CS4218 Software Testing

Milestone 1

Preliminary Steps

- Understanding the requirements/specifications and list down our assumptions.
- Exploring the codebase to learn about the application modules and find out what has already been done/what needed to be done
- Coming up with a set of github contribution guidelines.
- Listing down a preliminary set of implementations and tests to be done using Github Issues.
- Each group member was to self-assign themselves issues to work on.

Implementation Plan

- From our preliminary steps, we have a rough idea of how the shell application works.
- From here, our team has decided to go with the approach of further understanding how the different modules interact with each other. The idea we had in mind was that if we can craft out/visualise the dependency graph of the application, then we are able to identify which components are very modularised and which components are highly dependent on the rest.
- As such, our goal was to first implement/fix the smaller modular components and slowly work our way up to the larger and more tightly coupled components. We believe that this can better sieve out the existing bugs in the application.
- For each functionality to be implemented:
    - We read and tried to understand the specification in the document and interface
    - If we had any doubts, we based the implementation on the behaviour of the UNIX shell and info in man pages, and documented assumptions if necessary
    - We then coded the implementation according to the specification and interface and did rough manual testing
    - To keep methods self-contained, we also ensured input validation is done in interface methods instead of the 'run' method

Testing Plan

- In general, we came up with both black-box tests (on the 'run' method') and white-box tests (on the interface methods)
- Several unit tests needed to be mocked or stubbed
- To generate test cases, we aimed to cover all the different combinations of arguments and applied techniques like MC/DC and branch coverage if they were suitable
- For negative test cases, we aimed to cover classes of corner cases such as: no arguments, null arguments, single argument, insufficient arguments, multiple arguments, invalid flags, and nonexistent files etc.
- For positive test cases, we aimed to cover classes of valid cases such as: minimum arguments, maximum arguments, no flag, single flag, multiple flags, stdin, single file, multiple files

- We tried to check that the test output is as expected as far as possible: exception, number of files, file contents, number of lines printed, line content etc.
- So that there are no files or directories left behind that may affect subsequent tests, we ensured that there is clean setup before tests and clean teardown after tests,
- After working on unit tests to a satisfactory level, we started working on integration tests such as using multiple commands
    - See CallCommandIT.java, PipeCommandIT.java and CommandBuilderIT.java (under ef2)

Summary of Test Cases

Package	Test Classes	Method Coverage	Line Coverage
sg.edu.nus.comp.cs4218.exception	*ExceptionTest	100%	100%
sg.edu.nus.comp.cs4218.impl.parser	*ParserTest	100%	100%
sg.edu.nus.comp.cs4218.impl.result	*ResultTest	100%	100%
sg.edu.nus.comp.cs4218.impl.app	CatApplicationTest	100%	90%
	CdApplicationTest	100%	100%
	EchoApplicationTest	100%	100%
	ExitApplicationTest	100%	100%
	GrepApplicationTest	100%	85%
	LsApplicationTest	100%	91%
	MvApplicationTest	100%	100%
	SplitApplicationTest	100%	85%
	TeeApplicationTest	100%	75%
	WcApplicationTest	100%	87%
sg.edu.nus.comp.cs4218.impl.util	ArgumentResolveTest.	62%	61%
	CollectionUtilsTest	100%	100%
	CommandBuilderTest	100%	81%
	IORedirectionHandlerTest	100%	97%
	IOUtilsTest	100%	68%

	RegexArgumentTest	100%	100%
	StringUtilsTest	100%	100%

Positive Scenarios Considered	Negative Scenarios Considered
Different flag passed in one at a time and in combination Different types of input for arguments that lead to different paths / resulting in different conditions being met. Different number of arguments passed to the function under which the app still works. A variety of String content including empty string, different regex and special characters and null values A variety of file and folder types are included such as empty file, nested folder, empty folder and folder with hidden file. Different Input and output streams are considered (i.e. whether reading from stdin or file, whether the steam is null or a dash) Specific tests with unique behaviour to a function are considered. (i.e. when moving 2 files together, only the conflicting one is rejected; different byte option gives different number of splits)	Null arguments, wrong number of arguments, wrong input type for arguments. Null or wrong streams. Invalid flags, wrong number of flags and wrong combination of flags. I/O Exceptions such as file not found, wrong file permission, file is not a directory etc. Specific tests with unique behaviour that fails under a certain condition. (i.e. remove non empty directory)

Build Automation, Continuous Integration and Testing

- To further facilitate our implementation and testing workflow, our team has decided to include build automation as well as CI/CT into our project.

Gradle

- Using Gradle, we have automated the PMD static code analyzer to run during tests or builds. As such, whenever there are any PMD warnings, tests/builds will fail and we will be alerted.
- Additionally, we have included Gradle build scan that will automatically generate a test report to provide insights to our testing (see https://gradle.com/s/xtktcd6sg52oa). This has proven to be

extremely useful as some of our team members have sometimes relied on the test report to locate bugs.

Travis CI

- Furthermore, we have also enabled Travis CI to build and test our src files on Github (see https://www.travis-ci.com/nus-cs4218/cs4218-project-ay2021-s2-2021-team12). This is so that whenever we make a PR, we can be rather confident that there will not be any potential (regression) bugs and that our build is passing.

Codecov

- Finally, our team has also included Codecov for a deeper analysis of our code coverage. The code coverage provided by this platform is much more comprehensive than provided in Intellij. See https://app.codecov.io/gh/nus-cs4218/cs4218-project-ay2021-s2-2021-team12 (may need admin access to cs4218 repo).