

LABORATORIO DE DESARROLLO DE HARDWARE RASPBERRY PI

Model 3



MEMORIAS

2016/2017

Rafael arroyo alemán
Ingeniería Informática de
Computadores

ÍNDICE

Introducción	2
➤ Montaje de MicroSd.....	2
Control de los GPIO de Raspberry Pi	5
➤ Programa Python.....	5
Conexión ssh a Raspberry Pi	6
Control de Leds a través de un servidor web.....	7
➤ Instalación Web Framework para Python.....	7
➤ (Directorio HTML) Templates.....	9
Proyecto WebLamp	10
➤ Código script Python.py	10
➤ Código HTML	13
Variación del servidor Web Raspberry Pi con Arduino (Sensor de temperatura tmp36GZ y Relé con bombilla “Led Rojo”)	14
Código.....	15
➤ Montaje	18

Introducción

Llegados a este punto vamos a realizar las practicas relacionadas con las Single Board Computer “SBC”, en la cual para ello se utilizará la raspberry Pi en su versión más nueva V3.0.

Está desarrollada en Reino Unido por la fundación de RaspBerry Pi, y se emprendió con el objetivo de estimular la enseñanza de ciencias de la computación en las escuelas.



Es una placa base de 85 x 54 mm en el cual podemos encontrar un chip Broadcom BCM2835 con procesador ARM de 1Ghz aprox, GPU VideoCore IV y 1GB RAM.

Para que funcione, necesitamos de una tarjeta de memoria para usarlo como medio de almacenamiento “MicroSd en nuestro caso”, para que nos empiece a funcionar deberemos conectarlo a la corriente utilizando para ello cualquier cargador microUsb de al menos 2500mAh y además se recomienda guardarla en una carcasa para evitar cualquier impacto.

Dispondremos también de salida de video HDMI, un minijack de audio y 4 USB, una conexión a internet de Ethernet para enchufar el cable RJ-45 o también por medio wifi, ya sea por adaptador o por el que viene integrado en la nueva RPI3.

➤ Montaje de MicroSd

Antes de poder empezar a usar la raspberry Pi tenemos que preparar la tarjeta MicroSd, lo haremos mediante una imagen de un sistema operativo ya preparado:

1º Usaremos la herramienta ‘dd’ de Linux la cual puede sobrescribir cualquier partición ya que se realiza una escritura a bajo nivel.

Tenemos que ver primera las unidades que actualmente están montadas ‘df -h’.

Insertamos la tarjeta y volvemos a ejecutar ‘df -h’ para ver cual se ha añadido y así una vez que tenemos el nombre del dispositivo:

Tenemos que desmontarlo → `umount /dev/NombreDispositivo`

Y ejecutamos: `dd bs=4M if=2016-09-23-raspbian-jessie.img of=/dev/sdd`

Siendo **if=** el nombre de la imagen y **of=** de nombre del dispositivo.

Y con esto empezará a escribir la tarjeta microSD, si queremos ver el progreso tenemos que el comando `dd` no nos proporciona esta información, para ello ejecutaremos:

`kill -USR1 -n -x dd`

Una vez finalizado ya tendremos lista nuestra imagen para empezar a instalarla en nuestra Raspberry Pi.

Nos aparece la ventana de carga



```
[ 1.551752] EXT4-fs (mmcblk0p2): couldn't mount as ext2 due to feature incompatibilities
[ 1.575189] EXT4-fs (mmcblk0p2): mounted filesystem with ordered data mode. Opts: (null)
[ 1.577879] UFS: Mounted root (ext4 filesystem) on device 179:2.
[ 1.581544] devtmpfs: mounted
[ 1.584645] Freeing unused kernel memory: 364K (00085000 - 000c0000)
[ 1.691712] usb 1-1: new high-speed USB device number 2 using dwc_otg
[ 1.694439] Indeed it is in host mode hprt0 = 00001101
[ 1.883978] usb 1-1: New USB device found, idVendor=0424, idProduct=9514
[ 1.886705] usb 1-1: New USB device strings: Mfr=0, Product=0, SerialNumber=0
[ 1.890548] hub 1-1:1.0: USB hub found
[ 1.894019] hub 1-1:1.0: 5 ports detected
[ 1.936360] random: systemd random read with 44 bits of entropy available
[ 1.946613] systemd[1]: systemd 219 running in system mode. (+PAM +AUDIT +SELINUX +IMA +APPARMOR +SMACK +SYSVINIT +UTMP +LIBCRYPTSETUP +GCRYPT -GNUTLS +ACL +XZ -LZ4 -SECCOMP +BLKID -ELFUTILS +KMOD -IDN)
[ 1.952877] systemd[1]: Detected architecture arm.
[ 1.955915] systemd[1]: Running with unpopulated /etc.

Welcome to Ubuntu 15.04!

[ 1.970617] systemd[1]: Set hostname to ubuntu-mate.
[ 1.997827] systemd[1]: Initializing machine ID from random generator.
[ 2.167756] usb 1-1: new high-speed USB device number 3 using dwc_otg
[ 2.272040] usb 1-1: New USB device found, idVendor=0424, idProduct=ec00
[ 2.274990] usb 1-1: New USB device strings: Mfr=0, Product=0, SerialNumber=0
[ 2.281801] smac95xx v1.0.4
[ 2.336254] smac95xx 1-1:1.0 eth0: register 'smac95xx' at usb-hcd2700-usb-1.1, smac95xx USB 2.0 Ethernet, 08:27:eb:90:7c:07
[ 2.396154] systemd[1]: Populated /etc with preset unit settings.
[ 2.414550] usb 1-1: new low-speed USB device number 4 using dwc_otg
[ 2.513229] usb 1-1: New USB device found, idVendor=0458, idProduct=003a
[ 2.516466] usb 1-1: New USB device strings: Mfr=1, Product=2, SerialNumber=0
[ 2.519625] usb 1-1: Product: Optical Mouse
[ 2.522710] usb 1-1: Manufacturer: Genius
```

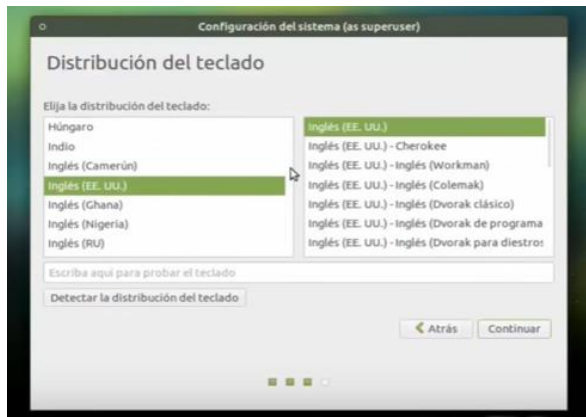
Y una vez que termine tendremos que configurar la instalación mate



Elegimos español



Y situación geográfica



Elegimos el idioma del teclado



Y un nombre de usuario

Esperamos a que termine el proceso de configuración e instalación y en la ventana de bienvenida :



Vamos a Raspberry Pi Information y le damos a Redimensionar tamaño.

Comprobamos que la fecha y la hora de nuestro Ubuntu Mate es la correcta para evitarnos futuros problemas

Ahora como nuestras pruebas las vamos a hacer en el laboratorio y nos vamos a conectar a la red mediante la red ethernet de esta tenemos que configurar la conexión de internet.

Editamos el archivo `/etc/network/interfaces`

```
# interfaces(5) file used by ifup(8) and ifdown(8)
# Include files from /etc/network/interfaces.d:
source-directory /etc/network/interfaces.d

# The loopback network interface
auto lo
iface lo inet loopback

auto enxb827eba97cd4 Para que se autoejecute
iface enxb827eba97cd4 inet static static para poner como ip fija
address 10.1.15.98 IP de conexion
netmask 255.255.252.0
gateway 10.1.15.78
dns-nameservers 8.8.8.8
```

Ejecutamos como superusuario:

```
$ sudo ifdown eth0
```

```
$ sudo ifup eth0
```

Y por último hacemos un ping a www.google.es por ejemplo para probar si se ha hecho la conexión.

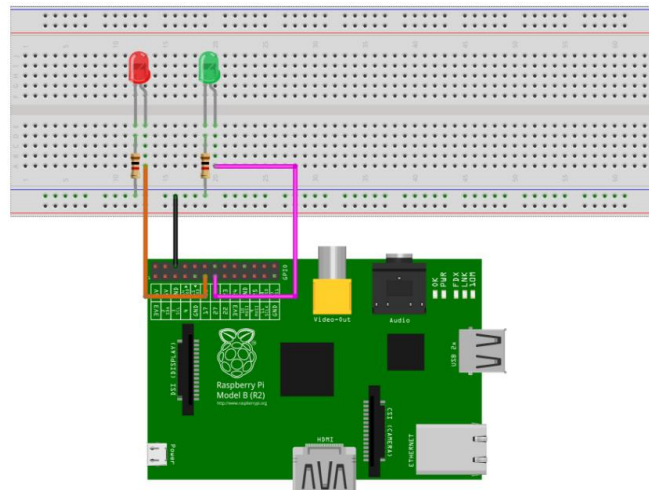
Control de los GPIO de Raspberry Pi

Ahora podemos empezar a hacer nuestras prácticas:

La primera será como toma de contacto con el manejo de los GPIOs desde la terminal:

➔ Necesitamos dos resistencias de por ejemplo 220ohm y un par de leds.

Usaremos los GPIO 17 y 27 la cuales irán conectadas a



Librería GPIO

Como la librería de las GPIOs ya vienen instaladas en Ubuntu mate no necesitaremos instalarlas.

➤ Programa Python

Ahora escribiremos un archivo con extensión “Nombre”.py en cualquier editor de texto, en nuestro caso usamos **Pluma** ya instalado en Mate.

Los pines que usaremos serán de salida ya que encenderemos y apagaremos Leds:

Importación de librerías y declaración "setup" de los pines:

```
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
```

Creamos una función para ejecutar el bucle que apaga y enciende los leds de manera intermitente

```
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
```

Ya por último solo nos quedaría **irnos a la terminal y ejecutar** el .py que acabamos de crear

```
sudo python blink.py
```

Aquí un ejemplo del resultado:

https://1drv.ms/v/s!Ao2_30mhw3bivVq5tH7QxntQ7G-O

Conexión ssh a Raspberry Pi

Para conectarnos a la raspberry Pi solo con el cable ethernet y desde la ventana de comandos del pc establecemos una conexión con la rpi con el siguiente comando:

Ssh -X [pi@10.1.15.XX](#) # Con la **opción -X** podremos ejecutar aplicaciones graficas remotamente y si **establecemos -P** indicamos por cual puerto se debe hacer.

➔ Generando clave pública ssh:

- Ante todo, asegurarse que no tengas ya una clave. Por defecto, las claves de cualquier usuario SSH se guardan en la carpeta ~/.ssh de dicho usuario. Para verificar si tenemos ya unas claves, simplemente nos situamos sobre dicha carpeta y vemos su contenido:

```
$ cd ~/.ssh
$ ls
authorized_keys2  id_dsa          known_hosts
config           id_dsa.pub
```

Hemos de buscar un par de archivos con nombres tales como **algo** y **algo.epub**, siendo ese "algo" normalmente `id_rsa`. El archivo terminado en `.epub` es nuestra clave pública, y el otro archivo es nuestra clave privada. Si no tenemos esos archivos (o no tenemos ni siquiera la carpeta `.ssh`), tenemos que crearla, utilizando un programa llamado `ssh-keygen`, que viene incluido en el paquete SSH de los sistemas Linux/Mac o en el paquete MSysGit en los sistemas Windows:

```
$ ssh-keygen
```

```
Generating public/private rsa key pair.
```

```
Enter file in which to save the key (/Users/Rafa/.ssh/id_rsa):
```

```
Enter passphrase (empty for no passphrase):
```

```
Enter same passphrase again:
```

```
Your identification has been saved in /Users/Rafa/.ssh/id_rsa.
```

```
Your public key has been saved in /Users/Rafa/.ssh/id_rsa.pub.
```

```
The key fingerprint is:
```

```
43:c5:5b:5f:b1:f1:50:43:ad:20:a6:92:6a:1f:9a:3a Rafa@10.1.15.98
```

Como se ve, este comando primero solicita confirmación de dónde van a guardar las claves (`.ssh/id_rsa`), y luego solicita, dos veces, una contraseña (*passphrase*), contraseña que podemos dejar en blanco si no deseamos tener que teclearla cada vez que usemos la clave.

Tras generarla, debemos encargarnos de enviar nuestra clave pública al servidor con SSH. Esto se puede realizar simplemente copiando los contenidos del archivo terminado en `.epub` dentro del archivo `.ssh/authorized_keys` **sino existe lo creamos con los permisos adecuados**.

Control de Leds a través de un servidor web

Los Single Board Computer permiten controlar otros dispositivos hardware (Sensores y actuadores), el cual se puede realizar también a través de internet.

En esta práctica se hace a través de un Servidor Web por el que podremos enviar y recibir datos del hardware que estemos usando.

➤ Instalación Web Framework para Python

La práctica se hará en una Web Framework para Python, donde podremos ejecutar código Python desde el servidor web.

- ➔ Para descargar paquetes del repositorio *PyPi* se pueden utilizar varias herramientas, pero en este caso se va a usar **pip**. Es necesario instalar esta herramienta en el sistema en caso de no estar disponible, antes de poder instalar un paquete *Python*.

```
pi@raspberrypi ~ $ sudo apt-get install python-pip
```


➔ Después de haber instalado el pip podremos instalar Flask y sus dependencias.

Ojo, esto nos puede dar problemas si no tenemos **la fecha y hora** correcta del sistema

```
pi@raspberrypi ~ $ sudo pip install flask
```

➔ Para probar la instalación creación un fichero.py con el siguiente código:

#Cargamos el módulo flask en el script python

```
from flask import Flask
```

```
app = Flask(__name__)
```

#Creamos un objeto flask llamado app

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

```
def hello():
```

```
    return "Hello World!"
```

#Función que se ejecuta cuando se llama desde la web

```
if __name__ == "__main__":
```

```
    app.run(host='0.0.0.0', port=80, debug=True) #ponemos host Ip de nuestro servidor
```

➔ Y ya solo nos queda ejecutarlo:

```
pi@raspberrypi ~ $ sudo python hello-flask.py
```

```
* Running on http://0.0.0.0:80/
```

```
* Restarting with reloader
```

➔ Aquí la primera línea muestra que se ha accedido a la url de nuestro servidor http correctamente.

La segunda línea es una respuesta de un pequeño icono enviado para mostrar la url en la barra de direcciones del navegador

```
10.0.1.100 - - [19/Nov/2012 00:31:31] "GET / HTTP/1.1" 200 -
```

```
10.0.1.100 - - [19/Nov/2012 00:31:31] "GET /favicon.ico HTTP/1.1" 404 -
```

➤ (Directorio HTML) Templates

- ➔ Si queremos enviar al buscador a un sitio en formato html, no ponemos todo este código dentro del script Python para eso usaremos jinja2 incorporado en flasky para separar el código HTML del script.py.

Para ello creamos una carpeta llamada **templates** en el mismo directorio donde se encuentre el script.py.

Dentro de **templates** meteremos el archivo .html y lo llamaremos main.html e insertaremos el código en él.

Cualquier cosa entre dos llaves dentro de la plantilla HTML se interpreta como una variable que se le pasará desde el script Python a través de la función render_template.

➔ Script hello_template.py

- Nuevamente escribimos el script Python pero con la diferencia que aquí incorporamos el paso de objetos al HTML en el templateData y render_template()

```
from flask import Flask, render_template
```

```
import datetime
```

```
app = Flask(__name__)
```

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

```
def hello():
```

```
# Obtenemos la hora actual y la guardamos en un objeto
```

```
    now = datetime.datetime.now()
```

```
    timeString = now.strftime("%Y-%m-%d %H:%M")
```

```
#Variables a pasar al HTML
```

```
    templateData = {
```

```
        'title' : 'HELLO!',
```

```
        'time': timeString
```

```
    }
```

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

```
if __name__ == "__main__":
```

```
    app.run(host='0.0.0.0', port=80, debug=True)
```

➔ En el main.html

```
<!DOCTYPE html>

<head>

  <title>{{ title }}</title>

</head>


<body>

  <h1>Hola, Mundo!</h1>

  <h2>La fecha y hora del servidor es: {{ time }}</h2>

</body>

</html>
```

Cuando corramos el script.py y pongamos en el buscador la dirección del servidor web podremos ver la página en el formato HTML con el título HELLO! Y la fecha y hora actual de la Raspberry Pi.

Proyecto WebLamp

- ➔ Ahora que sabemos cómo usar Python y Flask, ahora podemos controlar el estado de una lámpara en la web. Este proyecto básico es simplemente un punto de partida para crear dispositivos conectados a Internet con el Raspberry Pi.

Creamos un nuevo directorio en nuestro directorio de inicio llamado WebLamp. Dentro de éste, creamos un archivo llamado weblamp.py y ponemos el código:

➤ [Código script Python.py](#)

- Explicación en el código en negrita

```
import RPi.GPIO as GPIO

from flask import Flask, render_template, request

app = Flask(__name__)


GPIO.setmode(GPIO.BCM)
```

Creamos un diccionario llamado pins para almacenar el número de pin, nombre y el estado del pin:

```
pins = {  
    24 : {'name' : 'coffee maker', 'state' : GPIO.LOW},  
    25 : {'name' : 'lamp', 'state' : GPIO.LOW}  
}
```

para cada pin establecemos como salida y lo ponemos a 0:

```
for pin in pins:  
    GPIO.setup(pin, GPIO.OUT)  
    GPIO.output(pin, GPIO.LOW)
```

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

```
def main():
```

#Para cada pin, leemos el estado del pin y lo almacenamos en el diccionario de pins:

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

Ponemos el diccionario de pins en el diccionario de datos template:

```
templateData = {  
    'pins' : pins  
}
```

Pasamos los datos template al template main.html and y lo devolvemos al usuario

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

La función es ejecutada cuando alguien requiere respuesta de una URL con el número de pin y la acción:

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

```
def action(changePin, action):
```

Convertimos el pin de la URL a un entero:

```
changePin = int(changePin)
```

#Obtenemos el nombre del dispositivo para el pin que está siendo cambiado:

```
deviceName = pins[changePin]['name']
```

```
# Si la parte de la acción de la URL es "on", ejecuta el código :
```

```
if action == "on":
```

```
    # ponemos el pin en alto:
```

```
    GPIO.output(changePin, GPIO.HIGH)
```

```
    # Salvar el estado del mensaje que va a ser pasado al template:
```

```
    message = "Turned " + deviceName + " on."
```

```
if action == "off":
```

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

```
    message = "Turned " + deviceName + " off."
```

```
if action == "toggle":
```

```
    # Lee el pin y lo establece a lo que sea:
```

```
    GPIO.output(changePin, not GPIO.input(changePin))
```

```
    message = "Toggled " + deviceName + "."
```

```
# Para cada pin, lee el estado del pin y lo almacena en el diccionario pins:
```

```
for pin in pins:
```

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

```
# A lo largo del diccionario de pin, pone el mensaje en el diccionario de datos template:
```

```
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>Current Status</title>
```

```
</head>
```

```
<body>
```

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

Esto establece un bucle for para recorrer cada pin, imprimir su nombre y su estado. A continuación, da la opción de cambiar su estado, en función de su estado actual

```
  {% 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 %}
```

Esta instrucción if imprimirá un mensaje transmitido desde su secuencia de comandos Python si hay un conjunto de mensajes (es decir, se ha producido un cambio).

```
  {% if message %}
```

```
    <h2>{{ message }}</h2>
```

```
  {% endif %}
```

```
</body>
```

```
</html>
```

--> En terminal, navegamos hasta el directorio de WebLamp e iniciamos el servidor (asegurándonos de usar Control-C para matar cualquier otro servidor de Flask que haya ejecutado primero):

```
pi@raspberrypi ~/WebLamp $ sudo python weblamp.py
```

➔ Conexionado del led a los pines.

Se realiza el circuito conectado para la prueba el Led (Ánodo) directamente al GPIO 25 (Pin 11) y al GND el cátodo del led (Pin 3) hacemos clic directamente en el servidor web a Lámpara para comprobar el correcto funcionamiento encendiéndolo y apagándolo.



Ejemplo del resultado de la práctica:

https://1drv.ms/v/s!Ao2_30mhw3bivVTIVCcJ-OFzh7y

Variación del servidor Web [Raspberry Pi con Arduino](#) (Sensor de temperatura tmp36GZ y Relé con bombilla “Led Rojo”)

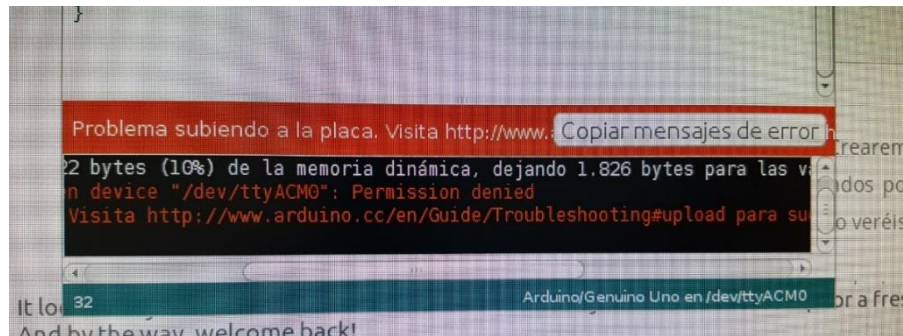
En esta práctica vamos a usar una placa Arduino conectada a Raspberry Pi para transmitir vía serie datos de temperatura a una página web y para recibir comandos de activación y desactivación de un relé (led rojo video) desde ésta.

Desde el servidor web seremos capaces **de encender y apagar la luz “Bombilla”**, así como encender led **verde “Cafetera”** y azul **“Lámpara”** (manteniendo la lectura de la temperatura **tmp36GZ**).

Para ello lo primero que se ha hecho es hacer la instalación del IDE de arduino en la Raspberry Pi ya que la usaremos para cargar el programa en Arduino.

El procedimiento es el mismo que para la instalación de Ubuntu, con la diferencia que nos encontramos con un error de comunicación a la hora de subir el .HEX a Arduino.

➔ Error :



Para ello lo único que tenemos que hacer es agregarlos al Dialout de la siguiente forma:

➔ Sudo adduser <username> dialout

Tras esto cerramos la sesión y volvemos a entrar en ella de manera normal y ya podremos subir el programa a arduino.

Código

- **Código Arduino:**

Este código básicamente está esperando la recepción de un carácter el cual dependiendo de éste estaremos pidiendo el encendido de la bombilla “e”, apagado de la bombilla “a” o la lectura de un dato del sensor de temperatura.

```
int LDRPin = 0;
int rele = 2;

void setup () {
  Serial.begin(9600);
  pinMode(2,OUTPUT);
}

void loop () {
  if(Serial.available()) {
    char c = Serial.read();
    if(c == 'e'){
      digitalWrite(rele,HIGH);
    }else if(c == 'a'){
      digitalWrite(rele,LOW);
    }else{
      int value = analogRead(LDRPin);
      float millivolts = (value / 1023.0) * 5000;
      float celsius = millivolts / 10;
      Serial.print(celsius);
      Serial.println(" C");
    }
  }
}
```

➔ Script Python .Py

Añadido al proyecto WebLamp:

El procedimiento seguido es el siguiente:

1. Se añade una comunicación Serial con un timeout de 1.0 para que no se quede colgado en caso de fallo de lectura o escritura
2. Se crea una estructura set of keys para temperatura para que se mantenga el dato en una carga de página por cualquier otro comando.
3. Se define un método `@app.route ("</temper>")` y definimos el método **leerTemper(temper)** donde trataremos el argumento que se pase.
Para cada dato de pin, leemos el estado del pin y lo almacenamos en el diccionario de pins, para que cuando se lea temperatura nos aparezcan también las opciones de cafetera y lámpara en su estado.
4. Y ahora hacemos el tratamiento del dato que se pasa por html:
 - Si es L: pedimos una temperatura
 - Si es e: pedimos encender la bombilla "relé"
 - Si es a: pedimos apagar la bombilla "relé"
5. Por ultimo le pasamos al `templateData` los datos de pins y de temperatura al `render template` para que se puedan pasar al documento HTML.

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

GPIO.setmode(GPIO.BCM)
arduino=serial.Serial('/dev/ttyACM0',baudrate=9600,timeout=1.0)

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

temperatura = {
    1 : {'temp' : ''}
}

# 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,
        'temperatura' : temperatura
    }

    # 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 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":
        # Set the pin high:
        GPIO.output(changePin, GPIO.HIGH)
        # Save the status message to be passed into the template:
        message = "Puesta " + deviceName + " on."
    if action == "off":
        GPIO.output(changePin, GPIO.LOW)
        message = "Puesta " + 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,
        'temperatura' : temperatura,
        'pins' : pins
    }

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

@app.route("/<temper>")
def leerTemp(temper):
    # For each pin, read the pin state and store it in the pins dictionary:
    for pin in pins:
        pins[pin]['state'] = GPIO.input(pin)

    if temper == "I":
        arduino.write("N")
        temperatura[1]['temp'] = ""
        time.sleep(0.8)
        while arduino.inWaiting() != 0:
            temperatura[1]['temp'] += arduino.read(1)

    if temper == "e":
        arduino.write("e")

    if temper == "a":
        arduino.write("a")

    templateData = {
        'temperatura' : temperatura,
        'pins' : pins
    }

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

if __name__ == "__main__":
    app.run(host='192.168.1.133', port=80, debug=True)

```

➔ Código HTML:

En este código HTML se añade al de proyecto WebLamp el:

1. **Actualiza temperatura:** Es un enlace el cual manda la letra L al script.py y lee el valor pasado {{temperatura[1].temp}}.
2. **Enciende Bombilla:** Enlace que pasa la letra 'e' al script para encender bombilla
3. **Apaga Bombilla:** Enlace que pasa la letra 'a' al script para apagar la bombilla

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

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

  {% for pin in pins %}
  <p>La {{ pins[pin].name }}
  {% if pins[pin].state == true %}
  | esta actualmente (<a href="/{pin}/off">apagada</a>)
  {% else %}
  | esta actualmente (<a href="/{pin}/on">encendida</a>)
  {% endif %}
  </p>
  {% endfor %}

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

  <h2>Actualiza temperatura: (<a href="/L">Leer</a>)</h2>
  El valor es : {{temperatura[1].temp}}

  <h2>Enciende Bombilla: (<a href="/e">Enciende</a>)</h2>

  <h2>Apaga Bombilla: (<a href="/a">Apaga</a>)</h2>

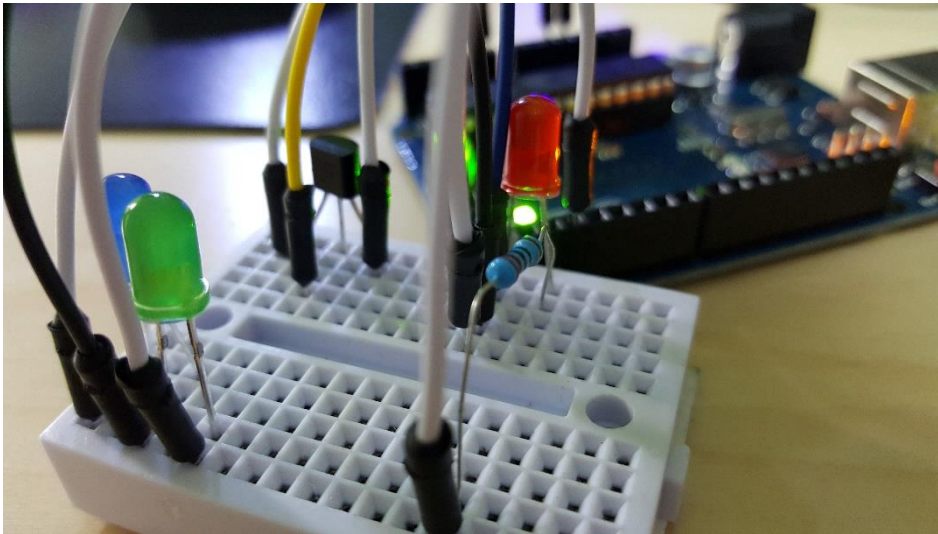
  </body>
```

➤ Montaje

Ya por último nos queda el montaje del circuito, en el que se encontraran incorporados los **leds de cafetera y Lámpara** (Leds verde y azul) conectados a los **GPIO de la Raspberry Pi** y el **relé**, el cual queda **representado por el led rojo** junto a su resistencia de 220ohm en el ánodo, en el que lo único que tendríamos que hacer es cambiar la conexión a éste ,se encuentra conectado al pin 2 de Arduino y por último **el sensor de LM395** que se encontrará leyendo temperatura, se encuentra conectado a la entrada **analógica A0** de Arduino.

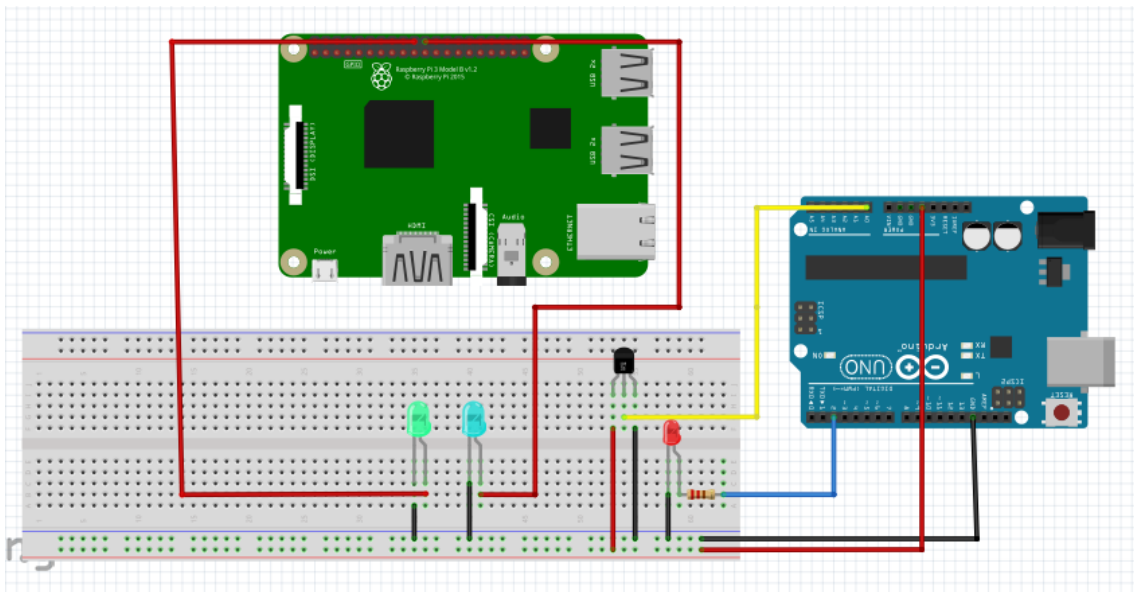
Por ultimo tenemos que la placa Arduino se encuentra conectada a la raspberry Pi por el USB, a través de esta conexión se realizara la **carga del archivo de configuración de Aduino** y de la **comunicación Serial** entre ambas placas tanto en el envío como en la recepción de datos.

Componentes

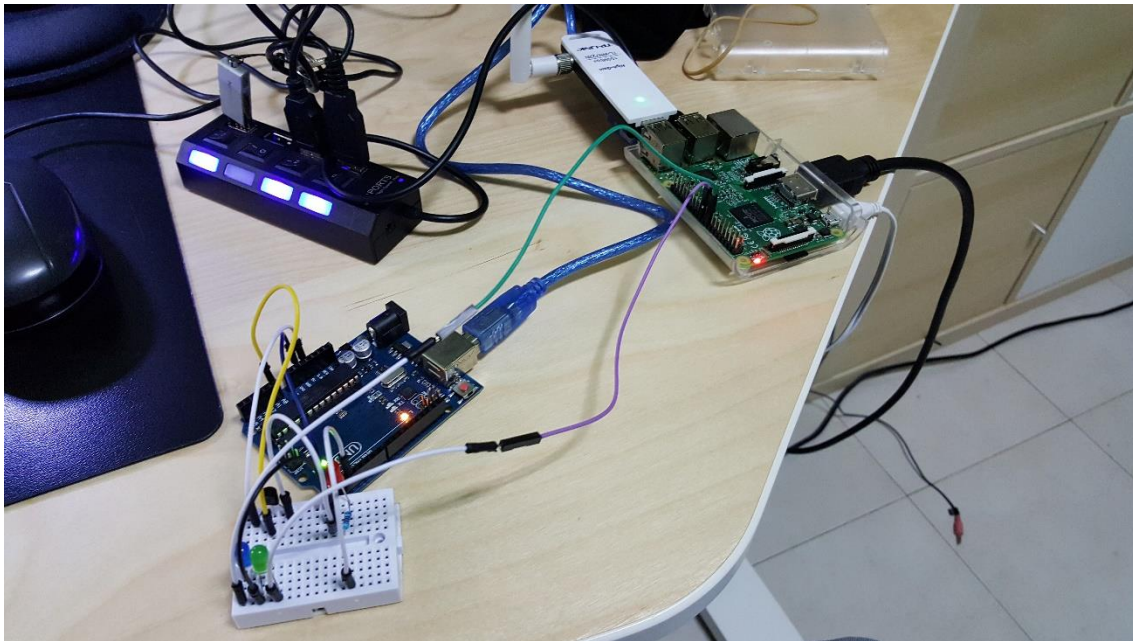


- .Led Rojo
Relé
- .Led Verde
Cafetera
- .Led azul
Lámpara
- .Sensor
temperatura

Circuito: Realizado con Fritzing (Herramienta OpenSource)



Proyecto ya montado:



Demostración del resultado:

https://1drv.ms/v/s!Ao2_30mhw3bivgLIq23M0u604JJj