ESP32

Upgrade Your ESP32 Firmware Remotely (OTA) Using GitHub

Kamruzzaman Tipu
18 Dec 2025 — 7 min read

Learn how to perform *true* over-the-air (OTA) updates on ESP32-based IoT devices anywhere in the world using a public GitHub repository as a free, reliable firmware server.

Introduction: The Real OTA Problem in IoT

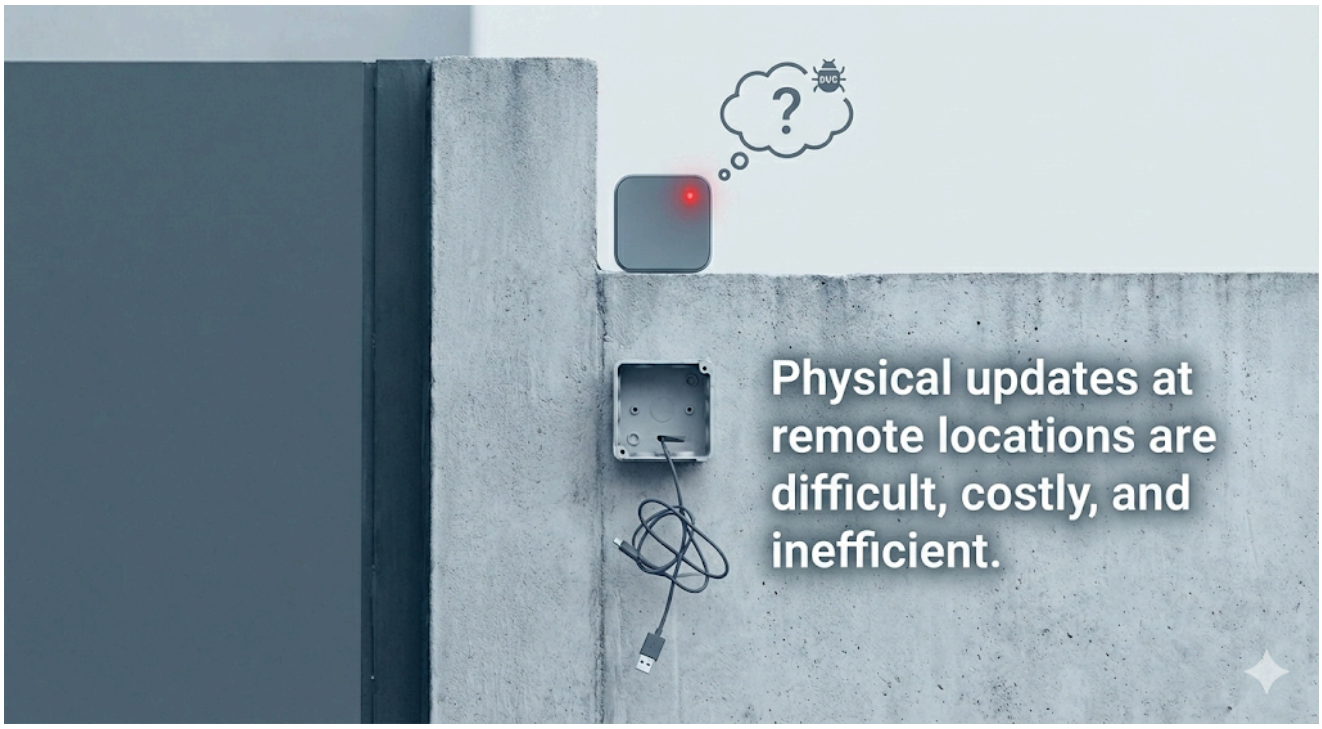
If you've ever deployed an ESP32 project in the real world, you already know the

IoT Bhai

You finish the prototype. You solder everything cleanly. You mount the device on a wall, inside a control box, on a pole, in an attic, or maybe somewhere outdoors. Then a few days later… you notice a bug. Or worse, a missing feature.

Now comes the nightmare scenario:

- Physical access is difficult or expensive
- The device is installed far from your desk
- USB flashing means downtime and manual labor



This is **not scalable IoT**.

That’s why **OTA (Over-the-Air) firmware updates** are not a luxury — they’re a requirement.

Many tutorials stop at *local OTA* (same WiFi network, Arduino IDE pushing code). That’s fine for demos, but it completely breaks down in real deployments.

IoT Bhai

- No port forwarding
- No local discovery
- Fully autonomous update logic

In this guide — based on my recent YouTube tutorial — we'll build a **production-style OTA system** where:

- The ESP32 checks GitHub for new firmware
- Compares versions safely
- Downloads the update automatically
- Installs and reboots without user interaction

And best of all — **GitHub is used as a free, highly reliable OTA server**.

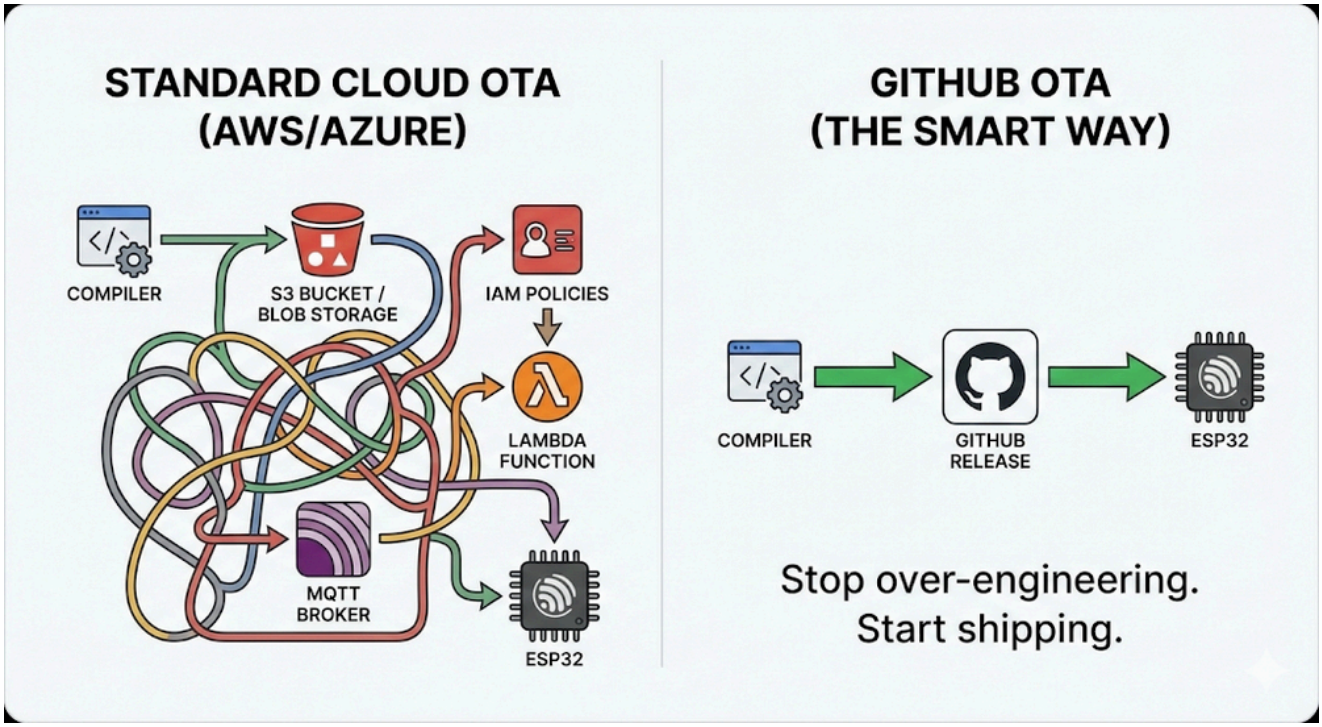
IoT Bhai

Why Use GitHub for ESP32 OTA?

There are many OTA approaches available today:

- AWS IoT
- Azure IoT Hub
- Firebase hosting
- Blynk OTA
- Self-hosted servers

IoT Bhai



So why GitHub?

1. Completely Free

Public GitHub repositories cost nothing. No servers, no monthly bills, no bandwidth planning for small-to-medium projects.

2. Extremely Reliable

GitHub runs on infrastructure backed by Microsoft. Uptime, CDN performance, and global availability are excellent.

3. Perfect for Versioned Firmware

GitHub releases are *designed* for binary distribution:

- Version tags

IoT Bhai

- Clean URLs

4. True Internet OTA

Your ESP32 only needs:

- WiFi access
- Internet connectivity

It does **not** need to be on the same LAN as your computer.

This mirrors how commercial IoT devices work in the real world.

The Comparison

Feature	AWS / Azure	Self-Hosted / DIY	GitHub (Our Method)
Cost	Free tier (tricky limits)	$5–$20/mo (VPS)	**Free** (Public Repos)
Setup Time	Hours (Certs, Policies)	Hours (Linux config)	**Minutes**
Maintenance	High (Policy management)	High (Security updates)	**Zero**
Workflow	Manual Uploads	Manual FTP/SCP	**Git Push & Tag**

High-Level OTA Architecture

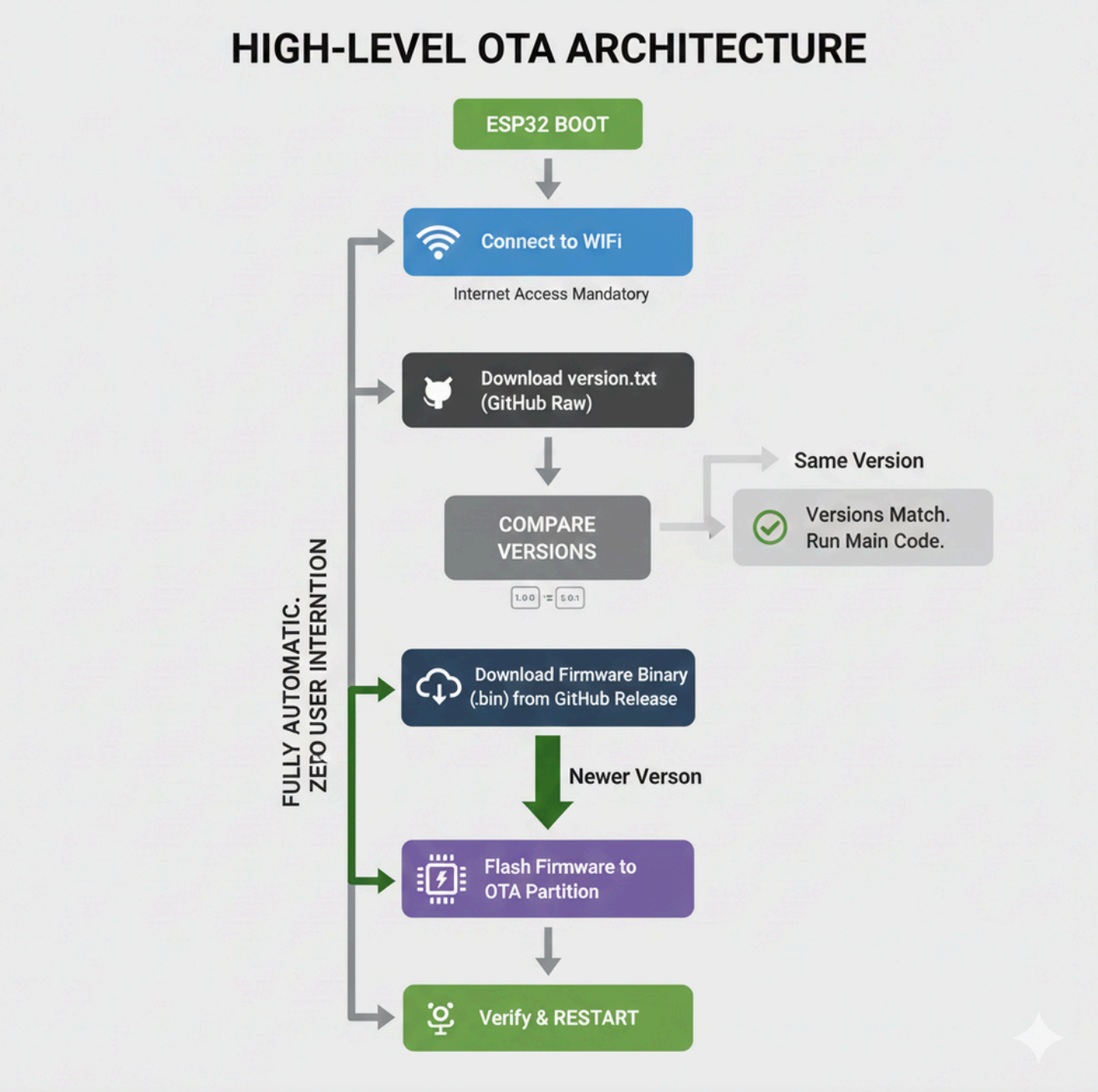
Before touching any code, let's understand the system logic.

On every boot (or scheduled interval), the ESP32 follows this sequence:

1. **Connect to WiFi**
   Internet access is mandatory.
2. **Check Firmware Version**

IoT Bhai

3. **Compare Versions**
   Compare cloud version vs currently running firmware.
4. **Decision Logic**
   - Same version → Continue normal operation
   - Newer version → Start OTA update
5. **Download Firmware Binary (.bin)**
6. **Flash Firmware to OTA Partition**
7. **Verify & Restart**



IoT Bhai

This process is fully automatic and requires **zero manual intervention** once deployed.

Watch the Video Tutorial

If you prefer learning visually, I have put together a complete step-by-step video walkthrough of this project.

Hardware Required

- ESP32 Development Board
  - 🛒 Buy on Amazon | 🛒 Buy on AliExpress

Software Tools

IoT Bhai

- **ESP32 Board Support:** You must have the ESP32 board manager installed in your IDE. *Need help? Check out my guide:* **How to Set Up ESP32 in Arduino IDE 2.0 (Windows/Ubuntu)**

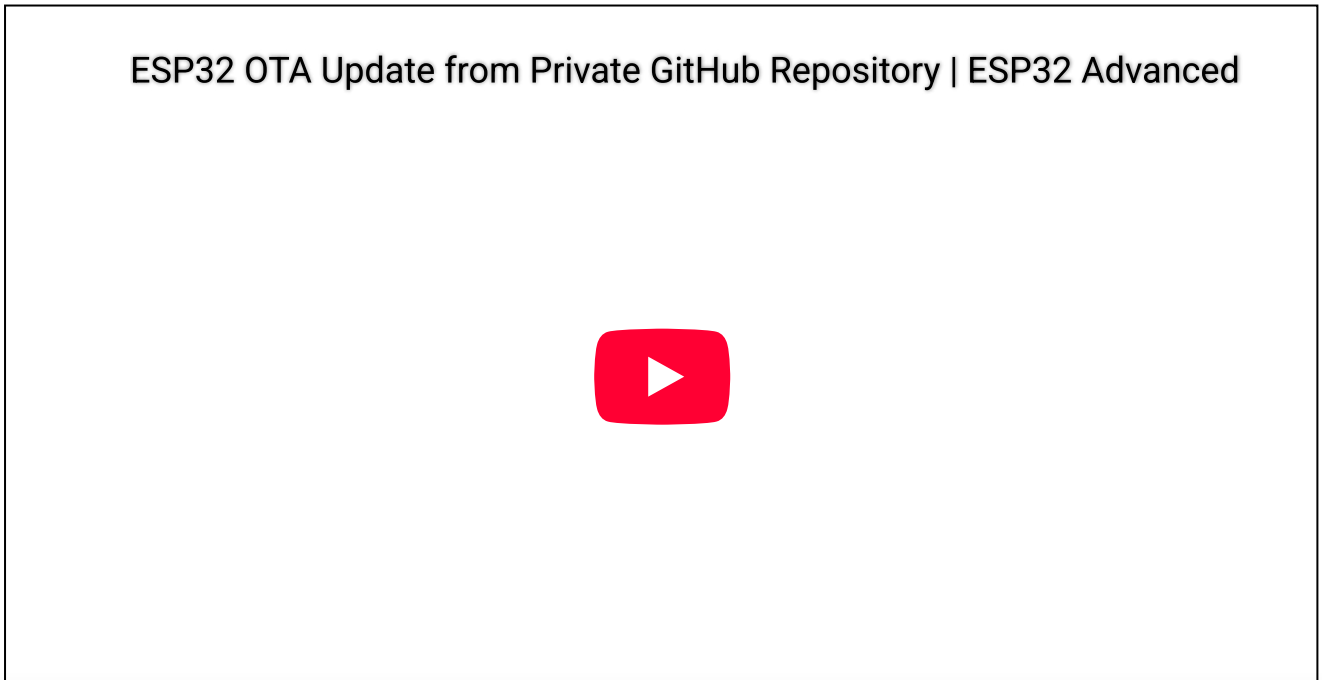
Phase 1: GitHub Repository Setup (Critical Step)

Most OTA failures happen here. Pay attention.

Step 1: Create a Public Repository

1. Go to GitHub → New Repository
2. Name it something meaningful, for example:ESP32-OTA-Firmware
3. Set **Visibility: Public**
4. Initialize with a `README.md`

> ⚠️ Private repositories require authentication tokens and HTTPS headers — that's an advanced topic and outside the scope of this guide. [Video Already in My Youtube channel)



IoT Bhai

Step 2: Create the Version File

Inside the repository:

1. Create a new file named:

```
version.txt
```

2. Add **only the version number** inside:

```
1.0.0
```

3. Commit the file

Keep this file clean:

- No spaces
- No extra lines
- No comments

The ESP32 will parse this file directly.

Step 3: Get the *Raw* File URL (Most Common Mistake)

The ESP32 cannot read GitHub webpages. It needs **raw file content**.

1. Open `version.txt` in GitHub
2. Click the **Raw** button
3. You should see a blank page with only:

IoT Bhai

4. Copy the URL from the address bar

It must look like:

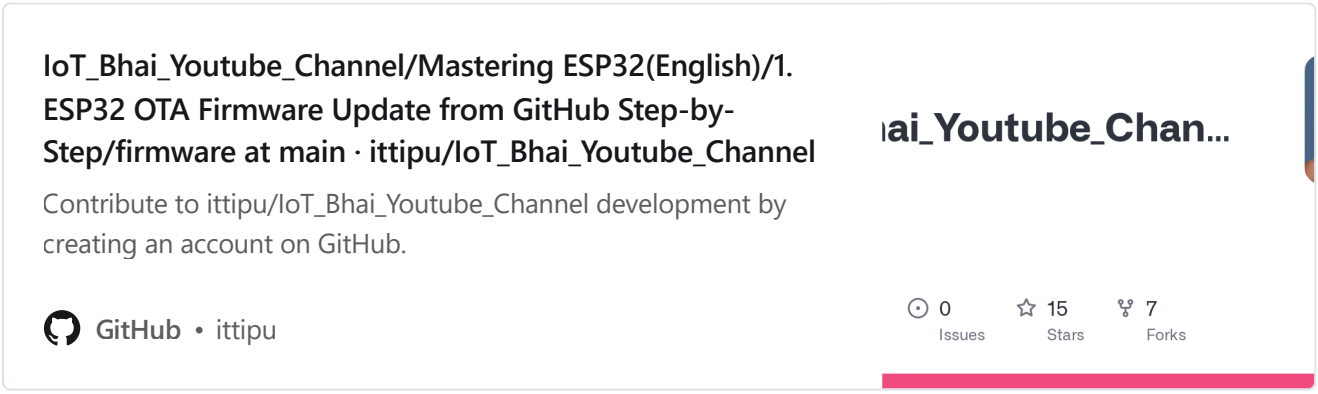
```
https://raw.githubusercontent.com/USERNAME/REPO/main/version.txt
```

Save this carefully — this is your **VERSION_FILE_URL**.

Phase 2: ESP32 Firmware Configuration (Arduino IDE)

Download the code first

Github



Now we configure the ESP32 side.

1. WiFi Credentials

Your device must reach the internet:

2. The Current Internal Version

You need a variable in your code that tells the ESP32 what version it is currently running. Let's start it at 1.0.0 to match our text file.

```
String FirmwareVer = "1.0.0";
```

3. The GitHub URLs

Paste the "Raw" link you copied in Phase 1 into the version URL variable.

You also need to define where the `.bin` file will eventually live. Even though we haven't uploaded the binary yet, we know the structure of GitHub release URLs.

- **Version URL:** The raw link to `version.txt`.
- **Firmware URL:** The link to where your release binary will be. It usually looks like: `https://github.com/<username>/<repo>/releases/download/<tag>/<filename.bin>`

```
// URL to the raw text file containing the latest version number
const char* versionUrl= "https://raw.githubusercontent.com/YOUR_USERNA

// URL to the actual firmware binary file in releases
const char* firmwareUrl = "https://github.com/YOUR_USERNAME/YOUR_REPO_
```

(Note: We will use tag `v1.0.1` for the update test, so put that in the code now).

Phase 3: OTA Deployment Workflow (Real-World Process)

IoT Bhai

- GitHub `version.txt` says `1.0.0`.
- Your ESP32 code variable `FirmwareVer` says `1.0.0`.

Result: **No update triggered** ✔

Step 1: Prepare New Firmware (v1.0.1)

Update your code:

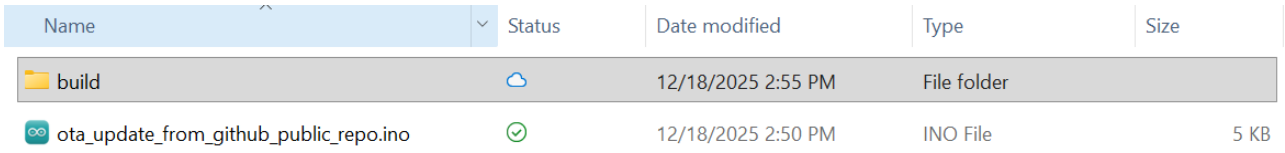
```
const char* currentFirmwareVersion = "1.0.1";
```

Step 2: Export Binary File

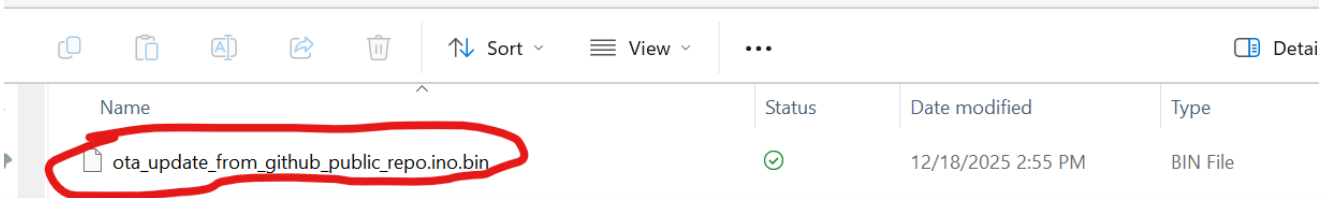
In Arduino IDE:

```
Sketch → Export Compiled Binary
```

his will compile the code and save a `.bin` file in the same folder where your sketch (`.ino`) file is saved.



bin file is the inside of build

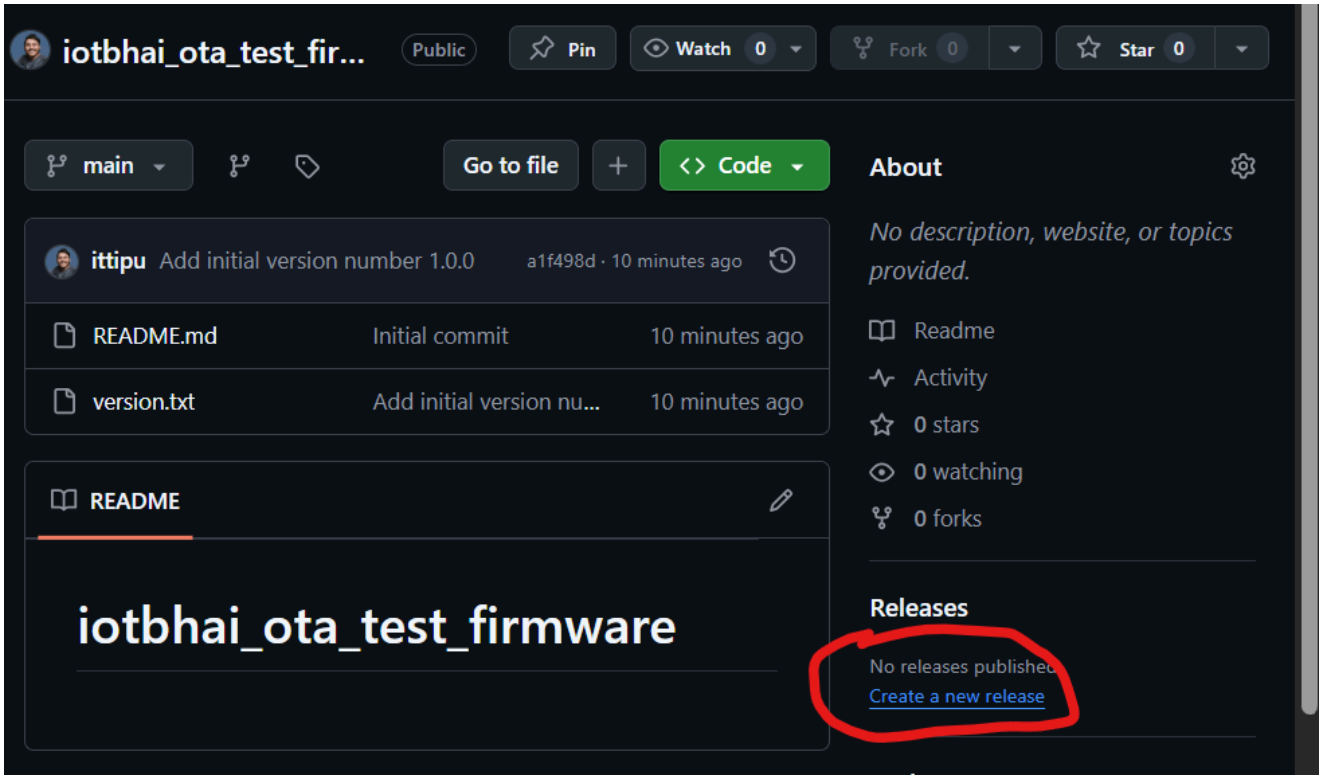


IoT Bhai

choose the .ino.bin

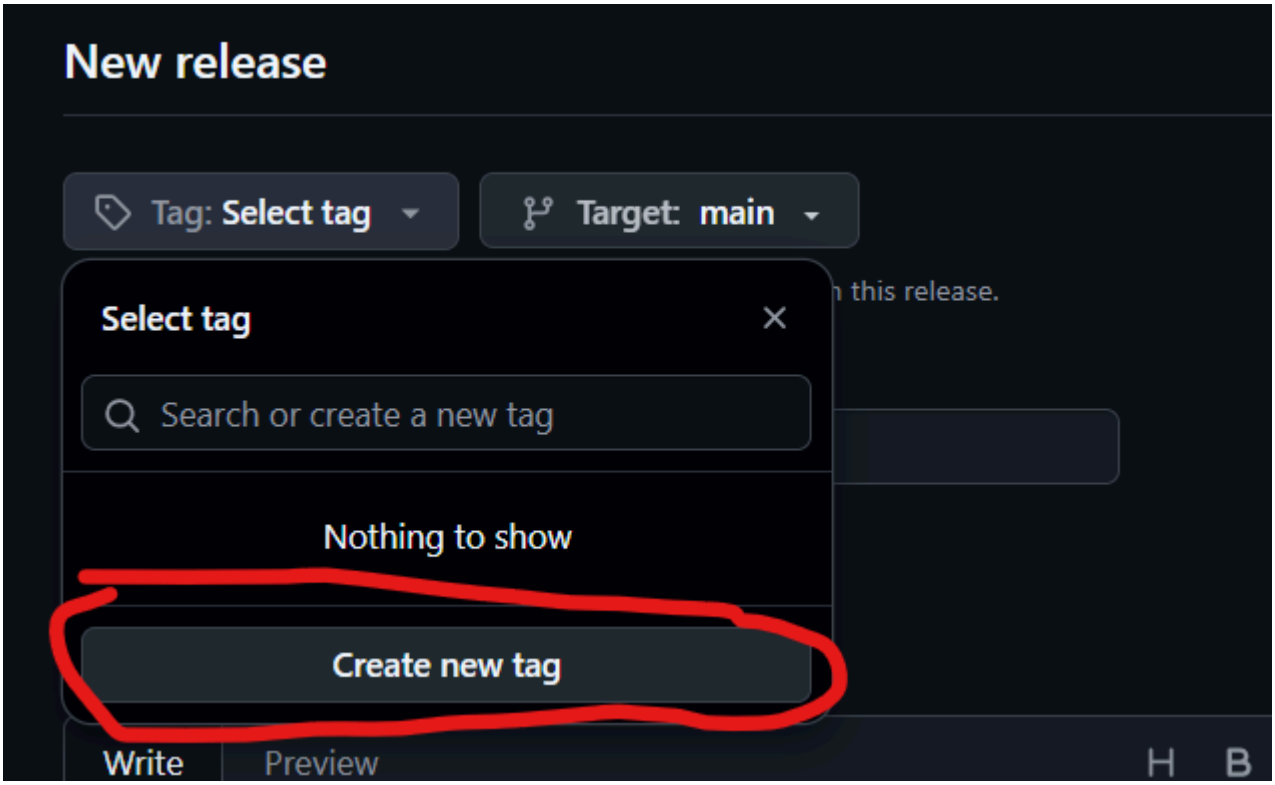
Step 3: Create a GitHub Release

- Go back to your GitHub repository main page.
- On the right side, click "Releases", then "Draft a new release".

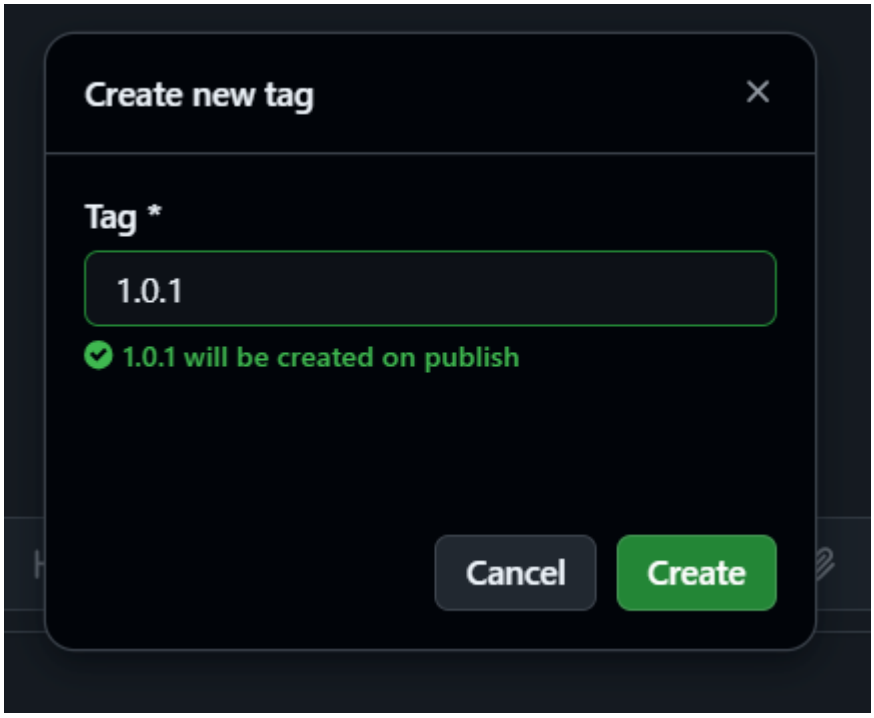


- **Tag version:** Name it `v1.0.1` (This *must* match the tag used in your `firmwareUrl` in the code!).

IoT Bhai



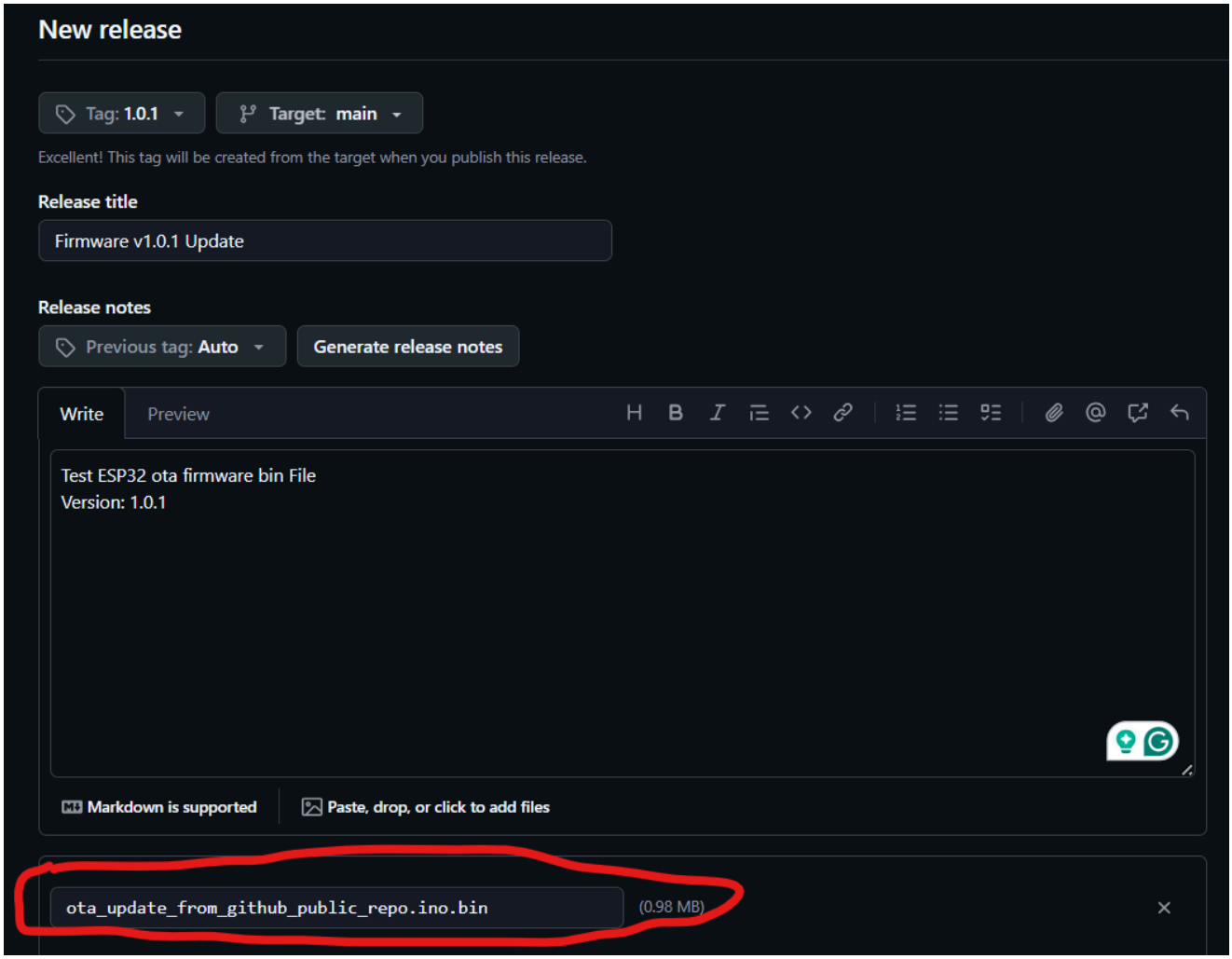
Click on Create new tag



Name it exactly as firmwareUrl tag like 1.0.1

- **Release title:** e.g., "Firmware v1.0.1 Update".

IoT Bhai



- Click **Publish release**.

Step 4: Update the Version Check File

The final trigger is updating the text file.

1. Go back to your repo code view.
2. Edit `version.txt`.
3. Change `1.0.0` to `1.0.1`.
4. Commit changes.

IoT Bhai

1. Plug your ESP32 into your computer.
2. Upload the code with version 1.0.0 because we will update it 1.0.1 with ota

```
#include <WiFi.h>
#include <HTTPClient.h>
#include <Update.h>

// WiFi credentials
const char* ssid = ""; // put your wifi name
const char* password = ""; // put your wifi password

const char* firmwareUrl = "https://github.com/ittipu/iotbhai_ota_test_firmware/releases/download/1.0.1/ota_update
const char* versionUrl = "https://raw.githubusercontent.com/ittipu/iotbhai_ota_test_firmware/refs/heads/main/vers

// Current firmware version
const char* currentFirmwareVersion = "1.0.0";
const unsigned long updateCheckInterval = 5 * 60 * 1000;  // 5 minutes in milliseconds
unsigned long lastUpdateCheck = 0;
```

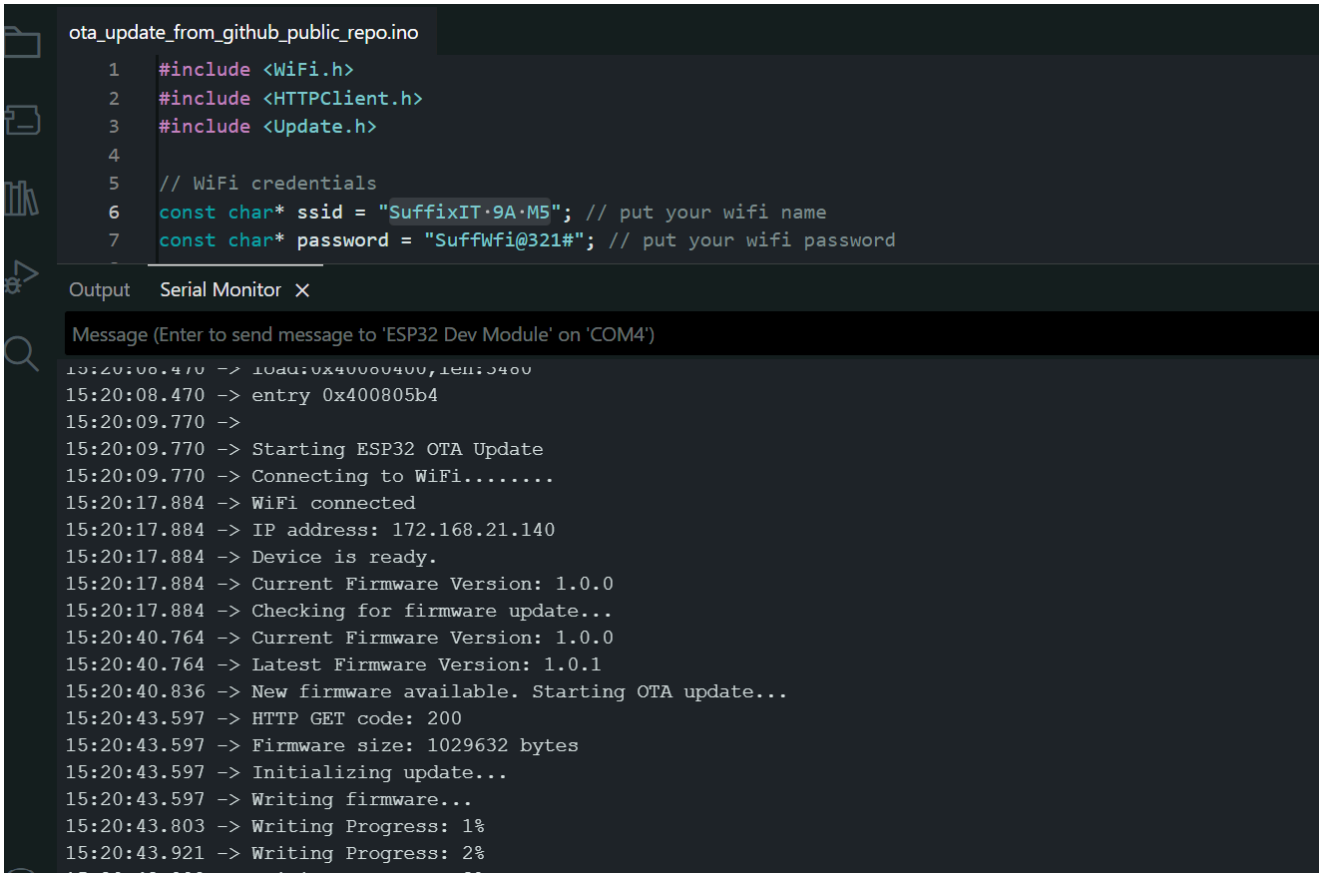
u

3. Open the Serial Monitor (115200 baud).
4. Reset the ESP32.

See the Serial monitor

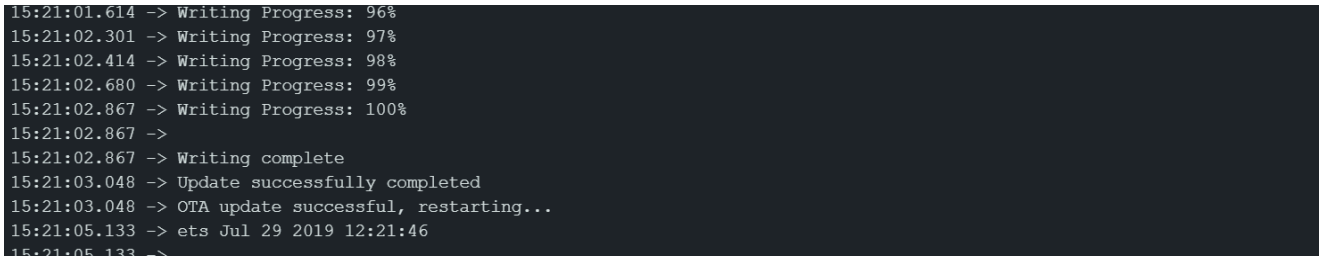
You will see the ESP32 connect to WiFi. It will then check the raw GitHub URL. It will see that the online version (`1.0.1`) is different than its current version (which is likely still `1.0.0` if you haven't uploaded the new code manually yet).

IoT Bhai



New firmware Available so it start downloading and writing

It will begin downloading the binary from the release URL. Once complete, it will restart, and you should see your new startup message confirming that Version 1.0.1 is running.



OTA Update Successfully

IoT Bhai

```
15:21:05.133 -> load:0x40078000,len:16612
15:21:05.133 -> load:0x40080400,len:3480
15:21:05.133 -> entry 0x400805b4
15:21:06.474 ->
15:21:06.474 -> Starting ESP32 OTA Update
15:21:06.474 -> Connecting to WiFi...
15:21:09.604 -> WiFi connected
15:21:09.604 -> IP address: 172.168.21.140
15:21:09.637 -> Device is ready.
15:21:09.637 -> Current Firmware Version: 1.0.1
15:21:09.637 -> Checking for firmware update...
15:21:10.679 -> Current Firmware Version: 1.0.1
15:21:10.715 -> Latest Firmware Version: 1.0.1
15:21:10.715 -> Device is up to date.
15:21:10.715 -> Current Firmware Version: 1.0.1
15:21:13.725 -> Current Firmware Version: 1.0.1
15:21:16.679 -> Current Firmware Version: 1.0.1
15:21:19.707 -> Current Firmware Version: 1.0.1
15:21:22.704 -> Current Firmware Version: 1.0.1
```

Now our device is up to date

Conclusion

You have now successfully unlocked the power of remote updates for your ESP32 projects using entirely free tools. This method is highly scalable for hobbyist projects and ensures you never have to climb a ladder to fix a bug again.

For a visual walkthrough of every step, be sure to watch the full video tutorial above!

Upgrade Your ESP32 Over-the-Air (OTA) Using GitHub [Public Repo]

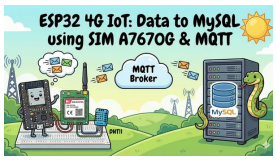
IoT Bhai

Happy deploying 🚀

0 reactions

0 comments

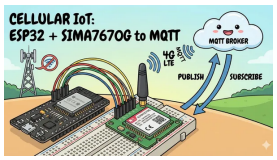
READ MORE



ESP32 4G IoT: Send Data to MySQL using SIM A7670G & MQTT

Take your IoT projects off the...

By Kamruzzaman Tipu — 31 Jan 2026



How to Connect SIM7670G 4G LTE Module to MQTT with ESP32 | Complete IoT Guide

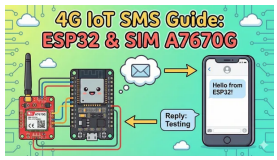
Take your IoT projects off the...

By Kamruzzaman Tipu — 28 Jan 2026

The Ultimate Guide: Headless Raspberry Pi 5 Setup (WiFi, SSH, & VNC)

Don't have a spare monitor? No...

By Kamruzzaman Tipu — 18 Jan 2026



How to Send & Receive SMS with ESP32 and SIM A7670G (4G LTE)

In Part 2 of our 4G IoT series, we...

By Kamruzzaman Tipu — 13 Jan 2026

IoT Bhai