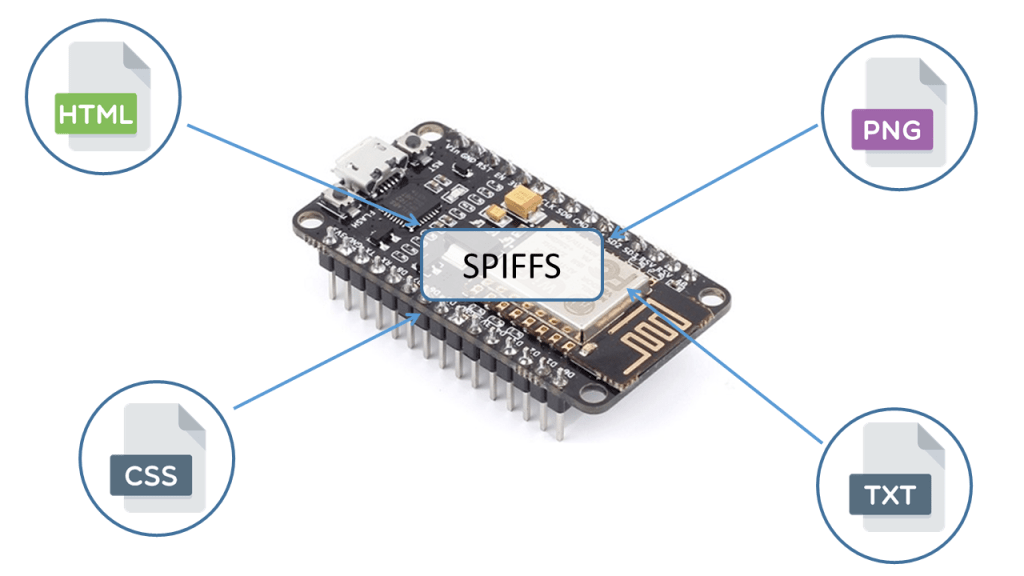
<https://randomnerdtutorials.com/install-esp8266-filesystem-uploader-arduino-ide/>

**Install ESP8266 Filesystem Uploader**

The ESP8266 contains a Serial Peripheral Interface Flash File System (SPIFFS).

SPIFFS is a lightweight filesystem created for microcontrollers with a flash chip.

This article shows how to easily upload files to the ESP8266 filesystem using a plugin for Arduino IDE.



**Introducing SPIFFS**

SPIFFS permite el acceso a la chip de memoria flash de forma simple y limitada, puede leer escribir, crear, borrar archivos, una limitacion es que no se crea directorios.

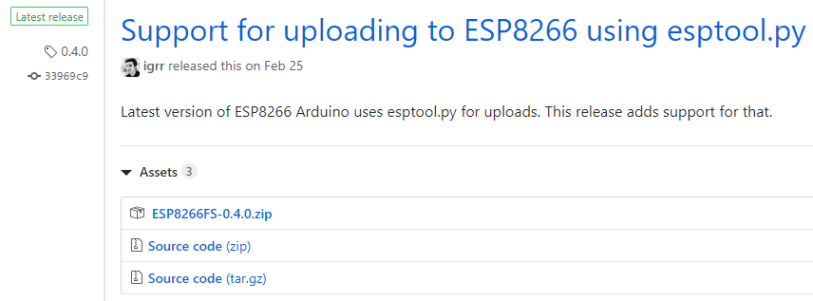
Se usa SPIFFS en ESP8266 para

* Crear archivos de configuración
* Guardar datos
* Una alternativa a usar micro SD Card
* Guardar archivos separados de HTML y CSS para crear un web server
* Guardar imágenes, iconos.
* Y mas.

**Installing ESP8266 Filesystem Uploader**

Follow the next steps to install the filesystem uploader:

**1)** Go to the [releases page and click the ESP8266FS-X.zip](https://github.com/esp8266/arduino-esp8266fs-plugin/releases) file to download.



**2)** Go to the Arduino IDE directory, and open the **Tools** folder.

Arduino IDE Tools to Install ESP8266 SPIFFS Filesystem fs

**3)** Unzip the downloaded *.zip* folder to the **Tools** folder.:

**<home\_dir>/Arduino-<version>**/**tools/ESP8266FS/tool/esp8266fs.jar**

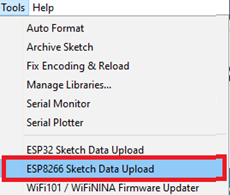
**Ayudara al conocer donde esta el ejecutable de Arduino**

**C:\Programs\arduino-1.8.9\tools\esp8266\tools\esp8266fs.jar**

**4)** Finally, restart your Arduino IDE.

Select your ESP8266 board.

In the **Tools** menu check that you have the option “**ESP8266 Sketch Data Upload**“.

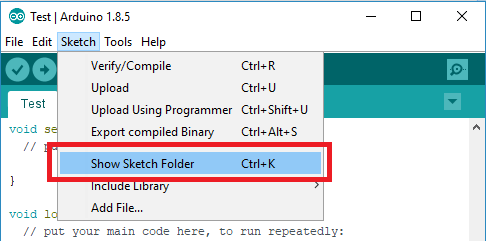


**Uploading Files using the Filesystem Uploader**

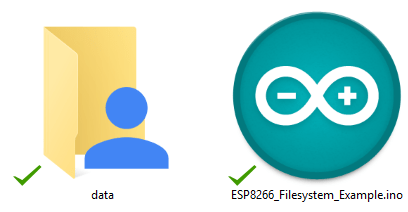
To upload files to the ESP8266 filesystem follow the next instructions.

**1)** Create an Arduino sketch and save it. For demonstration purposes, you can save an empty sketch.

**2)** Then **Sketch**> **Show Sketch Folder**. The folder where your sketch is saved should open.

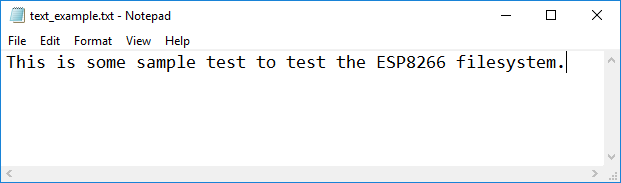


**3)** Inside that folder, create a new folder called ***data***.

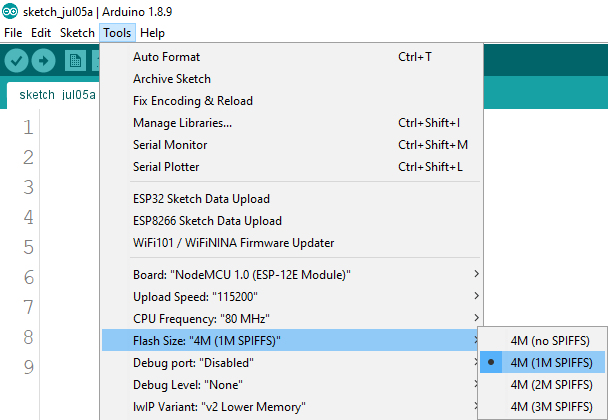


**4)** Inside the ***data***folder is where you should put the files you want to be saved into the ESP8266 filesystem.

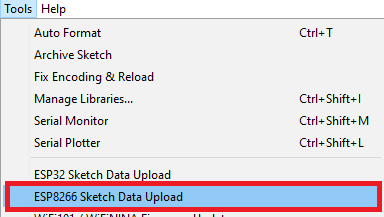
As an example, create a *.txt* file with some text called *test\_example*.



**5)** In the Arduino IDE, in the **Tools**menu, select the desired SPIFFS size (this will depend on the size of your files)



**6)** Then, to upload the files, in the Arduino IDE, you just need to go to **Tools**> **ESP8266 Sketch Data Upload**.



You should get a similar message on the debugging window. The files were successfully uploaded to the ESP8266 filesystem.

SPIFFS Image Connecting to ESP8266 board


**Testing the Uploader**

Now, let’s just check if the file was actually saved into the ESP8266 filesystem. Simply upload the following code to your ESP8266 board.

#include "FS.h"

void setup() {

Serial.begin(115200);

if(!SPIFFS.begin()){

Serial.println("An Error has occurred while mounting SPIFFS");

return;

}

File file = SPIFFS.open**("/test\_example.txt", "r");**

if(!file){

Serial.println("Failed to open file for reading");

return;

}

Serial.println();

Serial.println("File Content:");

while(file.available()){

Serial.write(file.read());

}

file.close();

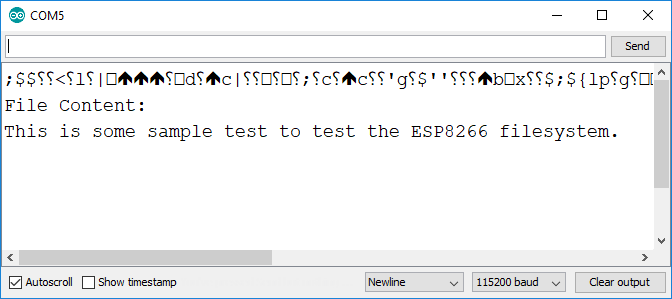
}

void loop() {

}

After uploading, open the Serial Monitor at a baud rate of 115200.

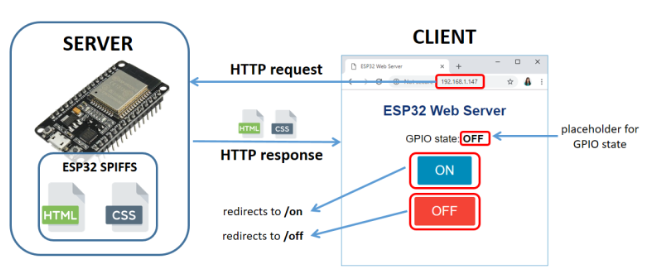
Press the ESP8266 “**RST**” button. It should print the content of your *.txt* file on the Serial Monitor.



You’ve successfully uploaded files to the ESP8266 filesystem using the plugin.

<https://randomnerdtutorials.com/esp32-web-server-spiffs-spi-flash-file-system/>

# ESP32 Web Server using SPIFFS (SPI Flash File System)



## Installing Libraries

In most of our projects we’ve created the HTML and CSS files for the web server as a String directly on the Arduino sketch. With SPIFFS, you can write the HTML and CSS in separated files and save them on the ESP32 filesystem.

One of the easiest ways to build a web server using files from the filesystem is by using the ESPAsyncWebServer library. The ESPAsyncWebServer library is well documented on its GitHub page. For more information about that library, check the following link:

* <https://github.com/me-no-dev/ESPAsyncWebServer>

**Installing the ESPAsyncWebServer library**

Follow the next steps to install the [ESPAsyncWebServer](https://github.com/me-no-dev/ESPAsyncWebServer" \t "_blank) library:

1. [Click here to download](https://github.com/me-no-dev/ESPAsyncWebServer/archive/master.zip) the ESPAsyncWebServer library. You should have a .zip folder in your Downloads folder
2. Unzip the .zip folder and you should get ESPAsyncWebServer-master folder
3. Rename your folder from  to ESPAsyncWebServer
4. Move the ESPAsyncWebServer folder to your Arduino IDE installation libraries folder

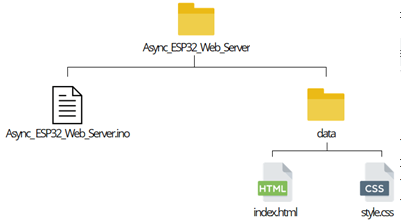
**Installing the Async TCP Library for ESP32**

The [ESPAsyncWebServer](https://github.com/me-no-dev/ESPAsyncWebServer" \t "_blank) library requires the [AsyncTCP](https://github.com/me-no-dev/AsyncTCP" \t "_blank) library to work. Follow the next steps to install that library:

1. [Click here to download](https://github.com/me-no-dev/AsyncTCP/archive/master.zip) the AsyncTCP library. You should have a .zip folder in your Downloads folder
2. Unzip the .zip folder and you should get AsyncTCP-master folder
3. Rename your folder from  to AsyncTCP
4. Move the AsyncTCPfolder to your Arduino IDE installation libraries folder
5. Finally, re-open your Arduino IDE

## Organizing your Files

To build the web server you need three different files. The Arduino sketch, the HTML file and the CSS file. The HTML and CSS files should be saved inside a folder called **data** inside the Arduino sketch folder, as shown below:



## Creating the HTML File

The HTML for this project is very simple. We just need to create a heading for the web page, a paragraph to display the GPIO state and two buttons.

<!DOCTYPE html>

<html>

<head>

<title>ESP32 Web Server</title>

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="icon" href="data:,">

<link rel="stylesheet" type="text/css" href="**style.css**">

</head>

<body>

<h1>ESP32 Web Server</h1>

<p>GPIO state: <strong> %STATE%</strong></p>

<p><a href="/on"><button class="button">ON</button></a></p>

<p><a href="/off"><button class="button button2">OFF</button></a></p>

</body>

</html>

Because we’re using CSS and HTML in different files, we need to reference the CSS file on the HTML text. The following line should be added between the <head> </head> tags:

<link rel="stylesheet" type="text/css" href="style.css">

The <link> tag tells the HTML file that you’re using an external style sheet to format how the page looks. The **rel** attribute specifies the nature of the external file, in this case that it is a **stylesheet**—the CSS file—that will be used to alter the appearance of the page.

The **type** attribute is set to **“text/css”** to indicate that you’re using a CSS file for the styles. The **href** attribute indicates the file location; since both the CSS and HTML files will be in the same folder, you just need to reference the filename: **style.css**.

In the following line, we write the first heading of our web page. In this case we have

“ESP32 Web Server”. You can change the heading to any text you want:

<h1>ESP32 Web Server</h1>

Then, we add a paragraph with the text “GPIO state: ” followed by the GPIO state. Because the GPIO state changes accordingly to the state of the GPIO, we can add a placeholder that will then be replaced for whatever value we set on the Arduino sketch.

To add placeholder we use **%** signs. To create a placeholder for the state, we can use %STATE%, for example.

<p>GPIO state: <strong>%STATE%</strong></p>

Attributing a value to the STATE placeholder is done in the Arduino sketch.

Then, we create an ON and an OFF buttons. When you click the on button, we redirect the web page to to root followed by /on url. When you click the off button you are redirected to the /off url.

<p><a href="/on"><button class="button">ON</button></a></p>

<p><a href="/off"><button class="button button2">OFF</button></a></p>

## Creating the CSS file

Create the style.css file with the following content or [download all the project files here](https://github.com/RuiSantosdotme/ESP32-Course/raw/master/code/SPIFFS/ESP32_Async_Web_Server.zip):

html {

font-family: Helvetica;

display: inline-block;

margin: 0px auto;

text-align: center;

}

h1{

color: #0F3376;

padding: 2vh;

}

p{

font-size: 1.5rem;

}

.button {

display: inline-block;

background-color: #008CBA;

border: none;

border-radius: 4px;

color: white;

padding: 16px 40px;

text-decoration: none;

font-size: 30px;

margin: 2px;

cursor: pointer;

}

.button2 {

background-color: #f44336;

}

This is just a basic CSS file to set the font size, style and color of the buttons and align the page. We won’t explain how CSS works. A good place to learn about CSS is the [W3Schools website](https://www.w3schools.com/css/default.asp).

## Arduino Sketch

Copy the following code to the Arduino IDE or [download all the project files here](https://github.com/RuiSantosdotme/ESP32-Course/raw/master/code/SPIFFS/ESP32_Async_Web_Server.zip). Then, you need to type your network credentials (SSID and password) to make it work.

/\*\*\*\*\*\*\*\*\*

Rui Santos

Complete project details at https://randomnerdtutorials.com

\*\*\*\*\*\*\*\*\*/

#include "WiFi.h"

#include "ESPAsyncWebServer.h"

#include "SPIFFS.h"

const char\* ssid = "REPLACE\_WITH\_YOUR\_SSID";

const char\* password = "REPLACE\_WITH\_YOUR\_PASSWORD";

// Set LED GPIO

const int ledPin = 2;

String ledState;

// Create AsyncWebServer object on port 80

AsyncWebServer server(80);

// Replaces placeholder with LED state value

String processor(const String& var){

Serial.println(var);

if(var == "STATE"){

if(digitalRead(ledPin)){

ledState = "ON";

}

else{

ledState = "OFF";

}

Serial.print(ledState);

return ledState;

}

return String();

}

**void setup(){**

Serial.begin(115200);

pinMode(ledPin, OUTPUT);

// Initialize SPIFFS

if(!SPIFFS.begin(true)){

Serial.println("An Error has occurred while mounting SPIFFS");

return;

}

// Connect to Wi-Fi

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED) {

delay(1000);

Serial.println("Connecting to WiFi..");

}

Serial.println(WiFi.localIP());

// Route for root / web page

server.on("/", HTTP\_GET, [](AsyncWebServerRequest \*request){

request->send(SPIFFS, **"/index.html**", String(), false, processor);

});

// Route to load style.css file

server.on(**"/style.css"**, HTTP\_GET, [](AsyncWebServerRequest \*request){

request->send(SPIFFS, **"/style.css"**, "**text/css**");

});

// Route to set GPIO to HIGH

server.on("/on", HTTP\_GET, [](AsyncWebServerRequest \*request){

digitalWrite(ledPin, HIGH);

request->send(SPIFFS, **"/index.html"**, String(), false, processor);

});

// Route to set GPIO to LOW

server.on("/off", HTTP\_GET, [](AsyncWebServerRequest \*request){

digitalWrite(ledPin, LOW);

request->send(SPIFFS, "/index.html", String(), false, processor);

});

// Start server

server.begin();

}

void loop(){

}

### How the Code Works

First, include the necessary libraries:

#include "WiFi.h"

#include "ESPAsyncWebServer.h"

#include "SPIFFS.h"

You need to type your network credentials in the following variables:

const char\* ssid = "REPLACE\_WITH\_YOUR\_SSID";

const char\* password = "REPLACE\_WITH\_YOUR\_PASSWORD";

Next, create a variable that refers to GPIO 2 called ledPin, and a String variable to hold the led state: ledState.

const int ledPin = 2;

String ledState;

Create an AsynWebServer object called server that is listening on port 80.

AsyncWebServer server(80);

### processor()

The processor() function is what will attribute a value to the placeholder we’ve created on the HTML file. It accepts as argument the placeholder and should return a String that will replace the placeholder. The processor() function should have the following structure:

String processor(const String& var){

Serial.println(var);

if(var == "STATE"){

if(digitalRead(ledPin)){

ledState = "ON";

}

else{

ledState = "OFF";

}

Serial.print(ledState);

return ledState;

}

return String();

}

This function first checks if the placeholder is the STATE we’ve created on the HTML file.

if(var == "STATE"){

If it is, then, accordingly to the LED state, we set the ledState variable to either ON or OFF.

if(digitalRead(ledPin)){

ledState = "ON";

}

else{

ledState = "OFF";

}

Finally, we return the ledState variable. This replaces the placeholder with the ledState string value.

return ledState;

### setup()

In the setup(), start by initializing the Serial Monitor and setting the GPIO as an output.

Serial.begin(115200);

pinMode(ledPin, OUTPUT);

Initialize SPIFFS:

if(!SPIFFS.begin(true)){

Serial.println("An Error has occurred while mounting SPIFFS");

return;

}

**Wi-Fi connection**

Connect to Wi-Fi and print the ESP32 IP address:

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED) {

delay(1000);

Serial.println("Connecting to WiFi..");

}

Serial.println(WiFi.localIP());

**Async Web Server**

The ESPAsyncWebServer library allows us to configure the routes where the server will be listening for incoming HTTP requests and execute functions when a request is received on that route. For that, use the on() method on the server object as follows:

server.on("/", HTTP\_GET, [](AsyncWebServerRequest \*request){

request->send(SPIFFS, "/index.html", String(), false, processor);

});

When the server receives a request on the root “**/**” URL, it will send the index.html file to the client. The last argument of the send() function is the processor, so that we are able to replace the placeholder for the value we want – in this case the ledState.

Because we’ve referenced the CSS file on the HTML file, the client will make a request for the CSS file. When that happens, the CSS file is sent to the client:

server.on("/style.css", HTTP\_GET, [](AsyncWebServerRequest \*request){

request->send(SPIFFS, "/style.css","text/css");

});

Finally, you need to define what happens on the /on and /off routes. When a request is made on those routes, the LED is either turned on or off, and the ESP32 serves the HTML file.

server.on("/on", HTTP\_GET, [](AsyncWebServerRequest \*request){

digitalWrite(ledPin, HIGH);

request->send(SPIFFS, "/index.html", String(),false, processor);

});

server.on("/off", HTTP\_GET, [](AsyncWebServerRequest \*request){

digitalWrite(ledPin, LOW);

request->send(SPIFFS, "/index.html", String(),false, processor);

});

In the end, we use the begin() method on the server object, so that the server starts listening for incoming clients.

server.begin();

Because this is an asynchronous web server, you can define all the requests in the setup(). Then, you can add other code to the loop() while the server is listening for incoming clients.

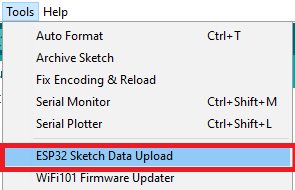
## Uploading Code and Files

Save the code as Async\_ESP32\_Web\_Server or [download all the project files here](https://github.com/RuiSantosdotme/ESP32-Course/raw/master/code/SPIFFS/ESP32_Async_Web_Server.zip). Go to **Sketch** > **Show Sketch Folder**, and create a folder called **data**. Inside that folder you should save the HTML and CSS files.

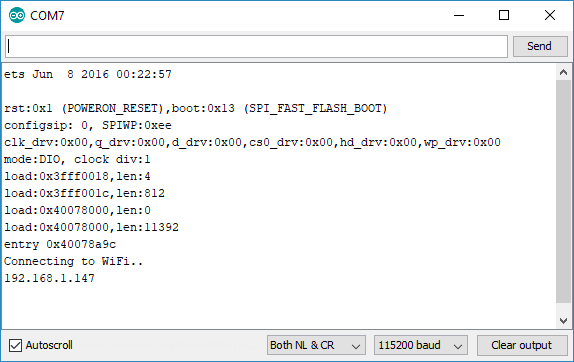
Then, upload the code to your ESP32 board. Make sure you have the right board and COM port selected. Also, make sure you’ve added your networks credentials to the code.

https://i0.wp.com/randomnerdtutorials.com/wp-content/uploads/2016/12/arduino-ide-upload-button.png?resize=34%2C29&quality=100&strip=all&ssl=1

After uploading the code, you need to upload the files. Go to **Tools**> **ESP32 Data Sketch Upload** and wait for the files to be uploaded.



When everything is successfully uploaded, open the Serial Monitor at a baud rate of 115200. Press the ESP32 “**ENABLE**” button, and it should print the ESP32 IP address.



## Demonstration

Open your browser and type the ESP32 IP address. Press the ON and OFF buttons to control the ESP32 on-board LED. Also, check that the GPIO state is being updated correctly.



<html>

<head>

<title>ESP32 Web Server</title>

</head>

<body>

<h1>ESP32 Web Server</h1>

<p>GPIO state: <strong> %STATE%</strong></p>

<p><a href="/on"><button class="button">ON</button></a></p>

<p><a href="/off"><button class="button button2">OFF</button></a></p>

</body>

</html>