag already results in the deletion of both empty and non-empty directories.
   b. Therefore, there is no distinction between using -r and -rd.
5. tee
   a. All outputs, such as with adding input to a new file, ends with a newline.
   b. Including a directory in the command will not throw an exception [and terminate the session], instead it prints an error message and continues to run for other arguments given, similar to in Bash.
   c. If an unwritable file is given as an argument, an exception is thrown.
6. grep
   a. Prints “grep: Invalid syntax” whenever syntax does not follow the command format specified in the project description (grep [Options] PATTERN [FILES]).
   b. Given a command that greps from file before standard input (-), the grep output for the file will only be printed after the standard input is closed (ctrl+D/ctrl+Z).
      i. **Example**: grep -icH abc test/abcfile.txt – invalid