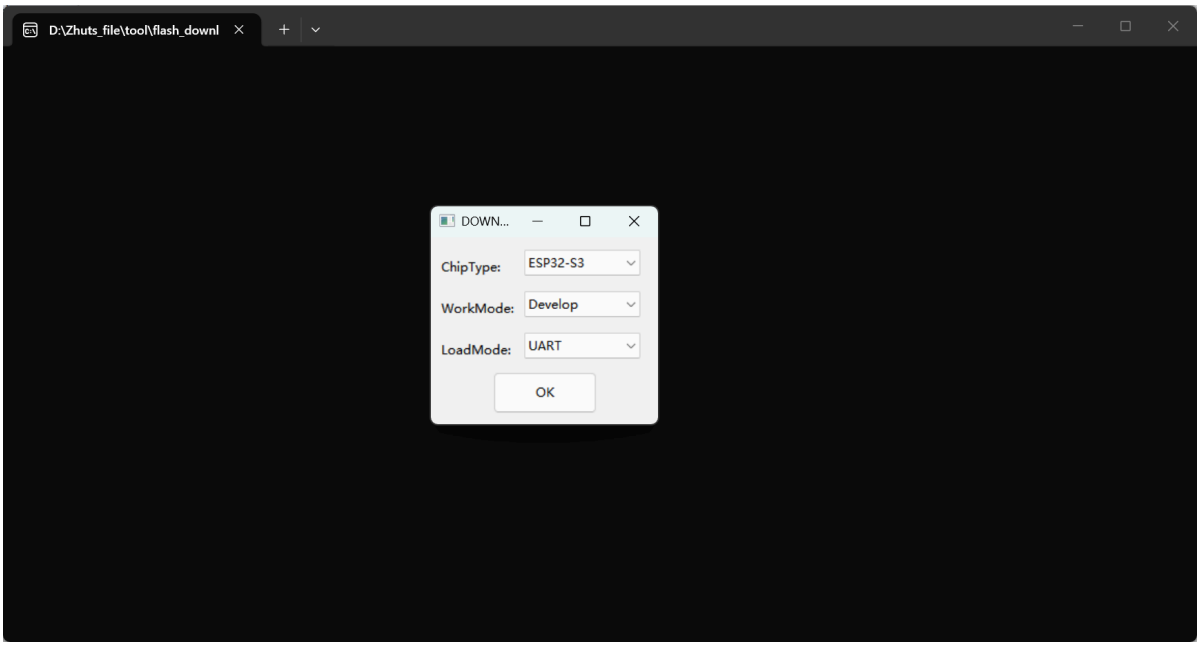
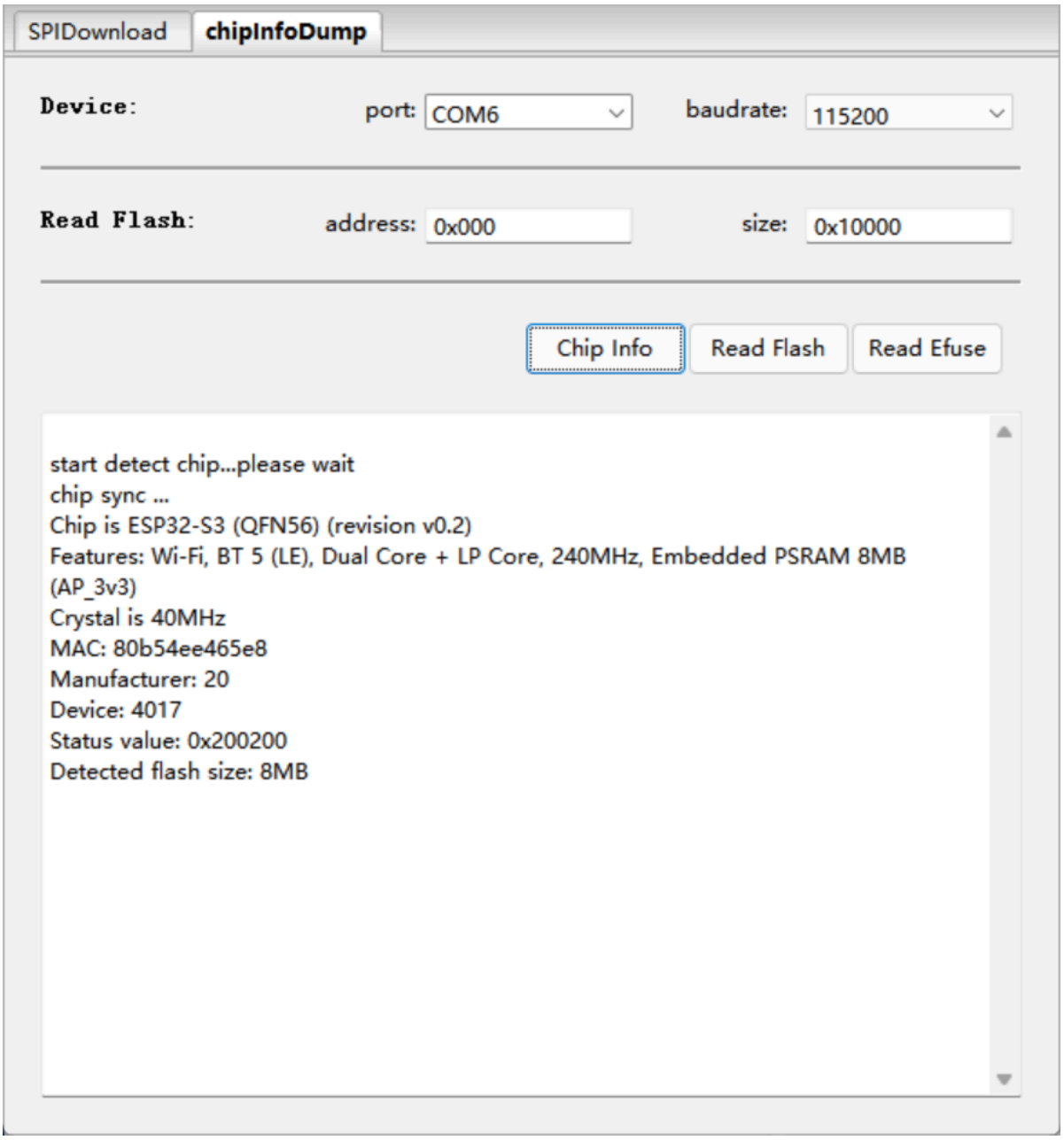
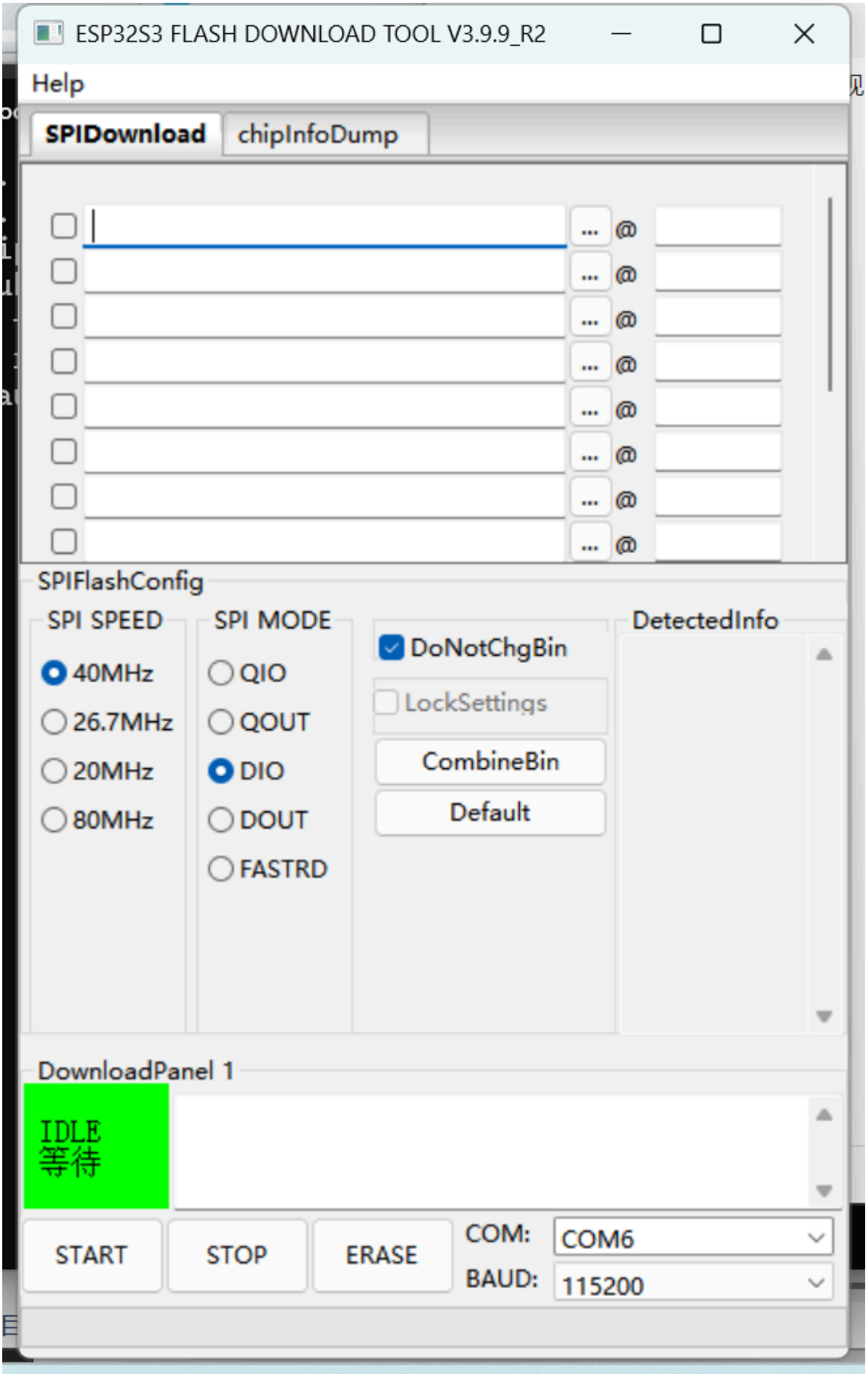
1、下载ESP32的FLASH烧录工具

https://docs.espressif.com/projects/esp-test-tools/zh_CN/latest/esp32s3/production_stage/tools/flash_download_tool.html




SPIDownload
chipInfoDump
Device:
port:
COM6
baudrate:
115200
Read Flash:
address:
0x000
size:
0x10000
Chip Info
Read Flash
Read Efuse
start detect chip...please wait
chip sync ...
Chip is ESP32-S3 (QFN56) (revision v0.2)
Features: Wi-Fi, BT 5 (LE), Dual Core + LP Core, 240MHz, Embedded PSRAM 8MB (AP_3v3)
Crystal is 40MHz
MAC: 80b54ee465e8
Manufacturer: 20
Device: 4017
Status value: 0x200200
Detected flash size: 8MB



2、安装WSL及Ubuntu

https://learn.microsoft.com/zh-cn/windows/wsl/install

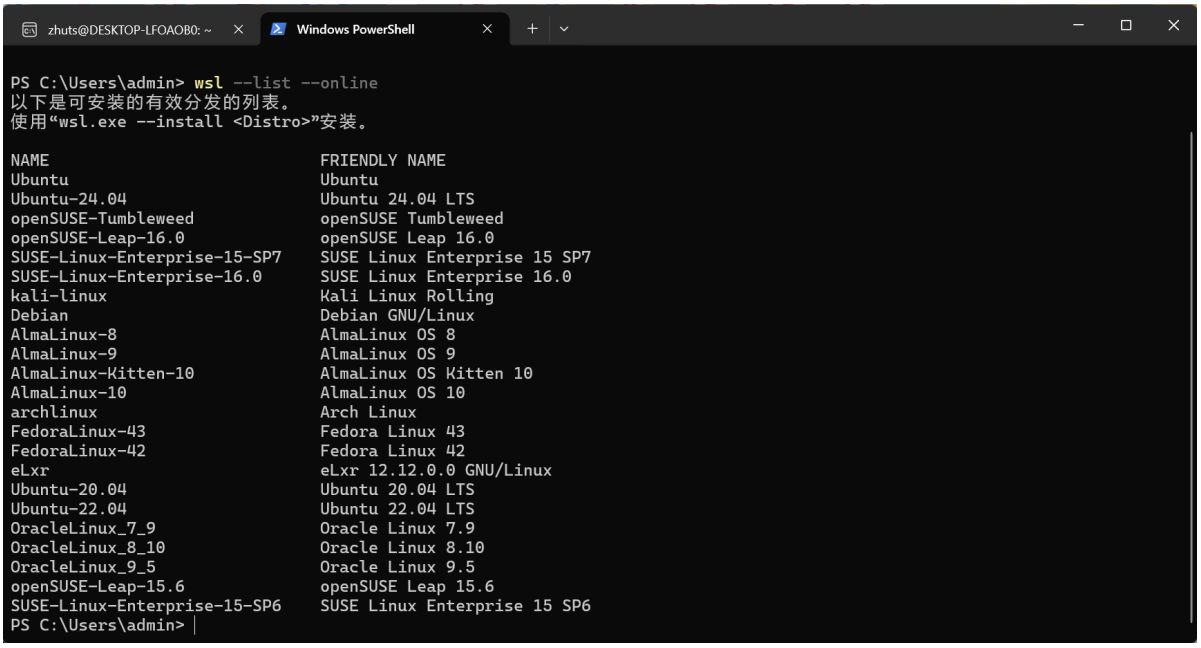
在PowerShell里输入

```
wsl --install
```

等一会会安装成功，重启

安装Ubuntu20.04

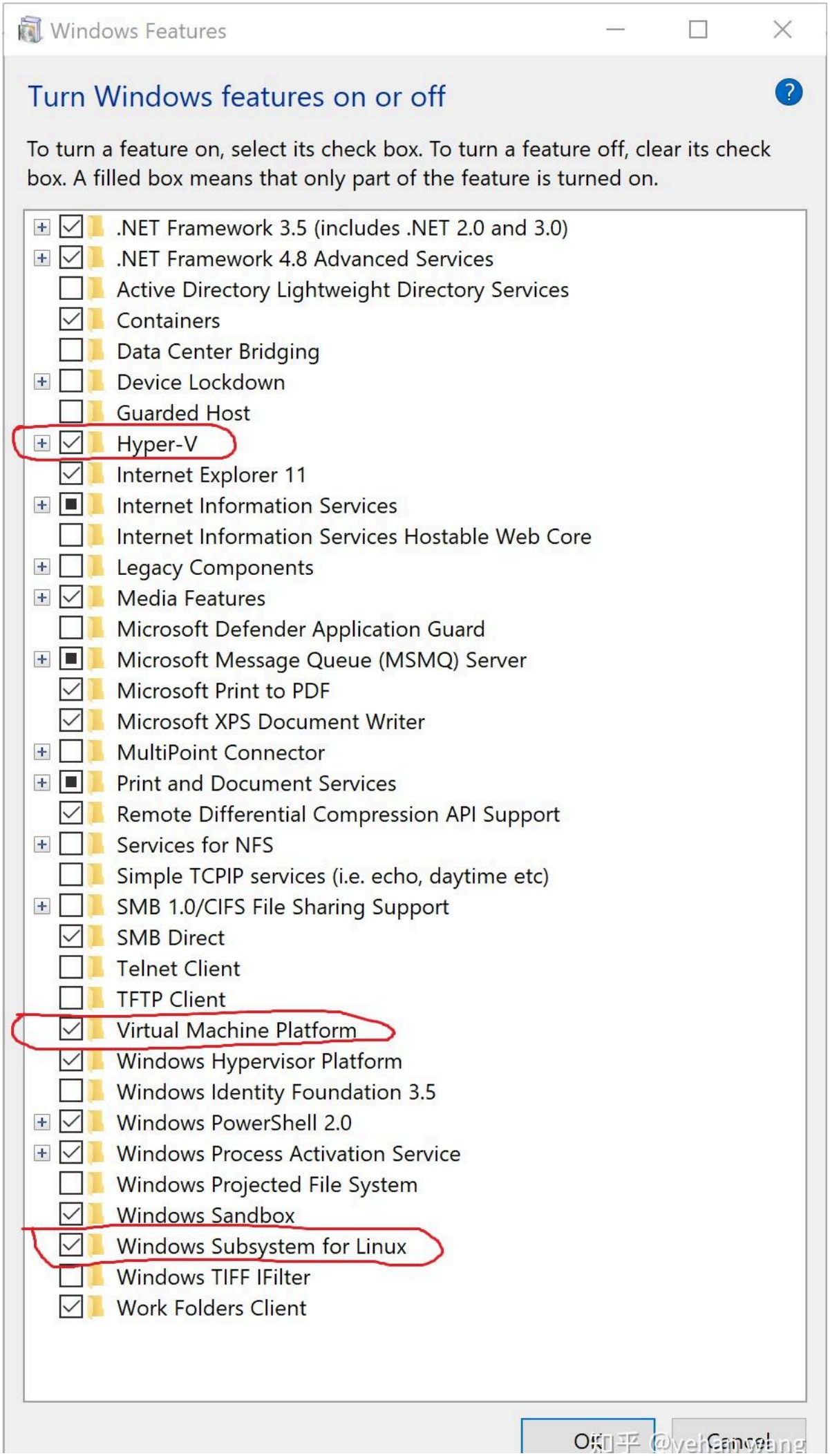
```
wsl.exe --list --online
```




```
wsl.exe --install Ubuntu-20.04
```

等待安装结束

设置-》应用-》程序和功能-》启用或关闭windows功能 中


Windows Features
Turn Windows features on or off
To turn a feature on, select its check box. To turn a feature off, clear its check box. A filled box means that only part of the feature is turned on.
.NET Framework 3.5 (includes .NET 2.0 and 3.0)
.NET Framework 4.8 Advanced Services
Active Directory Lightweight Directory Services
Containers
Data Center Bridging
Device Lockdown
Guarded Host
Hyper-V
Internet Explorer 11
Internet Information Services
Internet Information Services Hostable Web Core
Legacy Components
Media Features
Microsoft Defender Application Guard
Microsoft Message Queue (MSMQ) Server
Microsoft Print to PDF
Microsoft XPS Document Writer
MultiPoint Connector
Print and Document Services
Remote Differential Compression API Support
Services for NFS
Simple TCPIP services (i.e. echo, daytime etc)
SMB 1.0/CIFS File Sharing Support
SMB Direct
Telnet Client
TFTP Client
Virtual Machine Platform
Windows Hypervisor Platform
Windows Identity Foundation 3.5
Windows PowerShell 2.0
Windows Process Activation Service
Windows Projected File System
Windows Sandbox
Windows Subsystem for Linux
Windows TIFF IFilter
Work Folders Client
OK
Cancel

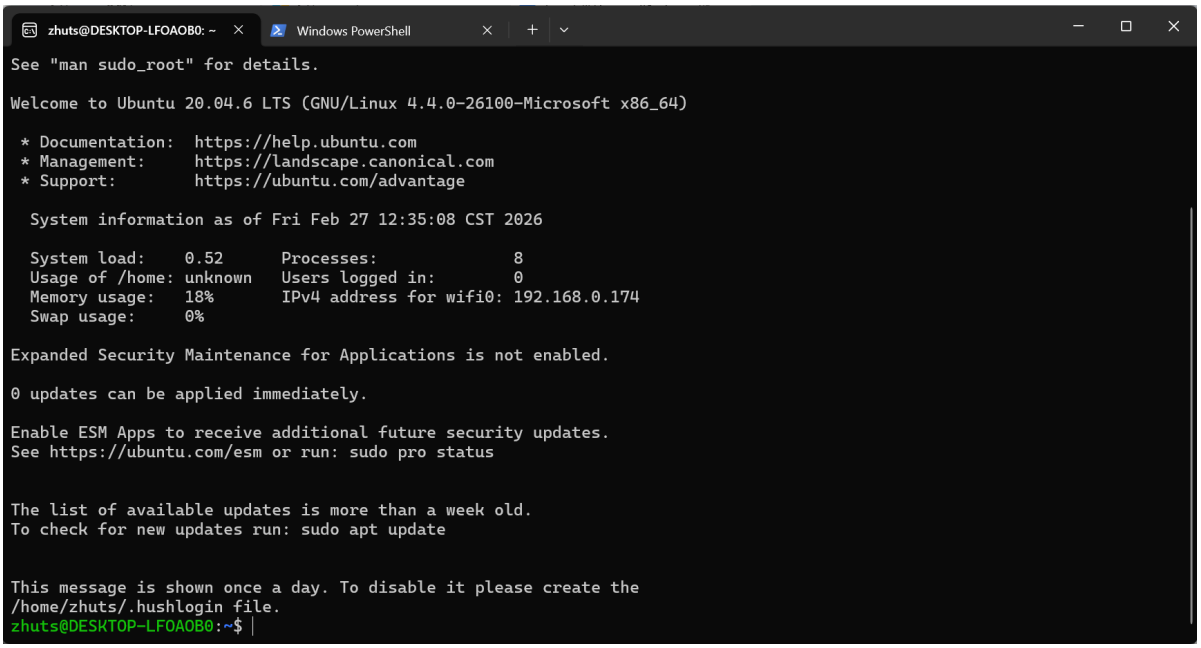
Hyper-V、VMP、Win Sub for Linux

问题解决方法：https://zhuanlan.zhihu.com/p/147233604

将WSL切换为WSL1

```
wsl --set-default-version 1
```

安装成功后是设置账户密码



重新改回

```
wsl --set-default-version 2
```

有推荐使用wsl2的，目前还没有解决其安装报错

3、配置工程环境

1、更新软件源

```
# 更新软件源索引
sudo apt update

# 安装可能缺失的基础依赖
sudo apt install -y software-properties-common apt-transport-https wget
```

2、下载arduino

```
sudo apt install arduino
```

查看版本

```
arduino --version
```

这里查看版本会报错，但是不影响后续操作

3、安装python3

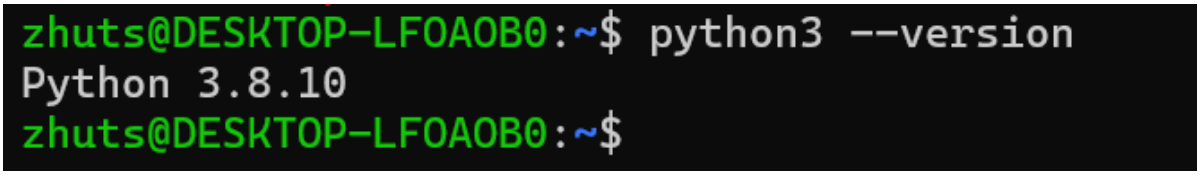
注意安装3.9版本的python，如果是3.9版本请跳过后续内容

```
sudo apt install python3 python3-pip
```

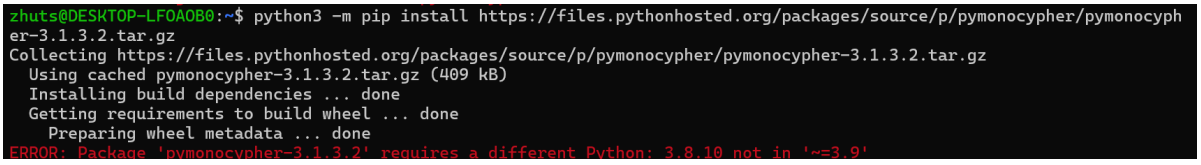
检查版本

```
python3 --version
pip3 --version
```

https://zhuanlan.zhihu.com/p/24469848200



如果版本安错了**需要更新python版本，不然安装不了加密包**




```
# 更新软件源
sudo apt update
# 安装Python 3.9全套依赖（含编译/运行库）
sudo apt install -y python3.9 python3.9-dev python3.9-distutils python3.9-venv
# 验证3.9安装成功
python3.9 --version  # 输出 Python 3.9.x 即可
```


```
# 备份默认python3链接
sudo mv /usr/bin/python3 /usr/bin/python3.bak
# 备份pip3链接（可选，后续同步更新）
sudo mv /usr/bin/pip3 /usr/bin/pip3.bak
# 将python3默认指向3.9
sudo ln -s /usr/bin/python3.9 /usr/bin/python3
# 将pip3默认指向3.9的pip（先安装3.9的pip）
curl https://bootstrap.pypa.io/get-pip.py | sudo python3.9
# 建立pip3软链接
sudo ln -s /usr/local/bin/pip3.9 /usr/bin/pip3
# 检查python3版本（应输出3.9.x）
python3 --version
# 检查pip3版本（应关联Python 3.9）
```

```
pip3 --version
```

```
zhuts@DESKTOP-LFOAOB0:~$ python3 --version
Python 3.9.5
zhuts@DESKTOP-LFOAOB0:~$ pip3 --version
pip 26.0.1 from /usr/local/lib/python3.9/dist-packages/pip (python 3.9)
zhuts@DESKTOP-LFOAOB0:~$ sudo pip3 install pymonocypher==3.1.3.2
Collecting pymonocypher==3.1.3.2
  Downloading pymonocypher-3.1.3.2-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (2.0 kB)
Collecting numpy>=1.23 (from pymonocypher==3.1.3.2)
  Downloading numpy-2.0.2-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (60 kB)
Downloading pymonocypher-3.1.3.2-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (370 kB)
Downloading numpy-2.0.2-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (19.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.0/19.5 MB 14.2 kB/s eta 0:21:40
```

必须得是3.9的python才行，如果后续使用更新的加密方式，这些库都得替换。

如果安装失败请按如下方式重装pymonocypher

```
python3 -m pip install pymonocypher==3.1.3.2
```

```
 python3
 import monocypher
```

```
>>> import monocypher
>>> monocypher.blake2b(b'hello world')
b'\x02\x1c\xed\x87\x99)l\xec\xa5W\x83*\xb9A\xa5\x0bJ\x11\xf84x\xcf\x14\x1fQ\xf93\xf6S\xab\x9f\xbc\xc0Z\x03|\xdd\xbe\xd0n
0\x9b\xf34\x94,NX\xcd\xf1\xa4n#y\x11\xcc\xd7\xfc\xf9x|\xbc\x7f\xd0'
>>>
```

仅设置父目录的 PYTHONPATH 找不到模块，因为 Python 不递归搜索子目录
最好导入环境变量，我测试了，安装在用户目录下不加环境变量无法使用

```
 echo "export PYTHONPATH=\$PYTHONPATH:/home/zhuts/.local/lib/python3.9/site-packages/pymonocypher" >> ~/.bashrc
source ~/.bashrc
```

输出是否能够成功导入

```
python3 -c "import monocypher; print('导入成功，版本：', monocypher.__version__)"
```

```
zhuts@DESKTOP-LFOAOB0:~/.local/lib/python3.9/site-packages/pymonocypher$ python3 -c "import monocypher; print('导入成功
，版本：', monocypher.__version__)"
导入成功，版本： 3.1.3.2
```

一般来说通过pip安装的都不需要手动添加环境变量

python3安装empty时会遇到settool工具版本低的问题

```
# 1. 升级 pip 到最新版（用户级安装，避开权限问题）
python3 -m pip install --user --upgrade pip

# 2. 升级 setuptools 到>=40.8.0（指定版本范围）
python3 -m pip install --user --upgrade setuptools>=40.8.0

# 验证版本（确保 setuptools 版本达标）
python3 -m pip show setuptools | grep Version
```

```
zhuts@DESKTOP-LFOAOB0:~/aruremoteid$ python3 -m pip show setuptools | grep Version
Version: 82.0.0
zhuts@DESKTOP-LFOAOB0:~/aruremoteid$ python3 -m pip install --user empy==3.3.4
Collecting empy==3.3.4
  Using cached empy-3.3.4.tar.gz (62 kB)
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Building wheels for collected packages: empy
  Building wheel for empy (pyproject.toml) ... done
  Created wheel for empy: filename=empy-3.3.4-py3-none-any.whl size=29379 sha256=e9325191d032497aa9c0563e9f1f4
a24ca791480598e11d8994f74
  Stored in directory: /home/zhuts/.cache/pip/wheels/c7/0a/c4/a0dc05ea753ca932849616e1c1886ad3c50aaa2c8f09c6a5
Successfully built empy
WARNING: Error parsing dependencies of distro-info: Invalid version: '0.23ubuntu1'
WARNING: Error parsing dependencies of python-debian: Invalid version: '0.1.36ubuntu1'
Installing collected packages: empy
Successfully installed empy-3.3.4
zhuts@DESKTOP-LFOAOB0:~/aruremoteid$
```

4、安装pymavlink（脚本也会自己安装）

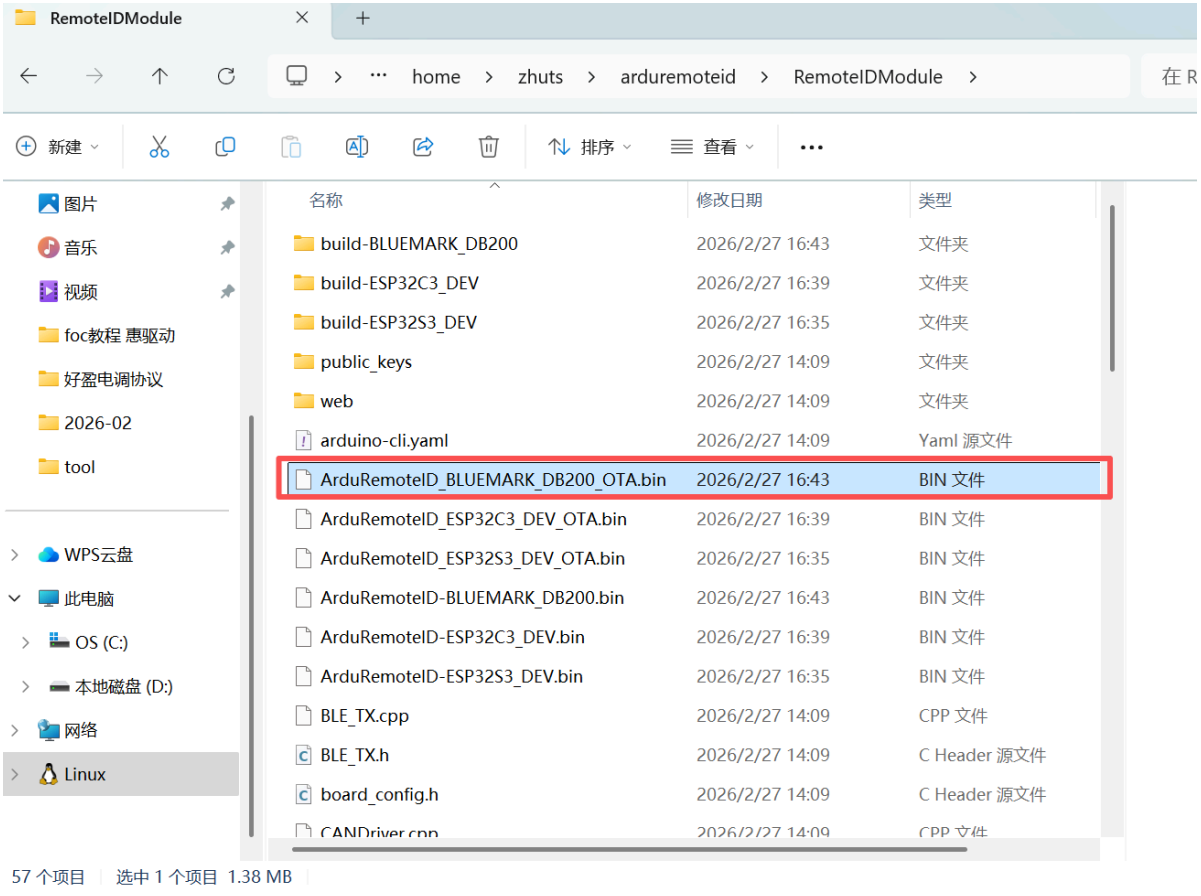
```
pip install pymavlink
```

```
zhuts@DESKTOP-LFOAOB0:~$ pip install pymavlink
Collecting pymavlink
  Using cached pymavlink-2.4.49-cp38-cp38-manylinux2014_x86_64.manylinux_2_17_x86_64.manylinux_2_28_x86_64.whl (6.4 MB)
Collecting fastcrc
  Downloading fastcrc-0.3.5-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (287 kB)
     |████████████████████████████████| 287 kB 17 kB/s
Collecting lxml
  Downloading lxml-6.0.2-cp38-cp38-manylinux2014_x86_64.manylinux_2_17_x86_64.whl (5.1 MB)
     |████████████████                | 3.6 MB 27 kB/s eta 0:00:53
Redirecting output to 'wget-log'.
     |████████████████████████████████| 5.1 MB 27 kB/s
Installing collected packages: fastcrc, lxml, pymavlink
Successfully installed fastcrc-0.3.5 lxml-6.0.2 pymavlink-2.4.49
zhuts@DESKTOP-LFOAOB0:~$
```

5、按照流程依次执行

```
- cd ~
- git clone https://github.com/ardupilot/aruremoteid
- cd aruremoteid/
- git submodule init
- git submodule update --recursive
- ./scripts/install_build_env.sh
- ./scripts/regen_headers.sh
- ./scripts/add_libraries.sh

- cd RemoteIDModule
- make setup

- cd RemoteIDModule
- make
// 需要其他配置暂时不弄，直接将要烧录的文件弄到win下，用FLASH工具烧录
- cd RemoteIDModule
- make upload
```

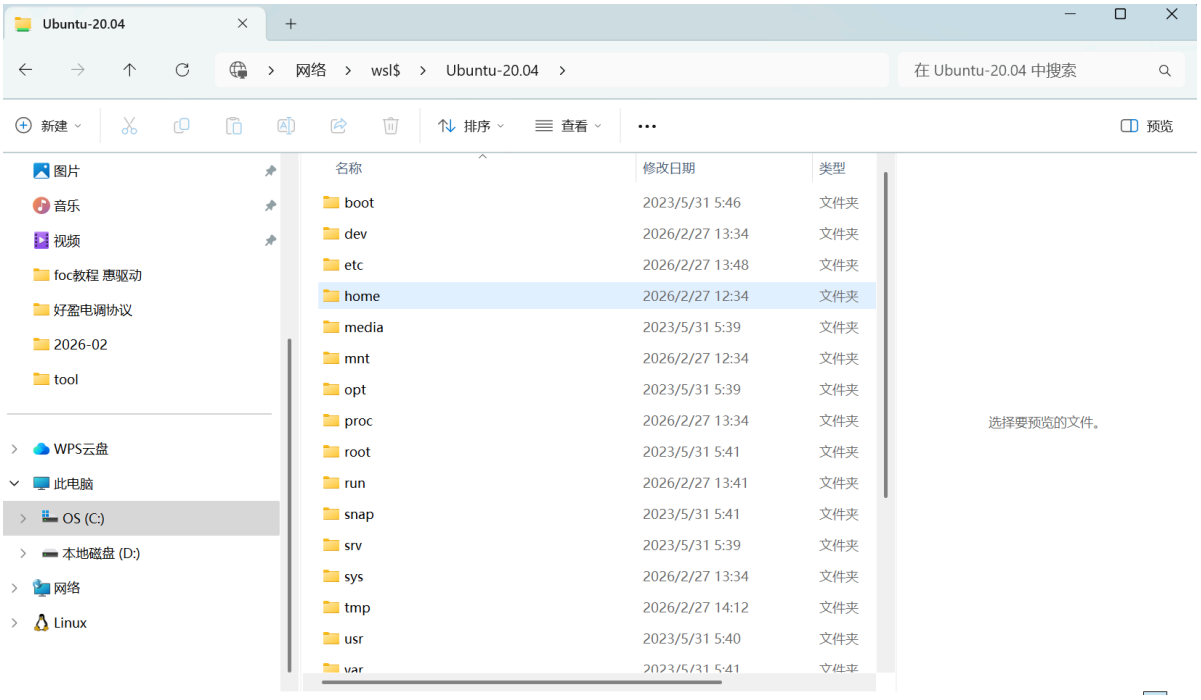
获得烧录后的bin文件，OTA是进行web升级的文件，DEV是flash工具烧录的全量文件，OTA默认是没有经过签名的。不签名的文件要想升级，要把Lock调节为-1，或者重新用flash工具进行烧写升级。



4、windows访问ubuntu文件夹

https://blog.csdn.net/djh3200/article/details/144693143

```
\\wsl$\Ubuntu-20.04\
```



5、在windows下的vscode中开发esp32

https://zhuanlan.zhihu.com/p/693938916

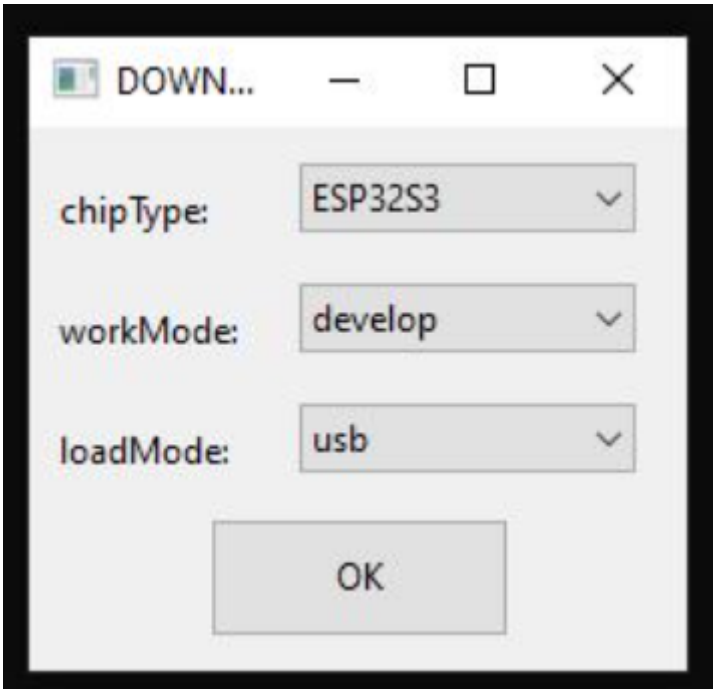
直接VSCODE里安装WSL插件，连接虚拟机即可。

6、RemoteID下载选择

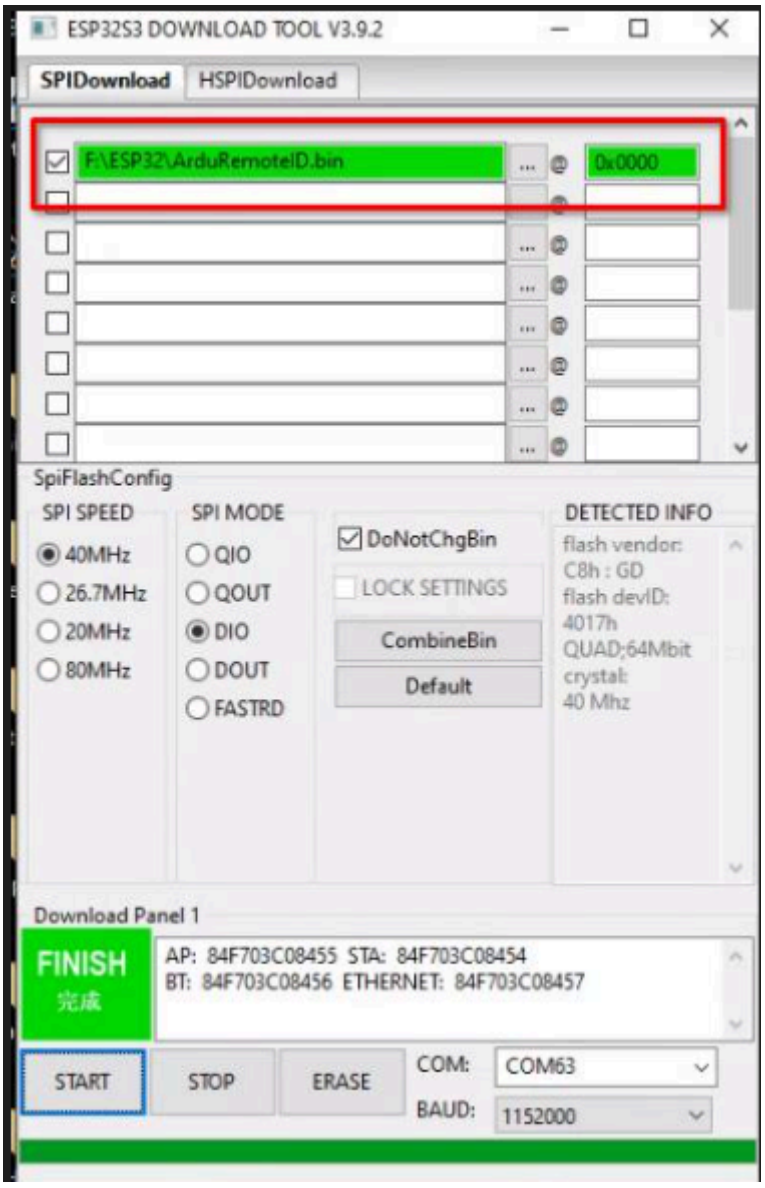
1、DroneCAN参数

Idx	Name	Type	Value	Default	Min	Max
0	LOCK_LEVEL	integer	0	0	0	2
1	CAN_NODE	integer	0	0	0	127
2	UAS_TYPE	integer	4	0	0	15
3	UAS_ID_TYPE	integer	1	0	0	4
4	UAS_ID	string	ABCD123456789			
5	BAUDRATE	integer	115200	57600	9600	921600
6	WIFI_NAN_RATE	real	0.0	0.0	0.0	5.0
7	WIFI_POWER	real	13.0	20.0	2.0	20.0
8	BT4_RATE	real	1.0	1.0	0.0	5.0
9	BT4_POWER	real	18.0	18.0	-27.0	18.0
10	BT5_RATE	real	1.0	1.0	0.0	5.0
11	BT5_POWER	real	18.0	18.0	-27.0	18.0
12	WEBSERVER_ENABLE	integer	1	1	0	1
13	WIFI_SSID	string	RID_123456			
14	WIFI_PASSWORD	string	********			
15	BCAST_POWERUP	integer	1	1	0	1
16	PUBLIC_KEY1	string	PUBLIC_KEYV1:WJbbpbjOz/yMB3JxnvqyTUInCQdZcStkA0qh…			
17	PUBLIC_KEY2	string	PUBLIC_KEYV1:X8jdVqxIIUmCuMSi8IhTZ40VkXW0gbRczzMt…			
18	PUBLIC_KEY3	string	PUBLIC_KEYV1:X8jdVqxIIUmCuMSi8IhTZ40VkXW0gbRczzMt…			
19	PUBLIC_KEY4	string				
20	PUBLIC_KEY5	string				

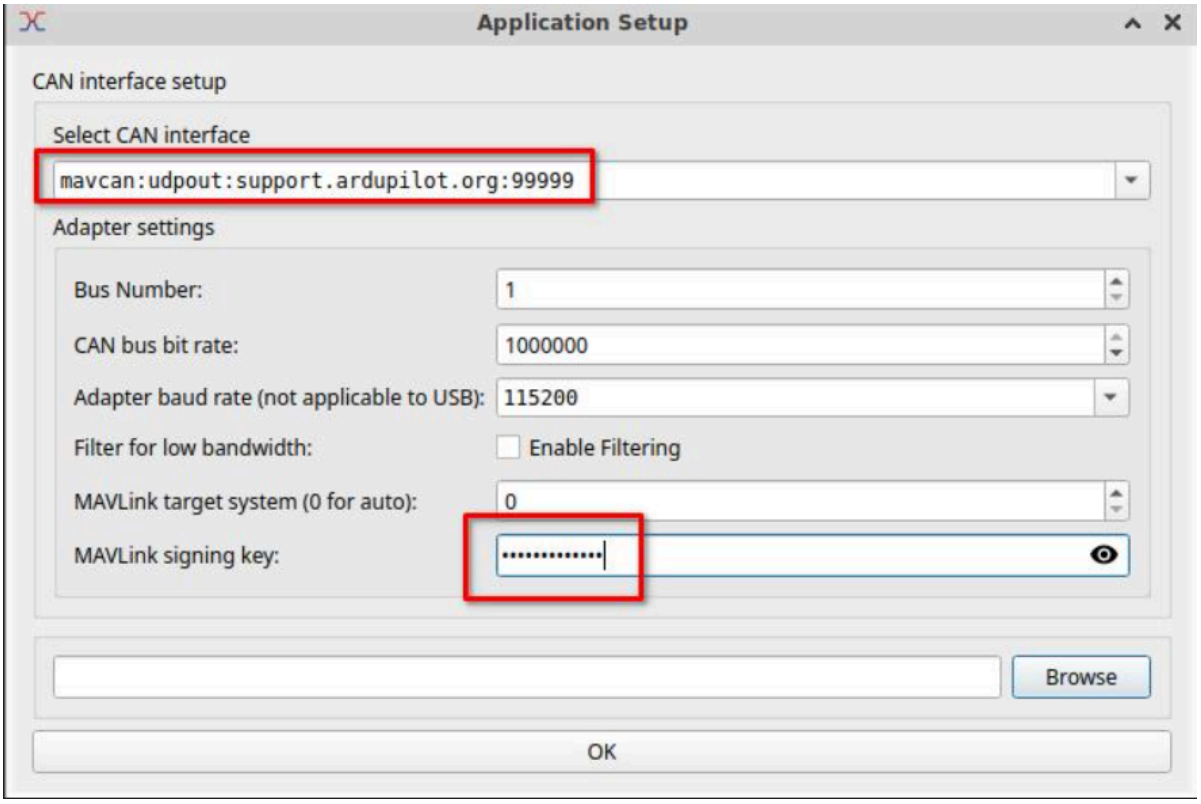
2、FLASH Tool选择



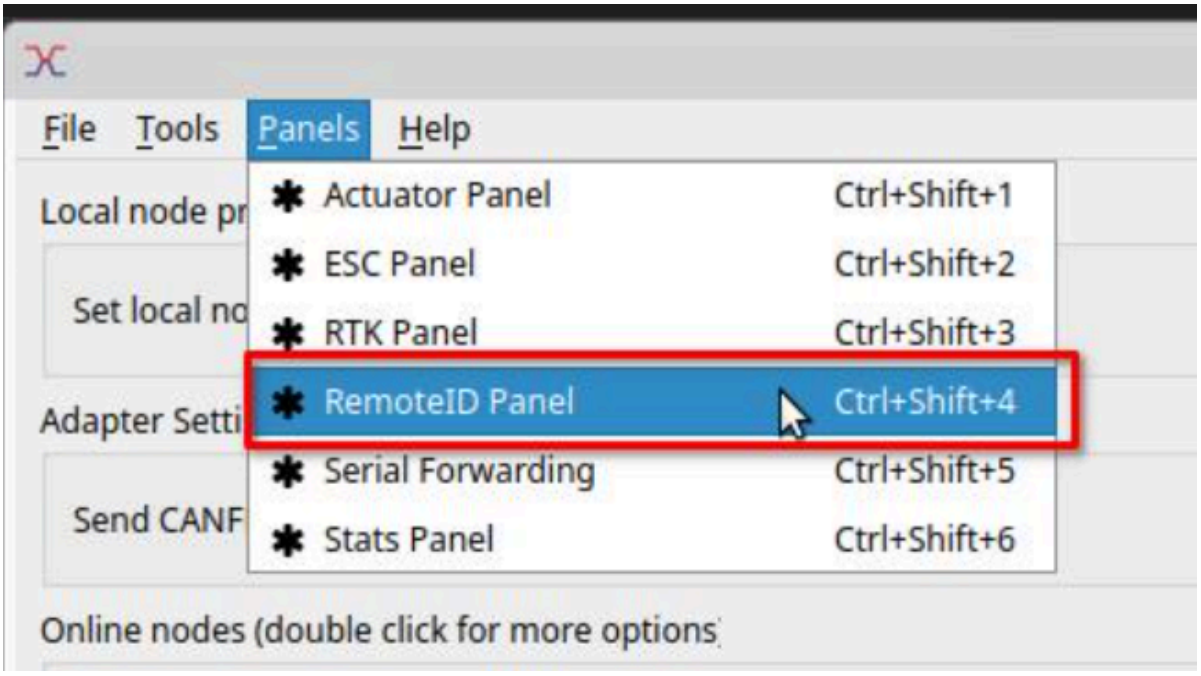
USB develop模式

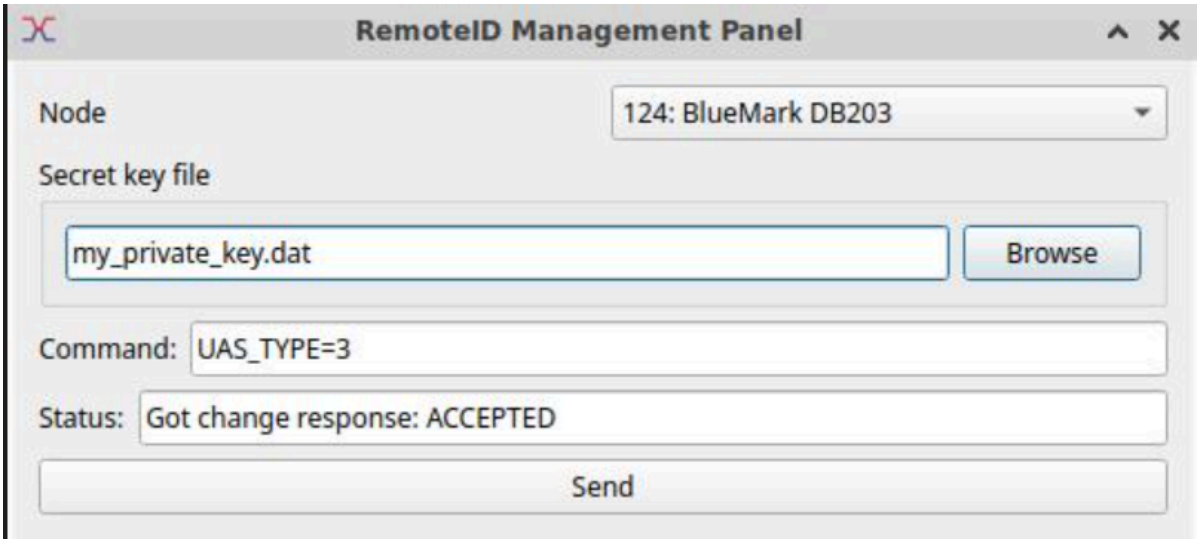

ESP32S3 DOWNLOAD TOOL V3.9.2
SPIDownload
HSPIDownload
F:\ESP32\ArduRemoteID.bin
0x0000
SpiFlashConfig
SPI SPEED
40MHz
26.7MHz
20MHz
80MHz
SPI MODE
QIO
QOUT
DIO
DOUT
FASTRD
DoNotChgBin
LOCK SETTINGS
CombineBin
Default
DETECTED INFO
flash vendor:
C8h : GD
flash devID:
4017h
QUAD;64Mbit
crystal:
40 Mhz
Download Panel 1
FINISH
完成
AP: 84F703C08455 STA: 84F703C08454
BT: 84F703C08456 ETHERNET: 84F703C08457
START
STOP
ERASE
COM:
COM63
BAUD:
1152000

3、CAN接口设置


Application Setup
CAN interface setup
Select CAN interface
mavcan:udpout:support.ardupilot.org:99999
Adapter settings
Bus Number:
1
CAN bus bit rate:
1000000
Adapter baud rate (not applicable to USB):
115200
Filter for low bandwidth:
Enable Filtering
MAVLink target system (0 for auto):
0
MAVLink signing key:
Browse
OK

4、RemoteID_Panel





1. **RemoteID Panel** 是drone can 上位机的图标

   这是一个专门的配置面板，用于管理无人机上的远程识别（Remote ID）模块，例如图中的 **BlueMark DB203**。Remote ID 是全球无人机法规强制要求的功能，它让无人机在飞行时广播自身的身份、位置等信息，以满足空域监管和安全要求。

2. **RemoteID Management Panel 界面**

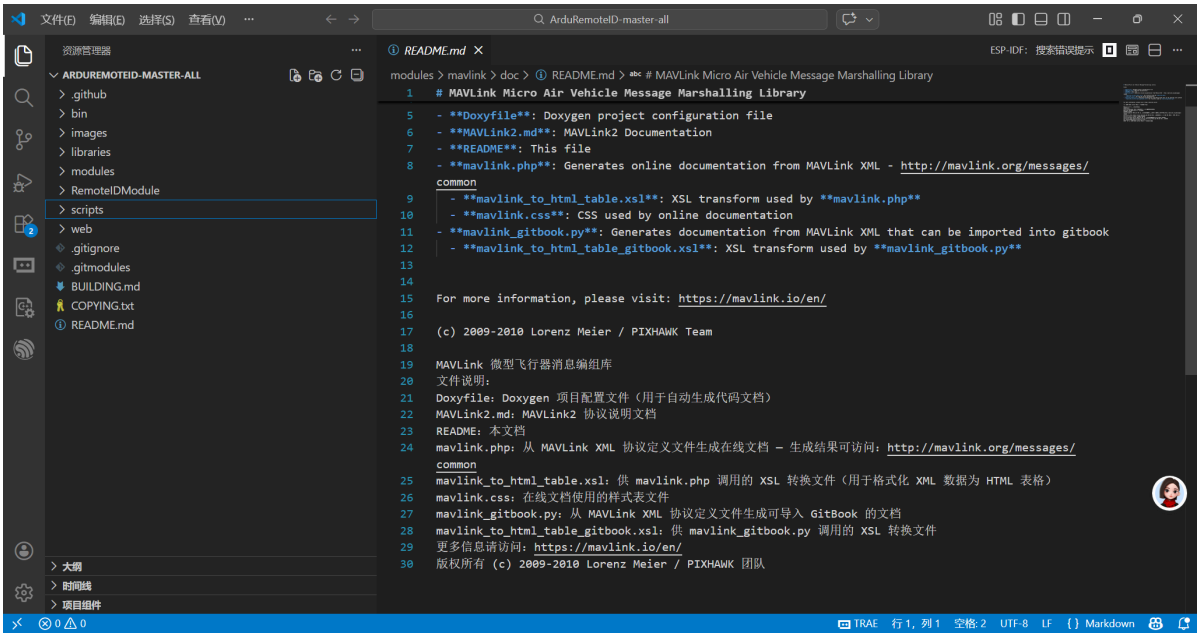
   - **Node**: 显示当前连接的 RemoteID 模块，这里是 `124: BlueMark DB203`。
   - **Secret key file**: 用于加载私钥文件（如 `my_private_key.dat`），对 RemoteID 广播信息进行签名，确保数据的真实性和不可伪造性。
   - **Command**: 发送配置指令，图中是 `UAS_TYPE=3`，这是设置无人机类型（UAS Type）的参数，`3` 通常代表某种特定类别的无人机（如小型或特定用途的无人机）。
   - **Status**: 显示指令执行结果，这里是 `Got change response: ACCEPTED`，表示模块已成功接收并接受了配置指令。

整体作用

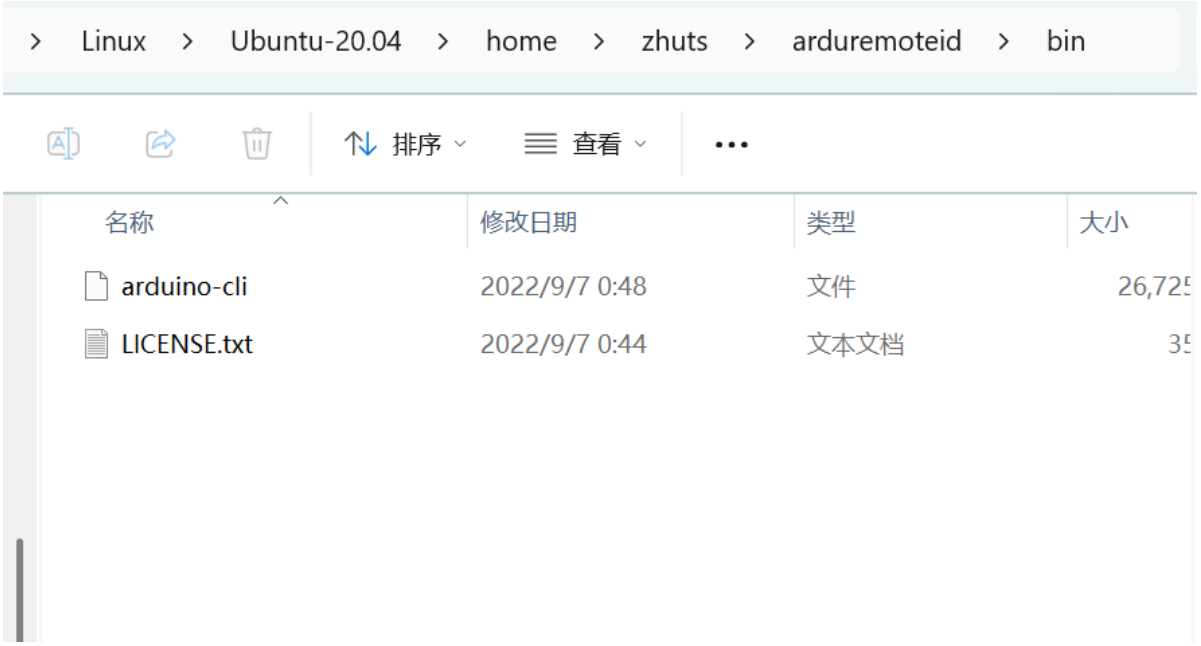
这个面板是无人机合规飞行的关键配置工具，通过它可以：

- 配置 RemoteID 模块的参数（如无人机类型、ID 等）
- 加载安全密钥，确保广播信息的真实性
- 验证配置是否成功，确保无人机满足 Remote ID 法规要求

7、工程文件夹里都是什么内容

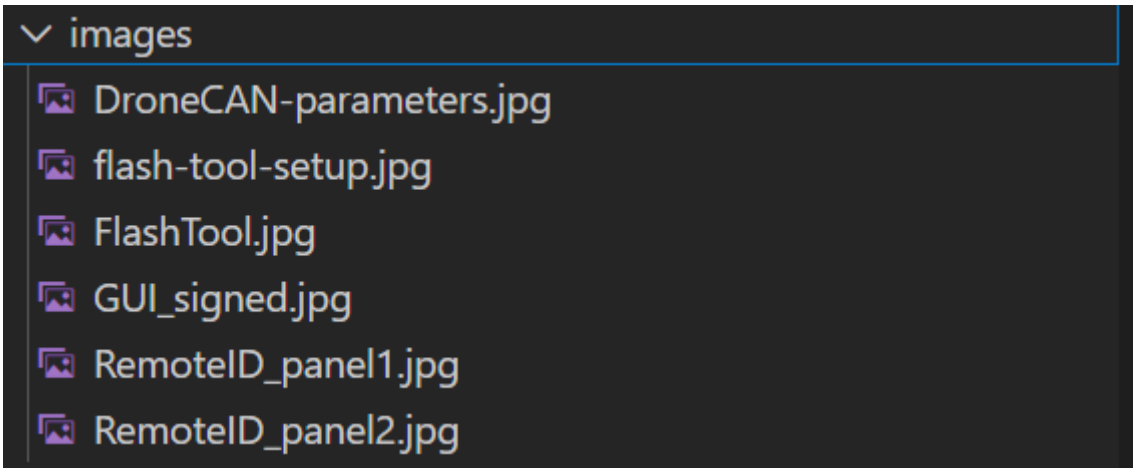


1 bin文件夹里装的是Arduino CLI的命令行工具



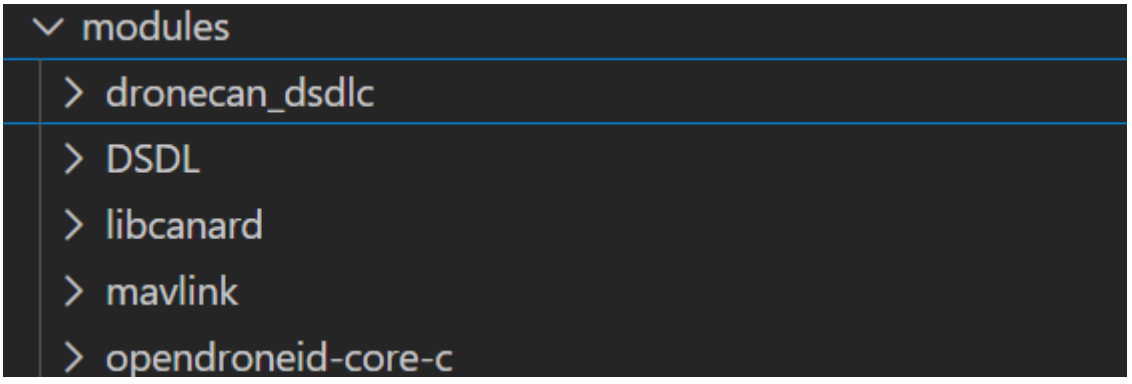

```
zhuts@DESKTOP-LFOAOB0:~/arduremoteid$ ./bin/arduino-cli version
arduino-cli  Version: 0.27.1 Commit: a900cfb2 Date: 2022-09-06T16:44:27Z


A new release of Arduino CLI is available: 0.27.1 → 1.4.1
https://arduino.github.io/arduino-cli/latest/installation/#latest-packages
zhuts@DESKTOP-LFOAOB0:~/arduremoteid$
```

image：装的是几个操作的图片，包括flash下载工具，CAN mavlink上位机等

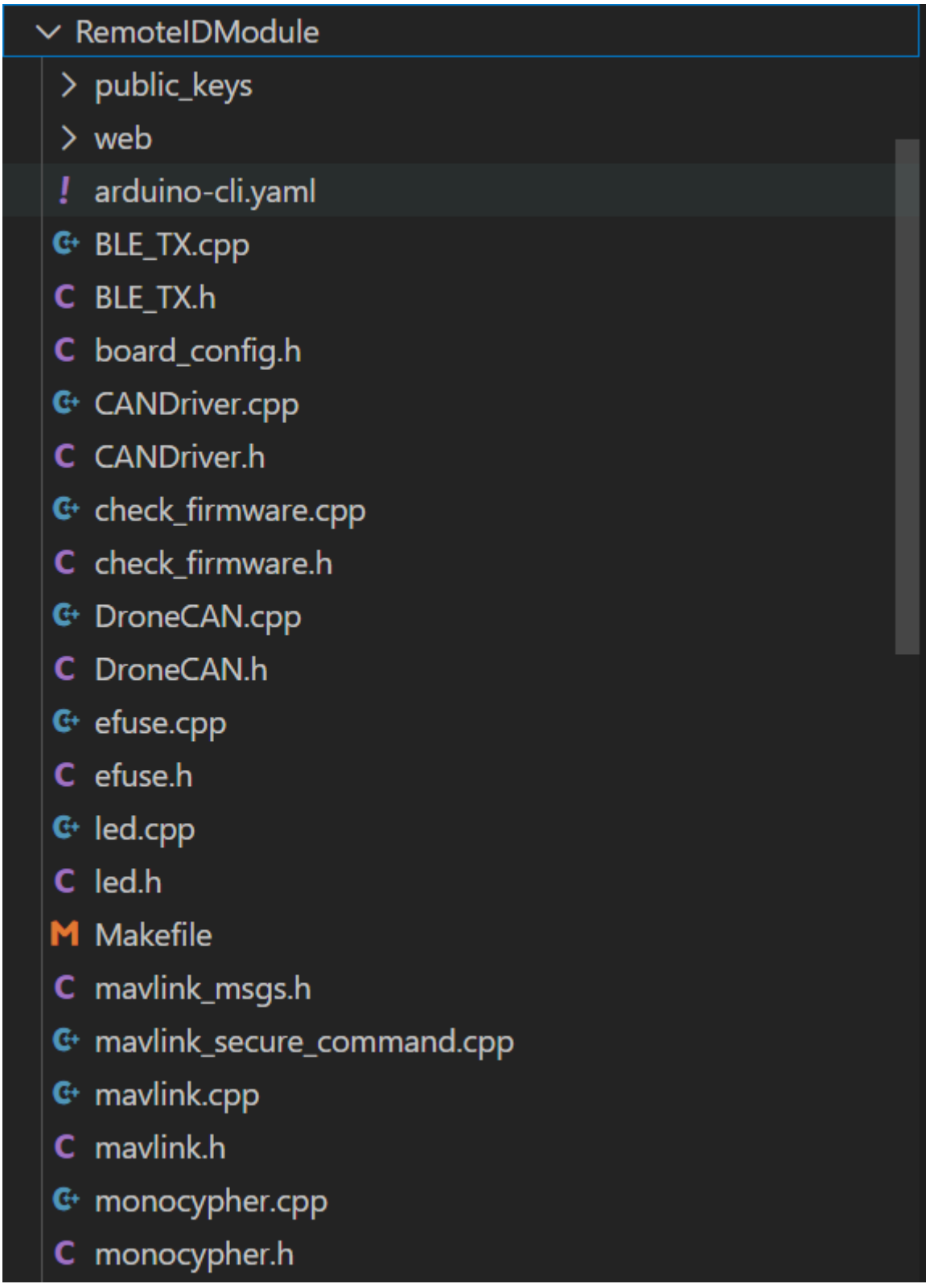


3、libraries：里面存放的是DroneCAN和mavlink2自动生成的c和h文件

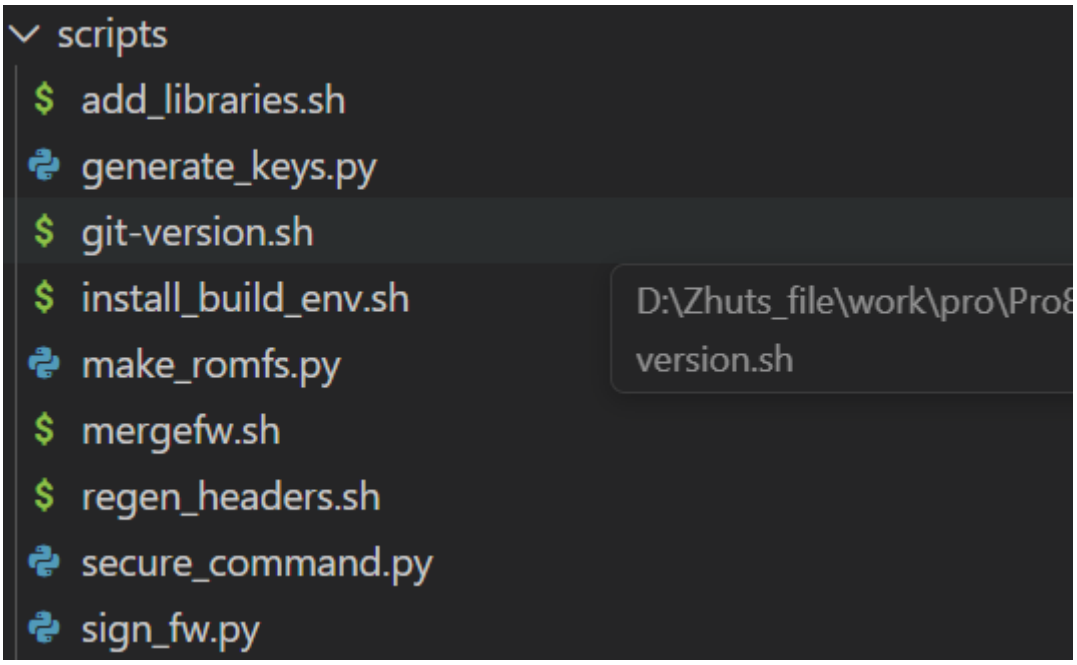
4、modules:git子模块从外部拉取的库文件



5、RemoteIDModule：工程模块，实现对应RomoteID作用的模块，作者自己编写的



6、scripts：脚本文件夹，放入了包括初始化等调用的sh文件及一些py脚本

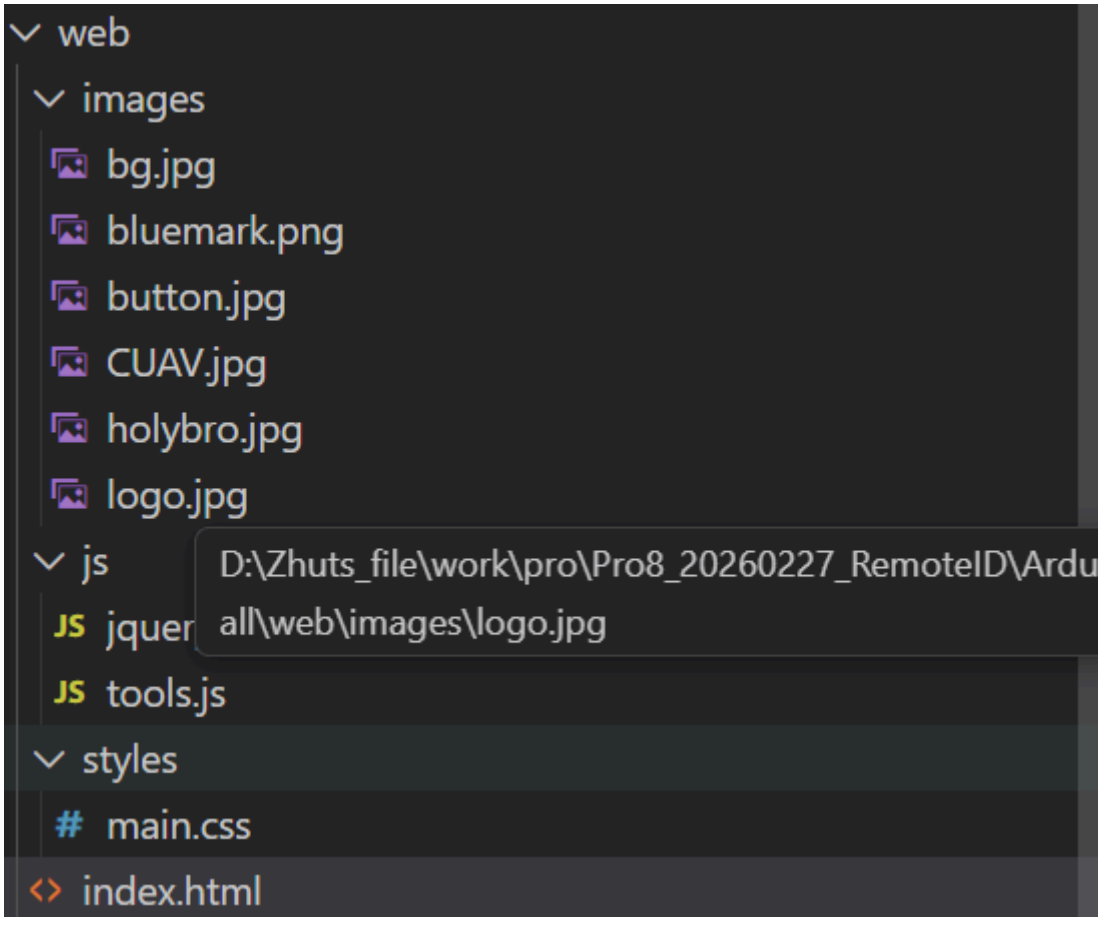


7、web：一些跟web服务器相关的内容

WEBSERVER_ENABLE（网页服务器使能）：启用后将创建 WiFi 热点和网页服务器，支持状态监控和安全固件升级。

PUBLIC_KEY1~PUBLIC_KEY5（公钥 1~ 公钥 5）：用于验证固件升级和参数安全更新的公钥。

网页服务器 若启用WEBSERVER_ENABLE参数，固件将内置网页服务器：



其中index.html就是对应的网页端口

项目里的前端源码（HTML/JS/CSS）经过编译 / 打包后，被嵌入到固件里，模块运行时再从固件里读取这些网页文件，提供给浏览器访问。

`index.html`（网页骨架）、`xxx.js`（交互逻辑，比如上传固件、改参数）、`xxx.css`（页面样式）；

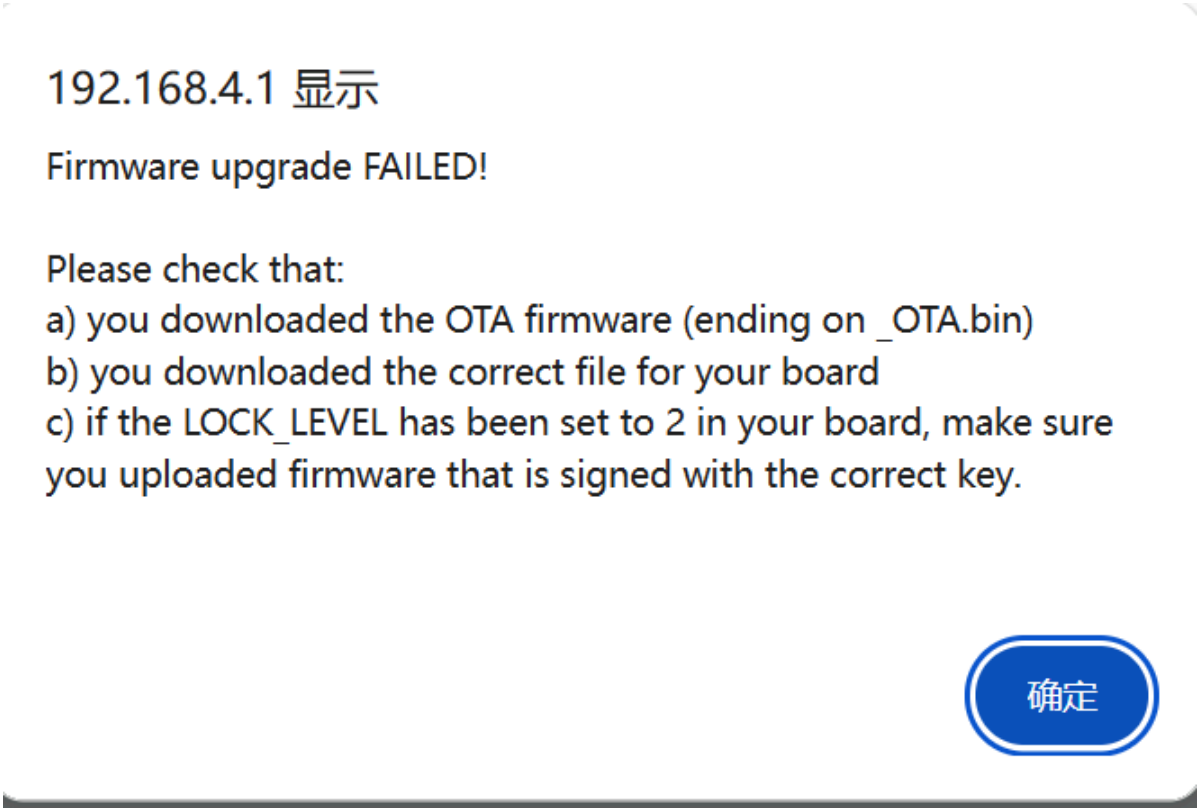
当模块上电、`WEBSERVER_ENABLE` 开启时：

1. 模块启动 WiFi 热点，同时启动内置的 Web 服务器程序；
2. 当你连热点、访问 `http://192.168.4.1` 时，Web 服务器程序会从 Flash 里读取之前嵌入的网页文件（HTML/JS/CSS）；
3. 把这些文件发送到你的浏览器，你就能看到网页界面了。

可以通过连接wifi后进入对应网页**192.168.4.1**查看设备状态以及进行在线升级。

试了一下，升级没成功，提示要使用签名后的固件，要仔细阅读关于lock取值的问题，取错了可能会造成芯片熔断，不可烧录，调试，擦除

核心原因是：**这个 WiFi 升级网页要求固件必须带数字签名（私钥签名），而你直接用编译后的无签名 OTA 文件，设备验签失败就会拒绝升级 整个程序只有网页升级和boot flash升级两种方式。**



8、.gitgnore

```
*generated/
git-version.h
*.bin
*.elf
*.log
romfs_files.h
```

你贴出的 `.gitignore` 内容是用来告诉 Git 哪些文件不需要纳入版本控制（简单说就是“不上传到代码仓库”），这和你 WiFi 升级失败的核心问题（固件未签名、公钥未预置）无关，但能帮你理清工程里的文件逻辑——先解读这些忽略规则，再告诉你关键的公钥预置位置该怎么找：

忽略项	含义	和你升级的关系
`generated/`	工程编译时自动生成的文件夹（比如自动生成的代码、配置）	大概率包含编译后的固件，但Git不上传，你需要自己编译生成
`git-version.h`	自动生成的版本号头文件	无直接关系
`*.bin`	所有二进制固件文件（包括OTA固件）	Git不上传，所以工程里没有现成的.bin/OTA文件，必须自己编译
`*.elf`	编译后的可执行文件（嵌入式固件的中间产物）	无直接关系，最终要的是.bin/OTA文件
`*.log`	日志文件	无直接关系

忽略项	含义	和你升级的关系
`romfs_files.h`	自动生成的文件系统头文件	若固件包含文件系统，可能和OTA升级路径有关，但不是核心

9、.gitmodules

```
[submodule "modules/opendroneid-core-c"]
    path = modules/opendroneid-core-c
    url = https://github.com/opendroneid/opendroneid-core-c.git
[submodule "modules/mavlink"]
    path = modules/mavlink
    url = https://github.com/ArduPilot/mavlink.git
[submodule "modules/DSDL"]
    path = modules/DSDL
    url = https://github.com/dronecan/DSDL.git
[submodule "modules/libcanard"]
    path = modules/libcanard
    url = https://github.com/dronecan/libcanard.git
[submodule "modules/dronecan_dsdlc"]
    path = modules/dronecan_dsdlc
    url = https://github.com/dronecan/dronecan_dsdlc.git
```

git子模块

10、BUILDING.md

```
基于 Linux 系统从源码编译
步骤 1：安装前置依赖
bash
运行
sudo apt install arduino   # 安装Arduino开发环境
pip install pymavlink      # 安装pymavlink库（用于MAVLink协议通信）
步骤 2：获取源码
bash 运行
cd ~                       # 进入用户主目录
git clone https://github.com/ardupilot/arduremoteid  # 克隆项目源码
cd arduremoteid/           # 进入项目目录
git submodule init         # 初始化子模块
git submodule update --recursive  # 递归更新所有子模块
./scripts/install_build_env.sh    # 执行脚本安装编译环境
./scripts/regen_headers.sh        # 执行脚本重新生成头文件
./scripts/add_libraries.sh        # 执行脚本添加所需库文件
使用 make 和 arduino-cli 编译
步骤 1：通过 make 安装 ESP32 开发支持
bash 运行
cd RemoteIDModule  # 进入RemoteIDModule子目录
make setup         # 执行make命令配置ESP32编译环境
步骤 2：通过 make 编译源码
bash 运行
cd RemoteIDModule  # 进入RemoteIDModule子目录
make               # 执行make命令编译项目
```

步骤 3：通过 make 烧录固件
bash 运行
```
cd RemoteIDModule  # 进入RemoteIDModule子目录
make upload        # 执行make命令将固件烧录到ESP32-S3开发板
```
若开发板无法正常烧录：按住 PCB 电路板上的 BOOT 按键，同时短暂按下 RESET 按键（强制进入烧录模式），然后重新执行烧录命令。
烧录完成后，ESP32-S3 会启动并通过蓝牙广播测试 / 演示用的远程识别（Remote ID）数据。
若出现 Python 串口支持缺失的错误：执行 python -m pip install pyserial 安装 pyserial 库。
可选配置 / 使用说明
UART 端口连接：
用另一根 USB 线将 ESP32-S3（注：原文笔误为 ep32-s3）连接到电脑（接入 PCB 上标注 “UART” 的端口）—— 该端口用于 MAVLink 通信和调试，你可以通过 mavproxy 等工具连接到此端口进行调试。
飞控 UART 连接：
将 ESP32-S3 通过 PCB 上的 RX (17)/TX (18)/GND 引脚接入飞控的 UART 端口 —— 它会监听来自飞控的 MAVLink 数据流，从中解析REMOTE_ID*相关数据包，并将这些无人机信息通过 2.4GHz 蓝牙 / Wi-Fi 广播出去；安卓手机可通过配套 App（https://play.google.com/store/apps/details?id=org.opendroneid.android_osm）接收，也可被其他符合 Open Drone ID（开放式无人机识别）标准的接收器识别。
飞控 CAN 端口连接：
将标准 CAN 收发器（如 VP231 等型号）连接到 PCB 上的 47 号引脚（TX）、38 号引脚（RX）和 GND 引脚，再将 ESP32-S3 接入飞控的 CAN 端口。
注：ArduPilot/MAVLink/CAN 之间的联动配置未在此文档中详述，可参考 ArduPilot 官方维基：https://ardupilot.org/copter/docs/common-remoteid.html

8、固件默认的烧录后，对应ESP32S3来说的串口配置是多少，且对应的波特率?

引脚定义

ESP32-S3 开发板默认引脚：

UART TX（发送）：18 号引脚

UART RX（接收）：17 号引脚

CAN TX（发送）：47 号引脚

CAN RX（接收）：38 号引脚

波特率默认是57600，看param那个文件就可以看到默认参数设置

9、如何修改web网页的字体?

你想把Web页面改成中文，**不需要给ESP32加中文字库**——核心原因是：这个Web页面是运行在你的浏览器里（电脑/手机），不是运行在ESP32上，ESP32只负责把网页文件（HTML/JS/CSS）传给浏览器，文字渲染、字体显示都是浏览器的事。

我用大白话拆解“改中文”的完整逻辑，一步就能搞定：

一、为什么不用给ESP32加中文字库?

- ESP32的角色：只是个“文件服务器”，只负责把网页的HTML/JS/CSS文件发给你的浏览器，它自己不显示任何文字；
- 文字渲染的角色：你的电脑/手机浏览器会处理所有文字显示，浏览器本身自带中文字库（比如宋体、微软雅黑），根本不用ESP32操心；
- 对比理解：就像你用浏览器看英文网页，改成中文后，只需要改网页里的文字内容，不用给服务器装中文字库是一个道理。

二、把Web页面改成中文的具体步骤（超简单）

只需要修改Web页面的源码文件（HTML/JS）里的英文文本，替换成中文就行，步骤如下：

步骤1：找到Web页面的源码文件

先找到项目里生成`web`目录的前端源码（不是生成后的`web`目录，是源码！），执行命令定位：

```
cd /home/zhuts/aruremoteid
# 搜索包含网页文字的文件（比如“Firmware Update”是固件更新的英文）
grep -r "Firmware Update" --include="*.html" --include="*.js" .
```

步骤2：修改文本内容

比如找到`web-src/index.html`里的这段英文：

```
<h1>Firmware Update</h1>
<p>Upload a signed firmware file to update the module</p>
<label>Password:</label>
```

直接改成中文：

```
<h1>固件更新</h1>
<p>上传已签名的固件文件来更新模块</p>
<label>密码：</label>
```

步骤3：重新生成web目录并编译固件

修改完源码后，执行项目的构建脚本（比如`scripts/build-webui.sh`），重新生成`web`目录，再编译固件：

```
# 重新生成web目录（打包中文页面）
./scripts/build-webui.sh
# 重新编译固件（把新的web文件嵌入ESP32）
./bin/arduino-cli compile --fqbn esp32:esp32:esp32s3 RemoteIDModule
```

步骤4：烧录固件，验证效果

把新固件烧录到ESP32后，连WiFi热点访问`http://192.168.4.1`，就能看到中文页面了。

三、补充：如果是ESP32串口打印的英文要改中文

如果除了Web页面，你还想把ESP32串口输出的日志（比如`Serial.print("Firmware start")`）改成中文，这时候才需要给ESP32加中文字库——因为串口打印的文字是ESP32直接输出的，需要它能识别中文字符编码（比如UTF-8），不过这和Web页面是两回事。

总结

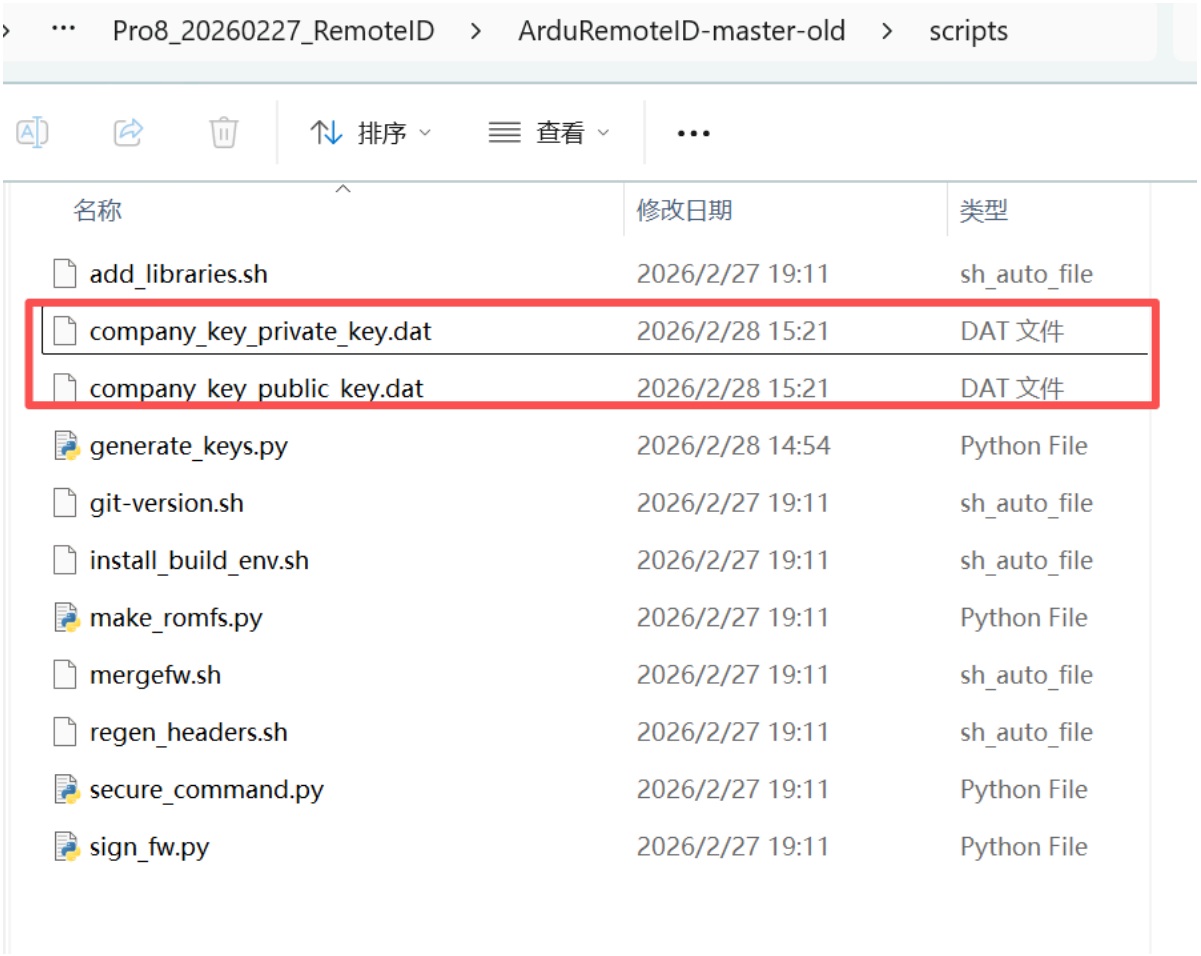
1. Web页面改中文：**不用给ESP32加任何东西**，只改网页源码里的英文文本，重新编译固件即可；
2. 核心逻辑：Web页面在浏览器里渲染，浏览器自带中文字库，ESP32只传文件不处理文字；
3. 关键步骤：找网页源码→改文本→重新生成web目录→编译烧录固件。

简单说，改Web页面的语言和改普通网页一样，只改文字内容就行，和ESP32硬件、字库完全无关。如果找不到网页源码文件，我可以帮你定位具体要改哪个文件。

10、在win下生成密钥

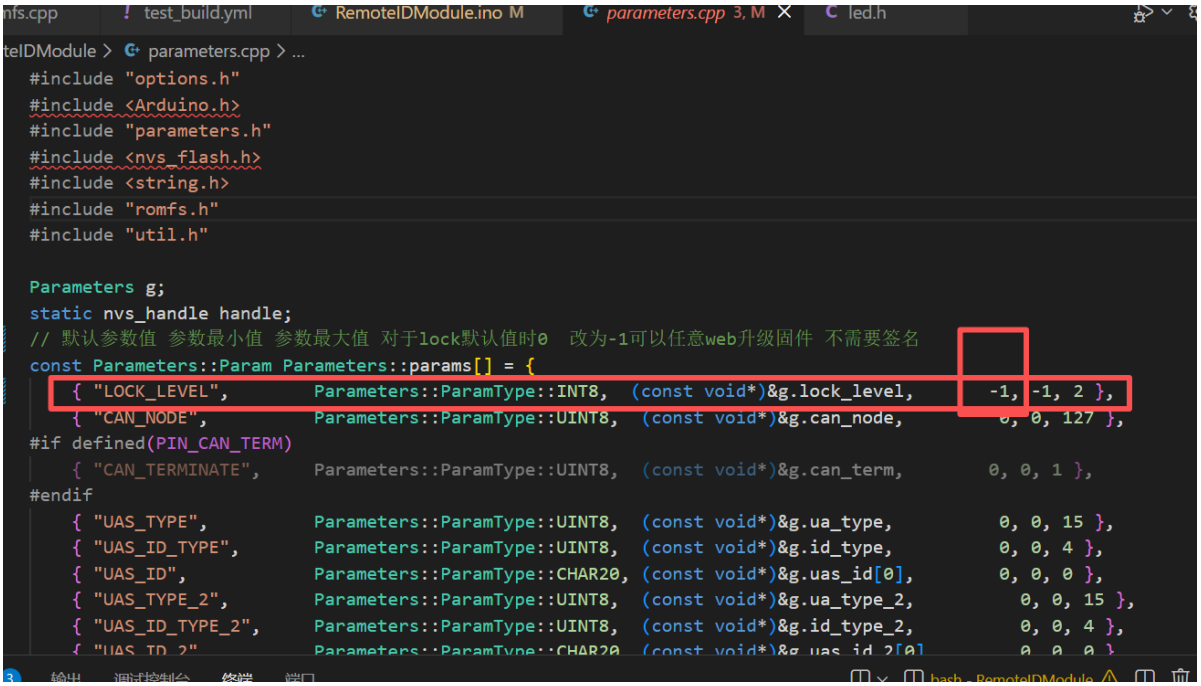
首先安装3.9版本的python

```
python -m pip install --upgrade pip
python -m pip install pyserial pexpect
python -m pip install empy==3.3.4 -i https://pypi.tuna.tsinghua.edu.cn/simple
python -m pip install pymavlink -i https://pypi.tuna.tsinghua.edu.cn/simple
python -m pip install dronecan -i https://pypi.tuna.tsinghua.edu.cn/simple
python -m pip install pymonocypher==3.1.3.2 -i
https://pypi.tuna.tsinghua.edu.cn/simple
cd D:\Zhuts_file\work\pro\Pro8_20260227_RemoteID\ArduRemoteID-master-old\scripts
python generate_keys.py company_key
```



11、如何测试Web升级功能?

在固件里面默认OTA升级时要使用带签名的固件，所以在OTA请求时会检查固件的签名。设置了一个调试专用的lock等级，lock=-1的时候什么固件都可以下载进模块，不会进行任何拦截。lock默认值是0，手动修改成-1即可。



将setup()函数内延时5秒钟可以看到一些驱动是否成功启动。

```
void setup()
{
    delay(5000); // 调试时使用 当lock为-1时 任意固件均可直接升级
    // disable brownout checking 禁用欠压检测
    WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0);
```

```
ArduRemoteID version 1.14 0fc42673
CAN/TWAI Driver installed
CAN/TWAI Alerts reconfigured
CAN/TWAI Driver started
DIS_DOWNLOAD_MODE = 0
DIS_USB_JTAG = 0
Checking partition app0


[2026-03-02 16:43:32.566]# RECV ASCII/25 <<<
app_descriptor not found


[2026-03-02 16:43:32.598]# RECV ASCII/74 <<<
Waiting for heartbeat
WAP start RID_80b54ee465e8 ArduRemoteID
WAP started
```

调试串口会打印mavlink向外发送的信息，例如心跳、状态两个报文。

附件

1、下载乐鑫的USB驱动

https://blog.csdn.net/qq_63922192/article/details/132832528

https://dl.espressif.com/dl/idf-driver/idf-driver-esp32-usb-jtag-2021-07-15.zip

2、编译出来的很多个固件是什么意思

该固件当前支持 ESP32-S3 和 ESP32-C3 芯片。目前支持 7 种主板，还有更多主板即将加入：

- ESP32-S3 开发板：https://au.mouser.com/ProductDetail/356-ESP32S3DEVKTM1N8
- ESP32-C3 开发板：https://au.mouser.com/ProductDetail/Espressif-Systems/ESP32-C3-DevKitM-1
- Bluemark DB110（旧款）来自 https://bluemark.io/ （产品页面）
- Bluemark DB200 来自 https://bluemark.io/ （产品页面 | 由 db201 替代）
- Bluemark DB201 来自 https://bluemark.io/ （产品页面 | 购买）
- Bluemark DB202mav 来自 https://bluemark.io/ （产品页面 | 购买）
- Bluemark DB210pro 来自 https://bluemark.io/ （产品页面 | 购买）
- Holybro Remote ID 模块 (产品页面)

预计很快将添加来自 https://wurzbachelectronics.com/ 的硬件。

对于 ESP32-S3 开发板，此固件假定的引脚为：

- UART TX 在引脚 18
- UART RX 在引脚 17
- CAN TX 在引脚 47
- CAN RX 在引脚 38

对于 ESP32-C3 开发板，此固件假定的引脚为：

- UART TX 在引脚 3
- UART RX 在引脚 2
- CAN TX 在引脚 5
- CAN RX 在引脚 4

对于 CAN，需要将一个合适的 1MBit bxCAN 转发器连接到 CAN TX/RX 引脚。

您还可以在 USB UART 端口（丝网上标记为 "UART" 的端口）上使用 MAVLink。这允许通过插入一个微型 USB 线缆进行简单的仿真测试。

3、ESP32S3是一款SOC

有内置和不内置flash，内置512kb的静态SRAM。模组配置外置最大8M的FLASH，和8M的PSRAM

4、为什么开始的时候工程里模块的文件夹代码都是空的。

```
- git submodule init
- git submodule update --recursive
```

因为这些库的代码都是别的仓库的东西，本工程在初始配置的时候直接从别人的仓库拉取最新的代码。

`git submodule init` 的核心作用（一句话总结）

这个命令的本质是：**读取主仓库的 `.gitmodules` 配置文件，将其中定义的子模块信息“注册”到主仓库的 `.git/config` 配置中，并创建子模块在主仓库 `.git/modules/` 下的空目录结构**（为后续拉取子模块代码做准备）。

`git submodule init`注册子模块配置、创建空 Git 目录❌ 否`.git/config`新增子模块配置

`git submodule update`拉取子模块代码、生成 .git 文件✅ 是`modules/DSDL`出现代码 +`.git`文件

5、工程里的这几个脚本都是什么意思

1、install_build_env.sh

```
#!/bin/bash

python3 -m pip install empy==3.3.4
python3 -m pip install pymavlink
python3 -m pip install dronecan
python3 -m pip install pyserial
python3 -m pip install pexpect
python3 -m pip install pymonocypher==3.1.3.2

wget https://downloads.arduino.cc/arduino-cli/arduino-
cli_0.27.1_Linux_64bit.tar.gz
mkdir -p bin
(cd bin && tar xvzf ../arduino-cli_0.27.1_Linux_64bit.tar.gz)

#这份脚本是你的项目开发环境初始化脚本，一键安装所有 Python 依赖 + 下载 Arduino 命令行编译工
具；
#核心目的：让你的 Linux 系统快速具备开发无人机 / Arduino 项目的能力，支持自动化编译、调试
MAVLink/DroneCAN/CAN 总线相关代码；
```

2、git_version.sh

```
#include "git-version.h"
// 打印固件版本（比如串口输出）
# 执行该脚本后，会在当前目录生成一个名为 git-version.h 的 C 语言头文件，文件内容是：#define
GIT_VERSION 0x<当前代码的Git提交哈希>
```

3、add_liraries.sh

```
这是一份Arduino 第三方库软链接配置脚本：把你项目中 modules/ 和 libraries/ 目录下的
Libcanard、MAVLink2、DroneCAN 等核心库，一键创建 “快捷方式” 到 Arduino 官方库目录，让
Arduino IDE/CLI 能直接识别并使用这些库，无需手动复制代码。
```

就是arduino编译的时候直接去工程路径下去找这些代码即可，不需要再下载。

4、mergefw.sh

调用arduino安装下的esp烧录工具，把生成的bin文件合并成一个

```
# 调用 ESP32 官方烧录工具 esptool.py（Arduino 安装目录下的版本）
python3 $HOME/.arduino15/packages/esp32/tools/esptool_py/3.3.0/esptool.py \
  # 1. 芯片与复位配置（告诉工具要操作的硬件和复位规则）
```

```
  --chip esp32s3 \                    # 指定目标芯片：ESP32-S3（必须和硬件匹配，错了会烧录失败）
  --before default_reset \            # 烧录前：执行默认复位（让芯片进入烧录模式）
  --after hard_reset \                # 烧录后：执行硬复位（重启芯片，直接运行新固件）

  # 2. 核心操作与输出配置（告诉工具要做什么、生成什么文件）
  merge_bin \                         # 核心指令：合并多个分块固件为一个完整固件
  -o ArduRemoteID.bin \               # 输出文件：合并后的固件命名为 ArduRemoteID.bin

  # 3. 闪存参数配置（必须和你的 ESP32-S3 模组一致，否则启动失败）
  --flash_mode dio \                  # 闪存模式：DIO（双线读写，ESP32-S3 主流模式，兼容大部分模组）
  --flash_freq 80m \                  # 闪存频率：80MHz（ESP32-S3 推荐频率，部分低成本模组需改 40m）
  --flash_size 4MB \                  # 闪存容量：4MB（需和模组实际容量匹配，8MB/16MB 需对应修改）

  # 4. 分块固件与烧录地址（ESP32-S3 固定布局，不能乱改）
  0xe000 boot_app0.bin \              # 地址 0xe000：烧录分区表备份文件（防止分区表损坏）
  0x0 RemoteIDModule.ino.bootloader.bin \  # 地址 0x0：烧录引导程序（芯片上电第一个运行的程序）
  0x10000 RemoteIDModule.ino.bin \          # 地址 0x10000：烧录你的主程序（RemoteID/CAN/MAVLink 核心逻辑）
  0x8000 RemoteIDModule.ino.partitions.bin # 地址 0x8000：烧录分区表（定义闪存各区域用途）
```

5、regen_headers.sh

这是一份**代码自动生成脚本**：一键生成 MAVLink2 协议头文件、DroneCAN 协议代码，并适配 Arduino 库的特殊处理规则，最后为固件生成 Git 版本号头文件 —— 核心是把 “协议描述文件” 转换成 “可直接编译的 C 代码”，无需手动编写通信协议相关代码。

```
#!/bin/bash
# 注释：重新生成 mavlink 头文件（前提是已安装 pymavlink）

# 1. 生成 MAVLink2 协议 C 语言头文件
echo "Generating mavlink2 headers"  # 打印日志，提示正在生成 MAVLink2 头文件
rm -rf libraries/mavlink2/generated  # 先删除旧的生成文件（避免缓存导致的版本问题）
# 核心命令：调用 mavgen.py（pymavlink 自带工具）生成代码
# --wire-protocol 2.0：指定 MAVLink 2.0 协议
# --lang C：生成 C 语言代码
# modules/mavlink/.../all.xml：MAVLink 协议描述文件（定义了所有消息格式）
# -o：输出目录（生成的代码放到 libraries/mavlink2/generated）
mavgen.py --wire-protocol 2.0 --lang C
modules/mavlink/message_definitions/v1.0/all.xml -o libraries/mavlink2/generated

# 2. 生成 DroneCAN 协议代码（适配 Libcanard 库）
echo "Generating DroneCAN headers for libcanard"  # 打印日志，提示正在生成 DroneCAN 代码
rm -rf libraries/DroneCAN_generated  # 删除旧的 DroneCAN 生成文件
# 核心命令：调用 dronecan_dsdlc.py（DroneCAN 代码生成工具）
# -O：输出目录（生成的代码放到 libraries/DroneCAN_generated）
# modules/DSDL/...：DroneCAN 协议描述文件（DSDL 格式，定义 CAN 总线消息）
python3 modules/dronecan_dsdlc/dronecan_dsdlc.py -O libraries/DroneCAN_generated
modules/DSDL/uavcan modules/DSDL/dronecan modules/DSDL/com
```

```
# 3. 适配 Arduino 库的“糟糕处理逻辑”（关键：让 Arduino 能识别 DroneCAN 代码）
# 定义需要暴露的核心协议消息（NodeStatus=节点状态、GetNodeInfo=获取节点信息等）
PACKETS="NodeStatus GetNodeInfo HardwareVersion SoftwareVersion RestartNode
dynamic_node_id remoteid param Log"
# 遍历每个消息类型，创建软链接
for p in $PACKETS; do
    (
        cd libraries/DroneCAN_generated  # 进入生成目录（子shell执行，不影响外层目录）
        # 为头文件创建软链接到当前目录（Arduino 只会扫描库根目录，不会递归找 include/ 下的文
件）
        ln -s include/*"$p"*.h .
        # 为源文件创建软链接到当前目录（同理，让 Arduino 找到 .c 文件）
        ln -s src/*"$p"*.c .
    )
done

# 4. 生成 Git 版本号头文件（为固件添加版本标识）
# 进入 RemoteIDModule 目录，执行 git-version.sh 脚本（生成 git-version.h）
(cd RemoteIDModule && ../scripts/git-version.sh)
```

6、generate_keys.py

创建公司钥匙的脚本，由公司自己把持。要保留好生成的公私钥匙，丢了就可能导致一批产品无法验证。

✅ 私钥：唯一签发权（证明“是我的 + 没被改”）；

✅ 公钥：唯一验证权（核对“是不是我的 + 有没有被改”）；

使用示例

```
python3 generate_keys.py company_test_key
```

```
#!/usr/bin/env python3  # 声明脚本用Python3解释器运行，跨平台兼容
'''
generate a public/private key pair using Monocypher
'''  # 文档字符串：说明脚本核心功能——用Monocypher生成公私钥对

import sys  # 导入系统模块：处理命令行参数、程序退出
import base64  # 导入Base64编码模块：将二进制密钥转为可读字符串

try:
    import monocypher  # 尝试导入Monocypher加密库
except ImportError:
    Logs.error("Please install monocypher with: python3 -m pip install
pymonocypher==3.1.3.2")
    sys.exit(1)  # 导入失败则提示安装，并退出程序（退出码1表示错误）

if monocypher.__version__ != "3.1.3.2":
    Logs.error("must use monocypher 3.1.3.2, please run: python3 -m pip install
pymonocypher==3.1.3.2")
    sys.exit(1)  # 版本不符则提示升级，退出程序
```

```
if len(sys.argv) != 2:
    print("Usage: generate_keys.py BASENAME")
    sys.exit(1)  # 检查命令行参数数量：必须传入1个基础文件名，否则提示用法并退出

bname = sys.argv[1]  # 获取命令行传入的基础文件名（比如传入"mykey"，则生成
mykey_private/public_key.dat）

def encode_key(ktype, key):
    # 定义密钥编码函数：将二进制密钥转为可读字符串
    # ktype：密钥类型（PRIVATE/PUBLIC），key：二进制密钥数据
    return ktype + "_KEYV1:" + base64.b64encode(key).decode('utf-8')
    # 编码规则：类型前缀 + 版本标识 + Base64编码后的密钥（转成UTF-8字符串）

# 核心：生成密钥对
private_key = monocypher.generate_key()  # 生成Ed25519私钥（32字节二进制数据）
public_key = monocypher.compute_signing_public_key(private_key)  # 从私钥推导对应的
公钥

# 定义输出文件名
public_fname = "%s_public_key.dat" % bname  # 公钥文件名：基础名_public_key.dat
private_fname = "%s_private_key.dat" % bname  # 私钥文件名：基础名_private_key.dat

# 写入私钥文件
open(private_fname, "w").write(encode_key("PRIVATE", private_key))
print("Generated %s" % private_fname)  # 打印提示：私钥文件已生成

# 写入公钥文件
open(public_fname, "w").write(encode_key("PUBLIC", public_key))
print("Generated %s" % public_fname)  # 打印提示：公钥文件已生成
```

7、secure_commond.py

```
#!/usr/bin/env python3
'''
perform a secure parameter change on a ArduRemoteID node via DroneCAN
user must supply a private key corresponding to one of the public keys on the
node
'''

import dronecan, time, sys, random, base64, struct
from dronecan import uavcan

try:
    import monocypher
except ImportError:
    print("Please install monocypher with: python3 -m pip install
pymonocypher==3.1.3.2")
    sys.exit(1)


# get command line arguments
from argparse import ArgumentParser
parser = ArgumentParser(description='secure_command')
```

```
parser.add_argument("--bitrate", default=1000000, type=int, help="CAN bit rate")
parser.add_argument("--node-id", default=100, type=int, help="local CAN node ID")
parser.add_argument("--target-node", default=None, type=int, help="target node
ID")
parser.add_argument("--private-key", default=None, type=str, help="private key
file")
parser.add_argument("--bus-num", default=1, type=int, help="MAVCAN bus number")
parser.add_argument("--signing-passphrase", help="MAVLink2 signing passphrase",
default=None)
parser.add_argument("--timeout", help="DroneCAN message timeout", type=float,
default=3)
parser.add_argument("uri", default=None, type=str, help="CAN URI")
parser.add_argument("paramop", default=None, type=str, help="parameter
operation")
args = parser.parse_args()

should_exit = False

if args.target_node is None:
    print("Must specify target node ID")
    should_exit = True

if args.private_key is None:
    print("Must specify private key file")
    should_exit = True

if should_exit:
    sys.exit(1)

SECURE_COMMAND_GET_REMOTEID_SESSION_KEY =
dronecan.dronecan.remoteid.SecureCommand.Request().SECURE_COMMAND_GET_REMOTEID_SE
SSION_KEY
SECURE_COMMAND_SET_REMOTEID_CONFIG =
dronecan.dronecan.remoteid.SecureCommand.Request().SECURE_COMMAND_SET_REMOTEID_CO
NFIG
session_key = None
sequence = random.randint(0, 0xFFFFFFFF)
last_session_key_req = 0
last_set_config = 0

# Initializing a DroneCAN node instance.
node = dronecan.make_node(args.uri, node_id=args.node_id, bitrate=args.bitrate)
node.can_driver.set_bus(args.bus_num)

if args.signing_passphrase is not None:
    node.can_driver.set_signing_passphrase(args.signing_passphrase)

# Initializing a node monitor
node_monitor = dronecan.app.node_monitor.NodeMonitor(node)

def get_session_key_response(reply):
    if not reply:
        print("Session key timed out")
        return
    global session_key
    session_key = bytearray(reply.response.data)
```

```
    print("Got session key")

def get_private_key():
    '''get private key, return 32 byte key or None'''
    if args.private_key is None:
        return None
    try:
        d = open(args.private_key,'r').read()
    except Exception as ex:
        return None
    ktype = "PRIVATE_KEYV1:"
    if not d.startswith(ktype):
        return None
    return base64.b64decode(d[len(ktype):])

def make_signature(seq, command, data):
    '''make a signature'''
    private_key = get_private_key()
    d = struct.pack("<II", seq, command)
    d += data
    if command != SECURE_COMMAND_GET_REMOTEID_SESSION_KEY:
        if session_key is None:
            print("No session key")
            raise Exception("No session key")
        d += session_key
    return monocypher.signature_sign(private_key, d)

def request_session_key():
    '''request a session key'''
    global sequence
    sig = make_signature(sequence, SECURE_COMMAND_GET_REMOTEID_SESSION_KEY,
bytes())
    node.request(dronecan.dronecan.remoteid.SecureCommand.Request(
            sequence=sequence,
            operation=SECURE_COMMAND_GET_REMOTEID_SESSION_KEY,
            sig_length=len(sig),
            data=sig),
        args.target_node,
        get_session_key_response,
        timeout=args.timeout)
    sequence = (sequence+1) % (1<<32)
    print("Requested session key")

def config_change_response(reply):
    if not reply:
        print("Config change timed out")
        return
    result_map = {
        0: "ACCEPTED",
        1: "TEMPORARILY_REJECTED",
        2: "DENIED",
        3: "UNSUPPORTED",
        4: "FAILED" }
    result = result_map.get(reply.response.result, "invalid")
    print("Got change response: %s" % result)
    sys.exit(reply.response.result)
```

```
def send_config_change():
    '''send remoteid config change'''
    global sequence
    req = args.paramop.encode('utf-8')
    sig = make_signature(sequence, SECURE_COMMAND_SET_REMOTEID_CONFIG, req)
    node.request(dronecan.dronecan.remoteid.SecureCommand.Request(
            sequence=sequence,
            operation=SECURE_COMMAND_SET_REMOTEID_CONFIG,
            sig_length=len(sig),
            data=req+sig),
        args.target_node,
        config_change_response,
        timeout=args.timeout)
    sequence = (sequence+1) % (1<<32)
    print("Requested config change")

def update():
    now = time.time()
    global last_session_key_req, last_set_config, session_key
    if session_key is None and now - last_session_key_req > args.timeout+1:
        last_session_key_req = now
        request_session_key()
    if session_key is not None and now - last_set_config > args.timeout+1:
        last_set_config = now
        send_config_change()

while True:
    try:
        update()
        node.spin(timeout=0.1)
    except Exception as ex:
        print(ex)
```

你这段代码是一个**基于DroneCAN协议的安全配置工具**，核心作用是：通过私钥签名的方式，向指定的ArduRemoteID节点发送加密的配置修改指令（比如修改RemoteID参数），全程保证指令的合法性和安全性——只有持有正确私钥的人，才能修改节点配置。

下面用「人话+核心逻辑+使用方法」拆解，让你快速懂怎么用、怎么跑：

一、核心功能（大白话总结）

这个脚本干两件关键事，一步都不能少：

1. **获取会话密钥**：先向目标节点请求一个临时的「会话密钥」（session key），相当于和节点建立一个加密会话；
2. **发送加密配置指令**：用自己的私钥给「配置指令+会话密钥+序列号」签名，把指令发给节点——节点会用对应的公钥验签，确认是合法操作后才执行配置修改。

简单说：这是给无人机RemoteID节点改配置的“安全遥控器”，只有拿对私钥（公司保留的那个），才能改节点参数，防止别人恶意篡改。

二、关键参数/逻辑拆解（重点看怎么填）

核心参数	作用	举例
`--target-node`	要修改的目标节点ID（必须填）	`--target-node 50`
`--private-key`	你的私钥文件路径（必须填）	`--private-key company_private_key.dat`
`uri`	CAN总线的连接地址（必须填）	Windows下通常是 `can0` / `socketcan:can0`，或串口 `serial:COM3:115200`
`paramop`	要执行的配置操作（必须填）	比如 `SET_UID=123456`（具体指令看节点支持的参数）
`--bitrate`	CAN总线波特率（默认1000000，不用改）	`--bitrate 1000000`
`--node-id`	本地CAN节点ID（默认100，不用改）	`--node-id 100`

三、Windows下使用步骤（照做就能跑）

前提：先搞定环境

1. 按之前说的，把Python降到3.9，装好依赖：

```
python -m pip install pymonocypher==3.1.3.2 dronecan
```

2. 确保你的电脑连好了CAN总线（比如通过CAN转USB模块连到无人机节点），并知道：
   - 目标节点ID（比如50）；
   - CAN总线的URI（比如Windows串口是 `serial:COM3:115200`）；
   - 私钥文件路径（比如 `C:\keys\company_private_key.dat`）；
   - 要执行的配置指令（比如 `SET_UID=123456`，具体问节点开发人员）。

执行命令（核心操作）

打开CMD，进入脚本所在目录，执行格式：

```
python 脚本名.py --target-node 目标节点ID --private-key 私钥文件路径 CAN总线URI 配置指令
```

举个完整例子（替换成你的实际参数）

```
python secure_command.py --target-node 50 --private-key
C:\keys\company_private_key.dat serial:COM3:115200 SET_UID=1234567890
```

四、执行后会发生什么（流程解读）

1. 脚本先检查参数：必须填目标节点ID、私钥文件，否则直接退出；
2. 连接CAN总线，向目标节点发送「获取会话密钥」的请求（用私钥签名，节点验签通过后返回session key）；
3. 拿到session key后，把你要改的配置指令（比如SET_UID）、session key、序列号打包，用私钥签名；
4. 把签名后的配置指令发给节点，节点验签通过后执行修改，并返回结果（ACCEPTED/拒绝/失败等）；
5. 脚本打印结果后退出，比如：`Got change response: ACCEPTED` 就是修改成功。

五、常见问题&解决

1. **提示"Session key timed out"**：
   - 原因：CAN总线没连好，或目标节点ID填错，或节点没上电；
   - 解决：检查硬件连接、节点ID是否正确，确保节点正常运行。
2. **提示"No session key"**：
   - 原因：没拿到session key就发配置指令；
   - 解决：先确保能拿到session key（检查私钥是否正确，节点是否认可你的私钥）。
3. **返回"DENIED"**：
   - 原因：私钥不对，或节点不允许你修改这个参数；
   - 解决：核对私钥文件是否是节点认可的公钥对应的私钥，确认配置指令是否合法。
4. **Windows下找不到CAN总线**：
   - 解决：安装CAN驱动（比如CAN转USB的驱动），确认URI格式正确（串口是 `serial:COMx:波特率`，CAN总线是 `socketcan:can0`）。

总结

1. **核心用法**：`python 脚本名.py --target-node 节点ID --private-key 私钥文件 CAN地址 配置指令`；
2. **核心逻辑**：私钥签名→拿会话密钥→签名配置指令→节点验签执行；
3. **成功标志**：终端输出 `Got change response: ACCEPTED`。

这个脚本是无人机RemoteID节点的"安全配置工具"，核心就是靠你之前生成的私钥做身份认证，确保只有授权方才能修改节点参数。如果执行时遇到具体报错，把报错信息贴出来，我帮你定位问题。

实际使用场景（一步一步看）
物理连接：你把电脑通过 CAN 转 USB 模块（或串口）连到无人机的 RemoteID 设备上（设备要上电、正常运行）；
你操作电脑：在电脑上打开终端 / CMD，执行这个脚本（带上私钥、目标节点 ID、配置指令等参数）；
脚本主动通信：脚本通过 CAN 总线给 RemoteID 设备发消息：
第一步："我要获取会话密钥，这是我的私钥签名（证明我是授权方）"；
第二步：设备验证签名通过后，返回会话密钥；
第三步：脚本用会话密钥 + 私钥给配置指令签名，发给设备："请执行 XX 配置修改"；
设备被动响应：设备验签通过→执行配置→给脚本返回 "成功 / 失败"；
脚本打印结果：你在电脑上看到 "ACCEPTED"，就知道设备配置改好了。

8、sign_fw.py

```
#!/usr/bin/env python3
'''
sign an OTA bin
'''

import sys
import struct
import base64

try:
    import monocypher
except ImportError:
    print("Please install monocypher with: python3 -m pip install
pymonocypher==3.1.3.2")
    sys.exit(1)

key_len = 32
sig_len = 64
descriptor = b'\x43\x2a\xf1\x37\x46\xe2\x75\x19'

if len(sys.argv) < 4:
    print("Usage: sign_fw OTA_FILE PRIVATE_KEYFILE BOARD_ID")
    sys.exit(1)

ota_file = sys.argv[1]
key_file = sys.argv[2]
board_id = int(sys.argv[3])

img = open(ota_file,'rb').read()
img_len = len(img)

def decode_key(ktype, key):
    ktype += "_KEYV1:"
    if not key.startswith(ktype):
        print("Invalid key type")
        sys.exit(1)
    return base64.b64decode(key[len(ktype):])

key = decode_key("PRIVATE", open(key_file, 'r').read())
if len(key) != key_len:
    print("Bad key length %u" % len(key))
    sys.exit(1)

desc_len = 80
ad_start = len(img)-desc_len
if img[ad_start:ad_start+8] == descriptor:
    print("Image is already signed")
    sys.exit(1)

signature = monocypher.signature_sign(key, img)
if len(signature) != sig_len:
    print("Bad signature length %u should be %u" % (len(signature), sig_len))
    sys.exit(1)
```

```
desc = struct.pack("<II64s", board_id, img_len, signature)
img = img + descriptor + desc

if len(img) != img_len + desc_len:
    print("Error: incorrect image length")
    sys.exit(1)

print("Applying signature")

open(ota_file, "wb").write(img)
print("Wrote %s" % ota_file)
```

你这段代码是一个**固件签名工具**，核心作用是：用你之前生成的私钥，给无人机/嵌入式设备的OTA升级固件（bin文件）添加数字签名，确保只有授权的固件能被设备识别和升级——简单说就是给固件“盖官方公章”，防止篡改和伪造。

下面用「人话+核心逻辑+使用方法」拆解，让你快速懂怎么用：

一、核心功能（大白话总结）

1. **校验固件**：检查要签名的OTA固件（bin文件）是否已经签过名，避免重复签名；
2. **加载私钥**：读取你之前生成的私钥文件，验证私钥格式和长度是否合法；
3. **生成签名**：用私钥对固件内容生成Ed25519签名（64字节）；
4. **添加签名到固件**：把“设备ID+固件长度+签名+固定标识”打包，追加到固件文件末尾；
5. **保存固件**：覆盖原固件文件，完成签名。

最终效果：签名后的固件包含你的官方签名，设备升级时会用对应的公钥验签，确认是正版固件才会执行升级。

二、关键参数（必须填对）

参数位置	作用	举例
第1个参数	要签名的OTA固件文件路径	`firmware.ota.bin`
第2个参数	私钥文件路径（之前生成的.dat文件）	`company_key_private_key.dat`
第3个参数	目标设备的board_id（设备唯一标识）	`123`（具体值问设备开发人员）

三、Windows下使用步骤（照做就能签）

前提：准备好3个东西

1. 要签名的OTA固件文件（比如 `D:\firmware\firmware.ota.bin`）；
2. 私钥文件（比如 `D:\keys\company_key_private_key.dat`）；
3. 目标设备的board_id（比如123，由设备硬件定义）。

执行命令（核心操作）

打开CMD，进入脚本所在目录，执行格式：

```
python 脚本名.py 固件文件路径 私钥文件路径 board_id
```

举个完整例子（替换成你的实际路径）

```
python sign_fw.py D:\firmware\firmware.ota.bin
D:\keys\company_key_private_key.dat 123
```

四、执行后会发生什么（流程解读）

1. **参数检查**：如果参数少于3个，提示用法 `Usage: sign_fw OTA_FILE PRIVATE_KEYFILE BOARD_ID` 并退出；
2. **加载固件**：读取指定的OTA固件文件（二进制）；
3. **加载私钥**：
   - 读取私钥文件，校验前缀是否为 `PRIVATE_KEYV1:`（和生成密钥时的格式对应）；
   - 解码Base64后检查私钥长度是否为32字节（Ed25519私钥标准长度），不对则报错；
4. **校验固件是否已签名**：
   - 检查固件末尾是否有固定标识 `0x432af13746e27519`；
   - 如果有，提示“Image is already signed”并退出（避免重复签名）；
5. **生成签名**：用私钥对固件内容生成64字节的Ed25519签名；
6. **追加签名到固件**：
   - 打包 `board_id（4字节）+ 固件长度（4字节）+ 签名（64字节）` → 72字节；
   - 加上8字节的固定标识，总共80字节，追加到固件末尾；
7. **保存固件**：覆盖原固件文件，打印“Applying signature”和“Wrote 固件路径”，签名完成。

五、常见问题&解决

1. **提示“Invalid key type”**：
   - 原因：私钥文件格式不对（前缀不是 `PRIVATE_KEYV1:`）；
   - 解决：确认用的是之前 `generate_keys.py` 生成的私钥文件，不是公钥文件。
2. **提示“Bad key length xx”**：
   - 原因：私钥解码后长度不是32字节（比如文件损坏、不是Ed25519私钥）；
   - 解决：重新用 `generate_keys.py` 生成私钥，确保文件完整。
3. **提示“Image is already signed”**：
   - 原因：固件已经签过名，脚本防止重复签名；
   - 解决：用未签名的原始固件，或删除固件末尾的80字节签名区域后重试。
4. **提示“ModuleNotFoundError: No module named monocypher”**：
   - 原因：没装对pymonocypher，或Python版本不对；
   - 解决：回到之前的步骤，确保Python 3.9 + pymonocypher==3.1.3.2安装成功。

总结

1. **核心用法**：`python sign_fw.py 固件路径 私钥路径 board_id`；
2. **核心逻辑**：私钥签名固件 → 签名追加到固件末尾 → 设备用公钥验签后升级；
3. **成功标志**：终端输出“Applying signature”和“Wrote 固件路径”，固件文件大小增加80字节。

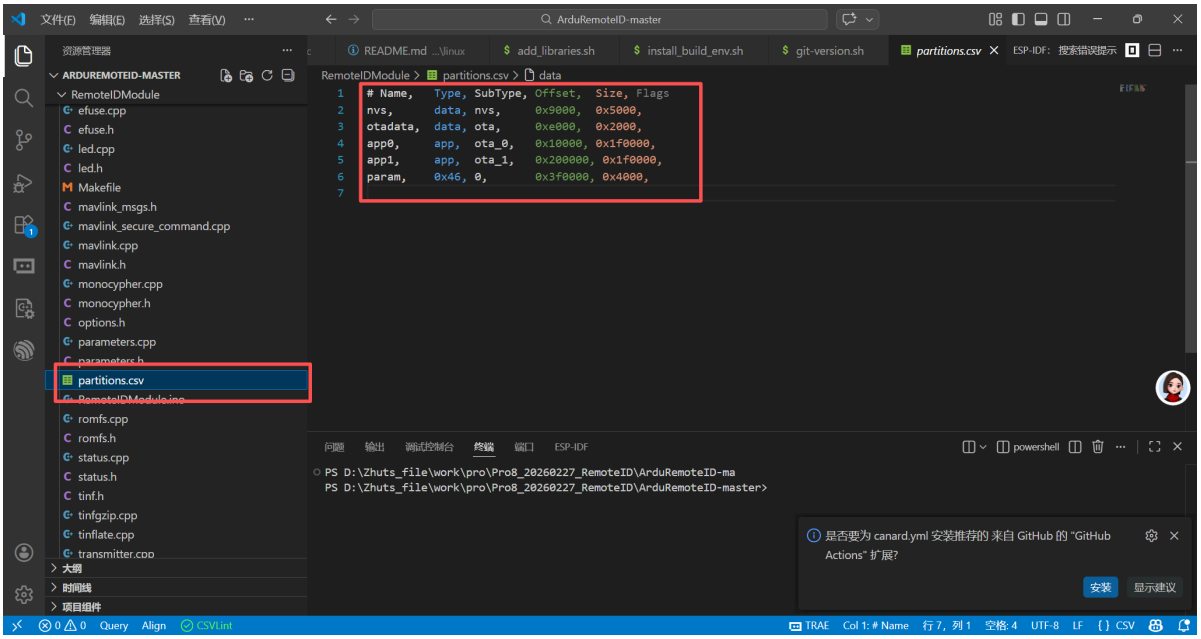
这个脚本是OTA固件发布前的关键步骤——只有签过名的固件，才能被你的RemoteID设备认可并升级，防止恶意固件篡改设备。如果执行时遇到具体报错，把报错信息贴出来，我帮你定位问题。

6、分区表是什么，有啥作用

分区表的生成路径（Arduino 环境下）

在你这个项目中，`RemoteIDModule.ino.partitions.bin` 不是手动编写的二进制文件，而是：

1. 先有一个 **分区表配置文件**（通常是 `partitions.csv` 或 `partitions.ino.csv`，纯文本格式）；
2. Arduino IDE/CLI 编译 ESP32-S3 代码时，会自动调用 ESP32 工具链中的 `parttool.py`，把这个 CSV 配置文件转换成二进制的 `partitions.bin`；
3. 最终输出到编译目录（比如 `build-ESP32S3_DEV`），也就是脚本中用到的 `RemoteIDModule.ino.partitions.bin`。



分区表从 “配置” 到 “二进制文件” 的全过程

1. 第一步：分区表的核心配置文件（CSV 格式）

这是分区表的 “源文件”，定义了 ESP32-S3 闪存的每个分区（比如主程序区、参数区、日志区）的**地址、大小、类型、名称**。

示例（ESP32-S3 4MB 闪存的典型分区表 `partitions.csv`）：

```
# Name,   Type, SubType, Offset,  Size,    Flags
nvs,      data, nvs,     0x9000,  0x6000,  # 非易失性存储（存配置参数）
phy_init, data, phy,     0xf000,  0x1000,  # WiFi/蓝牙 PHY 配置
factory,  app,  factory, 0x10000, 0x380000,# 主程序区（factory固件）
coredump, data, coredump,0x390000,0x10000, # 崩溃日志区
```

`Offset`：分区在闪存中的起始地址（和你脚本中 `0x8000` 对应分区表本身的地址不同，这是分区表定义的“其他分区地址”）；

`Size`：分区大小；

`Type`：`app`（应用程序）/`data`（数据）；

`Name`：分区名称（代码中可通过名称访问对应分区）

2. 第二步：Arduino 中分区表配置文件的位置

在你的项目中，分区表配置文件有两种常见存放方式：

- **方式 1（项目自定义）**：在 `RemoteIDModule.ino` 同目录下，有一个 `partitions.csv` 文件（或命名为 `RemoteIDModule.partitions.csv`），Arduino 编译时会优先使用这个自定义配置；
- **方式 2（使用 ESP32 官方模板）**：如果项目中没有自定义文件，Arduino 会使用 ESP32 核心库自带的默认分区表（路径：`$HOME/.arduino15/packages/esp32/hardware/esp32/<版本>/tools/partitions/`），比如 `default_4MB.csv`（对应你脚本中的 4MB 闪存）。

3. 第三步：编译时自动生成二进制分区表

当你在 Arduino IDE 中点击“编译”，或用 `arduino-cli compile` 编译时，背后会自动执行这几步：

1. Arduino 工具链读取分区表 CSV 文件；
2. 调用 ESP32 工具链中的 `parttool.py`（或 `esptool.py` 内置的分区表转换功能）；
3. 将 CSV 文本配置转换成二进制的 `partitions.bin`（即 `RemoteIDModule.ino.partitions.bin`）；
4. 把这个二进制文件放到编译输出目录（和 bootloader.bin、主程序.bin 同目录）

7、编译的时候是靠什么来操作的

`~/.arduino` 和 `~/.arduino15` 是 Arduino 工具的**配置和工具目录**，存放着 IDE 设置、核心库和编译工具。

`~/Arduino` 是你的**项目和库目录**，存放着你的代码和依赖库。

其中是靠esptool和arduino-cli以及编译器

```
zhuts@DESKTOP-LFOAOB0:~$ find / -name "arduino-cli" -type f 2>/dev/null
/home/zhuts/aruremoteid/bin/arduino-cli
```

```
zhuts@DESKTOP-LFOAOB0:~/aruremoteid/bin$ /home/zhuts/aruremoteid/bin/arduino-cli version
arduino-cli  Version: 0.27.1 Commit: a900cfb2 Date: 2022-09-06T16:44:27Z


A new release of Arduino CLI is available: 0.27.1 → 1.4.1
https://arduino.github.io/arduino-cli/latest/installation/#latest-packages
```

8、为什么没有找到那些对应的自动生成代码的描述文件?

你的项目里没有为 RemoteID 单独写 XML/DSDL 描述文件，而是直接使用：
MAVLink 官方库的 all.xml（里面已经包含 RemoteID 相关的 MAVLink 消息定义）；
DroneCAN 官方库的 DSDL 文件（modules/DSDL/dronecan/remoteid/ 目录下，专门定义 RemoteID 的 CAN 总线消息）；
脚本拉取这些 “包含 RemoteID 定义的官方描述文件” 后，自动生成能在 Arduino 中使用的 RemoteID 通信代码。

如果你的项目需要新增自定义的 RemoteID 消息（比如额外携带无人机电池信息），也**不用改官方 XML/DSDL**，而是：

1. 在项目中新建一个自定义描述文件（比如 `custom_remoteid.xml` / `custom_remoteid.dsdl`）；
2. 脚本中把官方描述文件和自定义文件一起传入生成工具（比如 `mavgen.py all.xml custom_remoteid.xml`）；
3. 生成的代码会同时包含 “标准 RemoteID” 和 “自定义 RemoteID”，兼顾合规性和定制化。

9、那些库的文件夹都是空的，到底是从哪拉取的呢

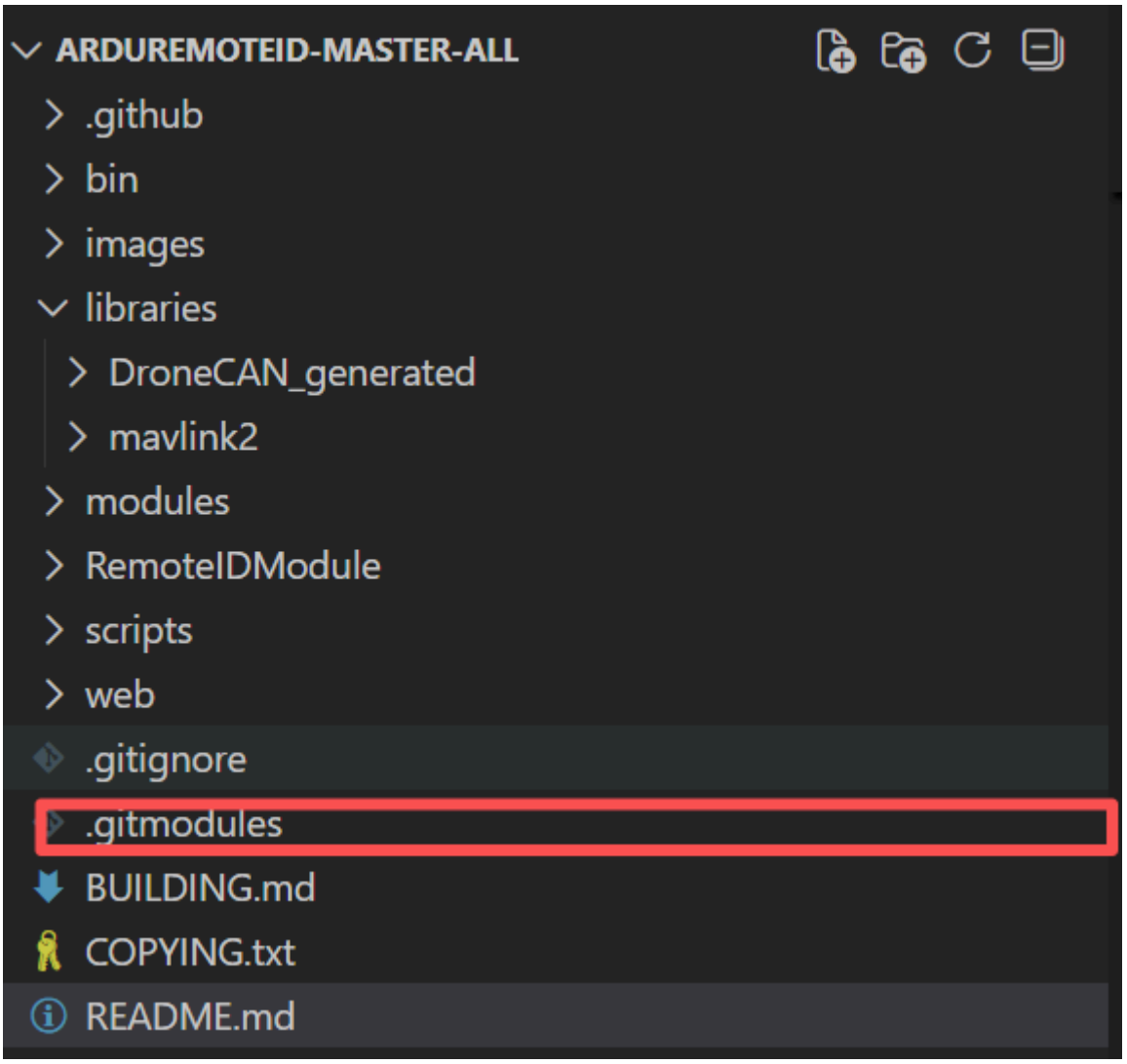
```
- git submodule init
- git submodule update --recursive
```

你已经通过 Git 子模块把 `modules/mavlink/`（含 XML）、`modules/DSDL/`（含 DSDL）、`modules/dronecan_dsdlc/`（生成工具）这些依赖拉到本地了；

你已经安装了 `pymavlink` 等工具（脚本开头注释也明确写了 `assumes pymavlink is installed`）

```
zhuts@DESKTOP-LFOAOB0:~/arduremoteid$ cat .gitmodules
[submodule "modules/opendroneid-core-c"]
        path = modules/opendroneid-core-c
        url = https://github.com/opendroneid/opendroneid-core-c.git
[submodule "modules/mavlink"]
        path = modules/mavlink
        url = https://github.com/ArduPilot/mavlink.git
[submodule "modules/DSDL"]
        path = modules/DSDL
        url = https://github.com/dronecan/DSDL.git
[submodule "modules/libcanard"]
        path = modules/libcanard
        url = https://github.com/dronecan/libcanard.git
[submodule "modules/dronecan_dsdlc"]
        path = modules/dronecan_dsdlc
        url = https://github.com/dronecan/dronecan_dsdlc.git
```

子模块里配置了这些子git的路径



10、如何查看工程最近一次提交的记录

```
cd /home/zhuts/arduremoteid
git log -1 RemoteIDModule/
```

```
zhuts@DESKTOP-LFOAOB0:~/arduremoteid$ git log -1 RemoteIDModule/
commit fc426739127d8c0e774934b6613c9c3c7a39e7f9 (HEAD -> master, origin/master, origin/HEAD)
Author: cuav-chen2 <chen2@cuav.net>
Date:   Sat Feb 15 09:38:50 2025 +0800

    public_key: add CUAV public key
zhuts@DESKTOP-LFOAOB0:~/arduremoteid$
```

该开源方案最新更新时间：**2025 年 2 月 15 日 09:38:50（UTC+8）**

已经过了一年没有修改了，所以应该是很稳定的。

核心更新内容：为 RemoteID 模块添加 CUAV 公钥，用于身份认证；

提交主体：CUAV的开发人员，说明这是官方维护的更新

11、git子模块是如何被工程git进行管理的

你疑惑的核心是：拉取的子模块代码本该自带 `.git` 文件夹，却没看到——答案是：**既不是被删除了，也不是由主工程 Git 直接管理，而是 Git 子模块机制自动将子模块的 `.git` 文件夹“重定向”到主仓库的 `.git/modules/` 目录下，并用 `.git` 文件替代。**

我帮你拆解这个过程，彻底讲清楚逻辑：

一、先明确：普通克隆 vs 子模块克隆的区别

操作方式	`.git` 位置/形态	版本管理主体
直接克隆子模块仓库（如 DSDL）	子模块目录下有 `.git` 文件夹	子模块自身 Git 管理
作为子模块克隆到主工程	子模块目录下是 `.git` 文本文件	主工程 Git 间接管理

简单说：子模块的 `.git` 文件夹没有被删除，只是“搬家”到了主仓库的 `.git/modules/` 目录，原位置只留一个“指向搬家地址”的 `.git` 文本文件。

二、子模块的 `.git` 是如何“搬家+重定向”的？

以你的 `modules/DSDL` 为例，完整流程：

1. **拉取子模块代码时**：
   Git 先把 DSDL 仓库的完整代码（含原生 `.git` 文件夹）拉到**主仓库的 `.git/modules/modules/DSDL/` 目录**（这是子模块的“真实 Git 目录”）；
2. **初始化子模块后**：
   Git 会删除 `modules/DSDL` 目录下原生的 `.git` 文件夹，替换成一个 `.git` 文本文件，内容是 `gitdir: ../.git/modules/modules/DSDL`（告诉 Git：“这个目录的 Git 数据在主仓库的这个路径里”）；
3. **版本管理逻辑**：
   - 子模块的代码文件（如 `.dsdl`）留在 `modules/DSDL` 目录，供你的脚本调用；

- 子模块的版本数据（提交记录、分支、远程仓库）存在主仓库的 `.git/modules/modules/DSDL/` 目录；
- 主工程 Git 不会“直接管理”子模块代码，只会记录“子模块当前的提交哈希”（在主仓库的 `git commit` 里能看到子模块的版本）。

三、验证：子模块的 `.git` 并没有被删除

你可以执行以下命令，找到“搬家”后的 `.git` 文件夹：

```
# 进入主仓库的子模块 Git 真实目录
cd /home/zhuts/arduremoteid/.git/modules/modules/DSDL

# 查看子模块的完整 Git 文件夹（和普通仓库的 .git 内容一致）
ls -l
# 会看到：objects/（版本数据）、refs/（分支）、config（远程仓库地址）等，这就是子模块原生的 .git 文件夹
```

四、主工程 Git 如何“管理”子模块？

主工程 Git 对於模块的管理是“间接的”：

1. **主仓库只记录“子模块的版本快照”**：
   主仓库的提交记录里，会保存子模块当前的提交哈希（比如 `fc42673`），相当于“标记此时用的是子模块的哪个版本”；
2. **子模块的代码仍由自身 Git 管理**：
   进入 `modules/DSDL` 目录执行 `git pull`/`git checkout`，依然能更新/切换子模块版本（因为 `.git` 文件会重定向到真实 Git 目录）；
3. **主仓库只“跟踪子模块版本”**：
   当你更新子模块版本后，主仓库会识别到“子模块版本变了”，执行 `git commit` 就能把新的子模块版本快照保存到主仓库。

总结

1. 子模块的 `.git` 文件夹**未被删除**，只是被 Git 子模块机制“迁移”到主仓库的 `.git/modules/` 目录；
2. 子模块目录下的 `.git` 是文本文件，作用是“重定向 Git 操作到迁移后的目录”；
3. 子模块代码**不由主工程 Git 直接管理**，主工程只记录子模块的版本快照，子模块自身的 Git 操作仍独立（但需通过主仓库的 `.git/modules/` 目录）。

简单说：子模块的 Git 数据“物理上”存在主仓库里，“逻辑上”仍属于子模块自身，主工程只做“版本快照记录”，既保证了子模块版本可控，又避免了嵌套 Git 仓库的混乱。