https://culearn.carleton.ca/moodle/pluginfile.php/2453795/mod_resource/content/1/myreport.pdf

Report template above

Title Page

1.0 Introduction

1.1 - Context

While using a terminal console in the UNIX system, there are three main activities the average user will spend their time doing - edit documents, compile/run software, and navigate their file system. Our project addresses the later of the three, the topic of navigating the file system in UNIX. More specifically, we will be adding functionality to the '*ls'* command, short for *list*, from the *GNU Coreutils* package.

Though '*ls'* has modifiers with arguments allowing for different display models of information, our team finds that they are lacking in the elegance required to properly model the file system in a sufficiently aesthetic fashion. You can read about the Linux command line and '*ls'* documentation in '*The Linux Command Line"* by William Shotts.

1.2 - Problem Statement

As previously mentioned, our team observes an insufficient implementation of the *ls* command. While many users use '*ls'* as a main tool for navigating and searching through directories, we find that there is not enough focus on an oversight of the file system, leaving the file structure surrounding your current working directory (CWD) unknown.

The current solution attempting to provide an oversight of directories is the addition of the argument -*R*, meaning the *'ls'* command is *'ls -R'* which calls *'ls'* on the CWD, as well as recursively on any subdirectories it may have. We find this insufficient for two main reasons. One, there is no input parameter for the depth of subdirectories which we would like it to display. Secondly, *'ls -R'* lists directories one at a time rather than in a clearly readable format, such as a tree layout.

Our first goal is to add an input parameter to limit the depth of this tree according to the user's requirement. We hope this will improve both ease of use as well as limit the amount of system memory and work that is to be allocated to this job - why should the system recurse through every folder if the user's goal is only to search through subdirectories one step down?

Our second goal is to ensure that when displayed, the file structure contents are not shown directory by directory, but instead in a tree format resembling an expanded cascading list. We hope to explore the color tools the command line offers in hopes of adding to the improved aesthetic of the command.

1.3 - Result

Need to wait for completed software.

In reference to your problem(s) and goal(s), describe briefly what you have been able to achieve, i.e., the highlights of your project. How have you solved your problem(s)? How have you achieved your goal(s)?

1.4 - Outline

The rest of this report will detail the background, process, and results of our efforts. The structure will be as follows; section 2.0 presents background information relevant to *'ls'* and *'ls -R'*. Section 3 debriefs the reader on the results obtained, and section 4 continues to describe any barriers to the development of the improvement, and new findings that may understanding of our original project scope. As part of this section, ease of use and user evaluation is mentioned. We will conclude in section 5 with our closing statements on the project.

Structure:

2.0 Background Information

2.1 - The *‘ls’* Command

The ‘ls’ command, abbreviated from the word list, is specified by POSIX and is one of the most standard commands in the GNU Core Utilities package. When invoked the command lists all information about the files in the specified directory.

The utility first appeared in the original AT&T version of Unix and was derived from similar commands named ‘listf’ from the Multics operating systems. Even back in 1969, during the creation of Multics, ls was rooted as a core command, the Multics OS was one of the first to provide hierarchical file systems with symbolic links and removable devices and thus users needed efficient means to traverse data.

Nowadays the ‘ls’ command is used by most if not all terminal users running Unix based Operating systems. By default, without any flags, the ls command will print out the files from your CWD or current working directory, if you have just opened your terminal the PATH would be under your /home/$USER directory. The command has around 60 flags that can be used to change your output, this can range from changing the colour of directories printed, hyperlinking certain file names, or displaying block-sizes of files.

2.2 - The *‘ls -R’* Command

The -R flag in particular is used to recursively print out subdirectories. If you have ever wanted to be able to view not just the toplevel of your directory but also what is inside the child directories of your path then you would use this flag.

When this flag is initialized the ls command creates a data structure that detects and directory cycles and uses recursion to traverse a directory structure, that is why you will most likely view the output as messy and unorganized. Most use cases for this flag are in when you are in lower level directories with fewer files or when this command is piped into another for better readability.

2.3 - File System Usability

.

3.0 Result

3.1 - Walkthrough

Adlib words.

3.2 - Feature Breakdown

Adlib words.

Describe the work you have been able to achieve. Review the interface to your software with representative use cases. You may include screen shots. Describe the architecture and logic of your software. You may include UML like diagrams. It is rarely useful to include code, except if you wish to describe something really intricate. If you do, make it brief. Any code snippet must fit in less than a page. Describe the code in the text of your report, line-by-line.

If you make claims about your software such as better, faster, smaller or easier to use, be ready to support your claims with evidence, that is objective evaluations (see Section 4).

It is good to use figures to clarify your ideas. If you use figures (screen shots, diagrams), create them yourself. Number the figures sequentially. Accompany each figure with a caption. In the text of your report, refer to each figure by number. Describe every element present in the figure.

4.0 Results: Evaluation Debrief

4.1 - Developer Review

Adlib words.

4.2 - Issues & Findings

Adlib words.

4.3 - Ease of Use Study

Adlib words.

4.4 - Ease of Use Results

Adlib words.

Develop test cases that exercise typical execution paths of your software. Be honest, self critical and objective about your work. If you make claims related to performance, such as fast or small, provide an analytic model (e.g., complexity analysis) or statistical data to support your claims. Try your software under typical scenarios. Collect data, e.g., execution time, connection time, response time (time required to process a request), round trip time or transfer time, delay, latency, throughput (units of work versus units of time), resource, CPU, memory or bandwidth usage, error or success rate. Plot and analyze the data. If you make claims such as faster or smaller, then explain with respect to what and why. Provide comparative analyses or statistical data.

If you make claims such as easy to use, conduct a usability study involving people from the community of potential users of you software (see Ref. [3]). Prepare evaluation scenarios and evaluation questions. Collect comments. Analyze them. Ideally, the users who participate in your study should be independent and critical. You may involve class mates, friends or family members in your study. If you make claims such as easier to use, then explain with respect to what and why. Conduct a comparative usability study.

If your project is security related (e.g., attack method, detection or mitigation), then use evaluation metrics such as success rate, false positive alarms or false negative alarms.

Data is always easier to grasp when presented graphically, rather than in a cryptic numerical tabular form. Use plots, scatter plots, bar charts, pie charts, box plots, etc. You may use colors, but plots, charts and diagrams must also be readable when printed black and white, without any loss of information

5.0 Conclusion

5.1 - Goal Review

5.2 - Closing Statements

6.0 References and Appendices

6.1 - References

Adlib words.

6.2 - Appendices

Adlib words.