Reconstructing the 'LS' Command

Mathieu Schmid and Michael Stupich
December 9th 2016

1. Introduction

1.1. Context

In many UNIX systems, *ls*, short for the word *list*, is used to display files. When used without any argument, the *ls* command lists the files and folders in the current working directory (CWD), that is where one is currently positioned in the hierarchy of directories. This command, along with many other commonly used commands (*mkdir*, *rm*, *cp*, *whoami*), was introduced as part of the **GNU coreutils** package.

When using *ls* without any arguments, it is difficult to establish the permissions, type, and size of the files. There exists expansions (arguments) in order to expand the level of information the user receives when displaying files, but these are not always easy to read and understand.

1.2. Problem Statement

In our day-to-day lives as computer scientists, we have a lot of folders in which we use to organize our files. At the moment in the command line, if you want to view files and folders within the folders of the current directory you're in, you need to specify or travel to those directly and re-invoke the *ls* command. But what if you want to go more in depth and view the contents of the folders of your CWD, similar to a file tree? Currently, there's only one argument that exists (*-R*) for the *ls* command which recursively lists the elements within the subdirectories of your CWD. The way this works is for every folder within your CWD, it recursively calls the *ls* command and displays the results until there are no more subdirectories left to display. This expansion is far from perfect. Not only is this bad on the system's memory, it is hard to read and understand.

Our goal is the create a more readable expansion to the *ls* command. With *ls* being arguably the most consistently used bash command, why not improve it? We want the user to be able to see a greater depth of files within subdirectories while using this command. By default, we're setting the max depth of recursion to 2, but hope to allow the user to specify, using command line arguments, how many levels of depth they wish to display. This means that, by default, when invoking our command, the user will see the contents of the current working directory, as well as the contents of the folders within the CWC. We will be setting the argument to be *–M* (for Mat and Mike), so in order to invoke our expansion the user simply types ***ls –M*** in their terminal.

1.3. Results

We managed to add a new argument to the *ls* function, which by default displays 2 levels of folder depth and can be invoked using ***ls –M***. It is also fully integrated in *ls*'s *--help* manual which explains how to use our expansion as well as every other. Not only did we achieve our original goal, we also implemented an additional optional ***--depth*** argument, which allows the user to specify how many levels of depth they wish to view. For example, if the user wants to view 4 levels of depth, they would call the function by doing *$:* ***ls – M --depth=4***.

1.4. Outline

Section 2. Background Information:
- Locate *ls* source code
- Find what we can build off of
- Distinguish which features we want to integrate
- Visualize desired output
- Demonstrate how we test our modifications

Section 3. Result:
- Describe what we were able to achieve
- How to get it working on your machine
- Show how we displayed the output
- Runtimes

Section 4. Evaluation:
- How well did we achieve our goal
- Usability report

Section5. Conclusion
- Are we happy with the results
- Relevance to the course
- Final words

2. Background Information

In order to accomplish our goal of modifying the *ls* command, we need to find the source code for it. As opposed to where we thought the source code would be found, in the kernel, it's found and downloaded at the GNU coreutils website [1]. GNU is a software foundation which aims to give more freedom to UNIX based operating systems (OS) users. They provide the source code for an entire UNIX based OS. The coreutils package, one of many packages they have available which allows the users to study, change, and improve upon, contains the code we are looking for. Along with *ls.c*, this package contains the code for numerous other command line functions.

Now, to accomplish our vision of giving the users a greater depth into their folders and file, we need to implementing a file tree. To do this, we're going to need to dig through *ls.c* and find if there's any pre existing code we can build off of. Currently, the only occurrence of a file tree in the source code is for the *–R* argument. We will use this as our base code.

Firstly, we need to decide how we want to display the output of our new expansion. With our goal of making this a more *readable* and *user-friendly* addition to the already existing *–R* argument, it's going to need a few key features:

1. It needs to be as minimalistic as possible
    a. Remove all clutter and unnecessary fluff
    b. Set the recursion depth level to 2 subfolders by default
2. It needs to stay consistent with the current architecture of bash commands
    a. No need for complicated external functions, simply a new argument to *ls*
    b. Add functionality instructions in the already existing *--help* which contains the information about all other function arguments.

With these features in mind, it's time to visualize a desired output.

```
/user/Documents/ $: ls
file1.1.txt          folder1.1
file1.2.txt          folder1.2
```

Fig. 2.1

In figure 2.1 we demonstrate what the console output would be for the *ls* command in the *Documents* folder without invoking any arguments to the function. As we can see, all that is displayed is the names of the files and folders in the CWD.

```
/user/Documents/ $: ls -R
file1.1.txt          folder1.1
file1.2.txt          folder1.2

./folder1.1:
file2.1.txt    folder2.1

./folder1.1/folder2.1:
file3.1.txt    folder3.1

./folder1.1/folder2.1/folder3.1:
folder4.1

./folder1.1/folder2.1/folder3.1/folder4.1:

./folder1.2:
file2.1.txt
```

Fig 2.2

Figure 2.2 displays the result of *ls*'s recursive argument *-R*. This outputs the contents of the CWD and its subdirectories contents recursively until it reaches the end of the files tree. *Folder 1.1* is shown containing 3 other folders within itself. When each of these folders are populated with more files and folders, it becomes very unorganized.

```
/user/Documents/ $: ls -M
file1.1          folder1.1
file1.2          folder1.2

./folder1.1:
file2.1.txt    folder2.1

./folder1.2:
file2.1.txt
```

Fig 2.3

In figure 2.3 the ideal output of our expansion to the *ls* command is displayed. It is initiated with ***ls –M***, and is showing 2 levels of recursion depth. We hope that 2 levels are enough to give the user enough information about their CWD while not filling their terminal with unwanted fluff.

Next, we need to find out how to test our modifications. We have our base code, and an idea on how to make our expansion, but how do we test it? The coreutils package contains an executable which can be invoked using *$: ./configure prefix=/home/user/* which configures where the makefile will compile the code. This destination is given in the path specified by the prefix argument. If the prefix were set to equal //, it will override the bin directory on the machines root files which will allow the function modifications to take effect system wide.

Once configured, running the makefile and installing the new compiled files is required in order to see if our modifications are functioning as intended.

3. Results

3.1 What we achieved

We successfully implemented our expansion of the *ls* command. Our improved software is able to run on any UNIX based operating system. We designed and worked on it in our *Virtual Box* VM of *Ubuntu 16.04 LTS*. Although *ls.c* is the only file we modified, we will be submitting the entire *coreutils* package as a *.zip* since it includes all the makefiles and configure files needed to implement our expansion.

3.2 How to get it working on your machine

Once the package is downloaded and extracted on the machine, lets say the *Desktop*, we first need to travel to the coreutils directory in our terminal.

```
~/Desktop/coreutils-8.26$
```

Second, we need to specify where the package should compile its files. This is done using the *configure* executable and setting a prefix to a desired location.

```
$ ./configure prefix=/home/user/
```

In addition, we need to look in the prefix location for the *bin* folder by using the *PATH* command.

```
$ PATH=/home/user/:$PATH
```

Next, we simply compile the entire package, which contains our modified *ls.c* file, by using the *make* command.

```
$ make
```

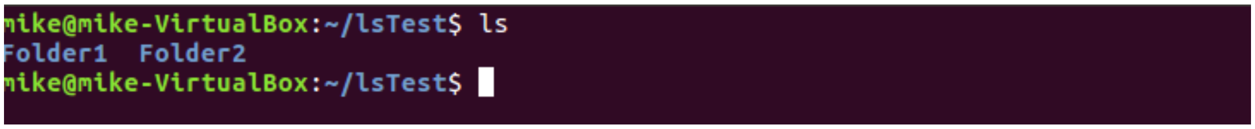
Finally, we need to install these compiled files and configurations to our system using the *install* command as admin user (*sudo*).

```
$ sudo make install
```

Once this is completed, the newly added expansion to the *ls* command should be working.

3.3 What does it output

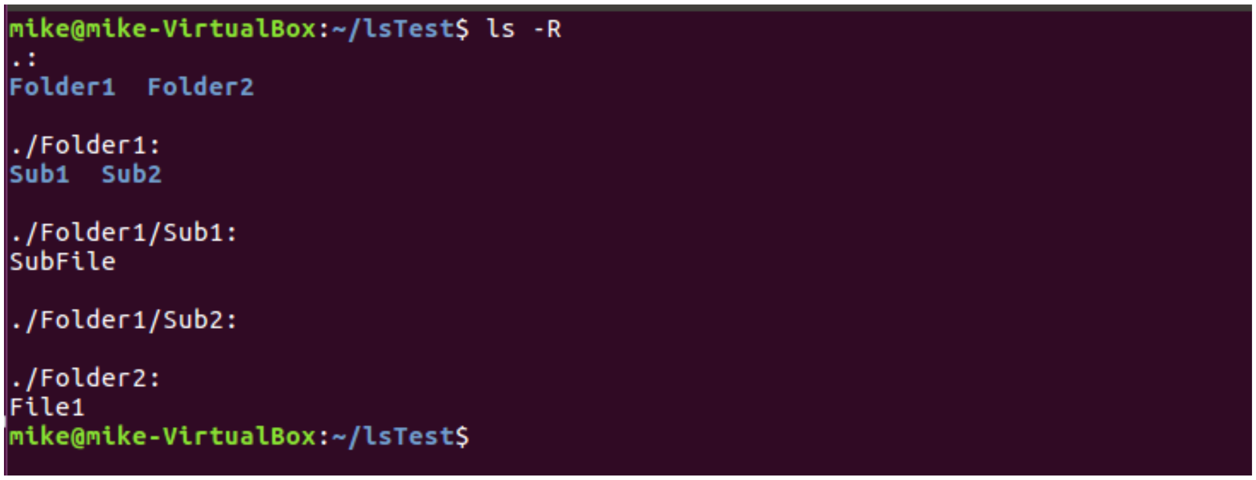
First in figure 3.1, the terminal command ***ls*** lists the file and folders in your current directory. It is very simple, but not very informative.



```
ike@mike-VirtualBox:~/lsTest$ ls
older1  Folder2
ike@mike-VirtualBox:~/lsTest$ █
```

Fig 3.1

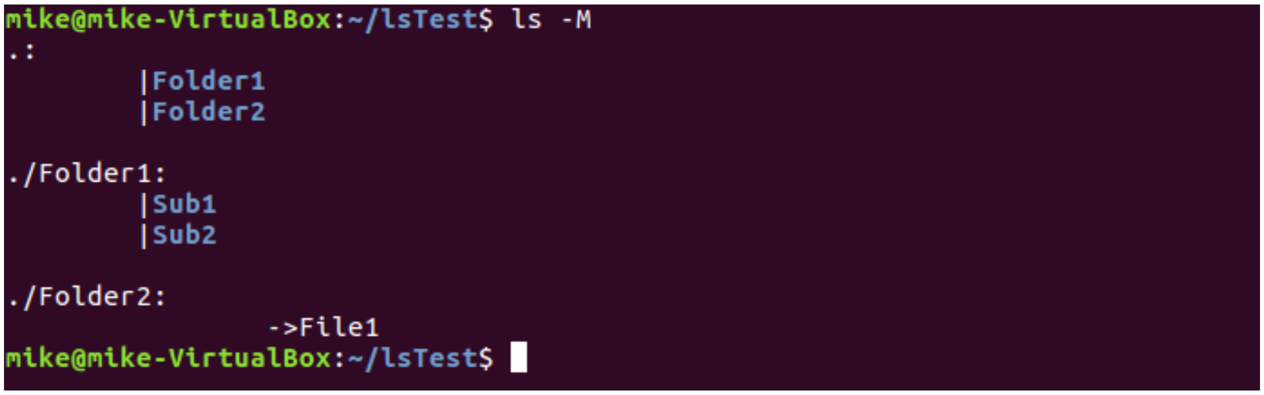
Second, the results of ***ls –R*** in figure 3.2. This shows the results of a maximum depth recursion algorithm. This is demonstrated on a relatively small folder, but once it's used on a more sizeable folder, it becomes difficult to read.



```
mike@mike-VirtualBox:~/lsTest$ ls -R
.:
Folder1  Folder2

./Folder1:
Sub1  Sub2

./Folder1/Sub1:
SubFile

./Folder1/Sub2:

./Folder2:
File1
mike@mike-VirtualBox:~/lsTest$
```

Fig 3.2

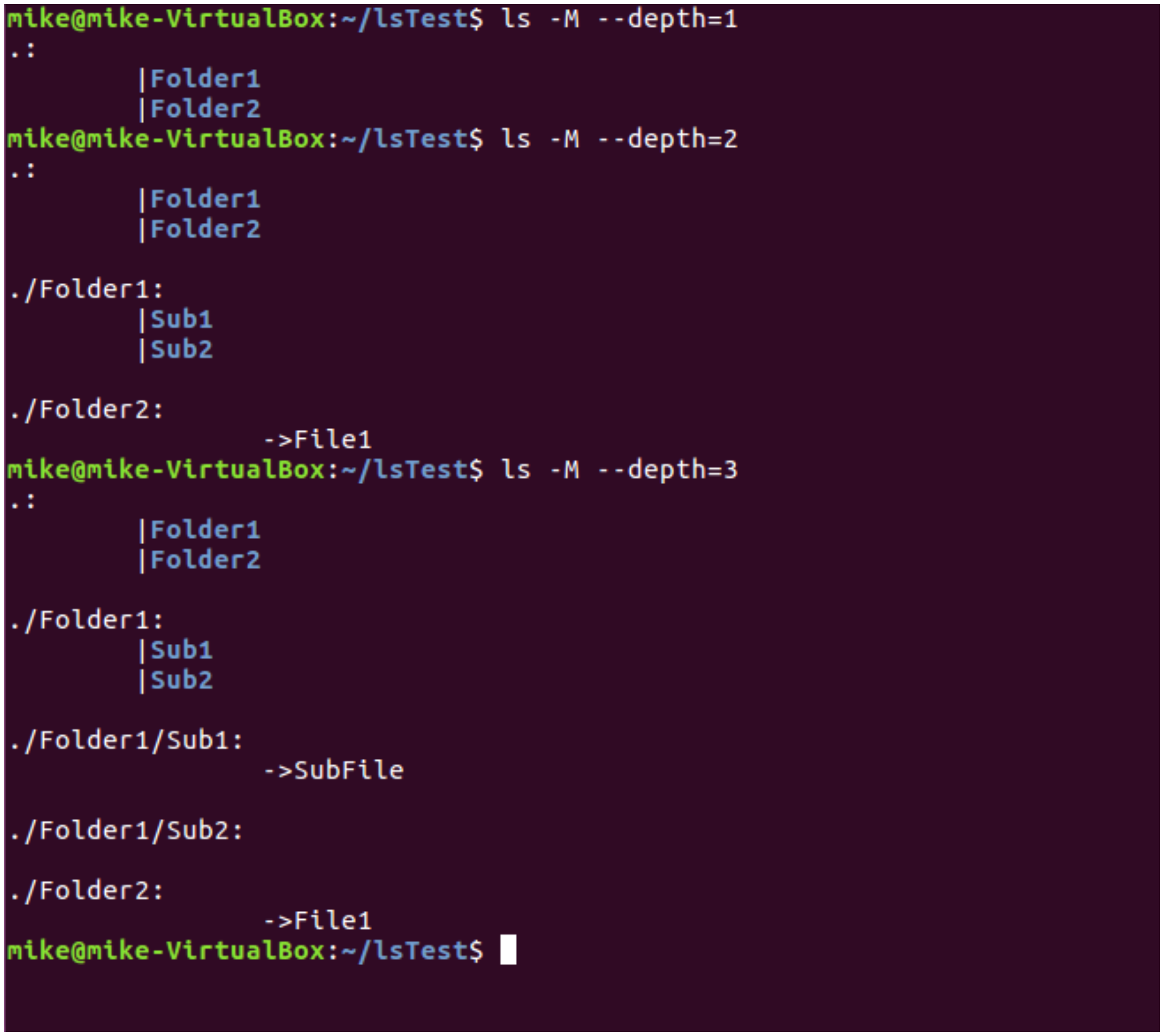
Third, in figure 3.3 the output of our command ***ls –M*** is shown being used without any additional arguments. It caps the depth of the recursion algorithm to 2, which allows the user to get much more information without being cluttered with unnecessary fluff.



```
mike@mike-VirtualBox:~/lsTest$ ls -M
.:
        |Folder1
        |Folder2

./Folder1:
        |Sub1
        |Sub2

./Folder2:
                ->File1
mike@mike-VirtualBox:~/lsTest$ █
```

Fig 3.3

Finally, figure 3.4 demonstrates the additional *depth* argument which can be invoked after the –*M* like this: ***ls –M --depth=<int>***, where the int entered will be the level of the depth the user wishes to view.



```
mike@mike-VirtualBox:~/lsTest$ ls -M --depth=1
.:
        |Folder1
        |Folder2
mike@mike-VirtualBox:~/lsTest$ ls -M --depth=2
.:
        |Folder1
        |Folder2

./Folder1:
        |Sub1
        |Sub2

./Folder2:
                ->File1
mike@mike-VirtualBox:~/lsTest$ ls -M --depth=3
.:
        |Folder1
        |Folder2

./Folder1:
        |Sub1
        |Sub2

./Folder1/Sub1:
                ->SubFile

./Folder1/Sub2:

./Folder2:
                ->File1
mike@mike-VirtualBox:~/lsTest$
```

Fig 3.4

3.4 Let's talk about runtimes

Given the fact that *ls –R* is a recursive function which runs until the maximum depth of its subdirectories is reached, the very worst case is that *ls –M* and *ls –R* have the same runtime. This would only happen if maximum depth of a subdirectory is the same as the depth specified in *ls –M*, which by default is 2, but can be specified by the user.

3.5 How it was implemented

To be able to implement the utility of ls -M, we first had to understand the code that makes the *ls* commands work. More specifically, we had to understand how the *ls -R* command worked. Once we understood some of its inner workings, we could use it as a template to create our new argument, *ls -M*, with its augmented functionality and design.

The first step was to create our new command as a valid argument for *ls.* To accomplish this, we had to see how other arguments were added and use them as a guide. This included adding recognition for 'M', as well as creating some dialog in the *ls –help* to explain to the user what the command will accomplish.

That was the simple part. After that, we had to add the new layout as well as the depth to our command. To alter the layout, we created a new print format (tree_format) which adds tabs and symbols (->, |) to the files and directories listed as shown above. This was to done to improve the readability of the command, which was our main goal.

The most difficult task to complete was to find the depth of our current directory, so we could choose which depth of directories to show, and which to hide. The reason this was a difficult task was because the *ls* command does not keep track of the current depth. Several different attempts were made to resolve this problem, the first being to track the number of times the function extract_dirs_from_files was called. The problem with this method, was that it was called recursively on every directory, so keeping a counter for this will leave us with a different number when it is called on different directories. Several more attempts were made to keep track of the depth, at different points in the file. These all had the same shortcomings of the one described above. The next method was to analyze the filename before it was written to the stdout. This also did not work, as the files and directories are written to stdout individually, and formatting is added between them to distinguish them. This meant that we were unable to look for the number of occurrences of the forward slash character in the filename. While this attempt did not work, it did give us the idea to look in other places in the code, to see if we could use the entire directory name and search it for the number of forward slashes to determine the depth.

We then tried to implement it inside the extract_dirs_from_files function. After some trial and error, we finally found success. We determined the number of occurrences of the '/' character using the function strchr on the variable dirname. We used this command to get the current depth of the folder, and compare this to either the provided depth, or the default depth of 2.

4. Evaluation

4.1 Self-Critical evaluation

As computer scientists, we generally have an idea of what an ideal piece of software would resemble, and how it would react to different actions before commencing a project. We then try and recreate it in the development process. This project was different since neither of us had any experience with OS development we had no idea what to expect.

After having completed the original goals we set ourselves, we are both extremely content with the way it turned out. There were many bugs at first, such as the *–R* command no longer working (because we were building our command off the code of that one), but after setting some flags we haven't been able to recognize any other bugs. Our code is not perfect, but it gets the job done without any consequences to the rest of the OS.

4.2 Usability Report

We distributed our software amongst peers in order to get feedback. They were told to compare and rate out of 5 readabilities in the already existing *ls –R* and our expansion *ls –M*. We got results from 20 students and coworkers, all in the field of computer science. Here are the results we collected.

Of the 20 peers, 7 of them were already aware of ls's recursive argument (*ls –R*)

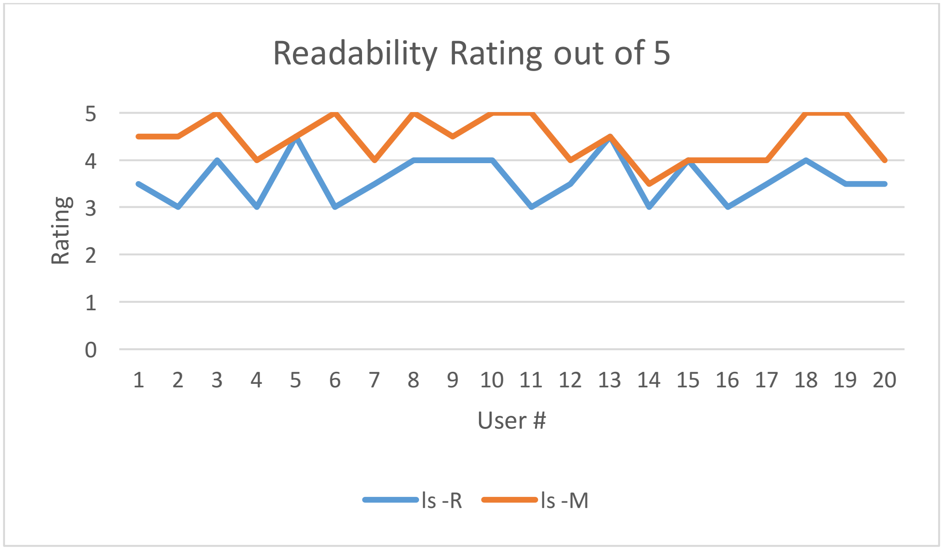
☻☻☻☻☻☻☻☹☹☹☹☹☹☹☹☹☹☹☹☹

The average readability rating for ***ls –R*** was **3.6 stars**

★★★☆☆

The average readability rating for ***ls –M*** is **4.45 stars**

★★★★☆



5. Conclusion

When presented with this project, our first concern was whether or not our goal was too ambitious and unrealistic. What if we weren't able to achieve our intentions? Before starting, we did some research and discovered that the size of the *ls.c* file was quite intimidating (5176 lines of code!). Not only is it a very sizeable file, it was originally written 30+ years ago. It definitely isn't the easiest task to study and modify code such as this.

We believe that the modifications made follow our two key features of being as minimal as possible and staying as consistent with the current framework for bash commands. Our claim that our expansion is more readable than the already existing –*R* is backed up by our usability report. Therefore, we can say that based on our user sample on which we tested our software, we can conclude that we made a more readable interface to the *LS* command. Additionally, our project stays very relevant to the material covered in this course in a sense that it not only taught us the inner workings of Linux OS development, it is applied on a function we both use very regularly.

With this said, we're really happy with the results we were able to achieve, and believe that this expansion, once familiar with the user, will be used as part of their day-to-day lives.

References:

[1] https://www.gnu.org/software/coreutils/coreutils.html

Our code for the LS command can be found at: https://github.com/schmidyy/DepthToLS