# Assumptions

**General:**

- Only ascii characters. No other non-english languages. Applies to filenames and text.
- Flags always come before the FILES, unless stated otherwise. We follow the command format provided in the project description which does not allow for flags to come after the FILES.
- Flags are also not joined together and must follow the project description format; prefixed with -, unless otherwise specified. I.e. -cdD is invalid for uniq
- Options will be case-sensitive unless specified.
- All files given in all commands are read and write accessible.
- All the applications and shell features should be tested on Windows.

**Grep:**

- Grep with -H flag and Inputstream outputs “(standard input)” as filename (same as linux)
- -c flag applies for individual input source (same as linux)
- -c has precedence over -i and -H flags and application will output count
- Grep from stdin will only print result after stdin has closed due to implementation of skeleton
- grep pattern < example.txt > example.txt will result in example.txt being empty as inputStream and OutputStream open the same file. (unix has the same behaviour)
- Unable to register empty pattern Implementation errors when pattern is empty while linux takes empty pattern and returns whole file
- Repeated flags are ignored as in linux
- Due to implementation of Skeleton, only when valid inputs are given for stdin and files will output be printed in OutputStream.
    - If [FILES] = “- exampleFile -” Only Exception is shown

**Split:**

- No upper bound of bytes for split file size(since maximum file size is dependent on OS)
- Split line factor by maximum of 65536
- Presence of <whitespace> between [options] and LINES/BYTES
- Only first flag passed will be considered as a flag, the subsequent flag passed will be seen as a file and an error thrown if does not exist
    - I.e. split -l 4 -b 4 testfile, will consider -b a file.
- For lines split, behaviour will follow unix, files ending with an empty new line will have the empty new line printed
- Files can only be split into existing directories same as linux
    - split exampleFile dir/prefix errors if dir does not exist
- Assume max of 676 * 3 files when no prefix given and 676 * 3 files when prefix given

**I/O redirection:**

- Stdin and stdout can only be redirected once
- After redirection, there can only be 1 argument as a file, additional arguments not supported, interpreted from documentation
- Due to implementation of IORedirectionHandler, output streams will be opened whenever output redirection is specified in the command. As such, for applications that produce no output, i.e. Split, the file will be generated regardless.

**Globbing:**

- The current test cases in GlobbingTest.java are expected to only be supported and passed in Windows computers as the test cases we designed use file separators for Windows devices.
- Follows the assumptions of the tdd globbing test cases that were given, in order to replicate unix shell behaviour
- Globbing does not support file separators after the asterisk, this is to replicate unix shell behaviour
  e.g. “/*/a.txt” not supported
- Globbing supports both “/” and “\” as file separators
  e.g. “ls folder/x*” and ‘ls folder\x*’

**Tee:**

- When the append option is set, -a, if the file to append does not end with a newline character, the entered text via stdin will be appended to the end of the file, i.e. does not start on a newline on the file.
- When the command asks the user to input via stdin, the user must hit enter for every line in stdin, hence if the user executes Ctrl+D mid typing a line, that line will not be read.
- Warning messages will be printed out instead of throwing as an exception or written to outputstream, this was designed to replicate the tee commands behaviour on Linux.

**Wc:**

- Must hit enter for every line in stdin
- If the user executes Ctrl+D mid typing a line, that line will not be read.

**Cat:**

- Must hit enter for every line in stdin
- Last empty line, if it exists in a file, will not get printed.
- Exceptions does not specify file/directory name

**Cd:**

- Cd with no arg does nothing. Assume no notion of home directory
- Assume windows os. Absolute path starting from root directory, no notion of ~ home directory
- Exceptions does not specify file/directory name
- cd < README.md is noop

**Echo:**

- If symbols like single, double or back quotes are used consecutively for an odd number of times, it will be considered invalid syntax
- There will be unexpected behaviour when command substitution is used with “echo” without any arguments. This is because “echo” without any arguments will return an empty line which will be interpreted as an application in command substitution.

**Ls:**

- Just like in the actual shell, there will not be any slash behind the directory name when it is listed out
- There will be a line break between every folders when the contents of folders are listed out
- There is a line break between every directory when ls “*” or when in recursion mode just like in UNIX.

- When the arguments include wildcard (eg “ls *”), there will be a line break between every file displayed so as to ensure easier readability.
- If command is in recursion mode and there is an invalid file, no exception will be thrown and the remaining valid files will be processed as expected.
- “ls -R” will only accept directory as argument. If a file is inputted as an argument, an empty line will be printed.
- Error messages may differ from that in Unix shell.
- The way or order in which files and folders are displayed may differ from UNIX shell.
- If “-XR” options are used together, the contents of the current directory will be displayed first alphabetically by extension, followed by the contents of the subdirectory in alphabetical order by extension. There will be 2 newlines in between the contents of the subdirectory.
- Result of `ls` is interpreted as one argument because there can be spaces in the filename. If we use spaces as delimiter, then the operation would not work for filename with spaces. Thus “ls `ls`” will throw an exception if there is more than 1 file/directory.

**Mv:**

- There must be at least 2 operands for any operations
- An error will be thrown when it try to rename to an existing filename for “-n” option.
- For rename operations, there must only be 1 source operands and 1 destination operand
- For moving of files/folders, the target operands must always be a folder just like in UNIX.
- If move a dictionary into a subdirectory that is within the source dictionary (e.g mv dir2 dir2/subdir”, error will be thrown as expected in UNIX.
- If move a dictionary into the same directory (e.g mv dir1 dir1), error will be thrown as expected in UNIX shell.
- Only the specification mentioned in the project description is included, there will be no checking done for read/write permissions or hidden files due to differences between operating systems.
- For “mv dir1 dir2”, regardless of whether dir2 is used, dir1 will be moved into dir2.
- If any operands operation is invalid, it will throw exception. Eg. “mv notExist exist1 existDir”
- If the command try to overwrite an existing file, an exception will be thrown for “-n” option.
- If the command attempt to move a current existing file (that already exist in current directory) into current directory, no exception will be thrown and it will remain unchanged.
- Error messages may differ from that in Unix shell.

**Paste:**

- For serial commands like “paste -s - file1 - < file2 > result”, the output file will have all the lines from file2 (separated by tab), followed by (in next line) all the lines from file1 (separated by tab) on one line just like in UNIX.
- For serial commands like “paste -s - - < file2”, the output will have all the lines from file2 (separated by tab) on the same line
- If a file does not exist, an exception will be thrown and the entire command would not be executed. Eg. paste – notExist.txt > AB.txt. However, an empty output file (AB.txt) will still be created if specified
- There is no new line if the argument consists of only a single stdin. However, if there is multiple stdin (without file arguments), there will be a new line in output.
- If the inputStream(s) or file(s) is empty (“”), there will be a new line in output.
- For commands where the arguments consists of files (with and without stdin), there is no new line for serial and a new line for non-serial in output.
- There is no tab at the back of each line in the output when there are files with different numbers of lines just like in UNIX.

- No exception will be thrown when there are no arguments for paste according to the TDD test cases given.

**Uniq:**

- No newline printed on last line when outputting to files (same as unix)
- -c -d/ -c -D combinations would give no meaningful output as stated in forums and hence throw an exception
    - -d -D only combination of multiple syntax accepted
- Due to interface restrictions, uniq with stdin will only print uniq results after stdin is closed instead of echoing the result where possible.
- Repeated flags are ignored, same as linux
- Additional parameters/filenames provided behind the output filename will be ignored. Same as in unix environment
- Flags passed after input file will be treated as output filename instead
- Last empty line, if it exists in a file, will not get considered as an adjacent line against a previously empty line. I.e. file ends with 2 new lines will not be printed when -D flag is set.

**Quoting:**

- If single quoting is used to surround an arg and the arg consists of single quotes, throw exception
- Assume double quotes with unclosed backquote results in everything after backquote being ignored e.g. echo “hello`bye” -> hello
- Assume double quotes with unclosed double quote results in an invalid syntax (shell exception) e.g. echo “hello”bye” -> invalid syntax
- Assume single quotes with unclosed single quote results in an invalid syntax (shell exception) e.g. echo ‘hello’bye’ -> invalid syntax
- Unix resolves above two by fetching further input (a closing quote) from stdin, which our Java shell specification did not specify

**Rm:**

- The flags -r and -d are not mutually exclusive and their order does not matter, if -r and -d are used in the same command, -r flag will supersede -d.
  e.g. ‘rm -r -d not_empty_dir’ will be executed similarly to ‘rm -r not_empty_dir’.
- If the command includes multiple files, and 1 of the file is not able to be removed (e.g. because it does not exist), the command will not terminate, will skip over that fill and continue executing for remaining files
  e.g. ‘rm 1.txt not_found.txt 2.txt 3.txt’
- When the command is not able to execute smoothly, warnings/ exception messages are printed on the screen via System.out.println() instead of via Outputstream.write() this is because using the latter will cause the warning messages to potentially be used as inputs during redirection or pipe command, e.g: “rm not_exisitng_file | tee 5.txt”, which is not how linux behaves.
- Warning messages: “rm: missing operand” , “rm: cannot remove” and related are printed on the screen via System.out.println(). Doing this is instead of throwing exceptions prevent the premature termination of the command, thus replicating linux’s implementation more closely.
  e.g. ‘rm 1.txt not_found.txt 2.txt 3.txt’ and ‘rm | tee 1.txt’
  would terminate prematurely if exceptions were thrown.
- When there are multiple files for the input redirection, the command will throw a shell exception due to too many files specified.
  e.g. ‘rm -r < 1.txt 2.txt’ t

**Cp:**

- When copying a file (not folder), if the file already exists in the destination folder, the destination file gets overwritten by the copied file.
- If a source file/folder is the same as the destination file/folder, an error message would be shown and no copy operation is executed for that particular file/folder.
- File/Folder related error does not output the name of the file/folder in error.
- All folder copy will fail if -r/-R is not specified. (behavior according to Unix)
- -r/-R option must be specified as the first argument after `cp`. Otherwise, the -r/-R option will be treated as a file, which likely does not exist.
- Destination must be specified as the last argument.

**SemiColon/Sequence:**

- Follows project description of <command> ‘;’ <command>, meaning inputs into shell ending with a semicolon will throw an error unlike unix. E.g. echo test; (errors)

**PMD rules**

God class: Possible God Class
Implementation of TODO in GrepApplication had already triggered this warning. Refactoring private methods to a Utils file and using GrepArgs still triggered PMD. Triggered PMD after refactoring file to be 4 methods (run, files, stdin and files&stdin). Violation of rule is also not clear on how to resolve, even with googling.

ClassNamingConventions:
Environment class violating but, environment does not act as util and does not require helper or Constants.

PreserveStackTrace:
Application specific exception messages require new exceptions to be thrown causing stack trace to be lost.

Close Resource:
In order to solve this PMD, we would need to close the opened streams whenever an exception is caught and thrown. As this project is IO intensive in each method, this PMD rule would conflict with the ExcessiveMethodLength PMD rules.