# CS 392, Systems Programming: Assignment 3

# Permission Find

## Overview

For this assignment, you will be creating a program that finds files with a specified set of permissions. The usage is to look as follows:

```
Usage: ./pfind -d <directory> -p <permissions string> [-h]
```

This program will recursively search for files whose permissions match the permissions string starting in the specified directory.

Permissions strings are to be formatted similarly to how the command `ls` formats them. There is, however, an important distinction. The `ls` permission string has 10 characters, where ours will only have the rightmost 9. In UNIX systems, the leftmost character specifies the type of file (d for directory, l for symlink, etc). Our program will not consider this, and will only worry about the rightmost 9.

Those characters are in the following format: `rwxrwxrwx`.

They can be broken up as follows:

```
| rwx  | rwx   | rwx   |
| USER | GROUP | OTHER |
```

(USER is synonymous with OWNER in this context)

The presence of `r` represents read permissions.

The presence of `w` represents write permissions.

The presence of `x` represents execute permissions.

If any of those characters are replaced by a dash (`-`), it represents a lack of permissions of that kind. For example, in the permission string `rw-r--r--`, the user has permission to read and write, while anyone who is not the file's owner but is in the user's group only has permission to read, as is the case with anyone else on the system.

Keep in mind, A DIRECTORY IS JUST A SPECIAL KIND OF FILE! So, you will be keeping directories in mind as well.

## Implementation specifics

You will be traversing the directory tree using the function `readdir` (see `man 3 readdir`) by first opening a directory (`man 3 opendir`), and for every file, checking the permissions on it using `stat` (`man 2 stat`). The function `stat` allows you to check what type of file it is (regular file, directory, symlink, block special, etc), and handle each accordingly.

## Step 1: Validate input

Check that all arguments passed to program are valid. Only -d, -h and -p flags should be passed, no arbitrary flags should be passed, and if -h is passed, then the usage statement should be printed with an exit status of EXIT_SUCCESS.

If no arguments are supplied, display the usage statement and exit the program with EXIT_FAILURE.

If -d is missing, print

```
Error: Required argument -d <directory> not found.
```

and exit the program with EXIT_FAILURE.

If -p is missing, print

```
Error: Required argument -p <permissions string> not found.
```

and exit the program with EXIT_FAILURE.

If an invalid directory `dir` is passed to -d, then the following error message will be printed to standard error and the status EXIT_FAILURE will be returned:

```
Error: Cannot stat 'dir'. No such file or directory.
```

If an invalid flag `f` is passed, the following error message will be printed to standard error and the status EXIT_FAILURE will be returned:

```
Error: Unknown option '-f' received.
```

## Step 2: Verify and resolve permissions string

You will be required to ensure that the permissions string is in proper format. That is, each of the 9 characters must either be a dash (`-`) or one of the characters `rwx`, in the proper position.

Some examples of valid permissions strings:

```
rwxrwxrwx
---------
rw-r--r--
rwx------
```

Some examples of invalid permissions strings:

```
abcdefghi
xrwxrwxrw (notice it's the right characters in the wrong places)
---rrr---
-
rwxrwxrwxrwxrwxrwxrwxrwxrwx
```

If an invalid permissions string is passed (denoted here as pstring), the following error message should be printed to standard error, and an exit status of EXIT_FAILURE should be returned:

```
Error: Permissions string 'pstring' is invalid.
```

The permissions are to be represented as an integer such that, if read as binary digits, will look as follows:

```
| 00000000000000000000000 | --- | --- | --- |
| Zeros from bits 10-32   | rwx | rwx | rwx |
```

For example, given the permissions string `rw-r--r--`, ignoring the leading zero bits, the binary number will look like: `110100100`.

### Step 3: Recursively navigate directory tree

Get yourself comfortable navigating the directory tree using `opendir`, `readdir`, and `closedir`. Each time you readdir, try calling `stat` on that file, and reading the permissions, the filename, etc. Then, start matching the permissions against the target.

### Step 4: Putting it all together

Your general program flow should be as follows:

1. Initialize the program by checking all arguments are valid.
2. Get the target permissions from the permission string.
3. Start recursing through the directories, printing out files you encounter where the permissions match the target permissions. Print the folder names before you recurse into them.

# Example executions

For these examples, I will be operating on a directory tree with the following format:

```
test_dir
├── subdir1
│   ├── file1
│   └── file2
└── subdir2
    ├── file1
    └── file2
```

The files will have the following permissions:

```
test_dir:
drwxrwxrwx subdir1
drwxr-xr-x subdir2

test_dir/subdir1:
---------- file1
-rw-r--r-- file2

test_dir/subdir2:
-rw-r--r-- file1
-rw-r--r-- file2

$ ./pfind -h
Usage: ./pfind -d <directory> -p <permission string> [-h]

$ ./pfind -v -h
Error: Unknown option '-v' received.

$ ./pfind -d test_dir
Error: Required argument -p <permissions string> not found.

$ ./pfind -p rwxrwxrwx
Error: Required argument -d <directory> not found.

$ ./pfind -d test_dir -p badpermis
Error: Permissions string 'badpermis' is invalid.

$ ./pfind -d invalid_dir -p rwxrwxrwx
Error: Cannot stat 'invalid_dir'. No such file or directory.

$ ./pfind -d invalid_dir -p badpermis
Error: Cannot stat 'invalid_dir'. No such file or directory.

$ ./pfind -d test_dir -p rwxrwxrwx -h
Usage: pfind -d <directory> -p <permission string> [-h]

$ ./pfind -d test_dir -p rw-r--r--
/home/user/test_dir/subdir1/file2
/home/user/test_dir/subdir2/file2
/home/user/test_dir/subdir2/file1

$ ./pfind -d test_dir -p --x--x--x
<no output>
```

If I create a new directory `danger_dir` with permissions `---------` inside my home directory, and try to run pfind on it, it will produce the following output:

```
$ ./pfind -d ~/danger_dir -p --x--x--x
Error: Cannot open directory '/home/user/danger_dir'. Permission denied.
```