|  |
| --- |
|  |
| CS4218 Milestone 1 |
|  |

|  |
| --- |
|  |

**Team 05**

KOH CHENG GEE A0126199W

MUHAMMAD RAZALI A0133267H

PHOON QIANONN A0126232U

TANG WEI REN A0125531R

Contents

[1. Implementation for Basic Functionality (BF) 3](#_Toc475529489)

[1.1 Implementation for Shell: Calling applications 3](#_Toc475529490)

[1.2 Implementation for Shell: Quoting 3](#_Toc475529491)

[1.3 Implementation for Shell: Semicolon Operator 3](#_Toc475529492)

[1.4 Implementation for Applications: cat 3](#_Toc475529493)

[1.5 Implementation for Applications: cd 3](#_Toc475529494)

[1.6 Implementation for Applications: pwd 4](#_Toc475529495)

[1.7 Implementation for Applications: echo 5](#_Toc475529496)

[1.8 Implementation for Applications: head 5](#_Toc475529497)

[1.9 Implementation for Applications: tail 6](#_Toc475529498)

[2. Implementation for Extended Functionality 1 (EF1) 8](#_Toc475529499)

[2.1 Implementation for Shell: Globbing 8](#_Toc475529500)

[2.2 Implementation for Shell: IO-redirection 8](#_Toc475529501)

[2.3 Implementation for Applications: cal 8](#_Toc475529502)

[2.4 Implementation for Applications: grep 8](#_Toc475529503)

[2.5 Implementation for Applications: sort 8](#_Toc475529504)

[3. Unit Testing for Basic Functionality (BF) 11](#_Toc475529505)

[3.1 Unit Testing for Shell: Calling applications 11](#_Toc475529506)

[3.2 Unit Testing for Shell: Quoting 11](#_Toc475529507)

[3.3 Unit Testing for Shell: Semicolon Operator 12](#_Toc475529508)

[3.4 Unit Testing for Applications: cat 12](#_Toc475529509)

[3.5 Unit Testing for Applications: cd 13](#_Toc475529510)

[3.6 Unit Testing for Applications: pwd 13](#_Toc475529511)

[3.7 Unit Testing for Applications: echo 13](#_Toc475529512)

[3.8 Unit Testing for Applications: head 14](#_Toc475529513)

[3.9 Unit Testing for Applications: tail 14](#_Toc475529514)

[4. Unit Testing of Extended Functionality 1 (EF1) 15](#_Toc475529515)

[4.1 Unit Testing for Shell: Globbing 15](#_Toc475529516)

[4.2 Unit Testing for Shell: IO-redirection 15](#_Toc475529517)

[4.3 Unit Testing for Applications: cal 15](#_Toc475529518)

[4.4 Unit Testing for Applications: grep 16](#_Toc475529519)

[4.5 Unit Testing for Applications: sort 16](#_Toc475529520)

[5. Basic Functions Test Cases 17](#_Toc475529521)

[5.1 Test Cases: cat 17](#_Toc475529522)

[5.2 Test Cases: cd 18](#_Toc475529523)

[5.3 Test Cases: pwd 19](#_Toc475529524)

[5.4 Test Cases: head 20](#_Toc475529525)

[5.5 Test Cases: tail 21](#_Toc475529526)

[6. Extended Functions (EF1) Test Cases 23](#_Toc475529527)

[6.1 Test Cases: cal 23](#_Toc475529528)

[6.2 Test Cases: grep 24](#_Toc475529529)

[6.3 Test Cases: sort 24](#_Toc475529530)

[6.4 Test Cases: glob 27](#_Toc475529531)

[7. Extended Functions (EF2) Test Cases 28](#_Toc475529532)

[7.1 Test Cases: date 28](#_Toc475529533)

[7.2 Test Cases: sed 28](#_Toc475529534)

[7.3 Test Cases: wc 29](#_Toc475529535)

[7.4 Test Cases: piping operator 30](#_Toc475529536)

# Implementation for Basic Functionality (BF)

## Implementation for Shell: Calling applications

* Shell splits call command into arguments and redirection operators.
* Filenames are expanded.
* Application name is resolved.
* Specified application is executed.

## Implementation for Shell: Quoting

* Our shell supports three kinds of quotes:
  + Single quote (‘)
  + Double quotes (“)
  + Backquotes (`)
* The ‘ and “ are used to disable interpretation of all or some special characters
* The ` used to make command substitution
* Special characters are: \t (tab), \* (globbing), ‘ (single quote), “ (double quote), ` (backquote), | (pipe), < (input redirection), > (output redirection), ; (semicolon), space
* Using regex and pattern matching to search the quote
* If quote found then run the corresponding method

## Implementation for Shell: Semicolon Operator

* The semicolon operator split a command into 2 command.
* Run the first command; when the first command terminates, run the second command.
* If an exception is thrown during the execution of the first command, the execution of the whole command must be terminated.
* Using regex and split to split the command

## Implementation for Applications: cat

|  |  |
| --- | --- |
| Command | Description |
| cat | Outputs error message |
| Cat [filepath]+ | Checks validity of file path. If file could not be found or read, an empty string is sent as an output.  Outputs the contents of the file. |

## Implementation for Applications: cd

The cd command changes the current working directory

|  |  |
| --- | --- |
| Command | Description |
| cd | change to home directory |
| cd ~ | Change to home directory |
| cd / | Change to the root directory  For Mac or Linux   * Linux uses “/” for its “file separator” * However, for Linux, if a user entered “\” as the “file separator” instead of “/” there will be no auto correction as “\” is a valid folder & file name in the Linux. An appropriate error message will be display if the “path” entered does not exist in the shell system. |
| cd / | Change to the root directory  For Windows   * Windows uses “\” for its “file separator”. * If “/” is mistakenly used as the “file separator” in the “path” entered, the shell system will auto correct “/” to “\”. * The following observation was made in the “command prompt” on Windows, hence this correction was implemented to help ease the user as their intention is clear. * As “/” is an invalid folder & file name in Windows, the auto-correction will not cause any issues |
| cd \ | For Windows:   * Windows uses “\” for its “file separator”. |
| cd . | No change in directory. Remain at current working directory |
| cd .. | Change to parent directory |
| cd [PATH] | Change the directory to the “PATH”. If the “PATH” does not exist it will return an error message   * Windows uses “\” for its “file separator”. * If the shell system is run on Windows, auto correction of the “file separator” from “/” to “\” will take if “/” is mistakenly used as the “file separator” in the “PATH” entered. * The following observation was made in the “command prompt” on Windows, hence this correction was implemented to help ease the user as their intention is clear. * As “/” is an invalid folder & file name in Windows, the auto-correction will not cause any issues * “PATH” is a “relative path directory” e.g. “folder1/folder2” * “PATH” as specific in the project document cannot be an “absolute path directory” e.g. /user/home/directory”. A “/” (for mac & Linux) and “\” (for Windows) indicates that the “PATH” is an “absolute path directory” hence the “shell system” will return an error message if “/” or “\” is detected as the first char of the “PATH” |

## Implementation for Applications: pwd

The pwd command prints the current working directory followed by a newline

|  |  |
| --- | --- |
| Command | Description |
| pwd | Display the current working directory followed by a newline   * Use “*Environment.currentDirectory*” to get the current working directory * Add “*System.lineSeparator*” to insert a newline after displaying the results |

## Implementation for Applications: echo

|  |  |
| --- | --- |
| Command | Description |
| echo | Prints new line |
| Echo [args]\* | Prints arguments |

## Implementation for Applications: head

**Command: head [OPTION] [PATH]**

|  |
| --- |
| Description |
| The “head” commands prints first N lines of a specific file or input |
| “head” with no PATH and OPTION will print the first 10 lines of the InputStream |
| OPTION is in the format: “-n 3”, where “-n” is the command and “3” is the number of lines |
| PATH is a “relative path directory” e.g. “folder1/folder2/123.txt” |
| “head” command accepts the rightmost OPTION if more than 1 OPTION exists. e.g. “head –n 3 –n 5 –n 2 123.txt”. It accepts –n 2 as the OPTION |
| “head” command accepts multiple PATH in the command. e.g. “head –n 6 123.txt 246.txt”. It prints out the first 6 lines of 123.txt and 246.txt files |
| OPTION can input be behind the PATH. e.g. “head 123.txt –n 6” |
| Other than the OPTION, any command that starts with “-” is invalid |
| Any input other than the OPTION will be regarded as PATH e.g. “head 1 5.txt 7”. The PATH of the inputs are: 1, 5.txt and 7 |

|  |
| --- |
| Methods & Description |
| public void run(String args[], InputStream stdin, OutputStream stdout)   * args[] takes in an array of String which consists of OPTION and PATH * stdin takes in inputStream * stdout takes in outputStream * this method process the arguments and print out the details in console |
| private void checkArgumentLengthAndRun(String[] args, InputStream stdin, OutputStream stdout)   * args[] takes in an array of String which consists of OPTION and PATH * stdin takes in inputStream * stdout takes in outputStream * this method check the details within args and run methods according to the args length |
| private void readEveryFilePathWithLineNumbers(OutputStream stdout, int lineNumber, ArrayList<String> listOfArgs)   * stdout takes in outputStream * lineNumber is the number of lines to be display on console * listOfArgs is the list that stores the PATH of files * this method reads the first lineNumber of lines in each file of listOfArgs and display in console through stdout |
| private void readEveryFilePath(String[] args, OutputStream stdout, ArrayList<String> listOfArgs)   * args[] is an empty array * stdout takes in outputStream * listOfArgs is the list that stores the PATH of files * this method lets args get each individual PATH for listOfArgs, reads the PATH from args and display on the console through stdout |
| private int checkDuplicateLineNumbers(String[] args1, ArrayList<String> listOfArgs)   * args[] takes in an array of String which consists of OPTION and PATH * listOfArgs is an empty ArrayList * this method checks the validity in args1, adding the PATHs to listOfArgs and return the rightmost OPTION |
| private void readWithLinesAndDirectory(String[] args, OutputStream stdout)   * args[] takes in an array of String which consists of OPTION and PATH * stdout takes in outputStream * this method reads the file of PATH and display first N lines given in args and display through stdout |
| private void readWithLines(String[] args, InputStream stdin, OutputStream stdout)   * args[] takes in an array of String which consists of OPTION * stdin takes in inputStream * stdout takes in outputStream * this method reads the file from stdin and display first N lines given in args and display through stdout |
| private void readFromPath(String[] args, OutputStream stdout)   * args[] takes in an array of String which consists of PATH * stdout takes in outputStream * this method reads the file from PATH and display through stdout |
| private void readFromStdin(InputStream stdin, OutputStream stdout)   * stdin takes in inputStream * stdout takes in outputStream * this method reads the file from stdin and display the first 10 lines through stdout |
| private boolean checkNullInput(String[] args, InputStream stdin, OutputStream stdout)   * args[] takes in an array of String which consists of OPTION and PATH * stdin takes in inputStream * stdout takes in outputStream * this method checks whether all args, stdin and stdout and return true if all 3 are null |

## Implementation for Applications: tail

**Command: tail [OPTION] [PATH]**

|  |
| --- |
| Description |
| The “tail” commands prints last N lines of a specific file or input |
| “tail” with no PATH and OPTION will print the first 10 lines of the InputStream |
| OPTION is in the format: “-n 3”, where “-n” is the command and “3” is the number of lines |
| PATH is a “relative path directory” e.g. “folder1/folder2/123.txt” |
| “tail” command accepts the rightmost OPTION if more than 1 OPTION exists. e.g. “tail –n 3 –n 5 –n 2 123.txt”. It accepts –n 2 as the OPTION |
| “tail” command accepts multiple PATH in the command. e.g. “tail –n 6 123.txt 246.txt”. It prints out the first 6 lines of 123.txt and 246.txt files |
| OPTION can be input behind the PATH. e.g. “tail 123.txt –n 6” |
| Other than the OPTION, any command that starts with “-” is invalid |
| Any input other than the OPTIONT will be regarded as PATH e.g. “tail 1 5.txt 7”. The PATH of the inputs are: 1, 5.txt and 7 |

|  |
| --- |
| Methods & Description |
| public void run(String args[], InputStream stdin, OutputStream stdout)   * args[] takes in an array of String which consists of OPTION and PATH * stdin takes in inputStream * stdout takes in outputStream * this method process the arguments and print out the details in console |
| private void checkArgumentLengthAndRun(String[] args, InputStream stdin, OutputStream stdout)   * args[] takes in an array of String which consists of OPTION and PATH * stdin takes in inputStream * stdout takes in outputStream * this method check the details within args and run methods according to the args length |
| private int checkDuplicateLineNumbers(String[] args1, ArrayList<String> listOfArgs)   * args[] takes in an array of String which consists of OPTION and PATH * listOfArgs is an empty ArrayList * this method checks the validity in args1, adding the PATHs to listOfArgs and return the rightmost OPTION |
| private void readWithLinesAndDirectory(String[] args, OutputStream stdout)   * args[] takes in an array of String which consists of OPTION and PATH * stdout takes in outputStream * this method reads the file of PATH and display first N lines given in args and display through stdout |
| private void readFromPath (String[] args, InputStream stdin, OutputStream stdout)   * args[] takes in an array of String which consists of OPTION * stdin takes in inputStream * stdout takes in outputStream * this method reads the file from stdin and display first N lines given in args and display through stdout |
| private void readWithDirectory (String[] args, OutputStream stdout)   * args[] takes in an array of String which consists of PATH * stdout takes in outputStream * this method reads the file from PATH and display through stdout |
| private void readFromStdin(InputStream stdin, OutputStream stdout)   * stdin takes in inputStream * stdout takes in outputStream * this method reads the file from stdin and display the first 10 lines through stdout |
| private boolean checkNullInput(String[] args, InputStream stdin, OutputStream stdout)   * args[] takes in an array of String which consists of OPTION and PATH * stdin takes in inputStream * stdout takes in outputStream * this method checks whether all args, stdin and stdout and return true if all 3 are null |

# Implementation for Extended Functionality 1 (EF1)

## Implementation for Shell: Globbing

* For each argument ARG in a shell command check if it contains unquoted \* (asterisk)
* Find the deepest path which does not contain \*
* Create path regex to search correct file
* Find all the file from that path by using pattern matching
* Store all the correct file into arraylist and pass it to next function as argument

## Implementation for Shell: IO-redirection

* Open InputStream from the file for input redirection (the one following “<”symbol).
* Open the OutputStream to the file for output redirection (the one following “>” symbol).
* If several files are specified for input redirection or output redirection, throw an exception.
* If no files are given, use the NULL value.
* If the file specified for input redirection does not exist, throw an exception.
* If the file specified for output redirection does not exist, create it.

## Implementation for Applications: cal

|  |  |
| --- | --- |
| Command | Description |
| cal | Prints current month, Sunday first |
| cal [year] | Prints all 12 months of year specified, Sunday first.  Prints error if year is invalid format |
| cal [month] [year] | Prints the calendar of the month and year specified. If any of it is invalid, error is sent. |

## Implementation for Applications: grep

|  |  |
| --- | --- |
| Command | Description |
| Grep | Prints new line |
| Grep [pattern] | Prints new line |
| Grep [pattern] [file]+ | Searches for matching strings in file and prints it. |

## Implementation for Applications: sort

* The “sort” commands sort a specific file or input in a sorted order
* The sorted order is defined in the order of “special character”, “numbers”, “capital letters” and “simple letters”
* If “-n” is specific after the “sort” command (e.g. sort [-n] [FILE]), the first word of a line will be treated as a number. The first word will be treated as number only if it contains all numeric characters (e.g 8 days later, so “8” is the first word). If the first word will be treated as a normal word if it contains other types of characters(e.g. 8days later, “8days” is the first word)
* If 2 or more “FILES” are provided, the lines in the “FILES” will be combined and sorted together

|  |
| --- |
| Methods & Description |
| public void run(String args[], InputStream stdin, OutputStream stdout)   * args[] takes in an array of String which contains either “-n” or “FILE” * stdin takes in inputStream * stdout takes in outputStream |
| public String sortStringSimple(String toSort)   * toSort takes in String. * Data from InputStream will have to be converted to “String” type first The conversion is done inside public void run(String args[], InputStream stdin, OutputStream stdout). * Sort and return a string of lines starting with simple letter |
| public String sortStringCapital(String toSort)   * toSort takes in String. * Data from InputStream will have to be converted to “String” type first The conversion is done inside public void run(String args[], InputStream stdin, OutputStream stdout). * Sort and return a string of lines starting with capital letter |
| public String sortNumbers(String toSort)   * toSort takes in String. * Data from InputStream will have to be converted to “String” type first The conversion is done inside public void run(String args[], InputStream stdin, OutputStream stdout). * Sort and return a string of lines starting with numbers |
| public String sortSpecialChars(String toSort)   * toSort takes in String. * Data from InputStream will have to be converted to “String” type first The conversion is done inside public void run(String args[], InputStream stdin, OutputStream stdout). * Sort and return a string of lines starting with special chars |
| public String sortSimpleCapital(String toSort)   * toSort takes in String. * Data from InputStream will have to be converted to “String” type first The conversion is done inside public void run(String args[], InputStream stdin, OutputStream stdout). * Sort and return a string of lines starting with capital letter and simple letter. In the order of capital letter followed by simple letter |
| public String sortSimpleNumbers(String toSort)   * toSort takes in String. * Data from InputStream will have to be converted to “String” type first The conversion is done inside public void run(String args[], InputStream stdin, OutputStream stdout). * Sort and return a string of lines starting with numbers and simple letter. In the order of numbers followed by simple letter |
| public String sortSimpleSpecialChars (String toSort)   * toSort takes in String. * Data from InputStream will have to be converted to “String” type first The conversion is done inside public void run(String args[], InputStream stdin, OutputStream stdout). * Sort and return a string of lines starting with capital letter and simple letter. In the order of capital letter followed by simple letter |
| public String sortCapitalNumbers(String toSort)   * toSort takes in String. * Data from InputStream will have to be converted to “String” type first The conversion is done inside public void run(String args[], InputStream stdin, OutputStream stdout). * Sort and return a string of lines starting with numbers and capital letter. In the order of numbers followed by captial letter |
| public String sortCaptialSpecialChars(String toSort)   * toSort takes in String. * Data from InputStream will have to be converted to “String” type first The conversion is done inside public void run(String args[], InputStream stdin, OutputStream stdout). * Sort and return a string of lines starting with special chars and capital letter. In the order of special chars followed by capital letter. |
| public String sortNumbersSpecialChars(String toSort)   * toSort takes in String. * Data from InputStream will have to be converted to “String” type first The conversion is done inside public void run(String args[], InputStream stdin, OutputStream stdout). * Sort and return a string of lines starting with special chars and numbers. In the order of special chars and numbers. |
| public String sortSimpleCapitalNumber(String toSort)   * toSort takes in String. * Data from InputStream will have to be converted to “String” type first The conversion is done inside public void run(String args[], InputStream stdin, OutputStream stdout). * Sort and return a string of lines starting with numbers, capital letter and simple letter. In the order of numbers, capital letter followed by simple letters. |
| public String sortSimpleCapitalSpecialChars(String toSort)   * toSort takes in String. * Data from InputStream will have to be converted to “String” type first The conversion is done inside public void run(String args[], InputStream stdin, OutputStream stdout). * Sort and return a string of lines starting with special chars, capital letter and simple letter. In the order of special chars, capital letter followed by simple letters. |
| public String sortSimpleNumbersSpecialChars (String toSort)   * toSort takes in String. * Data from InputStream will have to be converted to “String” type first The conversion is done inside public void run(String args[], InputStream stdin, OutputStream stdout). * Sort and return a string of lines starting with special chars, numbers and simple letter. In the order of special chars, numbers followed by simple letters. |
| public String sortCaptialNumbersSpecialChars(String toSort)   * toSort takes in String. * Data from InputStream will have to be converted to “String” type first The conversion is done inside public void run(String args[], InputStream stdin, OutputStream stdout). * Sort and return a string of lines starting with special chars, numbers and capital letter. In the order of special chars, numbers followed by capital letters. |
| public String sortAll (String toSort)   * toSort takes in String. * Data from InputStream will have to be converted to “String” type first The conversion is done inside public void run(String args[], InputStream stdin, OutputStream stdout). * Sort and return a string of lines starting with special chars, numbers, capital letter, simple letters. In the order of special chars, numbers, capital letters followed by simple letters. |

# Unit Testing for Basic Functionality (BF)

## Unit Testing for Shell: Calling applications

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Test Case(s) | Input | Expected Results |
| 1 | Cat App Called | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Null | Argument of application, Application name |
| 2 | Cd App Called | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Null | Argument of application, Application name |
| 3 | Pwd App Called | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Null | Argument of application, Application name |
| 4 | Echo App Called | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Null | Argument of application, Application name |
| 5 | Head App Called | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Null | Argument of application, Application name |
| 6 | Tail App Called | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Null | Argument of application, Application name |

## Unit Testing for Shell: Quoting

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Test Case(s) | Input | Expected Results |
| 1 | Echo No Quotes | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Valid | Contents of input with a New Line:‘\n’ at the end |
| 2 | Echo Single Quotes | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Valid | Contents of input with a New Line:‘\n’ at the end |
| 3 | Echo Double Quotes | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Valid | Contents of input with a New Line:‘\n’ at the end |
| 4 | Echo Back Quotes | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Valid | ShellException |
| 5 | Echo Double With Back Quotes | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Valid | Contents of input with a New Line:‘\n’ at the end |
| 6 | Echo Single With Back Quotes | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Valid | Contents of input with a New Line:‘\n’ at the end |

## Unit Testing for Shell: Semicolon Operator

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Test Case(s) | Input | Expected Results |
| 1 | Echo Semicolon | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Valid | Contents of input with a New Line:‘\n’ at the end |
| 2 | Echo Semicolon With Exception | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Valid | ShellException |
| 3 | Echo Semicolon With Quotes | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Valid | Contents of input with a New Line:‘\n’ at the end |

## Unit Testing for Applications: cat

Unit Test for interface **public void run(String args[], InputStream stdin, OutputStream stdout)**

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Test Case(s) | Input | Expected Results |
| 1 | Cat without Output Stream | **Arguments**: Valid  **InputStream**: Valid  **OutputStream**: Null | **CatException**, **Message**: cat: OutputStream not provided |
| 2 | Cat with one valid file | **Arguments**: Valid  **InputStream**: Valid  **OutputStream**: Valid | Contents of file with a **New Line:** ‘\n’ at the end |
| 3 | Cat with multiple valid files | **Arguments**: Valid  **InputStream**: Valid  **OutputStream**: Valid | Contents of all fileswith a **New Line:** ‘\n’ at the end |
| 4 | Cat with non-existent file | **Arguments**: Invalid  **InputStream**: Valid  **OutputStream**: Valid | **New Line:** ‘\n’ |
| 5 | Cat with no Arguments and No input stream | **Arguments**: Null  **InputStream**: Null  **OutputStream**: Valid | **CatException**, **Message**: cat: InputStream not provided |
| 6 | Cat with no Arguments, Valid Input Stream Exists | **Arguments**: Null  **InputStream**: Valid  **OutputStream**: Valid | Contents of InputStream |

## Unit Testing for Applications: cd

Unit Test for interface **public void run(String args[], InputStream stdin, OutputStream stdout)**

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Test Methods | Input | Expected Results |
| 1 | public void testEmptyArgs() | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Null | Return to user home directory |
| 2 | public void testRootDirectory() | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Null | Return to root directory |
| 3 | public void testHomeDirectory() | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Null | Return to user home directory |
| 4 | public void testCurrentDirectory() | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Null | Remain in the same directory |
| 5 | public void testParentDirectory() | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Null | Return to parent directory |
| 6 | public void testCdToInvalidDirectory() | **Arguments**: Invalid  **InputStream**: Null  **OutputStream**: Null | **CdException, Message:** Cd: The directory does not exist |
| 7 | public void testCdToValidDirectory() | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Null |  |

## Unit Testing for Applications: pwd

Unit Test for interface **public void run(String args[], InputStream stdin, OutputStream stdout)**

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Test Methods | Input | Expected Results |
| 1 | public void testPwdWithNoArgs() | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Valid | The current working directory is displayed |
| 2 | public void testPwdWithArgs() | **Arguments**: Invalid  **InputStream**: Null  **OutputStream**: Valid | **PwdException, Message:** Pwd: Invalid Arguments |

## Unit Testing for Applications: echo

Unit Test for interface **public void run(String args[], InputStream stdin, OutputStream stdout)**

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Test Case(s) | Input | Expected Results |
| 1 | Echo without Output Stream | **Arguments:** Valid  **InputStream**: Valid  **OutputStream**: Null | **EchoException:** OutputStream not provided |
| 2 | Echo without arguments | **Arguments:** Null  **InputStream**: Valid  **OutputStream**: Valid | **EchoException:** Null Arguments |
| 3 | Echo empty string | **Arguments:** Valid  **InputStream**: Valid  **OutputStream**: Valid | **Empty String** |
| 4 | Echo with arguments | **Arguments:** Valid  **InputStream**: Valid  **OutputStream**: Valid | **Arguments** |

## Unit Testing for Applications: head

Unit Test for interface **public void run(String args[], InputStream stdin, OutputStream stdout)**

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Test Methos | Input | Expected Results |
| 1 | testAllNullArgument | **Arguments**: N  **InputStream**: N  **OutputStream**: N | **HeadException**, **Message**: Head: args, stdin, stdout are null |
| 2 | testNullInputStream | **Arguments**: V  **InputStream**: N  **OutputStream**: V | **HeadException**,  **Message**:Head: stdin is null |
| 3 | testNullOutputStream | **Arguments**: V  **InputStream**: V  **OutputStream**: N | **HeadException**,  **Message**:Head: stdout is null |
| 4 | testNoArgument | **Arguments**: N  **InputStream**: V  **OutputStream**: V | **InputStream Message:** “test\nstring” |
| 5 | testOneArgument | **Arguments**: V  **InputStream**: N  **OutputStream**: V | **InputStream Message: “**31423\n 115ewafg\n gaqwtq345 \ntqtqt \nc592859v \ngasgsad" |
| 6 | testTwoArgument | **Arguments**: V  **InputStream**: V  **OutputStream**: V | **InputStream Message:**  "31423\n1" |
| 7 | testThreeArgument | **Arguments**: V  **InputStream**: V  **OutputStream**: V | **InputStream Message:**  "31423\n1" |

## Unit Testing for Applications: tail

Unit Test for interface **public void run(String args[], InputStream stdin, OutputStream stdout)**

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Test Case(s) | Input | Expected Results |
| 1 | testAllNullArgument | **Arguments**: N  **InputStream**: N  **OutputStream**: N | **TailException**, **Message**: Tail: args, stdin, stdout are null |
| 2 | testNullInputStream | **Arguments**: V  **InputStream**: N  **OutputStream**: V | **TailException**,  **Message**:Tail: stdin is null |
| 3 | testNullOutputStream | **Arguments**: V  **InputStream**: V  **OutputStream**: N | **TailException**,  **Message**:Tail: stdout is null |
| 4 | testNoArgument | **Arguments**: N  **InputStream**: V  **OutputStream**: V | **InputStream Message:** “test\nstring” |
| 5 | testOneArgument | **Arguments**: V  **InputStream**: N  **OutputStream**: V | **InputStream Message: “**31423\n 115ewafg\n gaqwtq345 \ntqtqt \nc592859v \ngasgsad" |
| 6 | testTwoArgument | **Arguments**: V  **InputStream**: V  **OutputStream**: V | **InputStream Message:**  "31423\n1" |
| 7 | testThreeArgument | **Arguments**: V  **InputStream**: V  **OutputStream**: V | **InputStream Message:**  "31423\n1" |

# Unit Testing of Extended Functionality 1 (EF1)

## Unit Testing for Shell: Globbing

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Test Case(s) | Input | Expected Results |
| 1 | Cat Glob No Path | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Valid | Empty |
| 2 | Cat Glob One File | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Valid | Contents of file with a New Line:‘\n’ at the end |
| 3 | Cat Glob Directories | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Valid | Contents of file with a New Line:‘\n’ at the end |
| 4 | Cat Glob Files Directories | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Valid | Contents of file with a New Line:‘\n’ at the end |
| 5 | Cat Glob Multiple Files Directories | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Valid | Contents of file with a New Line:‘\n’ at the end |

## Unit Testing for Shell: IO-redirection

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Test Methods | Input | Expected Results |
| 1 | Cat with < [file] | **InputRedirection:** Valid  **OuputRedirection:**  Null | **Prints current file contents** |
| 2 | Echo [args] with > f[ile] | **InputRedirection:** Null  **OuputRedirection:**  Valid | **Creates file with [args] as content** |
| 3 | Cal < [file1] > [file2] | **InputRedirection:** Valid  **OuputRedirection:**  Valid | **Creates file2 with contents from [file1]** |

## Unit Testing for Applications: cal

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Test Methods | Input | Expected Results |
| 1 | Cal without any arguments | **Year:** Null  **Month:** Null  **Monday:** Null  **InputStream**: Valid  **OutputStream**: Valid | **Prints current month** |
| 2 | Cal with valid year | **Year:** Valid  **Month:** Null  **Monday:** Null  **InputStream**: Valid  **OutputStream**: Valid | **Prints all month for that year** |
| 3 | Cal with invalid year | **Year:** Invalid  **Month:** Null  **Monday:** Null  **InputStream**: Valid  **OutputStream**: Valid | **CalException thrown** |
| 4 | Cal with Monday option | **Year:** Null  **Month:** Null  **Monday:** Valid  **InputStream**: Valid  **OutputStream**: Valid | **Prints current month starting from Monday** |
| 5 | Cal with valid year and Monday option | **Year:** Valid  **Month:** Null  **Monday:** Valid  **InputStream**: Valid  **OutputStream**: Valid | **Prints all months of that year starting from Monday** |
| 6 | Cal with valid year, month and Monday option | **Year:** Valid  **Month:** Valid  **Monday:** Valid  **InputStream**: Valid  **OutputStream**: Valid | **Prints the particular month of the year starting from Monday** |

## Unit Testing for Applications: grep

Unit Test for interface **public void run(String args[], InputStream stdin, OutputStream stdout)**

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Test Methods | Input | Expected Results |
| 1 | Grep without Output Stream | **Pattern**: Valid  **Files:** Valid  **InputStream**: Valid  **OutputStream**: Null | **GrepException**, **Message**: grep: OutputStream not provided |
| 2 | Grep without pattern and files | **Pattern**: Null  **Files:** Null  **InputStream**: Valid  **OutputStream**: Valid | **New Line:** ‘\n’ |
| 3 | Grep with pattern, without files, without input stream | **Pattern**: Valid  **Files:** Null  **InputStream**: Null  **OutputStream**: Valid | **GrepException**, **Message**: grep: InputStream not provided |
| 4 | Grep with pattern, without files, with valid input stream | **Pattern**: Valid  **Files:** Null  **InputStream**: Valid  **OutputStream**: Valid | **Line(s) from the input stream containing the matching pattern** |
| 5 | Grep with one valid file | **Pattern**: Valid  **Files:** Valid  **InputStream**: Valid  **OutputStream**: Valid | **Line(s) from the file containing the matching pattern** |
| 6 | Grep multiple valid files | **Pattern**: Valid  **Files:** Valid  **InputStream**: Valid  **OutputStream**: Valid | **Line(s) from the files containing the matching pattern** |
| 7 | Grep with some valid files | **Pattern**: Valid  **Files:** Valid & Invalid  **InputStream**: Valid  **OutputStream**: Valid | **Line(s) from the valid files containing the matching pattern** |
| 8 | Grep with non-existent file | **Pattern**: Valid  **Files:** Invalid  **InputStream**: Valid  **OutputStream**: Valid | **New Line:** ‘\n’ |

## Unit Testing for Applications: sort

Unit Test for interface **public void run(String args[], InputStream stdin, OutputStream stdout)**

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Test Methods | Input | Expected Results |
| 1 | public void testEmptyInputStreamArgs() | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Null | **SortException, Message:** Input stream empty |
| 2 | public void testInvalidFileName() | **Arguments**: Invalid  **InputStream**: Null  **OutputStream**: Null | **SortException, Message:** error reading file |
| 3 | public void testInvalidArgs() | **Arguments**: Invalid  **InputStream**: Null  **OutputStream**: Null | **SortException, Message:** Sort: error reading file |

Unit Test for Sort Interfaces

The following methods test the interfaces as defined in Sort.java. The test checks if the interfaces returns the correct strings

|  |  |  |
| --- | --- | --- |
| ID | Test Methods | Expected Results |
| 1 | public void testSortStringsSimple() | True |
| 2 | public void testSortStringsCaptial () | True |
| 3 | public void testSortNumbers () | True |
| 4 | public void testSortNumbersWithN | True |
| 5 | public void testSortSpecialChars | True |
| 6 | public void testSortSimpleCaptial | True |
| 7 | public void testSortSimpleNumbers | True |
| 8 | public void testSortSimpleNumbersWithN | True |
| 9 | public void testSortSimpleSpecialChars | True |
| 10 | public void testSortCapitalNumbers | True |
| 11 | public void testSortCapitalNumbersWithN | True |
| 12 | public void testSortCaptialSpecialChars | True |
| 13 | public void testSortNumbersSpecialChars | True |
| 14 | public void testSortNumbersSpecialCharsWithN | True |
| 15 | public void testSortSimpleCapitalNumbers | True |
| 16 | public void testSortSimpleCapitalNumbersWithN | True |
| 17 | public void testSortSimpleCapitalSpecialChars | True |
| 18 | public void testSortSimpleNumbersSpecialChars | True |
| 19 | public void testSortSimpleNumbersSpecialCharsWithN | True |
| 20 | public void testSortCaptialNumbersSpecialChars | True |
| 21 | public void testSortCaptialNumbersSpecialCharsWithN | True |
| 22 | public void testSortAll | True |
| 23 | public void testSortAllWithN | True |

# Basic Functions Test Cases

## 5.1 Test Cases: cat

Files:

* cat1.txt, cat2.txt
* input1.txt, input2.txt, input3.txt, input4.txt input5.txt, input6.txt, input7.txt input8.txt, input9.txt
* output1.txt, output2.txt, output3.txt output4.txt, output5.txt, output7.txt, output9.txt

Files Description:

* input files contains the input command
* output files contains the expected output
* There is no output6.txt and output8.txt

|  |  |  |
| --- | --- | --- |
| Input | Expected Output | Explanation |
| cat cat1.txt | this is a file "cat1.txt" | The name “cat1.txt” does not contain a space. Output the content of the file |
| cat “cat2.txt” | This is a file “cat2.txt” | The name “cat2.txt” contain a space. Double quote is require to output the content of the file |
| cat cat1.txt “cat2.txt” | this is a file "cat1.txt"  This is a file “cat2.txt” | Output the content of 2 files “cat1.txt” & “cat2.txt” |
| cat cat2.txt | cat: Cant read file | A space exist in the [FILE] so error in reading file |
| cat < cat1.txt | this is a file "cat1.txt" | Using input stream to output the file |
| cat < cat1.txt > output.txt |  | Redirect and save the content of “cat1.txt” into output.txt |
| cat | cat: InputStream not provided | There is no [FILE] provided |
| cat invalidfile.txt |  | “invalidfile.txt” does not exist so no output |
| cat cat1.txt invalidfile.txt | this is a file "cat1.txt" | Output the content of “cat1.txt”. “invalidfile.txt” does not exist so no output |

## 5.2 Test Cases: cd

* Current working directory: C:\User\Tom\Desktop
* Root Directory: C:\
* Home Directory: C:\User\Tom
* Folders on desktop “test1”

All the “cd” commands are executed on the current working directory “C:\User\Tom\Desktop”

The is no “input.txt & output.txt” test cases in the “BF test case” folder as the output will be different on different computer

|  |  |  |
| --- | --- | --- |
| Input | Output | Explanation |
| cd | C:\User\Tom | Return to user home directory |
| cd ~ | C:\User\Tom | Return to user home directory |
| cd \ | C:\ | Return to root directory |
| cd . | C:\User\Tom\Desktop | Remain in current working directory |
| cd .. | C:\User\Tom | Return to parent directory |
| cd test1 | C:\User\Tom\Desktop\test1 | Change directory to “test1” |
| cd test2 | Error message no such directory | No such directory exist |
| cd \User\Tom\Desktop\test1 | Error message absolute path not supported | Does not support absolute path |
| cd ..\.\.\.\\\\\Desktop\.\.\test1 | C:\User\Tom\Desktop\test1 | Using relative path and a combination of cd command to change directory to “test1” |

## 5.3 Test Cases: pwd

All the “pwd” command are executed on current working directory: “C:\User\Tom\Desktop”.

The is no “input.txt & output.txt” test cases in the “BF test case” folder as the output will be different on different computer

|  |  |  |
| --- | --- | --- |
| Input | Output | Explanation |
| pwd | C:\User\Tom\Desktop | Print current working directory |
| pwd test.txt | Throws an error message | Does not accept an argument so throws an error |

**Test Cases: echo**

Files:

* input1.txt, input2.txt, input3.txt, input4.txt input5.txt, input6.txt, input7.txt
* output1.txt, output2.txt, output3.txt output4.txt, output5.txt, output6.txt, output7.txt,

Files Description:

* input files contains the input command
* output files contains the expected output

|  |  |  |
| --- | --- | --- |
| Input | Expected Output | Explanation |
| echo hello | hello | Output “hello” |
| echo hello world | hello world | Output “hello world” |
| echo “hello world” | hello world | Output “hello world” using double quotes |
| echo “hello “ world” | shell: Invalid syntax encountered. | Extra double quotes |
| echo “hello “abc” world” | shell: Invalid syntax encountered. | Extra double quotes |
| echo “hello | shell: Invalid syntax encountered. | Missing double quotes |
| echo “hello `echo abc` world” | hello abc world | “echo” received one argument |
| echo ‘hello’ | Hello | Output “hello” using single quote |
| echo ‘hello “there”’ | Hello “there” | Output contains double quote in one of the words |
| echo ‘ this is a bq ` ‘ | this is a bq ` | ` is treated as part of the output rather than ignore |
| echo ee > e.txt |  | New file created named e.txt containing ee |

## 5.4 Test Cases: head

Files:

* head1.txt, head2.txt
* input1.txt, input2.txt, input3.txt, input4.txt input5.txt, input6.txt, input7.txt, input8.txt, input9.txt, input10.txt
* output1.txt, output2.txt, output3.txt output4.txt, output5.txt, output6.txt, output7.txt, output8.txt, output9.txt, output10.txt
* invalid command.txt

Files Description:

* head files contains the content for tail test
* input files contains the input command
* output files contains the expected output
* invalid command.txt contains the list of invalid command

|  |  |
| --- | --- |
| Input | Expected Output |
| head head1.txt | Select BOOLEAN such that Follows (5, 6)  false  5000  2 - comment  stmt s1;  Select s1 such that Follows (54, s1)  5000  3 - comment  stmt s1;  Select s1 such that Follows (s1, 178) |
| head head1.txt –n 3 | Select BOOLEAN such that Follows (5, 6)  false  5000 |
| head head1.txt –n 2 | Select BOOLEAN such that Follows (5, 6)  false |
| head head1.txt –n 2 –n 2 | Select BOOLEAN such that Follows (5, 6)  false |
| head –n 5 head1.txt –n 2 | Select BOOLEAN such that Follows (5, 6)  false |
| head –n 5 head1.txt –n 2 –n 5 | Select BOOLEAN such that Follows (5, 6)  false  5000  2 - comment  stmt s1; |
| head –n 2 head1.txt head2.txt –n 1 | Select BOOLEAN such that Follows (5, 6)  asdnfas;fnka |
| head –n 3 head1.txt –n 2 head2.txt | Select BOOLEAN such that Follows (5, 6)  false  asdnfas;fnka  sanla;fn;akdf; |
| head –n 3 head1.txt –n 2 head2.txt head1.txt | Select BOOLEAN such that Follows (5, 6)  false  asdnfas;fnka  sanla;fn;akdf;  Select BOOLEAN such that Follows (5, 6)  false |
| head –n 3 head1.txt –n 2 head2.txt head1.txt –n 4 –n 2 | Select BOOLEAN such that Follows (5, 6)  false  asdnfas;fnka  sanla;fn;akdf;  Select BOOLEAN such that Follows (5, 6)  false |

**Invalid Test Cases**

|  |
| --- |
| Input |
| head –head1.txt |
| head –n –n |
| head –n 3 –n |
| head head1.txt 3 |
| head –n 2 head1.txt 3 |
| head –n head2.txt –n 5 –n 6 head2.txt |
| head 6 head1.txt |
| head head1.txt Head1.txt |

## 5.5 Test Cases: tail

Files:

* tail1.txt, tail2.txt
* input1.txt, input2.txt, input3.txt, input4.txt input5.txt, input6.txt, input7.txt, input8.txt, input9.txt, input10.txt, input11.txt
* output1.txt, output2.txt, output3.txt output4.txt, output5.txt, output6.txt, output7.txt, output8.txt, output9.txt, output10.txt, output11.txt
* invalid command.txt

Files Description:

* tail files contains the content for tail test
* input files contains the input command
* output files contains the expected output
* invalid command.txt contains the list of invalid command

|  |  |
| --- | --- |
| Input | Expected Output |
| tail tail1.txt | stmt s1;  Select s1 such that Follows (\_, 1)  5000  5 - comment  stmt s1;  Select s1 such that Follows (\_, s1)  10,100,101,103,104,105,107,109,110,115,116,118,119,12,120,121,122,123,124,125  6 - comment  stmt s1;  Select s1 such that Follows (150, \_) |
| tail –n 3 | 6 - comment  stmt s1;  Select s1 such that Follows (150, \_) |
| tail –n 3 –n 2 | stmt s1;  Select s1 such that Follows (150, \_) |
| tail tail1.txt –n 2 | stmt s1;  Select s1 such that Follows (150, \_) |
| tail tail1.txt –n 2 –n 2 | stmt s1;  Select s1 such that Follows (150, \_) |
| tail –n 5 tail1.txt –n 2 | stmt s1;  Select s1 such that Follows (150, \_) |
| tail –n 5 tail1.txt –n 2 –n 5 | Select s1 such that Follows (\_, s1)  10,100,101,103,104,105,107,109,110,115,116,118,119,12,120,121,122,123,124,125  6 - comment  stmt s1;  Select s1 such that Follows (150, \_) |
| head –n 2 tail1.txt tail2.txt –n 1 | stmt s1;  Select s1 such that Follows (150, \_)  afa23  341234 |
| head –n 3 tail1.txt –n 2 tail2.txt | stmt s1;  Select s1 such that Follows (150, \_)  afa23  341234 |
| head –n 3 tail1.txt –n 2 tail2.txt tail1.txt | stmt s1;  Select s1 such that Follows (150, \_)  afa23  341234  stmt s1;  Select s1 such that Follows (150, \_) |
| head –n 3 tail1.txt –n 2 tail2.txt tail1.txt –n 4 –n 2 | stmt s1;  Select s1 such that Follows (150, \_)  afa23  341234  stmt s1;  Select s1 such that Follows (150, \_) |

**Invalid Test Cases**

|  |
| --- |
| Input |
| tail –tail1.txt |
| tail –n –n |
| tail –n 3 –n |
| tail tail1.txt 3 |
| tail –n 2 tail1.txt 3 |
| tail –n tail2.txt –n 5 –n 6 tail2.txt |
| tail 6 tail1.txt |
| tail tail1.txt Tail1.txt |

# Extended Functions (EF1) Test Cases

## 6.1 Test Cases: cal

Files:

* input1.txt, input2.txt, input3.txt, input4.txt input5.txt, input6.txt
* output2.txt, output3.txt, output4.txt output5.txt, output6.txt

Files Description:

* input files contains the input command
* output files contains the expected output
* There is no output1.txt as the results may varies

|  |  |
| --- | --- |
| Input | Expected Output |
| cal | February 2017  Su Mo Tu We Th Fr Sa  1 2 3 4  5 6 7 8 9 10 11  12 13 14 15 16 17 18  19 20 21 22 23 24 25  26 27 28  (Note: Output may vary. Prints current month) |
| cal 2017 | **Refer to Output1.txt** |
| cal abc | cal: Invalid argument |
| cal 2 2017 | February 2017  Su Mo Tu We Th Fr Sa  1 2 3 4  5 6 7 8 9 10 11  12 13 14 15 16 17 18  19 20 21 22 23 24 25  26 27 28 |
| cal 2 2 2 | cal: Invalid argument |
| cal –m 2 2017 | February 2017  Mo Tu We Th Fr Sa Su  1 2 3 4 5  6 7 8 9 10 11 12  13 14 15 16 17 18 19  20 21 22 23 24 25 26  27 28 |

## 6.2 Test Cases: grep

Files:

* grep.txt
* input1.txt, input2.txt, input3.txt, input4.txt
* output1.txt, output2.txt, output3.txt, output4.txt

Files Description:

* grep.txt contains the content for grep
* input files contains the input command
* output files contains the expected output

|  |  |
| --- | --- |
| Input | Expected Output |
| grep |  |
| grep pattern | grep: InputStream not provided |
| grep this grep.txt | this is a file "grep.txt" |
| grep this < grep.txt | this is a file "grep.txt" |

## 6.3 Test Cases: sort

Files:

* invalid command.txt
* sort1.txt, sort2.txt, sort3.txt, sort4.txt sort5.txt
* input1.txt, input2.txt, input3.txt, input4.txt input5.txt, input6.txt, input7.txt input8.txt
* output1.txt, output2.txt, output3.txt, output4.txt output5.txt, output6.txt, output7.txt output8.txt

Files Description:

* sort.txt contains files to be sorted
* invalid command contains a list of invalid command
* input files contains the input command
* output files contains the expected output

|  |  |  |
| --- | --- | --- |
| Input | Expected Output | Explanation |
| sort sort1.txt | $1000  %10$  010  1000 years of sorrow  18 days later  22  25 minutes ago  25jump street  Saving private ryan  kitten mitten  my world  my world  random world | sort in the order special chars, number, capital & simple letters |
| sort -n sort2.txt | "344"  //??  /n  \n  \n\r\  1  32 - 45  235235 sdfsdf  2"  23034-32  32-45  Chasing street food  Singing luluaby  kangoroo work  kk hospital  kkk  star wars | sort in the order special chars, number, capital & simple letters. Numbers are sorted in numerical order instead of ascii value  “input2.txt” contains empty new line in the file. These empty newlines are ignore by sort |
| sort sort3.txt | tab in front  tab1 in front  n space in front  space in front  123  2aa  f@llowing  f0llowing  following  words | sort in the order special chars, number, capital & simple letters. “input3.txt” contains empty new line in the file. These empty newlines are ignore by sort “tab” and “spaces” of each line is treated as a special char |
| sort –n sort4.txt | 132  234  315  324  1321  4523  324324  1535235 | sort numbers in numerical order |
| sort sort5.txt | #k#k  #kkk  $00  $999  ()  []  10 o'clock  100  123  20  20 dollars  33  A Star  Apple  Hello Panda  Hello world  Sleepy woods  addiTion  addition  g0od  gOOd  go0d  sad  sleeping beauty | sort in the order special chars, number, capital & simple letters. |
| sort –n sort3.txt sort4.txt | tab in front  tab1 in front  n space in front  space in front  123  132  234  315  324  1321  4523  324324  1535235  2aa  f@llowing  f0llowing  following  words | Combined the lines of sort3.txt & sort4.txt and sort them in order. Numbers are sorted in numerical order |
| sort sort1.txt sort999.txt | $1000  %10$  010  1000 years of sorrow  18 days later  22  25 minutes ago  25jump street  Saving private ryan  kitten mitten  my world  my world  random world | Sorted results of sort1.txt is displayed as there is no sort999.txt |
| sort sort999.txt sort1.txt | $1000  %10$  010  1000 years of sorrow  18 days later  22  25 minutes ago  25jump street  Saving private ryan  kitten mitten  my world  my world  random world | Sorted results of sort1.txt is displayed as there is no sort999.txt |

|  |  |
| --- | --- |
| Invalid command | Explanation |
| sort | No argument return error message |
| sort sort999.txt | No such file exist |

## 6.4 Test Cases: glob

Files:

* globTestSource1.txt, globTestSource2.txt
* input1.txt, input2.txt, input3.txt, input4.txt input5.txt
* output2.txt, output3.txt, output4.txt output5.txt

Files Description:

* input files contains the input command
* output files contains the expected output
* globTestSoure files contains the content to be glob

|  |  |  |
| --- | --- | --- |
| Input | Expected output | Explanaton |
| cat asds\*d.txt |  | File does not exist |
| cat ef1\_test\_cases\\glob\\glob\*Source1.txt | hello world\n | The content of the file is hello world |
| cat ef1\_test\_c\*s\\g\*b\\globTestSource1.txt | hello world\n | The content of the file is hello world |
| cat ef1\_test\_c\*s\\g\*b\\glob\*Sou\*e1.txt | hello world\n | The content of the file is hello world |
| cat ef1\_test\_c\*s\\g\*b\\glob\*Sou\*e1.txt ef1\_test\_c\*s\\g\*b\\glob\*e2.txt | hello world\nhello world 2\n | The content of the file is hello world and hello world 2 |
| cat Tests\\globFiles\\glob\*.txt | hello world\nhello world 2\n | There are 2 file meet this condition |

# Extended Functions (EF2) Test Cases

## 7.1 Test Cases: date

No test cases for “date” command as “date” is require to print the current time and data.

|  |  |
| --- | --- |
| Input | Expected Output |
| date | Tue Feb 20 18:24:43 SGT 2009 |

## 7.2 Test Cases: sed

Files:

* sed.txt
* invalid command.txt
* input1.txt, input2.txt, input3.txt, input4.txt input5.txt, input6.txt, input7.txt, input8.txt, input9.txt, input10.txt
* output1.txt, output2.txt, output3.txt, output4.txt output5.txt, output6.txt, output7.txt, output8.txt, output9.txt, output10.txt

Files Description:

* sed.txt contains the main content of the files for sed
* invalid command.txt contains the list of invalid command
* input files contains the input command
* output files contains the expected output

|  |  |
| --- | --- |
| Input | Expected Output |
| sed s/the/hello/ sed.txt | There was a cat which jump out of hello box with nothing in its mouth. It silence the fish that never return in a month |
| sed s/The/hello/ sed.txt | **hellore was a cat which jump out of the box with nothing in its mouth. It silence the fish that never return in a month** |
| sed s/o/0/g sed.txt | There was a cat which jump 0ut 0f the b0x with n0thing in its m0uth. It silence the fish that never return in a m0nth |
| sed s/' '/''/g sed.txt | Therewasacatwhichjumpoutoftheboxwithnothinginitsmouth.Itsilencethefishthatneverreturninamonth |
| sed s/' '/-/g sed.txt | There-was-a-cat-which-jump-out-of-the-box-with-nothing-in-its-mouth.-It-silence-the-fish-that-never-return-in-a-month |
| sed "s|o|/|g" sed.txt | There was a cat which jump /ut /f the b/x with n/thing in its m/uth. It silence the fish that never return in a m/nth |
| sed "s#o#|#g" sed.txt | There was a cat which jump |ut |f the b|x with n|thing in its m|uth. It silence the fish that never return in a m|nth |
| sed s/' mo'/???/g sed.txt | There was a cat which jump out of the box with nothing in its???uth. It silence the fish that never return in a???nth |
| sed s/th/"'#"/g sed.txt | There was a cat which jump out of '#e box wi'# no'#ing in its mou'#. It silence '#e fish '#at never return in a mon'# |
| sed s/things/mings/ sed.txt | There was a cat which jump out of the box with nothing in its mouth. It silence the fish that never return in a month |

|  |
| --- |
| Invalid Command |
| sed s/a/"/"/ sed.txt |
| sed s/the/here sed.txt |
| sed s/my/g sed.txt |
| sed s/the/'#"/ sed.txt |
| sed s/the/hello world/ sed.txt |
| sed s//replace/ sed.txt |

## 7.3 Test Cases: wc

Files:

* wc1.txt, wc2.txt
* input1.txt, input2.txt, input3.txt, input4.txt input5.txt, input6.txt, input7.txt, input8.txt, input9.txt, input10.txt
* output1.txt, output2.txt, output3.txt, output4.txt output5.txt, output6.txt, output7.txt, output8.txt, output9.txt, output10.txt

Files Description:

* wc1.txt & wc2.txt contains the main content of the files for wc
* input files contains the input command
* output files contains the expected output

|  |  |
| --- | --- |
| Input | Expected Output |
| wc wc1.txt | 22 74 457 wc1.txt |
| wc –l wc1.txt | 22 wc1.txt |
| wc –w wc1.txt | 74 wc1.txt |
| wc –m wc1.txt | 457 wc1.txt |
| wc -m wc1.txt -w | 74 457 wc1.txt |
| wc -m wc1.txt -w wc2.txt | 74 457 wc1.txt  6 60 wc2.txt  80 517 total |
| wc -m wc1.txt -m wc2.txt | 457 wc1.txt  60 wc2.txt  517 total |
| wc -m wc1.txt -l wc2.txt -l | 22 457 wc1.txt  5 60 wc2.txt  27 517 total |
| wc -m wc1.txt -l wc2.txt -l wc2.txt | 22 457 wc1.txt  5 60 wc2.txt  5 60 wc2.txt  32 577 total |
| wc wc1.txt wc2.txt wc2.txt | 22 74 457 wc1.txt  5 6 60 wc2.txt  5 6 60 wc2.txt  32 86 577 total |

## 7.4 Test Cases: piping operator

Files:

pipe1.txt, pipe2.txt

* input1.txt, input2.txt, input3.txt, input4.txt input5.txt, input6.txt, input7.txt, input8.txt, input9.txt, input10.txt
* output1.txt, output2.txt, output3.txt, output4.txt output5.txt, output6.txt, output7.txt, output8.txt, output9.txt, output10.txt

Files Description:

* pipe1.txt & pipe2.txt contains the main content of the files for pipe
* input files contains the input command
* output files contains the expected output

|  |  |
| --- | --- |
| Input | Expected Output |
| cat pipe1.txt | grep "Select" | Select BOOLEAN such that Follows (5, 6)  Select s1 such that Follows (54, s1)  Select s1 such that Follows (s1, 178)  Select s1 such that Follows (\_, 1)  Select s1 such that Follows (\_, s1)  Select s1 such that Follows (150, \_) |
| cat pipe1.txt | grep "HILEf" |  |
| head pipe1.txt -n 3 | grep "Select" | Select BOOLEAN such that Follows (5, 6) |
| head pipe1.txt -n 3 | grep "Select" | tail pipe2.txt | asdnfas;fnka  sanla;fn;akdf;  aer321324  alan  afa23  341234 |
| cat pipe1.txt | grep "Select" | grep "BOOLEAN" | Select BOOLEAN such that Follows (5, 6) |
| cat pipe1.txt | grep "Select" | grep "5000" |  |
| head pipe2.txt -n 2 | wc -m | 30 |
| tail pipe2.txt | head pipe1.txt -n 2 | wc | 2 8 48 |
| echo "bye" | echo "Hello" | Hello |
| cat pipe1.txt | echo "hello" | hello |

|  |
| --- |
| Invalid Command |
| head pipe1.txt -n 3 | grep "Select" -n 3 |
| head pipe1.txt -n | wc |
| cat pipe1. | grep “Select” |
| cat head pipe1.txt | grep "123" |
| cat head pipe1.txt | tail "123" |