# EECS 2031 3.0 A Software Tools

**Week 4: September 23, 2018**

# Functions and program structure

- The basic algorithmic unit is a function
  - There are no nested functions in C
- The basic compilation unit is a file (.c)
- Functions can be local to the file in which they are defined (declared static) or are global by default
- Definitions within different files are coordinated using include files (.h)
- Forward definitions of functions is good programming practice and avoids C's default assumption of int values.

# Basic stylistic rule

- Every .c file should have a .h file if the definitions will be used elsewhere
- It should define the signature for the externally visible functions and global variables, as well as any externally visible constants.
- In larger software packages where repeated inclusion is possible, this must be protected against

# Sharing things across files

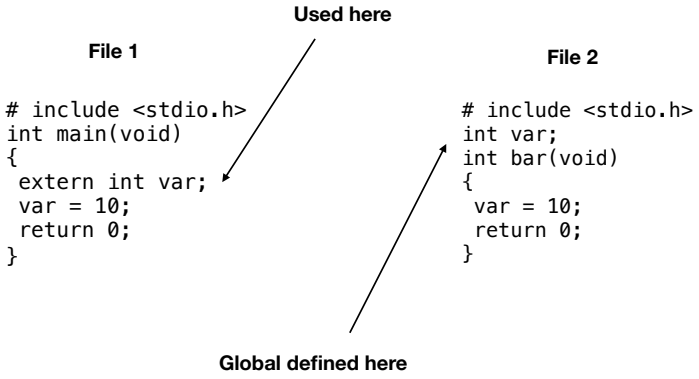
- Functions
  - Use prototypes
  - Put them in .h files
  - Don't declare the functions static
- Variables
  - Use extern for local definitions
  - Define the data storage somewhere

# K&R vs ANSI

- In K&R it was not necessary to provide a prototype for functions. C just made the assumption that the return type was int and when you called the function you passed the right set of arguments.
- This often resulted in really strange errors that were difficult to find.
- Mixing prototyped versus non-prototyped functions is a recipe for disaster. Don't. -Wall is your friend.

# extern keyword

- Not needed for functions, so don't.
- extern on a variable means its a global defined elsewhere
- Someone has to define it

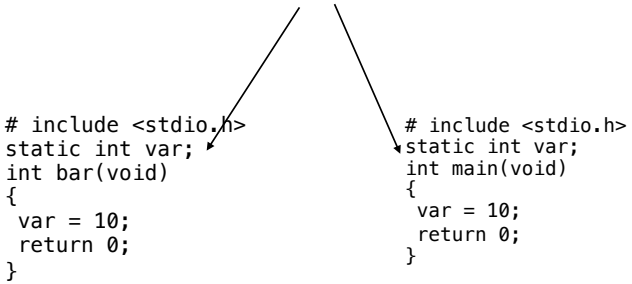



**Sharing variables across files**

# Hiding things in files

- Use static keyword for functions and global (to that file) variables
  - Limits definition to the file
- static has a different meaning when used for variable definitions within a file.
- Remember that all global variables, and all static variables, maintain their state throughout the program's life

**Two completely different var's**



```
# include <stdio.h>
static int var;
int bar(void)
{
 var = 10;
 return 0;
}
```

```
# include <stdio.h>
static int var;
int main(void)
{
 var = 10;
 return 0;
}
```

**Limiting scope to the local file**
**Avoid polluting the name space**

# Questions?

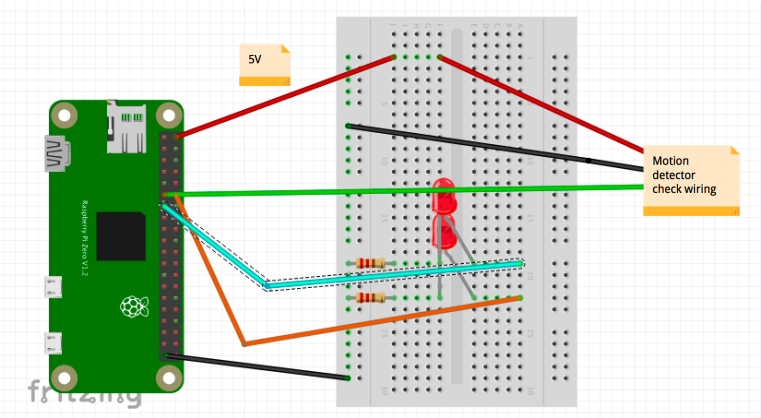
# Lab04

- Goal is to build an intruder alarm (think burglar alarm) that signals the event to some remote service.
  - Next lab you will make this remote service more sophisticated and add a simple (two button) alarm pad.
- This lab also
  - Makes you write more sophisticated C code (although not much)
  - Makes you modify a Makefile
  - Makes you use GitHub as a repository

# Motion detector

- In your kit is a motion detector
- Its a sophisticated piece of electronics that detects changes in IR over a wide field of view.
- It is powered with 5V and requires a ground and a signal output line.
  - Pop off the white cap to see the assignment of pins.
- There are two small potentiometers.
  - Turn first one counter-clockwise, second one clockwise during testing.

# The circuit




# tester.c

```
#include <stdio.h>
#include <wiringPi.h>

int main(int argc, char *argv[])
{
  int i;
  wiringPiSetup () ;
  pinMode(0, INPUT);
  while(1) {
    printf("Waiting for reset\n");
    while(digitalRead(0) == 1);
    printf("Waiting for event\n");
    while(digitalRead(0) == 0);
    printf("Alarm\n");
  }
  /*NOTREACHED*/
  return 0 ;
}
```

# tester.c

```
#include <stdio.h>
#include <wiringPi.h>

int main(int argc, char *argv[])
{
  int i;
  wiringPiSetup () ;
  pinMode(0, INPUT);
  while(1) {
    printf("Waiting for reset\n");
    while(digitalRead(0) == 1);
    printf("Waiting for event\n");
    while(digitalRead(0) == 0);
    printf("Alarm\n");
  }
  /*NOTREACHED*/
  return 0 ;
}
```

**wiringPi (see last week's lab)**

# tester.c

```
#include <stdio.h>
#include <wiringPi.h>

int main(int argc, char *argv[])
{
  int i;
  wiringPiSetup () ;
  pinMode(0, INPUT);
  while(1) {
    printf("Waiting for reset\n");
    while(digitalRead(0) == 1);
    printf("Waiting for event\n");
    while(digitalRead(0) == 0);
    printf("Alarm\n");
  }
  /*NOTREACHED*/
  return 0 ;
}
```

**Sensor returns 0 when it senses nothing**

# tester.c

```
#include <stdio.h>
#include <wiringPi.h>

int main(int argc, char *argv[])
{
  int i;
  wiringPiSetup () ;
  pinMode(0, INPUT);
  while(1) {
    printf("Waiting for reset\n");
    while(digitalRead(0) == 1);
    printf("Waiting for event\n");
    while(digitalRead(0) == 0);
    printf("Alarm\n");
  }
  /*NOTREACHED*/
  return 0 ;
}
```

**Sensor returns 1 when it senses something**

# tester.c

```
#include <stdio.h>
#include <wiringPi.h>

int main(int argc, char *argv[])
{
  int i;
  wiringPiSetup () ;
  pinMode(0, INPUT);
  while(1) {
    printf("Waiting for reset\n");
    while(digitalRead(0) == 1);
    printf("Waiting for event\n");
    while(digitalRead(0) == 0);
    printf("Alarm\n");
  }
  /*NOTREACHED*/
  return 0 ;
}
```

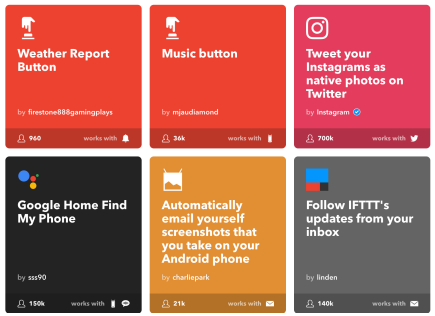
**Good programming practice to warn reader that the code loops forever**

# Letting you know that the alarm has gone off

- If this was to really be useful, you would want to have a monitoring company ‘know’ when the sensor has gone off.
- So lets do that.
- We are working towards using the IFTTT service to do this for real, but this week we will just use a simulation.

# IFTTT

- If <this> then <that> is an IoT services that allows you to have applets that respond to <this> events and then generate <that> effect.




**Here are some**

# IFTTT Webhooks

- To encourage individuals to play with IFTTT the system supports ‘Web Hooks’ (previously known as Maker).

**Triggers**

**Receive a web request**

This trigger fires every time the Maker service receives a web request to notify it of an event. For information on triggering events, go to your Maker service settings and then the listed URL (web) or tap your username (mobile)

Trigger Fields

Event Name

**Actions**

**Make a web request**

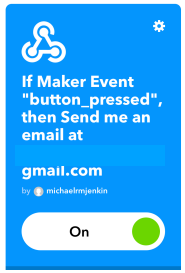
This action will make a web request to a publicly accessible URL. NOTE: Requests may be rate limited.

Action Fields

URL
Method
Content Type
Body

# Webhooks Trigger

- Basically, if your code makes a specific URL request, an IFTTT rule can fire.
- It can do many different things




Your key is: [redacted]

Back to service

**To trigger an Event**

Make a POST or GET web request to:

`https://maker.ifttt.com/trigger/{event}/with/key/[redacted]`

With an optional JSON body of:

```
{ "value1" : "", "value2" : "", "value3" : "" }
```

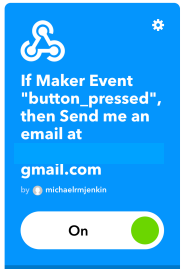
The data is completely optional, and you can also pass `value1`, `value2`, and `value3` as query parameters or form variables. This content will be passed on to the Action in your Recipe.

You can also try it with `curl` from a command line.

```
curl -X POST https://maker.ifttt.com/trigger/{event}/with/key/[redacted]
```

**When you register you get a ‘key’ (blacked out above).**
**Basically, you make a https request and register an event in an applet and you are good to go.**

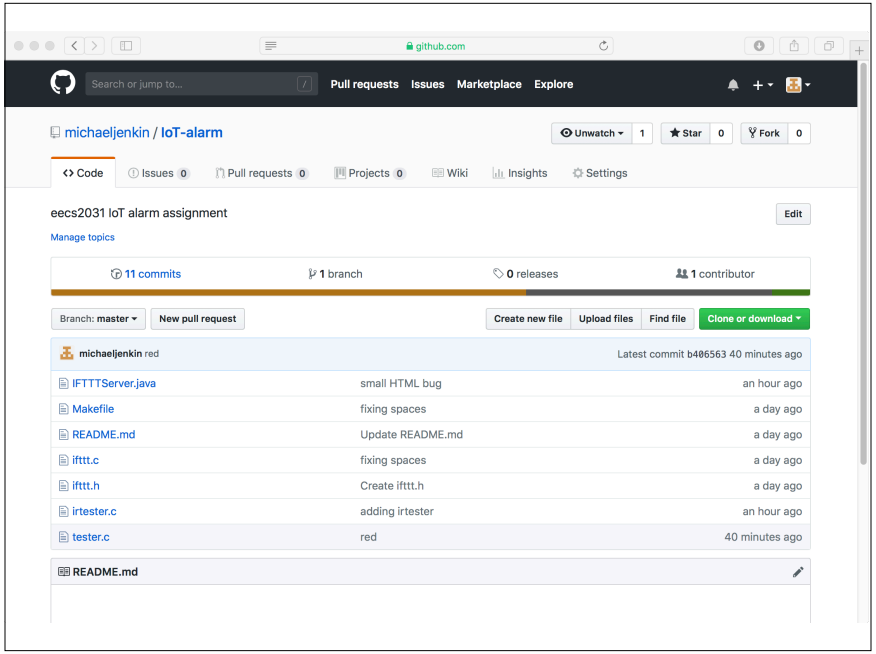
**https://maker.ifttt.com/trigger/button_pressed/with/key/jlkjsdfjakljfdklj**




**You get an email**

# But lets practice first

- I wrote a very simple web server that handles POST requests of this format, and also GET requests
- Code is on the GitHub repository so you can run your own.
- One should be running on red.eecs.yorku.ca on port 8080 for the next few weeks.
  - Note: you cannot run it effectively on your PRISM workstations due to security constraints on those machines.





**Any page on the server -> gets you this**
**But how do you do a POST request from C?**

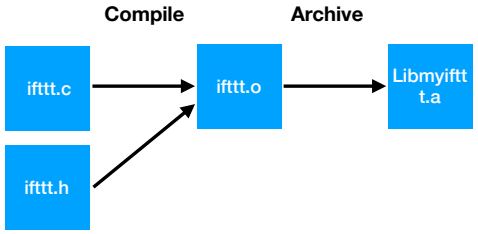
```
#include <stdio.h>
#include <curl/curl.h>
#define BUFFER_MAX 4096


int ifttt(char*where, char *v1, char *v2, char *v3)
{
  CURL *curl;
  CURLcode res;
  struct curl_slist *headers = NULL;
  char sbuf[BUFFER_MAX];

  sprintf(sbuf, "{ \"value1\" : \"%s\", \"value2\" : \"%s\", \"value3\" : \"%s\"}", v1, v2, v3);

  /* In windows, this will init the winsock stuff */
  curl_global_init(CURL_GLOBAL_ALL);

  /* get a curl handle */
  curl = curl_easy_init();
  if(curl) {
    curl_easy_setopt(curl, CURLOPT_URL, where);

    curl_easy_setopt(curl, CURLOPT_POSTFIELDS, sbuf);
    headers = curl_slist_append(headers,"Content-Type: application/json");
    curl_easy_setopt(curl, CURLOPT_HTTPHEADER, headers);
    curl_easy_setopt(curl, CURLOPT_WRITEDATA, fopen("/dev/null", "w"));

    /* Perform the request, res will get the return code */
    res = curl_easy_perform(curl);
    /* Check for errors */
    if(res != CURLE_OK)
      fprintf(stderr, "curl_easy_perform() failed: %s\n", curl_easy_strerror(res));

    curl_easy_cleanup(curl);
    curl_global_cleanup();
    return(res == CURLE_OK);
  }
  curl_global_cleanup();
  return 0;
}
```

```
#include <stdio.h>
#include "ifttt.h"

int main(int argc, char *argv[])
{
/*
  ifttt("https://maker.ifttt.com/trigger/button_pressed/with/key/56-YpOKO17vOh-gimC2xK1qRAhRdzXXXX", "my1", "my 2", "my 33333");
*/

  printf("Trying to connect to server\n");
  ifttt("http://red.eecs.yorku.ca:8080/trigger/event/with/key/123", "my1", "my 2", "my 33333");
  return 0;
}
```

**tester.c**

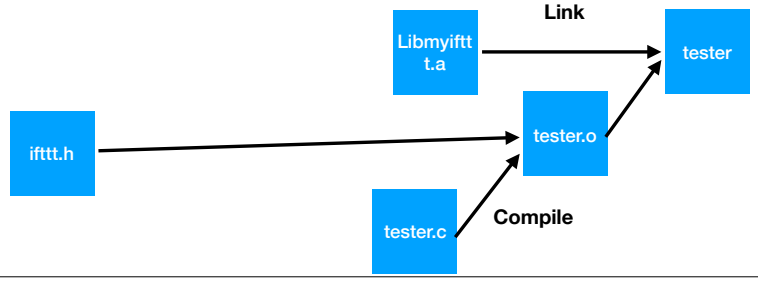
# Libraries

- UNIX supports both dynamic and static libraries
  - Dynamic libraries are linked in at run time, static ones at compile time. We will use a static one here.



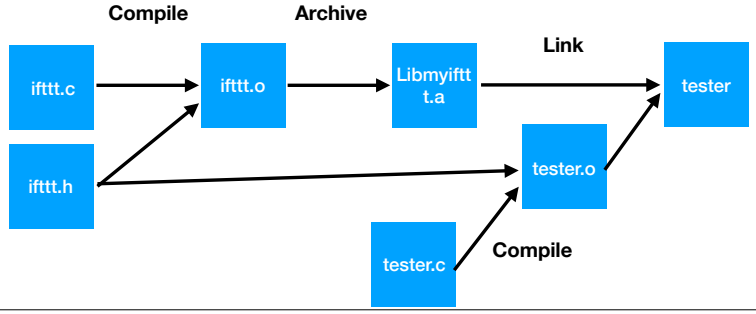

# Libraries

- UNIX supports both dynamic and static libraries
  - Dynamic libraries are linked in at run time, static ones at compile time. We will use a static one here.




# Libraries

- Lets you build a repository of useful and re-usable code into a single module.
- Contents controlled by header files.



```
CC=gcc
CFLAGS=-lWarn -pedantic

tester: tester.o libmyifttt.a
   $(CC) tester.o -L. -lmyifttt -lcurl -o tester

libmyifttt.a:ifttt.o
   ar -rcs libmyifttt.a ifttt.o

ifttt.o:  ifttt.c ifttt.h
   $(CC) $(CFLAGS) -c -ansi $<

tester.o: tester.c ifttt.h
   $(CC) $(CFLAGS) -c -ansi $<

clean:
   rm tester *.o
```

**Making the library libmyifttt.a**

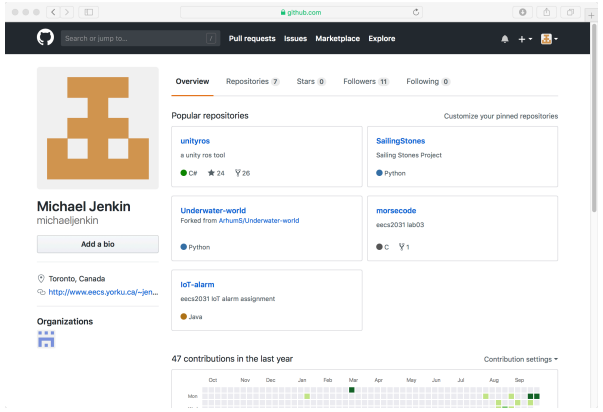
```
CC=gcc
CFLAGS=-lWarn -pedantic

tester: tester.o libmyifttt.a
   $(CC) tester.o -L. -lmyifttt -lcurl -o tester

libmyifttt.a:ifttt.o
   ar -rcs libmyifttt.a ifttt.o

ifttt.o:  ifttt.c ifttt.h
   $(CC) $(CFLAGS) -c -ansi $<

tester.o: tester.c ifttt.h
   $(CC) $(CFLAGS) -c -ansi $<

clean:
   rm tester *.o
```

**Using the library libmyifttt.a**

# Makefile

- You will have to add your own rules to the Makefile for this Lab.
- Learn to use and make (at least) simple Makefiles

# GitHub




**Sign up for an account**

# GitHub

- Create a new repository (call it eecs2031lab04 or something equally as obvious).
- When you create it, make a ‘README.md’
- Then clone it (as you did in lab02)
  - Get the URL for the repository
  - Go to your pi and type
    - git clone https:/github.com/whateveryougot
  - Change directory there

# GitHub commands

- There are many (many x many).
- GitHub maintains multiple development branches
  - We will just use one for now ‘master’
- For simplicity now, lets assume that you do all your work on the Raspberry Pi

# Basic Operation

**Suppose I had just edited tester.c**

```
pi@mypi:~/Documents/lab04/IoT-alarm$ git add tester.c
pi@mypi:~/Documents/lab04/IoT-alarm$ git commit -m "just cause"
[master 5cea852] just cause
 1 file changed, 1 insertion(+), 1 deletion(-)
pi@mypi:~/Documents/lab04/IoT-alarm$ git push
Username for 'https://github.com': michaeljenkin
Password for 'https://michaeljenkin@github.com':
Counting objects: 3, done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 285 bytes | 0 bytes/s, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/michaeljenkin/IoT-alarm.git
   767adf1..5cea852  master -> master
```

# Basic Operation

**Add the current version of tester.c to the index of files to be added to the next software version**

```
pi@mypi:~/Documents/lab04/IoT-alarm$ git add tester.c
pi@mypi:~/Documents/lab04/IoT-alarm$ git commit -m "just cause"
[master 5cea852] just cause
 1 file changed, 1 insertion(+), 1 deletion(-)
pi@mypi:~/Documents/lab04/IoT-alarm$ git push
Username for 'https://github.com': michaeljenkin
Password for 'https://michaeljenkin@github.com':
Counting objects: 3, done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 285 bytes | 0 bytes/s, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/michaeljenkin/IoT-alarm.git
   767adf1..5cea852  master -> master
```

# Basic Operation

**Record all the changes (add, deletes, others) to the repository**
**-m “why did you do this”**

```
pi@mypi:~/Documents/lab04/IoT-alarm$ git add tester.c
pi@mypi:~/Documents/lab04/IoT-alarm$ git commit -m "just cause"
[master 5cea852] just cause
 1 file changed, 1 insertion(+), 1 deletion(-)
pi@mypi:~/Documents/lab04/IoT-alarm$ git push
Username for 'https://github.com': michaeljenkin
Password for 'https://michaeljenkin@github.com':
Counting objects: 3, done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 285 bytes | 0 bytes/s, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/michaeljenkin/IoT-alarm.git
   767adf1..5cea852  master -> master
```

# Basic Operation

**We are changing one file, which has some very minor change**

```
pi@mypi:~/Documents/lab04/IoT-alarm$ git add tester.c
pi@mypi:~/Documents/lab04/IoT-alarm$ git commit -m "just cause"
[master 5cea852] just cause
 1 file changed, 1 insertion(+), 1 deletion(-)
pi@mypi:~/Documents/lab04/IoT-alarm$ git push
Username for 'https://github.com': michaeljenkin
Password for 'https://michaeljenkin@github.com':
Counting objects: 3, done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 285 bytes | 0 bytes/s, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/michaeljenkin/IoT-alarm.git
   767adf1..5cea852  master -> master
```

# Basic Operation

**Push the changes to GitHub so that they are stored remotely**

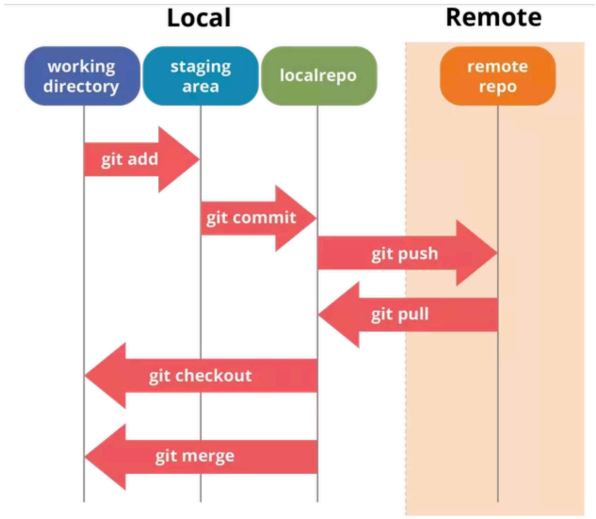
```
pi@mypi:~/Documents/lab04/IoT-alarm$ git add tester.c
pi@mypi:~/Documents/lab04/IoT-alarm$ git commit -m "just cause"
[master 5cea852] just cause
 1 file changed, 1 insertion(+), 1 deletion(-)
pi@mypi:~/Documents/lab04/IoT-alarm$ git push
Username for 'https://github.com': michaeljenkin
Password for 'https://michaeljenkin@github.com':
Counting objects: 3, done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 285 bytes | 0 bytes/s, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/michaeljenkin/IoT-alarm.git
   767adf1..5cea852  master -> master
```

# Basic Operation

**Push the changes to GitHub so that they are stored remotely**

```
pi@mypi:~/Documents/lab04/IoT-alarm$ git add tester.c
pi@mypi:~/Documents/lab04/IoT-alarm$ git commit -m "just cause"
[master 5cea852] just cause
 1 file changed, 1 insertion(+), 1 deletion(-)
pi@mypi:~/Documents/lab04/IoT-alarm$ git push
Username for 'https://github.com': michaeljenkin
Password for 'https://michaeljenkin@github.com':
Counting objects: 3, done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 285 bytes | 0 bytes/s, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/michaeljenkin/IoT-alarm.git
   767adf1..5cea852  master -> master
```

**So here there is only one branch (the master)**




# Summary

- Functions and variables defined in files
- Keyword static can be used to limit scope to the file.
  - Note: static has another meaning when used for local variables.
- Non static global structures should be coordinated through include files
- Use of ‘extern’ for global variables shared across files.
- Functions cannot define inner functions, but can be recursive.

# Makefile

- Automate the build process
- Macros and rules
- Can be used to build (effectively) arbitrarily complex things
  - Make ‘all’, and ‘clean’ are very common

# Git (and GitHub)

- Version control
- Remote storage of software (anything) with version control and mechanisms to combine different versions.

**Setup and use your own GitHub repository now.**
**(Who is backing up your Raspberry PI?)**