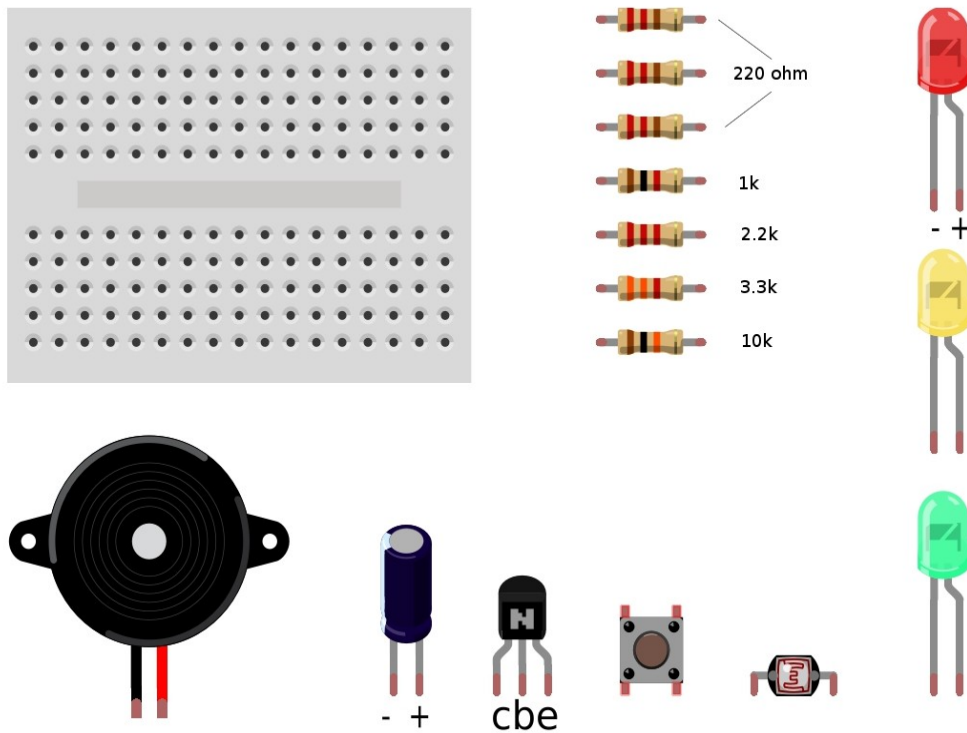


Electronics Kit for Raspberry Pi

Projects

1. Flash a red LED on & off
2. Get input from a push button switch
3. Use a push button switch to turn a red LED on and off
4. Traffic lights sequence with red, yellow and green LEDs
5. Turn the buzzer on and off using GPIO (3.3 volts)
6. Use a transistor to make the buzzer louder (5 volts)
7. Get a reading from the light dependent resistor (LDR)
8. Make a drawer/cupboard alarm that emails your phone when triggered.

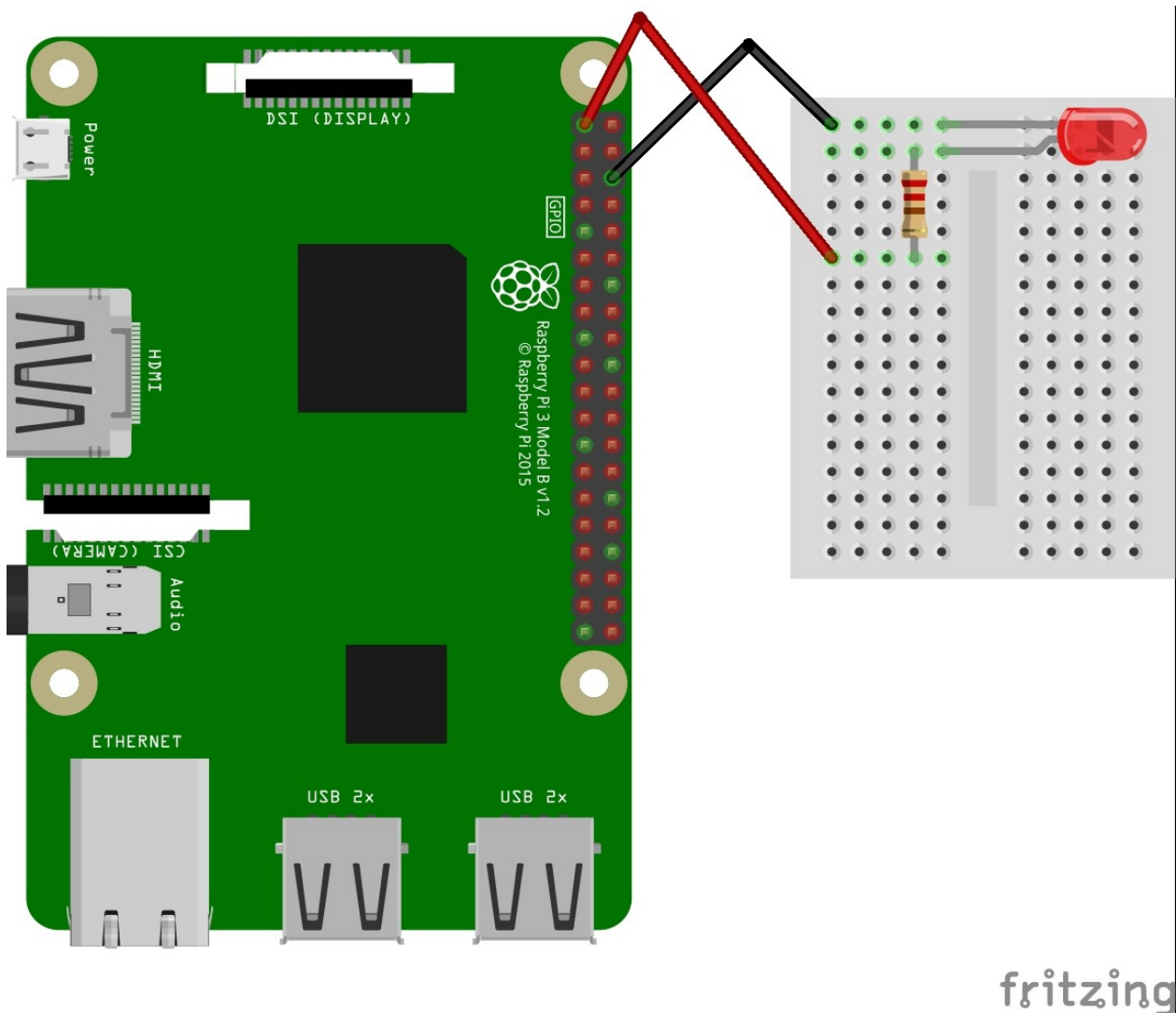


Parts (anti-clockwise)

1. Breadboard
2. Buzzer 5 volt
3. Capacitor 100uF
4. Transistor 2N3904 NPN (collector, base and emitter marked above)
5. Push button switch
6. Light dependent resistor (aka LDR or light sensor)
7. Green, Yellow and Red LEDs (light emitting diodes)
8. Fixed value resistors x7 (values shown above)
9. Breadboard cables male to female x9
10. GPIO pin number reference card

Test the LEDs

Connect up the circuit shown below and the red LED should light up. Make sure the flat side of the LED is connected to the black GND lead, and the longer lead is connected to the current-limiting 220 ohm resistor (red, red, brown). The red wire connects to +3.3 volt on the Pi & the black wire connects to - GND ground.



When you're happy the red LED is working correctly, you can also test the yellow & green LEDs.

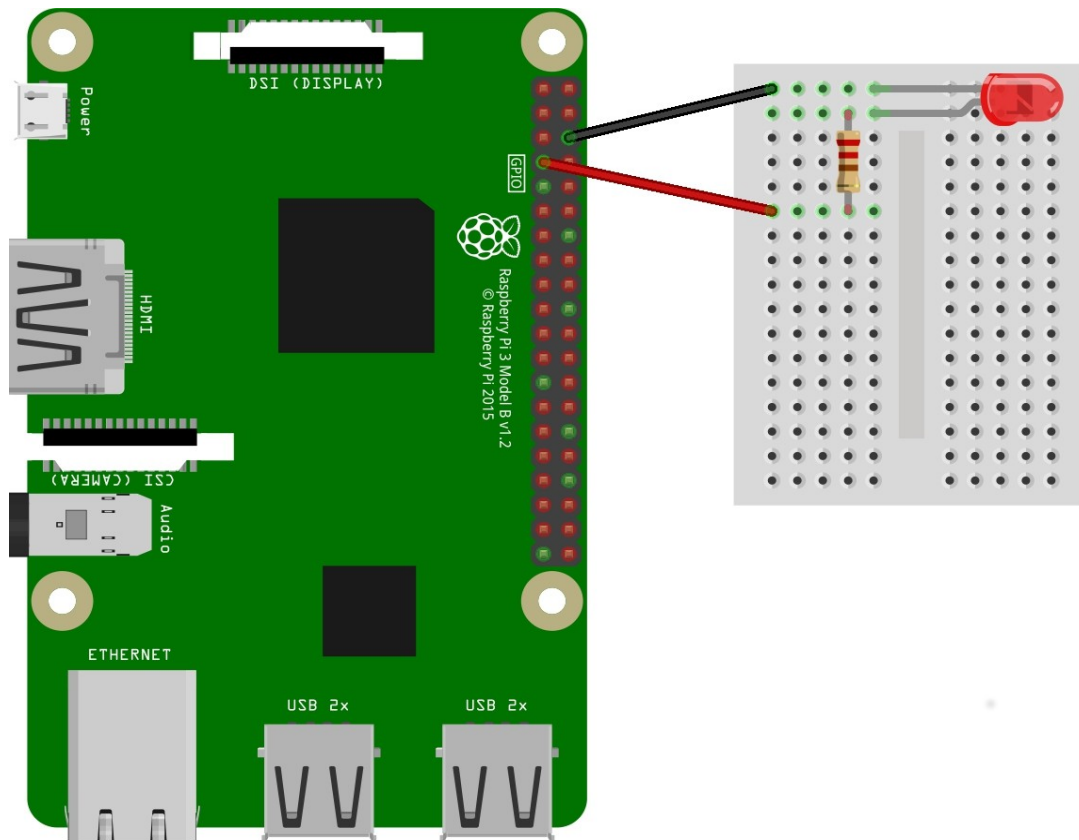
With this circuit we can't use the PI to turn the LED on and off, but if we attach the red wire to a GPIO pin instead, we can then turn the LED on and off under software control on the Pi.

You can download & prepare the scripts used in this manual from our website with

```
wget http://www.securipi.co.uk/kit1.zip
unzip kit1.zip
chmod a+x *.sh
ls
```

Flash the red LED

Wire up the circuit shown below, using the 220 ohm resistor (red, red, brown) and an LED.



fritzing

The red wire is now connected to GPIO pin 4, and we can turn it on and off using software.

On the Pi, open a terminal window and type in

```
nano led1.sh
```

Then enter the following

```
#!/bin/sh
led=4
echo $led > /sys/class/gpio/export
echo "out" > /sys/class/gpio/gpio$led/direction
clear
while true; do
    trap 'echo $led > /sys/class/gpio/unexport' 0
    echo "1" > /sys/class/gpio/gpio$led/value
    sleep 1
    echo "0" > /sys/class/gpio/gpio$led/value
    sleep 1
done
exit 0
```

do Ctrl-O to save the file and then Ctrl-X to exit.

Then type in

```
chmod a+x led1.sh
```

Finally run the program with

```
sudo ./led1.sh
```

If everything is working correctly, the LED should be flashing on & off once per second.

You can quit the program by pressing Ctrl-C on the keyboard.

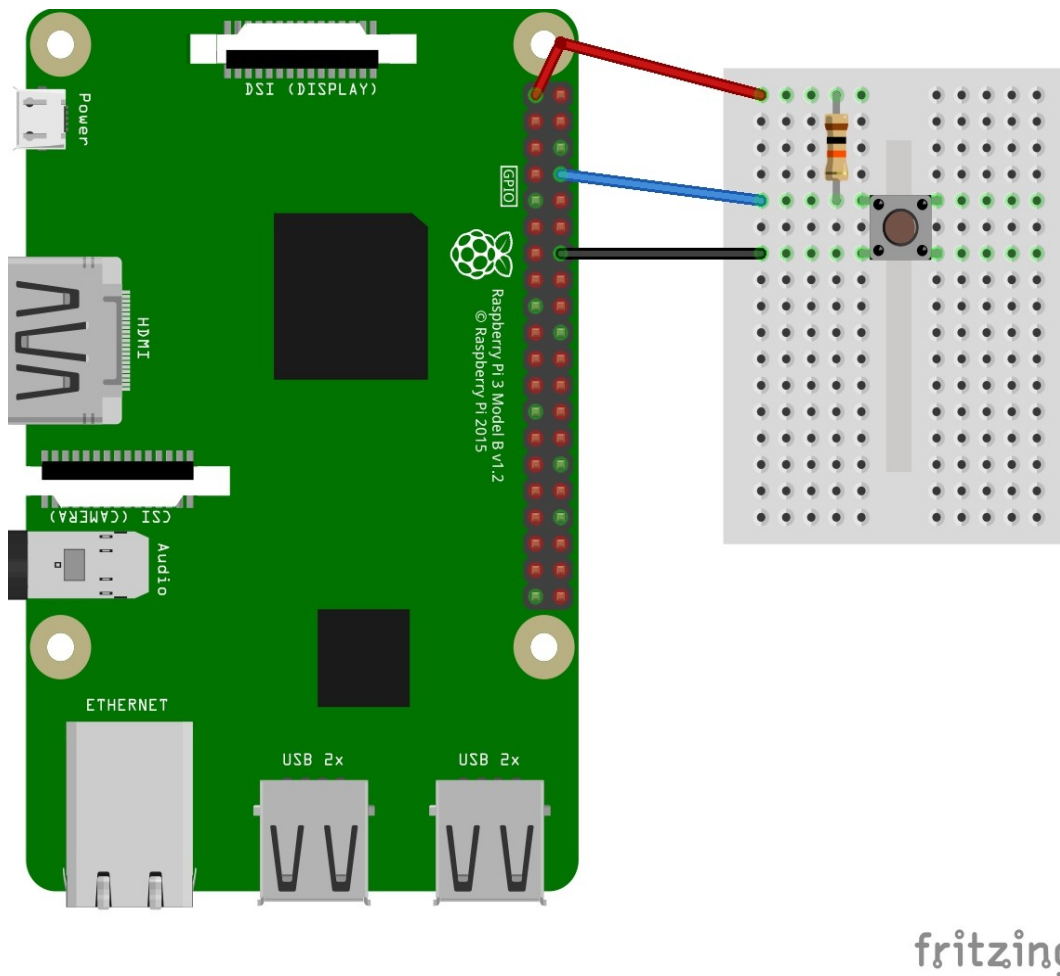
See if you can edit the script so that the LED flashes faster by changing the sleep 1 lines to sleep .5 Then try and make the LED only flash every two seconds.

The program first sets up GPIO pin 4 as an output (rather than an input), and then goes into a continuous loop, until you press Ctrl-C on the keyboard, at which point it safely releases GPIO4 using the trap line. The LED gets turned on by echoing 1 to the GPIO device and turned off by echoing 0 to the GPIO device.

We use the variable led=4 to select GPIO 4, but we could easily change it to a different GPIO pin number and also change the GPIO pin the red wire is attached to. Try that now, there are a full list of GPIO pins on the card that came with the pack.

Get input from a push button switch

Wire up the circuit shown below.



fritzing

On the Pi, open a terminal window and type in

```
nano button1.sh
```

Then enter the following

```
#!/bin/sh
button=14
echo $button > /sys/class/gpio/export
echo "in" > /sys/class/gpio/gpio$button/direction
clear
while true; do
    trap 'echo $button > /sys/class/gpio/unexport' 0
    state=`cat /sys/class/gpio/gpio$button/value`
    echo $state
done
exit 0
```

do Ctrl-O to save the file and then Ctrl-X to exit.

Then type in

```
chmod a+x button1.sh
```

Finally run the program with

```
sudo ./button1.sh
```

You should see 1's scrolling up the screen, and when you press the button you should see 0's instead. You can press Ctrl-C to quit the program.

Let's make the button do something useful. If you have a camera attached to your Pi you can make the button take a picture and say button pressed.

```
nano button2.sh
```

```
#!/bin/sh
button=14
echo $button > /sys/class/gpio/export
echo "in" > /sys/class/gpio/gpio$button/direction
clear
while true; do
    trap 'echo $button > /sys/class/gpio/unexport' 0
    state=`cat /sys/class/gpio/gpio$button/value`
    if [ $state = "0" ]
    then
        echo "You pressed the button"
        sleep .5
    fi
done
exit 0
```

```
chmod a+x button2.sh
```

Finally run the program with

```
sudo ./button2.sh
```

Quit the program with Ctrl-C. Under the line

```
echo "You pressed the button"
```

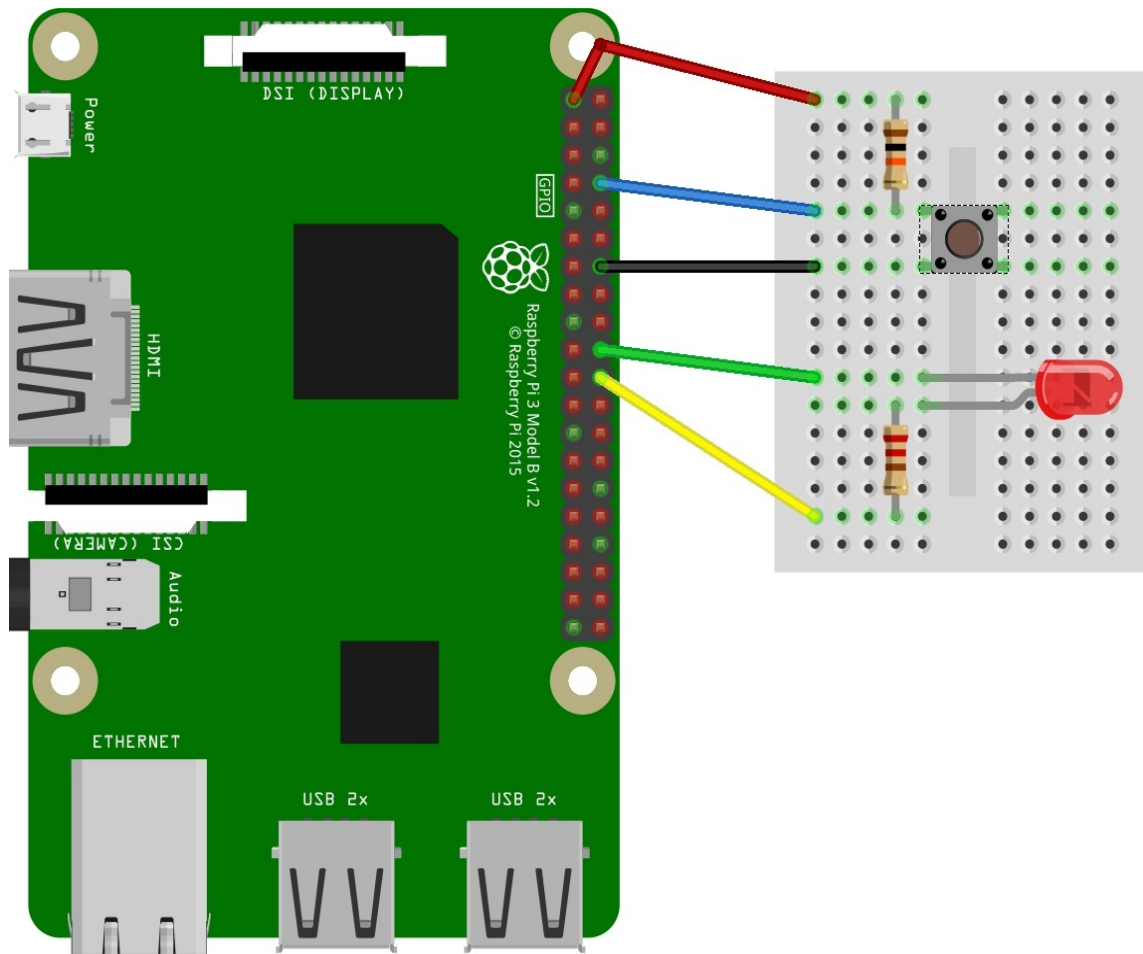
add

```
raspistill -o photo.jpg
```

to make the program take a photo from your Raspberry Pi camera.

Push button and an LED together

Wire the breadboard up like this:



fritzing

The shell script on the next page waits until the button is pressed, reads the current state of the LED, and flips it to the opposite state.

This time we're using GPIO pin 25 and a GND pin for the LED. (all the GND pins are highlighted in green) The button circuit is the same as the previous example.

First enter

```
nano button3.sh
```

and then enter the script on the following page:

```

#!/bin/sh

led=25
echo $led > /sys/class/gpio/export
echo "out" > /sys/class/gpio/gpio$led/direction

button=14
echo $button > /sys/class/gpio/export
echo "in" > /sys/class/gpio/gpio$button/direction
clear

while true; do
    #trap command should be all on one line!
    trap 'echo $led > /sys/class/gpio/unexport && echo $button
> /sys/class/gpio/unexport' 0

    buttonstate=`cat /sys/class/gpio/gpio$button/value`
    if [ $buttonstate = "0" ]
    then
        ledstate=`cat /sys/class/gpio/gpio$led/value`

        if [ $ledstate = "0" ]
        then
            echo "1" > /sys/class/gpio/gpio$led/value
            echo "turning LED on"
            sleep .5

            elif [ $ledstate = "1" ]
            then
                echo "0" > /sys/class/gpio/gpio$led/value
                echo "turning LED off"
                sleep .5

            fi
        fi
    fi

done
exit 0

```

Save the file with Ctrl-O and then Ctrl-X. Make the file executable with

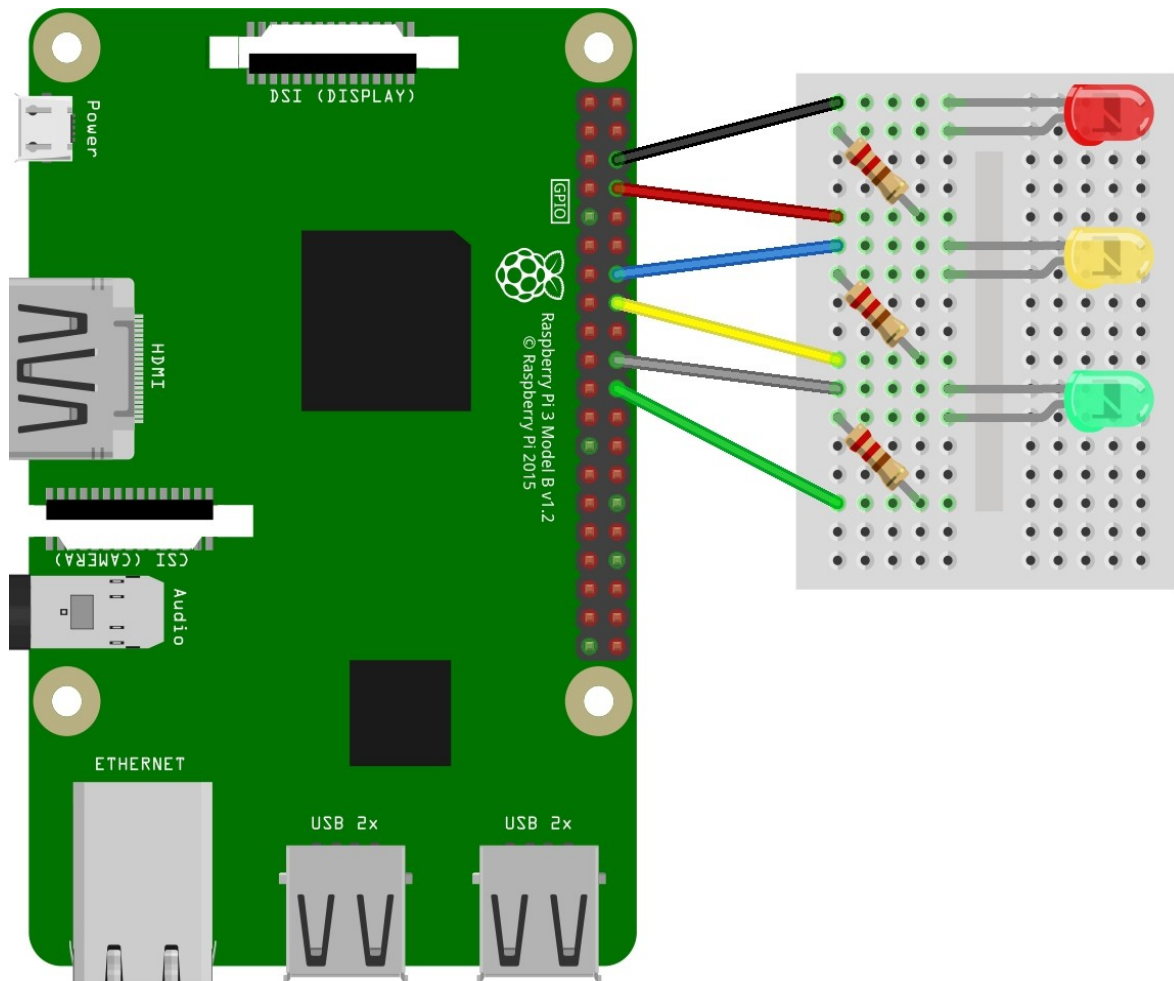
```
chmod a+x button3.sh
```

run the program with

```
sudo ./button3.sh
```


Use all 3 LEDs to make traffic lights.

Wire up the breadboard as shown below



fritzing

The shell script on the next page sets up GPIO pins 14,23 &25 as outputs, and then turns each one on and off in sequence.

First enter

```
nano led2.sh
```

and then enter the script on the following page

```

#/bin/sh
red=14
echo $red > /sys/class/gpio/export
echo "out" > /sys/class/gpio/gpio$red/direction

yellow=23
echo $yellow > /sys/class/gpio/export
echo "out" > /sys/class/gpio/gpio$yellow/direction

green=25
echo $green > /sys/class/gpio/export
echo "out" > /sys/class/gpio/gpio$green/direction
clear

while true; do
    #trap command should be all on one line
    trap 'echo $red > /sys/class/gpio/unexport && echo $yellow
> /sys/class/gpio/unexport && echo $green >
/sys/class/gpio/unexport' 0

    echo "1" > /sys/class/gpio/gpio$red/value
    echo "Red"
    sleep 1
    echo "0" > /sys/class/gpio/gpio$red/value
    echo "1" > /sys/class/gpio/gpio$yellow/value
    echo "Yellow"
    sleep 1
    echo "0" > /sys/class/gpio/gpio$yellow/value
    echo "1" > /sys/class/gpio/gpio$green/value
    echo "Green\n"
    sleep 1
    echo "0" > /sys/class/gpio/gpio$green/value

done
exit 0

```

then save the file with Ctrl-O and Ctrl-X

make the script executable with

```
chmod a+x led2.sh
```

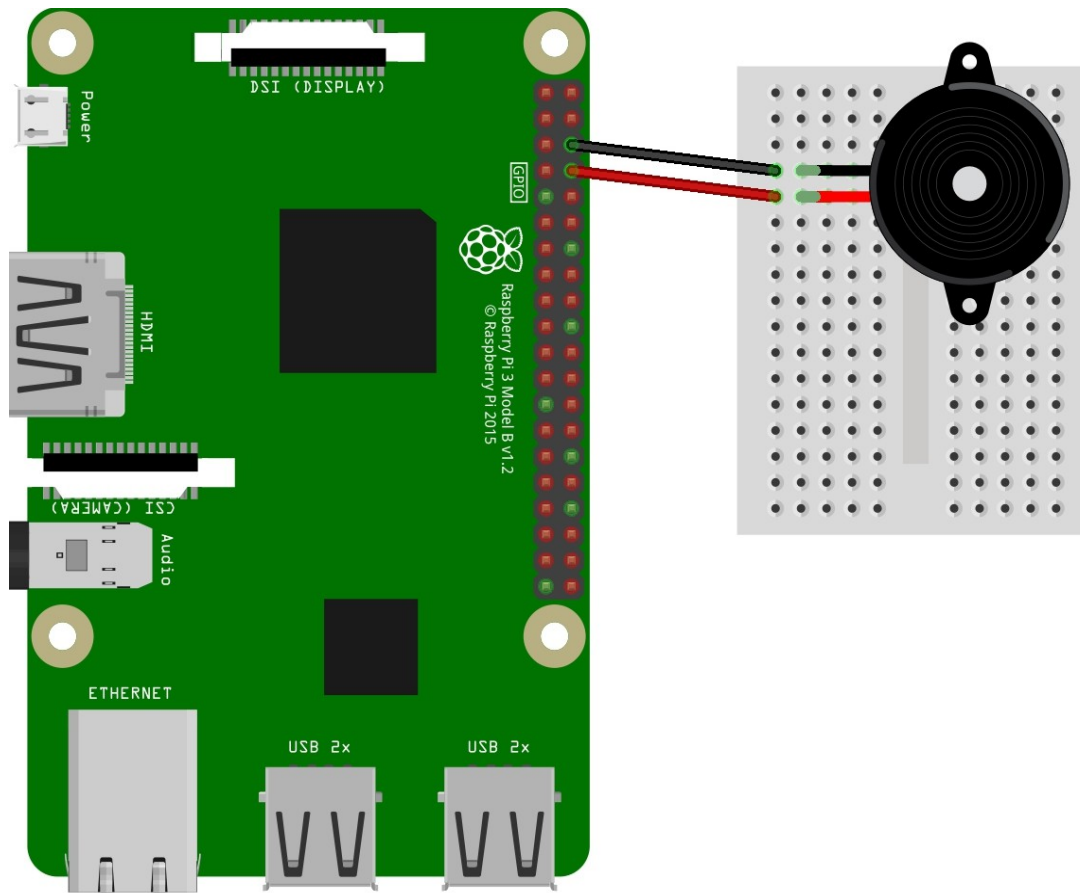
and run the script with

```
sudo ./led2.sh
```

Make the buzzer sound

Wire up the breadboard as shown below.

The buzzer has a positive + and negative – side. The longer pin is positive + red. Also note the spacing of the pins, when inserted into the breadboard, is slightly different from the diagram.



fritzing

```
nano buzzer1.sh
```

```
#!/bin/sh
buzzer=14
echo $buzzer > /sys/class/gpio/export
echo "out" > /sys/class/gpio/gpio$buzzer/direction
clear
    echo "1" > /sys/class/gpio/gpio$buzzer/value
    sleep 1
    echo "0" > /sys/class/gpio/gpio$buzzer/value
    echo $buzzer > /sys/class/gpio/unexport
exit 0
```

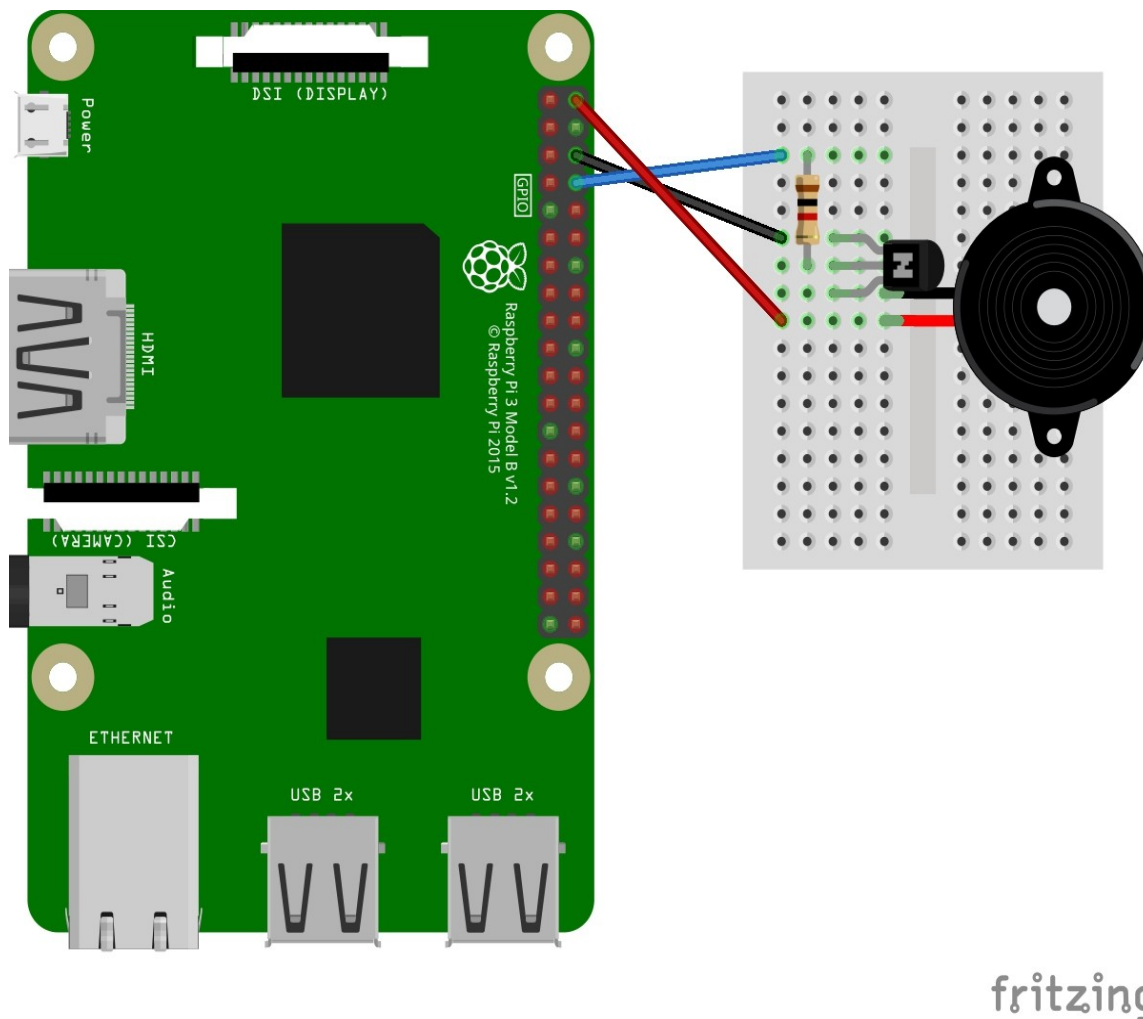
```
chmod a+x buzzer1.sh
```

then run the script with

```
sudo ./buzzer1.sh
```

The buzzer is connected to a 3.3volt GPIO pin, and does make a noise, but in reality the buzzer is a 5 volt device. In the next example we use a transistor as a switch, so when the GPIO is triggered it turns on the 5 volt supply to the buzzer.

Wire up the breadboard like this



Now run the original script again and the beep produced will be a bit more forceful.

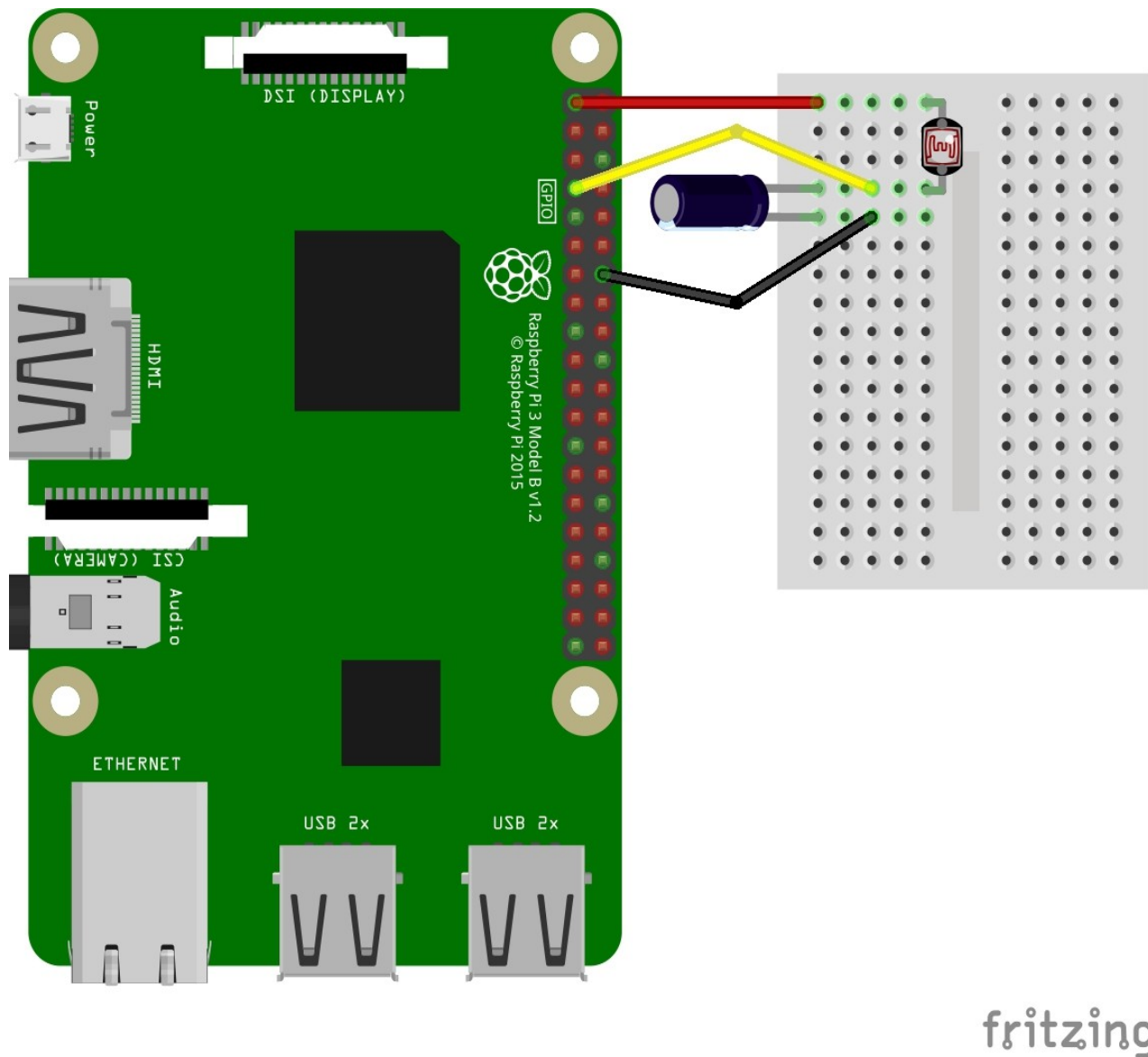
```
sudo ./buzzer1.sh
```

If you instead used a beefier Mosfet transistor, like the TIP120, you could use the GPIO pin to switch on 12 volts of power to an LED lighting panel. (look up our illumipi kit on Google if you are interested in that)

Light dependent resistor

Wire up the LDR and capacitor on the breadboard as shown below. Note the capacitor has to be inserted the right way around – longer leg to yellow wire and “-” short leg to black wire.

The light dependent resistor will vary the resistance according to how much light is falling on it. We can read that value by measuring how long the capacitor takes to fill up (like a bucket of water).



fritzing

The script on the following page will give values of around 23 in good light, 100 in shadow and 200 if you cover the sensor.

```
nano ldr1.sh
```

```
#!/bin/sh

ldr=4
counter=0
echo $ldr > /sys/class/gpio/export
echo "in" > /sys/class/gpio/gpio$ldr/direction
clear
while true; do
    trap 'echo $ldr > /sys/class/gpio/unexport' 0

    echo "out" > /sys/class/gpio/gpio$ldr/direction
    echo "0" > /sys/class/gpio/gpio$ldr/value
    sleep .01
    echo "in" > /sys/class/gpio/gpio$ldr/direction
    state=`cat /sys/class/gpio/gpio$ldr/value`
    while [ $state = "0" ]
        do
            counter=$((counter + 1))
            state=`cat /sys/class/gpio/gpio$ldr/value`
        done
    echo $counter
    counter=0
done
exit 0
```

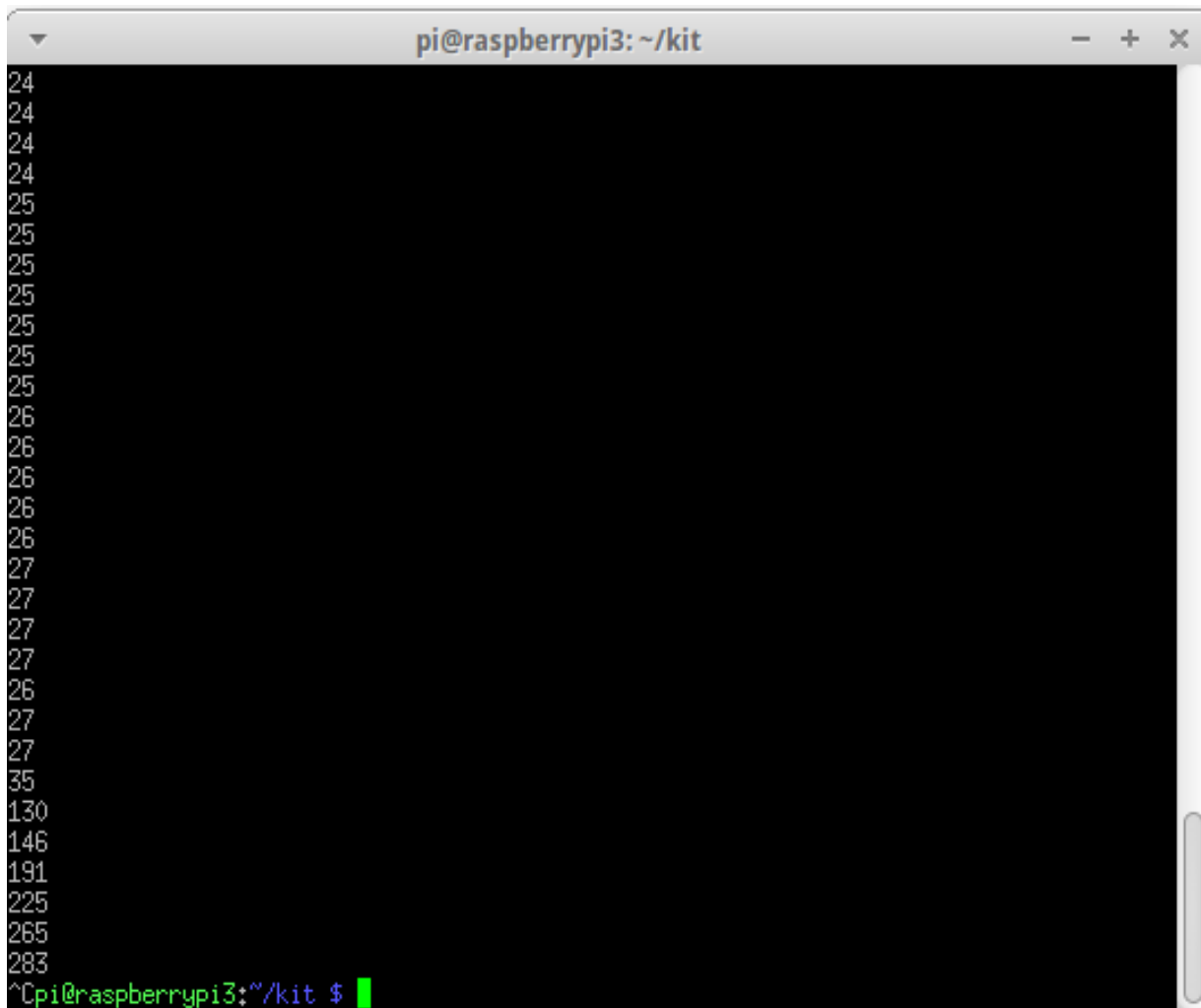
Then save the file with Ctrl-O and Ctrl-X

Make the file executable with

```
chmod a+x ldr1.sh
```

and run the script with

```
sudo ./ldr1.sh
```

A terminal window titled 'pi@raspberrypi3: ~/kit' with standard window controls. The terminal has a black background and displays a list of numbers in white text. The numbers are: 24, 24, 24, 24, 25, 25, 25, 25, 25, 25, 25, 26, 26, 26, 26, 26, 27, 27, 27, 27, 26, 27, 27, 35, 130, 146, 191, 225, 265, 283. At the bottom, the prompt '^Cpi@raspberrypi3:~/kit \$' is shown with a green cursor.

```
pi@raspberrypi3: ~/kit
24
24
24
24
25
25
25
25
25
25
25
26
26
26
26
26
27
27
27
27
26
27
27
35
130
146
191
225
265
283
^Cpi@raspberrypi3:~/kit $
```

We can use this LDR experiment as the basis for a drawer or cupboard alarm, and have the Raspberry Pi email us when the light level changes from darkness (200+) to daylight (less than 50).

You'll need to have your Raspberry Pi connected to the internet & also have a spare Gmail account just for the Pi to use to send emails to the regular email address on your phone or PC.

Setting up a spare Gmail account, just for your Pi to use.

The simplest way of sending emails & photos as attachments from your Pi, is to setup a new Gmail account for the Pi to use, even if you already have an existing Gmail account you use on your phone or PC. For one thing, it gets you 15GB of new cloud storage for your alert photos & secondly it removes the complication of generating application specific passwords for other apps on your existing Gmail account.

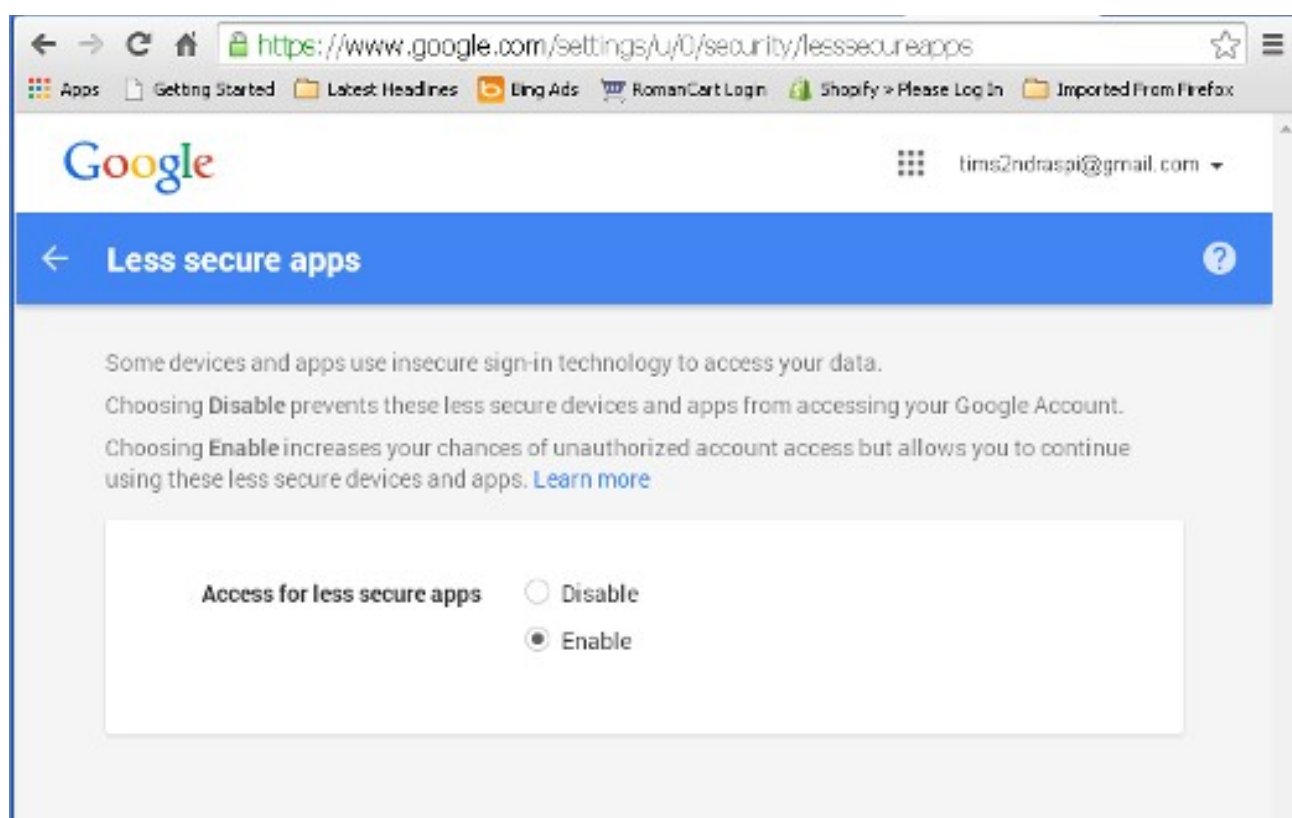
You need to create the new Gmail account in the web browser on your PC or Mac @

<https://accounts.google.com/SignUp?service=mail>

and note down the login & password for later. Don't use the # symbol in your password as it causes the Pi problems. Lower & upper-case letters & number combinations are always fine though.

Next, you need to set the new Gmail account to Enable “less secure apps”. While logged in go to:

<https://www.google.com/settings/u/0/security/lesssecureapps>



Now we have a working email account, just for the Pi to use when sending emails. Any photos sent from the Pi will be backed-up in the Sent folder & you only need delete old photos if you get near to the 15GB limit. Emails from the Pi can be sent to any other email address on your phone or PC

Once you have the extra Gmail account setup, do this:

```
sudo apt-get install ssmtp mailutils mpack
```

Setup default settings for SSMTP.

```
sudo nano /etc/ssmtp/ssmtp.conf
```

and add these lines to the end of the file

```
AuthUser=your-pi-gmail-account@gmail.com
AuthPass=your-user-password
FromLineOverride=YES
mailhub=smtp.gmail.com:587
UseSTARTTLS=YES
```

save file & exit.

Send a test email with this command, substituting the email address below with your own.

```
echo "testing 1 2 3" | mail -s "Subject" you@yourdomain.co.uk
```

Assuming the test came through okay, we can now modify the earlier script so it sends an email when the light level increases. Remember to change you@yourdomain.co.uk

```
#!/bin/sh

ldr=4
counter=0
echo $ldr > /sys/class/gpio/export
echo "in" > /sys/class/gpio/gpio$ldr/direction
clear
while true; do
    trap 'echo $ldr > /sys/class/gpio/unexport' 0

    echo "out" > /sys/class/gpio/gpio$ldr/direction
    echo "0" > /sys/class/gpio/gpio$ldr/value
    sleep .01
    echo "in" > /sys/class/gpio/gpio$ldr/direction
    state=`cat /sys/class/gpio/gpio$ldr/value`
    while [ $state = "0" ]
    do
        counter=$((counter + 1))
        state=`cat /sys/class/gpio/gpio$ldr/value`
    done
    echo $counter
    if [ $counter -lt "30" ]
    then
        echo "Daylight!"
        echo "Daylight" | mail -s "Subject" you@yourdomain.co.uk
    fi
    counter=0
done
exit 0
```

