Assumptions

General

- Unlike actual GNU where "\" can be used to escape space character, our software needs double quotes to specify files or keywords with spaces in them
- This Shell Implementation doesn't support storing history for storing and retrieving variables in commands

Application Commands

- Rm
    - If any of the files provided does not exist, the rm command will continue removing all other files and later report that the inexistent files were skipped for that reason.
    - Unlike the TDD cases seem to imply, our implementation of rm disregards file permissions completely.
    - Just like GNU rm, it is not possible to remove the current ('.') or the parent directory ('..') or any directory ending in '/.' or '/..'.
- Echo
    - Just like GNU echo, this should print a new line at the end of output in any call, so when called without arguments, an empty line is printed.
- Paste
    - Unlike GNU paste, the command is not interactive in the sense that it does not work as a read-evaluate-print-loop (REPL). It gets all input from the user, and then assembles the output only once.
    - When pasting contents from STDIN, in order to finish the input the user must enter an empty line. If the command is pasting STDIN with a file, the number of lines in the file is completely independent from how many lines the user can input, but in this case the empty line no longer applies and the app will continue reading input until EOF.
    - When pasting contents from STDIN, this implementation supports choosing the number of columns and their order. That is `paste -` will put all output in a single column and `paste - -` will separate it into two columns. If a file is included, such as `paste FILE -` the order is respected, making this different from `paste - FILE`.
    - When pasting only files, the command produces the same output as GNU paste.
    - Unlike GNU paste, this utility does not support being called without arguments, and should print a message warning the user of insufficient arguments.
- Sed
    - When the input is through STDIN, in order to complete the command Ctrl+D must be pressed. This terminated the program currently as well. For now, we have

  - decided to leave this implementation as given as we aim to come up with a better way in dealing with commands using multiple lines
  - Replaces content of file based on rule
  - The replacement text is exact and not a regex contrary to actual GNU shell
  - The replacement rule has to be quoted since it contains special characters to separate the replacement rule
  - Only two arguments maximum can be provided with the first being replacement rule. The second argument if provided must be a file with read permission
  - The separator used in a replacement rule can be any symbol which will appear right after “s”. However, it cannot be present anywhere in regex or replacement rule and should be only used as separator
- Exit
  - Exit can be called with any number of arguments but unlike GNU exit it always terminates the shell with exit code 0, ignoring all arguments.
- Diff
  - The output of running `diff` for directories with different file content is similar to the GNU shell’s implementation of `diff` command output.
  - When two files differ in **multiple lines** and the flag of `isSimple` is false, the output shown is in this format below (all differences from File1 first followed by File 2):

    ```
    < [File 1]
    < [File 1]
    > [File 2]
    > [File 2]
    ```
- Grep
  - The given implementation is correct. That is, it doesn’t look for an exact match as the pattern provided. If any words begin with the pattern provided and has some extension after, it will be still counted and returned to display.
- Wc
  - The output of running `wc` is similar to GNU shell’s implementation of `wc`.
  - The user only supplies c, l or the w flag for the command.
- Cd
  - Unlike GNU cd, calling cd with multiple arguments causes an error to be thrown like GNU implementation.
  - Calling cd without any arguments causes it to go to the home directory, just like the GNU implementation, and throws no exceptions.
  - The root directory is /.

    - Absolute paths refer to paths that start with a /.
    - Not supporting cd - and cd ~ (and the equivalent cd)

- Cp
    - When called with only a file name (source) and no destination argument, the exception thrown uses the text in ErrorConstants.ERR_NO_ARGS.
    - When attempting to copy a file to the directory it is already in, an exception should be thrown, including the text “same file” in its message. This is to mimic the behavior of GNU cp.
    - When a file argument is not found, an exception with ErrorConstants.ERR_FILE_NOT_FOUND in its message should be thrown
    - When one of the files to be copied does not exist the operation continues for the remaining files and the problem is reported at the end.
- Cut
    - The output of running `cut` is similar to GNU shell’s implementation of `cut`.
    - The user only supplies c or the b flag for the command. (Running -cb results in an error).
    - Input can ONLY be a list of comma-separated numbers, a range of numbers or a single number following ascending order
    - Input can ONLY allow up to 2 comma-separated numbers in ascending order as the interface's method can't accept multiple positions
    - '0' should not be supplied as a single number, a range of numbers or comma-separated numbers. The official command throws this error `cut: [-cf] list: values may not include zero`
- Ls
    - No longer throws error messages when it has to output a non directory result. Substantial effort was put into resolving the earlier hackish implementation and now it behaves very close to actual GNU shell
    - If more than 1 argument provided
    - Only flags -R and -d are implemented based on design requirements
- Sort
    - The output of running `sort` is similar to GNU shell’s implementation of `sort`.
    - The user only supplies n, r or the f flag for the command.
- Find
    - If a file is not found, the result will be empty. No error will be shown.

- Mv
    - If the destination argument is an existing file then it will be replaced under default conditions if only one source operand. If overwriting is not allowed it will fail silently as expected. If both are the same files then no change is made.
    - Works with globbing if target directory/file is specified otherwise will give unexpected result when incorrect input given due to globbing implementation fault and passing wrong arguments
    - More than 2 arguments need to be passed where last one will be the destination argument
    - Overwriting a directory doesn't work when the destination directory is not empty as Java VM throws a @DirectoryNotEmptyException
    - Like GNU the arguments can be relative paths
    - No output written to stdout for success or failure situations
    - Only flag -n has been implemented based on design requirements
    - If file is to be renamed then number of source operands should be one

Shell Operators

- Call Command
    - For commands with stdin input, it will continue to take in input in multiple lines. Different commands use different techniques to resolve this
- Pipe Operator
    - Works similar to the GNU shell implementation
- Semicolon Operator
    - Command cannot start with a semicolon. It can only be used in the middle to add more commands into the same input to GNU.
- Globbing
    - Due to EF1 implementation, we haven't fixed the scenario where doing `ls src/*` would give all files under src directory with output relative to current directory i.e. `src/sg`
    - It produces several command arguments and hence can result in multiple arguments being passed into a command when only one was passed by the user.
    - E.g. mv *.md -> mv README.md REPORT.md which now things its two arguments
- Quoting

    - Since this is an EF1 implementation, we have not changed any implementation details.

- IO-Redirection
    - Redirecting both input and output to/from the same file should never be used. As of now, this causes the file to be empty, just like in bash.
    - Only supports single streams.
- Command Substitution
    - Unlike GNU implementation, this only supports BACK_QUOTES (`) and not $().
    - Implementation changed where it replaces NEW_LINE of sub command output to EMPTY space. Now it removes it at the end as it may give extra characters in result and changes the rest of new lines to empty space
    - On sub command being evaluated, it can produce multiple arguments as result and the length of argument will depend on how its quoted
    - Performed before arguments are split