

## Raspberry Pi GPIO Electronics Starter Kit

Introduction 1. First Time Configuration .....	3
Overview .....	3
Using the Whole SD Card.....	3
Using the Whole Screen.....	5
Changing Timezone .....	6
Bootting into Desktop .....	7
Other Options .....	8
Running raspi-config After Booting.....	9
Test & Configure.....	10
Introduction 2. Using SSH .....	11
Overview .....	11
Enabling SSH.....	11
Using SSH on a Mac or Linux .....	13
SSH under Windows .....	14
Test & Configure.....	15
Troubleshooting .....	16
Introduction 3 -- Pinout! For the Revision 2.0 Raspberry Pi .....	17
GPIO Setup .....	19
The GPIO Connector.....	20
Raspberry Pi Code.....	23
Configuring GPIO .....	25
Configuring I2C .....	26
Lesson 1 - FLASHING LED.....	30
Overview .....	30
Wire LED : .....	30
Python Script.....	31
Lesson 2 – Button with LEDs .....	35
Overview .....	35
Python Script.....	37
Lesson 3 - DS18B20 Temperature Sensing.....	39
Overview .....	39
Hardware.....	39
Software .....	42
Configure Test.....	44
Lesson 4 - IR Remote with a Raspberry Pi Media Center .....	46
Overview .....	46
Hardware.....	46
LIRC .....	48

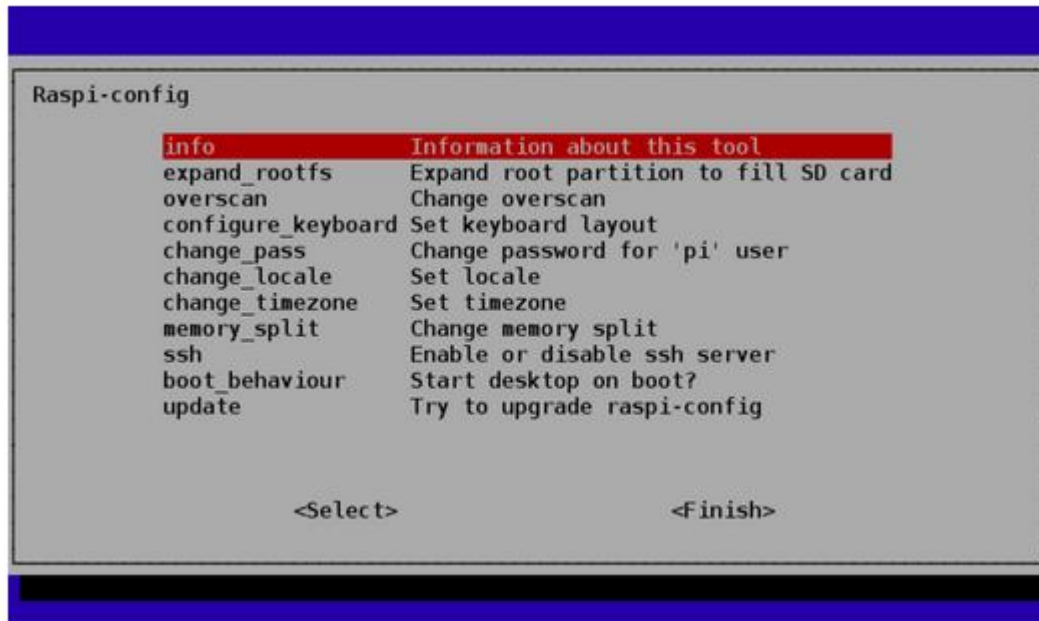
---

Configure and Test .....	49
Using Other Remotes .....	52
Lesson 5 - 16x2 LCD with the Raspberry Pi .....	54
Overview .....	54
Wiring the Cobbler to the LCD .....	55
LCD Pinout .....	56
Wiring Diagram.....	57
Schematic .....	57
Preparing the LCD .....	58
Necessary Packages .....	61
Python Script.....	62
Testing.....	63
IP Clock Example.....	64
Running the Code.....	65
Init Script.....	65
Time Zone .....	67

## Introduction 1. First Time Configuration

### Overview

In the first lesson of this series, we showed you how to prepare an SD card containing an operating system for your Raspberry Pi. In this lesson, we will show you how to setup your Raspberry Pi the first time you boot it up.

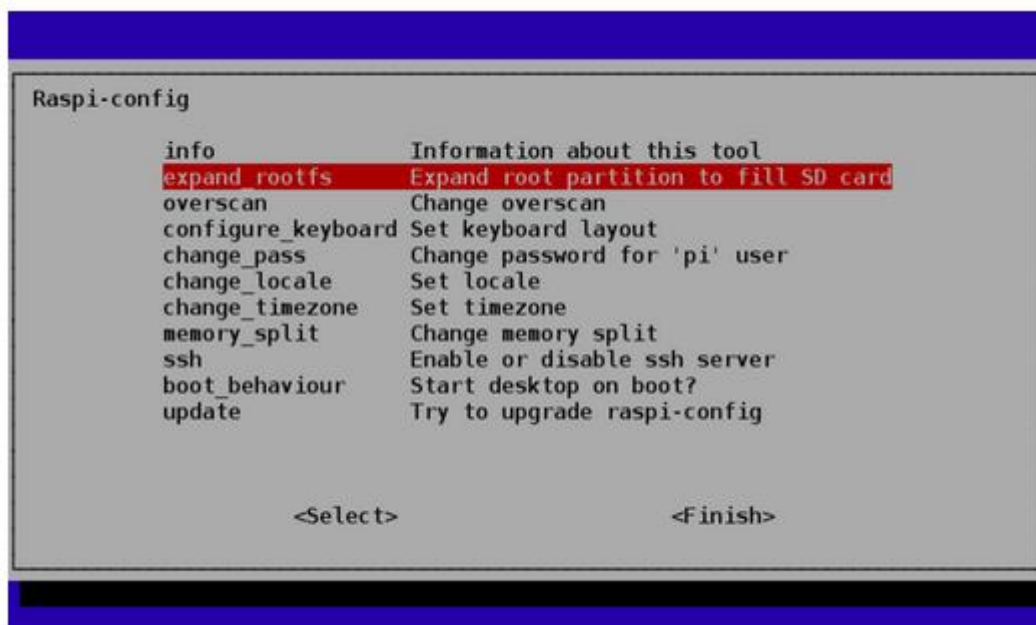


We do this using a tool called Raspi-Config that runs automatically the first time you boot your Raspberry Pi. This starts before the windowing system and so you have to use the cursor keys and Return key to navigate the menu system.

It is a bit like adjusting the BIOS settings on a PC, once you have things right, you probably won't need it again. We will start with the options that are most important and then look at some of the other options that you may wish to configure.

### Using the Whole SD Card

This may seem a bit strange, but by default the Raspberry Pi only uses as much of the SD card as the operating system requires. This means that even though you might have used a large SD card, the operating system won't use it.



To fix this so that all the space on the SD card can be used, use up / down cursor keys to select the 'expand\_rootfs' menu option and hit return.

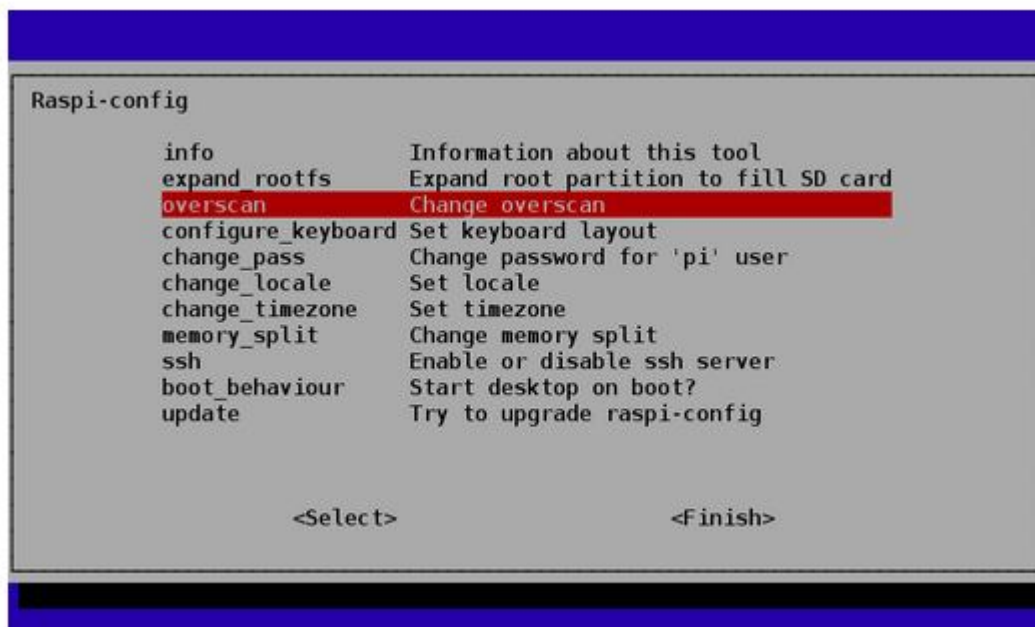
Once you do that, there will be some screen flashing as a script is run and then you will see the following confirmation.



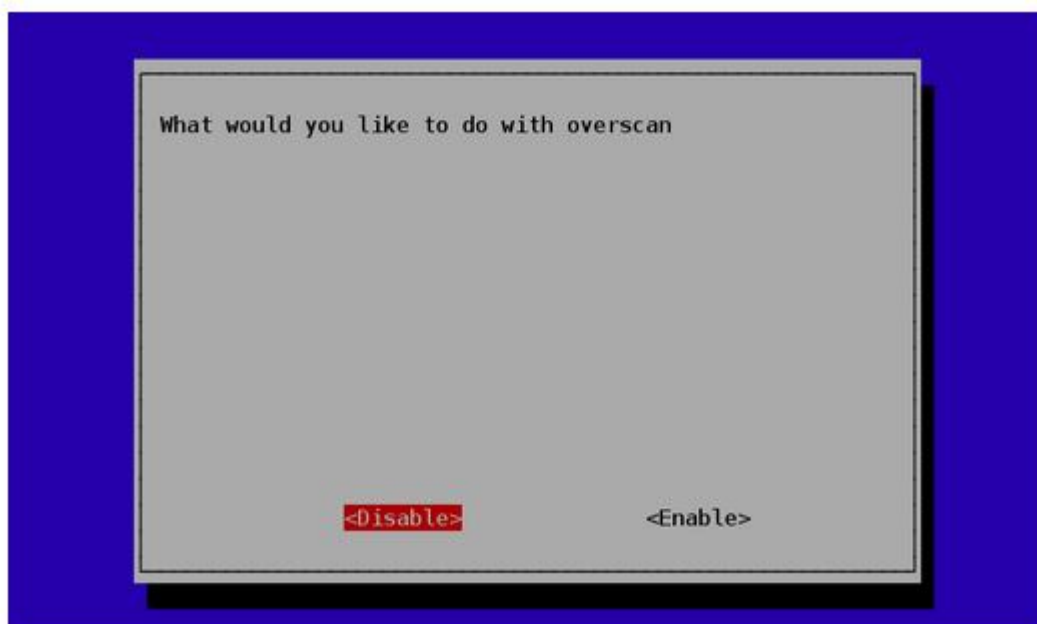
Press Return again to return to the main menu.

## Using the Whole Screen

Being designed to work with TVs, you may find that your Pi is only using the middle portion of the screen and there is a big unused area all round the screen.



This is not true of all TVs and monitors, but if its happening for you then selecting the option to Disable Overscan may fix this for you.

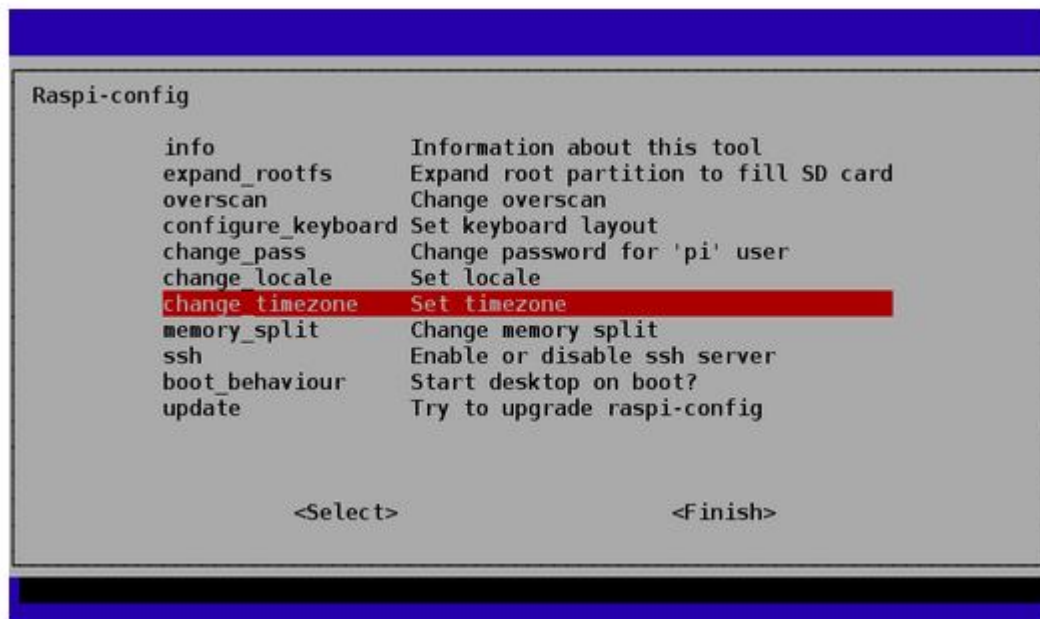


Use the left and right cursor keys to make your selection and then hit Return.

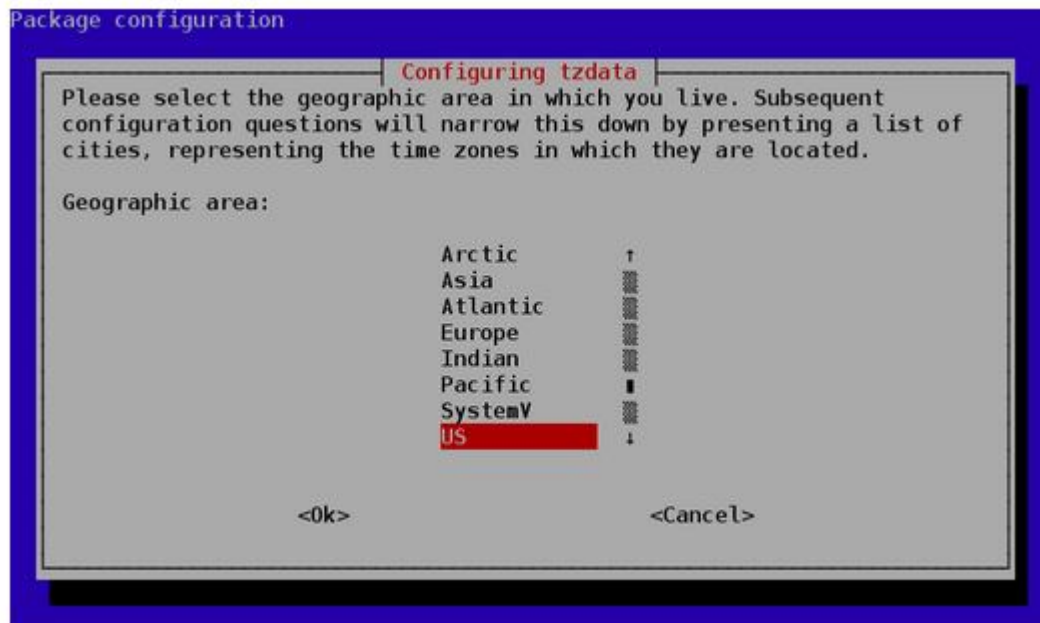
If after disabling overscan, you cannot see the left edge of the screen then see the section 'Running Raspi\_Config After Booting'.

## Changing Timezone

Skipping past a few options for a moment, the next thing that you almost certainly need to do is (unless you live in the GMT timezone) is to change the timezone.



From the options, first select the Geographic Area, then the Timezone within that area.

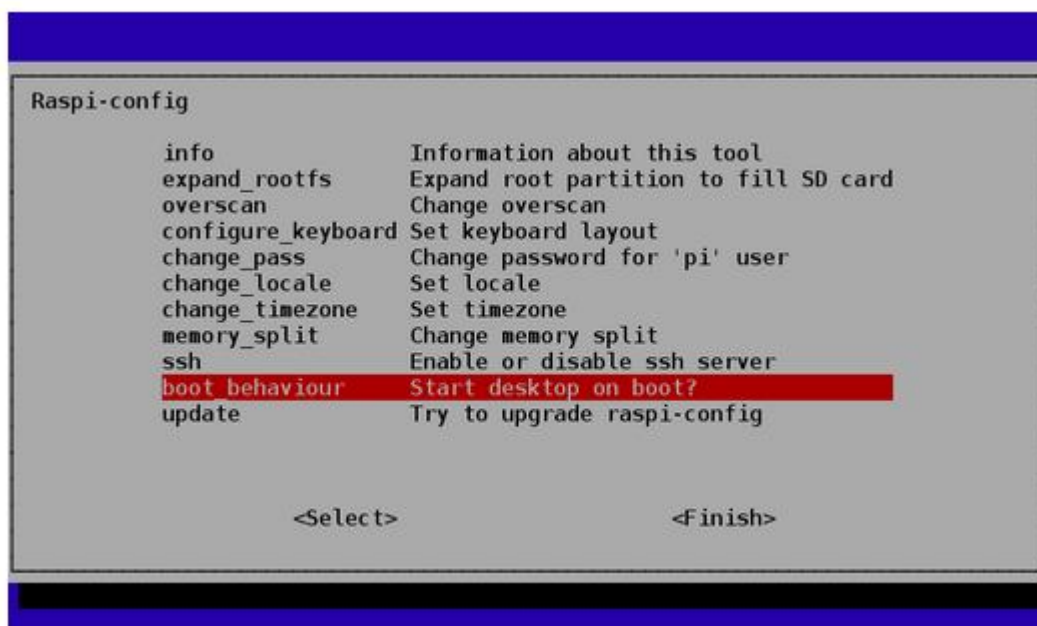




## Booting into Desktop

By default, when the Raspberry Pi boots, you just get a command line. No windows just a terminal where you can login and type commands.

The final configuration that you definitely want to make, unless you don't like windowing environments is to change the boot behaviour so that it automatically starts the windowing system and logs you in.



Select the sensible option and hit Return.



## Other Options

---

The options described above are those which could pretty much be considered essential when running your Raspberry Pi for the first time. There are some other options listed that are worth describing briefly.

- `configure_keyboard` – as it implies allows you to choose from a long list of keyboard layouts.
- `change_pass` – allows you to change the system password for the user 'pi' the default user on the system. By default, this password is 'raspberrypi' so those preoccupied with security may wish to change the password.
- `change_locale` – For non-English speakers, you can select which locales should be available on the system and which should be the default for the operating system.
- `memory_split` – allows you to adjust how much of the shared system memory is available for graphics and how much for the main processor. If you plan to run graphics hungry games, or video playback, then you may decide to alter these settings.
- `ssh` – in a later tutorial we will look at remote controlling your Pi from another computer using ssh. This option allows you to enable ssh so that you can do that.

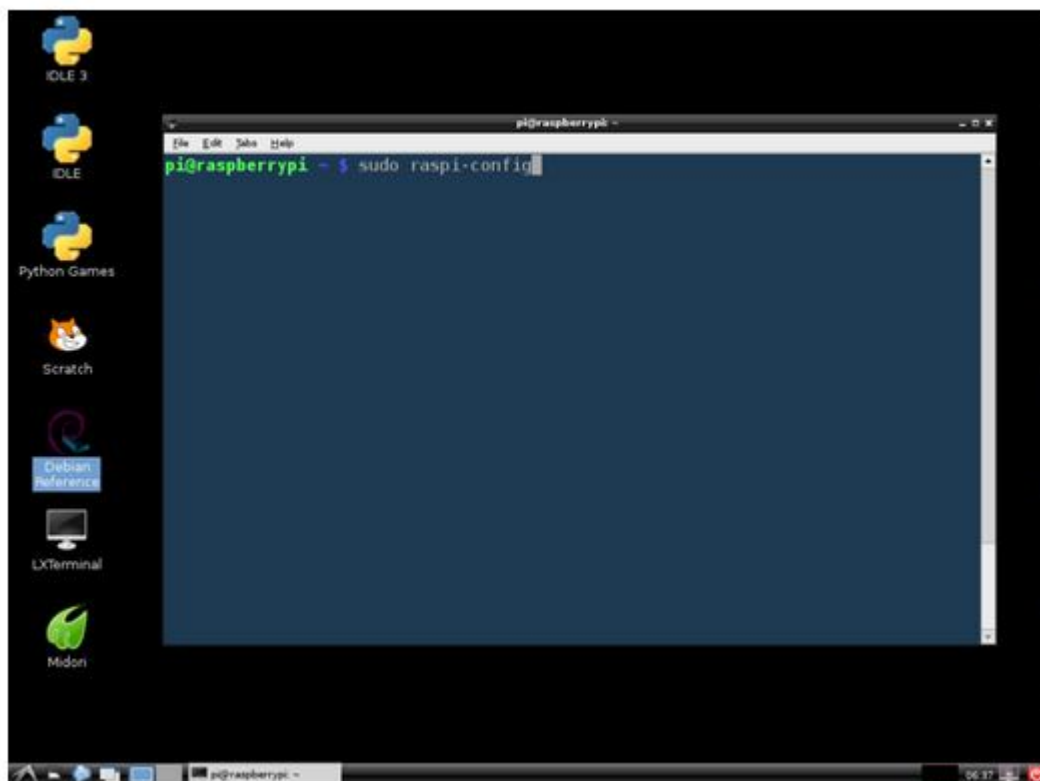


- update – this option tries to find a newer version of Raspi\_Config and download it. It is possible that new options will be added to the system in the future, so you may wish to do this.

## Running raspi-config After Booting

You can run **raspi-config** any time you like, if you find that there are some settings that you need to make.

Click on the desktop icon 'LXTerminal' to open a terminal session.



Then enter the following command and hit return:

```
1.      sudo raspi-config
```

---

## Test & Configure

---

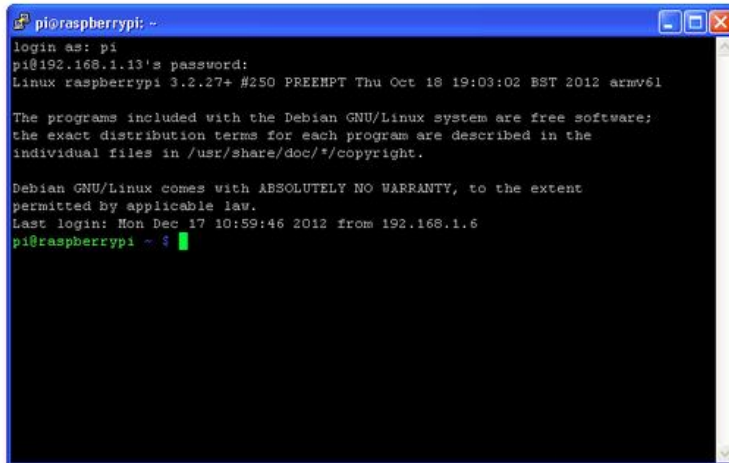


Reboot your Pi – click the icon bottom right – and this time you should boot straight up into the windowing environment.

## Introduction 2. Using SSH

### Overview

In this lesson you will learn how to remote control your Raspberry Pi over your local network using Secure Shell (SSH).



A common reason for remote controlling your Pi from another computer is that you may be using your Pi solely to control some electronics and therefore not need a keyboard, mouse and monitor, other than for setting it up.

It also can just save on desktop clutter, and the problem of having multiple keyboards and mice all over the place.

### Enabling SSH

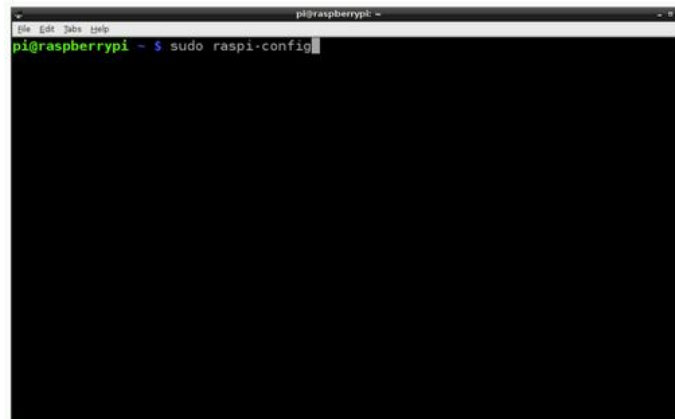
Secure Shell (SSH) is a feature of Linux that allows you to effectively open a terminal session on your Raspberry Pi from the command line of your host computer.

To use SSH, you need to first enable your Pi for using it. The easiest way to do this is to use Raspi Config, which you first saw back in “ **Introduction 1** ”

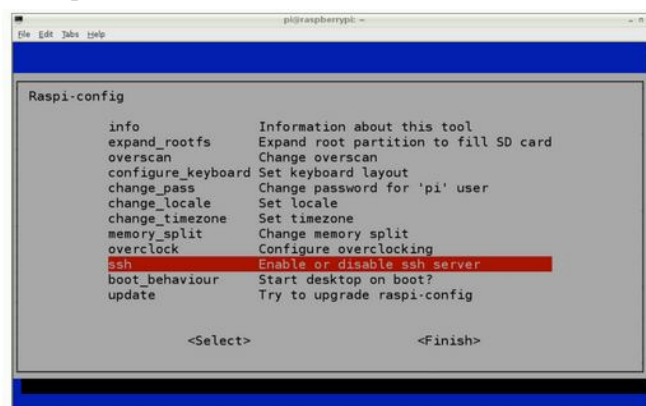
If you didn't setup your Pi for SSH when you first booted, no problem you can do it now.

Open LX Terminal on your Pi and enter the following command to start Raspi Config:

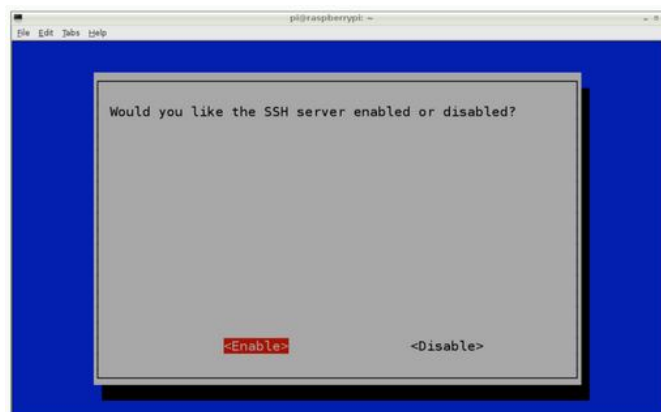
```
sudo raspi-config
```



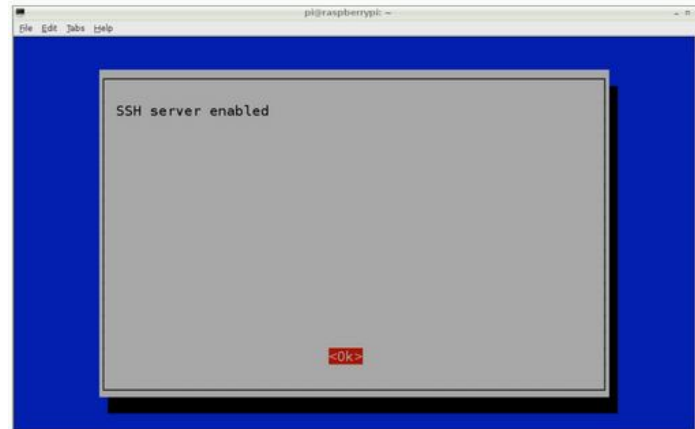
Scroll down to the “ssh” option.



Hit the Enter and then select “Enable”



A script will run and then you will see the following as confirmation:



## Using SSH on a Mac or Linux

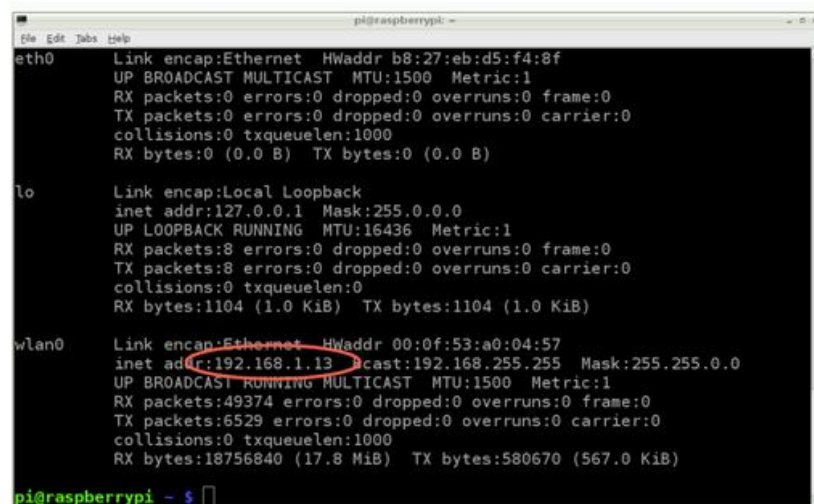
Now switch over to using the computer from which you wish to control the Pi.

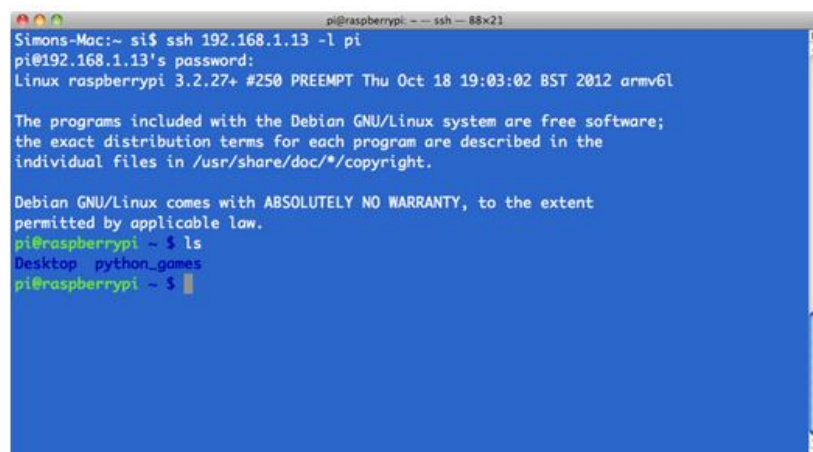
If you are using a Mac or Linux PC then open a Terminal. On the Mac, you can find this in the Utilities folder of your Applications folder.

Enter the following command into the Terminal window.

```
ssh 192.168.1.13 -l pi
```

Note that you will need to replace the IP address above with that of your Pi. You can find this by running the command “sudo ifconfig” from the Terminal.





```
pi@raspberrypi: ~ -- ssh -- 88x21
Simons-Mac:~ si$ ssh 192.168.1.13 -l pi
pi@192.168.1.13's password:
Linux raspberrypi 3.2.27+ #250 PREEMPT Thu Oct 18 19:03:02 BST 2012 armv6l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
pi@raspberrypi ~$ ls
Desktop  python_games
pi@raspberrypi ~$
```

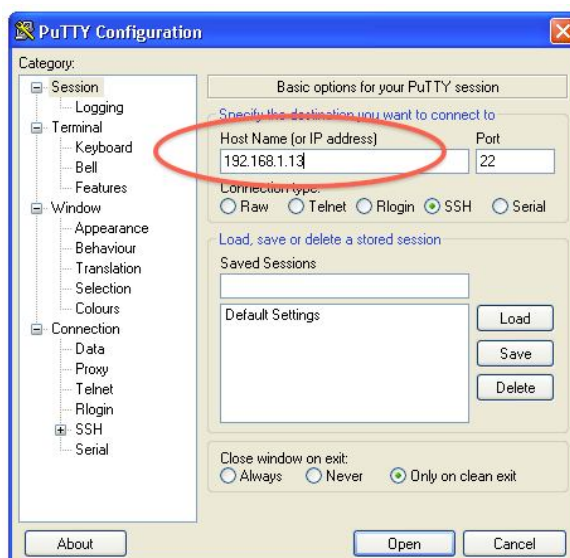
The option “**-l pi**” specifies that we want to log into the Pi as the user “pi”. The first time you run the command, you will get a security warning about not being able to verify the identity of the machine, say that you want to continue and enter your password (“raspberrypi” by default) when prompted.

You will notice that the command prompt will change to indicate that you are now connected to your Pi. Try using the “ls” command to show the contents of the current folder on the Pi.

## SSH under Windows

If you use windows, then you will need to download a free program called “putty” from here: <http://www.putty.org/>.

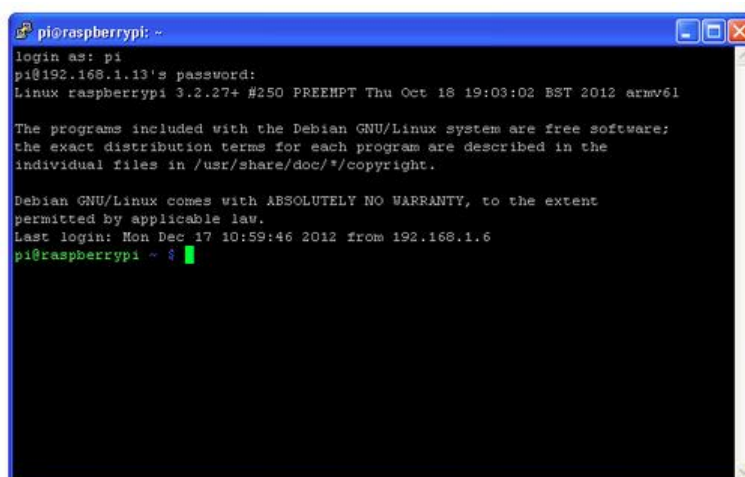
Having downloaded and installed Putty (its a single file called putty.exe), run the program.



Enter the IP address that you found earlier and click “Open”. This will give you a warning (the first time) and then prompt you for the user (“pi”) and password (“raspberrypi”).



The ssh window will then be ready for use.



## Test & Configure

Try exploring your files system by using 'ls' to list the files in the current directory and 'cd' followed by a directory name to change the current directory.

You can edit files using 'nano' followed by the file name and also install software using the 'apt-get' command, as described in some of the earlier tutorials in this series.

When finished with your ssh session, close the client application/window or simply type in **exit** into the shell window.

## Troubleshooting

---

If you encounter a **connection reset by peer** error when trying to connect to your Pi, there could be a problem with the SSH keys. You can 'reset' the keys with the following commands.

First, remove the old keys:

```
sudo rm /etc/ssh/ssh_host_*
```

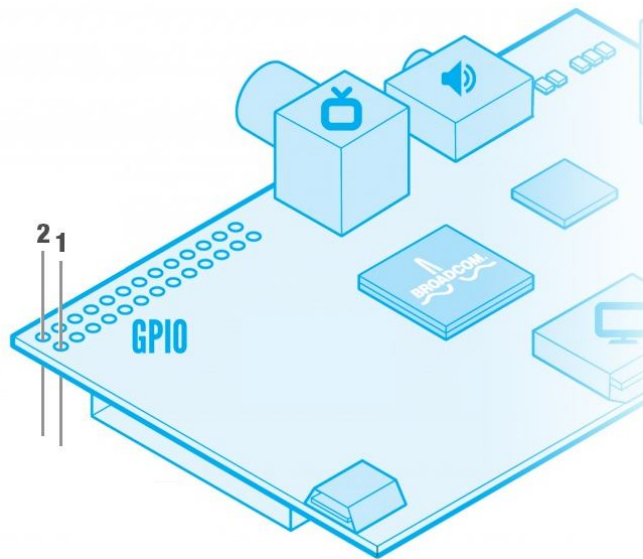
Then regenerate them

```
sudo dpkg-reconfigure openssh-server
```

Then try again!



## Introduction 3 -- Pinout! For the Revision 2.0 Raspberry Pi

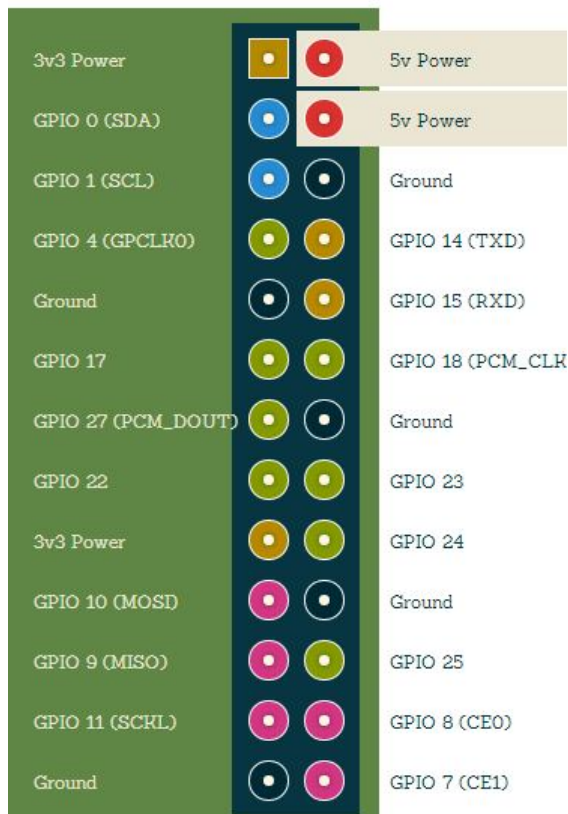


## Pinout

3v3 Power			5v Power	
GPIO 0 (SDA)			5v Power	
GPIO 1 (SCL)			Ground	
GPIO 4 (GPIR0)			GPIO 14 (TXD)	
Ground			GPIO 15 (RXD)	
GPIO 17			GPIO 18 (PCM_CLK)	
GPIO 27 (PCM_DOUT)			Ground	
GPIO 22			GPIO 23	
3v3 Power			GPIO 24	
GPIO 10 (MOSI)			Ground	
GPIO 9 (MISO)			GPIO 25	
GPIO 11 (SCKL)			GPIO 8 (CE0)	
Ground			GPIO 7 (CE1)	

1			2
3			4
5			6
7			8
9			10
11			12
13			14
15			16
17			18
19			20
21			22
23			24
25			26

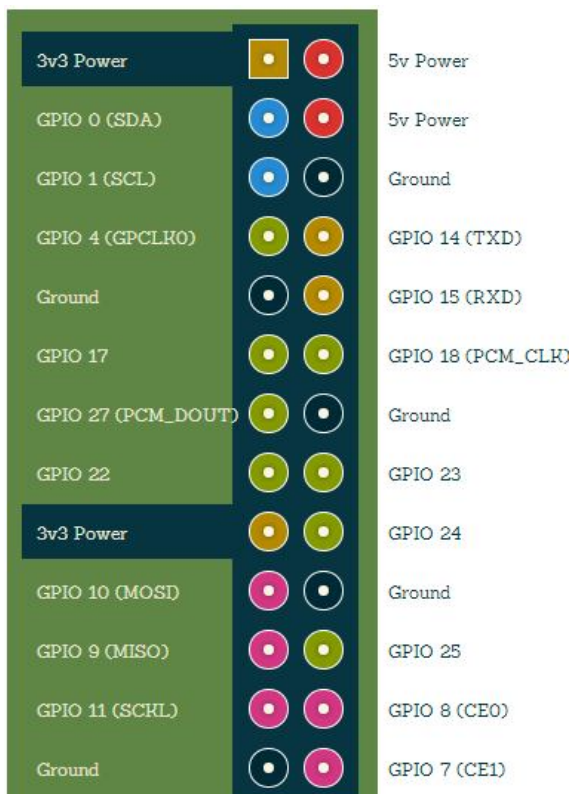


## 5v Power

The 5v power pins, two of them, provide your project with any current your power adaptor can muster minus that which is used by the Pi itself.

5v power is great for regulating down to 3.3v, which you can then use to power an ATmega that's logic-level compatible with your Pi.

If you need more than 5v, driving motors for example, you'll have to find your power elsewhere!



## 3.3v Power

At a measly 50mA max current, the 3.3v pin on the Raspberry Pi isn't terribly useful, but it's enough to light an LED or two.

## GPIO Setup

### Overview

One of the great things about the Raspberry Pi is that it has a GPIO connector to which you can attach external hardware.



The GPIO connector actually has a number of different types of connection on them. There are:

- True GPIO (General Purpose Input Output) pins that you can use to turn LEDs on and off etc.
- I2C interface pins that allow you to connect hardware modules with just two control pins
- SPI interface with SPI devices, a similar concept to I2C but a different standard
- Serial Rx and Tx pins for communication with serial peripherals

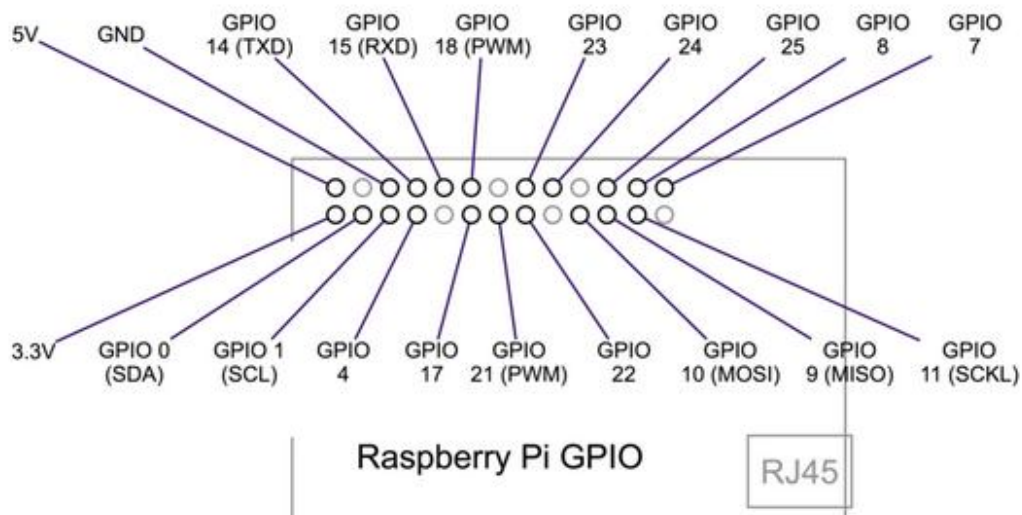
In addition, some of the pins can be used for PWM (pulse Width Modulation) for power control and another type of pulse generation for controlling servo motors called PPM (Pulse Position Modulation).

In this tutorial, you are not actually build anything, but you will learn how to configure your Raspberry Pi and install useful libraries ready to start attaching some external electronics to it.

This tutorial is written for Raspbian & Raspbian-derived installations (like Occidentalis) only

## The GPIO Connector

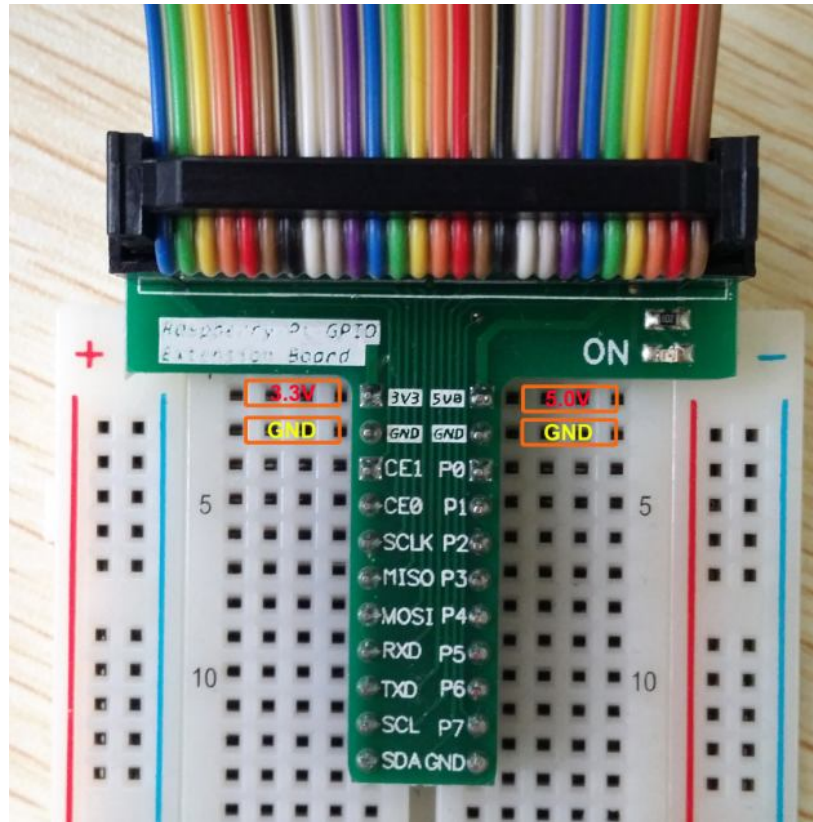
The diagram below show the pins on the GPIO connector for a **Raspberry Pi Version 1** (which is what existed when this tutorial was released) **Version 2** has pin 27 replacing pin 21 but it otherwise the same



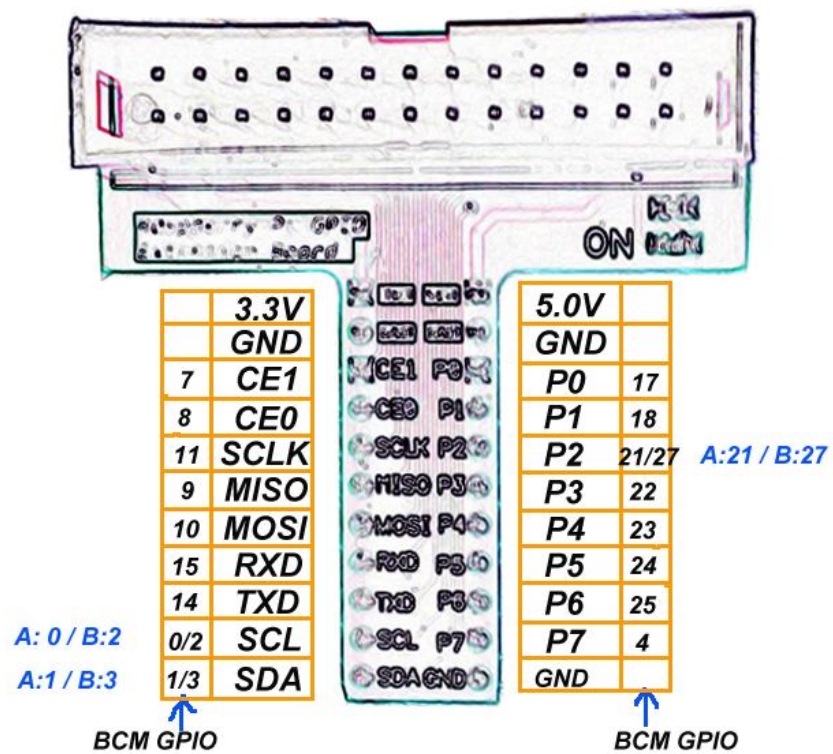
As well as supplying power (GND, 3.3V and 5V) all the GPIO pins can be used as either digital inputs or outputs. The pins labelled SCL and SDA can be used for I2C. The pins labelled MOSI, MISO and SCKL can be used to connect to high speed SPI devices. All the pins have 3.3V logic levels and are not 5V-safe so the output levels are 0-3.3V and the inputs should not be higher than 3.3V. If you want to connect a 5V output to a Pi input, [use a level shifter](#)

A popular way to actually make the connections to the Raspberry Pi is to use a Pi Cobbler.





For the T-Cobbler



P1: The Main GPIO connector						
WiringPi Pin	BCM GPIO	Name	Header	Name	BCM GPIO	WiringPi Pin
		3.3v	1 2	5v		
8	Rv1:0 - Rv2:2	SDA	3 4	5v		
9	Rv1:1 - Rv2:3	SCL	5 6	0v		
7	4	GPIO7	7 8	TxD	14	15
		0v	9 10	RxD	15	16
0	17	GPIO0	11 12	GPIO1	18	1
2	Rv1:21 - Rv2:27	GPIO2	13 14	0v		
3	22	GPIO3	15 16	GPIO4	23	4
		3.3v	17 18	GPIO5	24	5
12	10	MOSI	19 20	0v		
13	9	MISO	21 22	GPIO6	25	6
14	11	SCLK	23 24	CE0	8	10
		0v	25 26	CE1	7	11
WiringPi Pin	BCM GPIO	Name	Header	Name	BCM GPIO	WiringPi Pin

## BCM GPIO

To connect the T-Cobbler to your Pi Board, Please make sure it is correct, check the following picture .



Make extra extra double-check sure that the PIN 1 indicator is in the corner of the Pi. If you have a gray cable its probably a red stripe, for black cables, a white stripe. That pin must not be next to the TV connector. Turn around or twist the cable until it is right

This uses a ribbon cable to connect the GPIO connector to solderless breadboard, where you can add your own components.

## Raspberry Pi Code

To make life easy for those wishing to experiment with attaching electronics to their Pi, [Adafruit](#) have produced an extensive and extremely useful collection of code. This includes simple Python libraries for a large number of modules, including displays, sensors and PWM controllers etc.

To fetch this code, you need to use some software called 'git'. This comes pre-installed on Occidentalis, but on Raspbian you must install it by entering the following commands into LX Terminal.

You will find the icon for LX Terminal on your desktop.

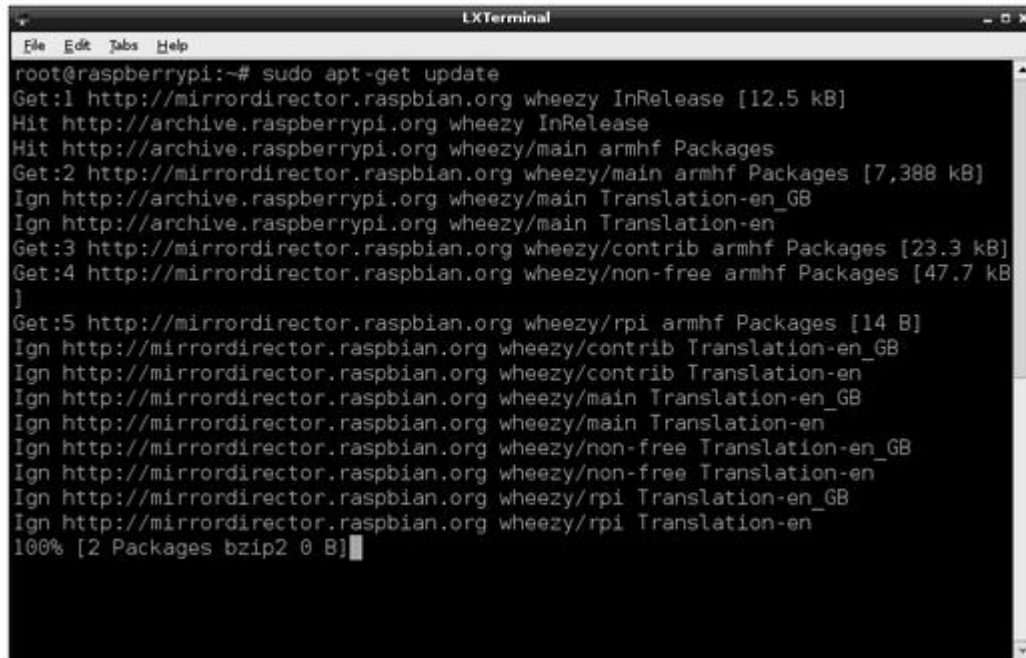


Before we go any further, issue the following command in LXTerminal. This will ensure your package can be found and that you get the latest version. It does not matter which directory you are in.

**Command :**

Before we go any further, issue the following command in LXTerminal. This will ensure your package can be found and that you get the latest version. It does not matter which directory you are in.

```
1.      sudo apt-get update
```



```
root@raspberrypi:~# sudo apt-get update
Get:1 http://mirrordirector.raspbian.org wheezy InRelease [12.5 kB]
Hit http://archive.raspberrypi.org wheezy InRelease
Hit http://archive.raspberrypi.org wheezy/main armhf Packages
Get:2 http://mirrordirector.raspbian.org wheezy/main armhf Packages [7,388 kB]
Ign http://archive.raspberrypi.org wheezy/main Translation-en_GB
Ign http://archive.raspberrypi.org wheezy/main Translation-en
Get:3 http://mirrordirector.raspbian.org wheezy/contrib armhf Packages [23.3 kB]
Get:4 http://mirrordirector.raspbian.org wheezy/non-free armhf Packages [47.7 kB]
]
Get:5 http://mirrordirector.raspbian.org wheezy/rpi armhf Packages [14 B]
Ign http://mirrordirector.raspbian.org wheezy/contrib Translation-en_GB
Ign http://mirrordirector.raspbian.org wheezy/contrib Translation-en
Ign http://mirrordirector.raspbian.org wheezy/main Translation-en_GB
Ign http://mirrordirector.raspbian.org wheezy/main Translation-en
Ign http://mirrordirector.raspbian.org wheezy/non-free Translation-en_GB
Ign http://mirrordirector.raspbian.org wheezy/non-free Translation-en
Ign http://mirrordirector.raspbian.org wheezy/rpi Translation-en_GB
Ign http://mirrordirector.raspbian.org wheezy/rpi Translation-en
100% [2 Packages bzip2 0 B]
```

The update may take a while, especially if this is the first time you have run it on your Pi. Eventually it should give you another command prompt '\$' and it will be ready for you to type the next command which is:

```
1.      sudo apt-get install git
```

Once git is installed (if its not already there) you can "check out" the Adafruit Pi Python repository onto your Pi using the following commands

sing the following commands

```
git clone http://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code.git
cd Adafruit-Raspberry-Pi-Python-Code
ls
```



If there is any problem during any of the steps above, you will see an error message. The most common reasons why something should fail to install are:

- a problem with your Internet connections
- a mis-typed command. Remember everything in Linux is case sensitive. It is best to open this page on your Raspberry Pi so you can just copy and paste the commands.

You will find all sorts of goodies in here, many of which we will use in later tutorials.

## Configuring GPIO

---

The GPIO pins can be used as both digital outputs and digital inputs. As digital outputs, you can write programs that turn a particular pin HIGH or LOW. Setting it HIGH sets it to 3.3V setting it LOW sets it to 0V. To drive an LED from one of these pins, you need a 560Ω resistor in series with the LED as the GPIO pins can only manage a small amount of power.

If you use a pin as a digital input, then you can connect switches and simple sensors to a pin and then be able to check whether it is open or closed (that is, activated or not) Some Adafruit projects that use just GPIO.

- Raspberry Pi E-mail Notifier Using LEDs
- Playing sounds and using buttons with Raspberry Pi

To program the GPIO ports in Python, we need to install a very useful Python 2 library called Rpi.GPIO. This module gives us a simple to use Python library that will let us control the GPIO pins.

The installation process for this is the same whether you are using Raspbian or Occidentalis. In actual fact, some versions of Raspbian include this library, but these instructions will also have the effect of updating to the latest version, which is worth doing.

```
1.      sudo apt-get update
```

To install RPi.GPIO, you first need to install the Python Development toolkit that RPi.GPIO requires.

To do this enter the following command into LXTerminal:

```
1.      sudo apt-get install python-dev
```

Then to install Rpi.GPIO itself type:

```
1.      sudo apt-get install python-rpi.gpio
```

You will probably be prompted to confirm by entering 'Y'.

Thats all there is to it. You are ready to try some of the projects I mentioned at the top of this section.

## Configuring I2C

---

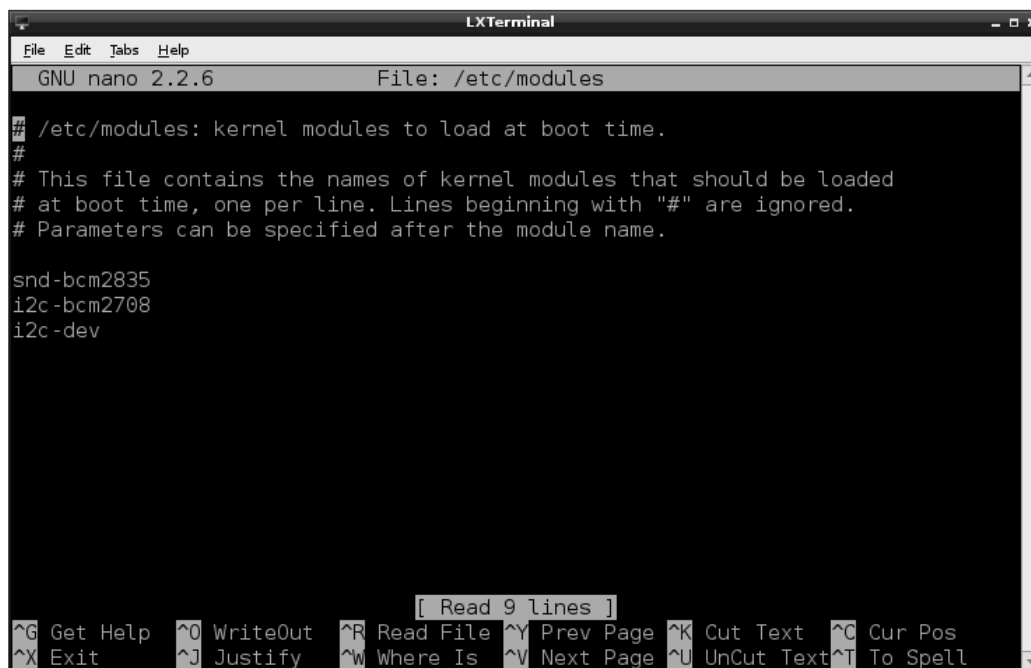
I2C is a very commonly used standard designed to allow one chip to talk to another. So, since the Raspberry Pi can talk I2C we can connect it to a variety of I2C capable chips and modules.

If you are using Occidentalis, then your Pi is ready to go with I2C as far as enabling the hardware goes. However, if you are using Raspbian, you will need to open LXTerminal and enter the following command:

```
1.      sudo nano /etc/modules
```

and add these  
two lines to the end of the file:

```
1.      i2c-bcm2708  
2.      i2c-dev
```



```
GNU nano 2.2.6 File: /etc/modules

/etc/modules: kernel modules to load at boot time.
#
# This file contains the names of kernel modules that should be loaded
# at boot time, one per line. Lines beginning with "#" are ignored.
# Parameters can be specified after the module name.

snd-bcm2835
i2c-bcm2708
i2c-dev
```

After editing the file, you will need to reboot for the changes to take effect.

If you have problems with I2C on Raspbian, then it is well worth updating to the latest version. These instructions were tested with the release dated 2012-10-28.

The I2C bus allows multiple devices to be connected to your Raspberry Pi, each with a unique address, that can often be set by changing jumper settings on the module. It is very useful to be able to see which devices are connected to your Pi as a way of making sure everything is working.

To do this, it is worth running the following commands in the Terminal to install the i2c-tools utility.

1. `sudo apt-get install python-smbus`
2. `sudo apt-get install i2c-tools`

Depending on your distribution, you may also have a file called **/etc/modprobe.d/raspi-blacklist.conf**

If you do not have this file then there is nothing to do, however, if you do have this file, you need to edit it and comment out the lines below:

1. `blacklist spi-bcm2708`
2. `blacklist i2c-bcm2708`

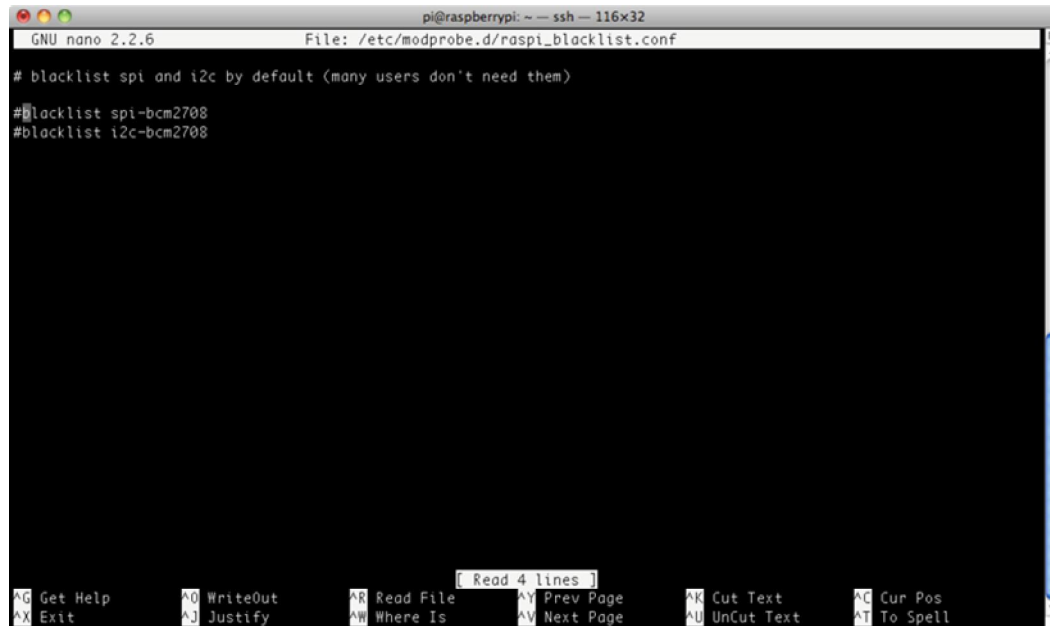
by putting a # in front of them.

Open an editor on the file by typing:

[Copy Code](#)

```
1.          sudo nano /etc/modprobe.d/raspi-blacklist.conf
```

.. then edit the file so that it appears as below, and then save and exit the file using CTRL-x and Y.



```
pi@raspberrypi: ~ — ssh — 116x32
GNU nano 2.2.6      File: /etc/modprobe.d/raspi-blacklist.conf

# blacklist spi and i2c by default (many users don't need them)

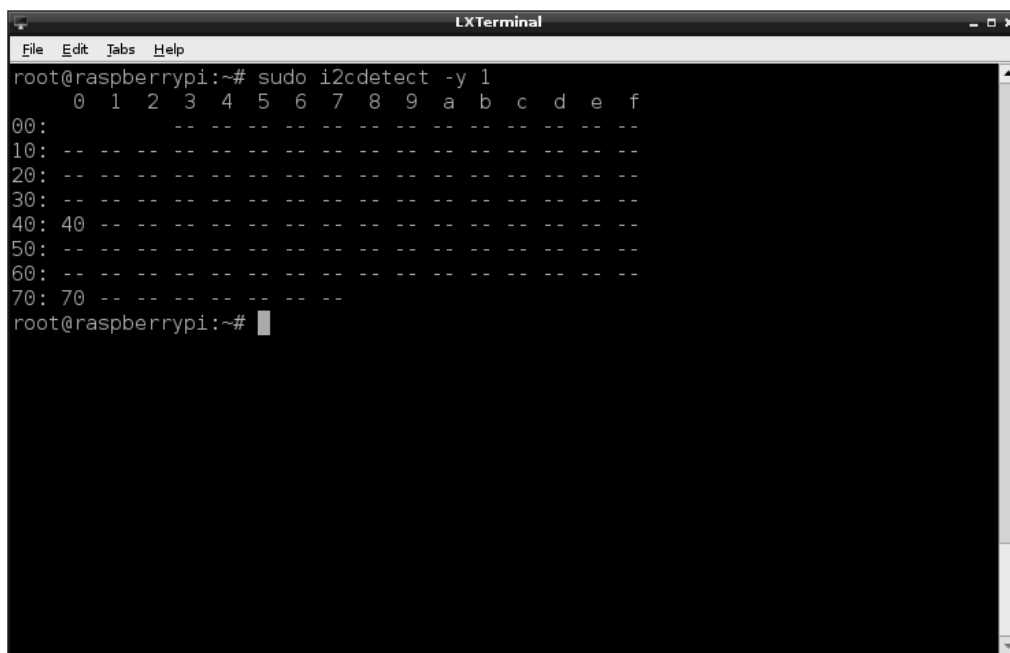
#blacklist spi-bcm2708
#blacklist i2c-bcm2708

[ Read 4 lines ]
^G Get Help      ^O WriteOut      ^R Read File     ^Y Prev Page     ^K Cut Text      ^C Cur Pos
^X Exit          ^J Justify       ^W Where Is     ^V Next Page     ^U UnCut Text   ^T To Spell
```

Once this is all done, you can type the following command to see all the connected devices (if you are running a 512MB Raspberry Pi Model B)

[Copy Code](#)

```
1.          sudo i2cdetect -y 1
```



```
root@raspberrypi:~# sudo i2cdetect -y 1
   0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
10:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
20:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
30:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: 40 -- -- -- -- -- -- -- -- -- -- -- -- -- --
50:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
60:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: 70 -- -- -- -- -- -- -- -- -- -- -- -- -- --
root@raspberrypi:~#
```

This shows that two I2C addresses are in use – 0x40 and 0x70.

Note that if you are using one of the very first Raspberry Pis (a 256MB Raspberry Pi Model B) then you will need to change the command to:

[Copy Code](#)

```
1.      sudo i2cdetect -y 0
```

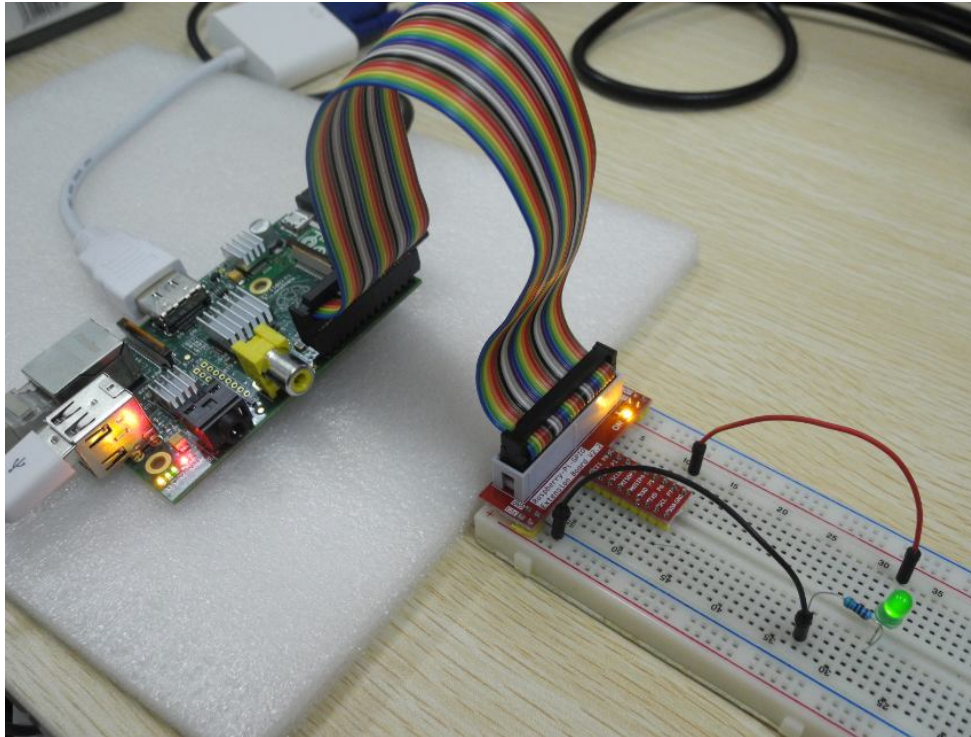
The Raspberry Pi designers swapped over I2C ports between board releases. Just remember: 512M Pi's use i2c port 1, 256M ones use i2c port 0!

## Lesson 1 - FLASHING LED

---

### Overview

---



**Difficulty:** Very Basic

This recipe will allow you to turn an LED into an output device for your Raspberry Pi and will guide you through writing a program to make it flash.

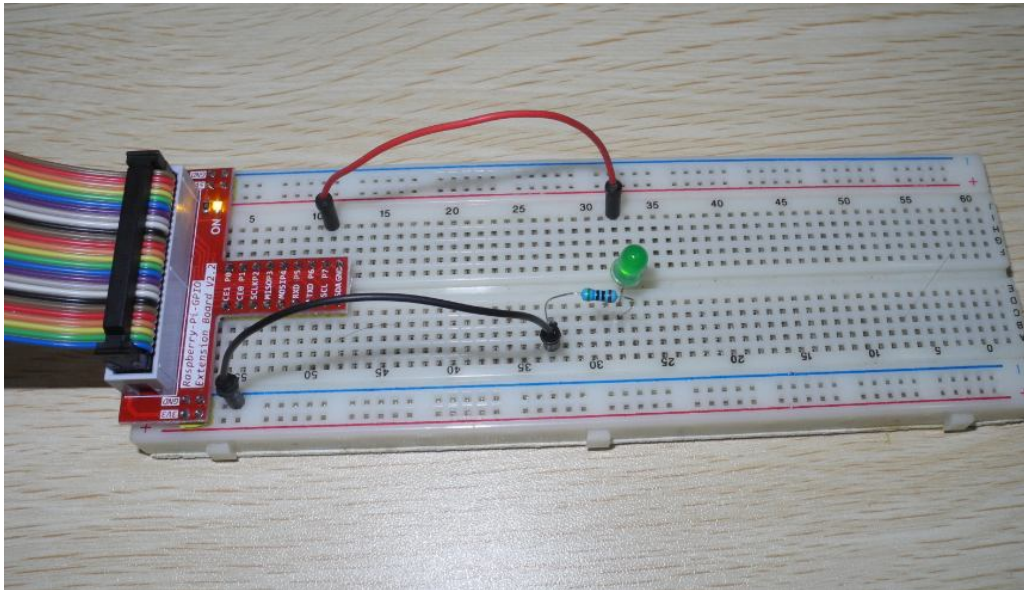
### Ingredients needed in addition to your Raspberry Pi:

- 1 x T-Cobbler
- 1 x GPIO cable .
- 1 x LED
- 1 x 560Ω Resistor
- 2 x Jumper Wire

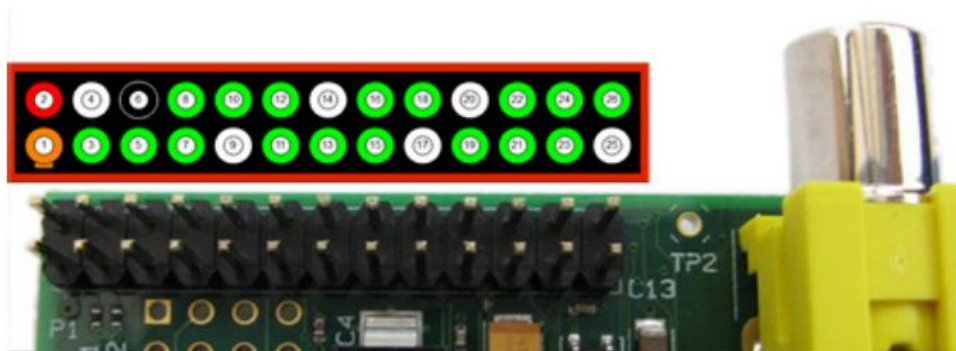
### Wire LED :

---

Turn an LED into an output for your program



1. Take one end of the resistor to the blackboard and connect it with the cathode of the LED (nearest flat edge and the shorter lead), and connect them to the GND .
2. Connect the anode ( longer lead ) of the LED to the P7 of the T-Cobbler with a jumper cable .

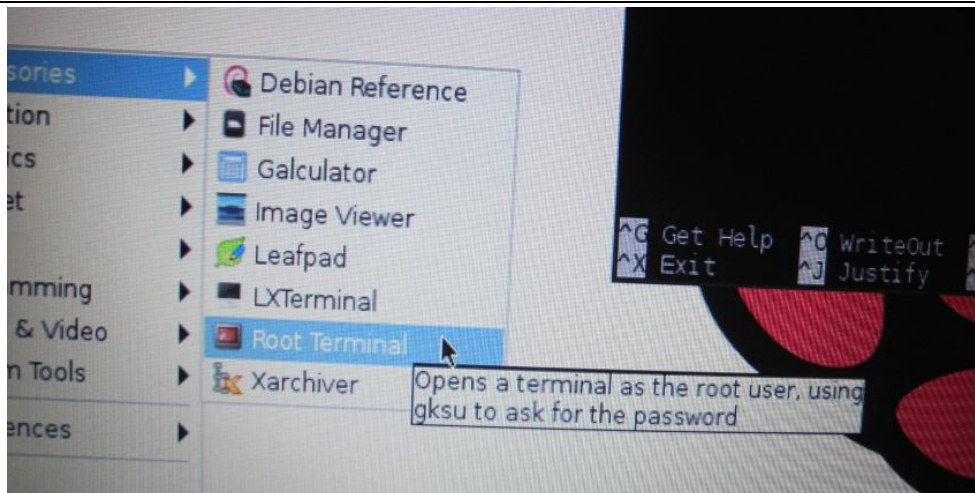


( In your Pi board, it is connect to the pin 7 , GPIO 4 )

## Python Script

(Please note that the program must be run as root by putting “sudo” in front of the Python command.)





1. Open a command line text editor

**nano FlashLED.py**

2. Type in the code below (Pro Tip: Any lines beginning with a # symbol are comments so don't need to be included for the program to work - they will, however, help you to understand the code)

```
# First we need to import the libraries that
# we need
# Import the time library so that we can make
# the program pause for a fixed amount of time
import time
# Import the Raspberry Pi GPIO libraries that
# allow us to connect the Raspberry Pi to
# other physical devices via the General
# Purpose Input-Output (GPIO) pins
import RPi.GPIO as GPIO
# Now we need to set-up the General Purpose
# Input-Output (GPIO) pins
# Clear the current set-up so that we can
# start from scratch
GPIO.cleanup()
# Set up the GPIO library to use Raspberry Pi
# board pin numbers
GPIO.setmode(GPIO.BOARD)
# Set Pin 7 on the GPIO header to act as
# an output
GPIO.setup(7,GPIO.OUT)
# This loop runs forever and flashes the LED
while True:
    # Turn on the LED
    GPIO.output(7,True)
```



```
# Wait for a second
time.sleep(1)
# Turn off the LED
GPIO.output(7,False)
# Wait for a second
time.sleep(1)
```



3. Type in the code below - type “Ctrl + X” and “Y” to exit nano and save the file.

4. Run the program

```
sudo python FlashLED.py
```

You can edit the file after the fact using the simple text editor **nano**, that is run

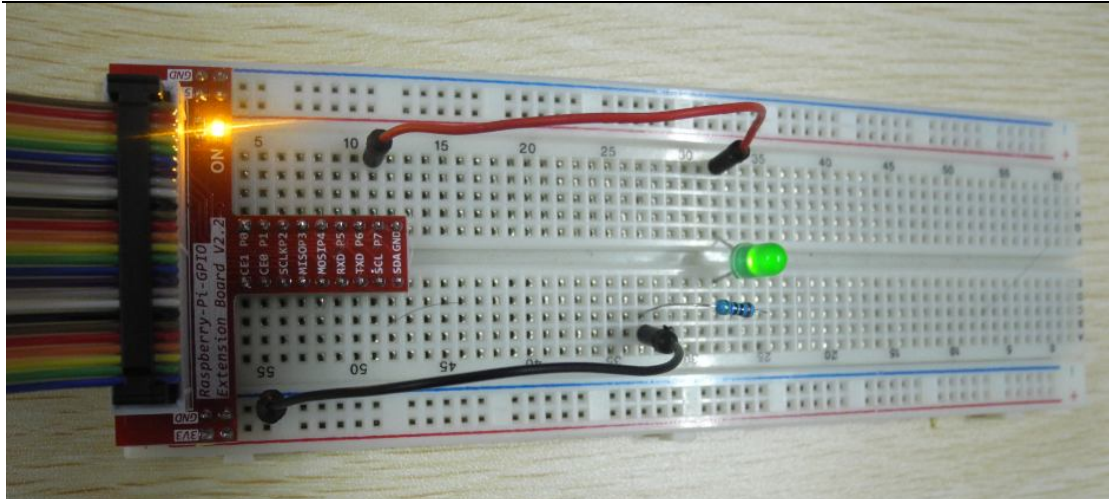
```
nano FlashLED.py to make simple edits
```

Next up, we'll make the file into an application (executable), type

```
chmod +x FlashLED.py
```

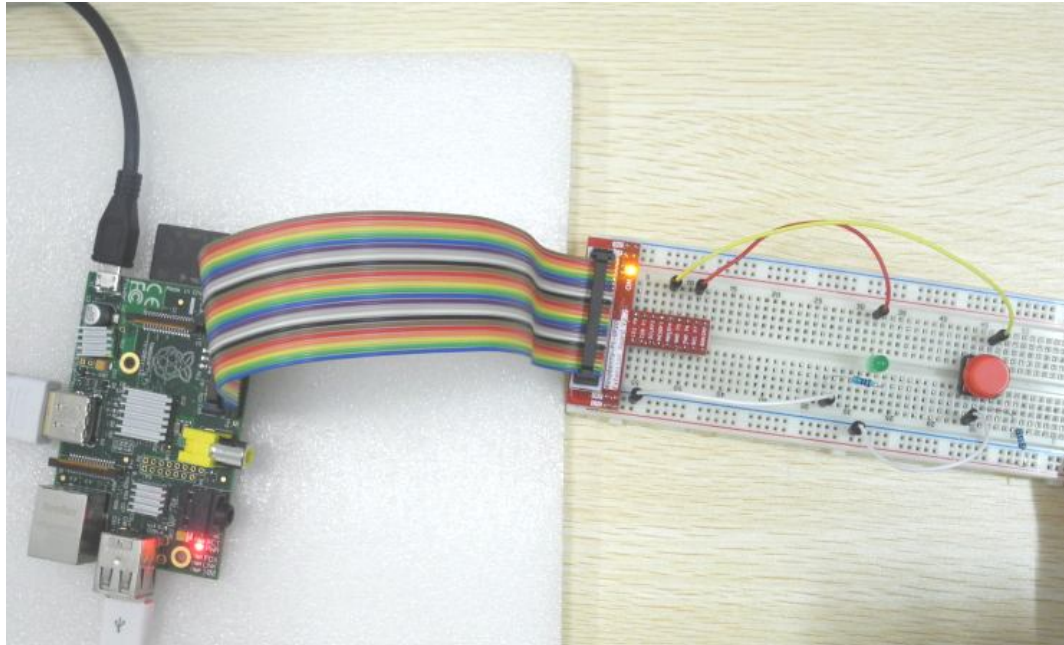
Finally you can run the script! Type in

```
sudo python ./ FlashLED.py
```



## Lesson 2 – Button with LEDs

### Overview



**Difficulty: Very Basic**

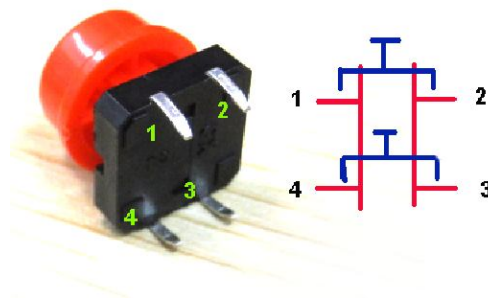
**Ingredients needed in addition to your Raspberry Pi:**

- 1 x T-Cobbler
- 1 x GPIO cable .
- 1 x LED
- 1 x 560 $\Omega$  Resistor for LED
- 1 x 10k pull-up resistor
- 1 x push-button switch
- some Jumper Wires

Cobbler 3.3v rail is tied to positive breadboard

Cobbler gnd rail is tied to negative breadboard

**About the Button :**

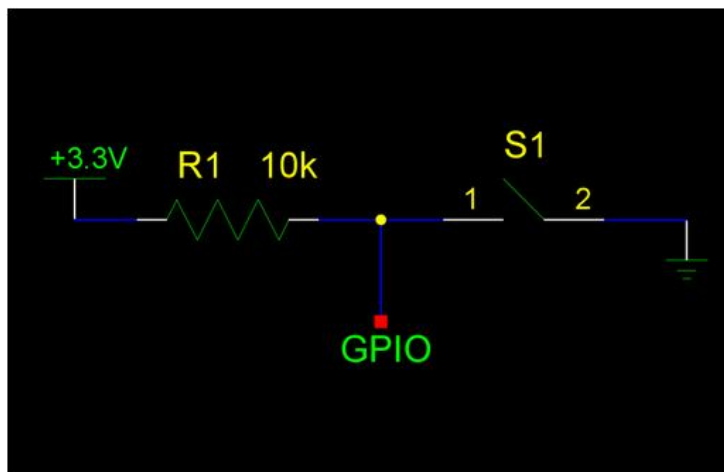


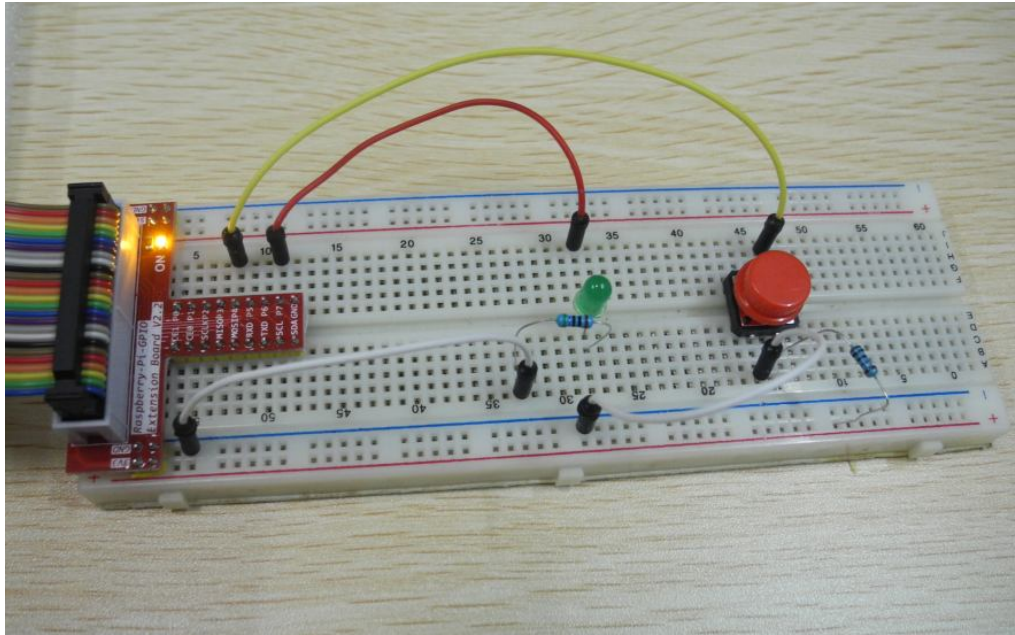
In the Button , the Pin “1” and “4” is conduction , and the Pin “2” and “3” is conduction .

## Wiring :

For the LED, Please check the pre-lesson 1 .

For the Button , please check this following wiring . And notice that to use the 3.3V .





- 1) Insert this code into a file named **Button.py**
- 2) Make the file executable with **chmod**

```
# First we need to import the libraries
# Import the Raspberry Pi GPIO libraries that
# allow us to connect the Raspberry Pi to
# other physical devices via the General
# Purpose Input-Output (GPIO) pins
import RPi.GPIO as GPIO

# Now we need to set-up the General Purpose
# Input-Output (GPIO) pins
# Clear the current set-up so that we can
# start from scratch
GPIO.cleanup()

# Set up the GPIO library to use Raspberry Pi
# BCM GPIO numbers
GPIO.setmode(GPIO.BCM)

# Set BCM GPIO 4 (Pin 7 on the GPIO header) to act as an output
GPIO.setup(4,GPIO.OUT)

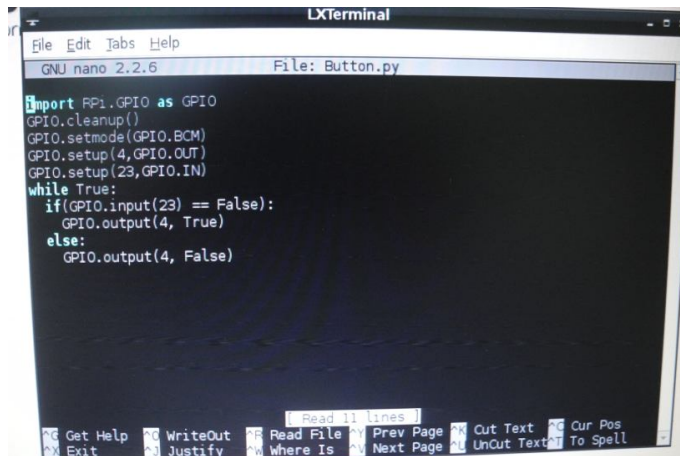
# Set BCM GPIO 23 (Pin 16 on the GPIO header) to act as an output
GPIO.setup(23, GPIO.IN)

# This loop runs forever and flashes the LED
while True:
    if(GPIO.input(23) == True):
```



```
# Turn on the LED
GPIO.output(4,True)
else:
    # Turn off the LED
    GPIO.output(4,False)
```

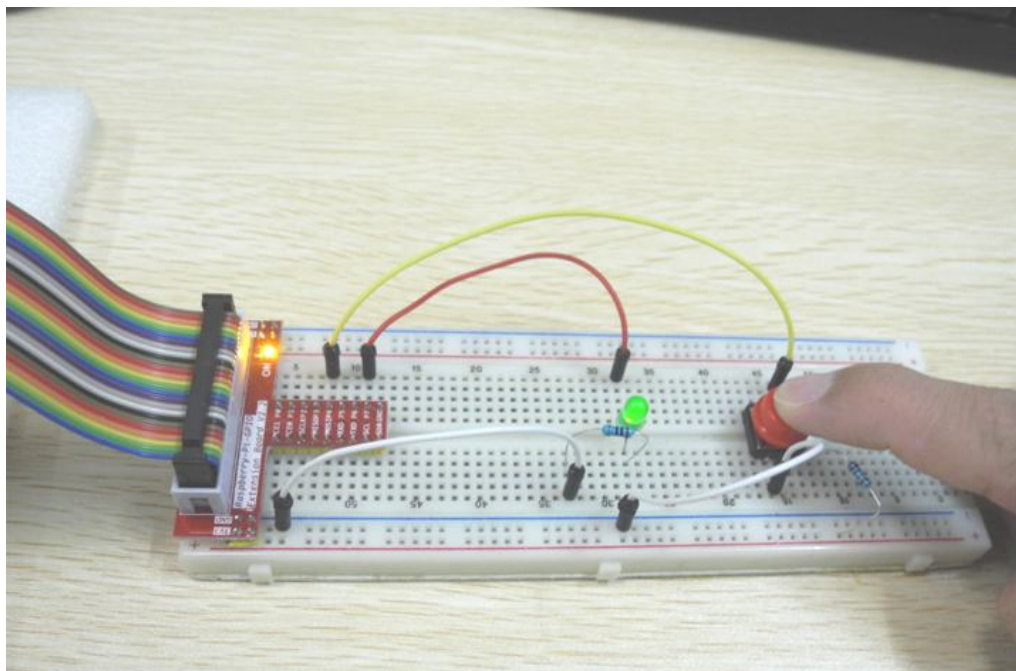
3. type “Ctrl + X” and “Y” to exit nano and save the file.



```
chmod +x raspi-audio-button.py
```

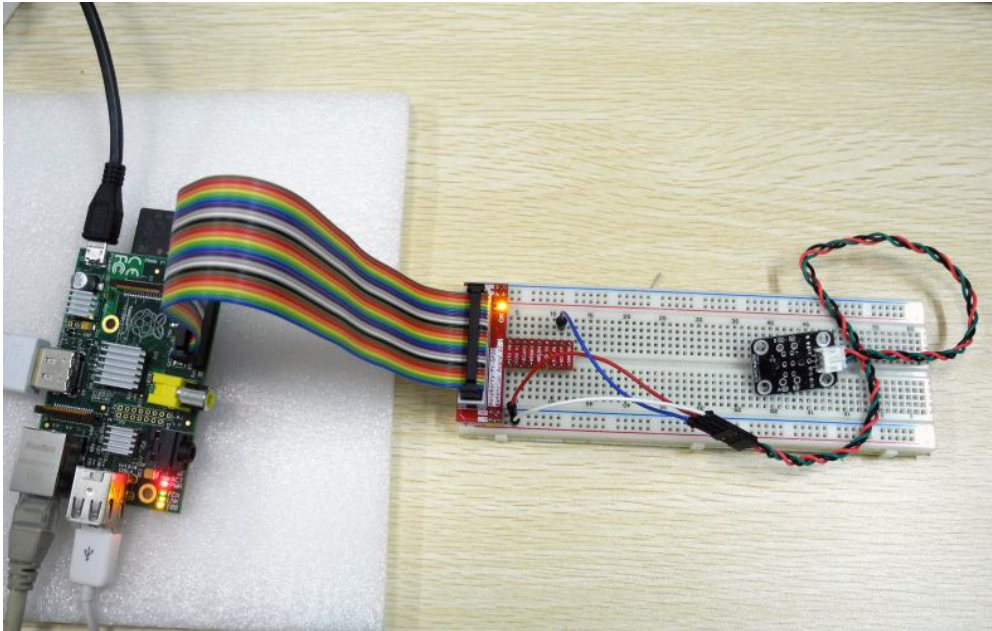
```
sudo python raspi-audio-button.py
```

When you push the button, the Led will be turn on , Otherwise , it is off .



## Lesson 3 - DS18B20 Temperature Sensing

### Overview



The Adafruit Occidentalis Linux distribution for Raspberry Pi (and Raspbian as of Dec 2012) includes support for the DS18B20 1-wire temperature sensor. These sensors come in a small three pin package like a transistor and are accurate digital devices

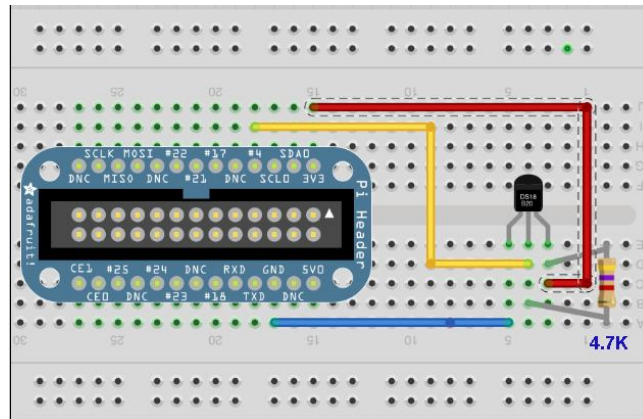
In this lesson, you will learn how to use a DS18B20 with the Raspberry Pi to take temperature readings.

Since the Raspberry Pi has no ADC (Analog to Digital Converter), it cannot directly use an analog temperature sensor like the TMP36, making the DS18B20 a good choice for temperature sensing.

### Hardware

For the DS18B20 component

The breadboard layout for just the basic DS18B20 is shown below.



The DS18B20 "1-wire" sensors can be connected in parallel - unlike nearly any other sensor sold! All sensors should share the same pins, but you only need one 4.7K resistor for all of them

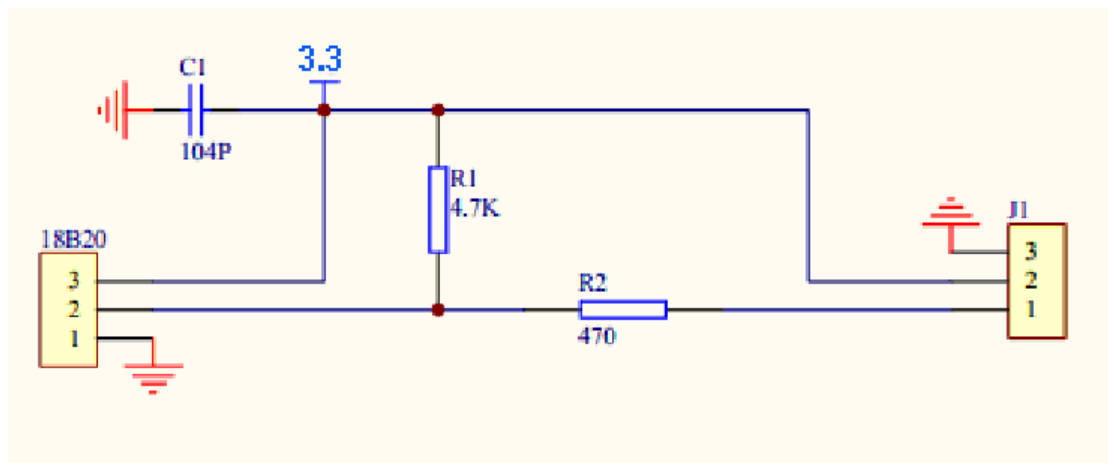
The resistor is used as a 'pullup' for the data-line, and is required to keep the data transfer stable and happy

Be careful to get the DS18B20 the right way around. The curved edge should be to the left as shown in the figure below. If you put it the wrong way around, it will get hot and then break.

**⚠** Despite both being temperature sensors, the DS18B20+ is totally different than the TMP36. You cannot use a TMP36 for this tutorial!

In this Lesson , we will use the DS18B20 module , that you can just plug it to the blackboard with jumpers .

The module's Schematic



**⚠** Please note that the pins of the DS18B20 module (as we find that some of the module cable is not correct ).

**3 → GND**

**2 → VCC (3.3v)**

**1 → P7 of the T-cobbler ( GPIO 4)**



Although the DS18B20 just looks like a regular transistor, there is actually quite a lot going on inside.

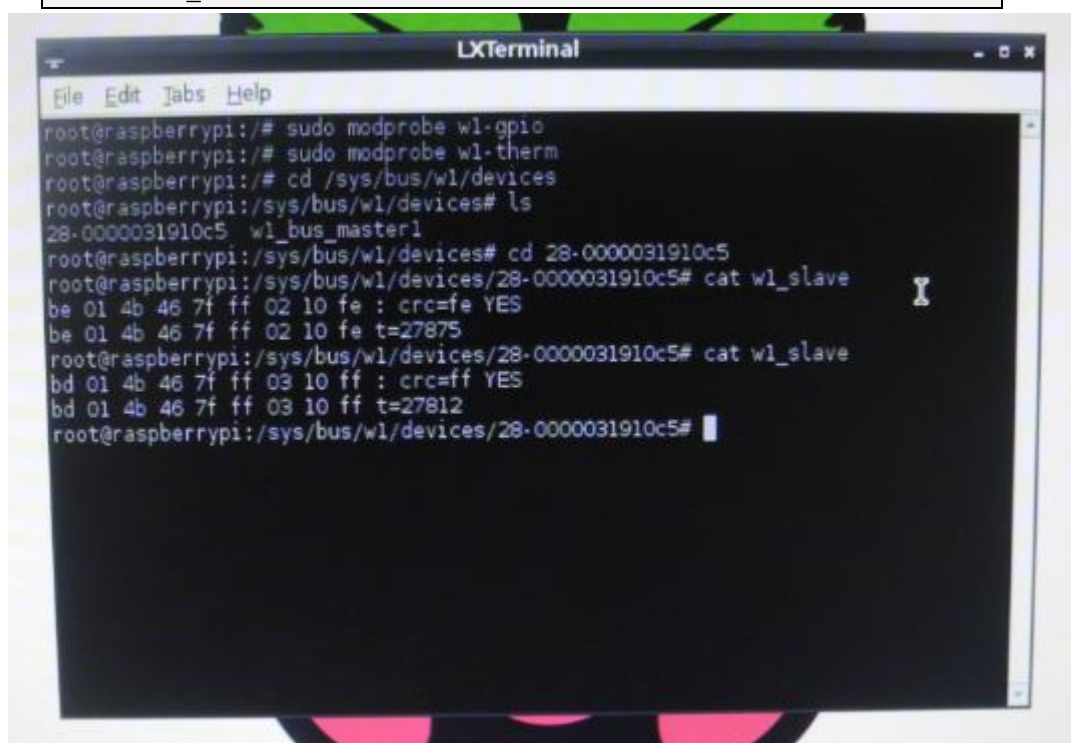
The chip includes the special 1-wire serial interface as well as control logic and the temperature sensor itself.

Its output pin sends digital messages and Raspbian/Occidentalis includes an interface to read those messages. You can experiment with the device from the command line or over SSH ([see Lesson 6](#)), before we run the full program.

Type the commands you see below into a terminal window. When you are in the 'devices' directory, the directory starting '28-' may have a different name, so cd to the name of whatever directory is there .

Please put the module connect to your pi first .

```
sudo modprobe w1-gpio
sudo modprobe w1-therm
cd /sys/bus/w1/devices
ls
cd 28-xxxx (change this to match what serial number pops up)
cat w1_slave
```



```
LXTerminal
File Edit Tabs Help
root@raspberrypi:~# sudo modprobe w1-gpio
root@raspberrypi:~# sudo modprobe w1-therm
root@raspberrypi:~# cd /sys/bus/w1/devices
root@raspberrypi:/sys/bus/w1/devices# ls
28-0000031910c5  w1_bus_master1
root@raspberrypi:/sys/bus/w1/devices# cd 28-0000031910c5
root@raspberrypi:/sys/bus/w1/devices/28-0000031910c5# cat w1_slave
be 01 4b 46 7f ff 02 10 fe : crc=fe YES
be 01 4b 46 7f ff 02 10 fe t=27875
root@raspberrypi:/sys/bus/w1/devices/28-0000031910c5# cat w1_slave
bd 01 4b 46 7f ff 03 10 ff : crc=ff YES
bd 01 4b 46 7f ff 03 10 ff t=27812
root@raspberrypi:/sys/bus/w1/devices/28-0000031910c5#
```

The interface is a little unreliable, but fortunately it tells us if there is a valid temperature to read. It's like a file, so all we have to do is read

The response will either have YES or NO at the end of the first line. If it is yes, then the temperature will be at the end of the second line, in 1/1000 degrees C. So, in the example above, the temperature is actually read as 20.687 and then 26.125 degrees C.

If you have more than one Sensor connected, you'll see multiple 28-xxx files.

Each one will have the unique serial number so you may want to plug one in at a time, look at what file is created, and label the sensor!

## Software

---

The Python program deals with any failed messages and reports the temperature in degrees C and F every second.

```
import os
import glob
import time

os.system('modprobe w1-gpio')
os.system('modprobe w1-therm')

base_dir = '/sys/bus/w1/devices/'
device_folder = glob.glob(base_dir + '28*')[0]
device_file = device_folder + '/w1_slave'

def read_temp_raw():
    f = open(device_file, 'r')
    lines = f.readlines()
    f.close()
    return lines

def read_temp():
    lines = read_temp_raw()
    while lines[0].strip()[-3:] != 'YES':
        time.sleep(0.2)
        lines = read_temp_raw()
    equals_pos = lines[1].find('t=')
    if equals_pos != -1:
        temp_string = lines[1][equals_pos+2:]
        temp_c = float(temp_string) / 1000.0
        temp_f = temp_c * 9.0 / 5.0 + 32.0
        return temp_c, temp_f

while True:
    print(read_temp())
    time.sleep(1)
```

The program starts by issuing the 'modprobe' commands that are needed to start the interface running.

The next three lines, find the file from which the messages can be read.

A problem has been reported with occasional hangs when reading the temperature file when using Raspbian. If you find you have the same problem, try replacing the function `read_temp_raw` with the code below. You will also need to add a line at the top of the file 'import subprocess'.

```
def read_temp_raw():
    catdata = subprocess.Popen(['cat', device_file],
                                stdout=subprocess.PIPE,
                                stderr=subprocess.PIPE)
    out, err = catdata.communicate()
    out_decode = out.decode('utf-8')
    lines = out_decode.split('\n')
    return lines
```

Reading the temperature takes place in two functions, `read_temp_raw` just fetches the two lines of the message from the interface. The `read_temp` function wraps this up checking for bad messages and retrying until it gets a message with 'YES' on end of the first line. The function returns two values, the first being the temperature in degrees C and the second in degree F.

You could if you wished separate these two as shown in the example below:

```
deg_c, deg_f = read_temp()
```

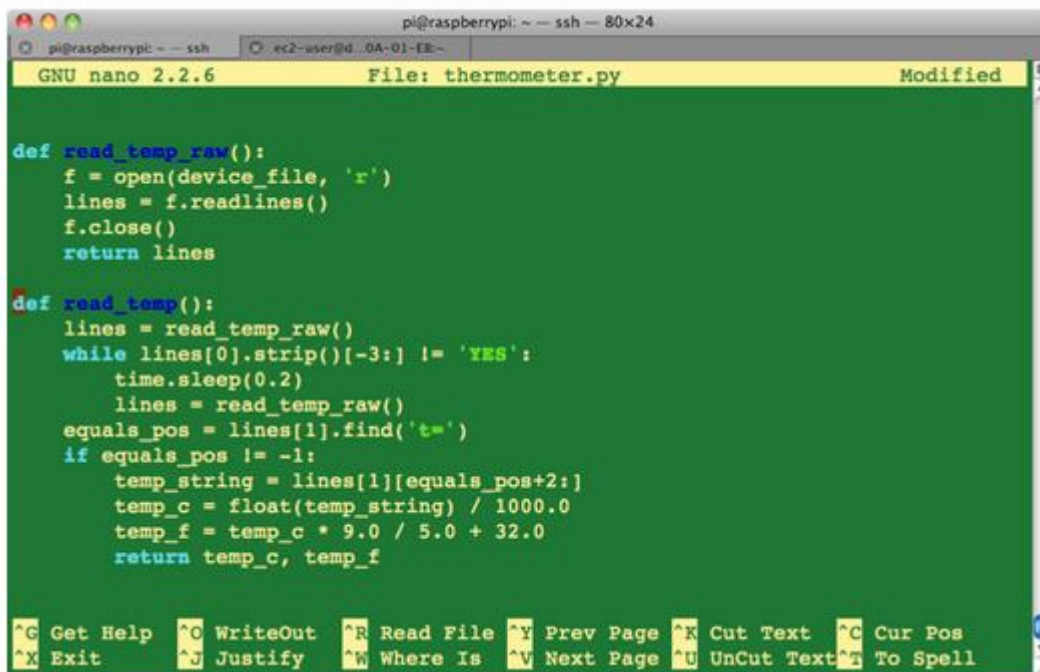
The main loop of the program simply loops, reading the temperature and printing it, before sleeping for a second.

To upload the program onto your Raspberry Pi, you can use SSH to connect to the Pi, start an editor window using the line:

```
nano thermometer.py
```

and then paste the code above, before saving the file with CTRL-x and Y.

```
deg_c, deg_f = read_temp()
```



```
pi@raspberrypi: ~ -- ssh -- 80x24
GNU nano 2.2.6 File: thermometer.py Modified

def read_temp_raw():
    f = open(device_file, 'r')
    lines = f.readlines()
    f.close()
    return lines

def read_temp():
    lines = read_temp_raw()
    while lines[0].strip()[-3:] != 'YES':
        time.sleep(0.2)
        lines = read_temp_raw()
    equals_pos = lines[1].find('t=')
    if equals_pos != -1:
        temp_string = lines[1][equals_pos+2:]
        temp_c = float(temp_string) / 1000.0
        temp_f = temp_c * 9.0 / 5.0 + 32.0
    return temp_c, temp_f

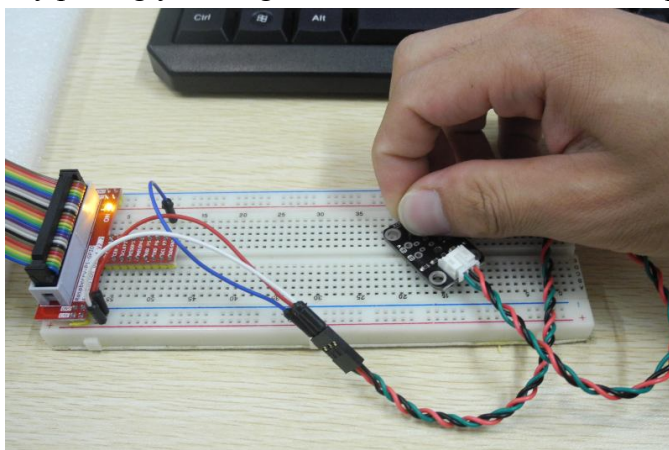
^G Get Help ^C WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

## Configure Test

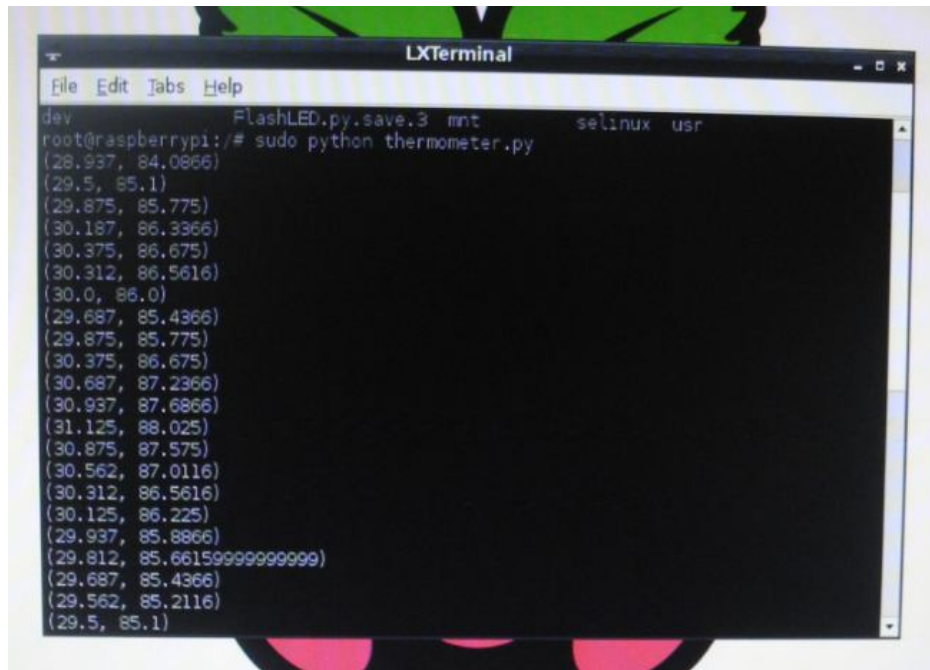
The program must be run as super user, so type the following command into the terminal to start it:

```
sudo python thermometer.py
```

Try putting your finger over the sensor to warm it up.



If all is well, you will see a series of readings like this:



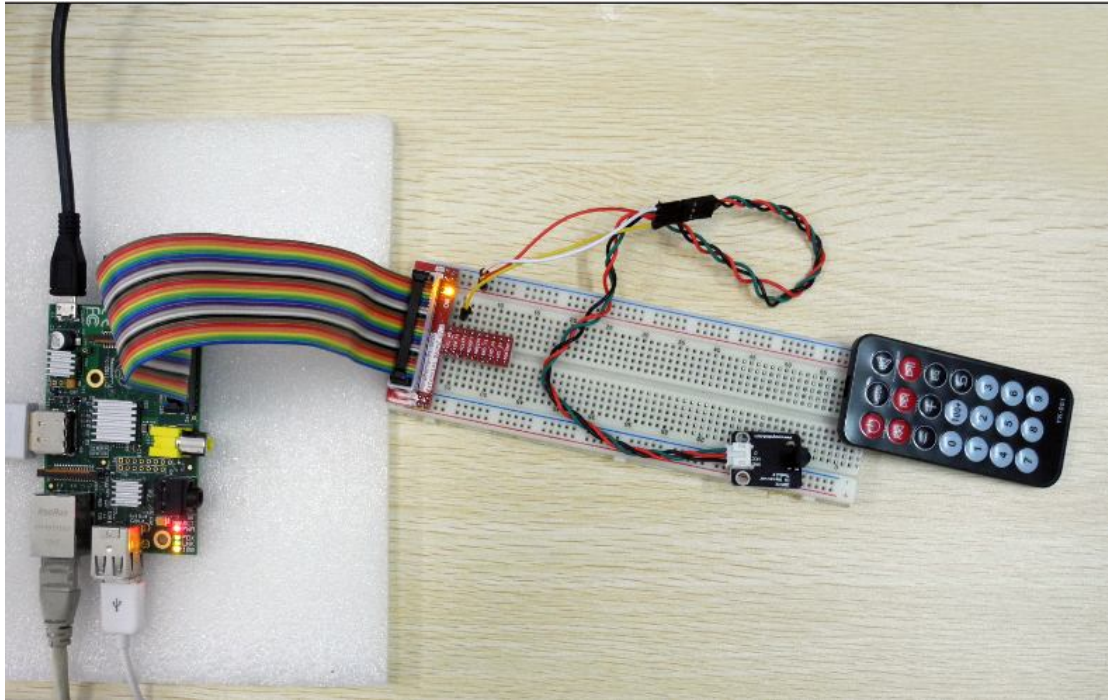
```
LXTerminal
File Edit Tabs Help
dev FlashLED.py.save.3 mnt selinux usr
root@raspberrypi:/# sudo python thermometer.py
(28.937, 84.0866)
(29.5, 85.1)
(29.875, 85.775)
(30.187, 86.3366)
(30.375, 86.675)
(30.312, 86.5616)
(30.0, 86.0)
(29.687, 85.4366)
(29.875, 85.775)
(30.375, 86.675)
(30.687, 87.2366)
(30.937, 87.6866)
(31.125, 88.025)
(30.875, 87.575)
(30.562, 87.0116)
(30.312, 86.5616)
(30.125, 86.225)
(29.937, 85.8866)
(29.812, 85.66159999999999)
(29.687, 85.4366)
(29.562, 85.2116)
(29.5, 85.1)
```

## Lesson 4 - IR Remote with a Raspberry Pi Media Center

---

### Overview

---



In this tutorial, you will learn how to use an Infrared remote with a Raspberry Pi configured as a media center.

The IR receiver is attached to the GPIO connector on the Raspberry Pi.

Before tackling this project, you need to follow this tutorial to set up your Raspberry Pi as a media center.

### Hardware

---



Pin Connect :

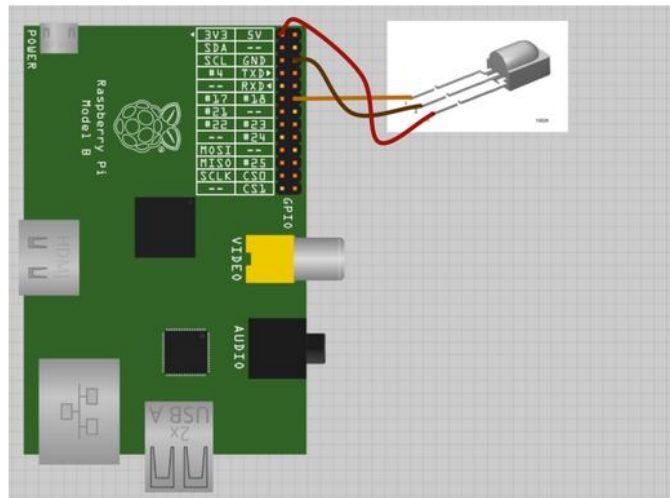
**GND** → **gnd**



**VCC** → **+5v**  
**D** → **cobbler P1 (GPIO 18)**

The IR sensor has just three pins, that will connect with three pins on the GPIO connector. To do the connecting, we can use jumper cable and blackboard . These make a good reliable connection as the IR sensor has unusually thick leads for an IC.

Make the connections as shown below. Note that you do not have to use the same colored jumper wires. But selecting adjacent wires that are still in a 'ribbon' will help keep things neat.



Note that the IR sensor chip needs to be operated at 3.3V not 5V when used with the Raspberry Pi.



For the Component,  
 The picture is the showing in the picture .

**IRM\_3638** Infrared Receiver . The Pin of it is

## LIRC

The interface between the hardware and the Raspberry Pi media centre is managed by a piece of software called LIRC (Linux Infrared Remote Control). This is pre-installed on most recent Raspberry Pi distributions and is included in the Rasbmc distribution, so there is nothing to install, however, there is some setting up to do.

To make sure that the IR hardware is correct, we can connect to the Raspberry Pi running Rasbmc using SSH, which is automatically enabled on this distribution.

If you have not connected to a Raspberry Pi using SSH before, please see this tutorial. “ **Introduction 2. Using SSH** “

You can find the IP address of the Raspberry Pi using the XBMC System Info page.



To be able to test the IR receiver without XBMC, you need to make sure that the IR remote feature is turned off, or you will not be able to use LIRC from the SSH. So run the Rasbmc Settings program and make sure that the option `Enable GPIO TSOP IR Receiver` is disabled.





If you needed to change this you will need to reboot.

Now connect to the Raspberry Pi using SSH and issue the commands shown below:

```
pi@raspbmc: ~ -- ssh -- 80x24
bash
pi@raspbmc:~$ sudo modprobe lirc_rpi
pi@raspbmc:~$ sudo kill $(pidof lircd)
pi@raspbmc:~$ mode2 -d /dev/lirc0
space 3992872
pulse 8947
space 4438
pulse 576
space 528
pulse 610
space 487
pulse 608
space 521
pulse 590
space 535
pulse 587
space 504
pulse 621
space 505
pulse 584
space 537
pulse 586
space 506
pulse 614
space 1654
```

Now hold the remote in front of the receiver and you should see a series of 'pulse' / 'space' messages appear each time you press a button.

Congratulations! The IR receiver is working.

## Configure and Test

Now that we know that the hardware is okay, we need to give LIRC a config file to tell it about the keys on the remote that we are using.

From the SSH session, issue the command:

```
nano lircd.conf
```

... and then paste the following text into it, before saving the file by clicking CTRL-x then Y.

```
# Please make this file available to others
# by sending it to <lirc@bartelmus.de>
#
# this config file was automatically generated
# using lirc-0.9.0-pre1(default) on Thu Mar 14 14:21:25 2013
#
# contributed by
#
# brand:                /home/pi/lircd.conf
# model no. of remote control:
# devices being controlled by this remote:
#

begin remote

    name    /home/pi/lircd.conf
    bits    16
    flags SPACE_ENC|CONST_LENGTH
    eps     30
    aeps    100

    header   8945  4421
    one      594   1634
    zero     594   519
    ptrail   598
    repeat   8949  2187
    pre_data_bits  16
    pre_data      0xFD
    gap           106959
    toggle_bit_mask 0x0

    begin codes
        KEY_VOLUMEDOWN          0x00FF
        KEY_PLAYPAUSE           0x807F
        KEY_VOLUMEUP            0x40BF
        KEY_SETUP                0x20DF
        KEY_UP                   0xA05F
        KEY_STOP                 0x609F
```

```
KEY_LEFT      0x10EF
KEY_ENTER     0x906F
KEY_RIGHT     0x50AF
KEY_KP0       0x30CF
KEY_DOWN      0xB04F
KEY_BACK      0x708F
KEY_KP1       0x08F7
KEY_KP2       0x8877
KEY_KP3       0x48B7
KEY_KP4       0x28D7
KEY_KP5       0xA857
KEY_KP6       0x6897
KEY_KP7       0x18E7
KEY_KP8       0x9867
KEY_KP9       0x58A7
```

```
end codes
```

```
end remote
```

This file should be saved in the home directory for the user pi.

Now, return to the Rasbmc Settings program and enable the option **Enable GPIO TSOP IR Receiver**. At the same time, change the GPIO Remote Profile as shown below:



Restart XMBC and when it has rebooted, you should see a small popup message in the bottom right corner like the one below.



You should now find that your IR remote control will work and that you no longer need the keyboard and mouse to control XMBC.

## Using Other Remotes

I generated the config file for this remote using a utility that is part of LIRC called 'irrecord'. If you have a different remote, then you can generate a config file for it using this tool. The process is as follows:

- Turn the remote off on XMBC using Rasbmc as we did before using 'mode2'.
- Rename the existing lircd.conf out of the way
- Type the command 'irrecord --list-namespace'. This will tell you the allowed key names that you can use when prompted.
- Type the command 'irrecord -d /dev/lirc0 ~/lircd.conf'
- Follow the instructions to the letter. It all seems a bit odd, but the program has to work out the timings and encodings used by the remote.

```
pi@raspbmc: ~ -- ssh -- 80x24
bash
Found const length: 106959
Please keep on pressing buttons like described above.
.....
Space/pulse encoded remote control found.
Signal length is 67.
Found possible header: 8945 4421
Found trail pulse: 598
Found repeat code: 8949 2187
Signals are space encoded.
Signal length is 32
Now enter the names for the buttons.

Please enter the name for the next button (press <ENTER> to finish recording)
KEY_VOLUMEDOWN

Now hold down button "KEY_VOLUMEDOWN".

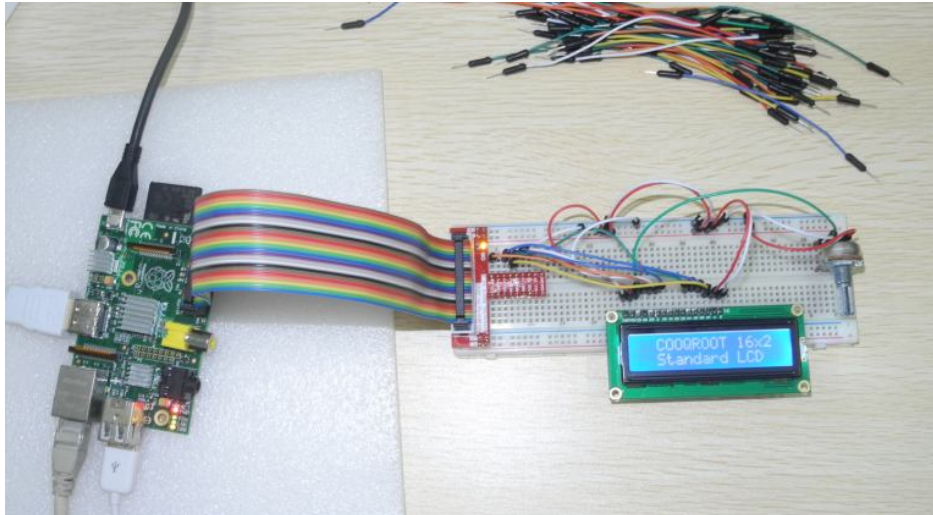
Please enter the name for the next button (press <ENTER> to finish recording)
KEY_PLAYPAUSE

Now hold down button "KEY_PLAYPAUSE".

Please enter the name for the next button (press <ENTER> to finish recording)
```

## Lesson 5 - 16x2 LCD with the Raspberry Pi

### Overview

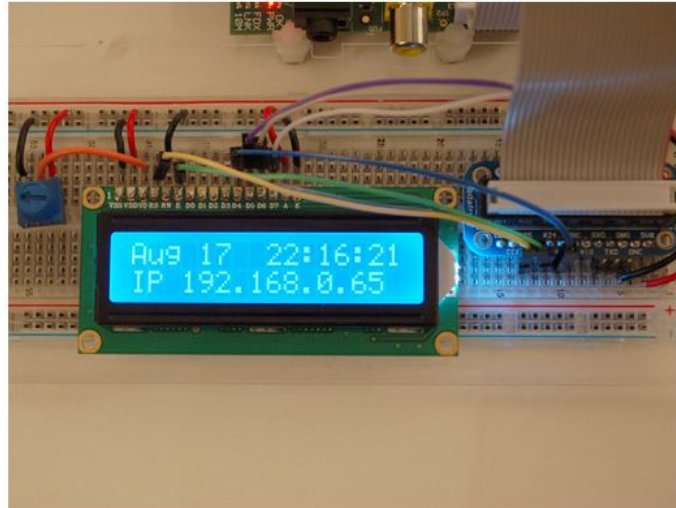


Adding a LCD to any project immediately kicks it up a notch. This tutorial explains how to connect a inexpensive HDD44780 compatible LCD to the raspberry pi using 6 GPIOs. While there are other ways to connect using I2C or the UART this is the most direct method that get right down to the bare metal.

This technique:

- allows for **inexpensive LCDs** to be used
- does not require any **i2c drivers**
- won't steal the only **serial port** on the Pi.

The example python code sends date, time and the IP address to the display. If you are running a Pi in headless mode being able to determine the IP address at a glance is really handy.



## Wiring the Cobbler to the LCD



You can use nearly any character LCD with this tutorial - it will work with 16x1, 16x2, 20x2, 20x4 LCDs. It will not work with 40x4 LCDs

### The LCD



Whenever you come across a LCD that looks like it has 16 connectors it is most likely using a HD44780 controller. These devices provide the same pinouts making them relatively easy to work with. The LCD uses a parallel interface meaning that we will need many pins from our raspberry pi to control it. In this tutorial we will use 4 data pins (4-bit mode) and two control pins.



The **data pins** are straight forward. They are sending data to the display (toggled high/low). We will only be using write mode and not reading any data.

The **register select** pin has two uses. When pulled low it can send commands to the LCD (like position to move to, or clear the screen). This is referred to as writing to the instruction or command register. When toggled the other way (1) the register select pin goes into a data mode and will be used to send data to the screen.

The **read/write** pin will be pulled low (write only) as we only want to write to the LCD based on this setup.

The **enable** pin will be toggled to write data to the registers.

## LCD Pinout

---

1. Ground
2. VCC - **5v not 3.3v**
3. Contrast adjustment (VO) from potentiometer
4. Register Select (RS). RS=0: Command, RS=1: Data
5. Read/Write (R/W). R/W=0: Write, R/W=1: Read (**we won't use this pin**)
6. Clock (Enable). Falling edge triggered
7. Bit 0 (**Not used in 4-bit operation**)
8. Bit 1 (**Not used in 4-bit operation**)
9. Bit 2 (**Not used in 4-bit operation**)
10. Bit 3 (**Not used in 4-bit operation**)
11. Bit 4
12. Bit 5
13. Bit 6
14. Bit 7
15. Backlight LED Anode (+)
16. Backlight LED Cathode (-)

Before wiring, check that your LCD has an LED backlight, not an EL backlight. LED backlights use 10-40mA of power, EL backlights use 200+ma! EL backlights are often cheap to get but are not usable, make sure you don't use one or you will overload the Pi. Some cheap LCDs that have LED backlights do not include a resistor on the LCD module for the backlight, if you're not sure, connect a 1Kohm resistor between pin 15 and 5V instead of connecting directly.



---

## Wiring Diagram

---

First, connect the cobbler power pins to the breadboard power rail. +5.0V from the cobbler goes to the red striped rail (red wire) and GND from the cobbler goes to the blue striped rail (black wire)

In order to send data to the LCD we are going to wire it up as follows

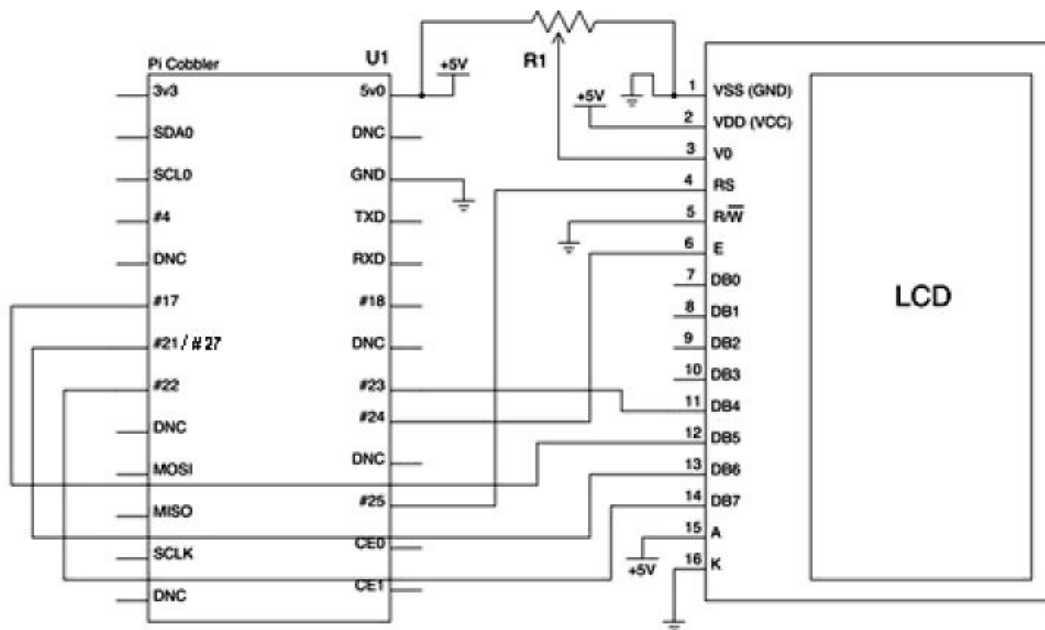
- Pin #1 of the LCD goes to ground
- Pin #2 of the LCD goes to +5V
- Pin #3 (Vo) connects to the middle of the potentiometer
- Pin #4 (RS) connects to the Cobbler P6 ( GPIO # 25 )
- Pin #5 (RW) goes to ground GND
- Pin #6 (EN) connects to cobbler P5 ( GPIO # 24 )
- Skip LCD Pins #7, #8, #9 and #10
- Pin #11 (D4) connects to cobbler P4 ( GPIO # 23 )
- Pin #12 (D5) connects to cobbler P0 ( GPIO # 17 )
- Pin #13 (D6) connects to cobbler P2 ( GPIO # 21 / GPIO # 27 )
- Pin #14 (D7) connects to cobbler P3 ( GPIO # 22 )
- Pin #15 (LED +) goes to +5V (red wire)
- Pin #16 (LED -) goes to ground (black wire)

Then connect up the potentiometer, the left pin connects to ground (black wire) and the right pin connects to +5V (red wire)

---

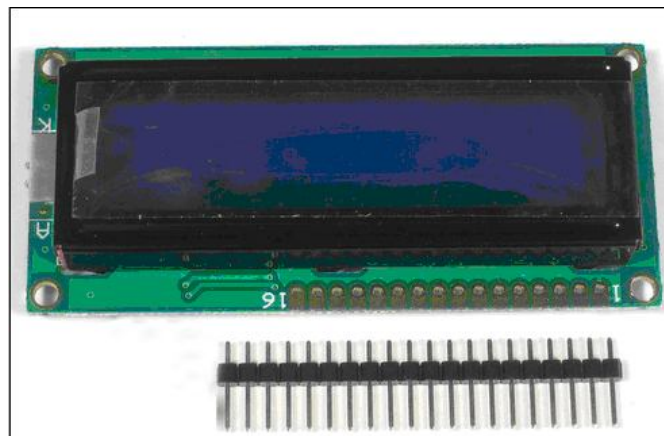
## Schematic

---

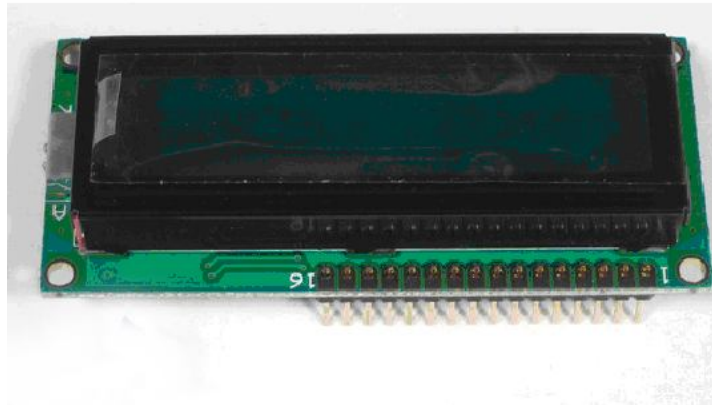


## Preparing the LCD

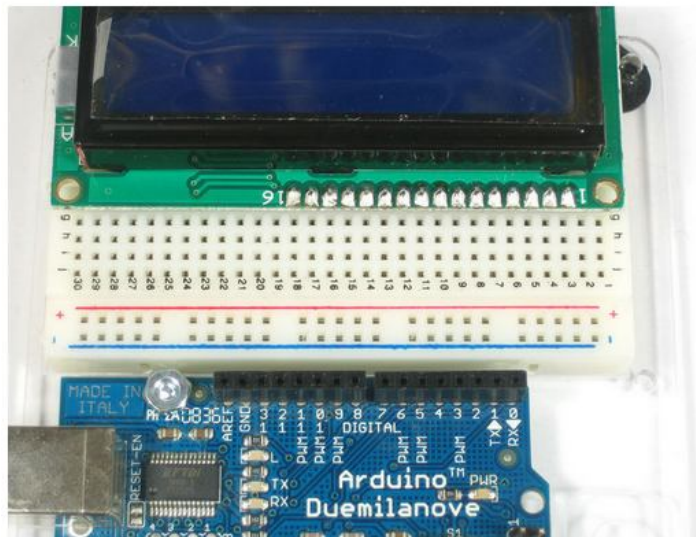
Before you start, make sure you have a strip of 0.1" male header and a 10K potentiometer. All Adafruit Character LCDs come with these parts so you should be good to go.



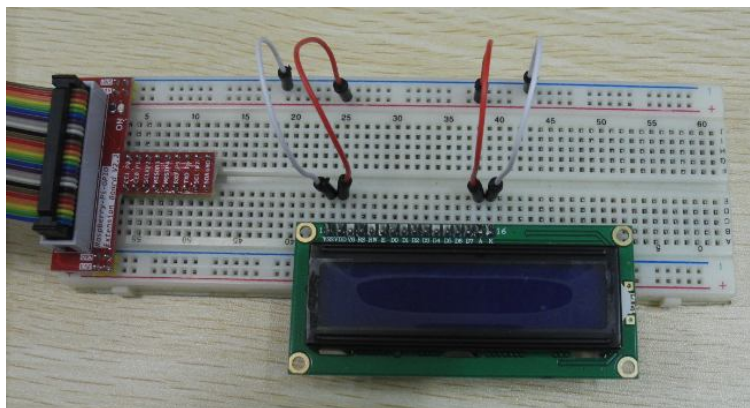
Most LCDs have a strip of 16 pins on the top, if the header is a little longer, just break it off until its the right length

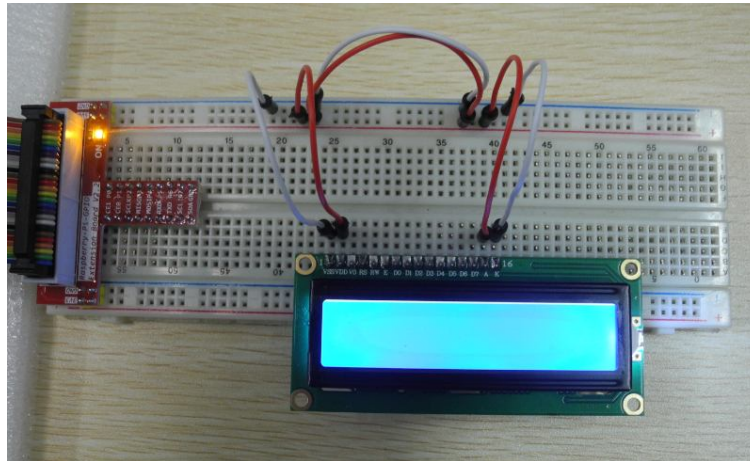


Next you'll need to solder the header to the LCD. !!! You must do this, it is not OK to just try to 'press fit' the LCD!



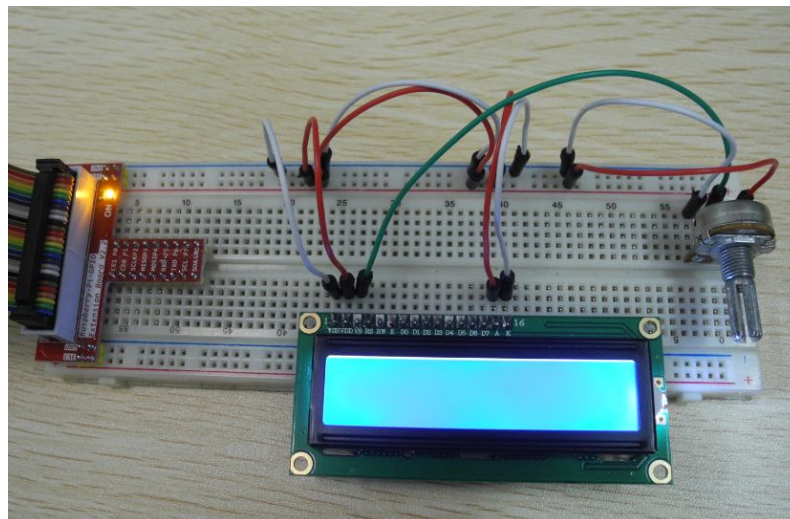
Start by connecting the **5V** and **GND** wires from the cobbler to the breadboard. Then connect pins #1, #2 and #15, #16 to the breadboard power rails as shown. The backlight should come on. If it doesn't, check the wiring!



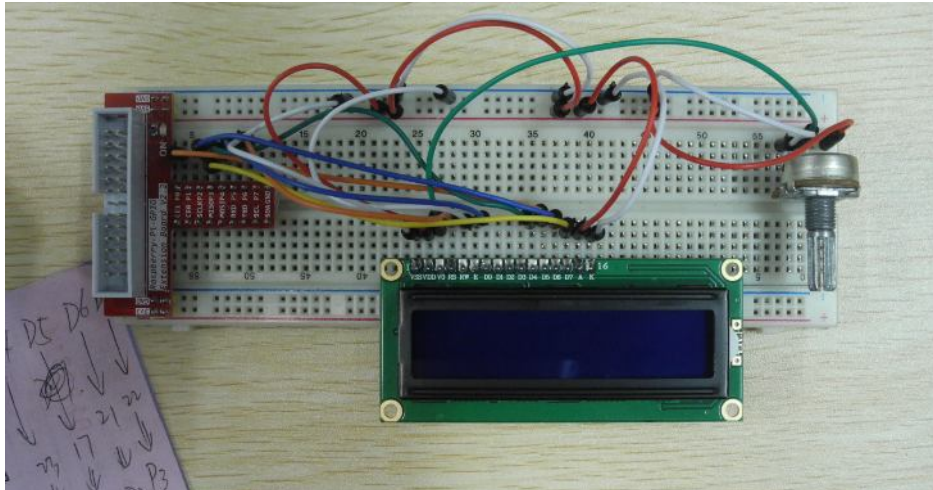


Next, wire up the contrast potentiometer as shown above, with the middle pin connecting to LCD pin #3 and the other two pins going to 5V and ground.

Twist the potentiometer until you see the first line of the LCD fill with boxes. If you don't see the boxes, check your wiring!



Finish the wiring for the RS, RW, EN, D4, D5, D6, and D7 pins as shown in the diagram up top



That's it! Now you're ready to run the python script to draw text on the screen!

## Necessary Packages

This guide is based on Debian's "Wheezy" release for Raspberry Pi. It was made available in Mid July 2012. The following items must be installed in order to utilize the Raspberry Pi's GPIO pins. If you are running [Adafruit's Occidentalis](#) you can just skip this page.

Add the latest dev packages for Python (2.x)

```
$ sudo apt-get install python-dev
```

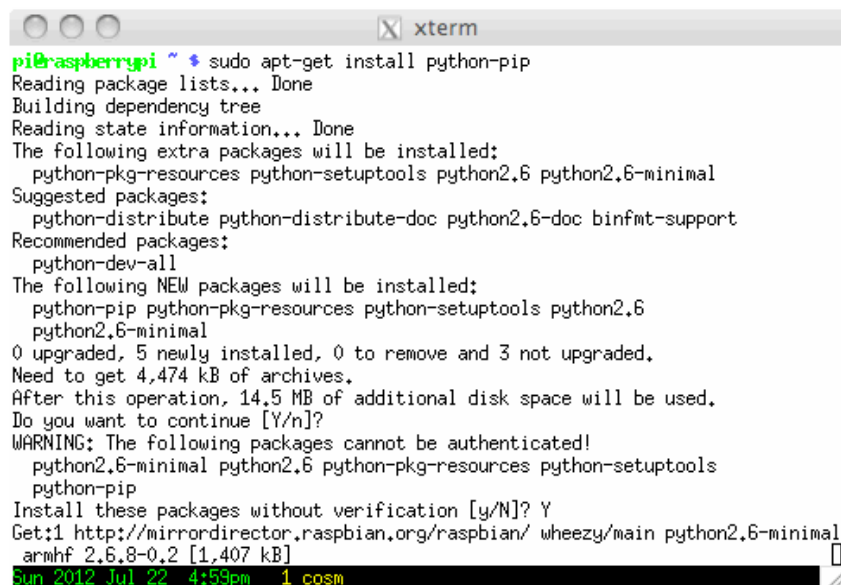
```
pi@raspberrypi ~ $ sudo apt-get install python-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  python2.7-dev
The following NEW packages will be installed:
  python-dev python2.7-dev
0 upgraded, 2 newly installed, 0 to remove and 3 not upgraded.
Need to get 28.4 MB of archives.
After this operation, 35.4 MB of additional disk space will be used.
Do you want to continue [Y/n]? Y
WARNING: The following packages cannot be authenticated!
  python2.7-dev python-dev
Install these packages without verification [y/N]? Y
```

```
Sun 2012 Jul 22 5:05pm 0 notes 1 pi
Sun 2012 Jul 22 5:05pm 1 cosm
```

Upgrade distribute (required for RPi.GPIO 0.3.1a) - [No image for this one]



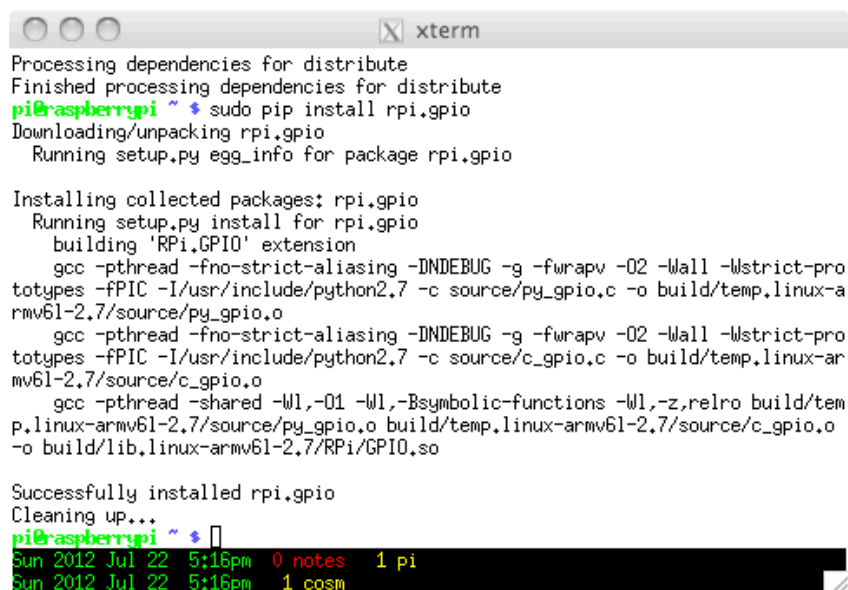
```
$ sudo apt-get install python-setuptools
$ sudo easy_install -U distribute
$ sudo apt-get install python-pip
```



```
pi@raspberrypi ~ $ sudo apt-get install python-pip
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  python-pkg-resources python-setuptools python2.6 python2.6-minimal
Suggested packages:
  python-distribute python-distribute-doc python2.6-doc binfmt-support
Recommended packages:
  python-dev-all
The following NEW packages will be installed:
  python-pip python-pkg-resources python-setuptools python2.6
  python2.6-minimal
0 upgraded, 5 newly installed, 0 to remove and 3 not upgraded.
Need to get 4,474 kB of archives.
After this operation, 14,5 MB of additional disk space will be used.
Do you want to continue [Y/n]?
WARNING: The following packages cannot be authenticated!
  python2.6-minimal python2.6 python-pkg-resources python-setuptools
  python-pip
Install these packages without verification [y/N]? Y
Get:1 http://mirrordirector.raspbian.org/raspbian/ wheezy/main python2.6-minimal
  armhf 2.6.8-0.2 [1,407 kB]
Sun 2012 Jul 22 4:59pm 1 cosm
```

Install rpi.gpio (0.3.1a) or later

```
$ sudo pip install rpi.gpio
```



```
Processing dependencies for distribute
Finished processing dependencies for distribute
pi@raspberrypi ~ $ sudo pip install rpi.gpio
Downloading/unpacking rpi.gpio
  Running setup.py egg_info for package rpi.gpio

Installing collected packages: rpi.gpio
  Running setup.py install for rpi.gpio
    building 'Rpi.GPIO' extension
      gcc -pthread -fno-strict-aliasing -DNDEBUG -g -fwrapv -O2 -Wall -Wstrict-pro
totypes -fPIC -I/usr/include/python2.7 -c source/py_gpio.c -o build/temp.linu
x-armv6l-2.7/source/py_gpio.o
      gcc -pthread -fno-strict-aliasing -DNDEBUG -g -fwrapv -O2 -Wall -Wstrict-pro
totypes -fPIC -I/usr/include/python2.7 -c source/c_gpio.c -o build/temp.linu
x-armv6l-2.7/source/c_gpio.o
      gcc -pthread -shared -Wl,-O1 -Wl,-Bsymbolic-functions -Wl,-z,relro build/tem
p.linu
x-armv6l-2.7/source/py_gpio.o build/temp.linu
x-armv6l-2.7/source/c_gpio.o
      -o build/lib.linu
x-armv6l-2.7/RPi/GPIO.so

Successfully installed rpi.gpio
Cleaning up...
pi@raspberrypi ~ $
Sun 2012 Jul 22 5:16pm 0 notes 1 pi
Sun 2012 Jul 22 5:16pm 1 cosm
```

## Python Script

The Python code for Adafruit's Character LCDs on the Pi is available on Github at

<https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code>



There are two files we will be working with:

1. **Adafruit\_CharLCD.py**- contains python class for LCD control
2. **Adafruit\_CharLCD\_IPclock\_example.py** - code for IP address and date/time calls methods from Adafruit\_CharLCD.py

The first file **Adafruit\_CharLCD.py** is a mashup from two different sources of LCD code. Github user lrvick put together a [nice python class](#). His work is the baseline that is slowly being changed as we are bringing in more elements from the [Arduino LiquidCrystal library](#).

The easiest way to get the code onto your Pi is to hook up an Ethernet cable, and clone it directly using 'git', which is installed by default on most distros. Simply run the following commands from an appropriate location (ex. "/home/pi"):

```
apt-get install git
git clone git://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code.git
cd Adafruit-Raspberry-Pi-Python-Code
cd Adafruit_CharLCD
```

## Testing

You can test the wiring from the previous step by simply running the **Adafruit\_CharLCD.py** python code, as it has a little code in it that will simply display a test message when wired correctly

If you're using a Version 2 Raspberry Pi, pin **#21** has been replaced with pin **#27** so edit Adafruit\_CharLCD.py and change:

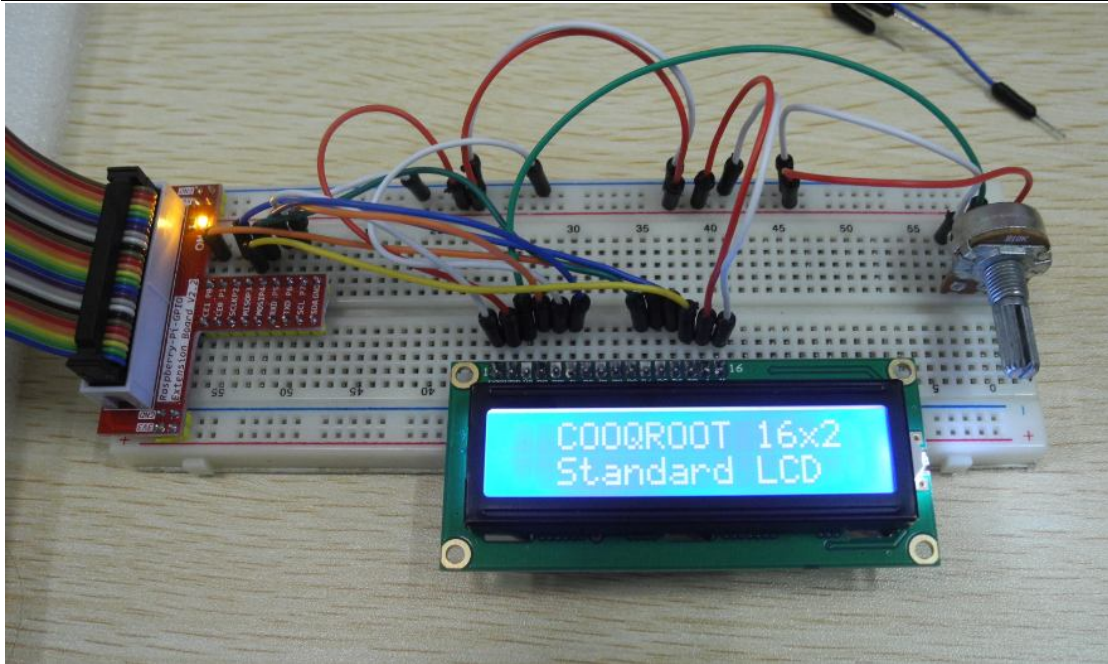
```
def __init__(self, pin_rs=25, pin_e=24, pins_db=[23, 17, 21, 22], GPIO = None):
```

to

```
def __init__(self, pin_rs=25, pin_e=24, pins_db=[23, 17, 27, 22], GPIO = None):
```

you can use **nano Adafruit\_CharLCD.py** to edit

```
chmod +x Adafruit_CharLCD.py
sudo python ./Adafruit_CharLCD.py
```



## IP Clock Example



This script assumes you'll want to display the Ethernet (eth0) IP address. You may want to replace eth0 with wlan0 or wlan1 (etc) for Wireless IP address!

```
#!/usr/bin/python

from Adafruit_CharLCD import Adafruit_CharLCD
from subprocess import *
from time import sleep, strftime
from datetime import datetime

lcd = Adafruit_CharLCD()

cmd = "ip addr show eth0 | grep inet | awk '{print $2}' | cut -d/ -f1"

lcd.begin(16,1)

def run_cmd(cmd):
    p = Popen(cmd, shell=True, stdout=PIPE)
    output = p.communicate()[0]
    return output

while 1:
```

```

lcd.clear()

ipaddr = run_cmd(cmd)

lcd.message(datetime.now().strftime('%b %d %H:%M:%S\n'))

lcd.message('IP %s' % ( ipaddr ) )

sleep(2)

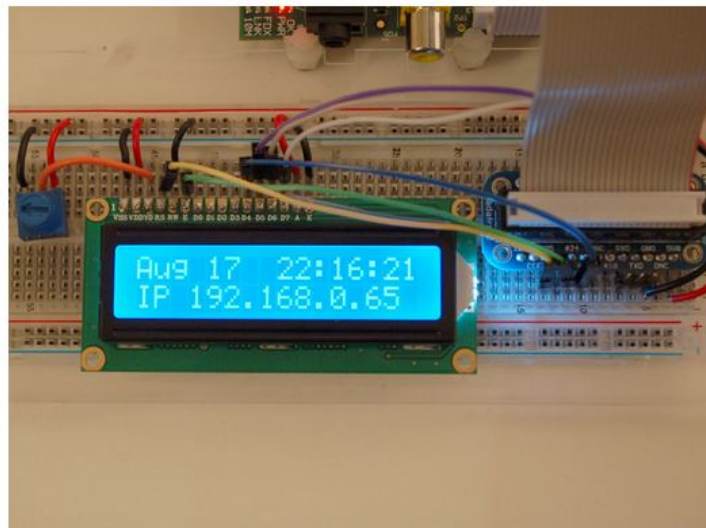
```

## Running the Code

Start the IP + Clock example

```
$ sudo python ./Adafruit_CharLCD_IPclock_example.py
```

## What You Should See



## Init Script

It's all fine and dandy to have a script like `Adafruit_CharLCD_IPclock_example.py` which we can manually run, but wouldn't it be nice to have the time and ip address pop up on the display when the raspberry pi boots up? This is done using a init script which runs the `Adafruit_CharLCD_IPclock_example.py` code on boot and kills it during system shut down.

Paste this code into `/etc/init.d/lcd`

(you will need to use `sudo` to write to this directory)

```
### BEGIN INIT INFO
```

```
# Provides: LCD - date / time / ip address
# Required-Start: $remote_fs $syslog
# Required-Stop: $remote_fs $syslog
# Default-Start: 2 3 4 5
# Default-Stop: 0 1 6
# Short-Description: Liquid Crystal Display
# Description: date / time / ip address
### END INIT INFO

#!/bin/sh

# /etc/init.d/lcd

export HOME
case "$1" in
    start)
        echo "Starting LCD"
        /home/pi/Adafruit-Raspberry-Pi-Python-Code/Adafruit_CharLCD/Adafruit_CharLCD_IPclock_example.py
2>&1 &
        ;;
    stop)
        echo "Stopping LCD"
        LCD_PID=`ps auxwww | grep Adafruit_CharLCD_IPclock_example.py | head -1 | awk '{print $2}`
        kill -9 $LCD_PID
        ;;
    *)
        echo "Usage: /etc/init.d/lcd {start|stop}"
        exit 1
        ;;
esac
exit 0
```

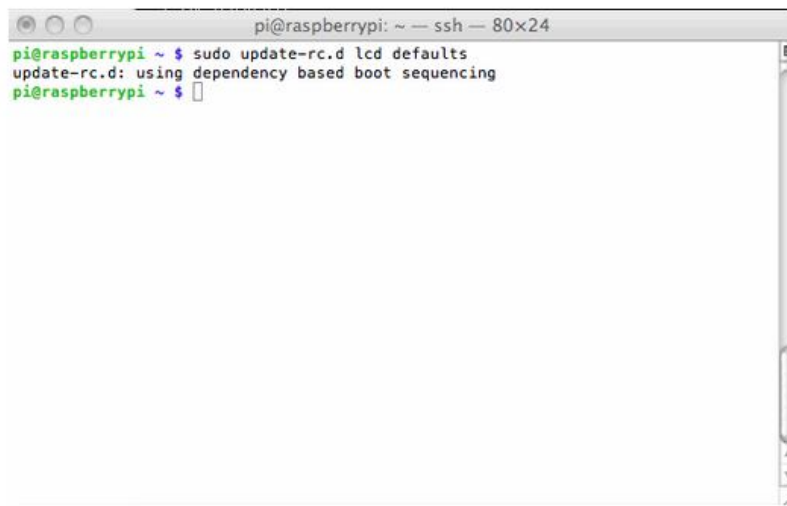
You should change  
**/home/pi/Adafruit-Raspberry-Pi-Python-Code/Adafruit\_CharLCD/Adafruit\_CharLCD\_IPclock\_example.py** to where-ever you are actually keeping the IPclock python script

Make the init script executable.

```
$ sudo chmod +x /etc/init.d/lcd
```

Make the lcd init script known to the system by using the update-rc.d command.

```
$ sudo update-rc.d lcd defaults
```

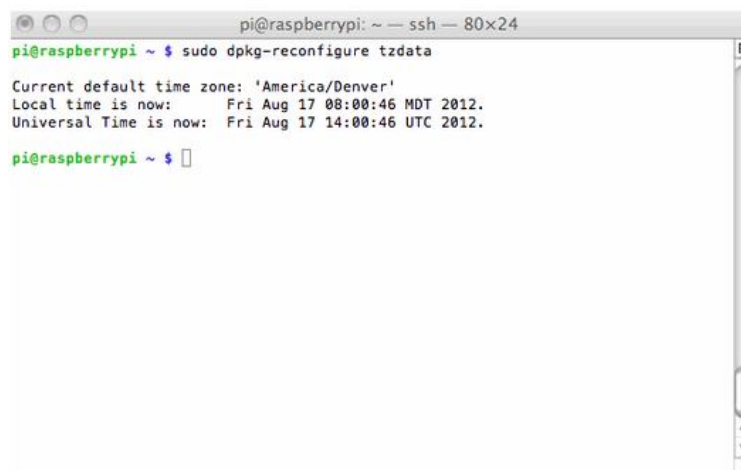


```
pi@raspberrypi: ~ — ssh — 80x24
pi@raspberrypi ~ $ sudo update-rc.d lcd defaults
update-rc.d: using dependency based boot sequencing
pi@raspberrypi ~ $
```

## Time Zone

Last, but not least. My pi came configured with UT (Universal Time). I prefer to see time based on my time zone which is Mountain. Here is how to configure time on the pi for any location. This is a one time configuration setting that will survive between reboots.

```
$ sudo dpkg-reconfigure tzdata
```



```
pi@raspberrypi: ~ — ssh — 80x24
pi@raspberrypi ~ $ sudo dpkg-reconfigure tzdata
Current default time zone: 'America/Denver'
Local time is now:      Fri Aug 17 08:00:46 MDT 2012.
Universal Time is now:  Fri Aug 17 14:00:46 UTC 2012.
pi@raspberrypi ~ $
```

The command launches a curses based program which allows arrow keys to be used to select the region specific time zone.

