

# Conexión MQTT, Broker - Cliente

Díaz Diego, García Giancarlo, Gutiérrez Miguel, Martínez Laura, Mora Diego, y Ortiz Jhon  
Ingeniería Electrónica  
Yopal-Casanare

[Miguelgutierrez@unisangil.edu.co](mailto:Miguelgutierrez@unisangil.edu.co)

[diegomora@unisangil.edu.co](mailto:diegomora@unisangil.edu.co)

[fabianandiaz@unisangil.edu.co](mailto:fabianandiaz@unisangil.edu.co)

[lauraktharinemartinez@unisangil.edu.co](mailto:lauraktharinemartinez@unisangil.edu.co)

[giancarlologarcia@unisangil.edu.co](mailto:giancarlologarcia@unisangil.edu.co)

[jhoanortiz@unisangil.edu.co](mailto:jhoanortiz@unisangil.edu.co)

**Resumen--** Por medio de este laboratorio se desarrolló y se implementó un sistema de domótico de apertura de una puerta, control de temperatura y notificación por medio de correo

**-Palabras claves:** ESP32 , MQTT, Broker, DHT11 y Broker

## I. INTRODUCCION

MQTT (Message Queue Telemetry Transport) es un protocolo de transporte de mensajes Cliente/Servidor basado en publicaciones y subscripciones a los denominados “tópicos”. Cada vez que un mensaje es publicado será recibido por el resto de dispositivos adheridos a un tópico del protocolo.

TST implementa el protocolo MQTT en sus dispositivos de Internet de las Cosas, lo cual simplifica y hace más sencilla la recogida de datos de sensores, la publicación de los diferentes valores obtenidos y la configuración remota de los nodos

## II. MATERIALES O EQUIPOS

- Multímetro Digital
- Fuente de alimentación
- Protoboard
- Jumpers Macho a Macho y Macho a Hembra ( o un metro de cable utp)
- Modulo ESP 01 (esp8266)
- Arduino uno (o ESP 32 u otro microcontrolador)

## III. OBJETIVO

- Realizar una conexión MQTT entre el Broker y el microcontrolador.

## IV. PROCEDIMIENTO

Diseño e implemente un sistema de domótico de apertura de una puerta, control de temperatura y notificación por medio de correo

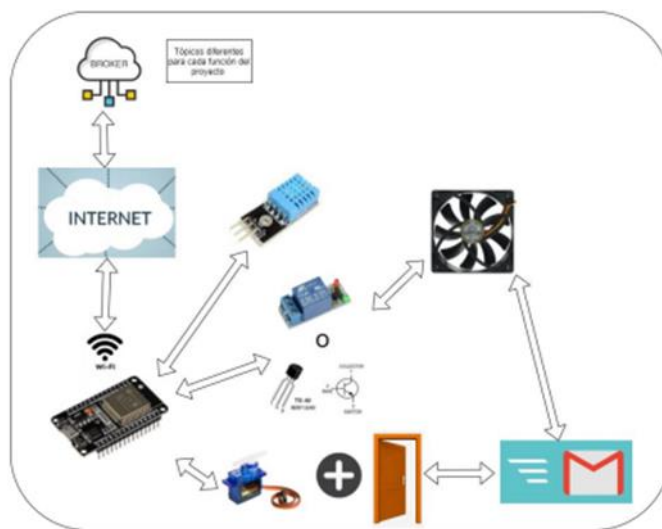


Figure 1 conexión del sistema

## A. EXPERIENCIA 1\*

El funcionamiento del MQTT es un servicio de mensajería push con patrón publicador/suscriptor (pub-sub). En este tipo de infraestructuras los clientes se conectan con un servidor central denominado broker

Para filtrar los mensajes que son enviados a cada cliente los mensajes se disponen en topics organizados jerárquicamente. Un cliente puede publicar un mensaje en un determinado topic. Otros clientes pueden suscribirse a este topic, y el broker le hará llegar los mensajes suscritos.

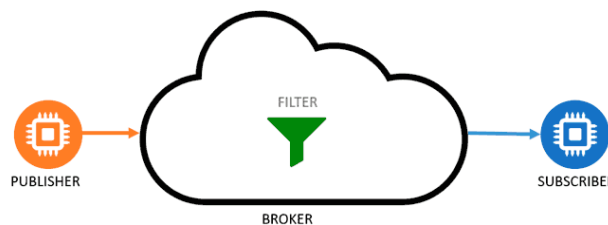


Figure 2 MQTT

Los clientes inician una conexión TCP/IP con el broker, el cual mantiene un registro de los clientes conectados. Esta conexión se mantiene abierta hasta que el cliente la finaliza. Por defecto,

MQTT emplea el puerto 1883 y el 8883 cuando funciona sobre TLS.

Para ello el cliente envía un mensaje CONNECT que contiene información necesaria (nombre de usuario, contraseña, client-id...). El broker responde con un mensaje CONNACK, que contiene el resultado de la conexión (aceptada, rechazada, etc).

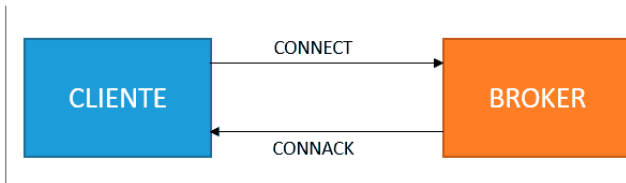


Figure 3 MQTT

Para enviar los mensajes el cliente emplea mensajes PUBLISH, que contienen el topic y el payload.

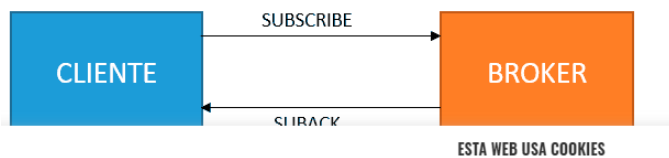


Figure 4 MQTT

Para suscribirse y desuscribirse se emplean mensajes SUBSCRIBE y UNSUBSCRIBE, que el servidor responde con SUBACK y UNSUBACK.



Figure 5 MQTT

Por otro lado, para asegurar que la conexión está activa los clientes mandan periódicamente un mensaje PINGREQ que es respondido por el servidor con un PINGRESP. Finalmente, el cliente se desconecta enviando un mensaje de DISCONNECT.

## B. EXPERIENCIA 2

Agregar un archivo a un repositorio utilizando la línea de comando

1. creado un repositorio en GitHub o que tienes un repositorio existente que es propiedad de alguien más con quien desees colaborar
2. clonado el repositorio de forma local en tu computadora
3. En tu computadora, mueve el archivo que desees cargar a GitHub en el directorio local que se creó cuando clonaste el repositorio.
4. Abre la Git Bash.
5. Cambia el directorio de trabajo actual por tu repositorio local.

6. Prepara el archivo para confirmarlo para tu repositorio local.

```
$ git add .
# Agrega el archivo a tu repositorio local y lo presenta para la confirmación. Para
```

Figure 6 COMANDO GIT ADD .

7. Confirma el archivo que has preparado en tu repositorio local.

```
$ git commit -m "Add existing file"
# Commits the tracked changes and prepares them to be pushed to a remote repository.
```

Figure 7 COMANDO GIT COMMIT -M

8. Sube los cambios en tu repositorio local a GitHub.

```
$ git push origin your-branch
# Pushes the changes in your local repository up to the remote repository you specif
```

Figure 8 COMANDO GIT BRANCH

## EXPERIENCIA 3\*

Conexión del sensor al arduino

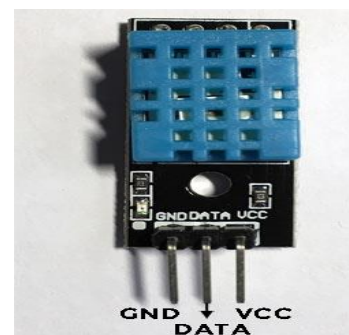


Figure 9 SENSOR DHT11

- GND: conexión con tierra
- DATA: transmisión de datos
- VCC: alimentación

a alimentación puede ser de 3,5 V a 5 V. Si vas a utilizar un Arduino MKR1000 o un ESP8266, tendrás que tenerlo en cuenta ya que estos dos dispositivos pueden dar problemas si se

alimentan con una batería o con una pila ya que no suministran más de 3,3 V.

el DHT11 integrado dentro de un PCB ya viene con la resistencia pull-up integrada. Puede resultar muy útil en ocasiones, pero si añadimos un cable de más de 20 metros, deberemos tener en cuenta este factor.

Este modelo de DHT11 dispone de 3 pines, la toma de tierra GND, para los datos DATA y para la alimentación VCC (de 3,5V a 5V). En la siguiente imagen puedes ver el esquema de conexión con Arduino.

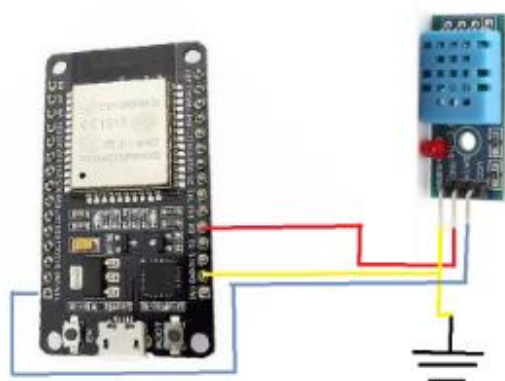


Figure 10 CONEXIÓN DEL SENSOR CON LA ESP32

### C. EXPERIENCIA 4

Hay varios modelos de servomotor con Arduino. En este caso vamos a utilizar un Micro Servo 9g SG90 de Tower Pro. Como siempre digo, hay que mirar la ficha técnica del producto. Todos tienen un funcionamiento muy parecido y la programación puede variar muy poco.

Cosas a tener en cuenta con este dispositivo. Lo primero, el ángulo de giro, en este caso nos permite hacer un barrido entre  $-90^\circ$  y  $90^\circ$ . Lo que viene a ser un ángulo de giro de  $180^\circ$ .

Aunque el servo puede moverse con una resolución de más de 1 grado, este es el máximo de resolución que vamos a conseguir debido a la limitación de la señal PWM que es capaz de generar la esp32.

Estos motores funcionan con una señal PWM, con un pulso de trabajo entre 1 ms y 2 ms y con un periodo de 20 ms (50 Hz). ¿Qué quiere decir todo esto? Este dato nos indica la velocidad máxima a la que podemos mover el servomotor con Arduino. Solo podremos cambiar de posición cada 20 ms. Esto dependerá del tipo y marca de nuestro servo.

El elegir una salida PWM u otra da lo mismo, todas las salidas de este tipo funcionan igual.

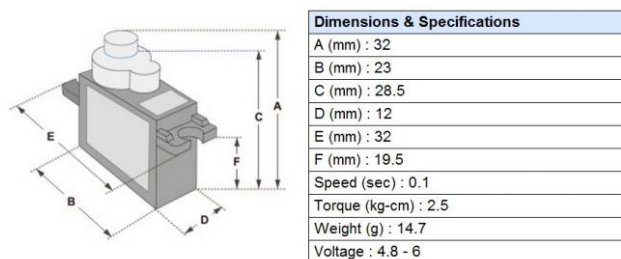


Figure 11 FICHA TECNICA DEL SERVO

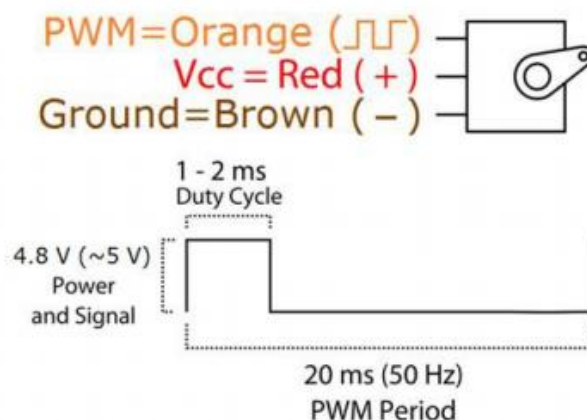


Figure 12 ENTRADAS Y SALIDAS DEL SERVO

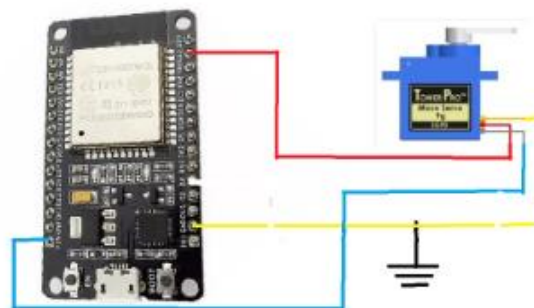


Figure 13 CONEXION DEL SERVO CON LA ESP 32

### D. EXPERIENCIA 5•

Conexión del sistema

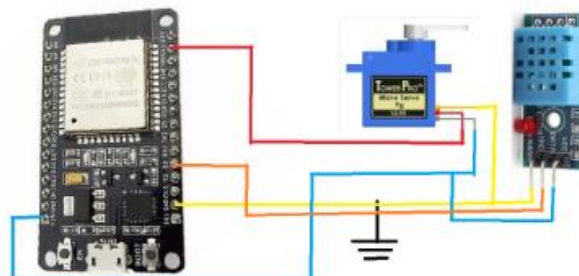


Figure 14 CONEXION DEL SENSOR Y SERVO

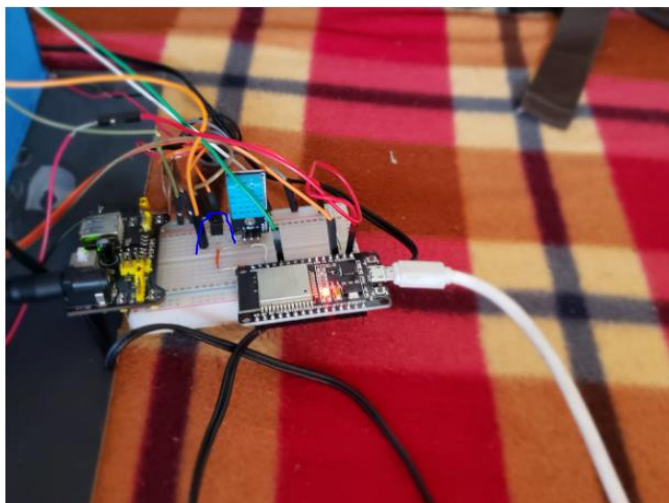


Figure 15 CONEXION DEL SISTEMA CON EL SENSOR , SERVO Y TRANSISTOR PARA EL VENTILADOR

```
// *****
// **** CONFIG MQTT ****
// *****

const char * mqtt_server = "ioticos.org" ;
const int  mqtt_port = 1883;
const char * mqtt_user = "jXbGbtv9uRczDJV" ;
const char * mqtt_pass = "tE3WSWBm65fzYc4" ;

const char * root_topic_subscribe = "Uv15cdzHn4yzDcj";
const char * root_topic_subscribe_Templ = "Uv15cdzHn4yzDcj/Templ";
const char * root_topic_subscribe_Puortal = "Uv15cdzHn4yzDcj/Puortal";

const char * root_topic_publish = "Uv15cdzHn4yzDcj/Raiz";
const char * root_topic_publish_Temp = "Uv15cdzHn4yzDcj/Temp";
const char * root_topic_publish_Hum = "Uv15cdzHn4yzDcj/Hum";
const char * root_topic_publish_Puerta = "Uv15cdzHn4yzDcj/Puerta";
```

Figure 17 CODIGO

### E. EXPERIENCIA 6\*

1. En esta sección de código incluimos las librerías y declaramos los pines de entrada del servomotor y el sensor de temperatura y humedad (DT11).

```
laboratorio

#include <WiFi.h>
#include <PubSubClient.h>
#include <ESP32_MailClient.h>
#include <ESP32Servo.h>
#include <DHT.h>
#define DHTPIN 4
#define DHTTYPE DHT11
#define servoPin 15
```

Figure 16 CODIGO

2. En este espacio se colocó el nombre de usuario del repositorio principal y su contraseña, además nos suscribimos al topic del repositorio y declaramos los topic de temperatura y el control de la puerta.

3. A continuación, se configura la red wifi que utilizaremos para la conexión.

```
// *****
// **** WIFICONFIG ****
// *****

const char * ssid = "SIN WIFI" ;
const char * password = "luanabra2021" ;
```

Figure 18 CODIGO

4. Declaramos las funciones principales de nuestro código.

```
// *** FUNCIONES ***
// *****

void callback(char* topic, byte* payload, unsigned int length);
void reconnect();
void setup_wifi();
void sensor();
void puerta();
void moditemp();
void CorreoPuertaAbierta();
void CorreoVentiladorOn();
```

Figure 19 CODIGO

5. Declaramos el pin 15 como la salida del servo



```
void setup() {
  Serial.begin(115200);

  dht.begin();

  miServo.attach(servoPin); //Conecta la variable miServo a un pin(15).
  miServo.write(0);
```

Figure 20 CODIGO

6. La puerta y el ventilador lo declaramos como pines de salida y lo colocamos en estado bajo

```
void setup() {
  Serial.begin(115200);

  dht.begin();

  miServo.attach(servoPin); //Conecta la variable miServo a un pin(15).
  miServo.write(0);

  pinMode(p, OUTPUT);
  digitalWrite(p, LOW);

  pinMode(v, OUTPUT);
  digitalWrite(v, LOW);
```

Figure 21 CODIGO

7. En esta línea de código se configuro las variables principales como String y se direcciono al repositorio principal.

```
String General = "Temperatura = " + String(t) + "°C" + " Humedad = " + String(h) + "° Puerta = " + String(EstadoPuerta);
General.toCharArray(msgG,60);
client.publish(root_topic_publish,msgG);
//count++;
delay(200);

String strT = "La temperatura es -> " + String(t) + "°C";
strT.toCharArray(msg,30);
client.publish(root_topic_publish_Temp,msg);
//count++;
delay(200);

String strH = "La humedad es -> " + String(h);
strH.toCharArray(msg,30);
client.publish(root_topic_publish_Hum,msg);
//count++;
delay(200);

String strP = "La puerta esta -> " + String(EstadoPuerta);
strP.toCharArray(msg,30);
client.publish(root_topic_publish_Puerta,msg);
//count++;
delay(200);
```

Figure 22 CODIGO

8. Se configuro la conexión con el MQTT e imprimimos los mensajes correspondiente a cada petición.

```
void reconnect() {
  while (!client.connected()) {
    Serial.print("Intentando conexión Mqtt...");
    // Creamos un cliente ID
    String clientId = "IOTICOS_R_W_";
    clientId += String(random(0xffff), HEX);
    // Intentamos conectar
    if (client.connect(clientId.c_str(),mqtt_user,mqtt_pass)) {
      Serial.println("Conectado!");
      // Nos suscribimos
      if(client.subscribe(root_topic_subscribe)){
        Serial.println("Suscripcion General");
      }else{
        Serial.println("Fallo Suscripción General");
      }
      if(client.subscribe(root_topic_subscribe_Templ)){
        Serial.println("Suscripcion a Temp");
      }else{
        Serial.println("Fallo Suscripción a Temp");
      }
      if(client.subscribe(root_topic_subscribe_Puertal)){
        Serial.println("Suscripcion a Puerta");
      }else{
        Serial.println("Fallo Suscripción a Puerta");
      }
    }
  }
}
```

Figure 23 CODIGO

9. Con esta línea se hace el llamado a cada una de las funciones.

```
void callback(char* topic, byte* payload, unsigned int length){
  String incoming = "";
  Serial.print("Mensaje recibido desde -> ");
  Serial.print(topic);
  Serial.println("");
  if(String(topic) == "Uv15cdzHn4yzDej/Templ"){
    Serial.println("TEMP1");
    for (int i = 0; i < length; i++) {
      incoming += (char)payload[i];
    }
    incoming.trim();
    Serial.println("Mensaje -> " + incoming);
    String cad = incoming.substring(0,2);
    Serial.println(cad);
    ModificarTemp = cad.toInt();
    Serial.print("La temperatura se modificó a: ");
    Serial.println(ModificarTemp);
    moditemp();
  }
  if(String(topic) == "Uv15cdzHn4yzDej/Puertal"){
    Serial.println("PUERTAL");
    for (int i = 0; i < length; i++) {
      incoming += (char)payload[i];
    }
    incoming.trim();
    Serial.println("Mensaje -> " + incoming);
    if(String(incoming) == "abrir"){
      digitalWrite(p, LOW);
    }
  }
}
```

Figure 24 CODIGO

10. Se imprime los resultados de las impresiones del sensor.

```
void sensor(){
  //Serial.println(F("DHT11 test!"));
  //dht.begin();
  delay(2000); //Es un sensor lento, por lo que hay que darle tiempo.
  h = dht.readHumidity();
  t = dht.readTemperature();
  // Comprobamos si las lecturas pueden dar algún fallo mediante la función isnan()
  // Esta función devuelve un 1 en caso de que el valor no sea numérico
  // Los caracteres || son como un OR. Si se cumple una de las dos condiciones mostramos error
  if (isnan(h) || isnan(t)) {
    Serial.println(F("No se pudo leer el sensor DHT11!"));
    return;
  }
  else{
    Serial.print(F("Humedad: "));
    Serial.print(h);
    Serial.print(F("% Temperatura: "));
    Serial.print(t);
    Serial.println(F("°C "));
  }
}
```

Figure 25 CODIGO

11. Colocamos el correo al que nos va a llegar el estado de la puerta y adicionamos los mensajes que irán a llegar.

```
void CorreoPuertaAbierta(){
  //Configuración del servidor de correo electrónico SMTP, host, puerto, cuenta y contraseña
  datosSMTP.setLogin("smtp.gmail.com", 465, "iotesp27@gmail.com", "esp32.1234");
  // Establecer el nombre del remitente y el correo electrónico
  datosSMTP.setSender("ESP32SERGIO", "iotesp27@gmail.com");
  // Establece la prioridad o importancia del correo electrónico High, Normal, Low o 1 a 5 (1 es el más alto)
  datosSMTP.setPriority("High");
  // Establecer el asunto
  datosSMTP.setSubject("ESTADO DE LA PUERTA");
  // Establece el mensaje de correo electrónico en formato de texto (sin formato)
  datosSMTP.setMessage(estadoPuerta, false);
  // Agregar destinatarios, se puede agregar más de un destinatario
  datosSMTP.addRecipient("iotesp27@gmail.com");
  //Comience a enviar correo electrónico.
  if (!MailClient.sendMail(datosSMTP))
    Serial.println("Error enviando el correo, " + MailClient.smtpErrorReason());
  //Borrar todos los datos del objeto datosSMTP para liberar memoria
  datosSMTP.empty();
  delay(100);
  Serial.println("Correo Puerta Enviado!");
}
```

Figure 26 CODIGO

## F. EXPERIENCIA 7°

Flujograma

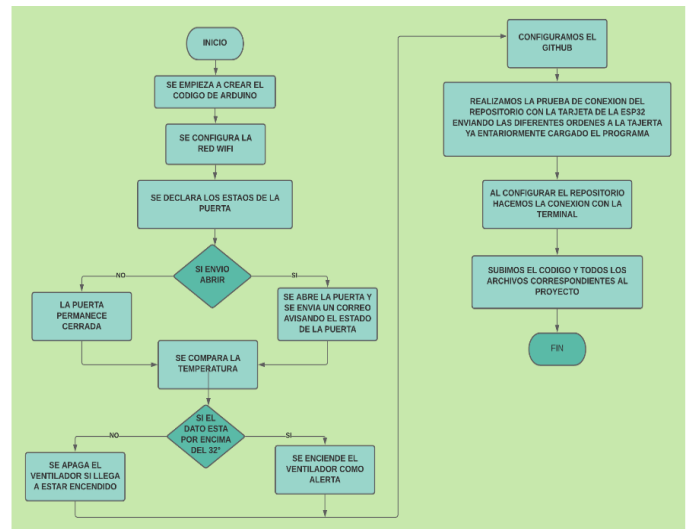


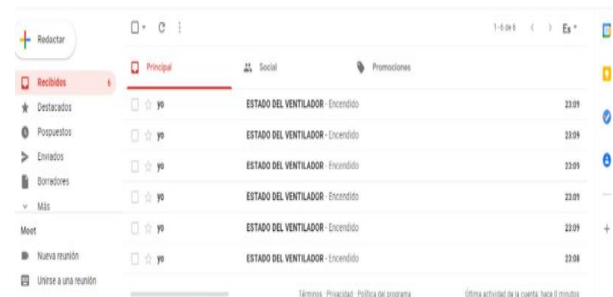
Figure 27 Flujograma

## V. RESULTADOS

enlace de video de los resultados y repositorio , copiar y pegar la dirección en la web, los videos ya se compartieron en el drive.

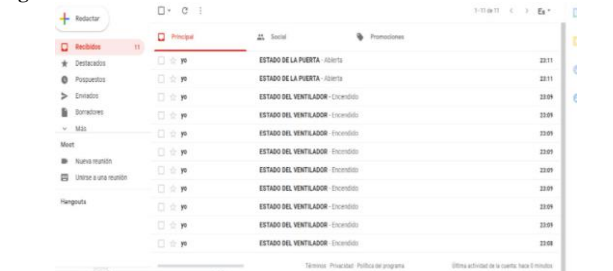
<https://drive.google.com/drive/u/0/folders/1ISJyMnUUhEOhrEWhToa1CaDnMg5cDpPr>

<https://github.com/miguelgutierrez02/laboratorio>



Destinatario	Asunto	Estado	Fecha
iotesp27@gmail.com	ESTADO DE LA PUERTA	Enviado	23:09
iotesp27@gmail.com	ESTADO DE LA PUERTA	Enviado	23:09
iotesp27@gmail.com	ESTADO DE LA PUERTA	Enviado	23:09
iotesp27@gmail.com	ESTADO DE LA PUERTA	Enviado	23:09
iotesp27@gmail.com	ESTADO DE LA PUERTA	Enviado	23:09
iotesp27@gmail.com	ESTADO DE LA PUERTA	Enviado	23:09

Figure 28 RESULTADOS DEL MENSAJE ENVIADO AL CORREO

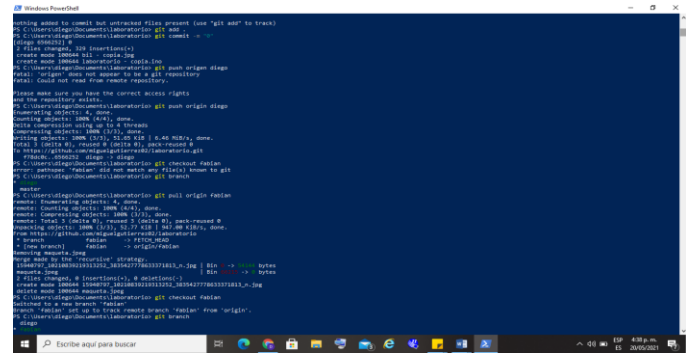


Destinatario	Asunto	Estado	Fecha
iotesp27@gmail.com	ESTADO DE LA PUERTA	Recibido	23:11
iotesp27@gmail.com	ESTADO DE LA PUERTA	Recibido	23:11
iotesp27@gmail.com	ESTADO DE LA PUERTA	Recibido	23:09
iotesp27@gmail.com	ESTADO DE LA PUERTA	Recibido	23:09
iotesp27@gmail.com	ESTADO DE LA PUERTA	Recibido	23:09
iotesp27@gmail.com	ESTADO DE LA PUERTA	Recibido	23:09

Figure 29 RESULTADOS DEL MENSAJE ENVIADO AL CORREO

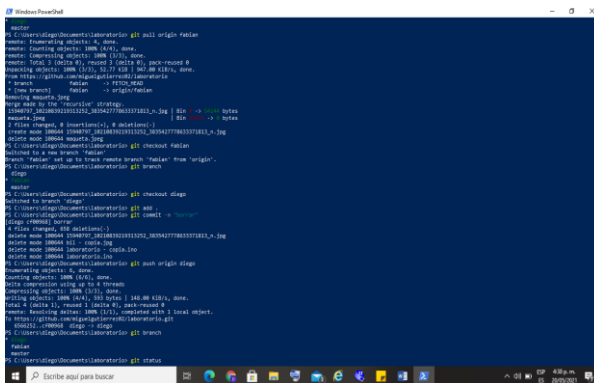
## VI. CONCLUSIONES

- Para concluir GitHub también ofrece una serie de herramientas propias con las que complementar las ventajas que ya tiene el sistema Git de por sí solo. Por ejemplo, puedes crear una Wiki para cada proyecto, de forma que puedas ofrecer toda la información sobre él y anotar todos los cambios de las diferentes versiones
- Finalmente con la plataforma de github se puede llevar el control una de las tareas fundamentales para la administración de un proyecto de desarrollo de software en general.
- En relación a lo expuesto, el protocolo MQTT es un protocolo ampliable y fácil de usar, gracias a su estructura de mensajería push con patrón publicador/suscriptor (pub-sub).
- Finalmente El protocolo MQTT requiere un ancho de banda mínimo, lo cual es importante en redes inalámbricas (3G), o conexiones con posibles problemas de calidad.

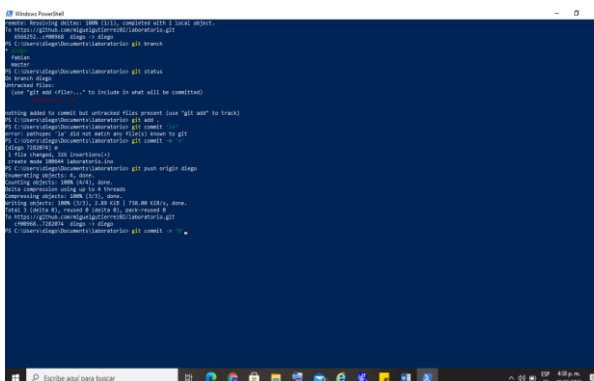


### Figure 32 COMANDOS GIT

## VII. ANEXOS



### Figure 30 COMANDOS GIT



**Figure 31 COMANDOS GIT**