0x01. C - ls

System programming & Algorithm — Linux programming

by Wilfried Hennuyer, Software Engineer at Holberton School

weight: 2

Ongoing project - started 06-12-2017, must end by 06-19-2017 (in 3 days) - you're 0% done.

QA review fully automated.

Re write the ls command.

Readme

Read `man ls` , Everything you need to know to write your own ls (https://intranet.hbtn.io/concepts/71).

man: see bellow.

What you should learn from this project

At the end of this project you are expected to be able to explain to anyone, without the help of Google:

- How does `ls` work?
- what are all the functions used by `ls`

Requirements

- Allowed editors: `vi` , `vim` , `emacs`
- All your files will be compiled on Ubuntu 14.04 LTS
- Your C programs and functions will be compiled with `gcc 4.8.4` (`C90`) using the flags `-Wall -Werror -Wextra and -pedantic`
- All your files should end with a new line
- A `README.md` file, at the root of the folder of the project is mandatory
- Your code should use the `Betty` style. It will be checked using betty-style.pl (https://github.com/holbertonschool/Betty/blob/master/betty-style.pl) and betty-doc.pl (https://github.com/holbertonschool/Betty/blob/master/betty-doc.pl)
- No more than 5 functions per file

- All your header files should be include guarded
- Valgrind should show 0 memory leak and 0 error
- Unless specified otherwise, your program **must have the exact same output** as `ls` as well as the exact same error output - except for alignment.

List of authorized functions and system calls

- `opendir` (man 3 opendir)
- `readdir` (man 3 readdir)
- `closedir` (man 3 closedir)
- `exit` (man 3 exit)
- `free` (man 3 free)
- `lstat` (man 2 lstat)
- `malloc` (man 3 malloc)
- `perror` (man 3 perror)
- `write` (man 2 write)
- `printf` (man 3 printf)
- `readlink` (man 2 readlink)
- `ctime` (man 3 ctime)
- `getpwuid` (man 3 getpwuid)
- `getgrgid` (man 3 getgrgid)

Compilation

Your program will be compiled this way:

```
gcc -Wall -Werror -Wextra -pedantic *.c -o hls
```

Every task depends on the previous ones. We strongly encourage you to read the entire project and think about the way you are going to design your entire `ls` before starting. (This is actually something you should do all the time :)).

Testing

Your program should give the same result as the real `ls` :

```
$ ls test
abc  BCD  file  file2  file3  folder1  folder2  folder3
$ ./hls test
abc  BCD  file  file2  file3  folder1  folder2  folder3
$ ls -1
abc
BCD
file
file2
file3
folder1
folder2
folder3
hls
$ ./hls -1
abc
BCD
file
file2
file3
folder1
folder2
folder3
hls
$
```

Tasks

0. What about options? mandatory

Implement the `-1` option.

Usage : `hls [-1] [FILE]...`

For the rest of the project, an option will be identified with the character `-` at the beginning of the parameter containing the option (like `ls`).

Done?

Help!

```
/simple_ls$ ls test -1
abc
ABC
file
File2
file3
folder1
Folder2
folder3
/simple_ls$ ./hls test -1
abc
ABC
file
File2
file3
folder1
Folder2
folder3
/simple_ls$
```

Pay attention to "edge cases": you should be able to handle multiple options, in any order.

Repo:

- GitHub repository: `holbertonschool-linux_programming`
- Directory: `0x01-ls`

Done with the mandatory task? Unlock 10 advanced tasks now! (/projects/308/unlock_optionals)