# 筆記 Raspberry Pi 2 Homekit - From Zero to Hey Siri

## Step 1: Step 1: Install Jessie

Tools needed:

- Jessie OS (version 2015-08-30)

[http://sysprogs.com/files/SD/2015-08-30-jessie-ras...](http://sysprogs.com/files/SD/2015-08-30-jessie-raspbian.7z)

- WinFlashTool (for write image to SD)

<http://winflashtool.sysprogs.com/>

Download the image and the WFTool from the links, put the image file into the WFTool, and install the OS to the SD

(First step in the video)

## Step 2: Step 2: Installs, Installs Everywhere...

**1:**Boot the RPi2 with Jessie.

Open a terminal program on your PC/MAC (im using putty)

Connect to your RPi2 with the putty (RPi2 ip found in your router DHCP client list)

Username: pi

Password: raspberry

In the youtube video find this step at 1:17.

**2:** next, the ragular update and upgrade:

sudo apt-get update && sudo apt-get upgrade -y

**3:** Add yourself superuser privileges (less "sudo" type)

sudo su

**4:** Install python-dev and python-pip

apt-get install python-dev python-pip

**5:** Install libavahi and libdnssd

apt-get install libavahi-compat-libdnssd-dev

**6:** Download and install the latest node

wget [http://node-arm.herokuapp.com/node\_latest\_armhf.deb](http://node-arm.herokuapp.com/node_latest_armhf.deb%20%20.)

dpkg -i node\_latest\_armhf.deb

**7:** Check node and npm version and installation

node -v

npm -v

**8:** Clone Homekit Bridge from github, and go into the HAP-NodeJS folder

git clone [*https://github.com/KhaosT/HAP-NodeJS.git*](https://github.com/KhaosT/HAP-NodeJS.git)

cd HAP-NodeJS

**9:** There is a lot of missing modules, so install all of them in one command.

npm install -g node-gypnpm install node-persist

# npm install srp 🡺 sudo rm –r ./node\_modules/srp

npm install mdns

npm install ed25519

npm install curve25519-n2

npm install debug

npm install python-shell

**10:** Rebuild

npm rebuild

## Step 3: Step 3: Add New Light Accessory

**1:** Go into accessories folder, and make a new light accessory:

cd accessories

nano myLight\_accessory.js

In the youtube video find this step at 5:13

**2:** Copy in this empty .js file everything from the linked homekit.txt file, from the

**//Light accessory** line

Homekit.txt link:

<https://drive.google.com/file/d/0B6GR9Hj5Ut61bDZlekRMNFdTVEE/view>

Rename the accessory, change a little the "username" (looks MAC address, watch out the HEX rules)

Save the file and exit

In the youtube video find this step at 5:46

## Step 4: Make the Python Files to Control GPIO sudPins

**1:** in the HAP-NodeJS folder, make a new python folder, go into.

cd ..

mkdir python

cd python

In the youtube video find this step at 7:07

**2:** Make the GPIO "on" and "off" python file

nano light1.py

Copy this into:

**import RPi.GPIO as GPIO  
GPIO.setwarnings(False)**

**GPIO.setmode(GPIO.BOARD)**

**GPIO.setup(16, GPIO.OUT)**

**GPIO.output(16, 1)**

Save, and close the file

nano light0.py

Copy this into:

**import RPi.GPIO as GPIO  
GPIO.setwarnings(False)**

**GPIO.setmode(GPIO.BOARD)**

**GPIO.setup(16, GPIO.OUT)**

**GPIO.output(16, 0)**

Save, and close the file

In the youtube video find this step at 7:19

**3:** Connect a led GND into one of the RPi2 **GND** pin. Im used the outer line 3. pin, and connect the led "+" pin tho the RPi2 **GPIO.16** pin. Thats the 8. pin in the outer line.

Test the python codes:

python light1.py (led turn on)

python light0.py (led tudn off)

If everything ok, go back to HAP-NodeJS folder.

cd ..

npm rebuild

In the youtube video find this step at 8:14

**4:**

Start the Homekit Bridge.

node Core.js

## Step 5: Stap 5: Config Ios Device

**1:**

In the video, im using Apple HMCatalog app, but find better homekit app, like EVE elgato.

Add a new home, some room, and one of the rooms, add your new light accessory.

The PIN: 03145154

Add anyway.

In the youtube video find this step at 9:28

**2:**

* **使用 Linux Kernel 提供的 sysfs 來控制 GPIO**

在要寫程式之前，我們先來使用 Linux Kernel 提供的 sysfs 來控制 GPIO。

* 首先先將 GPIO4 設定成可以用 sysfs 控制

echo 4 > /sys/class/gpio/export

* 設定 GPIO4 為輸出腳

echo out > /sys/class/gpio/gpio4/direction

* 設定 GPIO4 輸出值為 1 (0: 低電位, 1: 高電位)

echo 1 > /sys/class/gpio/gpio4/value

* 設定 GPIO4 輸出值為 0 (0: 低電位, 1: 高電位)

echo 0 > /sys/class/gpio/gpio4/value

* 取消建立出來的 GPIO4 node

echo 4 > /sys/class/gpio/unexport

在你執行以上第 3 步的時候，你可以看到 LED 亮了起來，直到第 4 步時，才又變 回原本的狀態。

若想要使用 Bash 來控制 GPIO，則可以採用此種方式。

* **使用 debugfs 來觀看目前的 GPIO 設定**

我們可以使用 debugfs 來察看目前的 GPIO 設定，首先掛載 debugfs

root@raspberrypi:/home/pi# mount -t debugfs debug /d

接著就可以使用

root@raspberrypi:/home/pi# cat /d/gpio

來取得目前 GPIO 的狀況

root@raspberrypi:/home/pi# cat /d/gpio

GPIOs 0-53, bcm2708\_gpio:

gpio-4 (sysfs ) out hi

**使用 Python 控制 GPIO**

在 Raspbian 發行版當中，已經預先將 RPi.GPIO 模組包入，因此你可以直接 寫以下的 python 程式來控制 GPIO4，要注意的是，RPi.GPIO 設定的 GPIO 是採 用 Pin number，也就是說若我要修改 GPIO4 (Pin 7)，則實際上呼叫 Rpi.GPIO 的號碼為 **7** ，而不是 **4** 。

import RPi.GPIO as GPIO

import time

*# blinking function*

def blink(pin):

GPIO.output(pin,GPIO.HIGH)

time.sleep(1)

GPIO.output(pin,GPIO.LOW)

time.sleep(1)

return

*# to use Raspberry Pi board pin numbers*

GPIO.setmode(GPIO.BOARD)

*# set up GPIO output channel, we set GPIO4 (Pin 7) to OUTPUT*

GPIO.setup(7, GPIO.OUT)

*# blink GPIO4 (Pin 7) 50 times*

for i in range(0,50):

blink(7)

GPIO.cleanup()

**使用 C 語言控制 GPIO (不使用外部函式庫)**

C 語言因為提供了指標，可以任意的去修改記憶體的某個部份，因此我們也可 以透過修改記憶體區塊的方式，來設定我們的 GPIO。

以下程式修改自 [elinux](http://elinux.org/RPi_Low-level_peripherals) 的範例，首先點亮 GPIO4 上的 LED 一秒後，再關閉。

*/\* Modified from http://elinux.org/RPi\_Low-level\_peripherals \*/*

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <sys/mman.h>

#include <unistd.h>

#define BCM2708\_PERI\_BASE 0x20000000

#define GPIO\_BASE (BCM2708\_PERI\_BASE + 0x200000) */\* GPIO controller \*/*

#define PAGE\_SIZE (4 \* 1024)

#define BLOCK\_SIZE (4 \* 1024)

int mem\_fd;

void \*gpio\_map;

*/\* I/O access \*/*

volatile unsigned \*gpio;

*/\* GPIO setup macros. Always use INP\_GPIO(x) before using OUT\_GPIO(x) or SET\_GPIO\_ALT(x,y) \*/*

#define INP\_GPIO(g) \*(gpio+((g)/10)) &= ~(7<<(((g)%10)\*3))

#define OUT\_GPIO(g) \*(gpio+((g)/10)) |= (1<<(((g)%10)\*3))

#define SET\_GPIO\_ALT(g,a) \*(gpio+(((g)/10))) |= (((a)<=3?(a)+4:(a)==4?3:2)<<(((g)%10)\*3))

#define GPIO\_SET \*(gpio+7) */\* sets bits which are 1 ignores bits which are 0 \*/*

#define GPIO\_CLR \*(gpio+10) */\* clears bits which are 1 ignores bits which are 0 \*/*

*/\*\**

*\* Set up a memory regions to access GPIO*

*\**

*\*/*

void setup\_io()

{

*/\* open /dev/mem \*/*

if ((mem\_fd = open("/dev/mem", O\_RDWR|O\_SYNC) ) < 0) {

printf("can't open /dev/mem \n");

exit(-1);

}

*/\* mmap GPIO \*/*

gpio\_map = mmap(

NULL, */\* Any adddress in our space will do \*/*

BLOCK\_SIZE, */\* Map length \*/*

PROT\_READ|PROT\_WRITE, */\* Enable reading & writting to mapped memory \*/*

MAP\_SHARED, */\* Shared with other processes \*/*

mem\_fd, */\* File to map \*/*

GPIO\_BASE */\* Offset to GPIO peripheral \*/*

);

close(mem\_fd); */\* No need to keep mem\_fd open after mmap \*/*

if (gpio\_map == MAP\_FAILED) {

printf("mmap error %d\n", (int)gpio\_map); */\* errno also set! \*/*

exit(-1);

}

*/\* Always use volatile pointer! \*/*

gpio = (volatile unsigned \*)gpio\_map;

}

int main(int argc, char \*\*argv)

{

*/\* Set up gpi pointer for direct register access \*/*

setup\_io();

*/\* Must use INP\_GPIO before we can use OUT\_GPIO \*/*

INP\_GPIO(g);

OUT\_GPIO(g);

*/\* Set GPIO4 to 1 \*/*

GPIO\_SET = 1 << 4;

sleep(1);

*/\* Clear GPIO 4 \*/*

GPIO\_CLR = 1 << 4;

sleep(1);

return 0;

}

**使用 C 語言控制 GPIO (使用 BCM2835 函式庫)**

[BCM2835 C library](http://www.airspayce.com/mikem/bcm2835/) 是針對 BCM2835 CPU 用的 C 語言函式庫，透過他你可以輕鬆 的使用 C 語言去控制 Raspberry Pi 的 GPIO、I²C 等裝置，第一次使用這個函式庫 時，需要自己下載來編譯。

wget http://www.airspayce.com/mikem/bcm2835/bcm2835-1.25.tar.gz

tar zxvf bcm2835-1.25.tar.gz

cd bcm2835-1.25

./configure

make

sudo make install

接著我們就可以寫如下的 C 語言程式來閃爍我們的 LED。

*/\* Compile with: gcc blink.c -o blink -l bcm2835 \*/*

#include <bcm2835.h>

*/\* Blinks on RPi pin 7 \*/*

#define PIN RPI\_GPIO\_P1\_7

int main(int argc, char \*\*argv) {

if ( !bcm2835\_init() ) {

perror("Error");

return 1;

}

*/\* Set the pin to be an output \*/*

bcm2835\_gpio\_fsel(PIN, BCM2835\_GPIO\_FSEL\_OUTP);

*/\* Blink LED \*/*

while (1) {

*/\* Turn on LED \*/*

bcm2835\_gpio\_write(PIN, HIGH);

*/\* Delay 500ms \*/*

delay(500);

*/\* Turn off LED \*/*

bcm2835\_gpio\_write(PIN, LOW);

*/\* Delay 500ms \*/*

delay(500);

}

return 0;

}

## Toggle GPIO on Raspberry Pi using HomeKit

<https://delog.wordpress.com/2016/05/04/toggle-gpio-on-raspberry-pi-using-homekit/>

1. **install homebridge-gpio-wpi ,** That should install all node modules under ~/node\_modules/.

**🡺** npm install homebridge-gpio-wpi

1. Configure homebridge by editing ~/.homebridge/config.json

**{**

**"bridge": {**

**"name": "Homebridge",**

**"username": "CC:22:3D:E3:CE:32",**

**"port": 51826,**

**"pin": "031-45-155"**

**},**

**"description": "This has some fake accessories",**

**"accessories": [**

**{**

**"accessory":      "FakeBulb",**

**"name":**       "Test lamp",

            "bulb\_name":      "Lamp 1"

        },

        {

            "accessory": "GPIO",

            "name": "GPIO2",

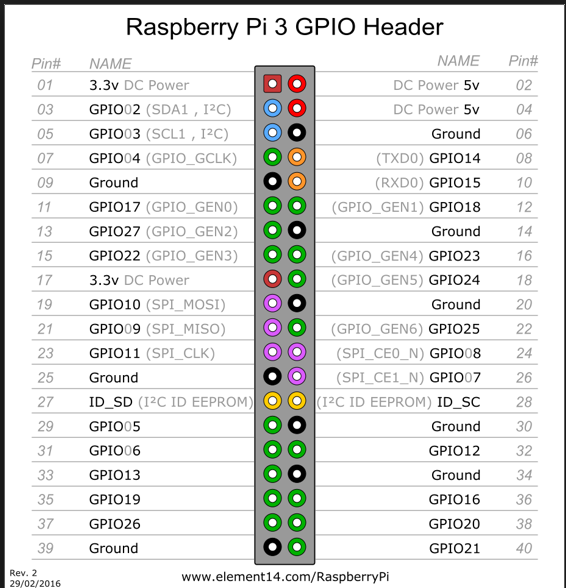
            "pin": 27

        }

    ],

    "platforms": []

}



* TUTORIAL: HAP-NodeJS (HomeKit) Installation on Raspberry Pi
  + curl –sSL goo.gl/k8QMGm | bash
* TUTORIAL: Homebridge Installation on Raspberry Pi
  + curl –sSL goo.gl/Ksdhph | bash
  + refer to :

<https://github.com/nfarina/homebridge/wiki/Running-HomeBridge-on-a-Raspberry-Pi>