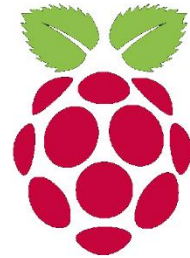
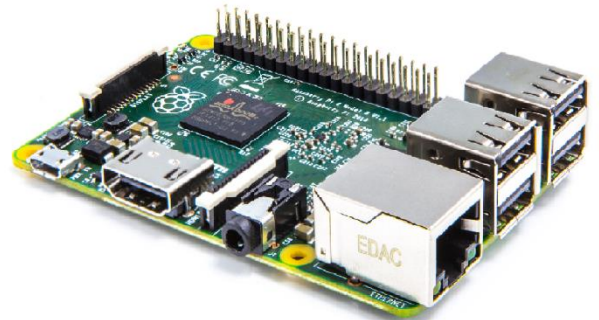
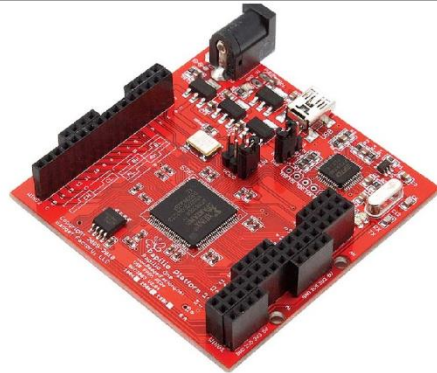
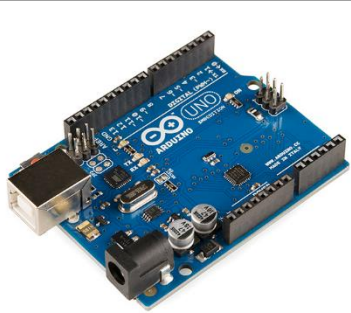


CURSO 2016-2017

# MEMORIA DE PRÁCTICAS LABORATORIO DE DESARROLLO DE HARDWARE



Autor: Carlos Crespo Jiménez

NIF: 28785141-C

Grado de Ingeniería de Computadores  
Memoria de Prácticas de Laboratorio de  
Desarrollo de Hardware.

# Índice

<b>Plataforma Raspberry PI .....</b>	<b>3</b>
1.1   Objetivos .....	3
1.2   Introducción .....	3
1.3   Instalación y configuración de Ubuntu Mate en la tarjeta microSD .....	4
1.4   Manejo de GPIOs desde la línea de comandos.....	7
1.5   Control de LEDs con Python .....	8
1.6   Conexión con la Raspberry Pi mediante SSH .....	9
1.7   Instalación servidor web para manejar los LEDs.....	10
1.8   Lectura remota de la temperatura .....	13
1.9   Lectura remota de la temperatura y relé con bombilla LED .....	18
1.10   Conclusión .....	20

## 1.1 Objetivos

Los objetivos mínimos en la plataforma Raspberry PI son:

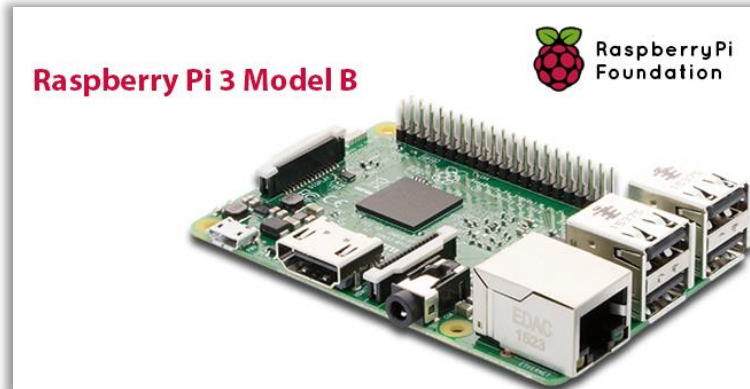
- Preparar la plataforma Raspberry PI para que puedan cargarse diferentes versiones de Sistema Operativo.
- Arrancar y comprobar el funcionamiento de la placa Raspberry PI.
- Desarrollar ejemplos de utilización de los pines de expansión GPIOs.
- Instalar un servidor web que pueda ejecutar código Python.

## 1.2 Introducción



¿Qué es la Raspberry PI?

Raspberry Pi es una placa computadora (SBC) de bajo costo desarrollada en Reino Unido por la Fundación Raspberry Pi, con el objetivo de estimular la enseñanza de ciencias de la computación en las escuelas.



Las características del modelo (Raspberry PI 3) que vamos a utilizar este año en las prácticas son las siguientes:

- ✓ Procesador: Chipset Broadcom BCM2387. 1,2 GHz de cuatro núcleos ARM Cortex-A53.
- ✓ GPU. Dual Core VideoCore IV ® Multimedia Co-procesador.
- ✓ RAM: 1GB LPDDR2.
- ✓ Ethernet socket Ethernet 10/100 BaseT. 802.11 b / g / n LAN inalámbrica y Bluetooth 4.1 (Classic Bluetooth y LE).



## ¿Qué sistema operativo (S.O) vamos a utilizar?

Utilizaremos este año la versión 16.04 de Ubuntu Mate. Ubuntu MATE es una distribución Linux basada en Ubuntu. Está mantenida por la comunidad y es un derivado de Ubuntu oficialmente reconocido por Canonical, usando el entorno de escritorio MATE.

El proyecto Ubuntu MATE fue fundado por Martin Wimpress y Alan Pope, y comenzó como un derivado no oficial de Ubuntu, usando como base a Ubuntu 14.10 para su primer lanzamiento; el lanzamiento 14.04 LTS le siguió pronto. En febrero de 2015, Ubuntu MATE fue reconocido por Canonical Ltd. como sabor oficial de Ubuntu en el lanzamiento de 15.04 Beta 1, la versión 14.04 LTS no es considerada distribución oficial de Ubuntu. La versión 16.04 LTS tiene imágenes creadas específicamente para dispositivos Raspberry Pi 2 y Raspberry Pi 3.

### 1.3 Instalación y configuración de Ubuntu Mate en la tarjeta microSD

Como ya hemos explicado, la Raspberry Pi no tiene disco duro propio, por lo que hay que instalar dicho sistema operativo en una tarjeta microSD. Para ello, necesitaremos descargar una imagen del sistema operativo en la siguiente URL: <https://coria.dte.us.es/~bellido> → Descargaremos el archivo de nombre ubuntu-mate-16.04-desktop-armhf-raspberry-pi-resize.img (ocupa 4.3 GBytes).

Posteriormente, prepararemos nuestra tarjeta microSD siguiendo las instrucciones del siguiente tutorial: <https://www.raspberrypi.org/documentation/installation/installing-images/linux.md>.

Primero, desmontaremos la tarjeta SD de la siguiente forma tal como viene explicado en el anterior tutorial (en nuestro caso es /dev/sdb2):

```
practicass@MCR-92:~$ umount /dev/sdb1
No se ha encontrado la orden «umount», quizás quiso decir:
La orden «umount» del paquete «mount» (main)
umount: no se encontró la orden
practicass@MCR-92:~$ umount /dev/sdb1
umount: /dev/sdb1 no está montado (según mtab)
practicass@MCR-92:~$ umount /dev/sdb2
practicass@MCR-92:~$
```

Una vez esté desmontado todos los archivos que no pueden ser leídos o escritos en la tarjeta microSD, procederemos a cargar la imagen del sistema operativo descargada de la versión 16.04 de Ubuntu Mate en nuestra tarjeta microSD:

```
practicass@mcr-92: ~/Descargas
practicass@mcr-92:~$ cd Descargas
practicass@mcr-92:~/Descargas$ ls
b374k-master.zip
eagle-lin64-7.5.0.run
passwd
sockets-1.2.0.tar.gz
ubuntu-mate-16.04-desktop-armhf-raspberry-pi-resize.img
ubuntu-mate-16.04-desktop-armhf-raspberry-pi-resize.img.part
practicass@mcr-92:~/Descargas$ sudo dd bs=4M if=ubuntu-mate-16.04-desktop-armhf-r
aspberry-pi-resize.img of=/dev/sdb
288+0 registros leídos
288+0 registros escritos
1207959552 bytes (1,2 GB) copiados, 52,3116 s, 23,1 MB/s
1110+1 registros leídos
1110+1 registros escritos
4658233856 bytes (4,7 GB) copiados, 401,808 s, 11,6 MB/s
practicass@mcr-92:~/Descargas$
```

Si durante la instalación de la imagen en la tarjeta SD quisiéramos ver cómo va dicha instalación, podremos abrir otro terminal y escribir el siguiente comando:

```
sudo pkill -USR1 -n -x dd
```

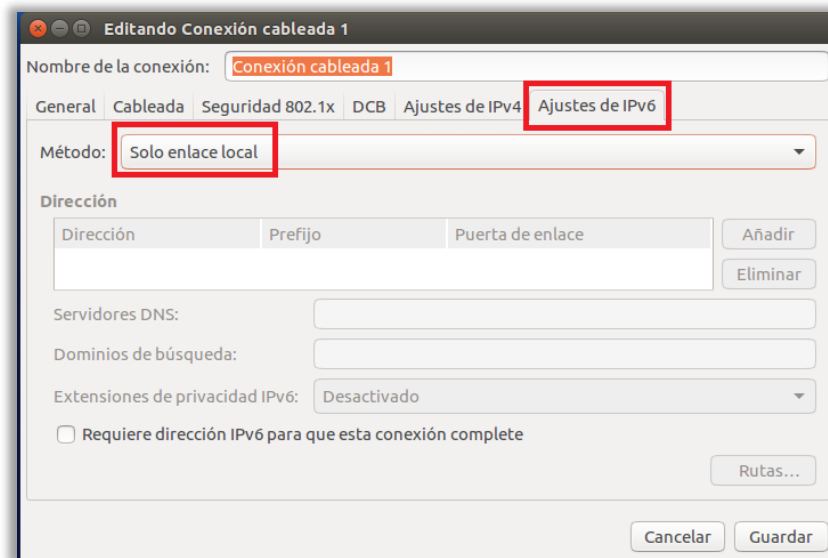
Una vez ya preparada la tarjeta SD, debemos realizar los siguientes pasos:

1. Conectar la tarjeta microSD a la Raspberry PI.
2. Desconectar teclado, ratón y cable Ethernet del PC y conectarlo todo a la Raspberry PI.
3. Conectar el cable HDMI-DVI a la Raspberry PI con el monitor del PC.
4. Por último, conectar el cable micro-USB a la Raspberry PI con el PC para alimentar la Raspberry PI.

A continuación, seguiremos los pasos en el arranque configurando el idioma, el teclado, la localización, y algo muy importante, la fecha y la hora del sistema.

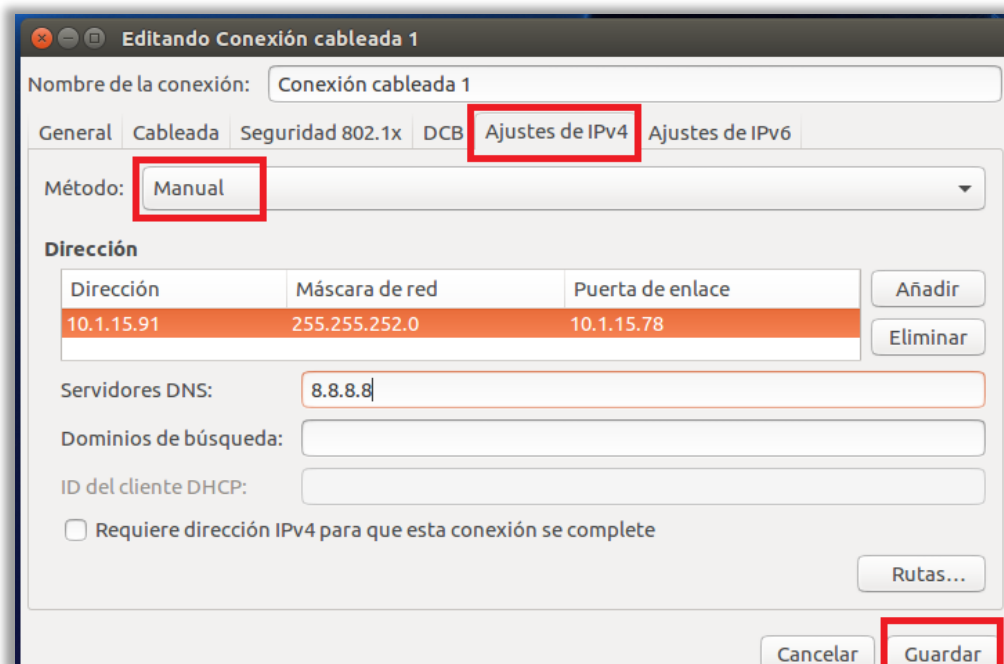


Configuraremos la red manualmente:



Los datos correctos son:

- ✓ Dirección IP (Mirar etiqueta de nuestro PC del Laboratorio): 10.1.15.91
- ✓ Máscara de red: 255.255.252.0
- ✓ Puerta de enlace: 10.1.15.78
- ✓ Servidor DNS: 8.8.8.8 (Google)
- ✓ Pulsaremos Guardar para finalizar.



En el primer arranque de Ubuntu Mate es cuando se realiza una configuración del sistema y se completa la instalación del sistema de ficheros. No nos debemos de olvidar pulsar el botón de reajustar el tamaño del sistema de ficheros al tamaño de la imagen desde la página de bienvenida (welcome) de Ubuntu Mate.

## 1.4 Manejo de GPIOs desde la línea de comandos



### ¿Qué son los puertos GPIOs?

Son unos pines genéricos en un chip, cuyo comportamiento (incluyendo si son de entrada o salida) se pueden controlar por el usuario en tiempo de ejecución. Los pines GPIO no tienen ningún propósito especial definido, y no se utilizan de forma predeterminada. En el caso de una Raspberry Pi pueden usarse para controlar el encendido y apagado de una bombilla, para recibir el comportamiento de un determinado botón.



### Desarrollo

Seguiremos las instrucciones del siguiente tutorial: <https://geekytheory.com/tutorial-raspberry-pi-gpio-parte-1-control-de-un-led/>.

El objetivo de este ejercicio es poder controlar un led a través de los pines GPIOs mediante un nuevo terminal en nuestra computadora. Los comandos para la realización de este ejercicio son los siguientes:

**1) Creamos nuestro pin GPIO en el numero 17:**

```
echo 17 > /sys/class/gpio/export
```

**2) Declaramos nuestro pin como salida:**

```
echo out > /sys/class/gpio/gpio17/direction
```

**3) Encendemos led, Apagamos led:**

```
echo 1 > /sys/class/gpio/gpio17/value ENCENDEMOS
```

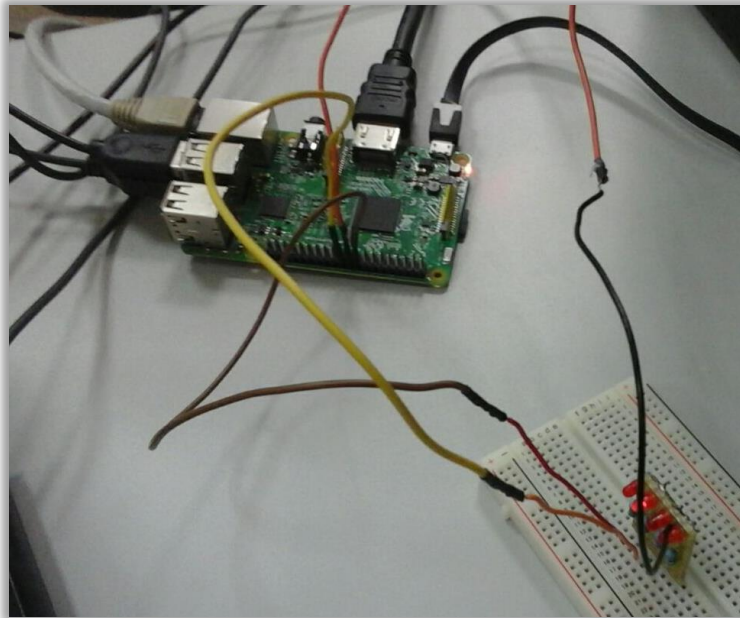
```
echo 0 > /sys/class/gpio/gpio17/value APAGAMOS
```

**4) Eliminamos nuestro pin GPIO (no es que lo quitemos de la placa)**

```
echo 17 > /sys/class/gpio/unexport
```

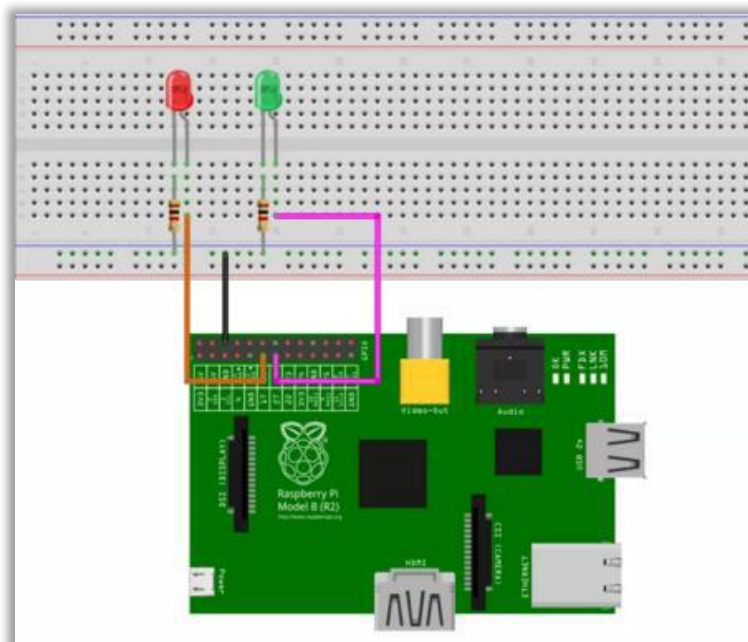


El LED irá conectado a GND y al GPIO 17 de la placa de la siguiente forma:



## 1.5 Control de LEDs con Python

Ahora vamos a encender y/o apagar dos LEDs usando código Python. Para ello seguiremos las instrucciones del siguiente tutorial: <https://geekytheory.com/tutorial-raspberry-pi-gpio-parte-2-control-de-leds-con-python/>.





Como muestra la imagen superior, los pines serán el 17 y el 27, una vez esté todo conectado correctamente, pasaremos a escribir nuestro código Python para poderlo ejecutar. Para comprobar que funciona tal y como queremos, nos vamos a nuestro terminal y tecleamos: `sudo python blink.py`.

 Código Python

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setup(17, GPIO.OUT) ## GPIO 17 como salida GPIO.setup(27, GPIO.OUT) ## GPIO 27 como salida
def blink():
    print "Ejecucion iniciada..."
    iteracion = 0
    while iteracion < 30: ## Segundos que durara la funcion
        GPIO.output(17, True) ## Enciendo el 17
        GPIO.output(27, False) ## Apago el 27
        time.sleep(1) ## Esperamos 1 segundo
        GPIO.output(17, False) ## Apago el 17
        GPIO.output(27, True) ## Enciendo el 27
        time.sleep(1) ## Esperamos 1 segundo
        iteracion = iteracion + 2 ## Sumo 2 porque he hecho dos parpadeos
    print "Ejecucion finalizada"
    GPIO.cleanup() ## Hago una limpieza de los GPIO
blink() ## Hago la llamada a la funcion blink
```

## 1.6 Conexión con la Raspberry Pi mediante SSH



¿Qué es el protocolo SSH?

SSH (Secure SHell, en español: intérprete de órdenes seguro) es el nombre de un protocolo y del programa que lo implementa, y sirve para acceder a máquinas remotas a través de una red. Permite manejar por completo la computadora mediante un intérprete de comandos, y también puede redirigir el tráfico de X para poder ejecutar programas gráficos si tenemos ejecutando un Servidor X (en sistemas Unix y Windows).

Además de la conexión a otros dispositivos, SSH nos permite copiar datos de forma segura (tanto archivos sueltos como simular sesiones FTP cifradas), gestionar claves RSA para no escribir claves al conectar a los dispositivos y pasar los datos de cualquier otra aplicación por un canal seguro tunelizado mediante SSH.

 Desarrollo:

Vamos a desarrollar como podemos manejar nuestra placa sin necesidad de usar ningún tipo de entorno gráfico y sin tener contacto físico con la placa (control remoto). Para ello necesitaremos conocer el protocolo SSH, que es el que vamos a utilizar en los

siguientes ejercicios. Dicho protocolo SSH lo utilizaremos para comunicarnos con la placa vía Ethernet a través del PC.

Para establecer conexión con el ordenador (PC) mandaremos el siguiente comando por consola (consola del 2º PC):

```
ssh -X "usuario"@IPRaspberryPi -> En mi caso: ssh -X carlos@10.1.15.91
```

Después de ejecutarlo en el terminal nos pedirá la clave que le pusimos al configurar nuestro Ubuntu Mate.

Uno de los aspectos más interesantes de los SBC (Raspberry Pi) es que permiten controlar otros dispositivos hardware (con sensores y /o actuadores). Además, este control puede realizarse de forma remota (a través de internet).

Una forma de hacerlo es montando un servidor web a través del cual podemos tanto enviar datos a los actuadores (por ejemplo: encender y/o apagar luces) como para leer datos de sensores y dar información (temperatura del sensor).

## 1.7 Instalación servidor web para manejar los LEDs



### ¿Qué es el Flask?

Es un framework minimalista escrito en Python y basado en la especificación WSGI de Werkzeug y el motor de templates Jinja2. Tiene la licencia BSD.



### Desarrollo

Para instalar el servidor web necesitaremos instalarnos antes Flask ejecutando en la consola de la Raspberry Pi el siguiente comando:

```
sudo pip install flask
```

Una vez lo tengamos instalado, podemos dar comienzo el montaje de nuestro servidor web. Los pasos a seguir están descritos en el siguiente tutorial: <http://mattrichardson.com/Raspberry-Pi-Flask/index.html>.

Para realizar el ejercicio /tarea de encender y apagar un LED, utilizaremos el código mostrado en el tutorial (WebLamp). Para su ejecución escribiremos en el terminal el siguiente comando:

## Código Python

```
import RPi.GPIO as GPIO
from flask import Flask, render_template, request
app = Flask(__name__)

GPIO.setmode(GPIO.BCM)

# Create a dictionary called pins to store the pin number, name, and pin state:
pins = {
    24 : {'name' : 'coffee maker', 'state' : GPIO.LOW},
    25 : {'name' : 'lamp', 'state' : GPIO.LOW}
}

# Set each pin as an output and make it low:
for pin in pins:
    GPIO.setup(pin, GPIO.OUT)
    GPIO.output(pin, GPIO.LOW)

@app.route("/")
def main():
    # For each pin, read the pin state and store it in the pins dictionary:
    for pin in pins:
        pins[pin]['state'] = GPIO.input(pin)
    # Put the pin dictionary into the template data dictionary:
    templateData = {
        'pins' : pins
    }
    # Pass the template data into the template main.html and return it to the user
    return render_template('main.html', **templateData)

# The function below is executed when someone requests a URL with the pin number and
@app.route("/<changePin>/<action>")
def action(changePin, action):
    # Convert the pin from the URL into an integer:
    changePin = int(changePin)
    # Get the device name for the pin being changed:
    deviceName = pins[changePin]['name']
    # If the action part of the URL is "on," execute the code indented below:
    if action == "on":
        # Set the pin high:
        GPIO.output(changePin, GPIO.HIGH)
        # Save the status message to be passed into the template:
        message = "Turned " + deviceName + " on."
    if action == "off":
        GPIO.output(changePin, GPIO.LOW)
        message = "Turned " + deviceName + " off."
    if action == "toggle":
        # Read the pin and set it to whatever it isn't (that is, toggle it):
        GPIO.output(changePin, not GPIO.input(changePin))
        message = "Toggled " + deviceName + "."

    # For each pin, read the pin state and store it in the pins dictionary:
    for pin in pins:
        pins[pin]['state'] = GPIO.input(pin)

    # Along with the pin dictionary, put the message into the template data dictionary:
    templateData = {
        'message' : message,
        'pins' : pins
    }

    return render_template('main.html', **templateData)
```

Modificaremos “coffee maker” por LED1 y “lamp” por LED2.

#### Código HTML

```
<!DOCTYPE html>
<head>
  <title>Current Status</title>
</head>

<body>
  <h1>Device Listing and Status</h1>

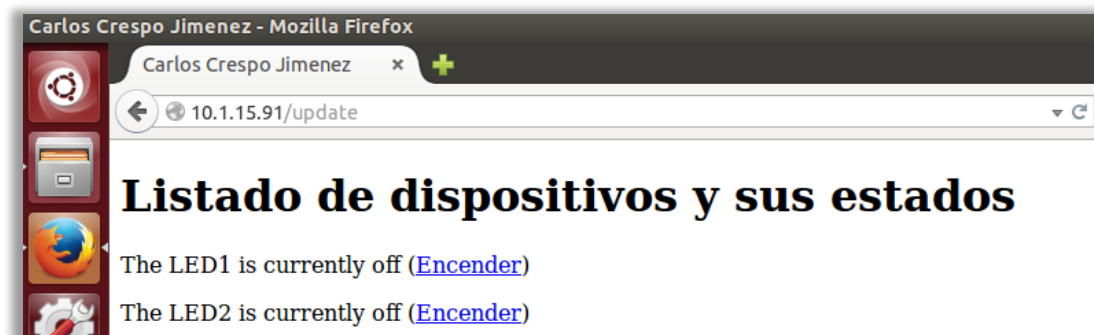
  {% for pin in pins %}
  <p>The {{ pins[pin].name }}
  {% if pins[pin].state == true %}
    is currently on (<a href="/{{pin}}/off">turn off</a>)
  {% else %}
    is currently off (<a href="/{{pin}}/on">turn on</a>)
  {% endif %}
  </p>
  {% endfor %}

  {% if message %}
  <h2>{{ message }}</h2>
  {% endif %}

</body>
</html>
```

Modificaremos el código html para que se pueda entender mejor, tal y como se muestra en la siguiente imagen.

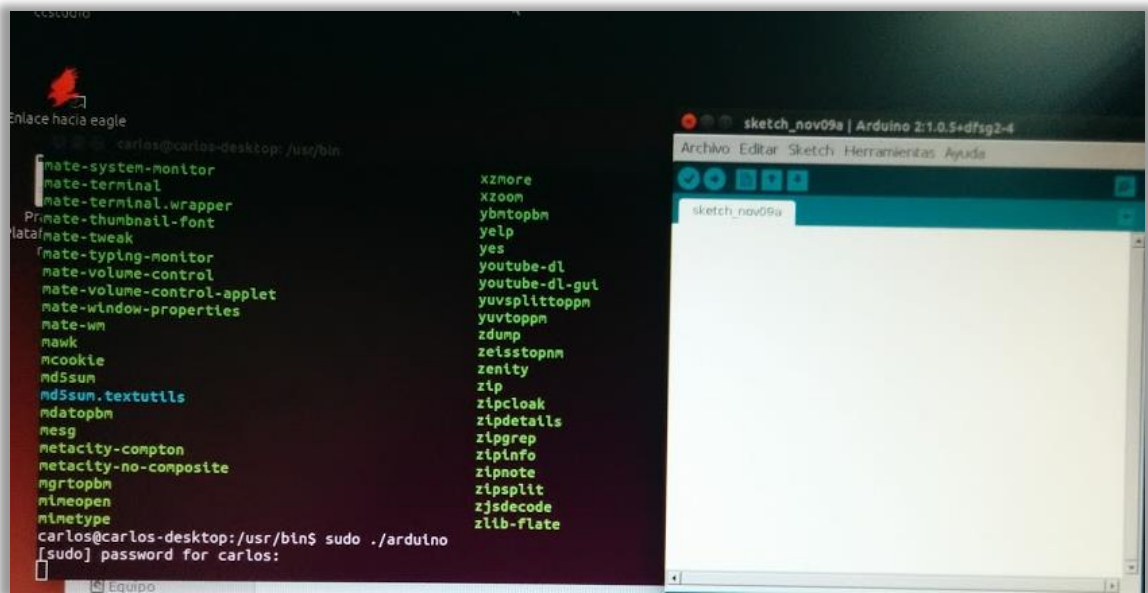
#### Resultado Servidor Web



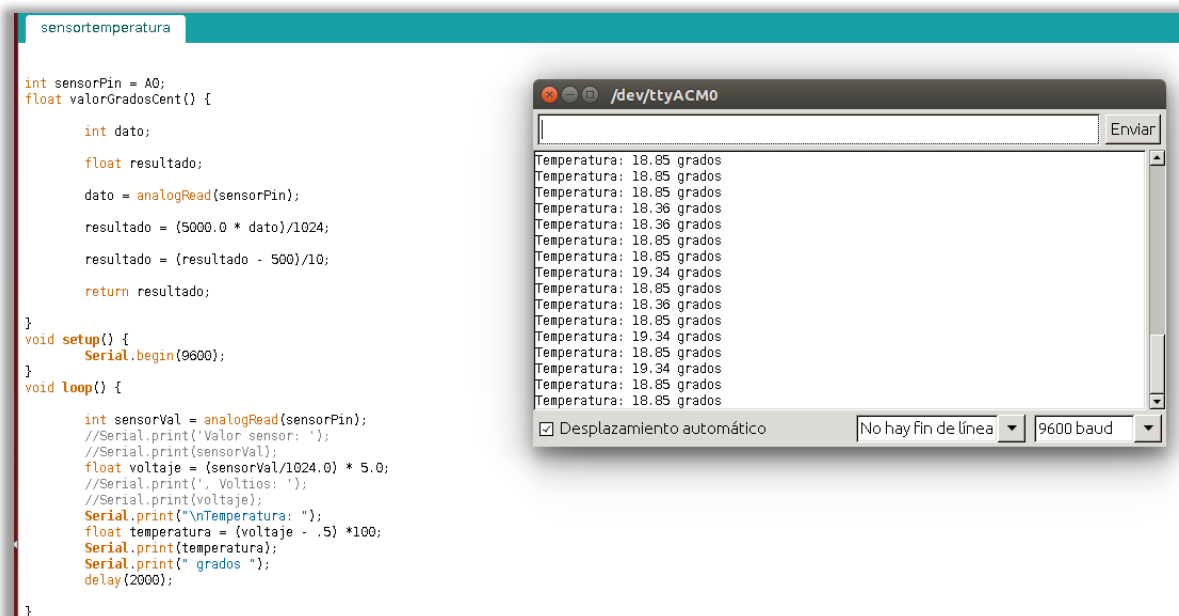
## 1.8 Lectura remota de la temperatura

En este ejercicio vamos a utilizar el código del sensor de la temperatura que utilizamos para Arduino. La Raspberry Pi se encargará de leer la información del sensor de temperatura y la mandará remotamente al PC.

Primero tendremos que instalarnos el IDE de Arduino en la tarjeta microSD de forma remota tal y como viene en la siguiente imagen:



### Código Arduino

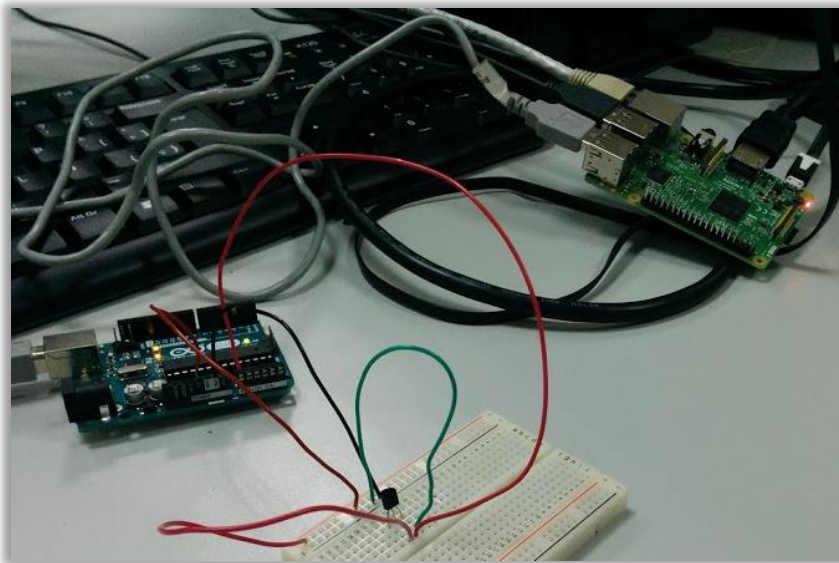


Para enviar los datos de temperatura solo con su valor, sin ninguna cadena extra, escribiremos el siguiente código:

```
int sensorPin = A0;
void setup() {
    Serial.begin(9600);
}

void loop(){
    int sensorVal = analogRead(sensorPin);
    float voltaje = (sensorVal/1024.0) * 5.0;
    Serial.print("\n ");
    float temperatura = (voltaje - .5) *100;
    Serial.print(temperatura);
    delay(2000);
}
```

🔧 Montaje placa Arduino con sensor de temperatura conectada a la Raspberry Pi



🔧 Código Python

```
import RPi.GPIO as GPIO
import serial
from flask import Flask, render_template, request
app = Flask(__name__)

GPIO.setmode(GPIO.BCM)
```

#Create a dictionary called pins to store the pin number, name, and pin state

```
pins = {  
    17: {'name' : 'LED1', 'state' : GPIO.LOW},  
    27: {'name' : 'LED2', 'state' : GPIO.LOW}  
}
```

```
ser = serial.Serial('/dev/ttyUSB0', 9600)
```

# Set each pin as an output and make it low:

for pin in pins:

```
    GPIO.setup(pin, GPIO.OUT)
```

```
    GPIO.output(pin, GPIO.LOW)
```

```
@app.route("/")
```

```
def main():
```

```
    temp = ser.readline()
```

# For each pin, read the pin state and store it in the pins dictionary:

```
templateData = {
```

```
    'pins' : pins,
```

```
    'temps' : temp,
```

```
}
```

#Pass the template data into the template main.html and return it to the user

```
return render_template('main.html', **templateData)
```

```
@app.route("/update")
```

```
def temp():
```

```
    temp = ser.readline()
```

for pin in pins:

```
    pins[pin]['state'] = GPIO.input(pin)
```

```
templateData = {
```

```
    'pins' : pins,
```

```
    'temps' : temp,
```

```
}
```

```
return render_template('main.html', **templateData)
```

# The function below is executed when someone requests a URL with the pin number and action in it:

```
@app.route("/<changePin>/<action>")
```



```

def action(changePin, action):
    changePin = int(changePin)
    deviceName = pins[changePin]['name']
    if action == "on":
        GPIO.output(changePin, GPIO.HIGH)
        message = "Turned " + deviceName + " on."

    if action == "off":
        GPIO.output(changePin, GPIO.LOW)
        message = "Turned " + deviceName + " off."

    if action == "toogle":
        GPIO.output(changePin, not GPIO.input(changePin))
        message = "Toogle " + deviceName + "."

    for pin in pins:
        pins[pin]['state'] = GPIO.input(pin)

#Along with the pin dictionary, put the message into the template data dictionary:

templateData = {
    'message' : message,
    'pins' :pins,
}
return render_template('main.html', **templateData)
if __name__ == "__main__":
    app.run(host='0.0.0.0', port=80, debug=True)

```

#### Código HTML

```

<!DOCTYPE html>
<head>
<title>Carlos Crespo Jimenez</title>
</head>
<body>
    <h1> Listado de dispositivos y sus estados</h1>
    {% for pin in pins %}
    <p> The {{ pins[pin].name }}
    {% if pins[pin].state == true %}

```

```

        is currently on (<a href="/{{pin}}/off">Apagar</a>)
    {% else %}
        is currently off (<a href="/{{pin}}/on">Encender</a>)
    {% endif %}
</p>
{% endfor %}

<p> La temperatura actual es: {{temps}} (<a
href="/update">Actualizame</a> </p>
    {% if message %}
    <h2> {{ message }} </h2>
    {% endif %}
</body>
</html>

```

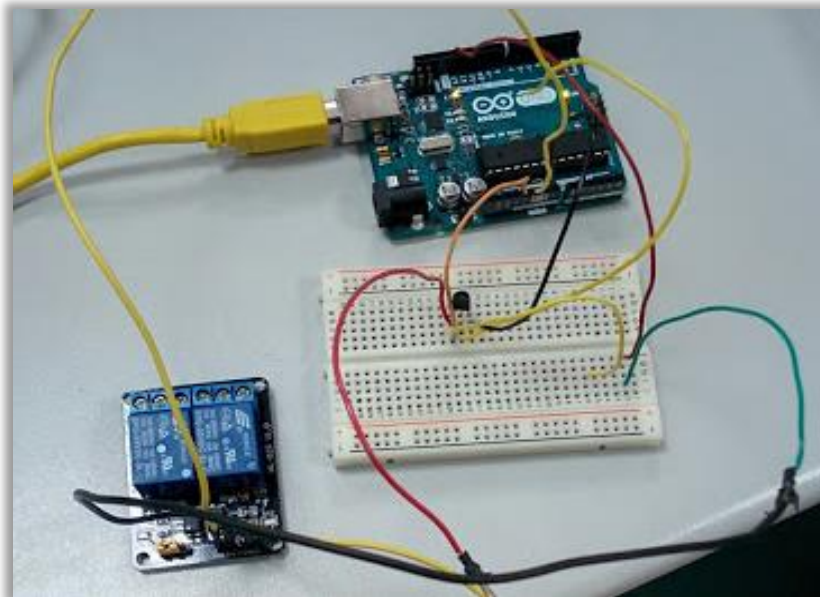
✚ Resultado Servidor Web con temperatura



## 1.9 Lectura remota de la temperatura y relé con bombilla LED

En este ejercicio /tarea estaremos esperando datos vía serie para encender o apagar el LED del relé manteniendo la lectura de la temperatura. Para ello, conectaremos un relé al circuito anterior para el encendido y/o apagado de una bombilla LED. El relé se añadirá a la placa de Arduino de la misma forma que lo hicimos en la práctica de Arduino.

- Montaje placa Arduino con sensor de temperatura y relé conectada a la Raspberry Pi



- Código Relé para Arduino

```
int pinRele = 13;
int sensorPin = 0;
char val;
void setup(){
    pinMode(pinRele, OUTPUT);
    Serial.begin(9600);
}
void loop(){
    float sensorVal;
    float temperatura;
    if(Serial.available() > 0){
        char c = Serial.read();
        if(c == 'H'){
            digitalWrite(pinRele, HIGH);
        }
        else if(c == 'L'){
            digitalWrite(pinRele, LOW);
        }
    }
    sensorVal = analogRead(sensorPin);
    temperatura = ((sensorVal/1024)*5 - .5)*100;
    Serial.println(temperatura);
    delay(2000);
}
```

```

import RPi.GPIO as GPIO

import serial

from flask import Flask, render_template, request

app = Flask(__name__)

GPIO.setmode(GPIO.BCM)
# Create a dictionary called pins to store the pin number, name, and pin state
pins = {
    17: {'name': 'RELE', 'state': GPIO.LOW},
    24: {'name': 'LED1', 'state': GPIO.LOW},
    25: {'name': 'LED2', 'state': GPIO.LOW}
}
ser = serial.Serial('/dev/ttyACM0', 9600)
# Set each pin as an output and make it low:
for pin in pins:
    GPIO.setup(pin, GPIO.OUT)
    GPIO.output(pin, GPIO.LOW)

@app.route("/")
def main():
    temp = ser.readline()
    # For each pin, read the pin state and store it in the pins dictionary:
    templateData = {
        'pins': pins,
        'temps': temp,
    }

    # Pass the template data into the template main.html and return it to the user
    return render_template('main.html', **templateData)

@app.route("/update")
def temp():
    temp = ser.readline()
    for pin in pins:
        pins[pin]['state'] = GPIO.input(pin)

    templateData = {
        'pins': pins,
        'temps': temp,
    }
    return render_template('main.html', **templateData)

# The function below is executed when someone requests a URL with the pin number and action in it
@app.route("/<changePin>/<action>")
def action(changePin, action):
    # Convert the pin from the URL into an integer:
    changePin = int(changePin)
    # Get the device name for the pin being changed:
    deviceName = pins[changePin]['name']
    # If the action part of the URL is "on," execute the code indented below:
    if action == "on":
        if changePin == 17:
            ser.write('L')
            GPIO.output(changePin, GPIO.HIGH)
            message = "Cambiado" + deviceName + "Ya ha encendido"

    if action == "off":
        if changePin == 17:
            ser.write('H')
            GPIO.output(changePin, GPIO.LOW)
            message = "Cambiado" + deviceName + "Ya ha sido apagada"

    if action == "toggle":
        GPIO.output(changePin, not GPIO.input(changePin))
        message = "Toggle " + deviceName + "."

    # For each pin, read the pin state and store it in the pins dictionary:
    for pin in pins:
        pins[pin]['state'] = GPIO.input(pin)

    # Along with the pin dictionary, put the message into the template data dictionary:
    templateData = {
        'message': message,
        'pins': pins
    }

    return render_template('main.html', **templateData)

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=80, debug=True)

```

## Código HTML



```
carlos@carlos-desktop: ~/weblamp/templates
GNU nano 2.5.3 File: main.html

<!DOCTYPE html>
<head>
<title>Carlos Crespo Jimenez</title>
</head>

<body>

<h1> Listado de dispositivos y sus estados</h1>
    {% for pin in pins %}
    <p> The {{ pins[pin].name }}
    {% if pins[pin].state == true %}
    is currently on (<a href="{{pin}}/off">Apagar</a>)
    {% else %}
    is currently off (<a href="{{pin}}/on">Encender</a>)
    {% endif %}
    </p>
    {% endfor %}

    <p> La temperatura actual es: {{temps}} °C (<a href="/update"> Actualizame por favor </a>) </p>
    {% if message %}
    <h2> {{ message }} </h2>
    {% endif %}

</body>
</html>
```

## Resultado Servidor Web con temperatura y relé



## 1.10 Conclusión

He aprendido algunas de las tantas utilidades que tiene la placa Raspberry Pi. Con esta placa podemos realizar cosas básicas desde fusionar dicha placa con otra (Arduino) para ampliar su funcionalidad como obtener la temperatura ambiental o encender leds.

Para mí, es la placa más completa y útil de las 3 que hemos estudiado y trabajado en las sesiones de prácticas. Como ventaja de esta placa es que le podemos incluir un sistema operativo como Ubuntu Mate y conectarnos fácilmente a Internet ya sea desde

el cable RJ45 o por el receptor wifi n que lleva incorporado. En definitiva, podemos llevar un pequeño ordenador de bolsillo y controlar el comportamiento de diferentes aparatos desde internet.