

## Ex2: Patches, Libraries, and Makefiles

### Overview

In this exercise you will learn about patches, libraries and Makefiles. You will first modify the patch you created for Project 0. You will then create a static library that will perform a simple arithmetic operation and finally you will create a Makefile to compile that library.

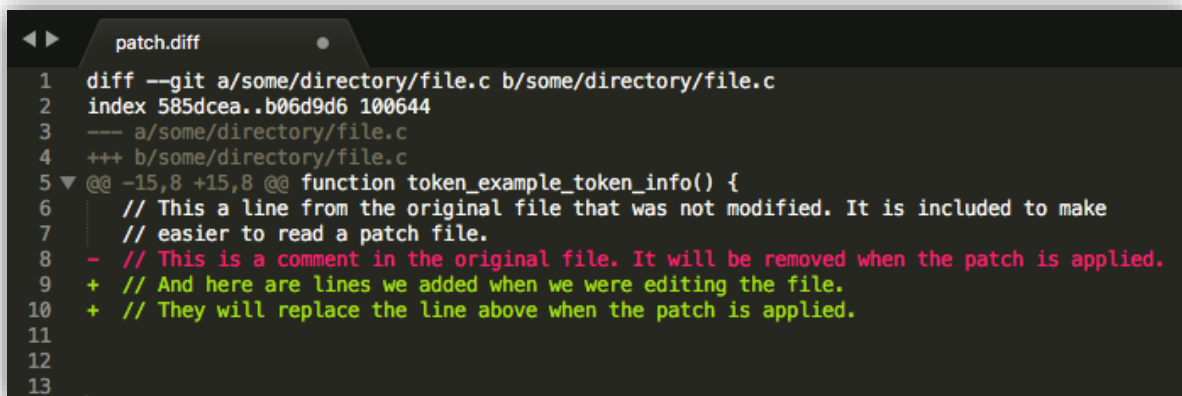
### Structure

The project is broken into three main parts:

- 1) Modify the patch file from P0.
- 2) Create a static library.
- 3) Create a Makefile to re-compile your library.

### Patch Files<sup>1</sup>

A patch file is a text file that contains instructions on how to update other files. In Project 0 you modified a certain file(s) to add your name to the boot message. You then rebuilt the kernel and saw that your changes were applied successfully (hopefully)! Finally, you created a patch that contained all the changes. In other words, the patch contained the *differences* between your modified source and the original clean source you downloaded. Here's an example patch file:



```
1 diff --git a/some/directory/file.c b/some/directory/file.c
2 index 585dcea..b06d9d6 100644
3 --- a/some/directory/file.c
4 +++ b/some/directory/file.c
5 @@ -15,8 +15,8 @@ function token_example_token_info() {
6     // This a line from the original file that was not modified. It is included to make
7     // easier to read a patch file.
8     - // This is a comment in the original file. It will be removed when the patch is applied.
9     + // And here are lines we added when we were editing the file.
10    + // They will replace the line above when the patch is applied.
11
12
13
```

<sup>1</sup> <https://www.drupal.org/node/367392>, [https://en.wikipedia.org/wiki/Patch\\_\(Unix\)](https://en.wikipedia.org/wiki/Patch_(Unix))

The first 5 lines specify which file to modify. The remainder indicates which lines are removed and inserted. When applying this patch file to a clean source of your file system, the patch program will know exactly what to remove and what to add.

Patches are useful because they only keep track of changes. (Imagine if you had to export the 1.4 GB Reptilian image and upload it to Canvas every time you had to submit something!)

For this exercise you will need to do the following:

1. Create a patch file to remove the message you added in P0 and insert a new one that prints “**### First Last Name (Exercise 2) ###**” by modifying your P0 patch.

*NOTE:* You will apply the patch to the kernel source that already contains the changes from P0, so you will need to remove the P0 changes and add the new changes.

2. You will apply the patch by switching to `/usr/rep/src/reptilian-kernel` and running:

```
$ git apply patch_name.diff
$ make && sudo make install && sudo make modules_install
```

3. Take a screenshot of the boot message showing the modified message.

## Libraries<sup>2</sup>

Libraries are simply a collection of previously defined declarations and procedures. If we often use the same subroutines, we can create a library and use it across all our projects. For example, we can use a math library to perform matrix multiplication. You can read about different types of libraries here: [http://www.hep.wisc.edu/~pinghc/generate\\_your\\_own\\_library.htm](http://www.hep.wisc.edu/~pinghc/generate_your_own_library.htm)

In this project you will create a simple static library using the following steps:

1. Create `mean.c` in `/home/reptilian/math`. (You will need to create the `math` directory.)

```
int mean(int a, int b)
{
    return (a + b) / 2;
}
```

2. Create the `mean.h` in `/home/reptilian/math`.

```
#pragma once
int mean(int a, int b);
```

3. Compile `mean.c` and create `libmath.a`.

```
$ cc -c mean.c
$ ar cr libmath.a mean.o
```

---

<sup>2</sup>[https://www.cs.swarthmore.edu/~newhall/unixhelp/howto\\_C\\_libraries.html](https://www.cs.swarthmore.edu/~newhall/unixhelp/howto_C_libraries.html)

4. Create `test.c` in `/home/reptilian`.

```
#include <stdio.h>
#include "math/mean.h"

int main()
{
    printf("The average between 3 and 5 is %d\n", mean(3, 5));
    return 0;
}
```

5. Compile and run `test.c`.

```
$ cc -o test test.c -L ./math -lmath
$ ./test
```

6. Take a screenshot of the output.

## Makefiles

Makefiles are used to organize code compilation. Earlier in this exercise you manually entered the commands to create your library... but imagine if you had to compile this and 50 other programs! In such a circumstance, **Makefiles** come in handy. A Makefile contains all the instructions needed to compile programs and can be run with the `make` command. For this exercise you will create a Makefile in `/home/reptilian/math` that will compile the library.

Read about Makefile here: <http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/>

To test it and make sure it works, delete `libmath.a` and `mean.o`, then run `make`. This should re-create `libmath.a` and `mean.o`. Submit the Makefile on Canvas. (You can add the “.txt” extension if necessary.)

## Submissions

You will submit the following at the end of this exercise:

- Screenshot of the boot message
- The modified patch file
- Screenshot of the library output
- Your Makefile