Assignment 1: Implementing linux `ls` with Python

EC602 Design by Software

Spring 2023

Contents

1 Introduction **1**
- 1.1 Assignment Goals . . . 1
- 1.2 Group Size . . . 2
- 1.3 Due Date . . . 2
- 1.4 Submission Link . . . 2
- 1.5 Points . . . 2

2 The Assignment **2**
- 2.1 About `ls` . . . 2
- 2.2 List files one-per line: `lsone` . . . 3
- 2.3 List files in column form: `lscol` . . . 3
- 2.4 Process file command line options: lsfiles . . . 4
- 2.5 Long Format: lslong . . . 4
- 2.6 Pulling it all together . . . 4

3 Program Restrictions **4**

1 Introduction

1.1 Assignment Goals

The first assignment goals are to ensure

- you have a suitable development environment available to you
- you are able to use the assignment submission system
- you understand executables and permission in the unix file system
- you can use the unix help system (man)
- you can write short programs in Python involving modules and terminal I/O

1.2 Group Size

For this assignment, the maximum group size is 2.

1.3 Due Date

This assignment is due 2023-9-24 at 23:59:00.

Late assignments will be accepted through a grace period of $H = 14.5$ hours.

The grade will be scaled on a linear scale from 100% at the deadline to 50% at the end of the grade period.

If the *natural grade* on the assignment is g, the number of hours late is h, and the grace period is H, then the grade is

```
grade = (h > H) ? 0 : g * (1- h/(2*H));
```

in C++ or

```
grade = 0 if h>H else g * (1- h/(2*H))
```

in Python.

If the same assignment is submitted ontime *and* late, the grade for that component will be the maximum of the ontime submission grade and the scaled late submission grade.

1.4 Submission Link

The submission link is

https://curl.bu.edu:9602/assess/hw/1/YOURUSERNAME/

where YOURUSERNAME would be replaced with your BU email.

1.5 Points

This assignment is worth 6 points.

2 The Assignment

This assignment will implement various versions and parts of the unix command `ls`.

2.1 About `ls`

All information about how `ls` works can be found using two methods

- reading the man page with `man ls`
- experimenting with `ls` in Linux or macOS to basically "reverse-engineer" its properties.

We will do both of these in class: you will probably need to do more investigations using these methods to complete the assignment.

To about confusion between our programs and the real `ls`, all of the assignment components will have names similar to but different from `ls`.

Note: the `ls` provided by macOS does not function in precisely the same way as `ls` as provided by Linux. Either operational style will be accepted by the grading system.

Note: `ls` provides short style options like `-t` as will as long-style options like `--time=WORD`. For this assignment, we will assume only short style options.

No testing will be done with any long style options, and you are not expected to handle these cases.

2.1.1 Quotes

Some filenames cause `ls` to add quote symbols around the actual filenames (‘filename’ or “filename”). This creates substantial additional complication to the problems described below.

Therefore, you can assume that the option to turn off this mechanism `-N` or `--literal` is always specified and so you should not attempt to add quotes around the filenames you print.

2.2 List files one-per line: `lsone`

The command

```
ls -1
```

where the `1` is a one not an ell, forces `ls` to print the file information one item per line.

Write an executable Python script `lsone` that performs this operation.

For this program, all command line arguments should be ignored.

2.3 List files in column form: `lscol`

The command

```
ls -C
```

forces `ls` to print the file information in columns.

Write an executable Python script `lscol` that performs this operation.

For this program, all command line arguments should be ignored.

2.4 Process file command line options: lsfiles

Write an executable Python script `lsfiles` that works the same way as `ls` when files and/or directories are specified on the command line.

Example:

```
ls *
```

No command line options will be specified.

Your program should automatically switch between one-line and column output in the same way that `ls` does.

2.5 Long Format: lslong

Write an executable Python script `lslong` that works the same way as `ls -l`

2.6 Pulling it all together

2.6.1 Real `lsreal`

Write an executable Python script `lsreal` that implements all of the previous program pieces but does so by processing command line arguments (both options and files) in the same way that `ls` does.

2.6.2 Wrapper: `lsall`

Write an executable Python script `lsall` that implements all of the previous parts of this assignment

- `lslong`
- `lsfiles`
- `lsone`
- `lscol`
- `lsreal`

by examining the 0th command line argument.

If these programs are all symbolic links to the real program `lsall`, then the real program `lsall` can behave in the same was as any of these five programs.

3 Program Restrictions

You may not access the actual program `ls` as part of any of your solutions, i.e. inside the Python scripts themselves. This means that, for example, you may not use:

- `os.system`
- `subprocess.run`

- `subprocess.Popen`

or any other such facility in Python to directly access the `ls` utility. You may use `os.listdir` and other features of the `os` and `sys` modules as needed.