Team 22 Assumptions Report

This report contains assumptions made in our implementation. Including clarifications made to the applications specification.

Shell specification Assumptions

Call Command

Call Command follows the grammar specified in the specifications and allows for command substitution.

Pipe Operator

1) As stated in the specification, if an exception occurred in any of the parts, the execution of the part is terminated hence for example, lsa | echo hello world where lsa is an invalid application. The lsa would throw an exception hence, the expected output of the above command is shell: lsa: Invalid app.
   The above behaviour differs from in unix where it will still echo hello world despite the exception thrown.

Sequence Operator

1) As stated in the specification, if an exception occurred in the execution of the first command, the execution of the second part can continue. Our shell would outputs the exception message followed by a new line for the first command and also execute the second command. **Note:** We ignore may return a non zero exit code as this is similar to the Pipe Operator case regarding non zero exit code mentioned in the Q&AReport that we can ignore.
   For example: lsa; echo hello world
   Where lsa is an invalid application.
   Our shell behaviour (outputs):
   shell: lsa: Invalid app
   hello world

IO Redirection

1) For input redirection, if the input is a directory, then it will throw an File not Found Exception. For example, ls < src where source is a folder will throw File not found exception..

Command Substitution

1) When the shell executes “echo `exit`”, the program will exit automatically. This is different from the Unix shell where the console will display an empty line.

Application specification Assumptions

rm

1) The rm command when used with the **-rd** or **-r -d** or **-r** flags (including the permutation of the flags) would attempt to delete all files and/or folders input as arguments for the rm command together with the contents of non empty folders.

2) The rm command would only throw one exception (the latest exception) when there are potentially one or more exceptions due to the files and/or folders passed into the rm command arguments. This behaviour defer from unix which throws more than one exception when there are more than one exceptions due to the files and/or folders passed into the rm command arguments.
   An example: *rm hello 1.txt hello2* where hello and hello2 are directories and 1.txt is a file which exists.
   Behaviour in our shell: 1.txt will be deleted while one exception will be thrown highlighting that a directory is passed as an argument as rm without any flags cannot remove folders. hello and hello2 which are directories will not be removed. This is inline with the behaviour in unix. Note: Only the latest exception will be thrown.

echo

1) If there are no arguments passed to the echo command, it will output an empty line. This behaviour is similar to in unix where an empty argument passed to echo command also outputs an empty line.
   An example: *echo*
   Behaviour in our shell:
   <prompt>: echo

   <prompt>:
   Note: As shown above, echo with empty arguments outputs an empty line.

2) If *echo "A*B*C"* it will output A*B*C without double quotes similar to in unix shell.
   Behaviour in our shell:
   <prompt>: echo "A*B*C"
   A*B*C
   <prompt>:
   Behaviour in unix shell:
   <prompt>: echo "A*B*C"
   A*B*C
   <prompt>:

cd (Not implemented in Milestone 1 as extended functionality 1)

For cd command, we generally test our shell on Windows platform and since Windows platform and unix platform has many differences in terms of PATH and allowed filenames, we will try to focus on the behaviour of our shell on Windows as much as possible.

1) The cd command PATH argument when it is input forward slash / or backslash \, in our shell the directory will remain unchanged when runned on Windows platform but it will change to the / directory when run in unix if the input is forward slash /. If the input is backslash \ in unix, it will attempt to switch to a directory with filename \ if it exists, else it will throw an error that the directory does not exist.

   In actual unix shell, *cd /* will direct to the / directory of unix on the other hand if *cd* \ is inputted it will request for further input to change to the input directory for example *cd* \ followed by input *hello* where hello is an existing directory in the current directory, it will change to the hello directory.

2) The cd command must take in one argument. (cd without arguments is not supported) as mentioned in the CdApplication.java comments in the skeleton code.
   For example: *cd*

The above example is not allowed in our shell and would throw a CdException of missing arguments message.

3) If our shell is being run on Windows it will not allow invalid path characters such as colon :
   For example: **In our shell,** *cd C:* is invalid when our shell is run on Windows due to the invalid path character colon : as Windows does not allow : as a path character or as a filename or folder names.

   Note: However, *cd C:/* will work if our shell is run on Windows assuming your system drive is C:/ (an existing drive letter) despite the colon as it is referring to the system drive of the current machine.

   On the other hand if our shell is run on unix, as colon is allowed as valid file names or folder names in unix hence, for example *cd :* in unix if there is a directory with the name :, it will change to the colon directory hence, similarly our shell if run on unix allows *cd :*

4) If cd command has more than one argument similar to in ubuntu, for example, *cd hello world*, would throw a CdException with Too many arguments.

Cp (Not implemented in Milestone 1 as extended functionality 1)

1) If the destination file is the same as the source file, for example, *cp 1.txt 1.txt*, where 1.txt exists, will throw a CpException with message The source file and the destination file are the same file. This behaviour is similar to unix which would highlight that both files are the same file.
2) If the destination file already exists, for example, *cp 1.txt 2.txt* where 1.txt and 2.txt already exists, overwrite 2.txt with the contents of 1.txt.
3) If the source file input is a directory, for example, *cp hello hello2*, where hello and hello2 are existing directories, will throw a CpException with message This is a directory. This behaviour differs from unix which supports copying of directory to another directory when used with -r flag. Our shell does not support -r flag based on project application specification hence, copying directory is not supported.
4) If the input source files have at least one source file that does not exist, for example, *cp 1.txt 2.txt dest* where 1.txt is a non existing file while 2.txt is an existing file and dest is an existing directory, it should throw an CpException with No such file or directory but the 2.txt is still copied to hello directory. This is similar to unix behaviour.
   Another example, *cp 1.txt 2.txt dest* where 1.txt is an existing file while 2.txt is a non existing file and dest is an existing directory, it should throw a CpException with no such file or directory but the 1.txt is copied to the hello directory. This is similar to unix behaviour.

5) If the destination directory contains a file with the same file name as the file to copied, overwrite the file in the directory with the file to be copied contents.
   For example: *cp 1.txt dest* where 1.txt is an existing file and dest is an existing directory containing a different 1.txt file.
   Expected Behaviour: Overwrites the 1.txt file in the dest directory with the contents of the 1.txt file to be copied.

Paste

1) In paste, there are two forms of input allowed, namely filenames and stdin.
2) When filename is not given or put dash ("-") after the paste command instead of file name, paste will read from standard input (i.e. <). Example: paste - file.txt < fileOne.txt. For this case, when merged, the file contents from the standard input will appear before that of the filename.
3) Please note that the dash has to be in the first argument. Else, an exception will be thrown.
4) Basic Stdin example: Paste < file.txt. The contents of the file will be in the inputstream, and will then be printed out by the paste command for this example.
5) If there is only one file provided, be it from Stdin or from filename, paste will print the contents of the file.
6) Stdin inputs can be command substitutions. For example, paste `echo file.txt` will print out the contents of file.txt.
7) Paste is expected to work with IO redirection (input '<' and output '>' for this case).
8) Files with different numbers of lines will paste as per normal.
9) Paste will throw an exception when Stdin is empty, invalid/empty/null filename is provided.
10) In terms of file type, theoretically, any text document file type will be acceptable. However, only ".txt" file type has been extensively tested. Therefore, we cannot guarantee that the performance of other types of text document file types will be consistent with the information stated here.

Sed

1) Only single quotation marks commands will be accepted.
2) Other separators characters acceptable, for example: '/' or '|' or '\' or ',' or '.' or ';' or ':'. Note that only one form of separator characters could be used.
3) Any number of files can be provided. The files will be printed in sequence with the replacements made, and separated by newline (note: only 1 newline, therefore there will not be an empty gap).
4) When filename is not given after the sed arguments, sed will read from standard input (i.e. <).

5) Sed will throw an exception when either the regex, replacement index, or replacement string given is invalid or null or empty.
6) Note that a replacement index that is greater than the words in each line will not throw an exception. It will simply print out the file contents without replacement. This follows Linux shell's behaviour.

cut

1) The order of argument in the cut command is fixed.
    a) Options LIST [FILES]
2) Options must be either -b or -c only.
3) Filename must only contains either:
    a) Alpha-numerical characters.
    b) Alpha-numerical characters with special characters where special characters cannot be the starting letter.
4) There can only be 2 real positive whole numbers provided for comma-separated numbers and number range.
5) It is not mandatory to have [FILES]. If FILE is not specified or '-', read from standard input.
    a) Shell can execute echo "baz" | cut -b 2
    b) Note that if you specified multiple '-', Shell will only take the first '-' and ignore subsequent '-' in the command.
6) Here are the list of file extension types that are accepted for cut
    a) .txt
    b) .html
    c) .sh
    d) .bat
    e) …

Mv

1) Assumption is that for a folder destination move, the folder does not contain a file with a similar name.
2) There cannot be more than 1 file in input arguments for source and target or source and directory. An error will be thrown.

Ls

1) Ls follows Linux implementation where folder1 will be shown
    a) ls -R folder1
    folder1:

file1.txt subfolder1

2) For multiple arguments, in linux,
    a) for example ls folder1 folder2 file3.txt
    b) The order will follow the input commands instead of being sorted to which directory comes first.
3) File can not start with dot(.)
4) -d and -R can not be true at same time. So ls -d -R FOLDERS is invalid
5) -d will only display folders and not any file
6) If -d or -R with no folders specified, use current directory

Find

1) Fixed format: <foldernames separated by space> -name <filename>
2) In the FIND application, the argument after '-name' must be quoted with double quotes and only can exist 1 file name.
3) If no files are found, an empty string will be returned.

Sort

1. The order of argument in the sort command is fixed.
    a. [-n][-r][-f] [FILES]
2. Our implementation of sort is different from Unix implementation of sort
    a. Based on unix implementation by default, `sort` sorts character by character using a locale specified sort order. Generally that's pretty close to ASCII order, but there may be some regional variations. From the man page: " *** WARNING *** The locale specified by the environment affects sort order. Set LC_ALL=C to get the traditional sort order that uses native byte values. "
    b. Refer to http://man7.org/linux/man-pages/man1/sort.1.html
3. The order of output will be as followed based on the first character
    a. Special characters followed by numbers followed by capital letters and followed by small letters
4. Unlike Unix shell, our program does not allow sort application FILES to be a dash.
5. Filename must only contains either:
    a. Alpha-numerical characters.
    b. Alpha-numerical characters with special characters where special characters cannot be the starting letter.

Wc (Not implemented in Milestone 1 as extended functionality 1)

1. The order of argument in the wc command is fixed.
    a. [-c][-l][-w] [FILES]
2. Unlike Unix shell, our program does not allow wc application FILES to be a dash.
3. Filename must only contains either:
    a. Alpha-numerical characters.
    b. Alpha-numerical characters with special characters where special characters cannot be the starting letter.